Vers une auto-organisation des réseaux informatiques

Florent Nolot

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Mémoire

présenté pour l'obtention du diplôme de

HABILITATION À DIRIGER DES RECHERCHES

Spécialité : Informatique

Vers une auto-organisation des réseaux informatiques

Florent NOLOT

Soutenance publique le Vendredi 29 Novembre 2013
devant le jury suivant :

Président	**M. Marc BUI** Professeur à l'Université Paris 8
Rapporteurs	**M. Hervé GUYENNET** Professeur à l'Université de Franche Comté
	M. Nouredine MELAB Professeur à l'Université Lille 1
	M. Jean-Frédéric MYOUPO Professeur à l'Université de Picardie Jules Verne
Examinateurs	**M. Guillaume GELLÉ** Professeur à l'Université de Reims Champagne-Ardenne
Directeur	**M. Olivier FLAUZAC** Professeur à l'Université de Reims Champagne-Ardenne

CENTRE DE RECHERCHE EN SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION
ECOLE DOCTORALE SCIENCES TECHNOLOGIE SANTÉ

A Chloé, Elisabeth et mes parents

Table des matières

Table des figures

Remerciements

Ce mémoire a été rendu possible grâce aux soutiens de nombreuses personnes de mon entourage proche au niveau personnel et professionel. Je remercie tout particulièrement les rapporteurs de ce mémoire pour l'intérêt qu'ils ont porté à mes travaux.

Cela va faire maintenant de nombreuses années que je connais Monsieur Jean-Frédéric MYOUPO. Lors de mes études à l'université de Picardie Jules Verne, il était le responsable du D.E.A. d'informatique fondamentale que j'ai obtenu et a fait partie de mon jury de doctorat. Il a toujours été de bon conseil. J'ai ensuite fait la connaissance de Monsieur Nouredine MELAB lors de mon poste d'ATER à l'université du Littoral Côte d'Opale. Il m'a initié aux réseaux informatiques et m'a encouragé dans mes travaux. Puis c'est à l'occasion de la soutenance de mon 1^{er} doctorant, Monsieur Bachar Salim HAGGAR que j'ai rencontré Monsieur Hervé GUYENNET qui par la suite, a toujours su me donner de précieux conseils.

Je remercie Monsieur Marc BUI d'avoir accepté d'examiner mes travaux et qui m'avait déjà fait l'honneur de rapporter ma thèse de doctorat.

Je remercie Monsieur Guillaume GELLÉ pour avoir accepté d'examiner mes travaux et de m'avoir contacté afin de prendre la responsabilité scientifique d'un sous-projet dans une ANR.

Je remercie Monsieur Olivier FLAUZAC pour tous ses conseils. Il a permis à cette habilitation de voir le jour en acceptant de co-encadrer plusieurs doctorants et en me faisant une entière confiance. Depuis mon arrivée à Reims, il m'a toujours soutenu, que ce soit pour développer de nouveaux projets de recherche ou des collaborations internationales.

Je remercie, pour la confiance qu'ils m'ont accordée, Messieurs Bachar Salim HAGGAR et Mandicou BA pendant leur doctorat ainsi que Monsieur Rafik MAKHLOUFI qui est venu faire un post-doctorat sous ma responsabilité.

Je remercie également Monsieur Michaël KRAJECKI pour tous ses précieux conseils, que ce soit au niveau scientifique que administratif, ainsi que tous les autres membres de l'équipe SysCom du CReSTIC, quel que soit leur statut et en particulier Messieurs Cyril RABAT et Luiz-Angelo STEFFENEL. Je n'oublierai pas non plus les nombreuses autres personnes avec qui j'ai fêté de nombreux événements où la bonne humeur était obligatoire.

Enfin, mes derniers remerciements iront à ma famille. Je remercie ma compagne, Elisabeth, qui m'a toujours soutenu et encouragé, et ma fille Chloé de ne pas se coucher trop tard pour que je puisse, de temps en temps, travailler sans regarder Tchoupi. Je remercie également mes parents qui ont toujours répondu présent, que ce soit pendant mes études ou pendant ma carrière professionnelle. Sans eux, je ne serais jamais parvenu jusque là aujourd'hui.

Introduction

Depuis les travaux de Leonard Kleinrock en 1961 [Kle61] sur la transmission de données par paquets et la création d'Internet [Kle69], les réseaux informatiques et leurs usages ont fortement évolué. Ces dernières années, entre 2000 et 2010, le pourcentage des ménages connecté à Internet est passé de 12% à 64% [1]. Les architectures réseaux, les solutions de gestion d'accès aux ressources et d'acheminement des informations sont régulièrement améliorées afin de répondre aux exigences et besoins des utilisateurs. La minimisation du délai de transmission, la haute disponibilité des services, l'augmentation des débits ne sont que quelques exemples de critères qui caractérisent les attentes des utilisateurs. De plus, avec la multiplication des équipements mobiles et des offres de connexion haut débit (filaire ou sans fil), les usages des utilisateurs ont également évolués. Dorénavant, les gens passent de plus en plus de temps sur Internet. Une enquête d'Avril 2012 [2] révèle qu'en moyenne la population passe 13 heures par semaine sur Internet, quel que soit le lieu ou le mode de connexion. En fonction des catégories socio-professionnelles et des diplômes, il existe des disparités. Les cadres supérieurs passent en moyenne plus de temps sur Internet que devant la télévision (en moyenne 20 heures par semaine sur Internet contre 13 heures devant la télévision). Les utilisateurs ont donc de nouveaux usages, de nouvelles attentes et tous les secteurs d'activité sont concernés par ces évolutions. L'enseignement a également dû suivre ces avancées technologiques et s'adapter aux nouvelles demandes et à la manière dont les apprenants consomment maintenant un enseignement. Pour preuve, le nombre de plates-formes MOOC (Massive Open Online Course) augmente régulièrement et fait apparaître de nouvelles problématiques. Comment peut-on réaliser des travaux pratiques à distance ? La virtualisation est une solution pour offrir à l'apprenant, sans modification de la configuration de son poste de travail, les bons outils mais dans le domaine des réseaux informatiques, il est important de travailler sur des équipements réels.

Pour résoudre les différentes problématiques évoquées, les réseaux informatiques ont donc besoin d'être organisés et hiérarchisés. D'autant plus quand ces réseaux sont composés d'entités à faibles ressources comme des réseaux de capteurs ou d'entités avec des connexions non permanentes comme des réseaux mobiles. En entreprise, tous les réseaux sont naturellement hiérarchisés. Pour pouvoir communiquer avec une destination, les données passent d'intermédiaires en intermédiaires. Chaque entité a la connaissance de l'intermédiaire suivant. Ces intermédiaires peuvent être des routeurs, des bornes wifi ou des commutateurs. Les données empruntent toujours des chemins déterministes construits soit grâce à des protocoles de couche 2 comme le protocole IEEE 802.1d ou IEEE 802.1w qui se basent sur le protocole Ethernet pour construire des arbres couvrants, soit grâce à des protocoles de couche 3 comme les protocoles de routage OSPF ou BGP. Dans tous les cas, des connaissances et des apprentissages sont nécessaires pour effectuer la construction de ces chemins. Dans les réseaux ad-hoc, les entités peuvent avoir des ressources limitées et leurs capacités à acheminer des informations vont donc être restreintes. La constitution de groupes logiques en clusters permet de diminuer les connaissances que les entités doivent avoir pour calculer leur route. De plus, cette restriction va devenir un impératif avec la migration vers le protocole IPv6 qui permettra à toutes entités connectées à Internet d'être joignable. Il n'est

1. Source Insee, enquête Technologies de l'information et de la communication d'avril 2010 : http://www.insee.fr/fr/themes/document.asp?ref_id=ip1340
2. Source CREDOC http://www.credoc.fr/pdf/Sou/Credoc_DiffusiondesTIC_2012.pdf

évidemment pas envisageable de construire des tables de routage après apprentissage de la topologie du réseau Internet mais bien que de certaines zones comme le fait OSPF.

C'est avec ces objectifs que j'ai mené mes travaux de recherche, toujours dans le cadre de projets financés, seul ou en collaboration, afin de pouvoir offrir des solutions à la fois sur la structuration automatique des réseaux ad-hoc, sur l'acheminement de l'information et sur la sécurisation de ces informations. J'ai pu apporter des solutions afin de faciliter l'administration de tout un système de traitement de l'information. Dans un premier temps, j'ai travaillé sur le problème de la clusterisation des réseaux ad-hoc afin de construire dynamiquement des groupes d'entités pour hiérarchiser le réseau. Puis je me suis penché sur la problématique de l'économie de l'énergie et ai analysé le comportement des solutions proposées dans les réseaux ad-hoc afin de les améliorer pour les réseaux de capteurs.

Dans un deuxième temps, j'ai exploité les constructions automatiques de clusters afin de pouvoir faire de l'acheminement d'informations. Pour cela, une solution de structures arborescentes a été proposée. Elle tire son originalité de l'exploitation des informations déjà échangées pour construire les clusters. Ainsi, j'ai conçu une solution novatrice de construction simultanée de clusters et d'arbres couvrants. J'ai également proposé des solutions de routage réactives, pro-actives mais également hybrides sur les réseaux en clusters.

Dans un troisième temps, je me suis concentré sur la fiabilité des informations échangées et surtout sur la confiance que l'on pouvait avoir dans des données reçues. En effet, lors de la constitution d'un groupe, comment s'assurer que toutes les entités du groupe sont fiables et de confiance ? Comment s'assurer qu'une entité ne va pas tenter de corrompre ma table de routage ? Dans une troisième partie, je présenterai une solution appelée RemoteLabz et développée dans le but de construire automatiquement des groupes d'équipements informatiques. Chacun de ces groupes partage une infrastructure physique identique tout en étant cloisonné afin d'éviter toutes communications entre-eux. Ce projet permet de mettre en évidence l'utilité et l'usage de la hiérarchisation des réseaux et du cloisonnement durant l'acheminement de l'information. Il en est né le concept de grille de sécurité. En effet, même s'il est possible d'éviter la propagation de données entre clusters, comment pouvons-nous assurer que l'ajout d'une nouvelle entité dans un cluster ne va pas le compromettre ? Pour parvenir à la définition de ce concept et de son fonctionnement, une approche grille a été utilisée afin de modéliser un réseau, de la couche physique à la couche applicative. Cette modélisation a permis de mettre en évidence et de prendre en compte le fait que des communications ne peuvent pas toujours aboutir en raison d'équipements ou de règles de sécurité positionnées dans une infrastructure. De cette modélisation et du modèle fonctionnel du réseau construit, j'ai décrit le fonctionnement d'un intergiciel assurant la sécurité globale d'un réseau ad-hoc et permettant d'assurer sa non compromission. Cet intergiciel est basé sur de la confiance entre utilisateurs ou entités mais surtout également sur une politique globale de sécurité qui est distribuée sur une grande partie des entités. Ainsi, toute personne du groupe n'a qu'une vue partielle de la politique de sécurité globale.

De ces travaux de recherche, il en résulte de multiples perspectives que je présente en conclusion de ce mémoire.

Chapitre 1

Les systèmes distribués et la tolérance aux fautes

Résumé.

Ce chapitre introduit les systèmes distribués en général, environnements dans lesquels figurent les réseaux ad-hoc. Puis j'exposerai brièvement le problème de la tolérance aux fautes avec différentes approches, qu'il est possible de retrouver de façon détaillé dans les ouvrages de M. Raynal ou G. Tel [Ray85, Ray92, Tel94a]. Je présenterai plus précisément l'auto-stabilisation, une approche de la tolérance aux pannes définie par E.W. Dijkstra en 1974 dans [Dij74], que j'utilise dans la quasi totalité des solutions présentées dans ce mémoire.

1.1 Les systèmes distribués

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

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

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

1.1.1 Quelques critères de différentiation des systèmes

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

- Topologie : les différentes topologies sont représentées par un graphe sur lequel les *arêtes* représentent les liens de communication et les *sommets* les processus. Dans la suite nous utilisons indifféremment les termes de sommets, processus ou machine et les canaux de communication seront indifféremment désignés par liens ou arêtes.
- Type de liens de communications : les liens de communication peuvent être soit *bidirectionnels* , soit *unidirectionnels*, c'est à dire que les processus échangent des données dans les deux sens ou

bien dans un seul sens. Ainsi si un processus P_1 envoie des données à P_2 par un lien unidirectionnel alors P_2 ne pourra pas envoyer d'informations à P_2 par ce même lien.
Il existe plusieurs hypothèses possibles sur les communications, de la communication globale à la communication point à point, en passant par la communication multi-points. Dans une *communication globale*, un processus peut communiquer avec tous les autres. Dans une *communication multi-points*, un sous-ensemble de processus peut communiquer ensemble alors que dans la *communication point-à-point*, seulement deux processus peuvent s'échanger des informations grâce à un lien de communication qui les relie. Dans la suite, nous n'utilisons que des communications point-à-point.

- Synchronisme : il existe deux types de systèmes : les systèmes *synchrones* et *asynchrones* [Awe85, Ray91]. Dans un système synchrone, tous les processus exécutent une action exactement en même temps, commandées par une pulsation générée par le système. Dans un système asynchrone, chaque composant fonctionne à sa propre vitesse.
- Connaissances globales : la première hypothèse est liée aux identifiants que peuvent avoir les unités de traitement. Chaque unité peut avoir un identifiant unique afin de pouvoir tous les différentier. Si toutes les unités sont identiques avec aucun moyen de les différencier, le système est dit *uniforme*. Et si seulement une unité est particulière et exécute un protocole différent des autres, le système est dit *semi-uniforme*. Le site distingué est appelé *racine* ou *leader*. La deuxième hypothèse concerne la connaissance que peut avoir chaque sommet sur le réseau : sa topologie, le nombre de sommets, les sommets à certaine distance, ...
- Les modèles de communication : il existe 3 principaux modèles de communication dans la littérature : le modèle à états, le modèle à registres 1.1 et le modèle à passage de message. Dans le *modèle à états*, défini par E.W. Dijkstra [Dij74], un processus peut lire et écrire dans sa propre mémoire mais peut également, sans message, lire la mémoire de ses voisins. Dans *le modèle à registres*, également appelé *modèle à mémoire partagée* par S. Dolev dans [Dol00], les liens de communication sont représentés par des registres attachés à chaque processus. Un lien l_{ij} reliant les processus i et j est associé à un registre r_{ij} dans lequel le processus i écrit les informations à transmettre au processus j et le processus j lit ce registre r_{ij} afin de connaître les valeurs "transmises" par i à j. Symétriquement, le processus j possède un registre r_{ji}. Dans le modèle à passage de messages, chaque processus est interconnecté par une lien de communication asynchrone. Les informations sont alors échangées par envoi de messages entre les processus. Ce dernier modèle est le plus réaliste et correspond au modèle que j'utilise durant mes travaux de recherche.

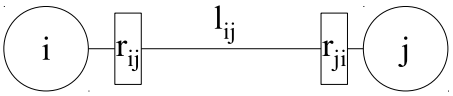


FIGURE 1.1 – Modèle à registres

- Avec ou sans infrastructure [VV05] : avant l'arrivée des protocoles de communication Wifi, les réseaux d'entreprises était considéré comme des réseaux avec une infrastructure fixe. Chaque ordinateur est connecté physiquement à un point précis du réseau. Avec les protocoles de communications Wifi, cette hypothèse a été modifié et le terme de WLAN pour Wireless Local Area Network est apparu, en même temps que les réseaux ad-hoc. Dans un WLAN, les ordinateurs communiquent avec un point d'accès qui permet d'étendre le réseau filaire. Dans un réseau ad-hoc, chaque ordinateur joue le rôle de relais de l'information et il n'existe alors plus aucune infrastructure physique de communication. Les liens de communication entre les ordinateurs deviennent mobiles.

L'affaiblissement de ces différentes hypothèses est une des principales motivations dans le développement ou l'amélioration d'algorithmes distribués. De cette façon, les études s'orientent vers l'étude de systèmes de plus en plus généraux.

1.1.2 Les réseaux ad hoc

Les réseaux ad hoc sont des réseaux locaux sans infrastructure fixe et en mesure de faire de l'échange d'informations entre entités communicantes, souvent appelés *nœuds* ou *sites*. Les réseaux ad hoc mobiles [SG85, CM99] sont des réseaux ad hoc dont les nœuds sont mobiles. Il faut donc que ce réseau supporte les changements de topologies, les technologies de communication multi-sauts et des bandes passantes pouvant être faibles. Les problèmes relatifs aux réseaux informatiques se retrouvent donc dans les réseaux ad hoc mais avec une contrainte supplémentaire : la gestion de la mobilité des nœuds. D'où l'existence depuis plus de 30 ans de travaux de recherche dans ce domaine.

Depuis 2003, mes travaux de recherche présentés dans ce mémoire portent sur les technologies de ce type de réseaux, en résolvant la problématique de la tolérance aux fautes transitoires par la solution de l'auto-stabilisation, présentée ci-dessous.

1.2 La tolérance aux fautes

Avec la multiplication d'ordinateurs mis en réseau, il devient primordial que tous les algorithmes distribués fonctionnant sur ces ordinateurs, tolèrent un maximum de fautes pouvant subvenir. Ces *fautes* sont des défaillances temporaires ou définitives d'un ou plusieurs composants du réseau. Un composant est dit défaillant lorsqu'il ne remplit plus ses fonctions. Le but des algorithmes tolérants aux fautes est d'assurer un bon fonctionnement du réseau ou de pouvoir retrouver le fonctionnement voulu, même quand des fautes surviennent. Toutes ces fautes sont généralement classées suivant certains critères :

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

Pour concevoir des algorithmes distribués qui gèrent ces différentes défaillances, il existe deux approches : la tolérance aux pannes et l'auto-stabilisation. Dans la littérature, toutes les exécutions d'un algorithme tolérant aux fautes peuvent être classées suivant deux propriétés dites de *sûreté* et de *vivacité*. Une exécution vérifie la *propriété de sûreté* si elle assure qu'une propriété non voulue n'arrive jamais. Autrement dit, aucun problème n'est provoqué durant cette exécution. Une exécution vérifie la *propriété de vivacité* s'il existe une configuration qui vérifie une propriété voulue. Par exemple, si une exécution résout le problème du consensus binaire, les propriétés de sûreté et de vivacité peuvent s'énoncer de la façon suivante :

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

Nous présentons maintenant trois domaines de la tolérance aux fautes : la tolérance aux pannes, l'auto-stabilisation et la stabilisation instantanée. Notons qu'il existe des approches mixtes qui essaient de gérer différents types de défaillances ([AK98b, KA97b, KA97a, KA98]).

1.2.1 La tolérance aux pannes

Lorsque certains composants du système sont en pannes, les algorithmes tolérants aux pannes essaient de maintenir un fonctionnement correct afin de continuer à vérifier les spécifications du sys-

tème. Autrement dit les propriétés de sûreté et de vivacité doivent toujours être vérifiées. Mais avec ces contraintes, tous les problèmes ne trouvent pas forcément de solution.

En 1985, Fisher, Lynch et Paterson ont montré dans [FLP85] qu'il est impossible de résoudre le problème du consensus défini ci-dessus, de manière déterministe dans un système asynchrone, même si la défaillance est définitive et limitée à un seul processus. Cette impossibilité est obtenue sous la condition que la vivacité et la sûreté soient toujours vérifiées, même en présence de pannes. Afin de trouver une solution à ce problème, plusieurs chercheurs ont décidé d'avoir une approche différente en modifiant les hypothèses sur le modèle ou en acceptant des solutions plus faibles. La même année que la publication de [FLP85], Bracha et Toueg proposent dans [BT85] deux algorithmes résolvant le problème du consensus. Pour cela, ils remettent en cause la propriété de la vivacité. Le premier algorithme est tolérant aux pannes définitives de type crash (arrêt de fonctionnement), le second aux pannes byzantines. Les deux garantissent que le système finit par atteindre le consensus si $n > 2k$ pour le premier et $n > 3k$ pour le second, n étant le nombre de processus du réseau et k le nombre maximum de processus susceptibles d'être défaillant. Dolev, Dwork et Stockmeyer choisissent une autre approche dans [DDS87]. Ils étudient le problème en analysant les différents niveaux de synchronisme nécessaire à la résolution du consensus en présence de fautes. Pour cela, ils reprennent les trois sources d'asynchronisme des systèmes asynchrones présentées dans [FLP85], à savoir les processus, les délais de transmission des messages et l'ordre d'arrivée des messages à leur destinataire. L'unique synchronisation des processus n'est pas une condition suffisante pour résoudre le problème du consensus. Ils mettent aussi en évidence qu'il existe une solution pour le consensus en faisant un minimum d'hypothèses supplémentaires sur la synchronisation des communications entre processus et en conservant la même hypothèse sur l'atomicité des opérations des processus. Une troisième approche tend à vouloir détecter les pannes. Chandra et Toueg introduisent dans [CT91] la notion de détecteur de pannes non fiable. Ils apportent ainsi une autre solution au problème du consensus. Le problème de la détection/correction de pannes est également étudié dans [AK98a].

Les défaillances transitoires peuvent être gérées dans un cadre original : celui de l'auto-stabilisation.

1.2.2 L'auto-stabilisation

L'auto-stabilisation a été introduite par Dijkstra en 1974 dans [Dij74] et peut être schématisé par la figure 1.2. Informellement nous pouvons la définir comme suit :
Quelle que soit sa configuration initiale, un système auto-stabilisant recouvre un comportement correct en un temps fini.

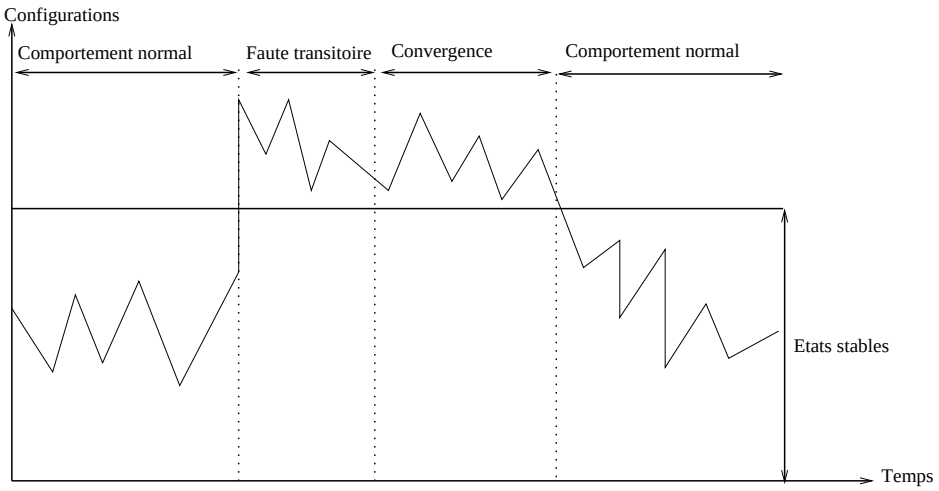


FIGURE 1.2 – Exemple de résultats

Bien que cette définition ne fasse pas appel explicitement à la notion de défaillance transitoire, la notion d'auto-stabilisation est particulièrement adaptée pour le traitement de ces défaillances. En effet, l'apparition de défaillances dans le système a pour conséquence première de modifier arbitrairement la configuration de ce système. La configuration obtenue après l'arrêt des défaillances peut-être considérée comme la configuration initiale de la définition ci-dessus. La nature des défaillances considérées (en dehors de leur coté transitoire) peut-être quelconque. En particulier, elles peuvent être byzantines, ce qui est le cas le pire, et le nombre de composants défaillants n'a pas d'autre borne à priori que le nombre total de composants du système. Classiquement, ces défaillances n'affectent que les variables des processeurs et le contenu des canaux de communications. Cependant si les défaillances ont aussi corrompu le code du protocole, il faut alors supposer qu'il existe un mécanisme de rafraîchissement de ce code. La configuration considérée est alors la configuration obtenue après rafraîchissement du code de tous les processus.

Cette définition sous-tend que le système ne respecte pas toujours les spécifications voulues. Le point de vue parfaitement justifié, plus de dix ans après dans [FLP85], permet de ne pas faire d'hypothèses restrictives sur le modèle asynchrone. Il constitue donc la meilleure approche pour traiter les défaillances transitoires dans le cadre de protocoles où la non vérification temporaire de la sûreté n'est pas rédhibitoire. Nous pouvons énoncer une définition plus formelle de l'auto-stabilisation :
Quelle que soit la configuration initiale, un système auto-stabilisant converge en un temps fini vers une configuration à partir de laquelle il satisfait les spécifications à jamais.

La démonstration de l'auto-stabilisation d'un système suit les deux points de cette définition : nous définissons un sous-ensemble $\mathcal{L}$ des configurations du système et nous montrons :

(a) que toute configuration de $\mathcal{L}$ est légitime, c'est à dire qu'à partir d'une telle configuration, le système vérifie les spécifications à jamais (propriété de clôture)

(b) que partant de n'importe quelle configuration, le système converge en un temps fini vers une configuration de $\mathcal{L}$ (propriété de convergence).

1.3 Conclusion

Après cette introduction sur les systèmes distribués et l'auto-stabilisation, je vais maintenant présenter mes travaux de recherche. Dans une première partie, j'aborderai la problématique de l'auto-organisation des réseaux ad-hoc et les solutions apportées. Puis dans une seconde partie, afin d'optimiser la gestion de l'acheminement de l'information dans les réseaux ad-hoc, je présenterai des solutions de gestion efficace du routage de l'information dans des réseaux structurés. Dans une dernière partie, avant de conclure sur des perspectives de recherche, je me focaliserai sur le problème de la gestion de la sécurité dans des environnements mobiles et j'introduirai le nouveau concept des grilles de sécurité.

Chapitre 2

Structuration des réseaux autonomes

Résumé.

Les réseaux ad-hoc sont des réseaux sans infrastructure fixe et donc pour que les échanges d'informations puissent se faire, chaque nœud doit être en mesure de connaître son voisinage et pouvoir transporter l'information de proche en proche jusqu'à la destination finale. Dans ce chapitre, je présente les solutions obtenues dans la construction auto-stabilisante de clusters et dans le modèle à passage de messages. Dans une première partie, je présenterai une solution de construction de clusters à 1 saut, c'est-à-dire des clusters de diamètre au plus égal à 2 *puis dans une deuxième partie, une solution de construction de clusters à* k *sauts. Toutes les solutions n'utilisent que des informations locales et se basent exclusivement sur les identités des nœuds voisins. Grâce à ces critères, je montrerai dans une troisième partie, que nos solutions sont, comparativement aux autres solutions existantes, meilleures en terme de nombre de messages échangés pour atteindre la stabilisation et donc également en énergie dans le cadre de leur exploitation dans un réseau de capteurs. De plus, les simulations montrent qu'elles se stabilisent dans la majorité des cas très rapidement. Les temps de stabilisation théoriques de* $D + 2$ *et* $n + 2$ *rounds, respectivement pour la construction de clusters à* 1 *saut et à* k *sauts, ne sont atteints que sur la configuration particulière de la chaîne dont les nœuds sont ordonnés.*

Ces travaux ont été effectués dans le cadre du co-encadrement avec Olivier Flauzac du doctorat de Bachar Salim Haggar et de Mandicou Ba. Ils ont donné lieu à 2 *revues d'audience internationale,* 4 *conférences d'audience internationale dont 2 Best Paper Award,* 1 *revue d'audience francophone et* 4 *conférences d'audience francophones, toujours avec comité de sélection. Ces doctorants ont respectivement participé au projet national ANR RNRT RISC et au projet régional CPER CapSec-ROFICA.*

2.1 Le clustering

2.1.1 Introduction

Un réseau informatique est composé d'équipements communicants. Ceux-ci ont donc besoin d'avoir des informations sur le chemin à suivre, pour que l'information, qu'ils souhaitent envoyer, atteigne sa destination. Avec la démocratisation d'Internet en France qui a pour conséquence l'augmentation des débits et du nombre d'équipements interconnectés, les routeurs, en charge de véhiculer les informations sur Internet, doivent gérer de plus en plus de chemins différents. Pour cela, il est donc devenu nécessaire de travailler sur des solutions d'optimisation de la taille des tables de routage. Ces optimisations passent par une solution : l'agrégation des réseaux. Ce problème se retrouve également dans les réseaux ad-hoc où chaque équipement joue le rôle de routeur. Or, les réseaux ad-hoc ne sont pas constitués d'équipements d'une grande quantité de mémoire. Une solution, pour pallier à la problématique de l'espace mémoire limité, est celle du regroupement virtuelle des équipements, en groupe appelé cluster. Mes recherches ont donc porté sur les algorithmes de construction de ces clusters dans le but de proposer ensuite de nouveaux algorithmes d'acheminements de l'information sur ces réseaux ainsi structurés.

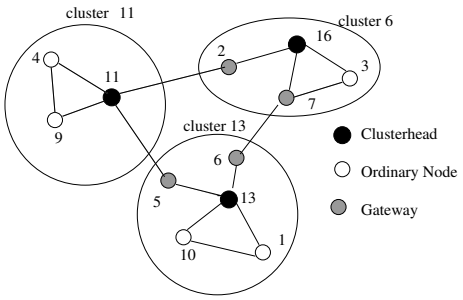


FIGURE 2.1 – Exemple de nœuds

En considérant un réseau comme un graphe dans lequel les liaisons de communication sont représentées par les arêtes du graphe et les équipements communicants par les nœuds du graphe, la majorité des solutions de clustering distinguent au moins 3 types de nœuds : des cluster-head, des nœuds passerelles et des nœuds "ordinaires", comme illustré sur la figure 2.1. Les nœuds dits ordinaires ont pour fonction principale la propagation de l'information. Les nœuds dits passerelles ont des fonctions supplémentaires d'interconnexion de clusters différents. Quant aux nœuds dits cluster-head, ils ont généralement des connaissances supplémentaires comme les identités de tous les nœuds du cluster qu'ils gèrent, les identités des clusters voisins ou bien les identités des nœuds passerelles. A partir de cette distinction entre les différents types de nœuds du graphe, les algorithmes de clusterisation peuvent être classés en plusieurs catégories, comme cela a été proposé par plusieurs auteurs [YC05, WC06, AY07, AM09]. Ainsi il est possible de distinguer les solutions qui se basent sur la construction d'un ensemble dominant [CNGS02], les algorithmes conscients de la mobilité, les algorithmes avec économie de l'énergie ou bien en fonction de critères de construction comme par exemple la métrique de sélection des cluster-heads, le recouvrement ou pas des clusters, la hiérarchisation ou pas des clusters, le type de maintenance des clusters, la taille des clusters. Mais finalement, quelques soient les catégories, comme les algorithmes de clusterisation peuvent appartenir à plusieurs catégories, je préfère classer les solutions de clustering en fonction des objectifs atteints qui peuvent être : l'économie d'énergie, l'homogénéisation de la taille des clusters, la minimisation du rayon le lecture d'un nœud, ... que je présente répondent tous à trois objectifs : avoir un minimum de messages échangés, des critères de décision uniquement basés sur des informations de voisinage direct et des solutions déterministes.

C'est avec ces objectifs que j'ai commencé mes travaux de recherche sur les algorithmes autostabilisants [Dij74] de clusterisation. Ils sont tous conçus pour fonctionner sur des systèmes asynchrones à passage de messages comme définis par Attiya et Welch [AW04].

Après une présentation des principales solutions et stratégies de construction de clusters, je détaillerai mes travaux dans ce domaine. Dans un premier temps, je présenterai des algorithmes de constructions auto-stabilisantes [Dij74] de clusters de diamètre au plus égale à 2. Ces travaux ont débuté avec le co-encadrement avec Olivier Flauzac du doctorat obtenu par Bachar Salim Haggar. Puis dans un second temps, je présenterai une généralisation des constructions auto-stabilisantes de clusters de diamètre au plus égale à $2k$, travaux effectués dans le cadre du co-encadrement avec Olivier Flauzac du doctorat en cours de Mandicou Ba.

2.1.2 Les principales stratégies de la construction de clusters

2.1.2.1 Cluster à 1 saut

Il existe de très nombreuses solutions pour construire des clusters de diamètre au plus égale à 2, également appelés clusters à 1 saut. L'une des plus anciennes solutions de clustering à 1 saut est celle présentée par Ephremides, Wieselthieer et Baker dans [EWB87], appelé Linked Cluster Architecture (LCA). Cette solution construit des clusters non recouvrants mais éventuellement aussi des clusters recouvrants, c'est à dire que certains noeuds peuvent appartenir à de multiples clusters. De plus, ce n'est pas une solution dite auto-stabilisante. Elle fonctionne de plus en 2 phases, une phase de construction des clusters puis une phase de maintenance, comme la solution appelé High-Connectivity Clustering (HCC) de Gerla et Tsai proposée dans [GT95], qui se base sur le nombre de voisin d'un nœud, également appelé degré d'un nœud, pour déterminer le cluster-head. D'autres auteurs ont ensuite amélioré ces solutions dans la majorité des cas soit pour s'adapter au mieux à différents environnements comme les réseaux de capteurs [JN06, LTCH06], les réseaux "quasi-statiques" [Bas99], les réseaux avec une mobilité connue [AP01] (par exemple à l'aide d'un GPS), soit pour étudier différents critères de choix des cluster-heads [LG97, AP01, CDT01, YC03, AJRA08, JN09] soit pour imposer des critères aux clusters comme une meilleure stabilité [BKL01] ou des tailles de clusters fixées [JN08]. Je n'ai fait ici état que des solutions fonctionnant exclusivement pour la construction de cluster à 1 saut et non de celles fonctionnant pour la construction de clusters à k sauts, k pouvant être égale à 1.

Dans [FHN09], nous avons proposé une solution déterministe et auto-stabilisante de construction de clusters de diamètre au plus égale à 2, en au plus $D + 2$ rounds (avec D diamètre du réseau). Chaque cluster est de plus disjoint, c'est à dire qu'aucun nœud ne peut appartenir à plusieurs clusters simultanément et 2 cluster-heads ne sont jamais voisins. Les messages échangés entre les voisins sont de taille au plus égale à $Log(2n + 3)$ où n est le nombre de nœuds du graphe.

2.1.2.2 Cluster à k sauts

Comme pour la construction de clusters à 1 saut, il existe de nombreux algorithmes pour construire des clusters à k sauts dans les 2 principaux modèles utilisés : le modèle à états [DDL09, CDDL10, DLD$^+$12] ou le modèle à passage de messages [MBF04, MFLT05]. De nombreux auteurs ont d'ailleurs étudiés les solutions existantes dans leur "survey" ou travaux pour les réseaux ad hoc [YC05, AN06, AM09, ASJ09, SHN11, HZZW12], pour les réseaux de capteurs [AMC07, AY07, LGF08b, LGF08a, LGF10, LG12, BLM$^+$10, SFH$^+$12] ou pour les réseaux véhiculaires [VAK12]. Par contre, à la différence des solutions pour des clusters à 1 saut, pour les clusters à k sauts, les auteurs se basent également sur la littérature des k-dominating set comme par exemple dans [DDL09, DHR$^+$11]. Un k-dominating set est un ensemble de nœuds définis de la façon suivante [DDL09] : si un nœud ne fait pas parti d'un k-dominating set alors il est à distance au plus de k d'un nœud du k-dominating set. Ceci signifie donc également que tous les cluster-heads sont les nœuds du k-dominating set. La recherche de k-dominating set optimal, c'est à dire de taille minimale, est connu pour être un problème NP-difficile [APVH00] et donc par équivalence, comme celle de la recherche des cluster-heads de clusters à k sauts.

Dans toutes les solutions de clustering présentées dans ce chapitre, le critère choisi pour l'élection du

cluster-head est celui de l'identité maximale. Je me suis concentré sur un critère d'élection ne nécessitant aucune connaissance d'informations voisines et ne variant pas au cours du temps. De plus, comme cela sera montré expérimentalement dans la partie 2.4, ce critère donne de très bons résultats, dans le cadre des réseaux de capteurs.

2.2 Quelques définitions

Dans la suite de ce chapitre, nous utiliserons les notations et définitions suivantes.
- $Neigh_u$: ensemble des voisins du nœud u ;
- C_i : Configuration i ;
- Cl_i : Cluster i ;
- $id(u)$: identité du nœud u ;
- $d(u, v)$: distance entre les nœuds u et v ;
- X_u : Variable X du nœud u ;
- Max : nœud du réseau dont l'identité est la plus grande, i.e., $id(Max)= max\{(id(u), u \in G)\}$;
- $m.x$: variable x contenu dans le message m ;
- G : graphe du réseau ;
- D : diamètre du graphe G.

Définition 2.2.1 (Cluster à 1 saut)
Un cluster est un sous-graphe connexe de diamètre au plus égal à 2 du graphe G.

Définition 2.2.2 (Clustering)
Le terme Clustering désigne le processus de construction des clusters, c'est à dire la sélection des cluster-head et l'affectation des nœuds dans un cluster.

Chaque nœud possède un unique identifiant id et utilise 2 variables : $cl\text{-}id$ et $status$. $cl\text{-}id$ identifie dans quel cluster un nœud se situe et $status$ est utilisé pour définir le type de noœud d'un cluster. Nous utilisons 3 types de nœuds : *cluster-head* noté CH, *ordinaire* noté ON et *passerelle* noté GW, définis comme suit :

Définition 2.2.3 (Cluster-head)
Un nœud u est dit cluster-head ssi $u = max(v/v \in Neigh_u \wedge cl\text{-}id_u = cl\text{-}id_v)$.

Un **nœud passerelle** est un nœud qui est connecté à au moins un autre cluster.

Définition 2.2.4 (Nœud Gateway)
Un nœud u est dit passerelle si et seulement si ($\exists\ v \in Neigh_u$, $cl\text{-}id_u \neq cl\text{-}id_v$).

Tous les nœuds qui n'ont que des voisins appartenant au même cluster que lui-même sont dit nœuds ordinaires.

Définition 2.2.5 (Nœud ordinaire)
Un nœud u est dit ordinaire si et seulement si ($\forall\ v \in Neigh_u$, $v \in Neigh\text{-}cl_u \wedge id(u) < id(v)$).

2.3 Solution de construction auto-stabilisante de clusters à 1 saut

2.3.1 Principe de fonctionnement

La solution que nous avons proposée dans [FHN09] satisfait les 2 propriétés suivantes :
(i) tous les nœuds du réseau doivent appartenir à un unique cluster et
(ii) tous les nœuds d'un cluster sont à une distance au plus de 1 de leur cluster-head.

Nous nous sommes focalisés sur des algorithmes de construction de clusters disjoints dans un but pratique. Chaque nœud n'appartienne qu'à un seul cluster et il est plus aisé de savoir s'il manque une information ou pas dans ses connaissances. Dans le cas de clusters disjoints, il n'est pas possible d'anticiper le nombre de cluster auquel le nœud appartiendra.

Pour cela, nous faisons les hypothèses suivantes :

– Un message envoyé par un nœud est reçu correctement dans un temps fini par son destinataire, à distance au plus de 1.
– Chaque cluster possède un unique cluster-head.

La construction des clusters se fait par échange périodique de messages $hello$ entre chaque nœud. Chaque message $hello$ envoyé par un nœud u contient que 3 variables : id, $status$ et $cl\text{-}id$ comme défini dans la partie 2.2. Ces messages $hello$ sont aussi utilisés par chaque nœud pour annoncer leur présence. Si un nœud ne reçoit aucun message $hello$ de la part d'un nœud voisin durant une certaine période, il considérera alors que ce nœud a disparu. C'est pourquoi chaque nœud attend un certain temps et nous supposons qu'au terme de cette durée, tous les nœuds du réseau ont reçu un message de leurs voisins. A la réception d'un message $hello$, chaque nœud va donc comparer son identité avec celle de l'expéditeur. Le nœud de plus grande identité va s'élire cluster-head et préviendra alors ses voisins. Comme les messages $hello$ ont une variable $status$ qui contient le statut de l'expéditeur, tous les nœuds seront informés du statut de leurs voisins. Le principe de base de la solution est alors le suivant. Lorsqu'un nœud reçoit pour la première fois un message $Hello$ contenant un $status = CH$, il devient alors nœud ordinaire. S'il reçoit d'autres messages de voisins qui sont également cluster-head, il devient donc nœud passerelle. A la fin du processus, chaque nœud aura un statut cluster-head, passerelle ou ordinaire, comme illustré à la figure 2.2.

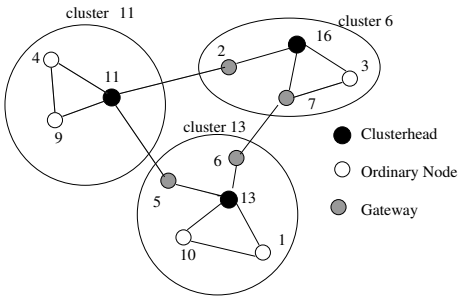


FIGURE 2.2 – Exemple de résultat obtenu

Si 2 nœuds voisins se trouvent être dans un état cluster-head, le 1er nœud qui détecte ce conflit sera en mesure d'abandonner son statut ou bien de le conserver en fonction de la valeur de son identité et de celle de son voisin. En cas d'abandon de l'état cluster-head, ce qui signifie alors que son voisins cluster-head possède une identité supérieure, il deviendra alors nœud ordinaire.

Dans un réseau ad hoc, en raison des modifications topologiques, la solution doit être en mesure de gérer également les apparition, disparition et déplacement de nœuds.

– *Quand un nœud apparaît* dans le réseau, celui-ci envoie un message $hello$ à tous ses voisins et va donc collecter leurs identités et donc exécuter le processus comme un nœud normal.
– *quand un nœud disparaît*, en absence d'expédition de messages $hello$, ses voisins vont s'apercevoir de cette disparition.
– *quand un nœud se déplace*, c'est équivalent à l'apparition et la disparition d'un nœud, comme illustré par les figures 2.3 et 2.4.

2.3.2 Schéma de la preuve

Le principe de la preuve se base sur l'unicité des identités et donc sur l'existence d'un nœud d'identité maximale. Comme chaque nœud compare son identité à l'identité de son voisinage, le nœud d'iden-

Algorithme 1 Algorithme de clusterisation du nœud u

$cl\text{-}id$: identité du cluster dans lequel est le nœud u.
$m.Status$: variable status contenue dans le message m.
id : identité de u
$m.j$: variable j contenue dans le message m.
$m.cl\text{-}id$: variable $cl\text{-}id$ contenue dans le message m.
$Status \in \{CH, ON, GN\}$
R1.a)
if $(Status = CH) \wedge (cl\text{-}id \neq id)$ **then**
  $cl\text{-}id \leftarrow id$;
  $Send\ Hello(id, Status, cl\text{-}id)$;
end if
R1.b)
if $(Status \neq CH) \wedge (\ cl\text{-}id = id \vee (\forall\ m \in Hello, cl\text{-}id \neq m.j) \vee (\exists\ m \in Hello, cl\text{-}id = m.j \wedge m.status \neq$ CH)) **then**
  $Status \leftarrow CH$;
  $cl\text{-}id \leftarrow id$;
  $Send\ Hello(id, Status, cl\text{-}id)$;
end if
R2 : On receiving $Hello(j, Status, cl\text{-}id)$
R2.a)
if $(Status \neq CH) \wedge (\forall\ m \in Hello, m.j < id)$ **then**
  $Status \leftarrow CH$;
  $cl\text{-}id \leftarrow id$;
  $Send\ Hello(id, Status, cl\text{-}id)$;
end if
R2.b)
if $(Status \neq CH) \wedge (\exists\ m \in Hello, m.Status = CH \wedge m.cl\text{-}id > cl\text{-}id)$ **then**
  $Status \leftarrow GN$;
  $cl\text{-}id \leftarrow m.cl\text{-}id$;
  $Send\ Hello(id, Status, cl\text{-}id)$;
end if
R2.c)
if $(Status \neq CH) \wedge (\exists\ m \in Hello, m.Status = CH \wedge m.cl\text{-}id < cl\text{-}id)$ **then**
  $Statut \leftarrow GN$;
  $Send\ Hello(id, Status, cl\text{-}id)$;
end if
R2.d)
if $(Status \neq CH) \wedge (\exists\ m \in Hello, (m.Status = GN \vee m.Status = ON) \wedge m.cl\text{-}id \neq cl\text{-}id)$ **then**
  $Status \leftarrow GN$;
  $Send\ Hello(id, Status, cl\text{-}id)$;
end if
R2.e)
if $(Status \neq CH) \wedge (\exists\ m \in Hello, (m.Status = CH \wedge m.cl\text{-}id = cl\text{-}id)$ **then**
  $Status \leftarrow ON$;
  $Send\ Hello(id, Statut, cl\text{-}id)$;
end if
R3)
if $(Status = CH) \wedge (\exists\ m \in Hello, m.Status = CH \wedge m.j > id)$ **then**
  $Statut \leftarrow ON$;
  $cl\text{-}id \leftarrow m.cl\text{-}id$;
  $Send\ Hello(id, Status, cl\text{-}id)$;
end if
R4)
$Send\ Hello(id, Status, cl\text{-}id)$;

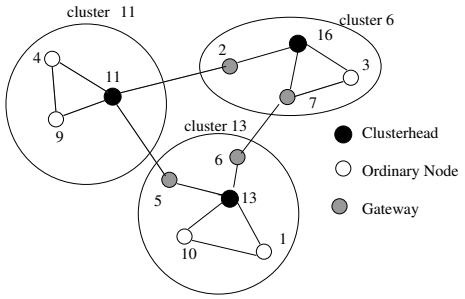


FIGURE 2.3 – Avant le déplacement du nœud 11

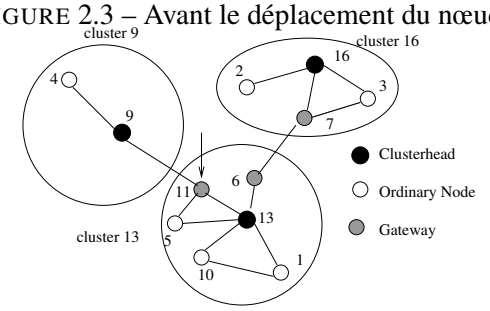


FIGURE 2.4 – Après le déplacement du nœud 11

tité maximale, appelé nœud Max, deviendra cluster-head durant la transition de la configuration initiale à la 1ère configuration. Son état ne changera plus et sera stable. A la configuration suivante, ses voisins vont appartenir au cluster-head du nœud Max. Par récurrence, nous prouvons alors que l'algorithme 1 stabilise en $D + 2$ rounds. Un round est défini de la manière suivante :

Définition 2.3.1 (Round)
Nous appelons un round i, l'exécution qui permet de passer de la configuration C_i à la configuration C_{i+1}.

La preuve complète a été publiée dans [FHN09].

Les solutions de construction de clusters de diamètre au plus de 2 se sont au final montrées peu convaincantes en raison de la quantité importante de clusters à construire sur des réseaux de grandes dimensions. Cette approche s'avère par contre intéressante si nous combinons des clusters de petites tailles avec des clusters de plus grandes tailles, en fonction du type de réseau. Par exemple, il est possible d'envisager un réseau de capteurs personnels, embarqué sur un être humain, qui permet de faire des mesures régulières des constantes physiques et les transmettre, sur un réseau plus complexe. Ce genre de capteur existe dans l'armée ou dans les équipes d'intervention d'urgence en milieu hostile afin d'avoir un suivi à distance des secouristes. Il est alors possible d'envisager que l'être humain constitue alors un cluster à 1 saut car tous les capteurs seront distance 1 du cluster-head et chaque cluster peut être interconnecté au sein d'un cluster de plus grande traille. Je me suis donc ensuite intéressé aux solutions de clustering à k sauts, c'est à dire des clusters de diamètre au plus égal à $2k$.

2.4 Solution de construction auto-stabilisante de clusters à k sauts

A la suite des travaux effectués sur la construction de clusters à 1 saut et dans le cadre du co-encadrement avec Olivier Flauzac du doctorat de Mandicou BA et du projet CPER CapSec ROFICA, j'ai mené des recherches sur la construction de clusters à k sauts. En se basant toujours sur le même critère de sélection des cluster-heads, c'est à dire sur l'identité maximale, dans [BFH$^+$12d, BFH$^+$12c, BFH$^+$12a, BFH$^+$12b, BFH$^+$13], nous avons proposé une nouvelle solution de clustering à k sauts, appelé SDEAC. Par simulations, nous avons également mesuré les impacts des variations de certaines

caractéristiques des réseaux (le degré et la taille) sur le temps de stabilisation. Puis dans [BFM$^+$13a], nous l'avons comparée avec les meilleures solutions de clustering à k sauts du moment et dans un modèle comparable. Nous avons ensuite effectué dans [BFM$^+$13d] une adaptation de notre solution pour les réseaux de capteurs. Je vais maintenant présenter l'algorithme puis les résultats des simulations.

La solution proposée est toujours auto-stabilisante et fonctionne par échange de messages. Notre algorithme structure le réseaux en clusters sans chevauchement et de diamètre au plus de $2k$. Il n'utilise que des informations des voisins, contrairement à d'autres solutions comme [MFLT05] qui a besoin de connaître des informations sur les nœuds situés à distance supérieure à 2. Les clusters sont construits en au plus $n + 2$ rounds et l'occupation mémoire par nœud est d'au plus $n * \log(2n + k + 3)$, où n est le nombre de nœuds du réseau.

Définition 2.4.1 (Stabilité d'un nœud)
Un nœud u est dit stable si et seulement si ses variables ne changent plus au cours du temps, s'il est cohérent et s'il vérifie l'une des conditions suivantes :

- si $status_u = CH$ alors $\forall v \in N_u, (status_v \neq CH) \wedge \{((cl_v = cl_u) \wedge (id_v < id_u)) \vee ((cl_v \neq cl_u) \wedge (dist_{(v,CH_v)} = k))\}$. (Exemple du nœud 9 dans le cluster V_9, figure 2.5)
- si $status_u = SN$ alors $\forall v \in N_u, (cl_v = cl_u) \wedge (dist_{(u,CH_u)} \leq k) \wedge (dist_{(v,CH_v)} \leq k)$. (Exemple du nœud 0 dans le cluster V_{10} ou d nœud 7 dans V_9 figure 2.5).
- si $status_u = GN$ alors $\exists v \in N_u, (cl_v \neq cl_u) \wedge \{((dist_{(u,CH_u)} = k) \wedge (dist_{(v,CH_v)} \leq k)) \vee ((dist_{(v,CH_v)} = k) \wedge (dist_{(u,CH_u)} \leq k))\}$. (Exemple du nœud 2 dans le cluster V_9 ou nœud 8 dans V_{10} figure 2.5)

Définition 2.4.2 (Réseau stable)
Le réseau est dit *stable* si et seulement si tous les nœuds sont stables.

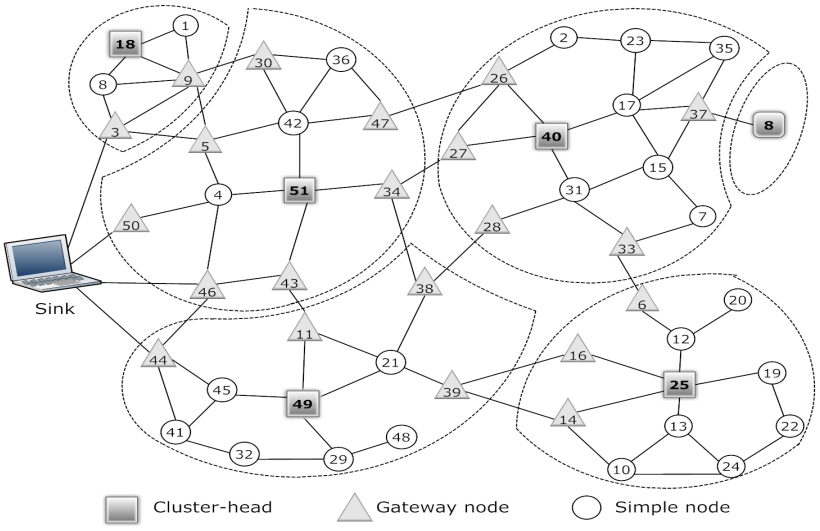


FIGURE 2.5 – Structuration du réseau en clusters

2.4.1 Principe de fonctionnement

La différence entre la solution de clusters à 1 saut et celle à k sauts, l'algorithme SDEAC, est dans le contenu du message $hello$. En effet, il est nécessaire de connaître le plus court chemin séparant un nœud de son cluster-head afin de construire un cluster de diamètre au plus de $2k$. Chaque message contient donc maintenant 4 informations : l'identité du nœud, l'identité du cluster, le statut du nœud et la distance jusqu'au cluster-head. A la réception d'un message $hello$, un nœud u exécute alors l'algorithme 2. Au

cours de cette exécution, 3 phases successives ont lieu : la mise à jour de la table de voisinage, la vérification de la cohérence des informations et finalement la participation à la construction du cluster.

Les phases de mise à jour de la table de voisinage et la vérification de la cohérence sont similaires à la solution pour les clusters à 1 saut. Pour la construction des clusters, c'est toujours l'identité maximale qui permet d'élire le cluster-head mais cette fois, la distance entre un nœud et son cluster-head va être vérifiée.

Chaque nœud u utilise les variables suivantes : k : distance maximale entre un nœud et son cluster-head
N_u : voisinage à distance 1 du nœud u
$StateNeigh_u$: Table des états des voisins de u ; $StateNeigh_u[v]$ contient l'état du nœud v voisin de u
cl_u : cluster identity of node u
id_u : identité du nœud u
$status_u \in \{CH, SN, GN\}$: état du nœud u
$dist_{(u,CH_u)}$: distance du nœud u à son cluster-head
gn_u : identité du nœud qui permet d'atteindre son cluster-head

2.4.2 Schéma de preuve

La méthodologie pour prouver la convergence de l'algorithme 2 est identique à celle utilisée pour l'algorithme 1 à 1 saut. Nous utilisons le principe de l'unicité des identités et donc de l'existence d'un nœud d'identité maximale appelé nœud Max. Nous prouvons ensuite par récurrence que le nombre d'éléments de l'ensemble des nœuds dits non fixés, noté $\overline{\mathcal{F}_i}$ à la configuration γ_i, (illustré par la figure 2.6) décroit strictement quand i augmente. Plus précisément, nous avons publié la totalité de la preuve formelle dans [BFH+13] et montré que notre algorithme stabilise en au plus $n + 2$ rounds.

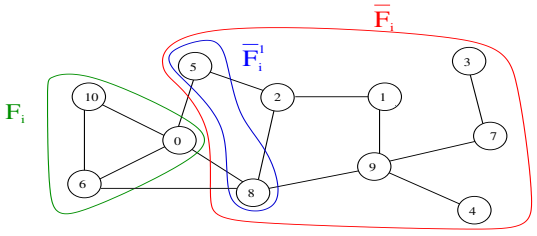


FIGURE 2.6 – Exemple de nœuds fixés et non fixés

Remarque 2.4.1
Avec cette approche, le pire des cas est rencontré quand les nœuds sont ordonnés sur une chaîne, comme illustré à la figure 2.7(a). Dans cette topologie, le temps de stabilisation est d'au plus $n + 2$ transitions. Cependant, dans la majorité des cas, sur des topologies quelconques, comme illustré à la figure 2.7(b), le temps de stabilisation est inférieur à $n + 2$, comme je vais le montrer dans la partie suivante 2.4.3.

Pour fonctionner, chaque nœud a besoin de connaitre l'identité de ses voisins ainsi que leur état. Il a également besoin de connaitre la distance qui sépare un nœud de son cluster-head et l'identité de son cluster. Pour chaque nœud, l'algorithme a donc besoin d'au plus $\log(2n + 3 + k)$ bits, soit au total, $n * \log(2n + 3 + k)$.

Algorithme 2 SDEAC : Algorithme de construction de clusters à k sauts

/ Upon receiving message from a neighbor */* $NeighCH_u = \{id_v / v \in N_u \wedge status_v = CH \wedge cl_u = cl_v\}$.
$NeighMax_u = (Max\{id_v / v \in N_u \wedge status_v \neq CH \wedge cl_u = cl_v\}) \wedge (dist_{(v,CH_u)} = Min\{dist_{(x,CH_u)}, x \in N_u \wedge cl_x = cl_v\})$.

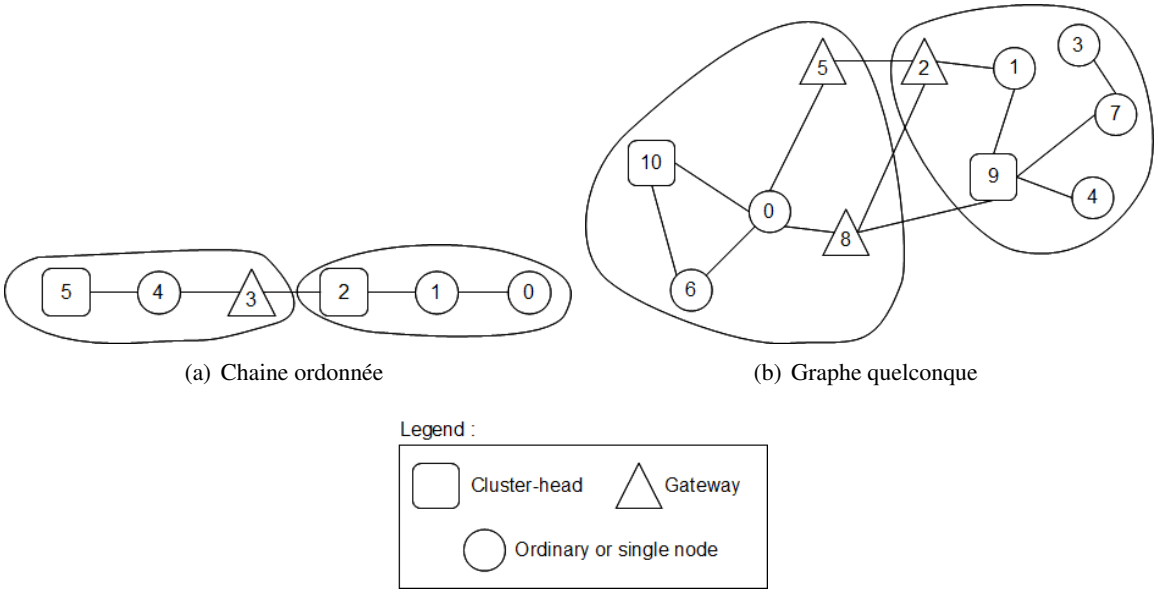
Predicates $P_1(u) \equiv (status_u = CH)$
$P_2(u) \equiv (status_u = SN)$
$P_3(u) \equiv (status_u = GN)$
$P_{10}(u) \equiv (cl_u \neq id_u) \vee (dist_{(u,CH_u)} \neq 0) \vee (gn_u \neq id_u)$
$P_{20}(u) \equiv (cl_u = id_u) \vee (dist_{(u,CH_u)} = 0) \vee (gn_u = id_u)$
$P_{40}(u) \equiv \forall v \in N_u, (id_u > id_v) \wedge (id_u \geq cl_v) \wedge (dist_{(u,v)} \leq k)$
$P_{41}(u) \equiv \exists v \in N_u, (status_v = CH) \wedge (cl_v > cl_u)$
$P_{42}(u) \equiv \exists v \in N_u, (cl_v > cl_u) \wedge (dist_{(v,CH_v)} < k)$
$P_{43}(u) \equiv \forall v \in N_u / (cl_v > cl_u), (dist_{(v,CH_v)} = k)$
$P_{44}(u) \equiv \exists v \in N_u, (cl_v \neq cl_u) \wedge \{(dist_{(u,CH_u)} = k) \vee (dist_{(v,CH_v)} = k)\}$

Rules
/ Update neighborhood */*
$StateNeigh_u[v] := (id_v, cl_v, status_v, dist_{(v,CH_v)})$;

/ Cluster-1 : Coherent management */*
$R_{10}(u) : : P_1(u) \wedge P_{10}(u) \longrightarrow cl_u := id_u; gn_u = id_u; dist_{(u,CH_u)} = 0$;
$R_{20}(u) : : \{P_2(u) \vee P_3(u)\} \wedge P_{20}(u) \longrightarrow status_u := CH; cl_u := id_u; gn_u = id_u; dist_{(u,CH_u)} = 0$;

/ Cluster-2 : Clustering */*
$R_{11}(u) : : \neg P_1(u) \wedge P_{40}(u) \longrightarrow status_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; gn_u := id_u$;
$R_{12}(u) : : \neg P_1(u) \wedge P_{41}(u) \longrightarrow status_u := SN; cl_u := id_v; dist_{(u,v)} := 1; gn_u := NeighCH_u$;
$R_{13}(u) : : \neg P_1(u) \wedge P_{42}(u) \longrightarrow status_u := SN; cl_u := cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; gn_u := NeighMax_u$;
$R_{14}(u) : : \neg P_1(u) \wedge P_{43}(u) \longrightarrow status_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; gn_u := id_u$;
$R_{15}(u) : : P_2(u) \wedge P_{44}(u) \longrightarrow status_u := GN$;
$R_{16}(u) : : P_1(u) \wedge P_{41}(u) \longrightarrow status_u := SN; cl_v := id_v; dist_{(u,v)} := 1; gn_u := NeighCH_u$;
$R_{17}(u) : : P_1(u) \wedge P_{42}(u) \longrightarrow status_u := SN; cl_u := cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; gn_u := NeighMax_u$;

/ Sending hello message */*
$R_0(u) : : hello(id_u, cl_u, status_u, dist_{(u,CH_u)})$;



(a) Chaine ordonnée (b) Graphe quelconque

FIGURE 2.7 – Exemple de cluster à 2 sauts

2.4.3 Analyse des performances

L'algorithme 2 stabilise en au plus $n+2$ rounds. Hors, dans la majorité des cas, en raison du caractère distribué de son exécution, il s'avère que le temps de stabilisation est bien inférieur à cette valeur de $n+2$. Pour s'en convaincre, un ensemble de simulation ont été réalisé avec *OMNeT++* [1] [VH08].

2.4.3.1 Impact de la densité de connexité du réseau

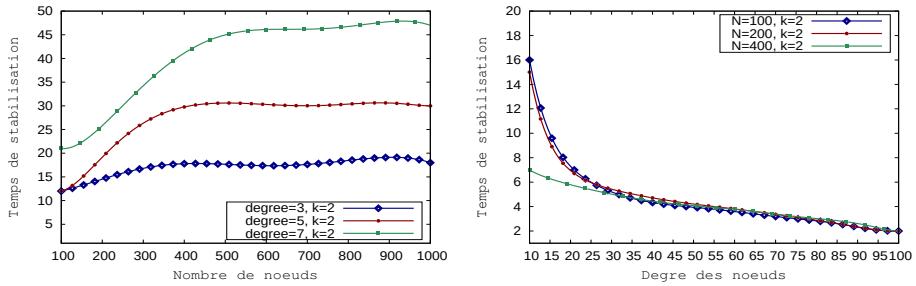
Nous étudions l'impact du degré (nombre de voisins) et du nombre de nœuds sur le temps de stabilisation du réseau. Nous fixons un degré de 3, 5 et 7, le paramètre k à la valeur 2 et nous faisons varier le nombre de nœuds de 100 à 1000 par pas de 100. Nous fournissons des valeurs moyennes. Au niveau de la figure 2.8(a), nous remarquons que le temps de stabilisation augmente légèrement avec l'augmentation du nombre de nœuds et varie peu à partir d'une certaine taille du réseau. De plus, nous notons que pour des topologies quelconques, le temps de stabilisation moyen est très en dessous de $n + 2$, valeur formelle prouvée dans le pire des cas.

Pour mieux observer l'impact de la densité du réseau comme illustré avec la figure 2.8(b), nous fixons le nombre de nœuds et nous faisons varier le degré. Nous observons que pour un nombre de nœuds fixé à 100, 200 et 400, plus le degré des nœuds augmente, plus le temps de stabilisation diminue. En effet, si le degré augmente, les nœuds ayant les plus grandes identités ont plus de voisins. Ils "absorbent" donc plus de nœuds dans leurs *clusters* durant chaque transition. Ainsi, avec notre approche, plus le réseau est dense, plus nous avons de meilleurs temps de stabilisation. Or, les réseaux *ad hoc* sont souvent caractérisés par une forte densité.

2.4.3.2 Impact du paramètre k

Pour analyser l'impact du paramètre k, nous fixons arbitrairement un degré de 5 et nous considérons des réseaux de taille 100, 200, 400, 500 et 1000 nœuds. Pour chaque taille de réseau, nous faisons varier la valeur de k de 2 à 10. La figure 2.9(b) montre le temps de stabilisation en fonction de la valeur de

1. http ://www.omnetpp.org



(a) Temps de stabilisation en fonction du nombre de noeuds

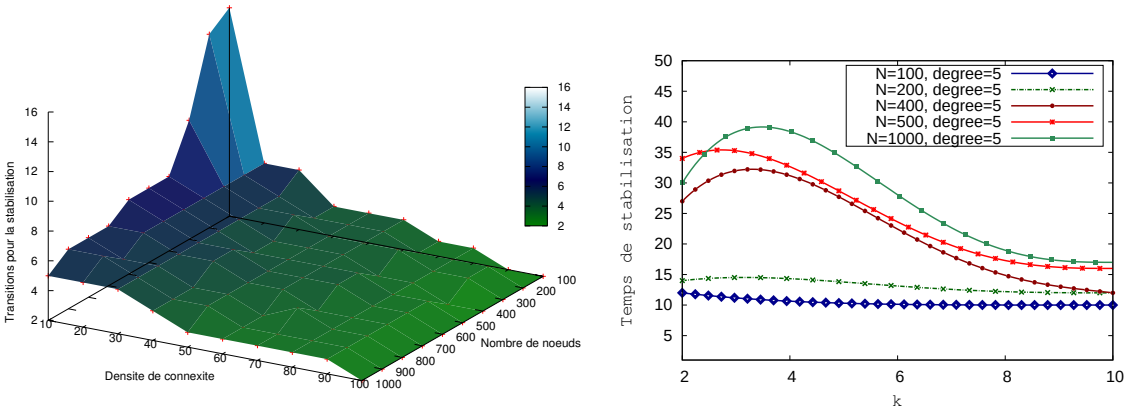
(b) Temps de stabilisation en fonction du degré des noeuds

FIGURE 2.8 – Impact de la densité et du nombre de nœuds sur le temps de stabilisation

k. Nous observons une diminution du temps de stabilisation avec l'augmentation de la valeur de k. En effet, comme les messages $hello$ échangés contiennent la valeur de la distance k, si k augmente, le champs d'influence des nœuds avec une plus grande identité augmente. Les nœuds effectuent moins de transitions pour se *fixer* à un CH. Avec de petites valeurs du paramètre k, nous avons des *clusters* de faibles diamètres. Donc, il nécessite plus de transitions pour atteindre un état stable dans tous les *clusters*. Notons que quelle que soit la valeur du paramètre k et pour des graphes de réseaux totalement aléatoires, nous obtenons des temps de stabilisation très inférieurs au pire des cas ($n + 2$ transitions).

2.4.3.3 Étude du passage à l'échelle

Pour étudier le passage à l'échelle de notre approche, nous faisons varier le nombre de nœuds dans le réseau en même temps que la densité de connexité, c'est à dire le nombre moyen de voisins (degré) qu'un nœud peut avoir.



(a) Passage à l'échelle

(b) Impacte du paramètre k

FIGURE 2.9 – Passage à l'échelle - Impact du paramètre k

Pour $k = 2$, nous faisons évoluer le nombre de nœuds de 100 à 1000 par pas de 100. Et pour chaque valeur de nombre de nœuds, nous faisons varier la densité du réseau de 10% à 100% par pas de 10. Nous obtenons la courbe *3D* de la figure 2.9(a). Nous remarquons que sauf pour de faibles densités (10% et 20%), le temps de stabilisation varie légèrement avec l'augmentation du nombre de nœuds. Mais avec l'augmentation du nombre de nœuds, le temps de stabilisation diminue et nous observons le même

phénomène que la figure 2.8(b). Avec cette série de simulations, nous pouvons soulever deux remarques. *(i)* Seule la densité de connexité est le facteur déterminant avec notre approche. *(ii)* En moyenne, pour des réseaux avec une topologie quelconque, le temps de stabilisation est très inférieur à celui du pire des cas ($n + 2$ transitions). Rappelons que ce pire cas est une topologie où les nœuds forment une chaine ordonnée comme nous l'avons prouvé dans [BFH$^+$13].

2.4.4 Comparaison des approches

Dans [BFM$^+$13c, BFM$^+$13a] nous avons effectué une comparaison entre les caractéristiques de notre solution et les meilleures solutions connues dans le modèle à états que sont les algorithmes de A.K. Datta et *al.* décrit dans [DDL09, CDDL10]. Puis nous avons mené des simulations avec l'algorithme de N. Mitton et *al.* décrit dans [MFLT05] car écrit dans le modèle à passage de messages, modèle similaire au notre.

Dans [MFLT05], N. Mitton et *al.* étendent leurs travaux décrits dans [MBF04] pour proposer un algorithme robuste de clustering auto-stabilisant à k sauts. Chaque nœud recueille les informations dans son voisinage à distance $k + 1$, calcule sa k densité et la diffuse à son voisinage à distance k. Le nœud avec la plus grande densité devient cluster-head.

2.4.4.1 Étude analytique

Dans [BFH$^+$13], nous avons prouvé que partant d'une configuration quelconque, une configuration légale est atteinte au plus en $n+2$ transitions. De plus, notre approche nécessite une occupation mémoire de $\log(2n + k + 3)$ par nœud ; donc $n * \log(2n + k + 3)$ espace mémoire au plus pour un réseau de n nœuds.

Le tableau 2.1 illustre une comparaison du temps de stabilisation, de l'occupation mémoire et du voisinage entre notre approche et celles basées sur un modèle à états.

	Temps de stabilisation	Occupation mémoire / nœud	Voisinage
Notre approche	$n + 2$	$\log(2n + k + 3)$	*1 saut*
Datta et al. [DDL09]	$O(n)$, $O(n^2)$	$O(\log(n))$	*k sauts*
Datta et al. [CDDL10]	$O(n * k)$	$O(log(n) + log(k))$	*k+1 sauts*

TABLE 2.1 – Comparaison entre temps de stabilisation et occupation mémoire

Notre temps de stabilisation ne dépend pas du paramètre k contrairement à la solution [CDDL10]. Comme nous avons une unique phase de découverte de voisinage et une construction simultanée des clusters à k sauts, nous avons un unique temps de stabilisation contrairement à l'approche décrite dans [DDL09]. De plus nous considérons uniquement un voisinage à distance 1 à l'opposé des solutions [DDL09, CDDL10].

2.4.4.2 Étude par simulation

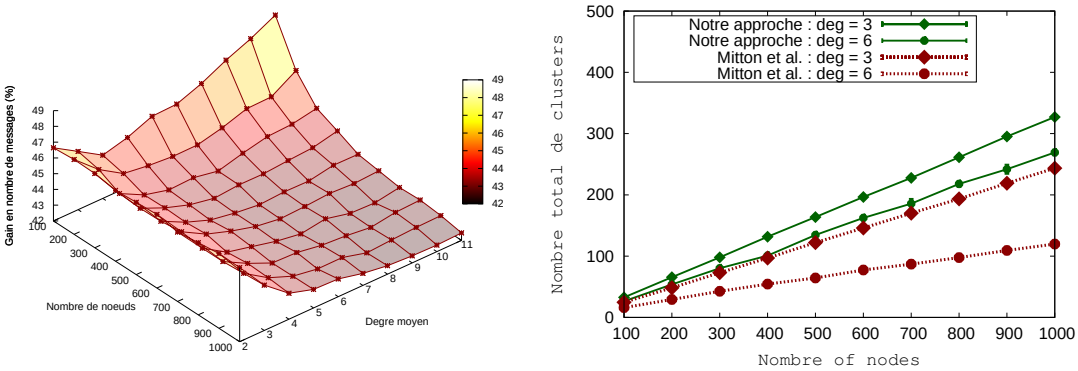
Dans cette section, nous donnons une implémentation et une évaluation de notre approche par simulation. De plus, nous comparons notre approche avec la proposition de N. Mitton et *al.* [MFLT05].

Comme nous l'avons montré dans [BFH$^+$13], notre réseau converge en au plus $n+2$ transitions. Ceci correspond au cas le plus défavorable dans une topologie où les nœuds forment une chaine ordonnée. Or, un réseau *ad hoc* est caractérisé par une topologie dense et aléatoire. C'est ainsi que nous avons réalisé une campagne de simulations avec *OMNeT++* [VH08] et la librairie *SNAP* [*07*], pour évaluer les performances moyennes de notre algorithme.

Il existe plusieurs propositions basées sur un modèle à passage de messages. Nous avons comparé notre approche avec la solution de N. Mitton et *al.* [MFLT05], connu pour être la meilleure solution au moment des expérimentations. Les comparaisons effectuées ont été faites sur le nombre de messages et le nombre de clusters, avec le simulateur *OMNeT++* et en générant des graphes aléatoires suivant la même loi et les mêmes caractéristiques que celles utilisées dans [MFLT05].

Pour évaluer le nombre de messages échangés, nous avons fixé un degré de 3 et 6 et nous faisons varier la taille du réseau de 100 à 1000 nœuds. Nous avons également fixé k à la valeur 2. Nous donnons le nombre moyen de messages échangés pour atteindre un état légal (formation des *clusters* stables). Comme illustré par la figure 2.10(a), notre approche obtient un gain d'environ 50% sur le nombre de messages échangés par rapport à la solution [MFLT05]. La principale raison est que notre approche se base uniquement sur l'information provenant du voisinage à distance 1 pour construire des *clusters* de diamètre au plus $2k$ (4 dans nos simulations car k est fixé à 2). Alors que pour la solution [MFLT05], chaque nœud doit connaître le voisinage à distance $k+1$, calculer sa *k-densité* puis la diffuser vers son voisinage à distance k.

Nous avons aussi évalué le nombre de *clusters* obtenus avec les deux approches. Comme illustré au niveau de la figure 2.10(b), nous construisons plus de *clusters* en générant moins de messages.



(a) Gain de notre algorithme par rapport à N. Mitton et *al.*

(b) Nombre de clusters

FIGURE 2.10 – Comparaison avec N. Mitton et *al.* [MFLT05]

Notons que nous avons effectué les comparaisons pour $k = 2$. Quand la valeur de k augmente, le champs de diffusion de la solution [MFLT05] augmente. Or, notre approche se base toujours sur l'information du voisinage à distance 1 pour construire des *clusters* à *k sauts*.

2.5 Optimiser la gestion énergétique

2.5.1 Adaptation aux réseaux de capteurs

Dans [BFM$^+$13d, BFM$^+$13b], une adaptation de notre algorithme 2 a été effectuée afin de pouvoir faire fonctionner notre solution sur un réseau de capteurs. Pour cela, nous avons donc comparé plusieurs critères de sélection du clusterhead et déterminé la consommation énergétique dans chacun des cas.

2.5.1.1 Méthodes d'élection des cluser-heads dans le cas d'un réseau de capteurs sans fil

Les approches de clusterisation, proposées dans la littérature et comparées entre elles, utilisent un ou plusieurs critères pour le choix d'un clusterhead (identifiant, degré, densité, mobilité, distance, temps de service, sécurité, nature des informations collectées ou une combinaison de ces critères). Cependant, à

notre connaissance, il n'existe aucun travail dans lequel la même approche de clusterisation proposée est comparée dans le cas d'utilisation de différents critères d'élection des clusterheads. Or, afin de mieux choisir l'un des critères d'élection, il est important d'étudier l'influence de chacun dans les mêmes conditions de test et, idéalement, sur le même framework. Pour cela, nous proposons une approche de clustering générique, auto-stabilisante et distribuée, et nous comparons son coût et sa performance en considérant les critères les plus utilisés pour le choix des CH, à savoir : l'identifiant, le degré et l'énergie résiduelle des nœuds.

Identifiant maximal

Dans cette approche, le nœud ayant le plus grand degré (nombre de voisins) est choisi comme clusterhead. Cette méthode est sensée limiter les communications en construisant des *clusters* denses ;

Degré idéal

Dans cette approche, la décision concernant le choix du nœud le plus adéquat pour devenir clusterhead se fait en fonction du degré D (nombre de voisins) des nœuds. Il existe deux types d'approches se fondant sur le degré, celles privilégiant le nœud ayant le degré maximal et celles privilégiant le nœud ayant le degré $\triangle_d$ qui se rapproche le plus du degré idéal ρ. Les études précédentes, menées par exemple par [GT95, CDT01, CW06, BAR07], ont montré que lors de la phase de routage, un nœud qui a un degré très élevé consomme son énergie très rapidement à cause du nombre de messages qu'il reçoit/envoie à ses voisins. Pour mieux répartir la consommation d'énergie, il est donc important de limiter le nombre de voisins du CH. Le meilleur candidat est celui qui a la plus petite valeur de $\triangle_d$, calculée selon l'équation suivante : $\triangle_d = |D - \rho|$

Dans le cas où deux ou plusieurs nœuds ont le même degré relatif $\triangle_d$ et sont tous deux candidats à la fonction de CH, le choix se fait alors en fonction d'un critère secondaire qui est l'identifiant maximal des nœuds. Ce critère est discriminant dans la mesure où chaque nœud a un identifiant unique.

Énergie résiduelle

Dans cette approche, le choix des cluster-head se fait en fonction de l'énergie résiduelle dans les batteries des nœuds. En effet, le cluster-head est généralement le nœud le plus sollicité dans la fonction de routage et a donc besoin d'une réserve d'énergie plus importante que celle des autres nœuds, dits simples ou ordinaires, afin d'éviter la reconstruction fréquente des clusters.

Un candidat à la fonction de cluster-head avec initialement le maximum d'énergie résiduelle en consomme continuellement une partie au fur et à mesure de l'avancement de la procédure de clusterisation. Après un certain temps, il risque d'avoir moins d'énergie que l'un de ses voisins et cela provoque des itérations supplémentaires visant à élire un autre nœud cluster-head. Afin d'éviter de changer fréquemment de candidat à cause d'une différence en énergie pouvant être négligeable et afin de garantir plus de stabilité à la procédure de clusterisation, nous considérons une variante de cette approche. Elle se fondent sur un seuil d'énergie E_T, pouvant être appelé tolérance, dans la différence énergétique restante entre des nœuds voisins. Ainsi, tant que la différence d'énergie entre le candidat initial et le nouveau ($\triangle_e = |E_i - E_j|$) est inférieure au seuil d'énergie E_T, le candidat initial conserve son statut de candidat idéal pour devenir cluster-head. Cette approche a été exploitée par M. Bsoul et *al.* dans [BAKAO13]. Nous pouvons noter que ce type de critère est également utilisé dans le protocole de routage EIGRP [SSN+13]. Quand 2 routes existent vers une même destination et que leurs métriques sont proches, une tolérance peut être configurée afin de signaler au routeur que les 2 routes doivent être considérées comme ayant la même métrique.

Combinaison des critères

Le choix des cluster-head se fait en fonction des deux critères précédents combinés. Ainsi, un nœud i est choisi comme cluster-head s'il maximise une valeur U_i représentant le nœud ayant la plus grande quantité d'énergie et le degré le plus proche du degré idéal. Cette approche permet également de considérer les préférences des utilisateurs à travers des poids qui déterminent le degré d'importance de chaque critère. Pour cela, nous nous fondons sur une méthode de prise de décision MADM (Multiple Attribute Decision-Making) nommée SAW (Simple Additive Weighting) [Zha04]. SAW est une technique de la

théorie de décision, efficace, simple à mettre en œuvre, peu coûteuse et facile à intégrer. Elle consiste en une somme pondérée des critères, comme le montre l'équation 2.1 qui suit :

$$U_i = w_e * E_r + w_d * \triangle_d \tag{2.1}$$

Dans cette équation w_e et w_d représentent les poids attribués à chaque critère, avec $\sum_{j=1}^{m} w_j = w_e + w_d = 1$. Afin de normaliser les valeurs utilisées dans l'équation, nous procédons comme suit :

$$Gain\ à\ maximiser\ (E_r) \ \Rightarrow \ \ r_{ij} = x_{ij}/x_j^{max} \tag{2.2}$$

$$Coût\ à\ minimiser\ (\triangle_d) \ \Rightarrow \ \ r_{ij} = x_j^{min}/x_{ij} \tag{2.3}$$

Lorsque l'écart entre les valeurs de U de deux nœuds ne dépasse pas un certain seuil, alors le choix se fait selon un critère secondaire qui est l'identité des nœuds.

2.5.2 Résultats obtenus

Les paramètres des simulations ont été choisis afin qu'ils correspondent à des valeurs déjà citées par d'autres auteurs. Le tableau 2.2 résume tous ces paramètres et les publications qui utilisent les mêmes valeurs. Les résultats que je reprends dans la suite du paragraphe ont été publiés en détail dans [BFM+13b].

TABLE 2.2 – Paramètres de simulation

Parameter	Value
Taille des messages [HCB00]	2000 bits
Distance entre 2 nœuds [HCB00]	100 mètres
Énergie initiale $\mathcal{E}_i^{init}$ [HCB00]	{1,2,3} Joules
Degré idéal [GT95, CDT01, CW06, BAR07]	{6,10,12,20}
Seuil d'énergie [BAKAO13]	{0.1,0.05} %
Nombre de nœuds	[100,1000]
Modèle de génération de graphe	Loi de Poisson
Moyenne des degrés [BC03, HLS05, MFLT05, CKS10, NQL11]	[2,11]
Paramètre k	[1,10]
Nombre de simulations pour chaque taille de réseau	100

Dans toutes nos simulations présentées ci-dessous, tous nos résultats sont obtenus avec une probabilité de 99%. Cette probabilité est appelé l'intervalle de confiance et les formules mathématiques qui permettent d'obtenir cette intervalle de confiance peuvent être trouvées dans [Jai91].

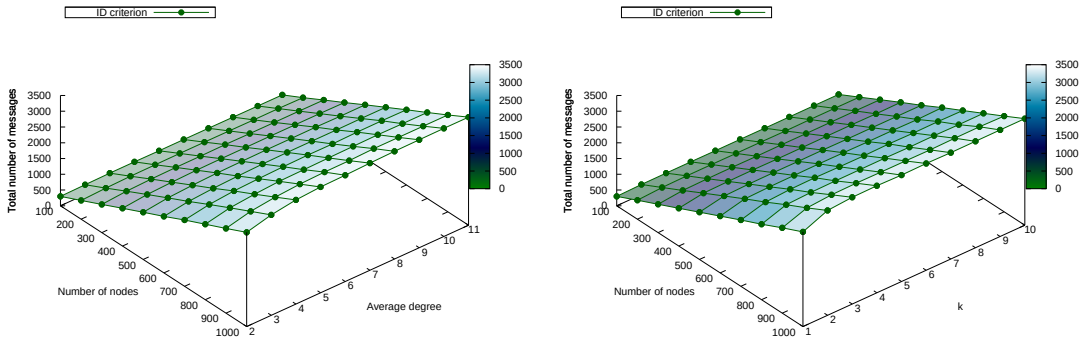
– Le nombre total de messages échangés ($\mathcal{M}_{total}$) est défini comme le nombre total de messages échangés dans le réseau jusqu'à ce que les clusters soient formés.

$$\mathcal{M}_{total} = \sum_{i=0}^{n-1} \mathcal{M}_i^{Send}$$

où $\mathcal{M}_i^{Send}$ est le nombre total de messages envoyés par le nœud i et n représente la taille du réseau.

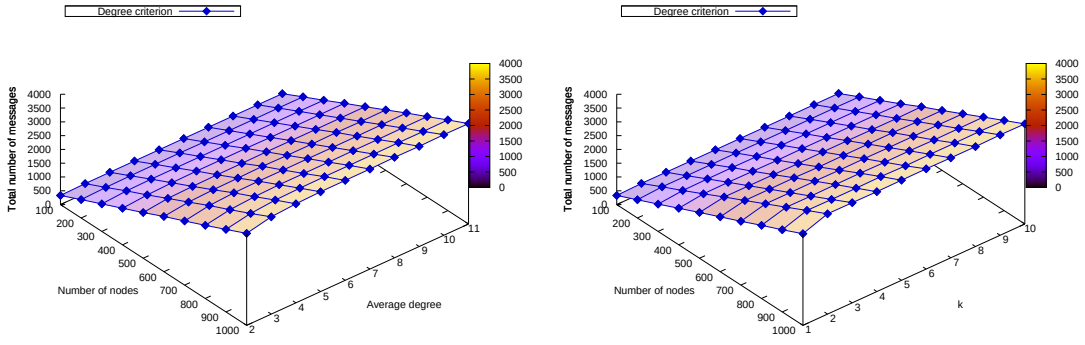
– La consommation totale d'énergie ($\mathcal{E}_{total}$) est définie comme la consommation énergétique nécessaire pour la construction des clusters.

$$\mathcal{E}_{total} = \sum_{i=0}^{n-1} (\mathcal{E}_i^{init} - \mathcal{E}_i^{av})$$



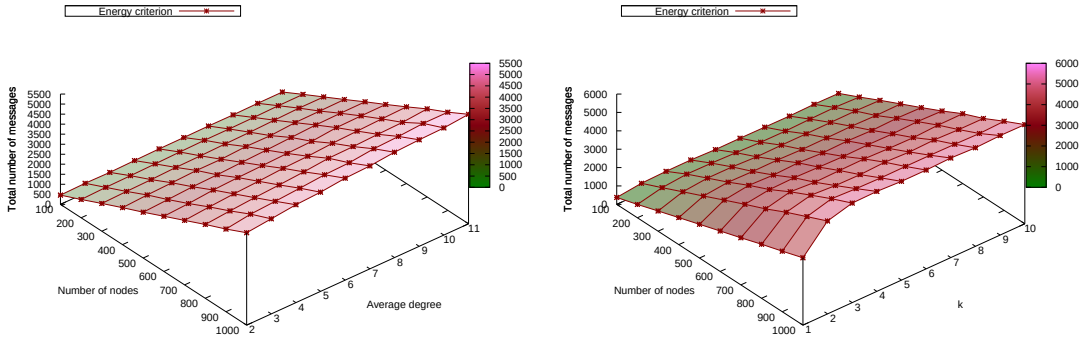
(a) Critère de l'identité en fonction du degré moyen

(b) Critère de l'identité en fonction du paramètre k



(c) Critère du degré en fonction du degré moyen

(d) Critère du degré en fonction du paramètre k



(e) Critère de l'énergie en fonction du degré moyen

(f) Critère de l'énergie en fonction du paramètre k

FIGURE 2.11 – Nombre total de messages échangés en fonction du degré moyen et k

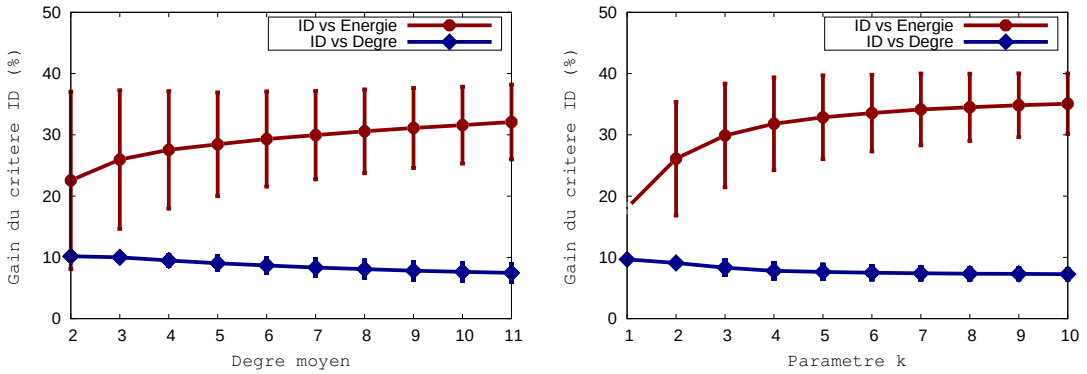
où $\mathcal{E}_i^{init}$ est l'energie initiale du nœud i et $\mathcal{E}_i^{av}$ est l'énergie disponible du nœud i à la fin de la construction du cluster.

– Le nombre de clusters est défini comme le pourcentage de clusters construit par rapport à la taille du réseau. Par exemple, si nous obtenons 10% de clusters construits sur un réseau de 100 nœuds, cela signifie que nous avons au total 10 clusters.

2.5.2.1 Coût des communications (messages) et consommation énergétique

La première évaluation de l'algorithme de construction de clusters à k sauts a été faite sur le nombre de messages échangés afin d'atteindre la stabilisation. Dans l'ensemble des expérimentations dont les résultats sont visibles sur la figure 2.11, le coût des communication est évalué, pour chaque critère, en fonction du degré moyen des nœuds (Fig 2.11(a), Fig 2.11(c) et Fig 2.11(e)) et en fonction de k (Fig 2.11(b), Fig 2.11(d) et Fig 2.11(f)). Ces simulations sont basées sur le même réseau pour chaque valeur de degré moyen et k.

Comme illustré dans les courbes des figures 2.11(a), 2.11(b), 2.11(c), 2.11(d), 2.11(e) et 2.11(f), nous observons que pour chaque critère d'élection des clusterhead, le nombre total de messages échangés augmente linéairement avec le nombre de nœuds du réseau. Cependant, la figure 2.11 montre que le degré moyen et k n'ont pas d'influence sur la quantité de messages échangés. La principale raison est que notre protocole est basé seulement sur les informations des nœuds voisins à distance 1 pour construire les clusters.



(a) Gain en terme de cout de communication avec le critère de l'ID

(b) Gain

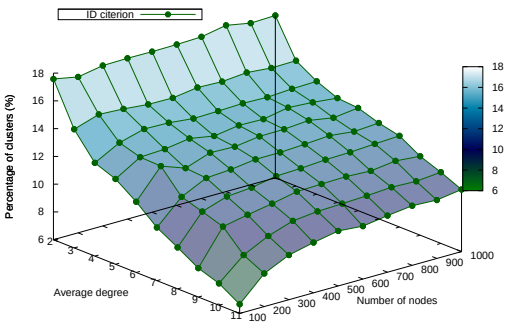
FIGURE 2.12 – % de gain avec le critère de l'identité

Les simulations de la figure 2.11 montrent que la clusterisation basée sur le critère de l'identité génère moins de messages. La figure 2.12(a) montre le gain offert par le critère de l'identité comparé au critère de l'énergie en fonction du paramètre λ pour $k = 2$. Le critère de l'identité permet d'obtenir un gain du coût des communication entre 7.5% et 10.2% comparativement au critère du degré et entre 22.6% and 32.1% comparativement au critère de l'énergie. La principale raison vient du fait que le critère de l'identité est un critère plus stable dans le temps par rapport aux autres critères.

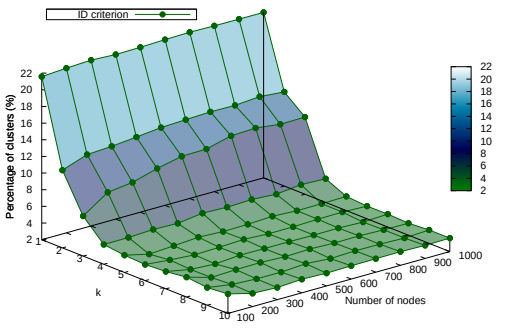
2.5.2.2 Nombre de clusters

Le nombre de clusters obtenu par simulation, en fonction du critère d'élection du cluster-head est illustré par la figure 2.13. Ces courbes représentent les pourcentages de clusters construits par rapport à la taille du réseau et fonction du degré moyen et de k.

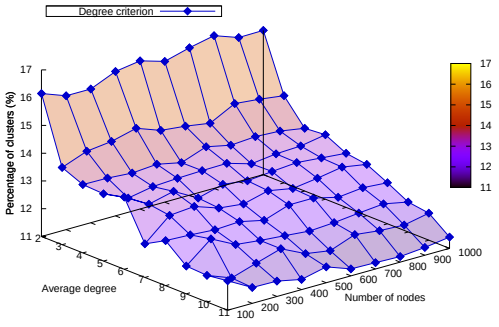
Dans les figures 2.13(a), 2.13(c) et 2.13(e), la valeur k a été fixée à 2 et le degré moyen varie entre 2 et 11. Premièrement, nous observons que pour chaque critère d'élection, le pourcentage de clusters par rapport à la taille du réseau est sensiblement le même, pour chaque valeur de degré moyen. Ensuite, pour chaque taille de réseaux fixée, le pourcentage de clusters décroit quand la valeur de degré décroit, ce qui est assez logique car le réseau tend à devenir une structure proche d'un arbre ou d'anneaux reliés. Il est également logique que si la taille des clusters augmente alors leur nombre décroit. Notons que le



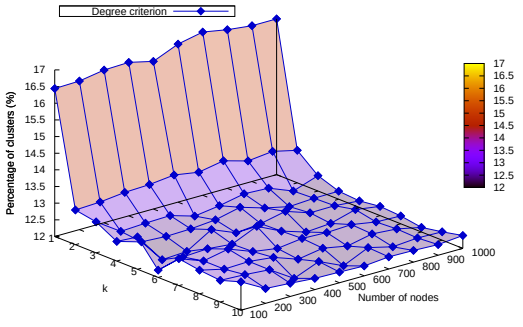
(a) Critère de l'identité en fonction du degré moyen



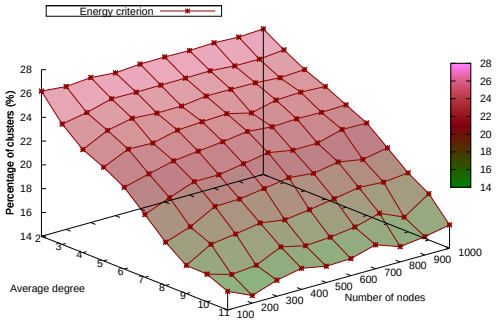
(b) Critère de l'identité en fonction du paramètre k



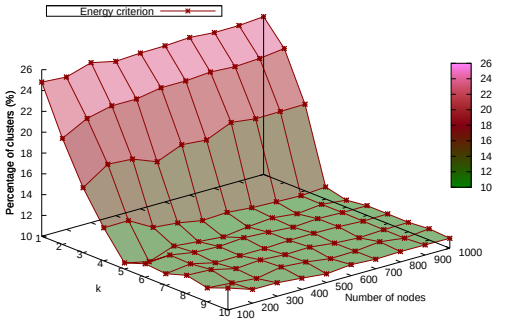
(c) Critère du degré en fonction du degré moyen



(d) Critère du degré en fonction du paramètre k



(e) Critère de l'énergie en fonction du degré moyen



(f) Critère de l'énergie en fonction du paramètre k

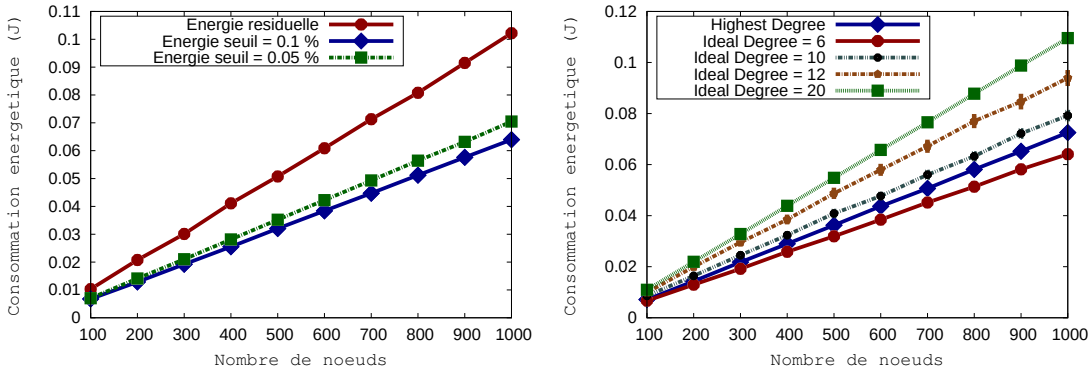
FIGURE 2.13 – Pourcentage du nombre de clusters en fonction du degré moyen et k

critère de l'identité fournit une meilleure distribution des clusters (entre 6% et 18%) comparativement au degré et à l'énergie.

Dans les figures 2.13(b), 2.13(d) et 2.13(f), nous avons fixé le degré moyen à la valeur 6 et nous avons fait varié les valeurs de k entre 1 et 10. Nous observons que pour chaque critère d'élection des cluster-head, le pourcentage de clusters construit ne varie pas significativement, quelque soit la valeur de k et que pour chaque taille de réseau, le pourcentage de clusters décroit quand les valeurs de k augmentent, ce qui est totalement conforme à l'idée intuitive.

2.5.2.3 Impact de l'énergie résiduelle par rapport à la notion de seuil limite

Nous avons également voulu étudier la durée de vie d'un nœud si nous prenions comme critère d'élection dans un premier temps, l'énergie résiduelle puis dans un second temps, la tolérance acceptée (seuil) dans la différence entre l'énergie résiduelle de 2 nœuds voisins, avant de faire une nouvelle élection de cluster-heads. Cette tolérance est fixée à 0.1% et 0.05%. La figure 2.14(a) montre que l'usage d'un seuil est préférable.



(a) Énergie résiduelle vs Énergie seuil

(b) Degré maximal vs degré idéal

FIGURE 2.14 – Énergie résiduelle vs Énergie seuil - Degré maximal vs degré idéal

2.6 Travaux en cours et perspectives

À la suite de ces travaux, je travaille actuellement sur la recherche d'un algorithme offrant, à la fois de bonnes propriétés énergétiques mais surtout une grande stabilité des clusters construits lors de modification topologique. Les pistes que j'aimerais suivre est l'exploitation des techniques utilisées à la fois dans le protocole IEEE 802.1w de construction d'arbres couvrants et dans le protocole de routage dynamique EIGRP dont le protocole est publique que depuis Février 2013 [SSN+13]. Ils utilisent tous deux des chemins de secours pré calculés afin d'anticiper les éventuelles modifications topologiques et ainsi éviter une attente de re-stabilisation de l'algorithme. Coupler ces techniques à la conservation du critère d'éligibilité, basé sur l'identité des nœuds, devraient permettre d'obtenir une solution consommant peu d'énergie et l'obtention de clusters stables lors des modifications topologiques. Pour évaluer la stabilité des clusters dans le cadre de la mobilité des nœuds, j'utilise la définition donnée dans [Raj02] qui est la suivante : "one measure of robustness of the topology is given by the maximum number of nodes that need to change their topology information as a result of a movement of a node".

Dans le domaine des réseaux de capteurs, je souhaite dans un premier temps évaluer notre solution à la nouvelle solution publié récemment par T. Ducrocq et *al.* dans [DMH13]. Il utilise comme critère d'élection la densité, définie par N. Mitton et *al.* dans [MFLT05]. Nous nous sommes déjà comparé à cette solution et nous avons pu constater par simulations que nous utilisions plus de 50% de messages en moins pour construire nos clusters à k sauts. Il est donc intéressant de continuer notre comparaison à la fois sur les consommations énergétiques mais aussi sur la stabilité des clusters.

2.7 Publications majeures

[1] Mandicou BA and Olivier FLAUZAC and Rafik MAKHLOUFI and Florent NOLOT and I. NIANG Fault-Tolerant and Energy-Efficient Generic Clustering Protocol for Heterogeneous WSNs International Journal On Advances in Networks and Services, vol. 6, num. 3&4, Dec 2013

[2] Mandicou BA and Olivier FLAUZAC and Rafik MAKHLOUFI and Florent NOLOT and I. NIANG Energy-Aware Self-Stabilizing Distributed Clustering Protocol for Ad Hoc Networks : the case of WSNs. KSII Transactions on Internet and Information Systems , Nov. 2013

[3] M. Ba, O. Flauzac, B.S Haggar, F. Nolot, and I. Niang. Self-stabilizing k-hops clustering algorithm for wireless ad hoc networks. In *Proceedings of ICUIMC '13, the 7th International Conference on Ubiquitous Information Management and Communication*, Acceptance rate : 29%, pages 38 :1–38 :10, 2013. ACM. **Best Paper Award**

[4] M. Ba, O. Flauzac, R. Makhloufi, F. Nolot, and I. Niang. Evaluation study of self-stabilizing cluster-head election criteria in wsns. In *CTRQ 2013, The Sixth International Conference on Communication Theory, Reliability, and Quality of Service*,Acceptance rate : 25%, pages 64–69, 2013. **Best Paper Award**

Chapitre 3

Auto-organisation du transport de l'information

Résumé.

À partir d'un réseau ad-hoc structuré, il faut pouvoir acheminer les informations d'une source vers une destination. De très nombreuses solutions de routage existent dans la littérature mais n'exploitent pas toujours la structure du réseau et encore moins souvent les informations déjà recueillies par le réseau. Dans ce chapitre, je présente différentes solutions d'acheminement de l'information qui se basent sur les algorithmes de construction de clusters présentés précédemment. Dans une première partie, j'aborde le thème de la construction d'arbres couvrants puis je présente dans une deuxième partie un nouvel algorithme de construction d'arbres couvrants qui tire son originalité du fait qu'il construit simultanément des clusters à 1 *saut. Pour cela, la construction de l'arbre se fait en partie grâce aux informations collectées pour la construction des clusters et nous obtenons un arbre dans chaque cluster et un arbre entre les clusters. Dans une troisième partie, je présente plusieurs approches de routage sur des réseaux structurés par des clusters à* 1 *saut et à* k *sauts. Plusieurs pistes ont été envisagées afin d'étudier les influences de certaines stratégies sur la durée de vie des nœuds en fonction de leur type. Dans le cas des clusters à* k *sauts, une nouvelle approche de routage utilisant de l'agrégation de données avec une coopération entre nœuds voisins est présentée.*

Ces travaux ont été effectués dans le cadre du co-encadrement avec Olivier Flauzac du doctorat de Bachar Salim Haggar et de Mandicou Ba. Ils ont donné lieu à 3 *conférences d'audience internationale avec comité de sélection. Ces doctorants ont respectivement participé au projet national ANR RNRT RISC et au projet régional CPER CapSec-ROFICA.*

3.1 Quelques usages des arbres couvrants

De très nombreuses solutions existent pour construire des arbres couvrants [CYH91, HC92, Tel94b, AW04, LHL03] dans le but de supprimer les boucles avant l'éventuelle diffusion d'une information. En effet, l'existence de boucles dans un réseau peut provoquer des dysfonctionnements majeurs. Si nous nous référons aux protocoles Ethernet et IPv4, pour l'acheminement d'un paquet IPv4, il est nécessaire de connaître l'adresse MAC du prochain saut pour construire la trame Ethernet. Cette connaissance est acquise grâce à des requêtes ARP envoyées en broadcast sur le réseau. Hors, dans un réseau Ethernet qui contient des boucles, les switchs ne seront pas en mesure de supprimer ces trames de broadcast. Le choix de conserver logiquement ces boucles peut également être un choix mais dans ce cas, il faut ajouter au protocole de diffusion de l'information un mécanisme de contrôle afin de détecter les duplications de messages. Ce mécanisme peut être plus coûteux, à l'usage, que la construction définitive d'un arbre couvrant. D'où l'intérêt majeur des algorithmes de spanning tree. L'exemple le plus significatif dans les réseaux informatiques de l'usage des arbres couvrants est peut-être le protocole de routage OSPF [Moy98] qui utilise l'algorithme du Shortest Path First de Dijkstra [Dij71] ou bien le protocole IEEE 802.1w fonctionnant sur tous les switchs administrables.

Lors de la structuration d'un réseau ad hoc en cluster, nous devons également être en mesure de supprimer les boucles. Il est donc possible de combiner à la fois notre solution auto-stabilisante de construction d'arbres couvrants avec des solutions auto-stabilisantes de construction d'arbres couvrants. Il existe des solutions [Her91, Var97, DH99] pour combiner 2 solutions afin d'obtenir une solution de clustering et de construction d'arbres couvrants auto-stabilisantes mais l'analyse de ces techniques nous amènent à constater que nous avons besoin d'informations similaires pour les 2 solutions. J'ai donc eu l'idée de ne pas faire de la combinaison mais plutôt de construire simultanément les clusters et l'arbre couvrant.

3.2 Solution de construction simultanée de clusters et d'arbre couvrant

Au lieu de considérer le réseau globalement pour construire un arbre couvrant sans prendre en compte les clusters construits, nous risquons d'obtenir un arbre inutilisable au sein des clusters pour pouvoir faire de la diffusion d'informations. Si nous considérons l'exemple de la figure 3.1, nous constatons que dans le cluster 10, nous n'obtenons pas un arbre couvrant. La construction globale produit donc dans le cluster une structure inutilisable.

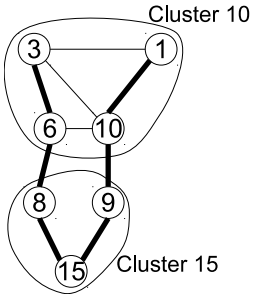


FIGURE 3.1 – Exemple d'arbre couvrant inutilisable au sein d'un cluster

D'après nos connaissances, avec O. Flauzac et B.S Haggar, nous avons été les premiers à proposer de construire à la fois des arbres couvrants dans chaque cluster et un arbre de clusters. Notre algorithme proposé, appelé $MaxCST$ et publié dans [FHN10b, FHN10a], permet de construire des clusters de diamètre au plus égal à 2 en utilisant le principe décrit dans la partie 2.3 mais également les arbres couvrants de chaque cluster et l'arbre des clusters qui se trouve être enraciné sur le cluster de plus grande

identité. Pour cela, la structure du message $hello$ est modifiée en y ajoutant 2 champs supplémentaires : les identités des clusters voisins et l'identité du cluster ayant la plus grande identité détectée.

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

La figure 3.2 représente l'arbre couvrant de clusters obtenu par l'algorithme $MaxCST$. L'arbre obtenu est bien enraciné sur le cluster de plus grande identité (cluster 15).

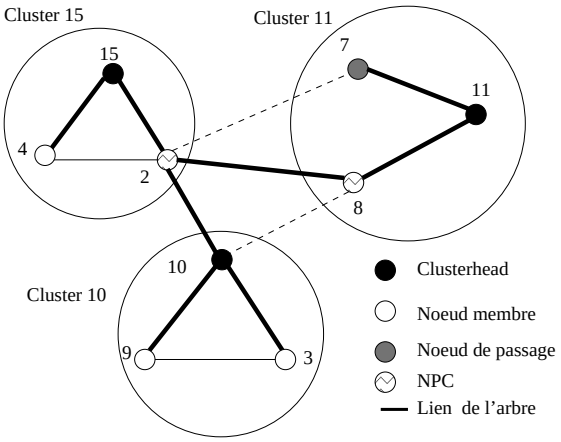


FIGURE 3.2 – Exemple de construction d'arbres

3.2.1 Quelques notations additionnelles

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

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

Un nœud u possède les données suivantes en plus de celles utilisées pour le clustering :

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

3.2.2 Sélection des Nœuds de Passage Choisis (NPC)

L'ensemble Nœud de Passage Choisis (NPC_u) est un ensemble de couples (cl, id) tels que id est le nœud de passage ayant la plus grande identité permettant de joindre le cluster $cl \neq cl_u$, et id appartient au cluster de u.
Notre algorithme de construction des NPC se déroule en deux étapes. Dans un premier temps, chaque nœud de passage collecte les identités de ses clusters voisins et transmet à son clusterhead (algorithme 3). Dans un second temps, chaque clusterhead sélectionne parmi les nœuds de passage candidats (s'il y a plusieurs nœuds de passage vers un cluster voisin), le nœud ayant la plus grande identité pour jouer ce rôle (algorithme 4).

Algorithme 3 Construction locale de NPC sur un NP

$NPC \leftarrow NPC_{cl_u} \backslash \{(cl_v, id_v) \in NPC_{cl_u} / id_v = id_u\}$
for all $v \in N_u$ **do**
  if $cl_v \neq cl_u$ **then** {u est voisin d'un autre cluster}
    update(NPC_u, $\{(cl_v, id_u)\}$)
  end if
end for

- Si u est nœud de passage. u sait s'il existe parmi ses voisins un nœud qui n'appartient pas au même cluster que lui. Donc u collecte les identités de clusters voisins (algorithme 3). Ensuite, il remonte ces informations à son clusterhead.

Algorithme 4 Construction locale de NPC sur un CH

$NPC \leftarrow \emptyset$
for all $v \in N_u$ **do**
  if $cl_v \neq cl_u$ **then** {u est le nœud de passage vers cl_v}
    update(NPC_u, $\{(cl_v, id_u)\}$)
  else {Récupération des données des NP de mon cluster}
    update(NPC_u, NPC_v)
  end if
end for

- Si u est clusterhead. u sélectionne un seul nœud de passage pour chacun de ses clusters voisins. S'il existe plusieurs nœuds de passage candidats vers un cluster voisin, u sélectionne parmi ceux-ci, celui ayant la plus grande identité comme Nœud de Passage Choisis (NPC_u) (algorithme 4). Cette étape nécessite que les informations concernant les clusters voisins soient remontées au clusterhead. Comme l'ensemble NPC est vidé au préalable, le clusterhead ne garde en mémoire que les informations les plus récentes. Les deux étapes de l'algorithme (collection des identités de clusters voisins et sélection des nœuds de passage) ne nécessitent que les informations locales. Le fait que ces étapes soient locales permet une maintenance rapide et permet également à l'algorithme de s'adapter à la mobilité des nœuds.

Algorithme 5 Fonction update(NPC_1, NPC_2)

for all $(cl_u, id_u) \in NPC_2$ **do**
  if ($\nexists (cl_v, id_v) \in NPC_1 / cl_v = cl_u$) **then** {Le cluster est inconnu dans mon NPC_1}
    $NPC_1 \leftarrow NPC_1 \cup (cl_u, id_u)$
  else {Mise-à-jour de NPC_1 avec l'identité maximale}
    if $id_v < id_u$ **then**
      $NPC_1 \leftarrow NPC_1 \backslash \{(cl_v, id_v)\}$
      $NPC_1 \leftarrow NPC_1 \cup \{(cl_u, id_u)\}$
    end if
  end if
end for

Exemple 3.2.1
La figure 3.3 montre la sélection des NPC. Dans un premier temps, chaque nœud de passage collecte les identités de ses clusters voisins et remonte ces informations à son clusterhead. Par exemple les nœuds de passage 5 et 6 envoient à leur clusterhead 13 les couples suivants : (11, 6), (16 ; 6) et (16, 5). De la même manière les nœuds de passage 2 et 7 envoient à leur clusterhead 16 les couples (11, 7), (13, 7) et (13, 2). À la réception de ces informations, chaque clusterhead sélectionne ses NPC pour joindre ses clusters voisins. Dans notre exemple, le clusterhead 13 sélectionne les couples suivants pour joindre ses clusters voisins : $NPC_{13} = \{(11, 6), (16, 6)\}$. De la même manière, le clusterhead 16 sélectionne les couples suivants : $NPC_{16} = \{(11, 7), (13, 7)\}$.

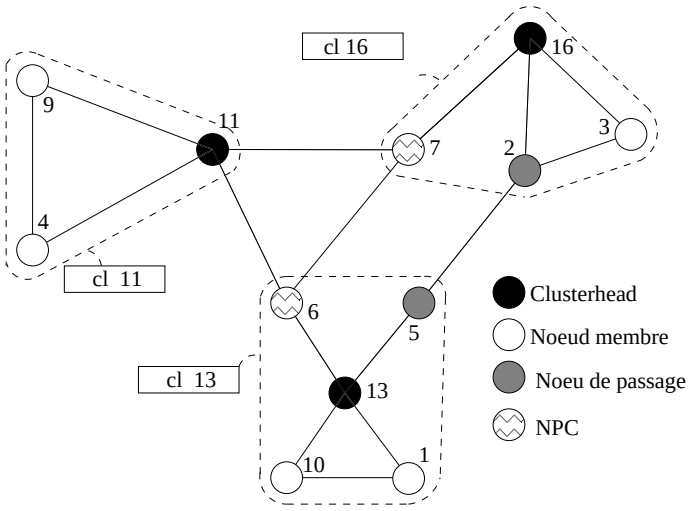


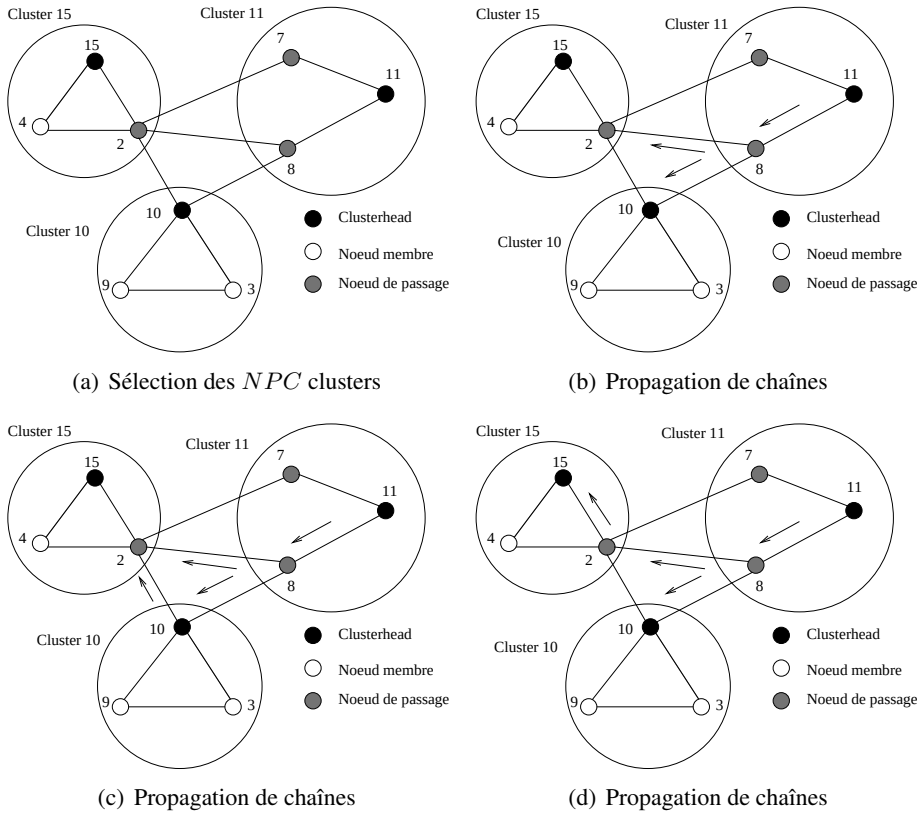
FIGURE 3.3 – Sélection des nœuds de passage choisis

3.2.3 Construction de l'arbre de clusters

La construction de l'arbre de clusters est basée sur les identités associées à chaque cluster. À la fin du processus, le cluster ayant la plus grande identité dans le réseau sera la racine de l'arbre de clusters. La construction de l'arbre de clusters se fait par échange périodique des messages $hello$. Chaque message $hello$ transmis par un nœud u contient : id_u, $statut_u$, cl_u, NPC, C_u et $F(cl_u)$. C_u est un ensemble de chaînes contenant les clusters cl d'identités maximales détectées et les listes des clusters permettant de les atteindre et $F(cl_u)$ est l'ensemble de clusters fils de cl_u. Seuls les clusterheads et les nœuds de passage maintiennent les ensembles C_u et $F(cl_u)$. La cardinalité de C_u est égale à 1 pour un clusterhead. Cependant, chaque nœud u exécute l'algorithme de façon distribuée et choisit comme père le cluster

ayant la plus grande identité qu'il a détecté (il est aussi la racine pour cl_u). Ensuite, u propage cette information ($C_u = \{(cl_u, pere(cl_u))\}$) à ses clusters voisins. À la réception du message de u par un nœud v, si la plus grande identité détectée par v auparavant est plus petite que celle annoncée par u, v choisit cl_u comme père et propage cette information ($C_v = \{(cl_v, pere(cl_v) = cl_u, pere(cl_u) = racine)\}$) à ses clusters voisins. Ce processus se répète jusqu'à ce que l'arbre soit construit et que le cluster ayant la plus grande identité du réseau soit choisi comme la racine de l'arbre. À la fin du processus, chaque cluster connaît la racine de l'arbre et la liste des clusters permettant de l'atteindre. Chaque cluster connaît également l'ensemble de ses clusters fils et de son père. Chaque cluster choisit un père parmi les clusters cl de C_u d'un saut au plus proche de lui. Un père est choisi lorsque sa distance au cluster ayant l'identité maximale détectée est plus courte. Par ailleurs, la décision du choix d'un père revient uniquement au clusterhead. Ainsi, un clusterhead n'a pas à attendre la décision des autres clusterheads et peut exécuter l'algorithme de façon distribuée. Notre solution est tolérante aux fautes, prend en compte les changements topologiques en choisissant une nouvelle racine, en mettant à jour l'arbre, en réexécutant l'algorithme sur la topologie modifiée.

3.2.4 Exemple détaillé d'exécution



(a) Sélection des NPC clusters

(b) Propagation de chaînes

(c) Propagation de chaînes

(d) Propagation de chaînes

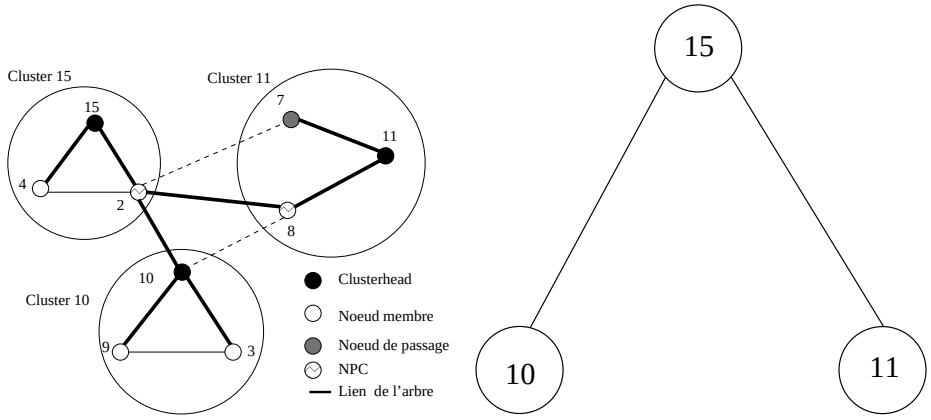
FIGURE 3.4 – Exemple de construction de l'arbre

Nous détaillons un exemple d'exécution sur un graphe de clusters. Considérons la topologie de la figure 3.4(a). Cette figure représente la construction de l'arbre suivant notre l'algorithme. Dans un premier temps, chaque clusterhead collecte les identités de ses clusters voisins par l'intermédiaire des nœuds de passage. Ensuite, chacun sélectionne les Nœuds de Passage Choisis (NPC) en exécutant les algorithmes 3 et 4. Par exemple les nœuds 7 et 8 du cluster 11 permettent d'atteindre le cluster 15. Cependant, le clusterhead 11 sélectionne le nœud 8 comme nœud de passage choisi pour joindre le cluster

15 car il possède la plus grande identité. D'où l'ensemble NPC du clusterhead 11 pour joindre ses clusters voisins sont : $NPC_{11} = \{(15, 8), (10, 8)\}$. Ceux des clusterheads 15 et 10 sont respectivement $NPC_{15} = \{(11, 2), (10, 2)\}$ et $NPC_{10} = \{(15, 10), (11, 10)\}$.

Dans un second temps, les nœuds exécutent l'algorithme de construction de l'arbre (algorithme 6). La chaîne que le nœud 8 va transmettre aux nœuds 2 et 10 est : $\{(11)\}$ et à la réception de cette chaîne, 2 remonte la chaîne $\{(11)\}$ à son clusterhead (figure 3.4(b)).

De la même manière, le clusterhead 11 va recevoir du NPC 8 l'ensemble de chaînes suivantes : $\{(10), (15)\}$. À la réception de ces chaînes, le clusterhead 11 va choisir la chaîne la plus courte vers le cluster ayant l'identité la plus grande détectée (cluster 15). Donc la chaîne $C_{11} = \{(11, 15)\}$. De la même manière, le clusterhead 15 va recevoir les chaînes suivantes : $\{(11), (10)\}$. À la réception de ces chaînes, le nœud 15 ignore ces chaînes car le nœud max contenu dans ces chaînes sont tous plus petits que celui de sa chaîne ($C_{15} = \{(15)\}$). Donc le nœud 15 s'élit comme la racine de l'arbre. La chaîne $C_{11} = \{(11, 15)\}$ et $C_{10} = \{(10, 15)\}$. Cependant le cluster 15 sera le cluster père, 10 et 11 seront ses clusters fils. La figure 3.5 représente l'arbre obtenu.



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

FIGURE 3.5 – La structure d'arbre obtenue

3.2.5 Présentation de l'algorithme $MaxCST$

À la réception d'un message $hello$, chaque clusterhead exécute l'algorithme 6. Dans un premier temps, chaque clusterhead vérifie sa cohérence. Un clusterhead u est incohérent si sa chaîne C_u est un ensemble vide ou si son père n'est pas dans son NPC, c'est à dire $pere(cl_u)$ n'est pas un cluster voisin. La règle $R1.a$ permet de corriger cette incohérence en mettant dans sa chaîne son cluster identité cl_u ($C_u = \{(cl_u)\}$) et en vidant l'ensemble de ses clusters fils ($F(cl_u)$).
Si un nœud u reçoit un message $hello$, il vérifie d'abord s'il est déjà dans la chaîne reçue. Si u n'est pas dans la chaîne reçue alors :

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

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

Algorithme 6 Réception de Ch sur u avec $statut_u = CH$

{Gestion de la cohérence}
if $(C_u = \emptyset) \vee (\forall w \in NPC_u / pere(cl_u) \neq cl_w)$ **then** {La règle R1.a}
  $C_u \leftarrow \{(cl_u)\}$ et $F_u \leftarrow \emptyset$
end if
{Traitement de la chaîne Ch}
if $u \notin Ch$ **then**
  if $pere(cl_u) = Ch[0]$ **then** {La règle R1.b}
    $Ch_u \leftarrow (cl_u, Ch)$
  else if $(Max(Ch) > Max(Ch_u)) \vee (Max(Ch) = Max(Ch_u)) \wedge (longueur(Ch) + 1) < (longueur(Ch_u))$ **then** {La règle R1.c}
    $Ch_u \leftarrow (cl_u, Ch)$ et $F_u \leftarrow \emptyset$
  else {La règle R1.d}
    $F_u \leftarrow F_u \backslash \{Ch[0]\}$
  end if
else
  if $id_u = Ch[1]$ **then** {La règle R1.e}
    $F_u \leftarrow F_u \cup Ch[0]$
  end if
end if

À la réception d'un message $hello$, chaque nœud de passage u exécute l'algorithme 7.

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

Algorithme 7 Réception de Ch_v sur u avec $statut_u = NP$

if $cl_v = cl_u$ **then**
  if $id_v = cl_u$ **then** {La règle $R2.a$}
    $C_u \leftarrow C_u \cup Ch_v$
  end if
else
  if $\exists (cl_w, id_w) \in NPC_u / (cl_w = cl_v) \wedge (id_w = id_u))$ **then** {La règle $R2.b$}
    $C_u \leftarrow C_u \cup Ch_v$
  end if
end if

3.3 Acheminement de l'information en utilisant des arbres

3.3.1 Exploitation de l'algorithme $MaxCST$

L'algorithme $MaxCST$ permet d'exploiter la structuration locale (le clustering), pour réaliser une structuration globale du réseau (l'arbre couvrant). Cette structure peut aussi être utilisée pour diffuser

de l'information dans le réseau. En effet, l'arbre couvrant apporte un avantage dans la diffusion d'informations, car il permet de réduire le nombre de messages échangés et d'atteindre la totalité des nœuds du réseau (au travers les clusters).

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

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

Pour la diffusion globale, nous nous basons sur l'arbre couvrant de clusters. Comme nous l'avons montré dans l'algorithme 4, un cluster connaît les identités des clusters à suivre pour atteindre la racine de l'arbre (*i.e.* le nœud d'identité max du réseau). En particulier, l'identité de son père dans l'arbre lui est connu (noté $pere(cl_u) = chu[1]$ si $longueur(ch) > 1$, cl_u si $longueur(ch) = 1$). Tout comme la diffusion classique dans un arbre, un cluster qui désire envoyer un message, l'envoie donc à son père et à ses fils. Lorsqu'un cluster reçoit un message :

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

connexité

Il nous reste maintenant à expliquer comment un cluster (*i.e.* l'un des nœuds du cluster) qui reçoit un message peut l'envoyer à un autre cluster. En effet, un nœud quelconque d'un cluster qui reçoit un message, doit l'envoyer aux clusters père et fils. Comme nous l'avons dit précédemment, cela passe donc par une étape de diffusion locale puis un acheminement aux clusters voisins. Nous avons ainsi deux solutions :

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

Pour réduire le nombre de messages et limiter la charge du clusterhead, nous avons choisi la deuxième solution. Notre algorithme général est décomposé en deux algorithmes (algorithmes 4 et 3), suivant si le nœud est clusterhead ou nœud de passage.

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

Si c'est un NP de son cluster, même avec l'ensemble NPC, il n'est forcément possible de déduire de quel cluster le message provient, comme le montre l'exemple suivant.

Exemple 3.3.1
La figure 3.6 montre qu'il n'est pas possible de déduire de quel cluster le message provient, ce qui peut induire une duplication du message. Pour le cluster 8, le nœud de passage choisi vers le cluster 10 est forcément le nœud 4. En effet, c'est l'identité la plus grande parmi les nœuds 2, 4. Pour le cluster 10, c'est le nœud 3 qui est choisi, car il possède l'identité la plus grande parmi les nœuds 1, 3. Si le message provient du nœud 8, il sera envoyé au nœud 4, puis au nœud 1. Si le nœud 1 transfère alors le message au nœud 10, ce dernier peut déduire que le message vient du cluster 9 et non pas du cluster 8.

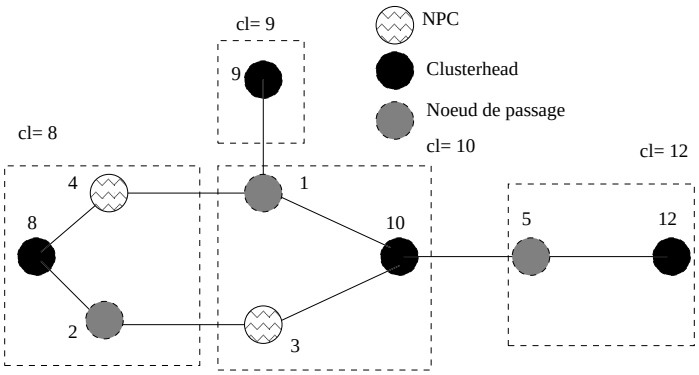


FIGURE 3.6 – Exemple d'une duplication du message

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

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

Algorithme 8 Réception d'un message (émetteur, données, IDC) sur u avec $statut_u$ = CH du nœud v

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

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

Algorithme 9 Réception d'un message (émetteur, données, IDC) sur u avec $statut_u$ = NP du nœud v

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

3.4 Introduction aux algorithmes et protocoles de routage

Il existe de très nombreuses solutions pour router une information sur un réseau et une très grande quantité de travaux de recherche sont publiés. Différentes approches et stratégies sont également exploitées et les algorithmes se retrouvent classés ou identifié comme faisant partie d'une famille. Les principales familles sont les suivantes : proactive, réactive, hybride, directionnel, vecteur distance, état de liens, vecteur chemin, ... Mais certains algorithmes peuvent faire partis de plusieurs familles et leur classement devient difficile. Surtout que certains algorithmes sont optimisés pour le réseau sur lequel ils doivent fonctionner. Dans [AWD04, WYGZ11], il est possible de trouver un résumé des algorithmes de routage sur les réseaux sans fil. Dans [LP09], les algorithmes spécifiques aux réseaux de capteurs sont regroupés. Je reprend ci-dessous la classification des caractéristiques des algorithmes de routages de B. Guizani [Gui12].

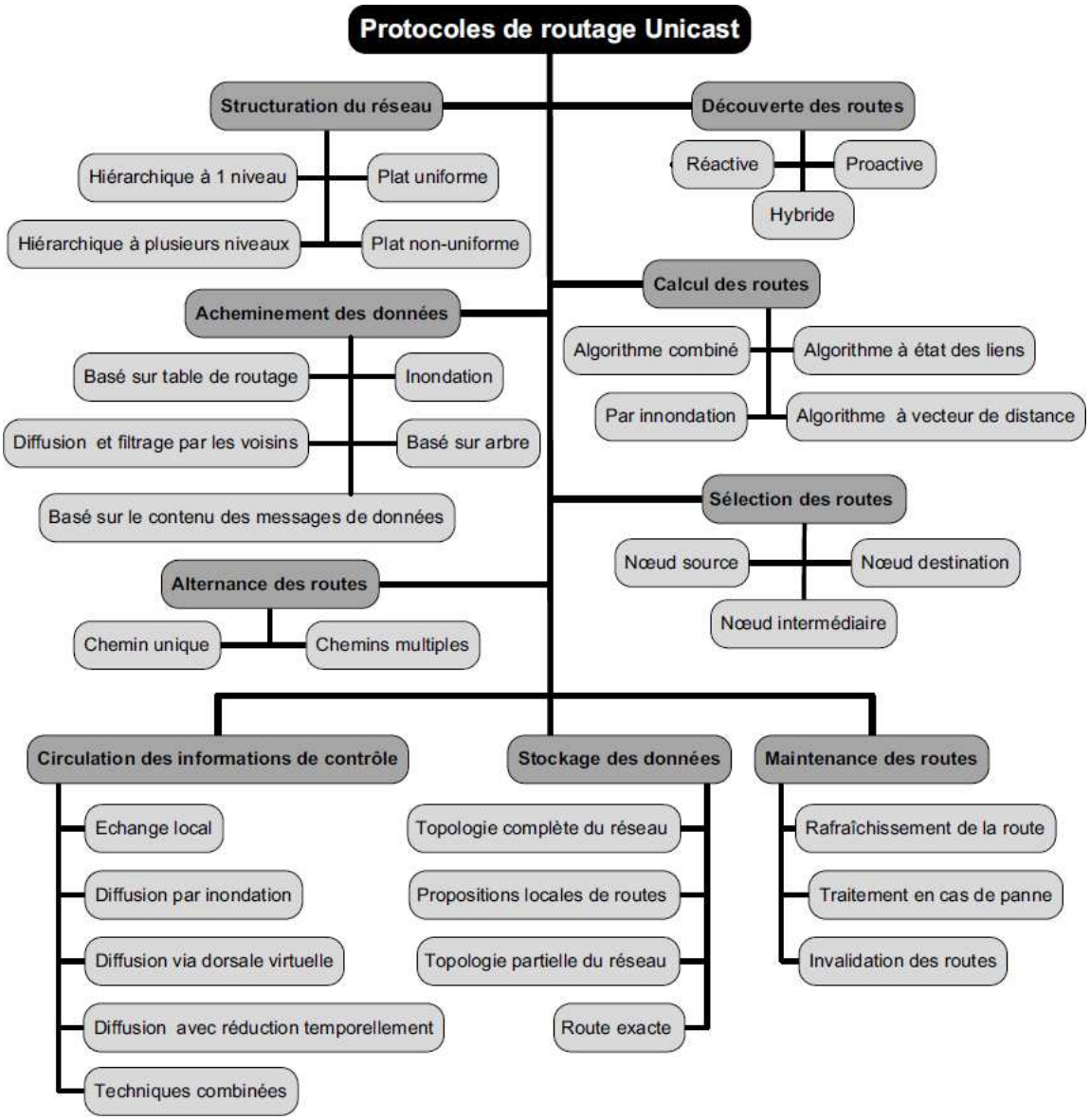


FIGURE 3.7 – Classification des caractéristiques d'un algorithme de routage de B. Guizani [Gui12]

Au cours de mes travaux, je me suis intéressé principalement aux solutions d'acheminement de

l'information sur des réseaux clusterisés à 1 saut ou à k sauts. L'objectif est, pour chaque solution, de réussir à diminuer le nombre de messages échangés et donc de se servir des messages déjà existant pour la construction des clusters. Dans la section 3.5, je présente des solutions d'acheminement de l'information qui se basent sur la solution de clustering de l'algorithme 1 de la partie 2.1, page 16. Puis dans la section suivante, je présente d'autres stratégies sur des réseaux en cluster à k sauts qui utilisent l'algorithme publié dans [BFH+13] et présenté dans la section 2.4 page 17.

3.5 Routage de l'information sur un réseau clusterisé à 1 saut

Dans le cadre du projet ANR RNRT Risc et en collaboration avec Jean Carle et la post doctorante Sylvia Romaszko de l'équipe POPS du LIFL de l'université de Lille 1, nous avons effectué une comparaison dans [RCN10] entre plusieurs solutions de routage. A partir des clusters construits à 1 saut, nous avons envisagé plusieurs scénarios de routage et la meilleure solution a été comparée par simulation à AODV [PRD03] et à Gradient [FH03]. Il s'est avéré que la solution de routage basée sur nos clusters à 1 saut n'était pas suffisamment performante dans la majorité des cas. En effet, le principe choisi pour effectuer le routage se basait sur un minimum d'informations que chaque nœud possèdait déjà, informations récoltées lors de la construction des clusters.

Durant le co-encadrement du doctorat de Bachar Salim Haggar, à la suite des résultats obtenus précédemment, j'ai donc orienté mes recherches vers des solutions utilisant au mieux les clusters et les informations que nous avions déjà récoltées lors de leurs construction. Deux solutions de routage différentes ont alors été étudiées :

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

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

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

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

3.5.1 Description générale

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

La première solution est décomposée en deux parties :

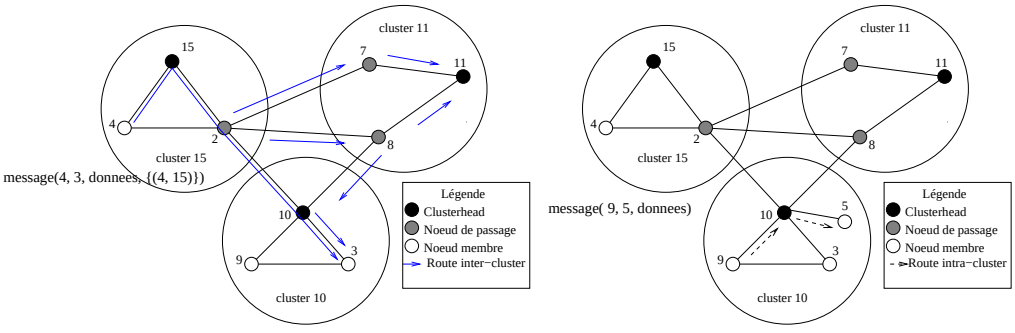
1. Réactive inter-cluster et proactive intra-cluster,
2. Hybride inter-cluster et proactive intra-cluster.

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

Remarque 3.5.1
La liste LP permet d'éviter les boucles de routage. Dans le cas de routage réactif inter-cluster, seuls les clusters id peuvent être utilisés. Cependant, pour optimiser l'entête du message, on peut seulement ajouter le cluster id dans la liste LP.

Exemple 3.5.2
Dans la figure 3.8(a), le nœud 4 souhaite envoyer un message au nœud 3. Le nœud source et le nœud destination ne sont pas dans le même cluster. Le protocole réactif inter-cluster sera utilisé pour envoyer le message au nœud 3. Donc le nœud 4 l'envoie à son clusterhead en ajoutant dans la liste LP son identité et son cluster id. Ensuite, le clusterhead ajoute à son tour son identité et son cluster id dans la liste LP et l'envoie à ses nœuds de passage. Chaque nœud de passage transmet ensuite le message aux autres clusterheads voisins et aux autres nœuds de passage voisins. Le message n'est relayé que par les nœuds de passage et les clusterheads.

Dans la figure 3.8(b), le nœud 9 souhaite envoyer un message au nœud 5. Le nœud source et le nœud destination sont dans le même cluster. Cependant, le nœud 9 et le nœud 5 ne sont pas voisins. Donc le nœud 9 l'envoie à son clusterhead 10 et celui-ci le transfère directement au nœud destinataire.



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

FIGURE 3.8 – Routage inter-cluster réactif et intra-cluster proactif

La partie hybride inter-cluster consiste à envoyer les données sans initier un processus de découverte de route mais chaque nœud qui relaie un message conserve une trace des nœuds constituant le parcours

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

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

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

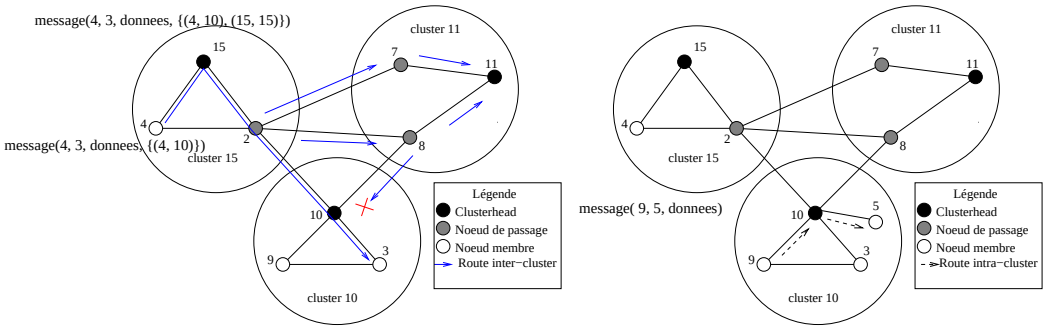
Remarque 3.5.3
Dans le cas du routage hybride inter-cluster, la liste LP contient tous les nœuds qui relaient le message ainsi que leur cluster id. Ces informations sont nécessaires pour permettre aux nœuds de construire leur table de routage.

Exemple 3.5.4
Dans la figure 3.9(a), le nœud 3 souhaite envoyer un message au nœud 3. Le nœud 3 et 4 ne sont pas dans le même cluster et le nœud 3 ne possède aucune entrée vers le nœud 4. Il l'envoie à son clusterhead en ajoutant dans l'entête du message son identité et son cluster identité. le clusterhead envoie à ses nœuds de passage en ajoutant dans l'entête du message son identité et son cluster identité. La route entre 4 et 3 passe par les clusterheads 15 et 10. Le clusterhead 15 envoie le message *(4, 3, données, {(4, 15), (15, 15)})* au nœud 2. À la réception, le nœud 2 ajoute à son tour son identité, son cluster id et l'envoie aux clusters voisins. Donc 2 envoie le message *(4, 3, données, {(4, 15), (15, 15), (2, 15)})* aux nœuds 7, 8 et 10. Ce processus se répète jusqu'à ce que le message arrive à destination. Donc le nœud 3 recevra le message suivant : *(4, 3, données, message{(4, 15), (15, 15), (2, 15), (10, 10)})*. Le message n'est relayé que par les nœuds de passage et les clusterheads.

Par exemple dans la figure 3.9(a), quand le nœud 10 relaie le message vers le nœud 3, le nœud 10 sait que la distance pour atteindre le nœud 4 est de 3 sauts, et la passerelle vers celui-ci est le nœud 2. Il sait aussi que la distance pour atteindre le nœud 15 est de 2 sauts, et la passerelle vers celui-ci est le nœud 2. Ainsi quand le nœud 3 reçoit le message, il sait que la distance pour atteindre le nœud 4 est de 4 sauts, et la passerelle vers celui-ci est le nœud 10. Il sait aussi que les distances pour atteindre les nœuds 2 et 15 sont respectivement de 2 et 3 sauts. Par exemple le nœud 3 maintient la table de routage suivante :

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

TABLE 3.1 – Table de routage du nœud 3



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

FIGURE 3.9 – Routage inter-cluster hybride et intra-cluster proactif

Dans la figure 3.9(b), le nœud 9 souhaite envoyer un message au nœud 5. Le nœud source et le nœud destination sont dans le même cluster. Cependant, le nœud 9 et le nœud 5 ne sont pas voisins. Donc le nœud 9 l'envoie à son clusterhead 10 et celui-ci le transfère directement au nœud destinataire.

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

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

Exemple 3.5.5
Dans la figure 3.10, le nœud 4 souhaite envoyer un message au nœud 9. Les nœuds 4 et 9 ne sont pas voisins et le nœud 4 ne possède aucune entrée vers le nœud 9. Le nœud 4 ajoute dans l'entête du message son identité et son cluster identité, envoie le message *(15, 9, données, {(4, 15)}*) à son clusterhead 15. Ensuite, le clusterhead 15 l'envoie à ses clusters fils. Donc le clusterhead 15 envoie le message *(15, 9, données, {(4, 15), (15, 15)}*) au nœud 2. À la réception, le nœud 2 envoie le message *(15, 9, données, {(4, 15), (15, 15), (2, 15)}*) aux nœuds 8 et 10. Quand le nœud 9 reçoit le message, il sait que la distance pour atteindre le 4 est de 4 sauts et la passerelle pour atteindre celui-ci est le nœud 10. Le nœud 11 sait aussi que la distance pour atteindre le nœud 4 est de 4 sauts et la passerelle pour atteindre celui-ci est le nœud 8.

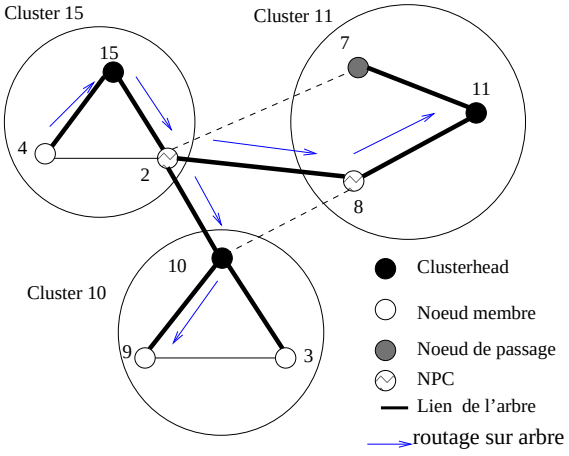


FIGURE 3.10 – Routage sur un arbre couvrant

3.5.2 L'algorithme

Dans cette section, je présente la solution de routage basée sur la structure en clusters (algorithmes 10 et 11).

Pour fonctionner, l'algorithme de routage utilise les données suivantes :

1. Données sur les nœuds.
   - LC : table des membres du cluster. Elle permet à un nœud de maintenir une connaissance proactive au sein de son cluster.
2. Données des messages.
   - LP : liste des nœuds où le message est passé ((id_A, cl_A), (id_B, cl_B), ...). Cette liste permet aux nœuds de construire leur table de routage. Elle permet également d'éviter les boucles de routage,
   - $destination$: destination du message,
   - $source$: source du message,
   - structure de la table de routage : $TR(id_e, distance_e, suivant_e)$, id_e est l'identité du nœud e, $distance_e$ correspond aux nombres de sauts nécessaires pour atteindre e, $suivant_e$ est le nœud de passage pour atteindre e depuis le nœud u,
   - structure d'un message à envoyer : $envoi(source, destination, données, LP)$.

3.5.2.1 Routage proactive intra-cluster

Nous utilisons un protocole proactif à l'intérieur des clusters. Grâce à notre structure hiérarchique, nous pouvons introduire, sans créer de surcoût important un protocole proactif au sein des clusters. Afin de maintenir une connaissance locale dans les clusters, chaque nœud du réseau maintient une table LC qui contient les membres du cluster. Chaque nœud construit sa table en se basant sur les informations reçues à partir de son clusterhead et par échange périodique de messages $hello$ utilisé dans l'algorithme de clustering. En plus de ces messages $hello$, le clusterhead envoie périodiquement sa table aux nœuds membres de son cluster. Contrairement à l'algorithme CGSR [LCWG97], où chaque nœud diffuse périodiquement sa table des membres du cluster, dans notre solution, les nœuds membres et les nœuds de passage ne diffusent pas leur table. Seul le clusterhead envoie par unicast ou multicast sa table à ses membres. En plus de cette table, chaque nœud maintient un ensemble N (voisinage).

Exemple 3.5.6
Par exemple dans la figure 3.11, le nœud 7 maintient la table des membres du cluster ci-dessous. Ainsi, avec ces deux tables, chaque nœud est en mesure de savoir si la destination est dans son voisinage ou dans son cluster. Si par exemple un nœud u souhaite envoyer un message à un nœud v qui est dans le même cluster que lui mais pas dans son voisinage, alors u l'envoie à son clusterhead.

Dans la figure 3.11, le nœud 7 souhaite envoyer un message au nœud 8. Les deux nœuds ne sont pas voisins mais ils sont dans le même cluster. Le nœud 7 l'envoie à son clusterhead et celui-ci le transfert directement au destinataire.

id	8	11
cl	11	11
Statut	Nœud de passage	Clusterhead

TABLE 3.2 – Table des membres du nœud 9

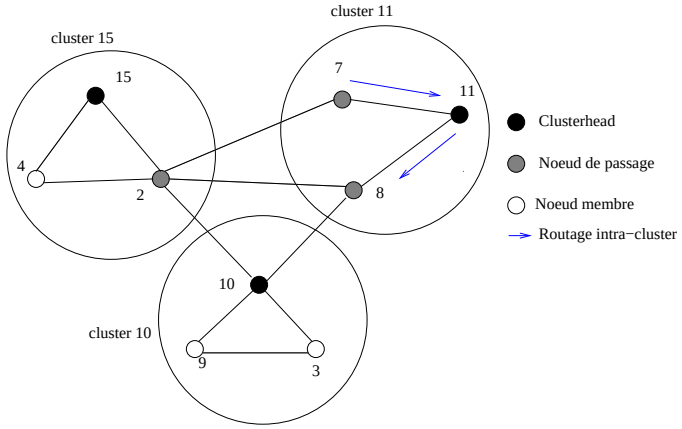


FIGURE 3.11 – Routage intra-cluster

3.5.2.2 Routage inter-cluster sans mémoire

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

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

– Si u est un clusterhead, alors il l'envoie à tous les nœuds de passage voisins,

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

Exemple 3.5.7
Dans la figure 3.12, le nœud 9 souhaite envoyer un message au nœud 4 mais le nœud destination 4 n'est pas dans le même cluster que lui. Donc le nœud 9 envoie le message contenant l'adresse de destination à son clusterhead 10 et celui-ci l'envoie aux nœuds de passage voisins, nœuds 2 et 8. Ainsi le nœud 2 l'envoie directement au nœud destinataire 4. La figure montre que le nœud 2 transfert trois fois le même message. Cela est dû du fait que les nœuds ne gardent aucune donnée en mémoire.

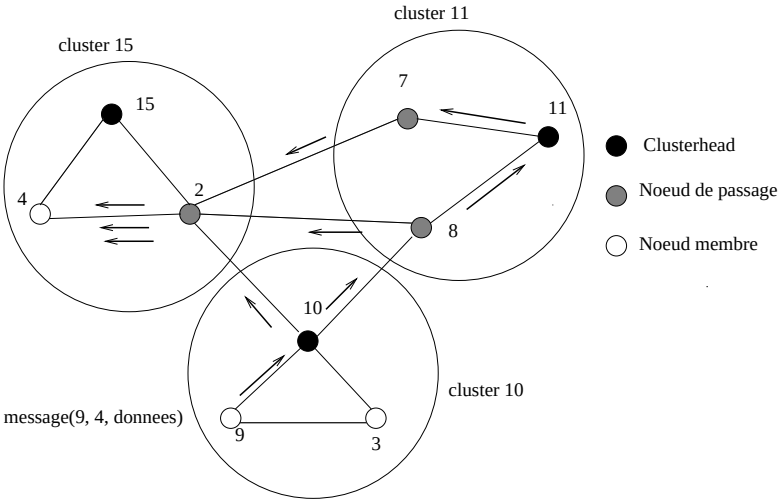


FIGURE 3.12 – Routage inter-cluster

3.5.2.3 Routage hybride inter-cluster avec mémoire

Dans cette partie nous ne décrivons pas la partie proactive intra-cluster. Cette partie est la même que celle décrite dans la section 3.5.2.1. La partie hybride inter-cluster consiste à envoyer les données sans initier un processus de découverte de route mais les nœuds construisent leur table de routage parallèlement à la transmission de messages. Notre solution combine dans une même phase la découverte des routes, la transmission de données et la construction des tables de routage. Les nœuds construisent leurs tables de routage au fur et à mesure qu'ils transmettent les messages. Sur réception d'un message, chaque nœud intermédiaire met à jour sa table de routage. Chaque entrée de cette table est associée à un nœud extérieur au cluster déjà rencontré, elle contient l'identité du nœud, la distance pour atteindre ce nœud et la passerelle vers ce nœud. Ainsi, chaque nœud qui relaie un message conserve une trace des nœuds constituant le parcours du message. Ces informations sont obtenues grâce à la liste LP contenue dans l'entête du message. Chaque nœud qui relaie un message ajoute à la liste LP son identité et son cluster identité. Cette liste contient le parcours complet du message. Il faut aussi préciser que dans le routage inter-cluster, on utilise deux modes de routage différents : routage proactif et réactif. La partie réactive concerne le routage inter-cluster. Grâce à notre structure hiérarchique, la diffusion est optimisée en limitant le nombre de nœuds chargés de relayer les paquets. On utilise un protocole proactif si un nœud souhaite envoyer un message et s'il possède une entrée dans sa table de routage vers cette destination. Comme mentionné précédemment, plus un nœud reçoit de messages, plus sa table sera complète.
Quand un nœud souhaite envoyer un message à un nœud destination et que ce dernier ne figure pas dans sa table de routage, il l'envoie en ajoutant son identité et son cluster id à ses clusters voisins. Ce

processus se répète jusqu'à ce que le message arrive à destination. Chaque nœud qui relaie ce message met alors à jour sa table de routage.

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

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

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

- Si un message qui a emprunté un chemin plus cours arrive par la suite, il le transfert aussi,
- Dans le cas contraire, il l'ignore.

Exemple 3.5.8
Dans la figure 3.13, le nœud 9 souhaite envoyer un message au nœud 4 mais il ne possède aucune entrée dans sa table de routage vers cette destination. Donc le nœud 9 envoie le message en ajoutant on identité et son cluster id à son clusterhead 10. Donc le nœud 9 envoie le message *(9, 4, données, {(9, 10)}*) au nœud 10. Le nœud 10 envoie à son tour *(9, 4, données, {(9, 10), (10, 10)}*) aux nœuds de passage voisins, nœuds 2 et 8. Ainsi le nœud 2 le transfert directement au nœud destinataire 4 et ajoute une entrée dans sa table de routage vers le nœud 9. A la réception du message, le nœud 4 ajoute également une entrée dans sa table de routage vers les nœuds 9 et 10. C'est ainsi que les nœuds construisent leur table de routage. Dans cet exemple, on considère que le nœud 2 a reçu le message du nœud 10 avant ceux des nœuds 7 et 8, donc il ignore les messages provenant des nœuds 7 et 8 car ces messages ont emprunté des chemins plus longs. Même si le nœud 2 ne transfert pas ces messages, il met à jour sa table de routage. Par contre si un message qui a emprunté un chemin plus long est arrivé en premier, il peut y avoir une duplication du message. Après transmission d'un message, si un message qui a emprunté un chemin plus court arrive par la suite, il le transfère aussi. Par exemple dans la figure 3.12, si on suppose que le nœud 2 a reçu le message provenant du nœud 8 avant celui du nœud 10, alors le nœud 2 transfert deux fois le même message mais dans ce cas, cela permet d'optimiser le contenu des tables de routage. Donc il peut y arriver qu'un nœud transfert deux fois le même message. Après transmission du message, tous les nœuds qui ont reçu le message complètent leur table de routage. Par exemple dans la figure 3.12, le nœud 2 maintient la table de routage suivante (table 3.4). A la réception du message par le nœud 4, la liste LP contenu dans l'entête du message est : $\{(9, 10), (10, 10), (2, 15)\}$.
Par exemple le nœud 2 maintient la table de routage suivante :

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

TABLE 3.3 – Table de routage du nœud 2

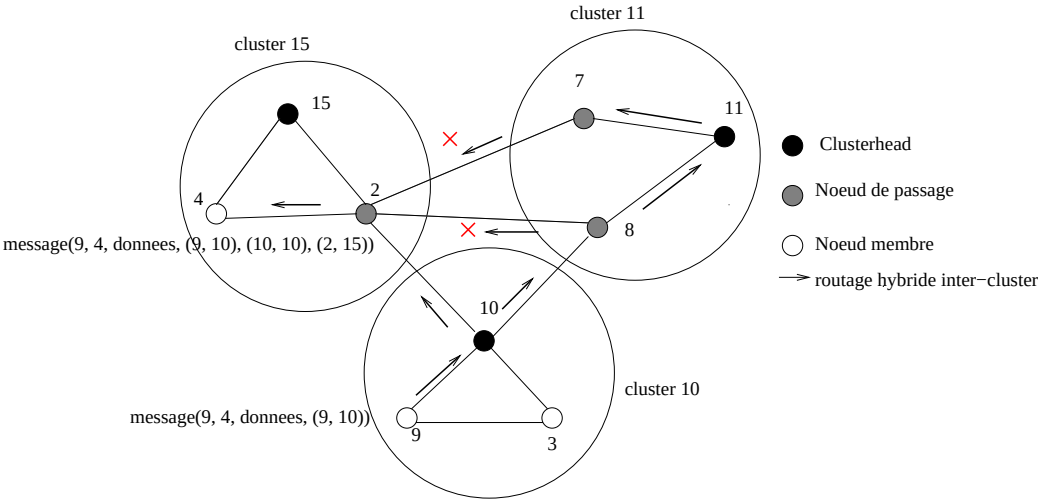


FIGURE 3.13 – Routage hybride inter-cluster

3.5.2.4 Routage hybride sur arbre

Dans cette partie nous ne décrivons pas la partie proactive intra-cluster. Cette partie est la même que celle décrite dans la section 3.5.2.1.
De la même manière que le routage hybride inter-cluster avec mémoire, les nœuds envoient les données sans initier un processus de découverte et ils construisent leur table de routage parallèlement à la transmission des données. Sur réception du message, chaque nœud met à jour sa table de routage.
Quand un nœud souhaite envoyer un message à un nœud destination et que ce dernier n'est pas dans sa table de routage, il l'envoie alors à son cluster père et à ses clusters fils en ajoutant son identité et son cluster id dans la liste LP. Chaque cluster connaît l'identité de son père dans l'arbre et celles de ses fils. Lorsqu'un cluster reçoit un message :

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

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

- Si u est un clusterhead et s'il possède des voisins dans l'arbre, il l'envoie en ajoutant son identité et son cluster id à tous ses voisins dans l'arbre,
- Si u est un nœud membre, alors il l'envoie à son clusterhead et celui-ci l'envoie à tous ses voisins dans l'arbre,
- Si u est un nœud de passage, alors il l'envoie à son clusterhead et celui-ci l'envoie à tous ses voisins dans l'arbre,
- Si u est un Nœud de Passage Choisi (NPC), alors il l'envoie à son clusterhead et à tous ses voisins dans l'arbre.

Exemple 3.5.9
Dans la figure 3.14, le nœud 9 souhaite envoyer un message au nœud 4 mais il ne possède aucune entrée dans sa table de routage vers le nœud 4. Donc le nœud 9 envoie le message (9, 4, données, {(9, 10)}) à son clusterhead 10. Le cluster 10 envoie le message (9, 4, données, {(9, 10), (10, 10)}) à son cluster père 15. Ainsi, le nœud 2 transfert directement au destinataire 4 et ajoute une entrée dans sa table de routage vers le nœud 9. À la réception, le nœud 4 ajoute également une entrée vers le nœud 10 et le nœud 9.
Par exemple le nœud 4 maintient la table de routage suivante :

id	distance	suivant
10	2	2
9	3	2

TABLE 3.4 – Table de routage du nœud 2

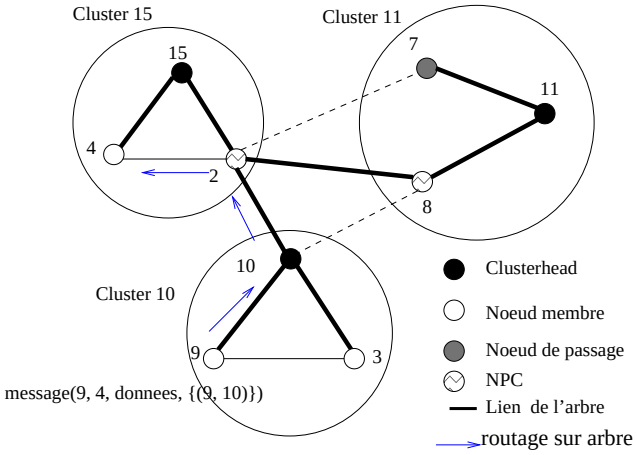
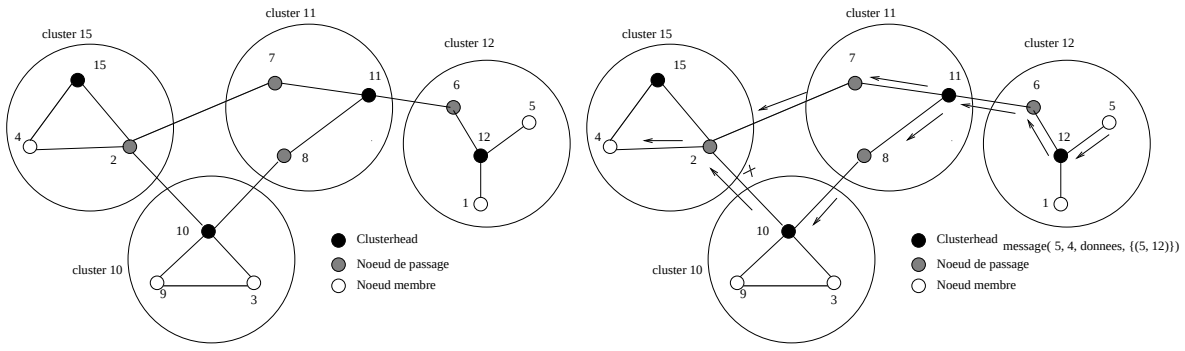


FIGURE 3.14 – Routage sur un arbre couvrant

3.5.2.5 Exemple détaillé d'exécution

Nous détaillons ici un exemple d'exécution de notre algorithme avec mémoire sur un graphe de 12 nœuds. Cet exemple décrit le routage inter-cluster avec mémoire selon le statut du nœud source (clusterhead, nœud membre ou nœud de passage). Dans la figure 3.15, le nœud source (avec $statut(Source) = NM$) souhaite envoyer un message au nœud destinataire. De plus, le nœud ne connaît aucune route vers la destination. Alors le nœud source l'envoie à son clusterhead et le clusterhead l'envoie aux clusters voisins. A la réception du message, si le destinataire n'est pas dans la table de routage alors le message sera envoyé aux clusters voisins. Ainsi, ce processus se répète jusqu'à ce que le message arrive au destinataire. Dans cet exemple, on suppose que le nœud 2 a reçu le message du nœud 7 avant celui du nœud 10. À la réception du message par le nœud 4, la liste LP contenue dans l'entête du message est : $\{(5, 12), (12, 12), (6, 12), (11, 11), (7, 11), (2, 15)\}$.

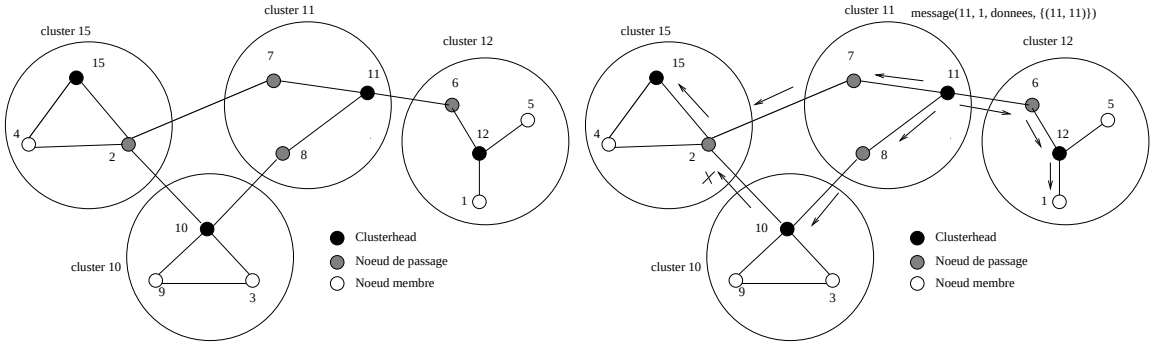


(a) Topologie (b) Routage de données

FIGURE 3.15 – Le nœud source est un nœud membre

Dans le figure 3.16, le nœud source (avec $statut(source) = CH$) souhaite envoyer un message au

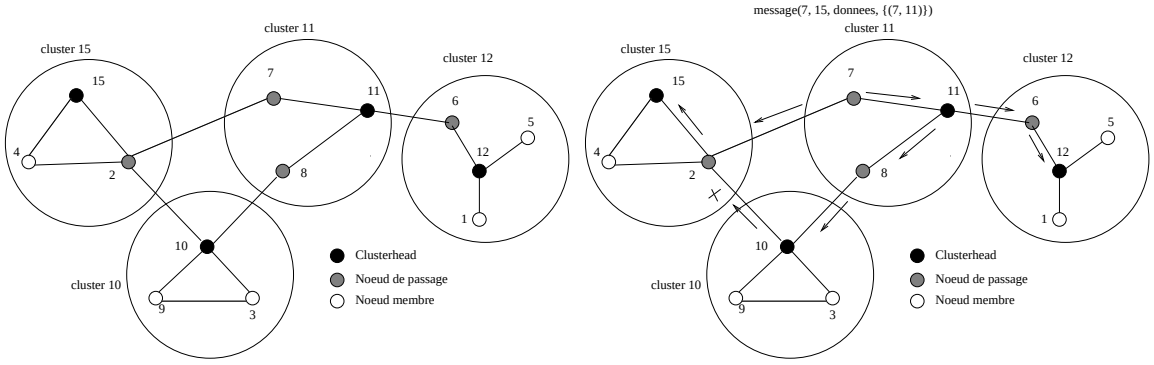
nœud destinataire. De plus, la source ne connait aucune route vers la destination. Alors le nœud source l'envoie aux clusters voisins. À la réception du message, si le destinataire n'est pas dans la table de routage alors le message sera envoyé aux clusters voisins. Ainsi, ce processus se répète jusqu'à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



(a) Topologie (b) Routage de données

FIGURE 3.16 – Le nœud source est un clusterhead

Dans la figure 3.17, le nœud source (avec $statut(source) = NP$) souhaite envoyer un message au nœud destinataire. De plus, la source ne connait aucune route vers la destination. Alors le nœud source l'envoie aux clusters voisins et à son clusterhead. A la réception du message, si le destinataire n'est pas dans la table de routage alors le message sera envoyé aux clusters voisins. Ainsi, ce processus se répète jusqu'à ce que le message arrive au destinataire. À la réception du message par le nœud 15, la liste LP contenu dans l'entête du message est : $\{(7, 11), (2, 15)\}$.



(a) Topologie (b) Routage de données

FIGURE 3.17 – Le nœud source est un nœud de passage

3.5.3 Présentation de l'algorithme

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

contient l'identité du nœud, la distance pour atteindre ce nœud et la passerelle vers ce nœud.

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

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

Algorithme 10 Envoi d'un message (source, destination, données, LP) depuis le nœud u

```
{(R1) Traitement local au cluster}
if destination ∈ N_u then {Le destinataire est dans le voisinage de u (R1.a)}
   Ajouter (id_u, cl_u) à LP
   Transfert du message à destination
else if destination ∈ LC_u then {Le destinataire est dans le cluster de u (R1.b)}
   Ajouter (id_u, cl_u) à LP
   Transfert du message au clusterhead cl_u
else {(R2) Routage du message}
   LE ← ∅
   for all w ∈ N_u do
      if (statut_w = NP) ∨ (statut_w = CH) then {Envoi à tous les NP et CH}
         if (cl_w = cl_u) then
            if (statut_u = CH) ∨ (statut_w = CH) then
               Ajouter (id_u, cl_u) à LP
               Transfert du message à w
            end if
         else if cl_w ∉ LE then {Envoi uniquement aux clusters non visités}
            LE ← LE ∪ cl_w
            Ajouter (id_u, cl_u) à LP
            Transfert du message à w
         end if
      end if
   end for
end if
```

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

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

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

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

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

Algorithme 11 Réception d'un message (destination, données, LP) du nœud v sur le nœud u

{(R1) Traitement local au cluster}
if $destination = u$ **then** {Le message est destiné à u}
  {Traitement du message}
else if $destination \in N_u$ **then** {Le destinataire est dans le voisinage de u (R1.a)}
  Ajouter (id_u, cl_u) à LP
  Transfert du message à destination
else if $destination \in LC_u$ **then** {Le destinataire est dans le cluster de u (R1.b)}
  Ajouter (id_u, cl_u) à LP
  Transfert du message au clusterhead cl_u
else {(R2) Routage du message}
  $LE \leftarrow \emptyset$
  for all $w \in N_u$ **do**
    if $(w \neq v) \wedge ((statut_w = NP) \vee (statut_w = CH))$ **then** {Envoi à tous les NP et CH, sauf v}
      if $(cl_w = cl_u)$ **then**
        if $(statut_u = CH) \vee (statut_w = CH)$ **then**
          Ajouter (id_u, cl_u) à LP
          Transfert du message à w
        end if
      else if $(\forall x \in LP / cl_x \neq cl_w) \wedge (cl_w \notin LE)$ **then** {Envoi uniquement aux clusters non visités}
        $LE \leftarrow LE \cup cl_w$
        Ajouter (id_u, cl_u) à LP
        Transfert du message à w
      end if
    end if
  end for
end if

Sur réception d'un message, l'algorithme 12 permet à chaque nœud intermédiaire de construire sa table de routage. Cet algorithme se place au niveau de la règle ($R2$) de l'algorithme 11. Chaque entrée de la table est associée à un nœud extérieur du cluster déjà rencontré. Elle contient l'identité du nœud, la distance pour atteindre le nœud et la passerelle vers ce nœud. La passerelle correspond au nœud émetteur du message. Chaque nœud qui relaie un message ajoute son identité et son cluster identité à la liste LP. Cette liste contient le parcours complet du message. Quand un nœud reçoit un message, il vérifie d'abord s'il possède une entrée vers chaque nœud de LP. S'il ne possède aucune entrée vers un nœud de LP, alors il ajoute dans sa table de routage une entrée correspondant à ce nœud. Par contre, s'il possède une entrée dans sa table de routage, il vérifie si la nouvelle entrée est meilleure que l'ancienne entrée. Si c'est le cas, il met à jour sa table avec la nouvelle entrée.

Algorithme 12 Réception d'un message (destination, données, LP) sur un nœud u

{Mise à jour de la table de routage}
$transfert \leftarrow vrai$
for i allant de 0 à $longueur(LP)$-1 **do**
  if $LP[i]_{id} \notin TR_u$ **then**
    $TR_u \leftarrow TR_u \cup \{(LP[i]_{id}, longueur(LP)\text{-}i, emetteur)\}$
  else
    $e \leftarrow \{(id_e, distance_e, suivant_e) \in TR_u / id_e = LP[i]_{id}\}$
    if $distance_e > longueur(LP)\text{-}i$ **then**
      $TR_u \leftarrow TR_u \backslash \{e\}$
      $TR_u \leftarrow TR_u \cup \{(id_e, longueur(LP)\text{-}i), emetteur, \}$
    else if $distance_e < longueur(LP)\text{-}i$ **then**
      $transfert \leftarrow faux$
    end if
  end if
end for
{Transfert du message}
if $transfert = vrai$ **then**
  {Suite algorithme 11}
end if

L'algorithme 13 permet de rechercher une entrée correspondante à une destination dans la table de routage. Cet algorithme se place au niveau de la règle ($R2$) d'algorithmes 10 et 11.

Algorithme 13 Recherche $destination$ dans la table de routage de u

if $\exists((id_e, distance_e, suivant_e) \in TR_u / id_e = destination)$ **then**
  Ajouter (id_u, cl_u)
  Transfert du message à $suivant_e$
else
  {Suite algorithme 10 ou 11}
end if

3.6 Le routage sur les réseaux en cluster à k saut

En partant du fait que la communication est de loin la première source de consommation d'énergie, dans le cadre du projet CPER CapSec ROFICA que je gère et de la collaboration avec l'équipe ERA du laboratoire ICD de l'UTT, ma réflexion m'a mené à envisager de ne plus faire de la transmission systématique des informations mais plutôt de façon collaborative. Les résultats obtenus par l'équipe ERA et publiés dans [SRAMBG10] ont donc été exploités pour concevoir une nouvelle approche du routage sur un réseau structuré en clusters à k sauts.

3.6.1 Routage avec agrégation de données

3.6.1.1 Scénario d'agrégation

Un agent implémenté sur chaque nœud permet d'utiliser les informations de routage pour établir une vue située locale et maintient ainsi une base de connaissance. Chaque agent pourra décider selon une stratégie donnée et d'une manière complètement décentralisée s'il veut coopérer ou non avec un autre agent.

Quand un capteur collecte une information sur son environnement, son agent décide si celle-ci est importante. Si c'est le cas, il initie l'agrégation en envoyant une demande de coopération à ses voisins directs afin qu'ils puissent participer s'ils le souhaitent à la session d'agrégation en cours. Les agents des nœuds voisins prennent la décision de coopérer ou non selon une stratégie donnée. Si un agent décide de coopérer, alors il envoie au demandeur les informations collectées par son nœud.

Si l'agent qui reçoit la demande de coopération est utilisé comme passerelle par le demandeur, alors il envoie directement une demande de coopération à ses voisins directs en attendant de recevoir l'agrégat calculé par l'agent demandeur. Lorsque le nœud demandeur reçoit les informations de ses voisins directs, il les concatène et élimine les redondances avant d'envoyer le résultat (i.e. l'agrégat) à sa passerelle. Cette dernière fusionne les informations des voisins et l'agrégat reçu, puis envoie le résultat à sa passerelle. Cette procédure se répète jusqu'à ce que les données agrégées atteignent la station de base.

3.6.1.2 Coopération entre agents

Nous avons défini une stratégie de coopération en prenant en compte plusieurs paramètres dans le processus de prise de décision, tels que le niveau d'importance des informations collectées, le niveau d'énergie résiduelle des capteurs, leur position dans le réseau et la densité de celui-ci. Ainsi, en fonction de la valeur de ces paramètres, chaque agent calcule un coefficient R qui détermine la pertinence de la coopération qui lui permet de décider s'il coopère ou non à l'opération de routage. Celui-ci est calculé selon l'équation 3.1 [SRAMBG10]. Par exemple, lorsqu'un nœud ne dispose que d'une petite quantité d'énergie, il peut refuser de participer au routage de certaines informations ou de toutes les informations lorsque son niveau de batterie est critique. Ainsi, il économise son énergie pour ne l'utiliser que dans la capture et la transmission de ses propres informations. Les paramètres de coopération utilisés sont décrits comme suit et sont issus de [SRAMBG10] :

- Énergie résiduelle (E) : Ce paramètre clé permet aux agents de maximiser la durée de vie de leur réseau en ne participant au routage que si la valeur de ce paramètre est assez élevée. Nous considérons le rapport entre l'énergie restante E_r^t à un instant t et l'énergie maximale E_{max} du capteur au moment de son déploiement, comme illustré dans cette équation : $E = \frac{E_r^t}{E_{max}}$;
- Degré (D) : Il permet en outre d'identifier les zones denses dans le réseau et celles où il y a peu de capteurs. Lorsqu'un nœud dispose d'un nombre de voisins très élevé, il consomme son énergie très rapidement et il a plus de chance de trouver un voisin avec qui coopérer qu'un autre nœud ayant un faible degré. Ses voisins évitent alors de le solliciter souvent avec des demandes de coopération ;
- Position (P) : Nous définissons trois positions possibles pour un nœud dans le réseau : normale, bordure ou critique. La position d'un nœud est considérée critique si celui ci relie deux parties du réseau. En outre, il représente une passerelle dans notre architecture en clusters. Un nœud est dit de bordure s'il est à l'extrémité du réseau et n'as donc qu'un ou deux voisins. Tout autre nœud occupe une position "normale". Celui-ci est entouré par plusieurs voisins et ne représente pas une passerelle ;
- Importance des informations (I) : Ce paramètre dépend surtout du type d'application cible. Une information est jugée importante dans le cas par exemple où l'écart entre la nouvelle valeur perçue et l'ancienne est assez éloigné et dépasse un certain seuil.

En fonction de la valeur de chacun de ces paramètres et de son poids, nous calculons le coefficient de coopération selon l'équation 3.1.

$$R = E * W_e + \frac{1}{d} * W_d + P * W_p + I * W_i \tag{3.1}$$

La figure 3.18 illustre l'exemple d'un nœud source (35) du cluster 40 qui détecte un événement important, demande à ses voisins directs 17, 23 et 37 de coopérer, agrège les données reçues et les siennes puis envoie l'agrégat qui en résulte à son nœud de passage (17). Dans cet exemple, le nœud

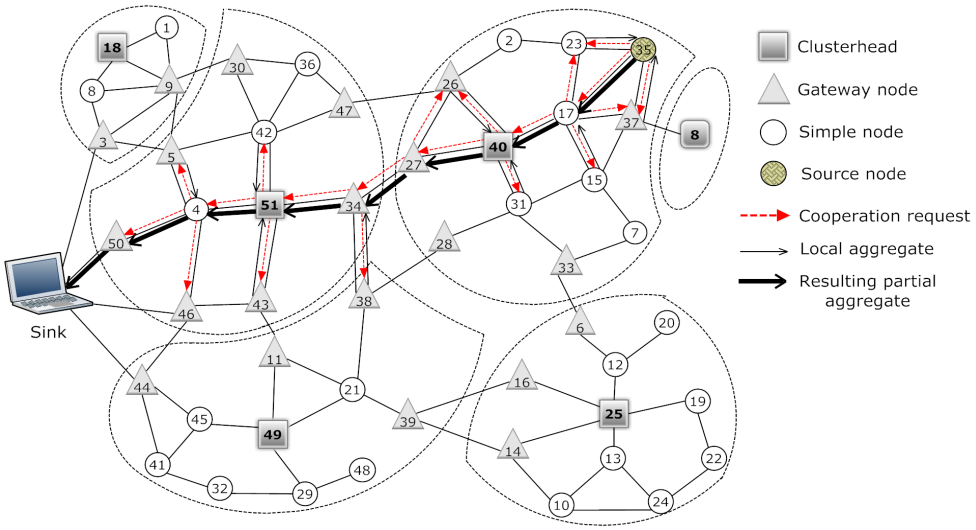


FIGURE 3.18 – Routage et agrégation de données

46 du cluster 51 décide de ne pas coopérer étant donnée sa position critique (voisin direct du puits et passerelle) et ses réserves d'énergie qui s'épuisent vite à cause de son emplacement.

3.6.2 Routage sans agrégation de données

3.6.2.1 Fonctionnement général

Dans le cadre de l'encadrement du post doctorat de Rafik Makloufi et du projet CPER CapSec ROFICA, nous avons proposé une approche de routage simple se fondant en partie sur un autre protocole de routage standard nommé DSR (*Dynamic Source Routing*). L'approche que nous avons mis en oeuvre fonctionne en deux phases : (1) une phase de *découverte des routes* durant laquelle les nœuds capteurs obtiennent des informations sur leur voisinage ainsi que leurs nœuds de passage (i.e. le premier voisin direct d'un nœud sur son chemin vers la station de base) et (2) une phase de *mise à jour des routes* qui permet aux nœuds de maintenir à jour les informations de routage acquises lors de la phase de découverte. A l'issue de la première phase, chaque nœud connaîtra ses voisins directs, son nœud de passage vers son CH et son nœud de passage qu'il peu contacter pour envoyer ses informations en direction du puits.

Ainsi, un nœud qui capture une information importante la transmet à son nœud de passage en direction de son clusterhead. Ce dernier envoie l'information à son nœud de passage en direction de la passerelle qui permet d'atteindre le puits et le processus continue jusqu'à ce que l'information atteigne le puits. Pour cela, chaque nœud sera doté d'un agent autonome qui le contrôle et définit sont comportement. Cet agent est implémenté sur la couche application du capteur. Il interviendra notamment dans le processus d'agrégation pour traiter localement les informations collectées par les capteurs et celles reçues de ses agents voisins, ainsi que pour la prise de décision concernant la coopération avec d'autres agents.

Chaque agent utilise les informations de routage pour établir une vue située [BKH+08, MDBG13] à un saut. Les connaissances d'un nœud sont ainsi limitées à seulement ses voisins directs à un saut. Cela permet de réduire le volume d'information maintenu par chaque nœud comparé au cas où un nœud a une connaissance globale de tout son réseau.

Direct Neighbors

Routing Table of node 41				
NODE_id	CLUSTER_id	STATUS	DIST_CH	DIST_Sink
41	49	SN	2	2
32	49	SN	2	3
44	49	GN	2	**1**
45	49	SN	**1**	2

TABLE 3.5 – Exemple d'une table de routage maintenue par le nœud 41

3.6.2.2 Maintenance du voisinage

La maintenance des routes et du voisinage se fait selon les deux phases :

Découverte des routes

Dans le but de découvrir les chemins qui mènent vers le puits tout en limitant les échanges de messages, une fois l'état stable est atteint les clusterheads exécutent une phase de découverte des routes. Les autres nœuds n'ont initialement besoin de connaître que le nœud de passage qui leur permet d'atteindre le CH du cluster auxquels ils appartiennent. Pour cela, ils réutilisent les mêmes informations acquises lors de la phase de clusterisation, car cette dernière permet à chaque nœud d'obtenir des informations sur ses voisins et le chemin vers son CH.

Afin de commencer cette phase, les clusterheads envoient des messages de découverte vers le puits en se fondant en partie sur le principe du protocole de routage réactif DSR (*Dynamic source routing*). Ce dernier se fonde sur une technique d'inondation pour découvrir l'itinéraire d'une source vers une destination. Durant ce processus, les nœuds construisent et mettent à jour leur tables de routage. Les clusterheads initient la phase de découverte en diffusant des requêtes (*RReq*, *route request*) sur tous leurs voisins. Ces derniers rajoutent leur identité au message reçu et le diffusent à leur tour sur leurs voisins, jusqu'à ce que celui-ci atteigne sa destination finale (la *BS*) ou un nœud connaissant le chemin vers le puits. La destination finale (nœud ou sink) répond alors de proche en proche aux nœuds sources *S* avec un message (*RResp*, *route response*). Tous les nœuds obtiennent ainsi un ou plusieurs chemins qui mènent à la *BS*.

Afin de limiter le volume d'informations maintenues durant ce processus, les agents que nous utilisons dans notre approche filtrent les informations reçues et ne gardent que les informations concernant leur voisinage direct. Étant donné que le protocole DSR n'est pas adapté aux RCSF, car consomme beaucoup d'énergie, les agents n'utilisent son principe que pour l'envoi des messages de découverte. La maintenance du voisinage est alors faite suivant une stratégie peu consommatrice en énergie que nous avons mis en oeuvre.

La figure 3.5 illustre la table de routage d'un nœud simple dont l'identifiant est 41 et appartenant au cluster 49 de la figure 2.5. La première ligne de la table de routage contient les informations sur le nœud lui même, puis les autres lignes contiennent les informations relatives aux voisins directs. Chaque ligne contient les informations suivantes : l'identifiant du nœud, l'identifiant du cluster (i.e. du CH) auquel il appartient, son statut dans le cluster (SN, GN ou CH), la distance qui le sépare de son CH et finalement la distance vers la BS.

Dans cet exemple qui représente une table de routage maintenue par le nœud 41, le nœud de passage de 41 vers le CH est le nœud 45 se trouvant à distance 1 de celui-ci, tandis que son nœud de passage vers le puits est le nœud 44 qui est à distance 1 de celle-ci.

Mise à jour des routes

Durant cette phase, les nœuds mettent à jour leurs tables de routage avec une technique de communication d'informations proactive de type *push* selon une vue située. Cette opération est déclenchée par

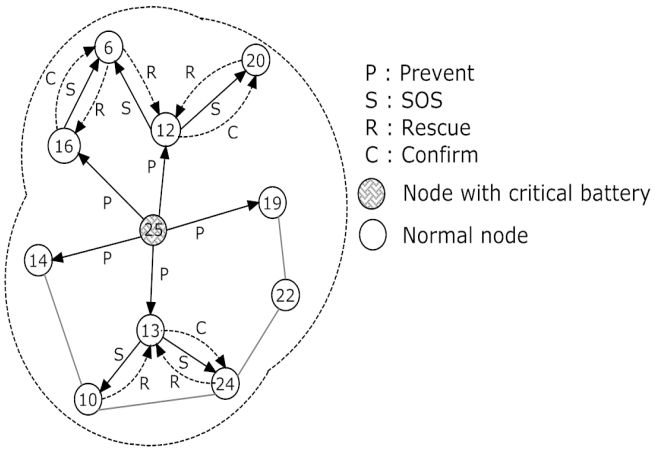


FIGURE 3.19 – Maintenance du voisinage

un agent lorsque son capteur n'a plus suffisamment d'énergie pour participer à l'opération de routage. Cela permet ainsi aux voisins qui le considèrent comme nœud de passage de mettre à jour leur table de routage et d'en choisir un autre pour cette fonction. Comme le montre la figure 3.19, la mise à jour des routes se fait au moyen de quatre types de messages :

- Messages de prévention (*P*) : l'agent du nœud ayant un niveau de batterie qui a atteint un seuil critique prévient ses voisins afin que ceux-ci puissent éviter dans le futur de le solliciter pour router leurs informations vers le CH ou la station de base ;
- Messages d'aide (*S*) : un agent qui reçoit un message de prévention et qui utilise le nœud source comme nœud de passage diffuse un message d'aide à ses autres voisins à 1 saut pour leur demander de devenir son nœud de passage ;
- Messages d'assistance (*P*) : lorsqu'un agent reçoit un message d'aide, il répond au demandeur en lui envoyant un message contenant son adresse et son niveau d'énergie résiduelle ;
- Messages de confirmation (*P*) : après la réception de messages d'assistance, l'agent en quête de passerelle choisit le nœud qui a la plus grande quantité d'énergie résiduelle comme passerelle, puis il informe celui-ci du changement en lui envoie un message de confirmation. Lorsque le nœud passerelle reçoit ce message, il met à jour sa base de connaissance, qui contient entre autres la liste des nœuds le considérant comme passerelle.

Dans cet exemple, le nœud 25 épuise sa réserve d'énergie et diffuse donc un message de prévention sur ses voisins directs 12, 13, 14, 16 et 19. Les nœuds 12, 13 et 16 l'utilisant comme nœud de passage, envoie une demande d'aide (d'adhésion) à leurs voisins respectifs. Ceux-ci, répondent s'ils ont un niveau d'énergie suffisant

3.7 Travaux en cours et perspectives

À la suite des travaux sur la clusterisation à 1 saut et durant le doctorat de Bachar Salim Haggar, nous avons proposé un algorithme de construction simultanée de clusters et d'arbres couvrants. Je souhaiterais à court terme étendre les résultats obtenus sur les clusters à k sauts. La difficulté majeure dans la construction de l'arbre simultanément à la construction de clusters à k sauts se situe dans la propagation du nœud choisi entre les clusters. Dans le chapitre 3.1, la construction de l'arbre couvrant est en réalité la jonction de plusieurs arbres. Dans chaque cluster, un arbre est construit et entre chaque cluster, nous avons également la construction d'un arbre. Donc quand des liaisons multiples existent entre plusieurs clusters, nous sommes dans l'obligation de désactiver certaines liaisons. Une piste de recherche est d'étudier le mécanisme utilisé par le protocole de routage BGP [RLH06] pour résoudre un problème équivalent. Grâce à ses métriques et dans le cadre d'une configuration dite de "dual homing" avec plusieurs routeurs locaux [1], un administrateur peut agir sur le choix de la liaison à utiliser vers une autre système autonome. Si nous assimilons les clusters à des systèmes autonomes, nous pourrons donc nous inspirer des échanges effectués entre les routeurs BGP afin de prioriser une liaison par rapport à une autre.

A partir de la construction d'arbres couvrants, nous pourrions enraciner nos arbres sur les puits afin de s'adapter au mieux aux réseaux de capteurs et ensuite utiliser les résultats obtenus sur l'agrégation collaborative des données sur cet arbre. Sachant que sur l'agrégation, il est aisé d'envisager plusieurs scénarios : un questionnement du voisinage quand un nœud souhaite transmettre, aucun questionnement mais avec une agrégation sur tous le chemin ou bien une agrégation régulière initiée par le cluster-head pour ensuite faire l'envoi. Le problème que l'on risque de rencontrer sur l'agrégation, c'est la taille des messages. En effet, il faut conserver l'identité de la source et les informations à transmettre. En pratique, en fonction de la taille des clusters et des tailles des identités, il se peut que l'on dépasse la limite du protocole de communication utilisé.

3.8 Publications majeures

[1] S. Romaszko, J. Carle, and F. Nolot. Ad hoc routing protocol analysis in civil safety context. In *Ad Hoc Networking Workshop (Med-Hoc-Net), 2010 The 9th IFIP Annual Mediterranean*, pages 1–6, 2010.

[2] O. Flauzac, B.S. Haggar, and F. Nolot. Self-stabilizing tree and cluster management for dynamic networks. In *IICS'10*, pages 20–29, 2010.

1. http://www.cisco.com/en/US/tech/tk365/technologies_configuration_example09186a00800945bf.shtml

Chapitre 4

Gestion des données et de leur sécurité

Résumé.

A la suite des travaux sur l'auto-organisation des réseaux ad-hoc puis sur l'acheminement de l'information dans les réseaux organisés en clusters, je me suis intéressé à la gestion des données qui transitent sur le réseau. En effet, nous construisons des mécanismes permettant de diminuer le nombre de messages de contrôle dans le but de construire des clusters, des arbres ou des tables de routage mais à aucun moment, nous nous intéressons à la donnée transportée. Quand les liaisons sont établies entre les différents nœuds, que les clusters sont construits et qu'une solution de routage nous permet d'envoyer une information, comment puis-je être sûr que l'information est fiable et sans risque ? Des liaisons sécurisées vont juste m'assurer que personne ne peut intercepter les informations qui me sont destinées. Mais si mon interlocuteur n'est pas de confiance, quelle solution vais-je avoir sur un réseau ad-hoc ?

Dans un premier temps, je vais présenter le projet RemoteLabz qui m'a permis d'initier ma réflexion sur la fiabilité des données provenant des ordinateurs dans les réseaux filaires. Ce projet est une solution en mode IaaS de contrôle d'infrastructure informatique dans le but que toute personne puisse réaliser des travaux pratiques à distance dans le domaine des réseaux. Pour cela, elle gère donc un acheminement dynamique et automatisé des informations dans le but de construire de multiples architectures réseaux complexes à partir d'un câblage physique toujours identique. De plus, toute personne peut insérer dans la solution ses propres équipements et machines virtuelles grâce à son accès Internet. De ce projet est né la réflexion sur la sécurité dans un réseau ad-hoc. Dans ce dernier, chaque entité peut infecter ses voisins car automatiquement, l'acheminement de l'information se fait sans contrôle. Je présente donc dans une deuxième partie une réponse à cette problématique : le concept des grilles de sécurité. Basé sur une approche grille et des échanges pair à pair, j'ai conçu une approche distribuée de la sécurité.

Le projet Remotelabz est le fruit des résultats obtenus dans le cadre d'aides aux transferts technologiques OSEO et les résultats sur les grilles de sécurité découlent des travaux et réflexion sur les extensions du RemoteLabz. Ces travaux ont donné lieu à 1 *chapitre de livre d'audience internationale et* 1 *conférence d'audience internationale.*

4.1 Administration automatisée de la gestion de l'acheminement des données

Lors de travaux pédagogiques dans le domaine des réseaux informatiques, une problématique est vite apparue : celle de la rationalisation des équipements réseaux et de leur usage à distance. En effet, pour que les étudiants puissent pratiquer dans un environnement proche de la réalité, il est indispensable de leur offrir un accès à des équipements informatiques professionnels. L'objectif a donc été d'étudier les solutions existantes et la faisabilité de développer un gestionnaire de contrôle d'infrastructure comprenant des équipements réseaux et des ordinateurs dans le but de pouvoir, à la demande, démarrer et utiliser des équipements pour la formation. Mon souhait était de mettre en place une solution, pour des équipements réseaux, avec un fonctionnement similaire à celui des solutions utilisées par les hébergeurs pour offrir des machines virtuelles à leurs clients. La principale différence avec les solutions des hébergeurs est de pouvoir offrir un service, à la demande, pour de courtes durées avec une gestion énergétique des équipements. Seules 3 solutions existantes [Cen, Gro, Lab] semblent être proche de notre objectif et seule la solution Netlab est une solution commerciale. Malheureusement, la solution Netlab est dépendante des constructeurs, imposent le matériel et n'offre pas d'extensions possibles comme l'interconnexion de plusieurs Netlab afin d'offrir un service en mode IaaS au sein d'un cloud, de réservation de ressources. Ce qui représente un de mes objectifs finals. J'ai également étudié la solution Grid5000 [Gri]. Cette solution est une plate-forme expérimentale permettant de mener des expériences larges échelles dans le domaine du calcul distribué ou parallèle. Malheureusement, elle n'offre pas de solutions à ma problématique : une solution de réservation de ressources puis un contrôle et un déploiement automatisés en mode IaaS d'équipements informatiques. La solution Grid5000 est dédiée au contrôle d'ordinateurs.

Dans la suite de cette section, je commencerai par une présentation du projet RemoteLabz puis par les technologies utilisées afin d'atteindre nos objectifs. Je conclurai par les perspectives de travail autour de ce projet.

4.1.1 Le projet RemoteLabz

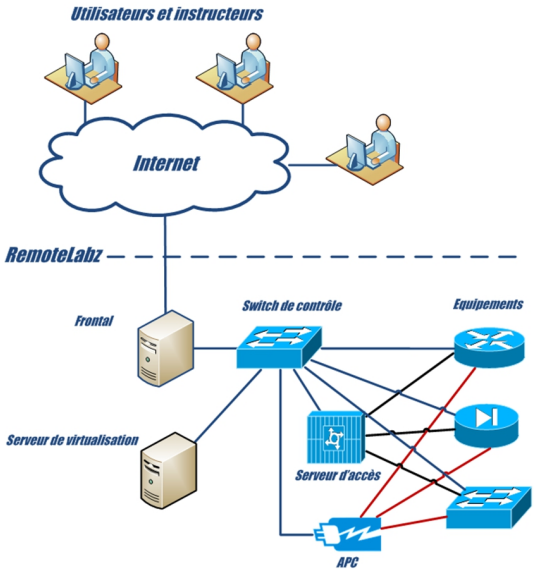


FIGURE 4.1 – Le projet RemoteLabz

La solution RemoteLabz [MN09, DN10b, DN10a] permet de réserver des équipements réseaux et ordinateurs virtualisés, via un navigateur Web puis de les utiliser à distance. Mais à partir d'un câblage

physique qui doit être le plus générique possible, nous devons pouvoir reconstruire, logiquement, de multiples architectures réseaux. Pour atteindre cet objectif, un contrôle électrique des équipements réseaux a été mis en place ainsi qu'une gestion des flux de données de chaque équipements informatiques.

Le RemoteLabz peut être schématisé par la figure 4.1. Les utilisateurs ne peuvent avoir un accès à distance, après réservation, que des équipements réservés. Le RemoteLabz quant à lui communique avec des équipements de contrôle, qui se classe en 4 catégories : les switchs de contrôle pour la gestion de l'acheminement de l'information, les serveurs d'accès pour offrir aux utilisateurs des consoles de contrôle aux équipements, les prises électriques appelées APC sur la figure 4.1 et les serveurs de virtualisation pour contrôler les machines virtuelles mises à disposition des utilisateurs.

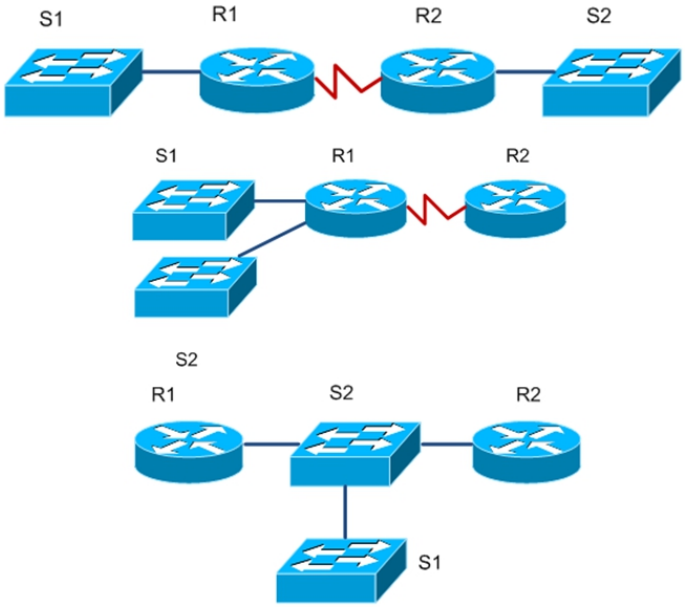


FIGURE 4.2 – Architecture créée

Les utilisateurs peuvent alors contrôler des architectures réseaux classiques ou complexes, comme illustrées à la figure 4.2. Hors, pour dynamiquement construire ces architectures, nous partons d'un câblage physique générique, représenté par la figure 4.3.

Un contrôle de l'acheminement de l'information a donc du être mis en place. Pour cela, il n'est pas envisageable de faire du routage, c'est à dire considérer des protocoles de la couche 3 du modèle OSI. En effet, il est nécessaire d'offrir un contrôle total de l'architecture créée à l'utilisateur pour qu'il puisse avoir l'impression que ses équipements sont inter-connectés par des liaisons physiques et non pas par des liaisons virtuelles. Je me suis donc orienté vers l'usage du protocole IEEE 802.1Q [gW]. Ainsi, un cloisonnement sur la couche 2 du modèle OSI par l'intermédiaire de Virtual Local Area Network (VLAN) a été effectué, puis en fonction du l'interconnexion souhaitée, le RemoteLabz commute les ports de connexion correspondant dans les VLANs corrects. Par exemple, si l'on souhaite obtenir la 1ère architecture de la figure 4.2, il suffit de commuter les ports 0 et 3 du switch de contrôle dans le même VLAN puis les ports 2 et 4, tout en veillant que les ports restant (1, 5 et 6) soient arrêtés. En utilisant la même technique, il est également possible de construire à la demande des maquettes plus complexes comme par exemple les architectures de la figure 4.4.

4.1.2 Les évolutions

Le projet RemoteLabz est donc une solution IaaS de gestion d'infrastructures informatiques sur un site identifié. J'ai ensuite travaillé à la mise en place d'une solution en mode SaaS, c'est à dire de localiser plusieurs RemoteLabz, géographiquement sur plusieurs sites et offrir la possibilité de réserver

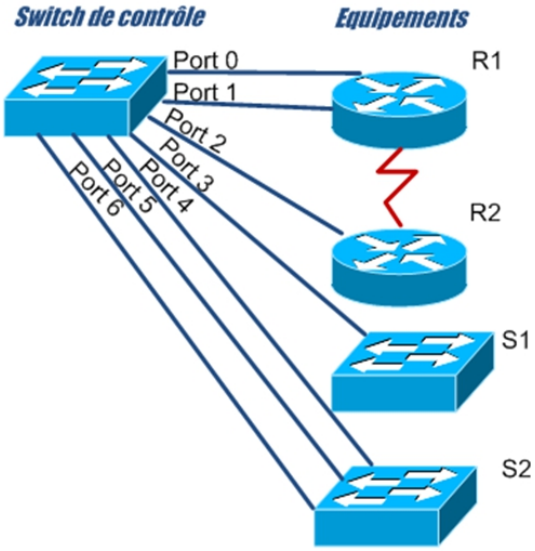


FIGURE 4.3 – Interconnexion des équipements

des équipements réseaux, en fonction de leur disponibilité. Évidemment, l'utilisateur ne sait pas où se situent les équipements et il a au final toujours la même vision de son architecture réservée. La vision qu'il est possible d'avoir de cette évolution CloudLabz est décrite par la figure 4.5.

Pour réussir à atteindre ce nouvel objectif, je me suis à nouveau tourné vers des solutions en mode SaaS qui permettent de faire de la réservation de ressources multi site. Grid5000 est en mesure d'offrir un service similaire mais il se base sur une architecture physique très spécifique, à savoir des liaisons dédiées entre les sites et exclusivement des protocoles de couche 2 comme le QinQ disponible dans le protocole IEEE 802.1q . Il n'était pas envisageable de passer par cette approche car la vocation du Cloudlabz est la rationalisation des équipements et des coûts. Cela passe donc dans un premier temps, par l'utilisation d'une interconnexion entre les RemoteLabz via IP. Afin de simuler de la connexion couche 2 entre les sites, je me suis orienté vers le protocole L2TP [TVR+99], implémenté dans le logiciel OpenVPN. Les expérimentations nous ont amené à aboutir à la réalisation de la maquette de la figure 4.6.

L'avantage de cette approche et de l'utilisation du protocole L2TP est sa souplesse. Ainsi, toute personne peut être en mesure d'insérer dans une maquette ses propres machines virtuelles par exemple, ou équipements informatiques. Mais pouvoir offrir une grande souplesse d'utilisation comme la possibilité à un utilisateur de connecter ses propres machines virtuelles à une architecture réservée risque de corrompre notre solution. Comment sécuriser la connexion des machines virtuelles d'utilisateur au CloudLabz et comment être sur que ces machines virtuelles ne vont pas compromettre le CloudLabz ? De ces questions est né le concept des grilles de sécurité présenté dans la suite de ce chapitre.

4.2 Les grilles de sécurité

A la suite des travaux de recherche sur le projet RemoteLabz, il est apparu qu'il était possible d'aller plus loin que juste la gestion des chemins que doivent prendre des données. Dans le projet RemoteLabz, j'ai utilisé le protocole IEEE 802.1q pour assurer l'isolation des données au sein des différents switchs traversés. La question a donc été de se dire : mais comment pourrions-nous faire sur un réseau ad-hoc et en plus, de façon non centralisée ? De ce questionnement est né le concept de grille de sécurité publié dans [FNRS09, FNRS12]. J'ai proposé une nouvelle solution de sécurité globale, totalement distribuée, basée sur une approche pair à pair des communications.

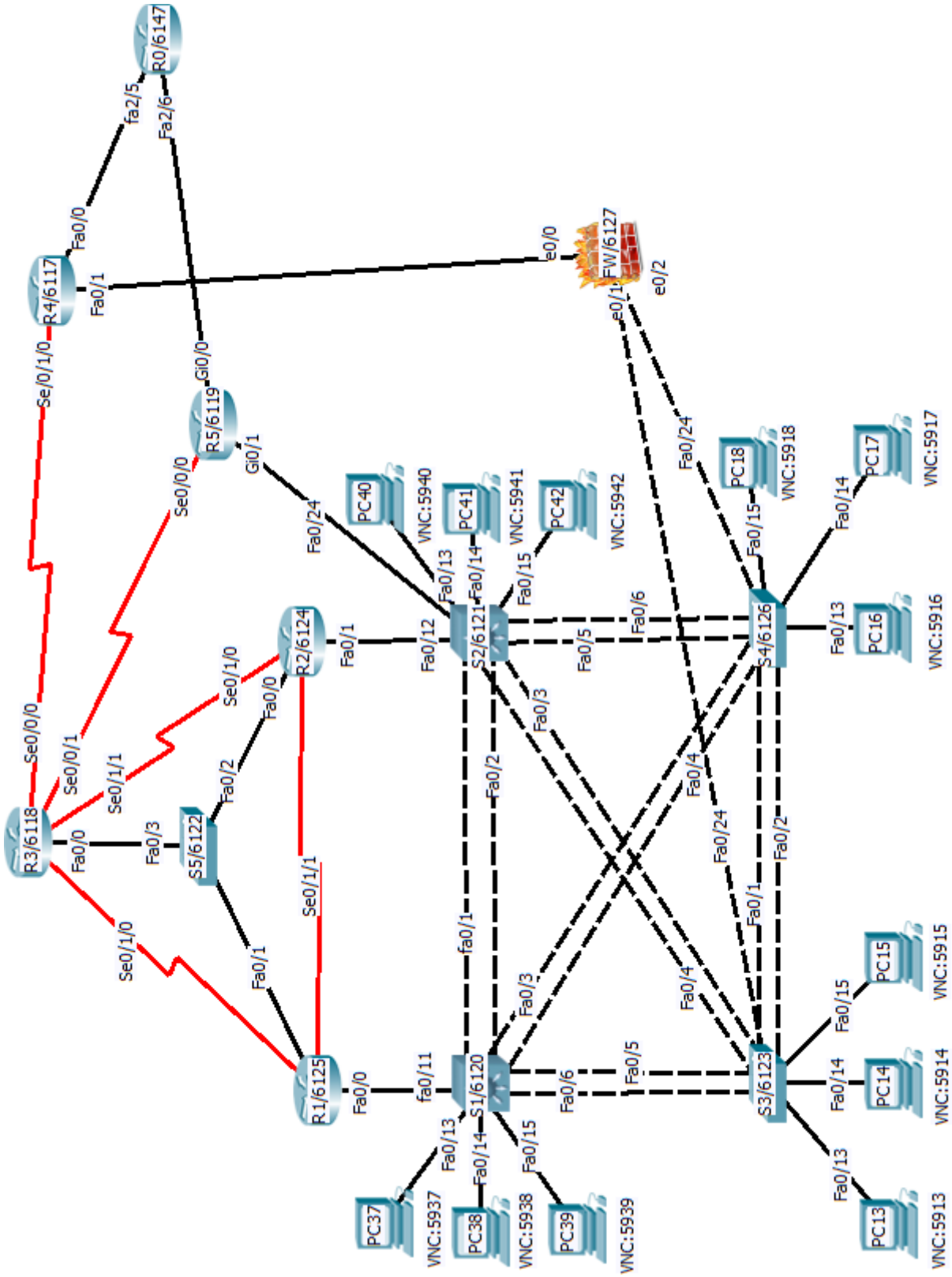


FIGURE 4.4 – Architecture complexe

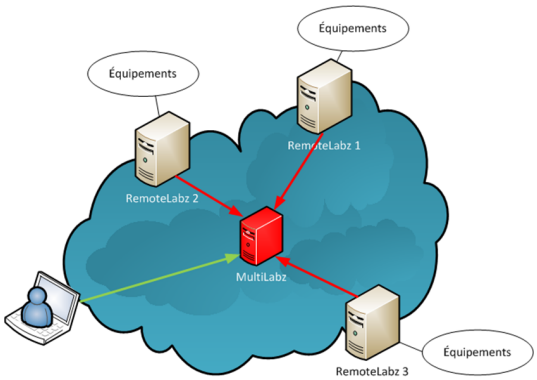


FIGURE 4.5 – Le CloudLabz

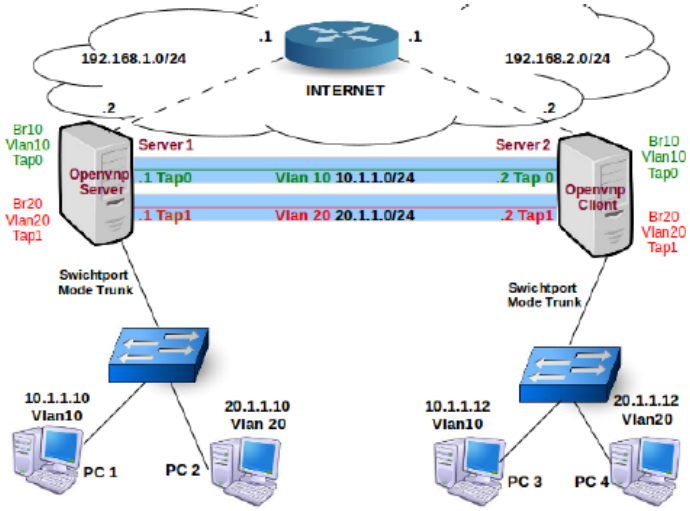


FIGURE 4.6 – Schéma technique du CloudLabz

4.2.1 Introduction

Le déploiement et la gestion des règles de sécurité se font actuellement encore de façon totalement centralisées. De très nombreuses solutions de sécurité comme la détection ou prévention d'intrusions, les réseaux virtuels privés ou les firewalls possèdent des règles de gestion centralisées et sont donc des solutions beaucoup plus adaptées aux réseaux filaires. Dans le cadre des réseaux sans fil, comme le wifi en mode infrastructure, ces solutions sont également satisfaisantes car les données Wifi transitent par la borne, c'est à dire un point central de communication qui relaie l'information sur le réseau filaire. Aujourd'hui, les équipements sans fil sont très nombreux et il est aisé de construire un réseaux sans fil que l'on interconnecte au réseau filaire par l'intermédiaire de son ordinateur. La problématique est donc de réussir à intercepter les données provenant de l'équipement mobile, qui vient interagir avec le réseau filaire. De même, si un utilisateur souhaite construire un réseau ad-hoc, comment fera-t-il pour sécuriser les échanges entre tous ses équipements et surtout veiller à ce que les données soient saines. En effet, l'usage démocratisé de liaison sécurisée comme les VPN permettent d'éviter que quiconque puisse comprendre les échanges mais cela n'empêche pas un virus ou un vers de se propager dans le réseau, si un ordinateur est contaminé.Ainsi, depuis l'intérieur d'un réseau, une personne qui possède un accès légitime, est donc en mesure de mener des attaques en toute tranquillité.

C'est à la suite de ce constat qu'est né le concept de *grille de sécurité*. Dans cette nouvelle approche, les règles de sécurité sont distribuées sur tous les équipements de confiance et c'est uniquement par l'association de toutes ces règles de sécurité qu'il est possible de connaître la politique globale de sécurité

du réseau. Pour cela, une communauté est définie, comme étant un ensemble d'équipements qui partage la même politique globale de sécurité. Avec cette approche collaborative, les équipements échangent entre-eux leurs règles locales de sécurité afin de savoir si les données doivent ou ne doivent pas être échangées. Ainsi si un utilisateur souhaite mettre en place un serveur FTP, il va ainsi créer une nouvelle règle de sécurité locale. Cette règle sera alors échangée avec les autres utilisateurs de sa communauté et ils devront valider cette règle. Ainsi, il devient aisé et rapide de construire un réseau dont les échanges et les usages seront sécurisés, sans aucune centralisation.

Cette solution de grille de sécurité exploite une architecture logiciel conçu pour le grid computing et des communications de type pair à pair. L'architecture logicielle de type grille apporte les avantages de la rationalisation de toutes les ressources de chaque équipement, que ce soit le stockage, les ressources de calcul ou d'analyse des données. Quant aux communications de type pair à pair, elles permettent de construire des interconnexions totalement décentralisées.

4.2.2 La problématique

De nos jours, Internet est l'infrastructure d'échange d'informations la plus utilisée au monde et aussi la moins sécurisée. De plus, avec l'augmentation croissante d'équipements connectés comme les réfrigérateurs, les climatisations, les systèmes de gestion d'énergie ou de surveillance, les voitures, les risques de sécurité de cessent de croître. Le vol d'informations deviendra de plus en plus aisé car le nombre de sources est de plus en plus important. Malheureusement, même si le budget en matière d'équipements de sécurité augmente régulière dans toutes les entreprises, cela reste toujours des outils centralisés, avec éventuellement de la collaboration entre-eux afin de pouvoir associer leur fonctionnalité mais rarement jusqu'au poste de l'utilisateur.

Dans l'approche des grilles de sécurité, les contraintes d'un réseau d'une entreprise connectée à Internet a été le point de départ de la modélisation. L'ensemble des équipements interconnectés dans une entreprise constitue ce qui est appelé "une zone de confiance". Cette zone est délimitée par tous les équipements connectés à Internet, i.e. les équipements avec une adresse IPv4 publique ou une adresse IPv6 globale. Officiellement, une zone de confiance inclues tous les équipements qui communiquent sur le réseau et dont les règles de sécurité sont contrôlées mutuellement. Une zone de confiance peut donc être étendue au travers de liaisons WAN ou réduite à quelques équipements si les équipements se mettent d'accord sur une politique de sécurité commune.

4.2.2.1 Quelques exemples actuels

Je présente dans cette section trois scénarios qui représentent la façon dont est actuellement géré la sécurité. Le premier scénario sera la protection contre les intrusions, le deuxième la connexion à une zone de confiance au travers d'une liaison internet et la dernière l'exemple de communication sécurisée à l'intérieur d'une zone de confiance. Dans tous les cas, il est aisé d'observer que dans les réseaux actuels, les informations sont relayées vers un service centrale. De ce fait, les communications entrantes et sortantes d'un réseaux passent généralement par un firewall, un détecteur d'intrusion et éventuellement par un concentrateur VPN. Quelques constructeurs comme Cisco Systems, Juniper ou CheckPoint augmentent la centralisation des fonctionnalités dans une même appliance. Je reste persuadé, même si ces boîtiers sont doublés pour assurer une haute disponibilité du système, que cela augmente le risque de défaillance en cas d'attaques.

Les systèmes de détection d'intrusion

Ce scénario représente typiquement un réseau derrière un firewall. Seules les données autorisées peuvent atteindre le réseau internet. Les politiques de sécurité identifient les adresses sources et destinations, les ports de communication et les protocoles. Généralement, à l'entrée du réseau de l'entreprise (cf. Figure 4.7), sont placés les firewalls ainsi que des firewalls sur les postes des utilisateurs.

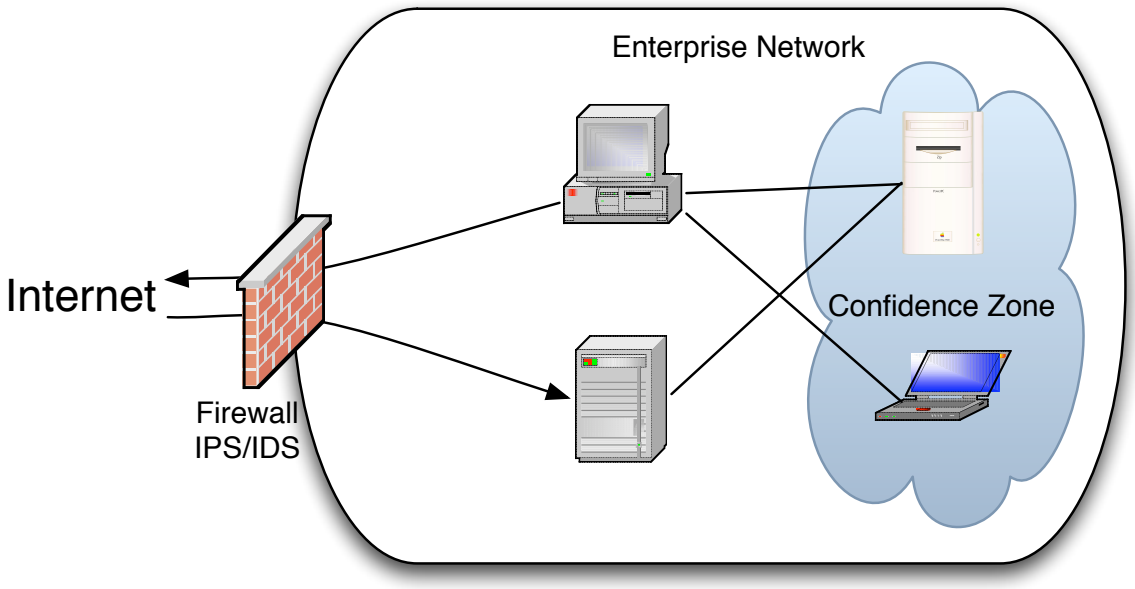


FIGURE 4.7 – Un réseau typique avec firewall

En raison de leur positionnement et leur fonctionnalité, les firewalls (personnels ou pas) sont de bons outils pour bloquer les attaques provenant de l'extérieur ou bien de l'intérieur. Cependant, quand un administrateur peut contrôler les flux qui entrent ou sortent de son réseau, il ne peut pas s'assurer que les autres points d'entrée de son réseau, ouverts à son insu par un utilisateur, sont sécurisés. En effet comment vérifier les données échangées par un utilisateur qui a interconnecté le réseau de l'entreprise à Internet grâce à son téléphone. Pour pouvoir s'assurer que tous les équipements de la zone de confiance partage la même politique de sécurité, il faut donc implémenter un contrôle additionnel sur les échanges internes. En raison de la complexité de la tâche, la coordination des politiques de sécurité doit donc être mise en place par des mécanismes transparents pour l'utilisateur.

Connexion à une zone sécurisée

Les données autorisées par un firewall à atteindre une zone sécurisée n'assurent pas la confidentialité des données qui transitent sur Internet. C'est pourquoi des fonctionnalités additionnelles sont nécessaires comme le chiffrement, l'authentification et l'intégrité. Ces propriétés sont notamment fournies par les Virtual Private Network (Figure 4.8). Cependant, d'autres protocoles fournissent des propriétés similaires (SSL, SSH) mais les VPN ont l'avantage d'intégrer toutes les fonctionnalités sur les données véhiculées et de plus permettent d'étendre le réseau local en intégrant les équipements distants. Ainsi, les utilisateurs peuvent utiliser les ressources de l'entreprise, comme les imprimantes ou les serveurs de mail internes. Mais les VPN n'empêche pas la propagation d'un virus ou d'un vers au sein du réseau de l'entreprise. Il se contente de sécuriser la propagation de ce virus.

L'accès à une zone de confiance par un VPN nécessite un équipement central (communément appelé concentrateur VPN). Comme tout le traffic est relayé par cette équipement centrale, la bande passante est limitée. Des constructeurs cumulent les fonctionnalités de VPN, détecteur d'intrusion et de firewall dans le même équipement, simplifiant ainsi la gestion des flux réseaux et l'architecture du réseau mais cela limite encore plus les bandes passantes disponibles. A titre d'exemple, le produit Cisco ASA 5550 est un firewall qui atteint seulement 425 Mbps si les fonctionnalités de VPN et de firewalling sont activées, contre 1.2 Gbps avec uniquement le firewall actif [1].

1. http ://www.cisco.com/en/US/products/ps6120/prod_models_comparison.html

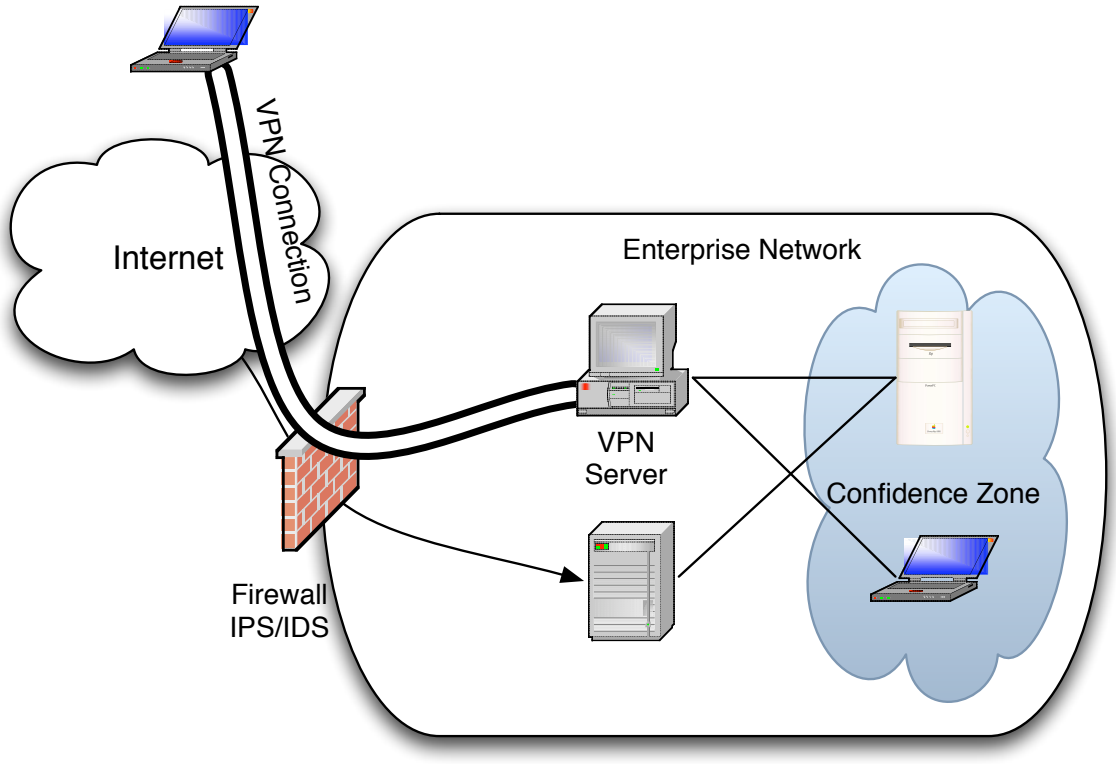


FIGURE 4.8 – Usage d'un VPN dans un réseau actuel

Établir une zone de confiance Permettre à une machine d ?accéder à une zone de confiance est toujours une décision dangereuse car elle peut infecter les autres. Pour réduire les risques et vérifier si un équipement est conforme aux règles de sécurité, des sociétés comme Cisco Systems, Microsoft ou Nortel Networks proposent des mécanismes appelés *Network Access Control* (NAC). Un NAC regroupe une authentification des utilisateurs et une vérification de leur poste de travail avant d'accepter les échanges de données avec le reste du réseau. Parmi les éléments que les mécanismes NAC vérifient (ou imposent), on trouve :

- le statut du logiciel anti-virus ; son activation et sa mise à jour ;
- les mises à jour de sécurité du système d'exploitation ;
- les certificats à clé publique installés ;
- le statut du firewall personnel et ses règles ;
- les applications autorisées à communiquer sur le réseau ;
- la permission pour activer des connexions Wifi ou Bluetooth ;
- etc.

Les appliances de contrôle de type NAC analysent également le comportement des équipements réseaux. Par exemple, si un téléphone tente d'établir une connexion *telnet* avec un ordinateur au lieu d'échanger de l'information avec le gestionnaire d'appels, une alerte est alors envoyé à l'administrateur du réseau. Parmi les réactions possibles, l'appliance peut également isoler automatiquement l'équipement générateur de problèmes et le mettre dans une zone dite de quarantaine en agissant sur le switch qui interconnecte au réseau cet équipement. L'inconvénient de cette approche réside dans la centralisation des échanges sur l'appliance NAC.

Dans le même philosophie de fonctionnement, il existe des équipements de supervision du comportement du réseaux (Figure 4.9). Avec des fonctionnalités qui vont du simple affichage des statistiques du

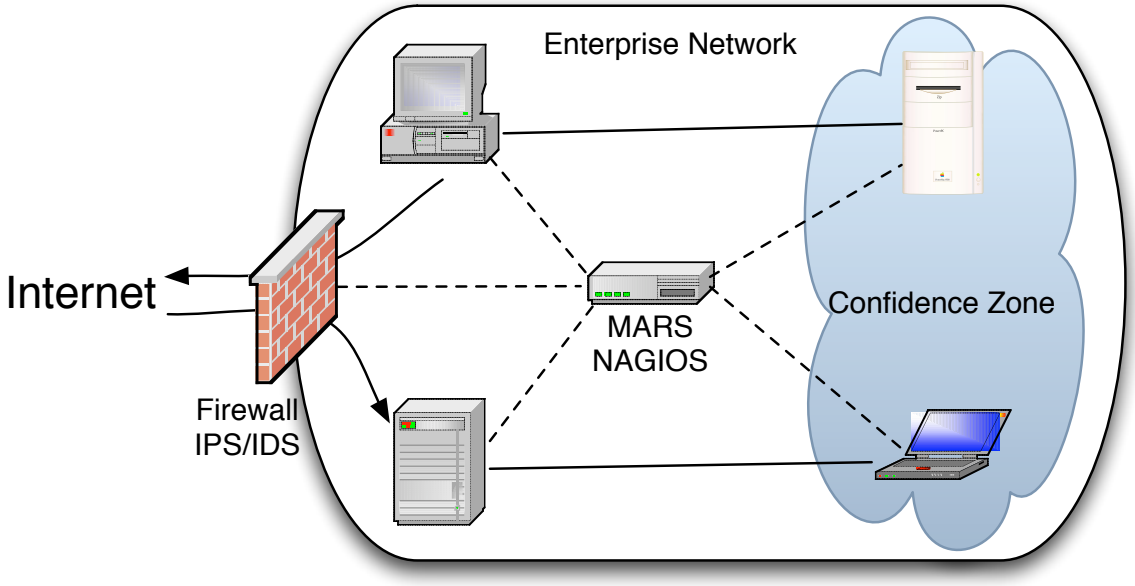


FIGURE 4.9 – Surveillance d'une zone de confiance

réseau à la gestion pro-active de celui-ci (à la manière d'un système de prévention d'intrusions - IPS), ces services sont utiles pour identifier la faiblesse d'un réseau. Cisco Systems proposent pour cela une solution appelée MARS [2] où les informations sont échangées au travers d'agent MARS, améliorant ainsi l'analyse du réseau.

Toutes ces approches sont des solutions centralisées. Dans l'approche des grilles de sécurité, les fonctionnalités des NAC sont distribuées dans plusieurs équipements dans le but d'éviter la surcharge d'un seul équipement.

4.2.3 Une piste pour sécuriser des réseaux IPv6

Le déploiement d'IPv6 fait partie des grandes évolutions des réseaux d'aujourd'hui et de demain. Avec plus de (2^{128} en IPv6 contre 2^{32} en IPv4, la plage d'adresses IPv6 est beaucoup plus importante. De plus avec l'abandon de la notion d'IPs privées et des mécanismes de NAT tels qu'ils existaient sur IPv4, tous les équipements réseaux connectés à Internet en IPv6 seront directement accessibles. Tout type d'équipements seront vulnérables (PC, téléphone, capteurs, ...) car posséderont une adresse IP publique. De plus, avec la technologie Mobile IPv6 [EV06, Val08], les téléphones pourront se déplacer d'un réseau à un autre, en conservant leur connexion précédemment établie. Les différents routeurs échangent les informations de connexion afin de permettre aux utilisateurs de passer d'un réseau à l'autre, de façon totalement transparente pour les utilisateurs.

Ce nouveau mécanisme peut être exploité par un attaquant [LZS08, EV06] afin de pénétrer le réseau en utilisant les connexions préalablement établies par l'équipement mobile. Quelques travaux [HB08, EE07] existent pour sécuriser Mobile IPv6 grâce notamment aux outils IPSec mais des fonctionnalités spécifiques sont donc à déployer.

2. http ://www.cisco.com/go/mars

4.2.4 Sécuriser un réseau ad-hoc

Un réseau ad-hoc est un réseau sans infrastructure dans lequel chaque équipement communique avec ses voisins. Cela différe des solutions d'entreprises qui ont été conçus pour des réseaux avec infrastructure, même dans les environnements sans fil où la sécurité est établie par des échanges sécurisés entre le client et le point d'accès. Actuellement, les travaux dans le domaine de la sécurité et des réseaux ad-hoc sont basés sur des solutions cryptographiques [LZ99, CS07, CF08]. Ces solutions sont suffisantes pour assurer la confidentialité et l'intégrité des données mais ne sont pas conçus pour déployer une politique de sécurité globale sur le réseau. D'autres travaux [PYY+04] se focalise sur les données et la sécurité des échanges mais toujours avec des oslutions similaires aux réseaux avec infrastructures fixes.

Dans l'approche des grilles de sécurité, les sécurisations de la communication entre 2 équipements peuvent être implémentées sur chaque équipement par la négociation mutuelle d'une politique de sécurité mais il est également possible de vérifier les règles d'accès et d'échanges des informations.

4.2.5 Présentation générale des grilles

Les grilles sont une des solutions qui permet de gérer les ressources disponibles sur un réseau. 2 types de grilles sont distinguées : les grilles de calcul et de données. Dans les solutions de grilles de calcul (SETI@home [ACK+02], BOINC [And04], XtremWEB [CDF+04], Diet [CD06], Globus[ACF+01], and CONFIIT [FKF03, KFM04, KF]), les ressources sont associées au calcul (processeur, mémoire, ...). Dans les grilles de données (OceanStore [KBC+00], Freenet [Fre]), les ressources sont associées aux stockage des données. Quelque soit le type de grille, il est necessaire de développer un intergiciel pour gérer les différentes ressources : la connectivité, les surveillances des ressources, la gestion des taches dans les grilles de calculs et la réplication des données dans les grilles de données. Aujourd'hui, la majorité des grilles sont basées, soit sur des architectures hiérarchiques ou centralisées et dans les deux cas, cela nécessite de très nombreuses tâches et chaque équipements doit être spécialisé.

En parallèle a ce concept des grilles, le modèle pair à pair a été développé. Dans le modèle de communication pair-à-pair (P2P), les architectures sont totalement décentralisées et peuvent être facilement agrandies. De plus, il est possible de gérer des architectures dynamiques, comme des réseaux ad-hoc. Le principal objectif des systèmes pair à pair est donc de permettre la communication entre chaque équipement sans outils additionnels.

Cependant, quand nous faisons le design d'un intergiciel ou d'une application pour les grilles, nous devons utiliser une modèle théorique. Comme les grilles, chaque équipement a une fonction spécifique et nous pouvons avec de très nombreux équipements dans le réseaux (commutateur, firewall, ordinateur personnel, serveur, ...). Le modèle doit décrire chaque fonction et chaque équipement dans la but de correctement étudier et évaluer l'application grille. Du modèle théorique et de cette étude, des solutions vont donc devoir être trouver pour chaque problème.

Pour être le plus efficace, un modèle doit donc prendre en compte tout le système. Le modèle de grille peut être appliqué à des environnements qui ne sont pas nécessairement dédiés. Dans ce cas, les applications et les mécanismes en dehors de la grille peuvent avoir des effets sur l'efficacité globale de l'intergiciel ou de l'application de grille. Le modèle doit également prendre en compte les caractéristiques physiques des équipements si nous souhaitons obtenir des mécanismes de gestion efficace. Cependant, les modèles proposés dans la littérature prend seulement une sous-partie du système. C'est pourquoi nous proposons une nouveau modèle théorique.

Dans [LPP04], les auteurs ont proposés une méthode qui se focalise sur la description des équipements. Ce modèle permet de décrire à la fois les protocoles utilisés et les équipements réseaux comme les commutateurs et les routeurs. Le réseau physique de la grille est représenté comme un graphe où chaque nœud représente un équipement réseau ou un type particulier de réseau (*(*Ethernet), *(*Myrinet) ...). L'intérêt d'un tel modèle, proche du réseau physique, est de pouvoir mettre en avant les problèmes de congestion réseaux ou de délai dans les transferts des données.

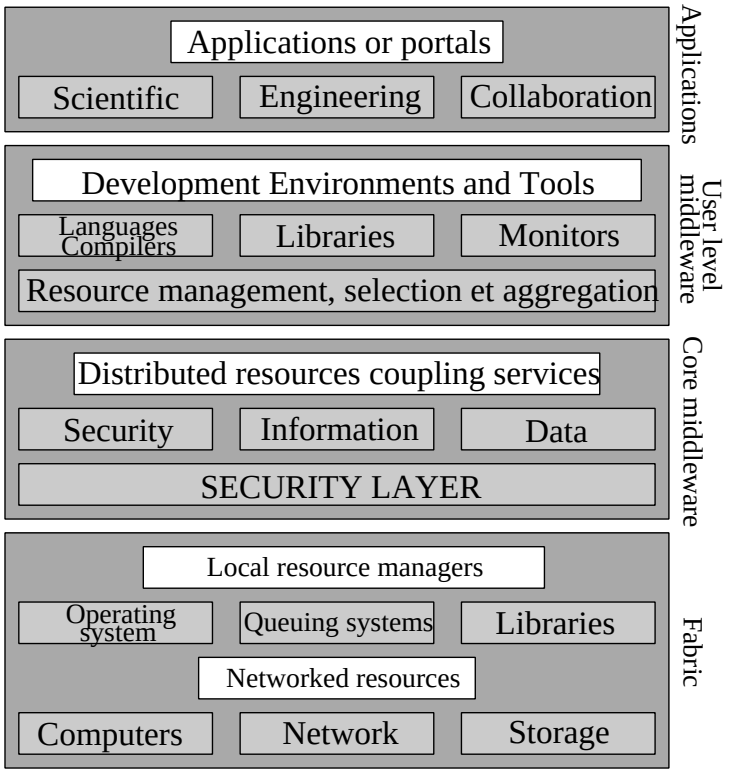


FIGURE 4.10 – Modèle de grille proposé dans [BBL02].

Dans [BBL02], une nouvelle approche est proposée, basé sur le concept d'une usine. Les composants et l'architecture de la grille sont organisés en couche comme illustré à la figure 4.10. La couche basse est proche du réseau physique et représente les ressources physiques de la grille. Ces ressources sont accessibles via un gestionnaire de ressources locales. La seconde couche représente et comment accéder aux ressources afin d'assurer la sécurité des connections. La troisième couche est l'intergiciel qui sert comme interface entre l'application et l ?accès aux ressources. La dernière couche représente l'application elle même qui fait fonctionner l'intergiciel. Contrairement au précédent modèle, il ne mais pas l'accent sur les problèmes proches du réseau physique mais plutôt sur les problèmes d'accès aux ressources en terme d'intergiciel et de service.

Un autre modèle communément utilisé est basé sur *Globus* [FK97]. Il se focalise de le matériel qui compose la grille comme l'illustre la figure 4.11. Un tel modèle est composé de 4 couches. La première représente le réseau physique, la deuxième les ressources de la grille : les ressources de calcul, le stockage ou les applications partagées. La troisième est relative aux composants et à l'intergiciel de services qui communiquent avec les ressources. Et la dernière couche est l'application qui utilise les services fournis par l'intergiciel.

Tous ces modèles ne prennent pas toutes les ressources du réseaux et de l'application de la grille. Un autre modèle a été proposé dans [RBF06]. Dans ce nouveau modèle théorique, les équipements physiques, les liens de communication et chaque ressource de l'application sont représentés. Ce modèle est le plus adapté au design d'une grille de sécurité dans lequel les équipements physiques, les liens de communication et leur politique ont un rôle crucial. Il est structuré en 5 couches indépendantes : couche physique, couche de routage, la couche de communication, la gestion des ressources puis tous les composants de l'intergiciel des services de la grille.

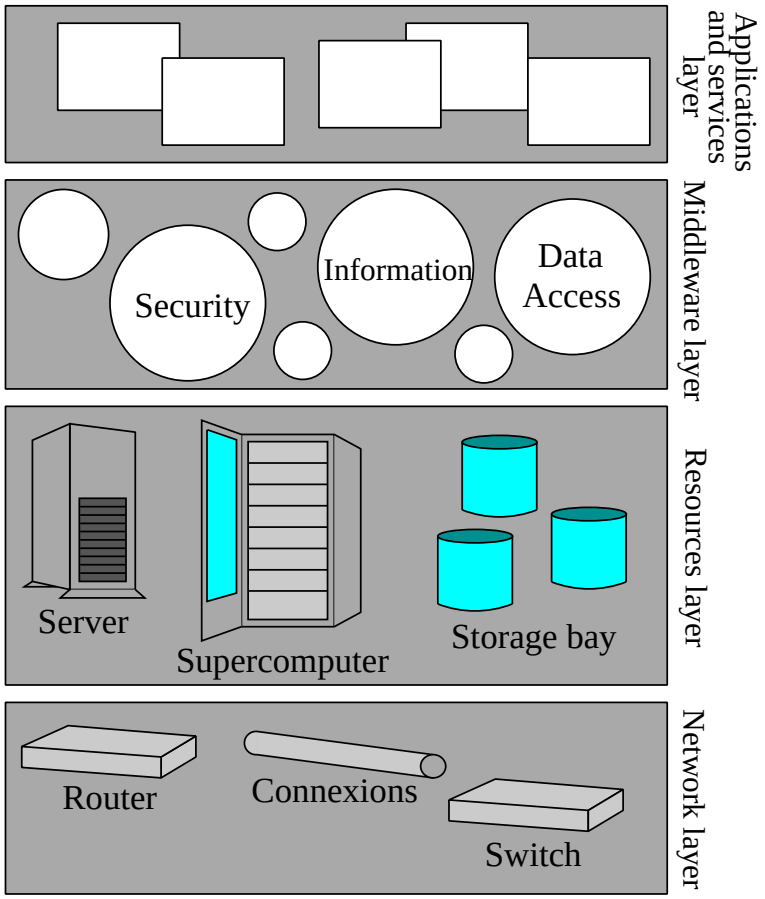


FIGURE 4.11 – Modèle de grille proposé in [FK97].

4.2.5.1 Le design de la grille

Le modèle publié dans [RBF06] est représenté par la figure 4.12. Il est utilisé pour modéliser le réseau de la grille qui interagit indépendamment de l'intergiciel de la grille. Il modélise également les composants de l'intergiciel et les interactions entre eux.

La couche 1 - Le réseau physique. La première couche concerne le réseau physique. Le réseau est représenté par un graphe $G_1 = (V_1, E_1)$. V_1 est l'ensemble des nœuds du réseau. Un nœud peut être un élément *actif* (comme les ordinateurs, les serveurs, ...), ou un composant *passif* (comme les routeurs, commutateurs, ...). E_1 est l'ensemble des liens qui interconnectent les nœuds du réseau. Nous distingeons deux types de liens : les connections filaires et sans fil. Les connexions sans fil sont naturellement non orientées. Mais pour les connexions sans fil, nous devons veiller aux differentes zones de couverture des nœuds du réseau. Si un nœud à une zone de couverture plus importante qu'un autre, cela induit une liaison orienté dans E_1.

La figure 4.13 montre un exemple de réseau (figure de gauche) representé suivant la première couche du modèle (figure de droite). Les nœuds (1 à 7) sont connectés avec des connexions filaires : les liens dans le graphe de correspondance sont non orientés. Pour les deux nœuds sans fil, nous pouvons remarquer que le nœud 9 a une plus grande couverture que le nœud 8 (les zones de couverture sont représentées par les cercles sur la figure).

Couche 2 - Le routage. Au dessus de la couche du réseau physique, les protocoles de routage permettent de construire et maintenir des chemins entre tous les nœuds du réseau. Deux nœuds peuvent donc communiquer même s'ils ne sont pas directement connectés l'un à l'autre. Par contre, la construction des chemins prend en compte les problématiques de sécurité déployées au dessus des sous-réseaux (fire-

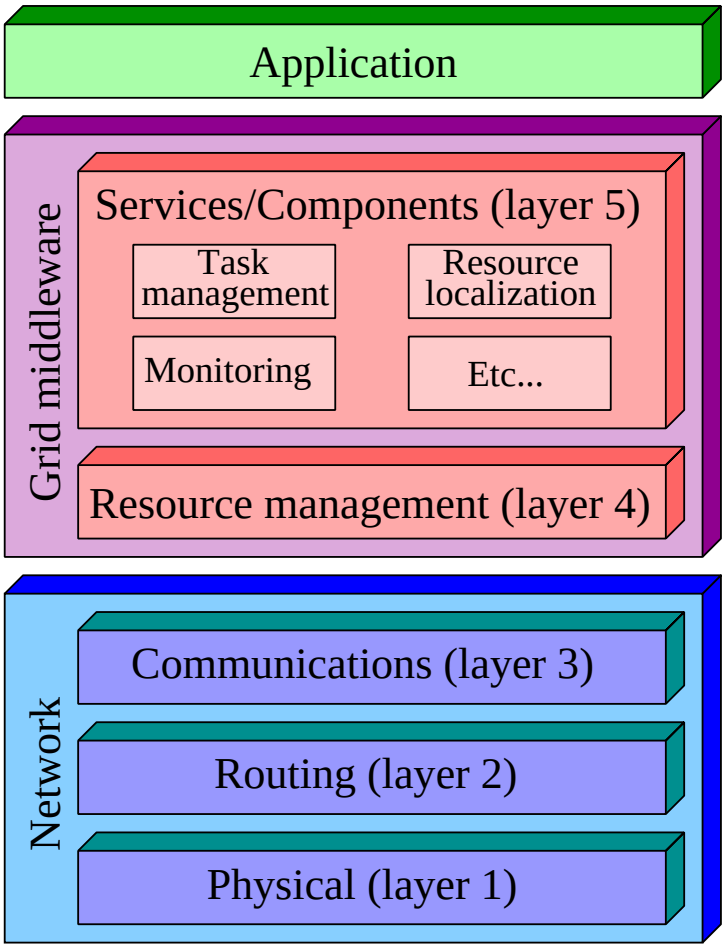


FIGURE 4.12 – Modèle théorique pour le design des applications de type grille

wall). De la même manière, des protocoles comme NAT peuvent limiter les accès aux nœuds. C'est pour cette raison que les liens de communication orientés sont utilisés pour modéliser la couche de routage. Dans la couche 2, le réseau est donc représenté par un graphe $G_2 = (V_2, E_2)$, où $V_2 = V_1$ et E_2 est l'ensemble des chemins entre tout nœud de V_2.

La figure 4.14 est basée sur l'exemple de la figure 4.13 et montre la représentation du réseau dans la couche 2. Pour simplifier, les chemins peuvent démarrer et se terminer sur des composants passifs (nœuds 4, 5, and 7) et ne sont pas affichés. Nous remarquons également les nouvelles liaisons qui représentent les chemins découverts par l'algorithme de routage. Pour les éléments sans fil, (les nœuds 8 et 9), nous remarquons que le lien orienté $(9, 8)$ a été supprimé : le nœud 8 peut contacter le nœud 9 au travers du nœud 7.

La couche 3 - les communications. Au dessus des chemins, il est possible d'envoyer des données entre deux sites distants qui ne sont pas directement connectés. Le réseau est représenté par un graphe $G_3 = G_2$. Dans cette couche, nous pouvons avoir l'envoi et la capacité de réception d'un message en fonction du protocole donné. De nombreux mécanismes peuvent être proposés pour gérer les problèmes de communication (perte ou duplication de message, corruptions de données). Un mécanisme d'acquittement peut assurer qu'un message est envoyé. Si le message est perdu, il est à nouveau expédié. D'autres mécanismes peuvent assurer l'intégrité du message.

La couche 4 - la gestion des ressources. Les 2 couches hautes se focalisent sur l'intergiciel de la grille. La couche 4 représente la couche de gestion des ressources et peut être vue comme une interface

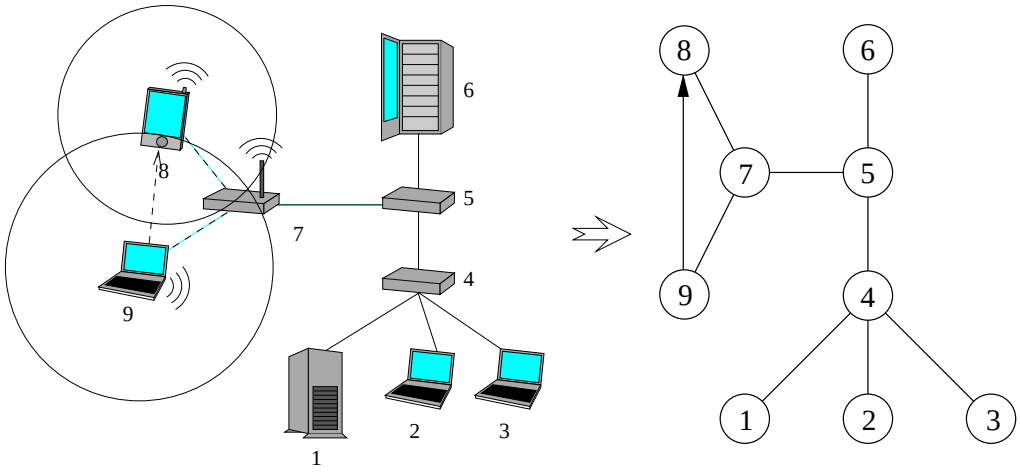


FIGURE 4.13 – Exemple de représentation de la couche 1 du réseau.

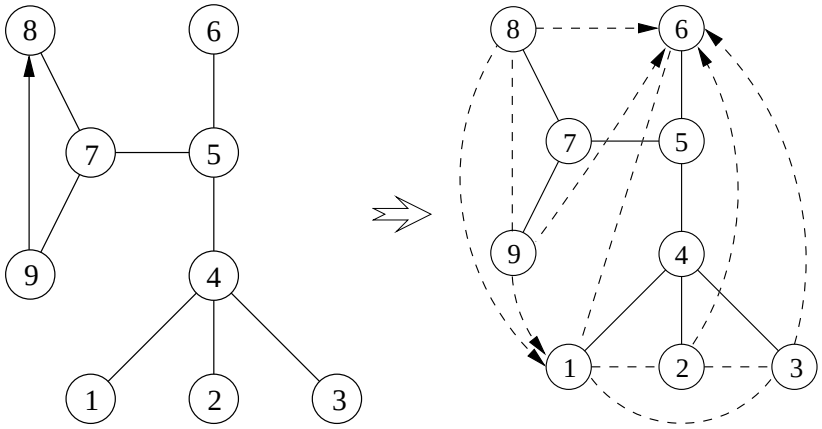


FIGURE 4.14 – Exemple de réseau suivant la représentation de la couche 2.

entre les composants et le services de la grille. Dans cette couche, deux types de nœuds sont distingués. Le premier type est appelé *nœuds actifs*, qui sont à l'intérieur de la grille. Ces nœuds partagent leurs propres ressources ou utilise les ressources de la grille. Les autres types de nœuds, les *nœuds passifs* sont à l'extérieur de la grille tels les routeurs et les switchs.

Dans cette couche, la grille est représenté comme un graphe $G_4 = (V_4, E_4)$ où V_4 est l'ensemble des nœuds actifs et E_4 est l'ensemble des liens de communications entre les liens actifs. Les nœuds passifs ne sont pas représentés dans le graphe (*cf* figure 4.15) mais ils ont une influence l'application de la grille en raison des congestions réseaux qu'ils peuvent produire.

La couche 5 - Les composants et les services. Cette dernière couche correspond aux composants de la grille et à ses services, incluant les taches de gestion et les services de surveillance des ressources. Par exemple, si l'intergiciel correspond à vouloir faire de l'échange de fichiers, il doit y avoir un composant pour l'échange de fichiers et peut-être un composant pour la gestion des droits d'accès et les files d'attende des utilisateurs.

La grille est représentée par un graphe $G_5 = (V_5, E_5)$ où $V_5 = V_4$ et E_5 est l'ensemble des liens de communications proposés par la couche topologie. E_5 n'est pas égal à E_4 : il dépend du protocole que gère la topologie de la grille ou du overlay réseau pair à pair

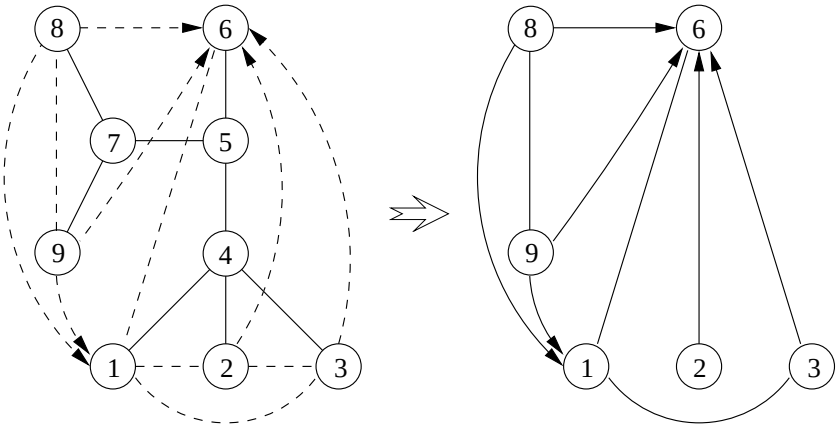


FIGURE 4.15 – Exemple de la représentation de la couche 4.

4.2.6 De la grille à une architecture de sécurité réseau

Des précédentes observations, avec Olivier Flauzac, Cyril Rabat et Luiz-Angelo Steffenel, j'ai proposé dans [FNRS09] une nouvelle architecture et un intergiciel de sécurité dans lequel chaque équipement ou utilisateur est un acteur de la sécurité globale du réseau. Chaque utilisateur peut gérer sa sécurité locale mais aussi, par l'intermédiaire d'échanges avec ses voisins de sa communauté, gérer la politique globale. Pour éviter les utilisateurs malveillants d'attaquer le réseau, chaque échange doit être sécurisé, contrôlé et validé par des utilisateurs autorisés. Un utilisateur autorisé est un utilisateur qui était déjà dans une communauté et qui a échangé des règles de sécurité avec d'autres voisins autorisés. Un équipement autorisé est soit un ordinateur, un servuer ou un équipement réseau qui est considéré être sécurisé et authentifié.

Pour construire une "zone de confiance", les échanges à l'intérieur du réseau doivent être sécurisés pour éviter des actions non autorisées qui peuvent compromettre la sécurité de la communauté. Notre travail, cependant, va au delà d'un simple mécanisme de contrôle de la sécurité comme nous pourrions l'avoir avec des contrôleurs wifi par exemple ou avec des contrôles de l'accès au réseau. Dans le solution des grilles de sécurité, tous les aspects sont pris en compte dans le but de construire une nouvelle architecture de sécurité et un intergiciel pour sécuriser un réseau. Un ordinateur ou un équipement, avec cet intergiciel de sécurité, peut être spécialisé pour contrôler quelques fonctions particulières. Par exemple, un ordinateur peut administrer une base de donnée d'anti-virus, un autre peut administrer l'authentification et un autre les règles du firewall. A partir de la spécification d'un ordinateur, nous pouvons choisir quelle fonction de sécurité l'ordinateur doit offrir. Dans un environnement ad-hoc, les communications peuvent être sécurisées par des méthodes cryptographiques mais cette approche n'est pas suffisante pour des réseaux complexes, où seulement une partie du trafic transite par les VPN ou les tunnels cryptés. De façon similaire, un contrôle dans points d'accès au réseau par des firewall ne fonctionne que sur des réseaux structurés et pas en ad-hoc. Il est difficile de contrôler l'existence d'un partage de connexion internet ou l'ouverture d'un serveur FTP dans un réseau ad-hoc.

Dans cette architecture de sécurité, les équipements doivent mutuellement se surveiller. Si un équipement devient "dangereux" car un virus ou un vers est détecté, il devra être bloqué et supprimé de la communauté (ou zone de confiance), comme illustré à la figure 4.16. Un nouvel utilisateur dans un environnement mobile doit également être autorisé pour accéder à la communauté. De nos jours, des mécanismes similaires peuvent être implémentées par de l'authentification 802.1x ou des liaisons VPN mais ce sont des mécanismes relativement complexe à mettre en place pour l'utilisateur. De plus, ces mécanismes n'assurent pas la sécurité des équipements, d'une façon globale. Dans la solution proposée, un nouvel équipement peut être ajouter dynamiquement, sans intervention humaine, faisant de la sécurité du réseau une fonctionnalité auto-administrée.

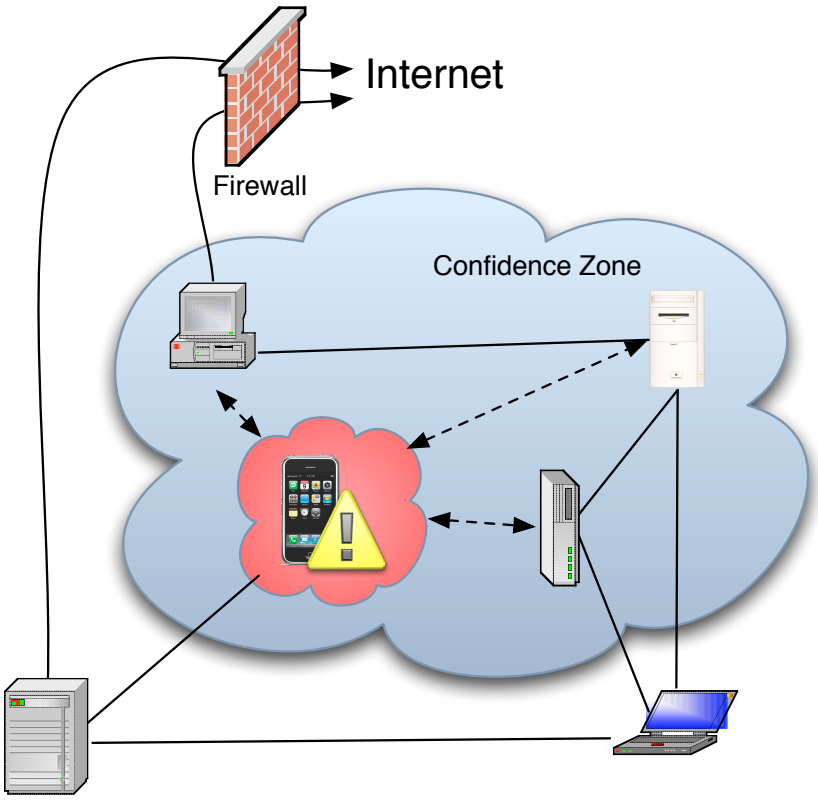


FIGURE 4.16 – Exemple de grille de sécurité

4.2.7 Les problèmes à résoudre

Pour atteindre les objectifs, il est nécessaire de se poser un certains nombres de question : comment distribuer les fonctions de sécurité sur chaque équipement, comment échanger les informations, quel protocole de communication utilisé, comment distinguer les différents types de trafic. Il est également nécessaire de savoir comment sécuriser le trafic vocal, comme bloquer une communication depuis un équipement, ... Je vais tenter d'apporter des réponses à quelques interrogations.

4.2.7.1 Distribution des ressources de sécurité sur les pairs

Depuis notre intergiciel (couche 4), toutes les données échangées entre les pairs dans la même communauté sont considérées comme sûres. Il n'est donc pas nécessaire de les vérifier. Cependant, si un utilisateur souhaite créer une nouvelle communauté, il doit contrôler le trafic entre cette nouvelle communauté et sa propre communauté. Ceci implique un nouveau service qu'il devra installer sur son pair. Par exemple, une approche similaire est utilisée dans les environnements sans fil avec la solution de contrôleur wifi et les protocoles comme LWAPP [CSCW+10] ou CAPWAP [CMS09]. Les données sont échangées entre les clients wifi et le point d'accès puis éventuellement contrôlées soit par le contrôleur, soit par le point d'accès. Avec la grille de sécurité, l'intergiciel fonctionnement de façon similaire. Un pair A peut échanger des informations avec un autre pair B pour utiliser un service de sécurité que A n'a pas mais que B a. A partir de la spécification technique d'un pair, un utilisateur peut donc définir quel service il veut et il pourra le déployer sur son pair.

4.2.7.2 Contrôle des données échangées dans une communauté

Au sein d'une communauté, si un pair reçoit trop de messages sur une période courte en provenance d'un de ses voisins, il peut décider de lancer une analyse de ce voisin. A partir de l'intergiciel de sécurité, un pair peut exécuter une tâche sur un nœud distant depuis la même communauté pour vérifier si quelque chose d'anormal à lieu. Par exemple , un pair peut décider de lancer une analyse anti-virus à distance sur ses voisins.

4.2.7.3 Quand un pair veut rejoindre une communauté existante

Depuis la ressource d'accès au réseau de la couche 4 de notre intergiciel, un contrôle de l'accès au réseau peut être effectué. Dans cette couche, des échanges de type 802.1x ont lieu et dans la couche 5, le service Radius. Quand l'authentification a abouti, les échanges peuvent être effectués avec d'autres gestionnaires de ressources pour contrôler tous les services qui sont présents sur les pairs.

4.2.7.4 Exclusion d'un pair de la communauté

Pour exclure un pair de la communauté car il est devenu non sûr, chaque voisin peut décider de changer ses règles de sécurité afin de bloquer toute communication avec ce pair non sûr.

4.2.8 Exemple de déploiement d'une architecture logique sécurisé sur un réseau physique existant

Dans de nombreux cas, les utilisateurs échangent des données via des liens de communication sécurisés. Mais les solutions actuelles ne sont pas simple à mettre en place. Le bon port de communication doit être ouvert et une demande doit souvent être faite à l'administrateur afin qu'il change les règles du firewall pour autoriser les échanges sur ce port et peut être pour une petite durée. L'approche des grilles de sécurité peut résoudre ce problème. Par exemple, si un utilisateur dans une communauté souhaite créer un VPN depuis son ordinateur A avec un serveur S, l'intergiciel demande si un VPN depuis un ordinateur de cette communauté existe avec ce serveur. Si un ordinateur B a déjà établi cette connexion, une simple connexion entre A et B pourra permettre d'assurer la connexion entre A et S sans aucune modification des règles de sécurité globale. Si aucune connexion VPN vers S existe dans la communauté, le gestionnaire de connexion (dans la couche 4 du modèle) informera le firewall (dans la couche 5) pour ouvrir les bons ports et permettre la connexion VPN entre A et B

Dans le même principe, de nombreux services de sécurité peuvent bénéficier d'une gestion distribuée comme les firewall, les détecteurs d'intrusion, les mises à jour système, le contrôle d'accès au réseau, les VPN, ...

Pour chacun de ces services, les règles sont définies sur des connexions entrantes et/ou sortante. Le gestionnaire de ressource (dans la couche 4) échange les informations entre chaque service de chaque pair. Par exemple, si un utilisatuer A veut ouvrir un port de communication pour un service existant sur son ordinateur, le gestionnaire de ressource devra contacter tous les autres pairs de la communauté et rechercher une règle similaire sur chaque pair. Si cette règle existe dans la communauté, le gestionnaire de ressource de l'ordinateur A informera le service des firewall (dans la couche 5) de l'ordinateur A qu'il peut ouvrir ce port.

4.2.9 Conclusions

L'approche des grilles de sécurité développé dans ce chapitre est un nouveau concept, alliant une architecture suivant le modèle des grilles et des communications basées sur les échanges pair à pair. Ainsi il serait plus aisé de construire des zones dites de confiance, non pas en faisant de la confiance comme dans les architectures de type PKI ou PGP mais plutôt en se basant sur des mécanismes alliant

les échanges d'authentification 802.1x avec les mécanismes de contrôle d'accès au réseaux type NAC. De plus, chaque pair pourrait lancer des vérifications sur les autres pairs afin de détecter d'éventuels problèmes et utiliser les autres pairs pour communiquer avec l'extérieur et ainsi limiter au maximum les points d'entrée dans le réseau. Ce concept va devoir tendre à se développer, surtout avec la multiplication d'équipements mobiles et le passage à IPv6. Tout équipement mobile avec une connexion IPv6 pourra jouer le rôle de passerelle Internet et sera en plus accessible facilement. Ce concept n'intéresse donc pas seulement le monde des réseaux ad-hoc.

4.3 Travaux en cours et perspectives

Les travaux sur la grille de sécurité ont permis de mettre en évidence que la modélisation utilisée, avec l'approche grille, pouvait être améliorée. En effet, ce modèle ne fait de distinction ni sur les ports de communication, ni sur les protocoles utilisés. Si, en entrée d'un réseau, de la redirection de ports est mise en place, la modélisation présentée dans la partie 4.2.5.1 ne permet pas de mettre en évidence ce fonctionnement. C'est pour cette raison que je pense améliorer ce modèle en introduisant l'utilisation de graphe pondéré. Ainsi, le poids de chaque arête pourra modéliser le nombre d'équipements devant être traversé ou ayant un impact sur le transport de l'information. De plus, à moyen terme, j'aimerai pouvoir mettre en application ce concept et développer un démonstrateur, démonstrateur qui pourra être intégrer à l'évolution du projet RemoteLabz : le CloudLabz. En effet, je vais avoir besoin de faire du contrôle de flux dans le projet CloudLabz afin de s'assurer que les utilisateurs n'injectent pas dans le réseau de données compromettantes pour le système.

Sur le projet CloudLabz, au delà de l'aspect technique et de l'usage des technologies existantes comme le protocole L2TP pour faire des liaisons couche 2 au dessus d'une interconnexion couche 3, de nombreux problèmes apparaissent uniquement sur la gestion des utilisateurs et des ressources. En effet, où peut-on stocker les données utilisateurs et où peut-on stocker leurs données d'authentification et leurs autorisations associées. Il n'est pas possible de raisonner de façon centralisé, chaque site doit avoir ses informations. Il faudra donc mettre en place des mécanismes d'échanges d'informations entre les sites, sans pour autant faire de la duplication de données d'authentification. De plus, il faudra pouvoir faire une recherche des ressources disponibles mais aussi des informations plus techniques sur chaque site afin de coordonner les choix des VLANs à utiliser pour le cloisonnement des données entre les sites et entre les ressources.

4.4 Publications majeures

[1] Olivier FLAUZAC and Florent NOLOT and Cyril RABAT and Luiz-Angelo STEFFENEL Grid of Security : a decentralized enforcement of the network security Dans Threats, Countermeasures and Advances in Applied Information Security, IGI Global, ISBN 9781466609785, Avril 2012

[2] Olivier FLAUZAC, Florent NOLOT , Cyril RABAT, Luiz-Angelo STEFFENEL. Grid of security : a new approach of the network security In *NSS 2009. 3rd International Conference on Network and System Security, IEEE Computer Society*, Acceptance rate : 41%, Octobre 2009

Chapitre 5

Perspectives et conclusion

J'ai présenté dans ce mémoire des algorithmes auto-stabilisants dont l'objectif principal est la structuration des réseaux pour minimiser les connaissances nécessaires à l'acheminement de l'information. Dans une première partie, j'ai présenté des algorithmes de construction de cluster à 1 saut puis à k sauts. Toutes ces solutions utilisent comme critère d'élection du clusterhead l'identité du nœud. L'algorithme de clusters à k sauts a également été comparé par simulation avec le meilleur algorithme connu à l'époque et nous avons obtenu une économie de plus de 50% de messages échangés. Nous avons ensuite remis en cause ce critère de l'identité maximale lors de l'adaptation de notre solution de construction de clusters à k sauts pour les réseaux de capteurs. Nous voulions savoir si notre choix était pertinent. Il est connu que l'envoi de messages est fortement consommateur d'énergie et donc le meilleur algorithme reste celui qui envoie le moins de messages pour atteindre la stabilisation. En utilisant toujours le même algorithme, nous avons alors uniquement changé le critère d'élection du cluster-head pour comparer le nombre de messages nécessaire pour atteindre la stabilisation. Les critères d'élection évalués ont alors été : l'identité, le degré, l'énergie résiduelle et la combinaison entre le degré et l'énergie résiduelle. Dans tous les cas, les simulations nous ont montré que le meilleur critère, en terme de nombre de messages échangés, reste le choix de l'identité maximale. Ces travaux ont donné lieu à une publication dans 2 revues d'audience internationale avec comité de sélection et 4 conférences d'audience internationale avec comité de sélection dont 2 Best Paper Award.

Dans une deuxième partie, je me suis intéressé à la diffusion de l'information sur un réseau clusterisé. L'objectif de la construction de clusters était la minimisation des informations à apprendre pour faire de l'acheminement de données au sein du réseau. Je me suis donc orienté vers deux approches : celle de l'arbre couvrant et celle du routage. Ainsi, nous retrouvons les deux techniques utilisées aujourd'hui dans un réseau d'entreprise sur son LAN, à savoir les arbres couvrants sur la couche 2 du modèle OSI et le routage sur la couche 3. Nous avons donc proposé un algorithme de construction d'arbres couvrants basé sur notre solution de construction de clusters à 1 saut. Nous sommes parti pour cela du postulat de ré-utiliser les informations échangées lors de la construction des clusters pour construire l'arbre. Nous avons au final obtenu un algorithme qui construit à la fois un arbre couvrant du réseau mais aussi simultanément les clusters. De plus, nous avons un arbre dans chaque cluster et un arbre qui relie les clusters. Puis nous avons proposé plusieurs stratégies de routage, exploitant à la fois du routage réactif, du routage proactif, du routage hybride et du routage avec ou sans agrégation de données. Ces travaux ont donné lieu à 3 conférences d'audience internationale avec comité de sélection.

Puis dans une troisième et dernière partie, j'ai présenté le projet RemoteLabz et le concept des grilles de sécurité. A la suite de mes travaux sur les clusters et sur l'acheminement de l'information, je me suis intéressé à la problématique des cloisonnements réalisés en entreprise. Les informations traversent différents équipements, sont cloisonnées par des Virtual Area Network, suivent des chemins déterminés par la construction d'arbres puis par des protocoles de routage. De plus certaines données sont interceptées par des équipements de sécurité. Nous avons alors étudié ces échanges d'informations pour arriver

au projet RemoteLabz dont l'objectif est de pouvoir reproduire des architectures d'entreprises, avec un cloisonnement supplémentaire, comme nous pourrions le faire avec les clusters. Chaque architecture est un cluster et la communication entre les clusters ne se fait que par l'outil de gestion du cloisonnement, c'est à dire par le RemoteLabz. Ce projet, financé par plusieurs aides aux transferts technologiques successives, nous a permis de concevoir une solution en mode IaaS de contrôle à la demande d'infrastructures informatiques. De plus, il a permis de mettre en évidence une des lacunes actuelles des réseaux, la surveillance de l'accès au réseau dans le cadre des réseaux mobiles. Dans le RemoteLabz et plus précisément son extension, le CloudLabz, chaque architecture réseau construite est indépendante mais comme il est donné la possibilité à l'utilisateur de connecter sa propre machine virtuelle à l'architecture créée, il est envisageable que l'utilisateur tente de compromettre, depuis sa machine virtuelle, les équipements de l'architecture. L'idée de la conception d'une solution de contrôle d'accès à un réseau est donc née. Nous avons alors commencé par modéliser un réseau avec un approche grille puis défini le concept des grilles de sécurité. Ces travaux ont donné lieu à un chapitre de livre, une conférence d'audience internationale avec comité de sélection, deux aides aux transferts technologiques, une aide à la création d'entreprise au sein de l'incubateur régionale de Champagne-Ardenne et un prix au Concours CreaReims Junior.

Au total, mes activités ont donné lieu à 1 chapitre de livre, 2 revues internationales, 12 conférences internationales dont 2 Best Paper Award et plusieurs aides aux transferts technologiques et récompenses pour le projet RemoteLabz, dans le cadre de l'encadrement de deux doctorats, d'un post-doctorat et plusieurs mémoires de master.

Le travail effectué au cours de ces dernières années m'ont permis de mettre en évidence plusieurs pistes de recherche, autant sur le plan théorique et scientifique que sur le plan technique.

5.1 Évolution des travaux sur la clusterisation et l'acheminement de l'information

5.1.1 Vers une meilleure tolérance à la mobilité

Les auteurs de [DMH13] optent pour un critère de choix du clusterhead basé sur une métrique appelée k densité. Cette densité est calculée grâce au nombre de voisins d'un nœud mais aussi de ses nœuds voisins à distance 2. Comme nous l'avons montré dans [BFM$^+$13a, BFM$^+$13c], cela nécessite un nombre de messages important. Par contre, ce critère semble permettre d'obtenir un réseau plus tolérant aux modifications topologiques. En cas de déplacement d'un nœud, la structure des clusters n'est pas forcément impactée. Par contre, en cas de disparition du clusterhead, de nouveaux clusters doivent être construits. L'objectif est donc maintenant de vérifier par simulation cette stabilité et de concilier notre approche, nécessitant peu de messages et une nouvelle solution pour une meilleure stabilité. L'idée intuitive est de ne plus construire un seul clusterhead mais d'anticiper la disparition du clusterhead en effectuant une élection d'un second clusterhead, pour chaque cluster. La difficulté réside à savoir s'il faut faire une redondance de clusterheads de type actif/passif ou bien actif/actif. C'est à dire, faut-il que les deux cluster-heads jouent un rôle dans l'acheminement des informations et apportent l'avantage d'équilibrer les liaisons utilisées ou bien garde-t-on un clusterhead actif et un autre dans un état passif ?

La problématique principale dans la construction des clusters réside dans leurs utilisations. Quel usage peut-on avoir des clusters et quelles sont les motivations de la clusterisation ? L'intérêt majeur que je porte à la clusterisation est la minimisation des informations à stocker en chaque nœud afin de diminuer les temps de traitement pour la découverte du chemin à emprunter vers une destination quelconque. Un nœud dans un réseau ad-hoc, comme un routeur dans un réseau d'entreprise, ne peut connaître les routes vers toutes les destinations. C'est un point encore plus critique dans les réseaux de capteurs où l'énergie est le critère déterminant dans le choix d'une solution d'acheminement d'informations.

5.1.2 Adaptation aux réseaux de capteurs

[EAA13] fournie une nouvelle classification des algorithmes de clustering et de routage en fonction de leur énergie consommée. De plus, il est connu que l'émission est fortement consommateur d'énergie. A titre d'exemple, des capteurs de type waspmote consomment en moyenne 3 fois plus d'énergie lors de l'émission d'un message sur la plage de fréquences des 868 Mhz. La première idée est donc de minimiser l'émission de messages. L'approche que nous avons utilisée et que j'ai présentée dans ce mémoire est celle de l'agrégation des données. Je souhaite continuer à travailler sur cette problématique, en étudiant plusieurs approches que j'appelle l'agrégation collaborative et l'agrégation déclenchée. La collaboration peut être vue comme étant un questionnement du voisinage quand un nœud souhaite envoyer une information vers le puits et comme je l'ai présenté dans ce mémoire. Dans ce cas, chaque nœud qui devra transmettre cette information, car présent sur la route entre la source et le puits, fera une demande à tous ses voisins afin d'éviter une transmission du voisinage dans un futur proche. Cette approche peut, à mon sens, être améliorée, en déclenchant ces questionnements par le clusterhead, sur tout le cluster puis en faisant la transmission par agrégation successive vers le puits. Le gain devrait être plus important si les chemins vers les puits passent par les clusterheads. C'est donc la combinaison d'un algorithme de routage particulier qui passent par les clusterheads et le déclenchement des interrogations par ces mêmes nœuds qui devraient être favorables à une économie énergétique. Un problème fonctionnel grave peut apparaître en pratique avec ces techniques. C'est la taille des messages générés. En effet, l'agrégation va augmenter la taille du message au fur et à mesure de son transport. De plus, en fonction de l'algorithme de routage utilisé, l'impact peut être conséquent. Dans le cadre d'algorithmes réactifs qui impliquent que la route est présente dans le paquet transmis et comme les adresses MAC sont sur 64 bits, comme actuellement sur les réseaux de capteurs existants, nous obtenons rapidement des paquets de tailles importantes. Sachant que les portées maximales obtenues par expérimentation en milieu urbain sont de 150 mètres en 868 Mhz et des antennes omnidirectionnelles de 4,5 dB, pour couvrir une chaîne de 10 kilomètres, cela implique l'existence d'environ 70 nœuds. Le paquet fera donc déjà plus de 560 octets. A cela, il faut ajouter les data à transporter. Comme la limite est de 1500 octets, il n'est donc pas envisageable d'utiliser tout type d'agrégation et tout type d'algorithme de routage. D'après mes connaissances, cette problématique n'a pas encore été étudiée par la communauté scientifique.

J'aimerai donc explorer cette voie sans dissocier les algorithmes de clustering et d'acheminement de l'information. Comme j'ai pu l'expliquer ci-dessus, il est aisé de se rendre compte que pour acheminer une information, tous les protocoles utilisés ont leur importance. Il serait donc très intéressant d'étudier l'impact global, en terme de messages échangés, d'un mauvais choix dans un des algorithmes de l'association clusterisation et routage. Un autre point très important en pratique réside dans la question de la veille des capteurs. Il existe plusieurs types de veille mais en fonction du type choisi, les messages peuvent ne pas être reçus car le composant qui gère les transmissions est également en veille. Il faut donc s'assurer de ne perdre aucun message et donc de bien être dans un état de veille qui permet de recevoir les messages. Dans la majorité des travaux existants, ce point n'est pas pris en compte. Les auteurs évoquent l'existence de plusieurs modes de veille mais ils oublient le point le plus important : le réveille du capteur n'est pas toujours effectué par un événement de type réception d'un message.

5.2 La structuration des réseaux et la sécurisation des données

5.2.1 L'extension cloud des travaux RemoteLabz

Les travaux sur le projet RemoteLabz m'ont amené à réfléchir plus précisément sur le cloisonnement des informations au sein d'un cloud privé. Lors de l'échange d'informations entre plusieurs sites, comme j'ai pu le montrer dans la partie 4.1 sur la gestion des données, de nombreux problèmes de sécurité peuvent apparaître. Que ce soit sur une compromission du cloisonnement des informations entre des machines virtuelles ou des équipements ou bien sur l'absence de confiance dans les informations

transmises par un utilisateur.

Les extensions vers un "CloudLabz" du projet RemoteLabz pourront nous permettre d'atteindre deux objectifs. Le premier sera la réalisation d'une solution qui permet de rationaliser les coûts en équipements informatiques pour la formation et la recherche, en regroupant virtuellement des équipements physiquement installés dans des sites distants. Ainsi, il sera possible de réserver des ensembles d'équipements informatiques puis de les contrôler à distance. Le principal verrou technologique est dans l'usage de technologies existantes sur une simple liaison Internet. Pour offrir un maximum de souplesse aux utilisateurs, il n'est pas envisageable d'utiliser des liaisons spécifiques entre les sites ou qu'un dialogue soit nécessaire avec un opérateur pour atteindre cet objectif. Les travaux déjà réalisés en utilisant le protocole L2TP ont permis de mettre en évidence que l'objectif est atteignable. Il me reste donc maintenant à modélisation la gestion des utilisateurs entre plusieurs sites distants, la gestion des réservations et des ressources disponibles mais surtout la gestion des cloisonnements dynamiques. Le deuxième objectif sera la possibilité d'offrir à des utilisateurs l'intégration de leur propre ressource au sein d'une architecture. L'offre de ce nouveau service reposera technologiquement sur la même solution que la mise en place de la solution "CloudLabz" car un utilisateur peut très bien être considéré comme un site. La principale difficulté sera de s'assurer de la sûreté des informations qu'ils transmettra à la solution. En réponse à cette nouvelle problématique, le concept des grilles de sécurité pourra alors être exploité. Pour la réalisation technique, je me suis basé sur des technologies utilisées depuis longtemps par les opérateurs, à savoir L2TP. Mais cela n'assure que de la liaison sécurisée couche 2 au-dessus d'une architecture réseau couche 3. Avec l'explosion des travaux sur la virtualisation et maintenant plus particulièrement sur la virtualisation des réseaux [DP13] avec le concept du *Software Defined Network* (SDN), il va devenir possible de fusionner les travaux de gestion dynamique des VLAN comme je le faisais avec le RemoteLabz et les travaux sur les grilles de sécurité. Le réseau devrait pouvoir être géré intégralement de façon logicielle.

5.2.2 L'usage du Software Defined Network

À ma connaissance, aucun démonstrateur n'a été réalisé à ce jour sur l'approche de sécurité distribuée comme présenté dans le paragraphe 4.2. Or il existe maintenant le protocole Openflow qui implémente le concept du SDN. Des recommandations sur son usage dans le cloud ou sur Internet [WH13] et une analyse des aspects sécurités ont été réalisées [Klö13]. Ce protocole peut être utilisé pour développer à la fois le CloudLabz et le concept des grilles de sécurité. En considérant qu'il est possible de gérer le réseau et surtout les flux de données de façon globale et applicative, pourquoi ne pas faire de même sur le réseau Internet ? Le challenge reste notamment la traversée des routeurs opérateurs. Je compte donc maintenant orienter mes recherches sur ce concept et étudier les possibilités actuellement offertes afin de les étendre.

Liste des publications personnelles

— **1 chapitre de livre d'audience internationale**

[1] Olivier FLAUZAC and Florent NOLOT and Cyril RABAT and Luiz-Angelo STEFFENEL Grid of Security : a decentralized enforcement of the network security Dans Threats, Countermeasures and Advances in Applied Information Security, IGI Global, ISBN 9781466609785, Avril 2012

— **3 revues d'audience internationale avec comité de sélection**

[1] Mandicou BA and Olivier FLAUZAC and Rafik MAKHLOUFI and Florent NOLOT and I. NIANG Fault-Tolerant and Energy-Efficient Generic Clustering Protocol for Heterogeneous WSNs International Journal On Advances in Networks and Services, vol. 6, num. 3&4, Dec 2013

[2] Mandicou BA and Olivier FLAUZAC and Rafik MAKHLOUFI and Florent NOLOT and I. NIANG Energy-Aware Self-Stabilizing Distributed Clustering Protocol for Ad Hoc Networks : the case of WSNs. KSII Transactions on Internet and Information Systems , Nov. 2013

[3] T. Bernard, A. Bui, O. Flauzac, and F. Nolot. A multiple random walks based self-stabilizing k-exclusion algorithm in ad hoc networks. *Int. J. Parallel Emerg. Distrib. Syst.*, 25(2) :135–152, April 2010.

— **2 revues d'audience nationale avec comité de sélection**

[1] M. Ba, O. Flauzac, B-S. Haggar, F. Nolot, and I. Niang. Clustering autostabilisant à k sauts dans les réseaux ad hoc. In *URED, Université Recherche et Développement, revue officielle de l'Université Gaston Berger*, 2012.

[2] Florent NOLOT, Vincent VILLAIN. Protocole universel auto-stabilisant de synchronisation d'horloges de phases. In TSI, Technique et Sciences Informatiques, Hermes, 21(5) : 735–756, 2002.

— **1 conférence d'audience internationale invité**

[1] Public IPv6 training provider's testimonials. Future Internet Conference Week, Gant, Belgique, Décembre 2010

— **13 conférences internationales avec comité de sélection dont 3 "Best Paper Award"**

[1] M. Ba, O. Flauzac, B.S Haggar, L. Merghem-Boulahia, F. Nolot, and I. Niang. A Novel Aggregation Approach Based on Cooperative Agents and Self-Stabilizing Clustering for WSNs In *CTRQ 2014, The Sevent International Conference on Communication Theory, Reliability, and Quality of Service*, to appear, Feb. 2014.

[2] N. Matta, R. Rahim-Anoud, L. Boulahia, A. Jrad, and F. Nolot PriBaCC : In-network sensor data processing for efficient smart grid monitoring applications In *Global Information Infrastructure and Networking Symposium*, to appear, 2013, IEEE Computer Sciences. **Best Paper Award**

[3] M. Ba, O. Flauzac, B.S Haggar, F. Nolot, and I. Niang. Self-stabilizing k-hops clustering algorithm for wireless ad hoc networks. In *Proceedings of ICUIMC '13, the 7th International Conference on Ubiquitous Information Management and Communication*, Acceptance rate : 29%, pages 38 :1–38 :10, 2013. ACM. **Best Paper Award**

[4] M. Ba, O. Flauzac, R. Makhloufi, F. Nolot, and I. Niang. Evaluation study of self-stabilizing cluster-head election criteria in wsns. In *CTRQ 2013, The Sixth International Conference on Communication Theory, Reliability, and Quality of Service*, Acceptance rate : 25%, pages 64–69, 2013. **Best Paper Award**

[5] M. Ba, O. Flauzac, R. Makhloufi, F. Nolot, and I. Niang. Comparison between self-stabilizing clustering algorithms in message-passing model. In *Proceedings of ICAS'13 the 9th International Conference on Autonomic and Autonomous Systems*, Acceptance rate : 28%, pages 27–32, 2013.

[6] S. Romaszko, J. Carle, and F. Nolot. Ad hoc routing protocol analysis in civil safety context. In *Ad Hoc Networking Workshop (Med-Hoc-Net), 2010 The 9th IFIP Annual Mediterranean*, pages 1–6, 2010.

[7] Olivier FLAUZAC, Bachar-Salim HAGGAR, Florent NOLOT . Tree and cluster management for MANET In *NOTERE 2010. 10ème Conférence Internationale sur les NOuvelles TEchnologies de la REpartition, ACM Tunisian Chapter et IEEE Computer Science Tunisian Chapter, Acceptance rate : 33%, pages 73–80, Juin 2010*

[8] O. Flauzac, B.S. Haggar, and F. Nolot. Self-stabilizing tree and cluster management for dynamic networks. In *IICS'10*, pages 20–29, 2010.

[9] Olivier FLAUZAC, Florent NOLOT , Cyril RABAT, Luiz-Angelo STEFFENEL. Grid of security : a new approach of the network security In *NSS 2009. 3rd International Conference on Network and System Security, IEEE Computer Society, Acceptance rate : 41%*, Octobre 2009

[10] Olivier FLAUZAC, Bachar-Salim HAGGAR, Florent NOLOT . Self-stabilizing Clustering Algorithm for Ad Hoc Networks *ICWMC 2009. The 5th International Conference on Wireless and Mobile Communications, IEEE Computer Society, Acceptance rate : 31%*, Nice, France, 2009

[11] Thibault BERNARD, Alain BUI, Olivier FLAUZAC, Florent NOLOT. A multiple random walks based self-stabilizing k-exclusion algorithm in ad-hoc networks *ISSADS 2006. 6th IEEE International Symposium and School on Advance Distributed Systems,LNCS* Guadalajara, Mexico, 2006

[12] Florent NOLOT, Vincent VILLAIN. Universal self-stabilizing phase clock protocol with bounded memory. *IPCCC 2001. 20th IEEE International Performance, Computing, and Communications Conference* Phoenix, Arizona, April 2001

[13] Florent NOLOT, Vincent VILLAIN. Limits and Power of the simplest Uniform and Self-Stabilizing Phase Clock Algorithm. In *IPDPS 2000. 14th IEEE International Parallel and Distributed Processing Symposium* Cancun, Mexico, May 2000.

— **2 ouvrages d'audience francophone**

[1] Florent NOLOT Stabilisation des horloges de phases dans les systèmes distribués Au Editions Universitaires Européennes, ISBN 9786131540448, Juillet 2011

[2] Gilles CHAGON and Florent NOLOT. XML. Chez Pearson Education, collection Synthex, ISBN 9782744072369 , Septembre 2007

— **6 conférences d'audience francophone avec comité de sélection**

[1] M. Ba, O. Flauzac, R. Makhloufi, F. Nolot, and I. Niang. Etude comparative entre solutions de clustering auto-stabilisantes à k sauts dans les réseaux ad hoc. In *5ème Colloque national sur la Recherche en Informatique et ses Applications*, CNRIA'13, 2013.

[2] M. Ba, O. Flauzac, B-S. Haggar, F. Nolot, and I. Niang. Clustering auto-stabilisant à k sauts dans les réseaux ad hoc. In *CNRIA'12, 4ème Colloque national sur la Recherche en Informatique et ses Applications*, 2012.

[3] M. Ba, O. Flauzac, B-S. Haggar, F. Nolot, and I. Niang. Clustering auto-stabilisant à k sauts dans les réseaux ad hoc. In *CARI'12, 11e Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées*, pages 420–427, 2012.

[4] M. Ba, O. Flauzac, B.S Haggar, F. Nolot, and I. Niang. Clustering auto-stabilisant a k sauts dans les réseaux ad hoc. In *MajecSTIC 2012*, 2012.

[5] Gautier QUESNEL, Raphael DUBOZ, Florent NOLOT, Eric RAMAT. Comparaison d'approches de simulations distribuées à événements discrets d'entités spatialisées *Majestic 2003. Manisfestation Jeunes Chercheurs en STIC 2003, Marseille, 2003*

[6] Florent NOLOT, Vincent VILLAIN. Protocole universel auto-stabilisant de synchronisation d'horloges de phases à mémoire bornée. *RenPar 2001. 13ième Rencontres francophones du Parallélisme, Paris, Avril 2001*

— **3 communications invités d'audience francophone**

[1] Conception d'une infrastructure de type Réseaux de Campus, Séminaires thématiques pour les entreprises (audience 100 personnes), Lyon, Mars 2013

[2] Usage des QR-Code dans l'enseignement supérieur, Journées thématiques de la CUME (audience 200 personnes), L'ingénierie pédagogique et les nouveaux outils, Paris, Mars 2013

[3] Principe et exploitation des chevaux de Troie, Journées nationales LOGCRI (audience 200 personnes), Paris, Novembre 2012

— **6 communications d'audience francophone**

[1] Le phishing et Trojan, principe et démonstration Journée des informaticiens de l'Université de Reims Champagne-Ardenne (audience 100 personnes), Reims, Juin 2012

[2] Principe de la sécurité informatique Journée des lycéens à l'Université de Reims Champagne-Ardenne, Reims, 2009

[3] La grille régionale GNP : une grille de calcul au service de la résolution de problèmes SAT Perpi'06, Workshop NPPar, Perpignan, France, Octobre 2006

[4] Une approche tolérante aux fautes et la simulation distribuée. LIL. Calais, France, Mars 2003

[5] Protocole universel auto-stabilisant de synchronisation d'horloges de phases à mémoire bornée. LaRIA. Amiens, France, Mai 2001

[6] Limites et performances du plus simple algorithme uniforme et auto-stabilisant de synchronisation d'horloges de phases. LRIA. Paris, France, 04/2000

— **Transferts technologiques et prix**

[1] Ahmet DEMIR, Florent NOLOT Projet RemoteLabz : solution de contrôle d'équipements informatique pour la réalisation de travaux pratiques, à distance, depuis un navigateur Web Incubation au sein de l'incubateur régionale de Champagne-Ardenne, Décembre 2009 à Juin 2012

[2] Ahmet DEMIR, Florent NOLOT. Réalisation de travaux pratiques, à distance, depuis un navigateur Web. Aide au transfert technologique OSEO, 2010

[3] Ahmet DEMIR, Florent NOLOT. Business Plan - Salle de travaux pratiques informatiques en ligne. Lauréat du Concours CreaReims Junior, 2010

[4] Matthieu MACHET, Florent NOLOT. Développement d'une salle virtuelle de travaux pratiques informatique en ligne - RemoteLabz. Aide aux Jeunes pour l'Innovation, 2009

[5] Guillaume NEUVENS, Florent NOLOT. Développement d'une solution de vidéo conférence de salon pour particulier. Sélectionné aux "Carrefours des possibles", Salon Innovact 2009

— **2 rapports de recherche**

[1] Florent NOLOT, Vincent VILLAIN. Universal self-stabilizing phase clock protocol with bounded memory. Rapport Technique LaRIA 01-02, Laboratoire de Recherche en Informatique d'Amiens, Mars 2001

[2] Florent NOLOT, Vincent VILLAIN. Limits and Power of the simplest Uniform and Self-Stabilizing Phase Clock Algorithm. Rapport Technique LaRIA 99-14, Laboratoire de Recherche en Informatique d'Amiens, Octobre 1999

— **Thèse de doctorat**

[1] Florent NOLOT Stabilisation des horloges de phases dans les systèmes distribués, PhD Thesis Université de Picardie Jules Verne, 31 octobre 2002

Bibliographie

[ACF+01] W. Allcock, A. Chervenak, I. Foster, L. Pearlman, V. Welch, and M. Wilde. Globus toolkit support for distributed data-intensive science. In *in International Conference on Computing in High Energy and Nuclear Physics (CHEP'01)*. IEEE Press, September 2001.

[ACK+02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home : an experiment in public-resource computing. *Communications of the ACM*, 45(11) :56–61, November 2002.

[AJRA08] S. Adabi, S. Jabbehdari, A.M. Rahmani, and S. Adabi. Sbca : Score based clustering algorithm for mobile ad-hoc networks. In *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, pages 427–431, 2008.

[AK98a] A. Arora and S. S. Kulkarni. Designing masking fault-tolerance via nonmasking fault-tolerance. *IEEE Transactions on Software Engineering*, 24(6), 1998.

[AK98b] A. Arora and S. S. Kulkarni. Detectors and correctors : A theory of fault-tolerance components. In *invited to IEEE Transactions on Computers*, 1998.

[AM09] Ratish Agarwal and Mahesh Motwani. Survey of clustering algorithms for manet. *CoRR*, abs/0912.2303, 2009.

[AMC07] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury. A survey on wireless multimedia sensor networks. *Comput. Netw.*, 51(4) :921–960, March 2007.

[AN06] L. M.C. Arboleda and N. Nasser. Comparison of clustering algorithms and protocols for wireless sensor networks. In *Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on*, pages 1787–1792, 2006.

[And04] David P. Anderson. BOINC : A System for Public-Resource Computing and Storage. In *GRID '04 : Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[AP01] B. An and S. Papavassiliou. A mobility-based clustering approach to support mobility management and multicast routing in mobile ad-hoc wireless networks. *Int. J. Netw. Manag.*, 11(6) :387–395, November 2001.

[APVH00] A.D. Amis, R. Prakash, T.H.P. Vuong, and D.T. Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 32–41 vol.1, 2000.

[ASJ09] A. Akbari, M. Soruri, and S. V. Jalali. Survey of stable clustering for mobile ad hoc networks. In *Machine Vision, 2009. ICMV '09. Second International Conference on*, pages 3–7, 2009.

[AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing : Fundamentals, Simulations and Advanced Topics (2nd edition)*. John Wiley Interscience, March 2004.

[AWD04] Mehran Abolhasan, Tadeusz Wysocki, and Eryk Dutkiewicz. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1) :1 – 22, 2004.

[Awe85] B. Awerbuch. Complexity of network synchronization. *Journal of the Association of the Computing Machinery*, 4(32) :804–823, 1985.

[AY07] A. A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14 15) :2826 – 2841, 2007. Network Coverage and Routing Schemes for Wireless Sensor Networks.

[BAKAO13] M. Bsoul, A. Al-Khasawneh, A. E. Abdallah, and I. Obeidat. An energy-efficient threshold-based clustering protocol for wireless sensor networks. *Wireless Personal Communications*, 70(1) :99–112, 2013.

[BAR07] M.R. Brust, A. Andronache, and S. Rothkugel. Waca : A hierarchical weighted clustering algorithm optimized for mobile hybrid networks. In *Wireless and Mobile Communications, 2007. ICWMC '07. Third International Conference on*, pages 23–23, 2007.

[Bas99] S. Basagni. Distributed clustering for ad hoc networks. In *Parallel Architectures, Algorithms, and Networks, 1999. (I-SPAN '99) Proceedings. Fourth International Symposium on*, pages 310–315, 1999.

[BBL02] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *Softw. Pract. Exper.*, 32(15) :1437–1466, December 2002.

[BC03] S. Bandyopadhyay and E.J. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1713–1723 vol.3, 2003.

[BFH+12a] M. Ba, O. Flauzac, B-S. Haggar, F. Nolot, and I. Niang. Clustering auto-stabilisant à k sauts dans les réseaux ad hoc. In *4ème Colloque national sur la Recherche en Informatique et ses Applications*, CNRIA'12, 2012.

[BFH+12b] M. Ba, O. Flauzac, B-S. Haggar, F. Nolot, and I. Niang. Clustering auto-stabilisant à k sauts dans les réseaux ad hoc. In *11e Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées*, CARI'12, pages 420–427, 2012.

[BFH+12c] M. Ba, O. Flauzac, B-S. Haggar, F. Nolot, and I. Niang. Clustering autostabilisant à k sauts dans les réseaux ad hoc. In *Université Recherche et Développement, revue officielle de l'Université Gaston Berger*, 2012.

[BFH+12d] M. Ba, O. Flauzac, B.S Haggar, F. Nolot, and I. Niang. Clustering auto-stabilisant a k sauts dans les réseaux ad hoc. In *MajecSTIC 2012*, 2012.

[BFH+13] M. Ba, O. Flauzac, B.S Haggar, F. Nolot, and I. Niang. Self-stabilizing k-hops clustering algorithm for wireless ad hoc networks. In *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '13, pages 38 :1–38 :10, New York, NY, USA, 2013. ACM.

[BFM+13a] M. Ba, O. Flauzac, R. Makhloufi, F. Nolot, and I. Niang. Comparison between self-stabilizing clustering algorithms in message-passing model. In *Proceedings of the 9th International Conference on Autonomic and Autonomous Systems*, ICAS '13, pages 27–32, 2013.

[BFM+13b] M. Ba, O. Flauzac, R. Makhloufi, F. Nolot, and I. Niang. Energy-aware self-stabilizing distributed clustering protocol for ad hoc networks : the case of wsns. *KSII Transactions on Internet and Information Systems*, page To appear, 2013.

[BFM+13c] M. Ba, O. Flauzac, R. Makhloufi, F. Nolot, and I. Niang. Etude comparative entre solutions de clustering auto-stabilisantes à k sauts dans les réseaux ad hoc. In *5ème Colloque national sur la Recherche en Informatique et ses Applications*, CNRIA'13, 2013.

[BFM+13d] M. Ba, O. Flauzac, R. Makhloufi, F. Nolot, and I. Niang. Evaluation study of self-stabilizing cluster-head election criteria in wsns. In *CTRQ 2013, The Sixth International Conference on Communication Theory, Reliability, and Quality of Service*, pages 64–69, 2013.

[BKH+08] T. Bullot, R. Khatoun, L. Hugues, D. Gaïti, and L. Merghem-Boulahia. A situatedness-based knowledge plane for autonomic networking. *International Journal of Network Management*, 18(2) :171–193, 2008.

[BKL01] P. Basu, N. Khan, and T.D C Little. A mobility based metric for clustering in mobile ad hoc networks. In *Distributed Computing Systems Workshop, 2001 International Conference on*, pages 413–418, 2001.

[BLM+10] O. Boyinbode, H. Le, A. Mbogho, M. Takizawa, and R. Poliah. A survey on clustering algorithms for wireless sensor networks. In *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, pages 358–364, 2010.

[BT85] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the Association of the Computing Machinery*, 32 :824–840, 1985.

[CD06] Eddy Caron and Frédéric Desprez. Diet : A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3) :335–352, 2006.

[CDDL10] E. Caron, A.K. Datta, B. Depardon, and L.L. Larmore. A self-stabilizing k-clustering algorithm for weighted graphs. *J. Parallel Distrib. Comput.*, 70(11) :1159–1173, November 2010.

[CDF+04] Franck Cappello, Samir Djilali, Gilles Fedak, Thomas Herault, Frédéric Magniette, Vincent Néri, and Oleg Lodygensky. Computing on Large Scale Distributed Systems : XtremWeb Architecture, Programming Models, Security, Tests and Convergence with Grid. In *FGCS Future Generation Computer Science*, volume 21, pages 417–437, March 2004.

[CDT01] M. Chatterjee, S. K. Das, and D. Turgut. Wca : A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks*, 5 :193–204, 2001.

[Cen] Fast Lane Data Center. Fast lane data center. http://www.fastlaneus.com/remotelabs.

[CF08] Claude Castellucia and Aurélien Francillon. Protéger les réseaux de capteurs sans fil. In *SSTIC08*, 2008.

[CKS10] J. Chen, C-S Kim, and F. Song. A distributed clustering algorithm for voronoi cell-based large scale wireless sensor network. In *Communications and Mobile Computing (CMC), 2010 International Conference on*, volume 3, pages 209–213, 2010.

[CM99] S. Corson and J. Macker. Mobile Ad hoc Networking (MANET) : Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501 (Informational), January 1999.

[CMS09] P. Calhoun, M. Montemurro, and D. Stanley. Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification, 2009.

[CNGS02] Geng Chen, F. Garcia Nocetti, J.S. Gonzalez, and I. Stojmenović. Connectivity based k-hop clustering in wireless networks. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 2450–2459, 2002.

[CS07] Claude Castelluccia and Angelo Spognardi. Rok : A robust key pre-distribution protocol for multi-stage wireless sensor networks. In *IEEE Securecomm*, September 2007.

[CSCW+10] Pat R. Calhoun, Rohit Suri, Nancy Cam-Winget, Scott Kelly, Michael Glenn Williams, Sue Hares, and Bob O'Hara. Lightweight Access Point Protocol, February 2010.

[CT91] T.D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. In *PODC91 Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 325–340, 1991.

[CW06] Wonchang Choi and Miae Woo. A distributed weighted clustering algorithm for mobile ad hoc networks. In *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pages 73–73, 2006.

[CYH91] Nian-Shing Chen, Hwey-Pyng Yu, and Shing-Tsaan Huang. A self-stabilizing algorithm for constructing spanning trees. *Information Processing Letters*, 39(3) :147 – 151, 1991.

[DDL09] A.K. Datta, S. Devismes, and L.L. Larmore. A self-stabilizing o(n)-round k-clustering algorithm. In *Reliable Distributed Systems, 2009. SRDS '09. 28th IEEE International Symposium on*, pages 147–155, 2009.

[DDS87] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the Association of the Computing Machinery*, 34 :77–97, 1987.

[DH99] S. Dolev and T. Herman. Parallel composition of stabilizing algorithms. In *Proceedings of the Fourth Workshop on Self-Stabilizing Systems*, pages 25–32. IEEE Computer Society Press, 1999.

[DHR+11] S. Devismes, K. Heurtefeux, Y. Rivierre, A.K. Datta, and L.L. Larmore. Self-stabilizing small k-dominating sets. In *Networking and Computing (ICNC), 2011 Second International Conference on*, pages 30–39, 2011.

[Dij71] Edsger W. Dijkstra. A short introduction to the art of programming. circulated privately, August 1971.

[Dij74] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17(11) :643–644, 1974.

[DLD+12] A.K. Datta, L.L. Larmore, S. Devismes, K. Heurtefeux, and Y. Rivierre. Competitive self-stabilizing k-clustering. In *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*, pages 476–485, 2012.

[DMH13] T. Ducrocq, N. Mitton, and M. Hauspie. Energy-based clustering for wireless sensor network lifetime optimization. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 968–973, 2013.

[DN10a] A. Demir and F. Nolot. Business plan - salle de travaux pratiques informatiques en ligne, 2010.

[DN10b] A. Demir and F. Nolot. Réalisation de travaux pratiques, à distance, depuis un navigateur web, 2010.

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

[DP13] Otto Carlos M.B. Duarte and Guy Pujolle. *Virtual Networks*. John Wiley & Sons, Inc., 2013.

[EAA13] M. El-Aaasser and M. Ashour. Energy aware classification for wireless sensor networks routing. In *Advanced Communication Technology (ICACT), 2013 15th International Conference on*, pages 66–71, 2013.

[EE07] Khaled Elgoarany and Mohamed Eltoweissy. Security in mobile ipv6 : A survey. *Information Security Technical Report*, 12(1) :32–43, 2007.

[*07]* Leskovec J. *et al.* SNAP : Stanford Network Analysis Platform, 2007. http ://snap.stanford.edu/index.html.

[EV06] Arnaud Ebalard and Guillaume Valadon. La sécurité dans mobile ipv6. In *SSTIC06*, 2006.

[EWB87] A. Ephremides, J.E. Wieselthier, and D.J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE*, 75(1) :56–73, January 1987.

[FH03] J. Faruque and A. Helmy. Gradient-based routing in sensor networks. *Mobile Computing and Communications Review*, 2, 2003.

[FHN09] O. Flauzac, B. S. Haggar, and F. Nolot. Self-stabilizing clustering algorithm for ad hoc networks. In *Wireless and Mobile Communications, 2009. ICWMC '09. Fifth International Conference on*, pages 24–29, 2009.

[FHN10a] O. Flauzac, B.S. Haggar, and F. Nolot. Self-stabilizing tree and cluster management for dynamic networks. In *IICS'10*, pages 20–29, 2010.

[FHN10b] O. Flauzac, B.S. Haggar, and F. Nolot. Tree and cluster management for manet. In *New Technologies of Distributed Systems (NOTERE), 2010 10th Annual International Conference on*, pages 73–80, 2010.

[FK97] I. Foster and C. Kesselman. Globus : a metacomputing infrastructure toolkit. In IEEE Press, editor, *Supercomputer Applications*, volume 11 (2), pages 115–128, 1997.

[FKF03] O. Flauzac, M. Krajecki, and J. Fugère. CONFIIT : a middleware for peer to peer computing. In C. Tan M. Gravilova and P. L'Ecuyer, editors, *The 2003 International Conference on Computational Science and its Applications (ICCSA 2003)*, volume 2669 (III) of Lecture Notes in Computer Science, pages 69–78, Montréal, Québec, June 2003. Springer-Verlag.

[FLP85] M.J. Fisher, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the Association of the Computing Machinery*, 32 :374–382, 1985.

[FNRS09] O. Flauzac, F. Nolot, C. Rabat, and L-A Steffenel. Grid of security : A new approach of the network security. In *NSS '09 : Proceedings of the 2009 Third International Conference on Network and System Security*, pages 67–72. IEEE Computer Society, 2009.

[FNRS12] O. Flauzac, F. Nolot, C. Rabat, and L-A Steffenel. *Threats, Countermeasures, and Advances in Applied Information Security*, chapter Grid of Security : a decentralized enforcement of the network security, pages pp. 426–443. IGI Global, 2012.

[Fre] Freenet. http ://www.freenet.sourceforge.net.

[Gri] Grid5000. Grid5000. http://www.grid5000.fr.

[Gro] Netdev Group. Netlab. http://www.netdevgroup.com.

[GT95] M. Gerla and J. T-C. Tsai. Multicluster, mobile, multimedia radio network. *Wirel. Netw.*, 1(3) :255–265, August 1995.

[Gui12] B. Guizani. *Algorithme de clusterisation et protocoles de routage dans les réseaux ad hoc*. Thèse de doctorat, Université de Technologie de Belfort-Montbéliard, Avril 2012.

[gW] Working group WG802.1. Ieee standard for local and metropolitan area networks–media access control (mac) bridges and virtual bridged local area networks. *IEEE Std 802.1Q-2011 (Revision of IEEE Std 802.1Q-2005)*.

[HB08] Kim HyunGon and Oh ByeongKyun. Secure and low latency handoff scheme for proxy mobile ipv6. In *Mobility '08 : Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, pages 1–9, New York, NY, USA, 2008. ACM.

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

[HCB00] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, page 10 pp. vol.2, 2000.

[Her91] T. Herman. Adaptativity through distributed convergence. *PhD thesis*, 1991.

[HLS05] J.C. Hou, N. Li, and I. Stojmenović. *Topology Construction and Maintenance in Wireless Sensor Networks*, pages 311–341. John Wiley & Sons, Inc., 2005.

[HZZW12] M. Hai, S. Zhang, L. Zhu, and Y. Wang. A survey of distributed clustering algorithms. In *Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on*, pages 1142–1145, 2012.

[Jai91] R.K. Jain. *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1 edition, 1991.

[JN06] C. Johnen and L. H. Nguyen. Self-stabilizing weight-based clustering algorithm for ad hoc sensor networks. In E. Sotiris Nikoletseas and José D.P. Rolim, editors, *Algorithmic Aspects of Wireless Sensor Networks*, volume 4240 of *Lecture Notes in Computer Science*, pages 83–94. Springer Berlin Heidelberg, 2006.

[JN08] C. Johnen and L. H. Nguyen. Self-stabilizing construction of bounded size clusters. In *Parallel and Distributed Processing with Applications, 2008. ISPA '08. International Symposium on*, pages 43–50, 2008.

[JN09] C. Johnen and L. H. Nguyen. Robust self-stabilizing weight-based clustering algorithm. *Theoretical Computer Science*, 410(6-7) :581 – 594, 2009. Principles of Distributed Systems.

[KA97a] S.S. Kulkarni and A. Arora. Fine-grain multitolerant barrier synchronization. Technical report, Ohio State University, 1997.

[KA97b] S.S. Kulkarni and A. Arora. Multitolerant barrier synchronization. *Information Processing Letters*, 64 :29–36, 1997.

[KA98] S.S. Kulkarni and A. Arora. Multitolerant distributed reset. *special Issue on Self-stabilization of Chicago Journal of Theoretical Computer Science*, 4, 1998.

[KBC+00] John Kubiatowicz, Divid Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore : An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM Press, November 2000.

[KF] Michaël Krajecki and Olivier Flauzac. Brevet numéro 0308501 - système de gestion distribuée de ressources informatiques et de données.

[KFM04] Michaël Krajecki, Olivier Flauzac, and Pierre-Paul Mérel. Focus on the communication scheme in the middleware CONFIIT using XML-RPC. In *International Workshop on Java for Parallel Distributed Computing (IW-JPDC'04)*, volume 6, page 160b, Santa Fe, New Mexico, USA, April 2004. IEEE Computer Society.

[Klö13] R. Klöti. *OpenFlow : A Security Analysis*. Master thesis, Swiss Federal Institute of Technology Zurich and Institut für Technische Informatik und Kommunikationsnetze, April 2013.

[Kle61] L. Kleinrock. *Information Flow in Large Communication Nets*. Ph.d. thesis proposal, Massachusetts Institute of Technology, May 1961.

[Kle69] L. Kleinrock. UCLA to be first station in nationwide computer network. UCLA Press Release, July 1969.

[Lab] Proctor Labs. Proctor labs. http://proctorlabs.com.

[LCWG97] W. Liu, C. Chiang, H. Wu, and C. Gerla. Routing in Clustered Multihop Mobile Wireless Networks with Fading Channel. In *Proc. IEEE SICON'97*, pages 197–211, April 1997.

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

[LG12] M. Lehsaini and M. Feham H. Guyennet. Efficient cluster-based fault-tolerant schemes for wireless sensor networks. In *New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on*, pages 1–5, 2012.

[LGF08a] M. Lehsaini, H. Guyennet, and M. Feham. Ces : Cluster-based energy-efficient scheme for mobile wireless sensor networks. In Ali Miri, editor, *Wireless Sensor and Actor Networks II*, volume 264 of *IFIP - The International Federation for Information Processing*, pages 13–24. Springer US, 2008.

[LGF08b] M. Lehsaini, H. Guyennet, and M. Feham. A novel cluster-based self-organization algorithm for wireless sensor networks. In *Collaborative Technologies and Systems, 2008. CTS 2008. International Symposium on*, pages 19–26, 2008.

[LGF10] M. Lehsaini, H. Guyennet, and M. Feham. An efficient cluster-based self-organisation algorithm for wireless sensor networks. *Int. J. Sen. Netw.*, 7(1/2) :85–94, February 2010.

[LHL03] N. Li, J.C. Hou, and S. Lui. Design and analysis of an mst-based topology control algorithm. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1702–1712 vol.3, 2003.

[LP09] T. P. Lambrou and C. G. Panayiotou. A survey on routing techniques supporting mobility in sensor networks. In *Mobile Ad-hoc and Sensor Networks, 2009. MSN '09. 5th International Conference on*, pages 78–85, 2009.

[LPP04] Sebastien Lacour, Christian Perez, and Thierry Priol. A Network Topology Description Model for Grid Application Deployment. In *GRID '04 : Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 61–68, Washington, DC, USA, 2004. IEEE Computer Society.

[LTCH06] J. J. Y. Leu, M-H Tsai, T-C Chiang, and Y-M Huang. Adaptive power-aware clustering and multicasting protocol for mobile ad hoc networks. In Jianhua Ma, Hai Jin, LaurenceT. Yang, and JeffreyJ.-P. Tsai, editors, *Ubiquitous Intelligence and Computing*, volume 4159 of *Lecture Notes in Computer Science*, pages 331–340. Springer Berlin Heidelberg, 2006.

[LZ99] Zhou Lidong and J. Haas Zygmunt. Securing ad hoc networks. In *IEEE Network*, volume 13, pages 24–30. IEEE, 1999.

[LZS08] J. Li, P. Zhang, and S. Sampalli. Improved security mechanism for mobile ipv6. *International Journal of Network Security*, 6(3) :291–300, May 2008.

[MBF04] N. Mitton, A. Busson, and E. Fleury. Self-organization in large scale ad hoc networks. In *Mediterranean ad hoc Networking Workshop (MedHocNet'04).*, page 0000, Bodrum, Turquie, June 2004.

[MDBG13] R. Makhloufi, G. Doyen, G. Bonnet, and D. Gaïti. A survey and performance evaluation of decentralized aggregation schemes for autonomic management. In *Wiley International Journal of Network Management (IJNM)*, 2013.

[MFLT05] N. Mitton, E. Fleury, I.G. Lassous, and S. Tixeuil. Self-stabilization in self-organized multihop wireless networks. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pages 909–915, 2005.

[MN09] M. Machet and F. Nolot. Développement d'une salle virtuelle de travaux pratiques informatique en ligne - remotelabz., 2009.

[Moy98] J. Moy. OSPF Version 2. RFC 2328, 1998.

[NQL11] M. Nasim, S. Qaisar, and S. Lee. An energy efficient cooperative hierarchical mimo clustering scheme for wireless sensor networks. *Sensors*, 12(1) :92–114, 2011.

[PRD03] C. Perkins, E. Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Informational), 2003.

[PYY+04] Yi Ping, Yao Yan, Hou Yafei, Zhong Yiping, and Zhang Shiyong. Securing ad hoc networks through mobile agent. In *InfoSecu '04 ; Proceedings of the 3rd internatianal conference on Information security*. ACM, 2004.

[Raj02] Rajmohan Rajaraman. Topology control and routing in ad hoc networks : a survey. *SIGACT News*, 33(2) :60–73, June 2002.

[Ray85] M. Raynal. *Algorithmes distribués et protocoles*. Eyrolles, 1985.

[Ray91] M. Raynal. *La communication et le temps dans les réseaux et les systèmes répartis*. Eyrolles, 1991.

[Ray92] M. Raynal. *Synchronisation et état global dans les systèmes répartis*. Eyrolles, 1992.

[RBF06] Cyril Rabat, Alain Bui, and Olivier Flauzac. A random walk topology management solution for grid. In *I2CS*, volume 3908 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2006.

[RCN10] S. Romaszko, J. Carle, and F. Nolot. Ad hoc routing protocol analysis in civil safety context. In *Ad Hoc Networking Workshop (Med-Hoc-Net), 2010 The 9th IFIP Annual Mediterranean*, pages 1–6, 2010.

[RLH06] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Informational), January 2006.

[SFH+12] P. Schaffer, K. Farkas, A. Horváth, T. Holczer, and L. Buttyán. Survey secure and reliable clustering in wireless sensor networks : A critical survey. *Comput. Netw.*, 56(11) :2726–2741, July 2012.

[SG85] R. Sinha and S.C. Gupta. Mobile packet radio networks : State-of-the-art. *Communications Magazine, IEEE*, 23(3) :53–61, 1985.

[SHN11] I. G. Shayeb, A. H. Hussein, and A. B. Nasoura. A survey of clustering schemes for mobile ad-hoc network (manet). *American Journal of Scientific Research*, 20(2011) :135–151, 2011.

[SRAMBG10] A. Sardouk, R. Rahim-Amoud, L. Merghem-Boulahia, and D. Gaiti. Agent strategy data gathering for long life wsn. volume 2, pages 71–90, 2010.

[SSN+13] D. Savage, D. Slice, J. Ng, S. Moore, and R. White. Enhanced Interior Gateway Routing Protocol. Draft Version 1, October 2013.

[Tel94a] G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, 1994.

[Tel94b] G. Tel. Networks orientation. *International Journal of Foundations of Computer Science*, 5(1) :23–57, 1994.

[TVR+99] W. Townley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter. Layer Two Tunneling Protocol "L2TP". RFC 2661, 1999.

[VAK12] S. Vodopivec, J. Bester A., and Kos. A survey on clustering algorithms for vehicular ad-hoc networks. In *Telecommunications and Signal Processing (TSP), 2012 35th International Conference on*, pages 52–56, 2012.

[Val08] Guillaume Valadon. *Mobile IPv6 : architectures et protocoles*. Thèse de doctorat, Université Pierre et Marie Curie, Juin 2008.

[Var97] G. Varghese. Compositional proofs of self-stabilizing protocols. *Proceedings of the Third Workshop on Self-Stabilizing Systems*, pages 80–94, 1997.

[VH08] András Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *Simutools*, pages 60 :1–60 :10, 2008.

[VV05] T. Velte and A. Velte. *Cisco 802.11 Wireless Networking Quick Reference*. Cisco Press, 2005.

[WC06] D. Wei and H.A. Chan. Clustering ad hoc networks : Schemes and classifications. In *Sensor and Ad Hoc Communications and Networks, 2006. SECON '06. 2006 3rd Annual IEEE Communications Society on*, volume 3, pages 920–926, 2006.

[WH13] M. Wasserman and S. Hartman. Security analysis of the open networking foundation (onf) openflow switch specification. Internet-Draft, 2013.

[WYGZ11] C. Wei, J. Yang, Y. Gao, and Z. Zhang. Cluster-based routing protocols in wireless sensor networks : A survey. In *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, volume 3, pages 1659–1663, 2011.

[YC03] J.Y. Yu and P. H J Chong. 3hbac (3-hop between adjacent clusterheads) : a novel non-overlapping clustering algorithm for mobile ad hoc networks. In *Communications, Computers and signal Processing, 2003. PACRIM. 2003 IEEE Pacific Rim Conference on*, volume 1, pages 318–321 vol.1, 2003.

[YC05] J.Y. Yu and P. H J. Chong. A survey of clustering schemes for mobile ad hoc networks. *Communications Surveys Tutorials, IEEE*, 7(1) :32–48, 2005.

[Zha04] W. Zhang. Handover decision using fuzzy madm in heterogeneous networks. In *WCNC*, pages 653–658, 2004.