Vers une structuration auto-stabilisante des réseaux ad hoc : cas des réseaux de capteurs sans fil

Mandicou Ba

École Doctorale Sciences Technologies et Santé

Thèse

présentée par **Mandicou BA**

pour l'obtention du grade de

Docteur de l'Université de Reims Champagne-Ardenne

Spécialité : Informatique

Vers une structuration auto-stabilisante des réseaux ad hoc: cas des réseaux de capteurs sans fil

Soutenue publiquement le Mercredi 21 Mai 2014 devant le jury composé de :

Président	Jean-Frédéric Myoupo	Professeur des Universités, Université Picardie Jules Verne, France.
Rapporteurs	Mohamed Mosbah	Professeur des Universités, Institut Polytechnique de Bordeaux, France.
	Maria Potop-Butucaru	Professeur des Universités, Université Pierre et Marie Curie, France.
	Eddy Caron	Maître de Conférences HDR École Normale Supérieure de Lyon, France.
Examinateurs	Thomas Noël	Professeur des Universités, Université de Strasbourg, France.
	Ibrahima Niang	Maître de Conférences HDR, Université Cheikh Anta Diop, Sénégal.
Directeur	Olivier Flauzac	Professeur des Universités, Université de Reims Champagne-Ardenne, France.
Co-directeur	Florent Nolot	Maître de Conférences HDR, Université de Reims Champagne-Ardenne, France.

Centre de Recherche en STIC (CReSTIC) - Équipe Systèmes Communicants (SysCom)

École Doctorale Sciences Technologies et Santé

Thèse

présentée par **Mandicou BA**

pour l'obtention du grade de

Docteur de l'Université de Reims Champagne-Ardenne

Spécialité : Informatique

Vers une structuration auto-stabilisante des réseaux ad hoc: cas des réseaux de capteurs sans fil

Soutenue publiquement le Mercredi 21 Mai 2014 devant le jury composé de :

Président	Jean-Frédéric Myoupo	Professeur des Universités, Université Picardie Jules Verne, France.
Rapporteurs	Mohamed Mosbah	Professeur des Universités, Institut Polytechnique de Bordeaux, France.
	Maria Potop-Butucaru	Professeur des Universités, Université Pierre et Marie Curie, France.
	Eddy Caron	Maître de Conférences HDR École Normale Supérieure de Lyon, France.
Examinateurs	Thomas Noël	Professeur des Universités, Université de Strasbourg, France.
	Ibrahima Niang	Maître de Conférences HDR, Université Cheikh Anta Diop, Sénégal.
Directeur	Olivier Flauzac	Professeur des Universités, Université de Reims Champagne-Ardenne, France.
Co-directeur	Florent Nolot	Maître de Conférences HDR, Université de Reims Champagne-Ardenne, France.

Centre de Recherche en STIC (CReSTIC) - Équipe Systèmes Communicants (SysCom)

"Le savant est fier d'avoir tant appris ; le sage est humble d'en savoir si peu."
William Cowper

Remerciements

Il y a un peu plus de trois ans, le 10 décembre 2010, lorsque que je quittais mon pays pour venir entamer une thèse à l'Université de Reims, l'accomplissement de cette thèse semblait être un Everest insurmontable devant moi. Il y a un peu plus huit 8 mois, en septembre 2013, lorsque je commençais la rédaction effective, le point final de ce manuscrit semblait se trouver au bout d'un long tunnel interminable.

Aujourd'hui, c'est chose faite et à moi seul, j'y serais sans doute jamais arrivé. Après avoir rendu grâce à Allah, le Tout Miséricordieux, le Très Miséricordieux, je souhaite consacrer les premières lignes et pages de ce manuscrit pour REMERCIER VIVEMENT toutes les personnes, de près ou de loin, qui ont contribué à la réalisation de cette thèse.

Je décerne mes premiers remerciements à mes directeurs de thèse *Olivier Flauzac* et *Florent Nolot*. Je les remercie de la confiance qu'ils m'ont accordée en m'offrant l'opportunité de faire une thèse, de leur gentillesse, de leur disponibilité à tout moment pendant ces trois années de thèse, de leur écoute attentive, de leurs précieuses orientations et surtout de la liberté qu'ils m'ont donnée dans la réalisation de ces travaux tout en étant exigeant pour un travail de qualité. J'espère avoir été à la hauteur de leurs attentes.

J'adresse mes vives remerciements aux membres du jury de m'avoir fait l'honneur d'accepter d'évaluer mes travaux. J'exprime d'abord mes sincères remerciements à mes trois rapporteurs *Eddy Caron*, *Mohamed Mosbah* et *Maria Potop-Butucaru* qui ont bien voulu étudier ma thèse en profondeur. Je vous remercie de vos nombreuses remarques et suggestions très constructives pour améliorer la qualité du manuscrit. Ensuite, je témoigne mes profonds remerciements aux examinateurs *Jean-Frédéric Myoupo*, *Thomas Noël* et *Ibrahima Niang* pour l'intérêt qu'ils portent à mes travaux et d'avoir accepté de faire parti du jury.

Je tiens à remercier vivement et très sincèrement *Ibrahima Niang*, une personne aux valeurs humaines exceptionnelles que je connais depuis 2007 lors de mon année de Licence en informatique à l'UCAD, une personne qui a toujours cru en moi et par qui tout est parti. Je te remercie de m'avoir recommandé pour cette thèse. Je te remercie de m'avoir accompagné pendant ces trois années à travers tes séjours de recherche annuels à l'Université Reims pour s'imprégner de mes travaux. Je te remercie également de m'avoir donné goût à la recherche en m'encadrant lors de mon mémoire de DEA. Je te remercie aussi pour tes nombreux conseils que tu n'as cessé de me donner, pour ta gentillesse, pour le soutient sans faille dont tu as toujours fait preuve à mon égard.

Je remercie *Rafik Makhloufi* qui fut post-doc dans le projet *CPER CapSec Rofica* et avec qui j'ai beaucoup travaillé dans la dernière phase du projet. J'ai pas oublié nos nombreuses discussions, parfois houleuses, mais toujours constructives et uniquement dans le but de ressortir les meilleures idées. Je remercie également Bachar Salim Haggar, qui finissait sa thèse lorsque j'allais démarrer la mienne, pour touts ses conseils.

Je remercie *Hervé Deleau*, *Bamba Gueye*, *Amadou Niakary Diallo* et *Luiz Angelo Steffenel*, d'avoir bien voulu procéder à de nombreuses relectures de correction du manuscrit, contribuant à son amélioration.

J'ai effectué mes recherches au sein d'une équipe de recherche d'un laboratoire. Je tiens donc à remercier *Michael Krajecki* directeur du laboratoire *CReSTIC* d'avoir mis à ma disposition toutes les ressources du laboratoire dont j'ai eu besoin. De même, je remercie *Mamadou Mboup*, directeur de l'équipe *SysCom*, d'avoir également mis à ma disposition toutes les ressources d'équipe dont j'ai eu besoin. Je le remercie également pour tous les conseils qu'il m'avait donné lorsque je venais d'arriver à Reims.

Je remercie chaleureusement *Mouhamadou Diallo* qui m'a toujours soutenu sur tous les plans. Tu as été une oreille attentive et je te remercie de tous tes conseils et encouragements. Je te dis tout simplement un Grand merci.

Je remercie tous les membres du laboratoire que j'ai côtoyé durant mes années de thèse. En particulier, je remercie Cyril Rabat pour nos nombreux échanges, ses conseils et belles idées qu'il m'a donnés, l'aide qu'il m'a apportée lors de la préparation de mes enseignements. Je remercie également *Luiz Angelo Steffenel*, *Christophe Jaillet* et *Hervé Deleau* pour leurs nombreux conseils. Je remercie tous les autres membres de l'équipe *Jean-Charles Boisson*, *Arnaud Renard*, *Pierre Delisle*, *Thibault Bernard*, *Hacene Fouchal*, *Audrey Delevacq*, pour leur sympathie et leur gentillesse. Je remercie également les doctorants, *Christophe Goessen*, *Bassirou Gueye*, *Thierno Amadou Diallo* de m'avoir supporté pendant tout le temps que nous avons partagé le bureau ensemble.

Mes remerciements vont aux secrétaires du *CReSTIC Ida Lenclume* et *Geraldine Vitry*, du Département *Sergine Bristiel* et *Christine Ardilly*, pour leur disponibilité et toute l'aide qu'elles m'ont apportées durant ces années de thèse.

Je me tourne maintenant du coté de ma vie privée pour remercier toutes les personnes qui me sont si chères, qui occupent une place de choix dans mon cœur, qui m'ont toujours apportées un soutient sans faille depuis ma naissance jusqu'à maintenant.

J'exprime mes plus sincères remerciements à mes chers parents et à toute ma famille. *Papa*, merci du fond du cœur. Sans toi, je ne serais pas là aujourd'hui. Tu es une référence pour moi et tu fais ma fierté. Tu m'as tout apporté. Ton expérience de la vie, ton amour, ton écoute, tes conseils, tes encouragements, tes prières et ton soutien tant moral que matériel ont toujours été sans faille : merci infiniment à toi *Papa*, je t'adore.

A ma maman adorée *Fatou Kine Fall*, je ne saurai jamais te remercier assez pour ta persévérance lors de toutes les épreuves surmontées pour moi. Je t'exprime toute ma gratitude pour toute l'attention et tout l'amour que tu m'as toujours vouée. Merci pour tes conseils et prières, saches que tu occupes une place spéciale dans mon cœur. J'associe à ses remerciements ma petite sœur adorée *Ndeye Aïda Ba*. Elle m'a toujours porté une attention particulière.

Il y a cependant une personne qui n'aura plus jamais l'occasion de lire ces lignes. C'est le lieu pour lui rendre un vibrant hommage. J'ai le profond regret de citer *Adama Dieng*, feue ma grande mère que je remercie pour tous les efforts qu'elle a consenti durant sa vie afin m'assurer une bonne éducation. Reposes en paix et trouves ici la joie de l'accomplissement d'un de tes souhaits.

Il y a une personne, de très grande importance dans ma vie, qui m'accompagne et qui me soutient tout le temps par la force et l'énergie qui lui sont disponibles. Je voudrais nommer mon marabout et guide spirituel *El Hadji Mouhamadoul Hady Thiam*, qui est comme un père pour moi. Je te remercie pour tes prières, tes conseils et encouragements.

Je ne saurai finir sans adresser mes chaleureux remerciements à mes *Amis* de toujours et pour toujours pour notre compagnonnage sincère et sans faille. Mes *tendres* remerciements vont à *une personne spéciale* qui m'a particulièrement accompagné et supporté pendant mes années de thèse.

MERCI ...

Table des matières

Table des figures

Liste des tableaux

Introduction

Partant de ses travaux théoriques sur la commutation de paquets pour le transfert de données [Kle61], Léonard Kleinrock créa en 1969 ce qui est devenu aujourd'hui Internet [Kle69]. L'explosion de ce dernier ainsi que sa démocratisation observées ces dernières décennies ont amené les réseaux informatiques à fortement évoluer, aussi bien au niveau de leurs conceptions, que de leurs usages. Les réseaux informatiques ont pour but d'inter connecter plusieurs équipements pour l'échange d'informations entre eux. Ils sont composés d'un ensemble de nœuds, souvent en très grand nombre, communiquant les uns avec les autres à l'aide d'un médium de communication qui peut être filaire ou sans fil. Les nœuds dotés d'un médium sans fil peuvent être fixes ou mobiles. Les architectures des réseaux informatiques, 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. De nos jours, les réseaux informatiques sont devenus incontournables en occupant une place centrale dans les activités humaines du quotidien.

Il existe plusieurs critères de classification des réseaux informatiques selon leurs tailles, leurs modes de communication, leurs fonctionnements ou leurs débits, etc. De façon plus générique, nous les classons en deux grandes catégories : les réseaux avec infrastructure et les réseaux sans infrastructure. Dans la première catégorie, il existe une entité centrale qui coordonne toute l'activité du réseau. Le système de communication est fondé essentiellement sur l'utilisation de réseaux filaires mais aussi de stations de bases (avec une longue portée) pour couvrir l'ensemble des nœuds mobiles. Cela fait que les réseaux avec infrastructure nécessitent une importante logistique pour leur déploiement et leur administration. Parmi ces types de réseaux, nous pouvons citer les réseaux GSM [1] pour la téléphonie mobile, les réseaux LAN [2], MAN [3], WAN [4], etc. Dans les réseaux sans infrastructure, il n'existe aucune entité centrale, les nœuds communiquent directement entre eux. L'absence d'infrastructure centralisant les communications oblige les nœuds à participer à l'acheminement de l'information. En outre, les nœuds d'un tel réseau peuvent être mobiles et munis de média de communication sans fil. Ainsi, chacun d'eux

1. *Global System for Mobile communications*
2. *Local Area Network*
3. *Metropolitan Area Network*
4. *Wide Area Network*

peut communiquer avec l'ensemble des autres nœuds se trouvant dans sa zone de couverture (portée radio). La mobilité des nœuds conduit à de fréquents changements topologiques. D'où la nécessité de recalculer fréquemment les bases de connaissances nécessaires à l'acheminement de l'information. Toutes ces contraintes rendent difficile l'établissement de protocoles de communication.

Dans les réseaux sans infrastructure, nous retrouvons les réseaux ad hoc. Ils sont généralement désignés sous le nom MANET (Mobile Ad hoc NETwork). Ce sont des systèmes distribués complexes qui regroupent un important nombre de nœuds mobiles et équipés d'un médium sans fil pour les communications. Les réseaux ad hoc offrent une alternative aux réseaux avec infrastructure, notamment pour des applications qui requièrent un déploiement dynamique du réseau. Parmi les réseaux ad hoc, il existe des types de réseaux particuliers désignés sous le nom de Réseaux de Capteurs Sans Fil (RCSF) qui sont composés d'un grand nombre de petits dispositifs appelés *capteurs*. Ces derniers ont connu un essor important ces dernières années. Les capteurs ont la capacité de collecter des informations telles que la température, l'humidité, la pression, etc. Ils peuvent effectuer un certain nombre de traitements sur les données recueillies puis de coopérer entre eux pour les acheminer vers un centre de contrôle appelé station de base (*Base Station* (BS)). Du fait de la taille réduite des capteurs et de leur faible coût de production, les RCSF offrent de nombreuses applications pratiques et parfois sensibles notamment dans le domaine militaire, médical, environnemental, etc. Cependant, dans les RCSF, deux contraintes majeures demeurent : *(i)* la faible ressource énergétique des capteurs et *(ii)* la vulnérabilité aux pannes.

Un capteur est doté d'une batterie avec une quantité énergétique très limitée. En outre, l'hostilité qui caractérise souvent les zones de déploiement fait qu'il est difficile, voire même impossible de recharger ou de changer la batterie des capteurs. A cela s'ajoute que l'énergie d'un capteur est principalement consommée par trois opérations ; le captage de données, les traitements CPU [5] et les communications. Soulignons par ailleurs que les communications sont la source majeure de la consommation énergétique [PK00]. Par conséquent, vu la nature critique et sensible des applications qu'offrent les RCSF d'une part et les faibles capacités énergétiques des capteurs d'autre part, il est plus que nécessaire de minimiser les communications dans le but d'optimiser la consommation énergétique afin de prolonger la durée de vie du réseau. Pour réduire les communications, une solution consiste à structurer le réseau.

Optimiser seulement les communications ne suffit pas. Une autre spécificité des RCSF est la vulnérabilité aux pannes transitoires. En effet, les capteurs sont susceptibles de tomber en panne du fait de l'épuisement total de leur énergie ou du dysfonctionnement d'un de leurs composants. Également, la mobilité des nœuds peut entrainer des ruptures de liens de communication conduisant ainsi à des changements topologiques. Ainsi, comme montré dans plusieurs études [BJX04, LWJ05, WXM07], la tolérance aux pannes est une problématique capitale dans les RCSF. Dans un système distribué comme les RCSF, Johnen et Mekhaldi ont montré dans [JM11] que l'auto-stabilisation est une solution efficace de tolérance aux pannes transitoires.

5. *Central Processing Unit*

Pour toutes les raisons que nous venons d'évoquer ci-dessus, des solutions de structuration auto-stabilisantes ont été proposées dans le but de minimiser les communications et de tolérer les pannes transitoires [MBF04, MFGLT05, DFG06, KM06, JN08, JN09, DDL09, OHN09, CDDL10, DLV10, KYSI11, LT11]. Selon le modèle de communication utilisé, nous distinguons les solutions fondées sur un modèle à états [DFG06, KM06, JN08, JN09, DDL09, CDDL10, DLV10] et celles qui utilisent un modèle à passage de messages [MBF04, MFGLT05, OHN09, KYSI11, LT11].

Contributions

Partant des observations sur les solutions listées ci-dessus, nous proposons un algorithme de structuration des réseaux ad hoc dans le but d'optimiser les communications et de tolérer les pannes transitoires. Notre solution, que nous intitulons *SDEAC* (self-Stabilizing Distributed Energy-Aware k-hops Clustering), est auto-stabilisante, déterministe, distribuée et utilise un modèle asynchrone à passage de messages. Elle ne nécessite aucune initialisation ni intervention extérieure et construit des *clusters* non-recouvrants de diamètre au plus $2k$, avec k étant le rayon maximal des *clusters* fixé au préalable. Nous utilisons le critère de l'identité maximale des nœuds pour l'élection des *cluster-heads*. Le choix d'une métrique fondée sur l'identité des nœuds, comme nous le montrons à travers nos résultats obtenus, apporte plus de stabilité lors de la phase de *clustering* par rapport à d'autres critères comme le degré ou l'énergie des nœuds.

Nous prouvons formellement que les propriétés fondamentales d'un système auto-stabilisant que sont la *convergence* et la *clôture* sont vérifiées. En effet, nous prouvons que partant d'une configuration quelconque et sans occurrence de pannes transitoires, une configuration légale est atteinte en au plus $n + 2$ transitions. Nous montrons également que l'exécution de SDEAC nécessite une occupation mémoire de $(\Delta_u + 1) \times \log(2n + k + 3)$ bits pour chaque nœud u du réseau, avec Δ_u étant le degré (nombre de voisins) de u, k le rayon maximal des *clusters* et n la taille du réseau. Le tableau 1 donne une comparaison analytique du voisinage, du temps de stabilisation et de l'occupation mémoire de SDEAC avec les principales solutions de structuration auto-stabilisantes à k sauts de la littérature [DDL09, DLV10, CDDL10, LT11] bien que ces dernières ne se fondent pas sur le même modèle de communication que SDEAC. Dans ce tableau, g désigne la borne maximale du nombre de nœuds dans les *clusters* et G_u^k voisinage du nœud u à distance k.

Par une campagne de simulation sous OMNeT++ [6], nous évaluons les performances moyennes de SDEAC pour des topologies quelconques. Les résultats montrent que, quel que soient la densité et la taille du réseau ainsi que le paramètre k, le temps de stabilisation est très inférieur à celui du pire des cas prouvé formellement ($n + 2$ transitions dans le cas de la chaine ordonnée). De plus, en présence de pannes transitoires après la stabilisation du réseau, nous constatons que le nombre de transitions supplémentaires nécessaires pour retrouver un état stable dans le réseau est inférieur par rapport à celui du *clustering*.

6. http ://www.omnetpp.org/

	Solutions	Voisinage	Modèle	Temps	Mémoire
***Clusters* de diamètre $2k$**	**SDEAC**	1 saut	à passage de messages	$n+2$	$(\Delta_u+1)\times\log(2n+k+3)$
	[LT11]	k sauts	à passage de messages	$O(k)$; $O(g\times k\times\log(n))$	$O(\lvert G_u^k\rvert\times(\log(n)+\log(k)))$
	[DDL09]	k sauts	à états	$O(n)$; $O(n^2)$	$O(\log(n))$
	[DLV10]	k sauts	à états	$O(k)$	$O(k\times\log(n))$
	[CDDL10]	k+1 sauts	à états	$O(n\times k)$	$O(log(n)+log(k))$

Table 1 – Performances de SDEAC vis-à-vis des solutions [DDL09, DLV10, CDDL10, LT11]

Nous étudions la consommation énergétique de SDEAC dans le contexte des réseaux de capteurs sans fils (RCSF). Dans cette étude, nous proposons une généralisation du critère d'élection des *cluster-heads*. Nous montrons que SDEAC est une approche générique et qu'elle peut s'utiliser avec d'autres critères parmi les plus usuels tels que le degré ou l'énergie des nœuds. Nous évaluons les coûts et performances de chacun de ces critères d'élection en mesurant et comparant deux métriques : la consommation énergétique qui est induite par les communications [HCB00, PK00, AY07] et les pourcentages de *clusters* construits [YF04, CS10, AVX+12]. Les résultats d'évaluation montrent que les trois critères d'élection des *cluster-heads* permettent le passage à l'échelle et construisent un pourcentage de *clusters* conforme à la recommandation. En effet, afin d'éviter la construction d'un faible nombre de *clusters* très denses ou un nombre conséquent de *clusters* de petite taille, il est recommandé d'avoir un pourcentage de *clusters* compris entre 5% et 20% de taille du réseau [YF04, CS10, AVX+12]. Cependant, le critère d'identité maximale, du fait de sa simplicité d'utilisation et de sa stabilité lors de la phase de *clustering*, présente une plus faible consommation énergétique.

Pour valider SDEAC vis-à-vis des algorithmes existants, nous comparons sa consommation énergétique avec celle de la solution de Mitton et *al.* [MFGLT05] qui à notre connaissance reste la seule à opérer dans le même modèle que la notre : auto-stabilisante, déterministe, distribuée, fondée sur un modèle asynchrone à passage de messages et construisant des *clusters* à k sauts. Les résultats montrent que les deux approches permettent le passage à l'échelle. Cependant, SDEAC offre l'avantage de réduire la consommation énergétique de 42% à 49%.

Enfin, nous présentons trois cas d'utilisations des *clusters* construits par SDEAC pour l'acheminement de l'information dans les RCSF : le Routage Sans Agrégation (RSA), le Routage avec Agrégation Partielle (RAP) et Routage avec Agrégation Totale (RAT). Les résultats montrent que pour le délai de bout en bout, le scénario de RSA, bien qu'engendrant d'importants échanges de messages, offre une meilleure performance comparé au RAP et au RAT. Pour la consommation énergétique totale dans le réseau, comme le RAP réduit au maximum les messages transitant dans le réseau, il présente la consommation énergétique la plus faible par rapport au RSA et RAT. Pour cette même raison, il offre la meilleure durée vie du réseau. Nous constatons que le RAT présente de moins bonnes performances en termes de consommation énergétique

totale à cause des échanges de messages nécessaires à la coopération entre les agents.

Organisation de la thèse

Nous organisons ce manuscrit en cinq (5) chapitres :

Chapitre 1 : Nous présentons les systèmes distribués qui sont le cadre général de nos travaux de recherche. Puis, nous abordons les réseaux ad hoc et le cas particulier des RCSF qui nous intéressent particulièrement.

Chapitre 2 : Nous décrivons d'abord les solutions de structuration auto-stabilisantes comme non auto-stabilisantes proposées dans la littérature dans le but de réduire les communications et de tolérer les pannes transitoires. Puis, nous soulevons notre problématique de recherche.

Chapitre 3 : Nous présentons les spécifications de SDEAC, son principe d'exécution, sa description formelle. Ensuite, nous établissons la preuve formelle des propriétés de convergence et de clôture ainsi que l'occupation mémoire requise pour son exécution. Enfin, par une campagne de simulation, nous évaluons ses performances moyennes.

Chapitre 4 : Nous étudions la consommation énergétique de SDEAC dans le contexte des RCSF et nous la comparons avec la solution de structuration fondée sur le même modèle c'est-à-dire auto-stabilisante, distribuée, déterministe, fondée sur un modèle asynchrone à passages de messages et construisant des *clusters* à k sauts.

Chapitre 5 : Nous présentons trois approches d'utilisation des *clusters* construits avec SDEAC pour l'acheminement de l'information dans les RCSF.

Enfin, nous clôturons ce document par une synthèse générale et soulevons quelques pistes de réflexions, tant sur le plan théorique et scientifique que sur le plan technique, vers lesquelles nous aimerions orienter nos futurs travaux de recherche.

Chapitre 1

Systèmes distribués et réseaux ad hoc

Résumé. *Dans ce chapitre, nous présentons le cadre général de nos travaux de recherche : les systèmes distribués. Après avoir présenté sa définition, son modèle d'exécution, ses modèles de communication et décrit quelques algorithmes classiques, nous abordons la tolérance aux pannes transitoires. Ensuite, nous poursuivons sur les réseaux ad hoc en donnant leurs spécificités avant d'aborder les réseaux de capteurs sans fil en suivant deux axes : la faible ressource énergétique des capteurs et la vulnérabilité aux pannes.*

Sommaire

1.1 Généralités sur les systèmes distribués

Dans cette section, nous présentons d'abord les systèmes distribués et leur modélisation. Ensuite, nous décrivons leur modèle d'exécution et leurs différents modèles de communication.

Enfin, nous donnons une discussion sur le choix du modèle de communication adopté.

1.1.1 Définitions

Un système distribué est défini comme étant une collection d'entités de traitement et de stockage qui sont autonomes, inter-connectées et peuvent communiquer les unes avec les autres [Ray85].

L'objectif d'un système distribué est de fournir un service qui ne pourrait pas être réalisé par une seule entité en termes de fonctionnalité, disponibilité, puissance de traitement ou de stockage, temps de réponse, fiabilité, etc.

Les entités peuvent être des processus, des processeurs ou des ordinateurs, etc. Elles sont généralement appelées *sites* ou *nœuds* du système distribué. Afin d'être qualifié d'autonome, chaque nœud doit avoir son propre système de contrôle local indépendant des autres nœuds. Pour être inter-connecté, chaque nœud doit disposer d'unité de communication pour assurer l'échange d'information. Ainsi, les nœuds sont inter-connectés et s'échangent des informations grâce à un modèle de communication.

Un système distribué peut être homogène. Dans ce cas tous les nœuds sont identiques. Mais aussi les nœuds du système peuvent être de divers types. On dit alors que le système distribué est hétérogène. Les nœuds coopèrent à l'aide d'un modèle de communication pour accomplir la tâche globale du système dont le déroulement est induit par un algorithme distribué. Un algorithme distribué est constitué de l'ensemble des algorithmes des différents nœuds du système distribué. Au niveau de chaque nœud, l'exécution d'algorithme distribué est déclenchée par un événement de type réception de message ou l'épuisement d'un «*timer*». Cet événement entraine donc une exécution d'une instruction en interne qui peut modifier l'état du nœud. Puis, cette modification de l'état du nœud peut conduire à l'émission d'un message ou au déclenchement d'un «*timer*». L'algorithme distribué s'exécute ainsi pour accomplir la tâche du système.

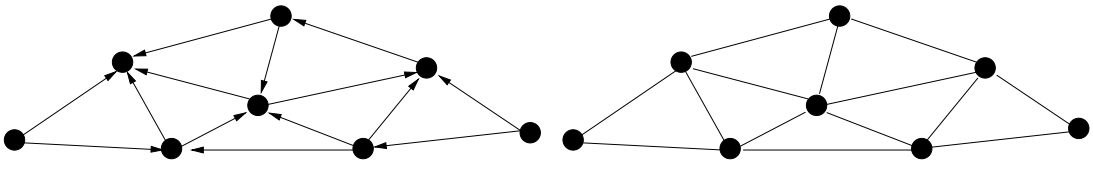
Les systèmes distribués offrent plusieurs utilisations qui ont favorisé leur développement massif. Ils peuvent permettre le partage de ressources, l'augmentation de la fiabilité des données par le biais de réplications, la haute disponibilité, l'amélioration des performances par le parallélisme, la simplification de la conception d'un système par une structuration modulaire etc.

Il existe plusieurs ouvrages dans la littérature traitant des systèmes distribués ainsi que les algorithmes distribués [Ray85, RH88, Ray91, Ray92, Lav95, Lyn96, Tel00, AW04, Ray13].

1.1.2 Quelques notions sur les graphes

Pour modéliser un système distribué, les notions et terminologies issues de la théorie des graphes sont employées [Gib85, Pri94, GM95, BFH12e]. Un système distribué est modélisé par un graphe connexe noté $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ désigne l'ensemble des sommets (aussi appelé nœuds) du graphe du réseau avec $|\mathcal{V}| = n$ et $n \geq 2$. L'ensemble $\mathcal{E}$ représente les connexions existantes entre tous les nœuds du graphe. Soient u et v deux nœuds du graphe. Le graphe $\mathcal{G}$ est connexe si et seulement il existe un chemin entre tout couple de nœuds (u, v) de $\mathcal{E}$. Le graphe $\mathcal{G}$ peut être orienté ou non-orienté. Le graphe $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ est dit orienté (cf. Figure 1.1(a)) si et seulement si $\mathcal{E}$ est un ensemble de couples ordonnés distincts (u, v) de sommets appelés *arcs* où u est l'origine et v la destination. Par contre, le graphe $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ est non-orienté (cf. Figure 1.1(b))

si et seulement si $\mathcal{E}$ est un ensemble de paires (u, v) de sommets distincts appelés *arêtes* où $(u, v) = (v, u)$.



(a) Graphe orienté (b) Graphe non-orienté

FIGURE 1.1 – Graphe orienté vs Graphe non-orienté

Le graphe $\mathcal{G}$ peut être anonyme. Dans ce cas, les nœuds de l'ensemble $\mathcal{V}$ ne possèdent pas d'identité, ils sont indiscernables. Par contre, lorsque tout nœud de $\mathcal{V}$ est distingué par l'attribution d'une identité unique, distincte de toutes les autres identités des autres nœuds, on parle alors de graphe avec identité.

Modèle de graphes adopté

Dans nos travaux de recherche, nous considérons des systèmes distribués représentés par des graphes connexes, non-orientés et avec identités. Ainsi, nous supposons que chaque nœud u du graphe $\mathcal{G}$ est caractérisé par une « valeur d'identification » qui lui est propre : son *identité* qui est alors notée id_u. Cette identité peut être un nombre entier ou une chaine de caractères. De plus, elle est tirée d'un ensemble $\mathcal{I}$ muni d'un *ordre total* qui peut être l'ordre numérique sur $\mathbb{N}$, l'ordre alphabétique ou lexicographique. Pour de tels graphes, nous donnons les définitions suivantes que nous utilisons par la suite.

Définition 1.1. ***(Voisinage)***
Le voisinage d'un nœud u, noté N_u, représente l'ensemble des nœuds $v \in \mathcal{V} | (u, v) \in \mathcal{E}$.

Définition 1.2. ***(Degré)***
Le degré d'un nœud u, noté Δ_u, représente le nombre de nœuds dans le voisinage de N_u de u. $\Delta_u = |N_u|$.

Définition 1.3. ***(Distance)***
La distance $d_{(u,v)}$ entre deux nœuds u et v quelconques dans le graphe $\mathcal{G}$ représente la longueur du plus court chemin (le nombre d'arêtes minimal) entre u et v.

Quelques topologies

Un système distribué peut être organisé suivant une topologie quelconque, c'est-à-dire les connexions existantes entre les nœuds ne suivent aucune organisation particulière. Dans ce cas, on dit que la topologie du système distribué suit un graphe quelconque. Cependant, les différentes connexions entre les nœuds d'un système distribué peuvent suivre certaines règles logiques. On dit alors que la topologie obéit à un graphe régulier. Dans la figure 1.2, nous listons quelques topologies régulières.

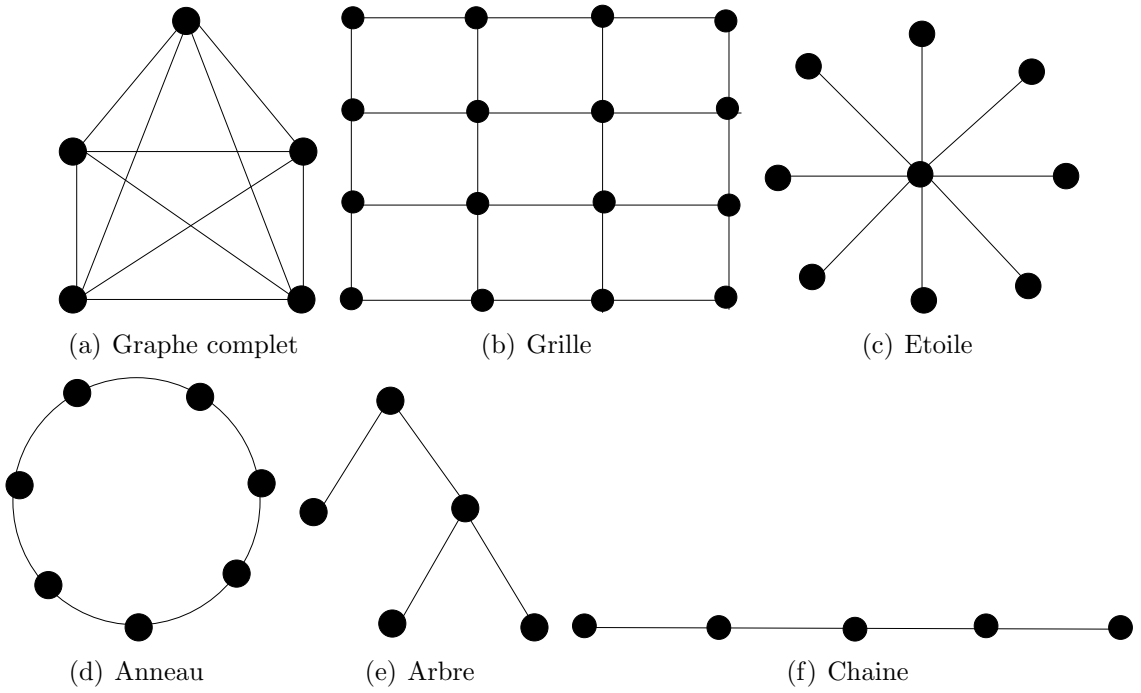


FIGURE 1.2 – Quelques topologies régulières de systèmes distribués

1.1.3 Modèle d'exécution

Après avoir vu qu'un système distribué pouvait être représenté par les graphes, nous abordons ici son modèle d'exécution.

Dans [Tel00], Tel modélise l'exécution d'un système distribué par un *système de transitions* introduit par Anrold-Nivat dans [Arn92]. Avant de présenter un système de transitions et le modèle d'exécution d'un système distribué, nous allons rappeler les définitions des notions de *état* et de *configuration*.

Définition 1.4. ***(État)***
L' état d'un nœud est défini par ses variables locales ainsi que celles de ses autres voisins.

L'état global du système est alors obtenu en faisant le produit cartésien de l'ensemble des états de tous les nœuds.

Définition 1.5. ***(Configuration)***
Une configuration est une instance de tous les composants du système, c'est-à-dire l'état global du système.

Un système de transitions se définit comme suit :

Définition 1.6. *(**Système de transitions**)*
Un système de transitions est un triplet $\mathcal{S} = (\mathcal{C}, \mapsto, \mathcal{I})$ où :
- *$\mathcal{C}$ est l'ensemble de toutes les configurations possibles ;*

- $\mapsto$ *est une relation binaire dans* $\mathcal{C}$ *;*
- $\mathcal{I}$ *un sous ensemble de* $\mathcal{C}$ *de toutes les configurations initiales.*

Donc, un système de transitions représente toutes les évolutions possibles d'un système distribué. La relation $\mapsto$ étant un sous ensemble de $\mathcal{C}\times\mathcal{C}$, tout couple de configurations (γ_i, γ_{i+1}) appartenant à $\mapsto$ telle que $\gamma_i \mapsto \gamma_{i+1}$ est une transition et est notée τ_i. Une transition est encore appelée *étape* ou *round*. L'exécution d'un système distribué suivant un système de transitions peut être maximale ou finie.

Définition 1.7. ***(Exécution maximale)***
Soit $\mathcal{S} = (\mathcal{C}, \mapsto, \mathcal{I})$ *un système de transitions. Une exécution maximale de* $\mathcal{S}$ *est une séquence maximale* $\mathcal{E} = (\gamma_0, \gamma_1, \gamma_2, \ldots)$ *où :*

- $\gamma_0 \in \mathcal{I}$ *;*
- $\forall i \geq 0, \gamma_i \mapsto \gamma_{i+1}$.

Définition 1.8. ***(Exécution finie)***
Une séquence $\mathcal{E} = (\gamma_0, \gamma_1, \gamma_2, \ldots)$ *avec* $\gamma_i \mapsto \gamma_{i+1}$ *pour tout i est dite finie si elle se termine par une configuration terminale.*

Définition 1.9. ***(Configuration terminale)***
Une configuration terminale est une configuration γ *pour laquelle il n'existe aucune autre configuration* δ *telle que* $\gamma \mapsto \delta$

En considérant l'exécution d'un système distribué qui suit un système de transitions, un algorithme s'exécutant au niveau de chaque nœud est une collection de règles disjointes de la forme suivante :

$$Algorithme \equiv < Regle_1 > | < Regle_2 > | < Regle_3 > | \ldots | < Regle_n >$$

Où :

$$< Regle_i > ::< garde > \longrightarrow < action >$$

Pour chaque règle $Regle_i$, $< garde >$ est un prédicat booléen qui est fonction de l'état local du nœud ainsi que ceux de ses voisins. $< action >$ est une instruction exécutée par le nœud en interne et qui peut modifier son état local.

1.1.4 Synchronisation

L'exécution d'un système distribué peut suivre deux modes de synchronisation différents : synchrone ou asynchrone [RH88].

Système synchrone

Dans un système synchrone, les horloges locales de tous les nœuds sont synchronisées et marquent la même valeur. Par conséquent, tous les nœuds commencent simultanément leurs activités au signal d'un initiateur global appelé *synchroniseur*, qui synchronise cycle par cycle chaque événement et chaque action de tout nœud du système. On parle alors d'ordonnancement de tâches. L'exécution d'un système synchrone est partitionnée en rounds. Un round correspond à un envoi et à une réception de tous les messages.

Système asynchrone

Un système asynchrone est caractérisé par l'absence d'horloge globale qui serait perceptible par les composants du système à un « instant » donné. Le concept d'instant est d'ailleurs exogène à un système asynchrone. Les seules classes d'événements perçues par un nœud quelconque sont soit les événements produits par le nœud lui même de façon « spontanée », soit des événements causés par d'autres nœuds. Le caractère asynchrone d'un système de n nœuds implique qu'il n'existe aucune horloge globale mais seulement n horloges locales aux nœuds sans relation entre elles. L'évolution de l'état global du système est induite par un événement de type réception d'un message ou par le déclenchement de «*timers*» autonomes dont l'exécution est indépendante de tout mécanisme externe.

Dans [RH88], Raynal et Helary ont proposé plusieurs algorithmes de synchronisation permettant de simuler et de contrôler l'exécution d'algorithmes synchrones dans un contexte asynchrone.

1.1.5 Modèles de communication

Nous avons vu qu'un système distribué pouvait être représenté par des graphes et que son exécution, qui peut être synchrone ou asynchrone, suit un système de transitions. Abordons maintenant les différents modèles de communication qui peuvent être utilisés pour les échanges d'information.

Dans un système distribué, les communications entre les nœuds peuvent s'effectuer suivant un modèle à états, à registres ou à passage de messages. Les deux premiers modèles constituent la communication dite à mémoires partagées. Chacun de ces trois modèles de communication présente des spécificités qui lui sont propres.

Modèle à états

Le modèle à états a été introduit par Dijkstra en 1974 dans [Dij74]. La particularité de ce modèle, comme illustré dans la figure 1.3, est que chaque nœud peut lire l'état de ses voisins.

Un système distribué est dit « *à états* » si et seulement si pour tout couple de voisins (u, v), u dispose exclusivement du privilège d'écriture (pour passer des informations) sur son état et peut lire (récupérer des informations) l'état de v. Respectivement, v peut lire l'état de u et dispose exclusivement du privilège d'écriture sur son état. Ainsi, chaque nœud connait directement l'état de tous ses voisins.

Dans un modèle à états, un ordonnanceur est utilisé pour diriger les différents changements d'états du système. Cet ordonnanceur est communément appelé démon (*Daemon*). Un démon est un mécanisme qui, durant chaque configuration γ_i, choisit à partir de l'ensemble de tous les nœuds *déclenchables* $\mathcal{N}$, un sous ensemble $\mathcal{Q} \subseteq \mathcal{N}$. Un nœud est dit déclenchable s'il est susceptible d'exécuter une action. Ainsi, à tout nœud de l'ensemble $\mathcal{Q}$, le démon choisit une règle qui va s'exécuter puis le nœud est libéré. Le démon peut être centralisé, distribué ou totalement distribué.

Le démon est dit centralisé (*Central Daemon*) si et seulement si à chaque configuration, $|\mathcal{Q}| = 1$. En d'autres termes, à chaque configuration γ_i, parmi l'ensemble de tous les nœuds

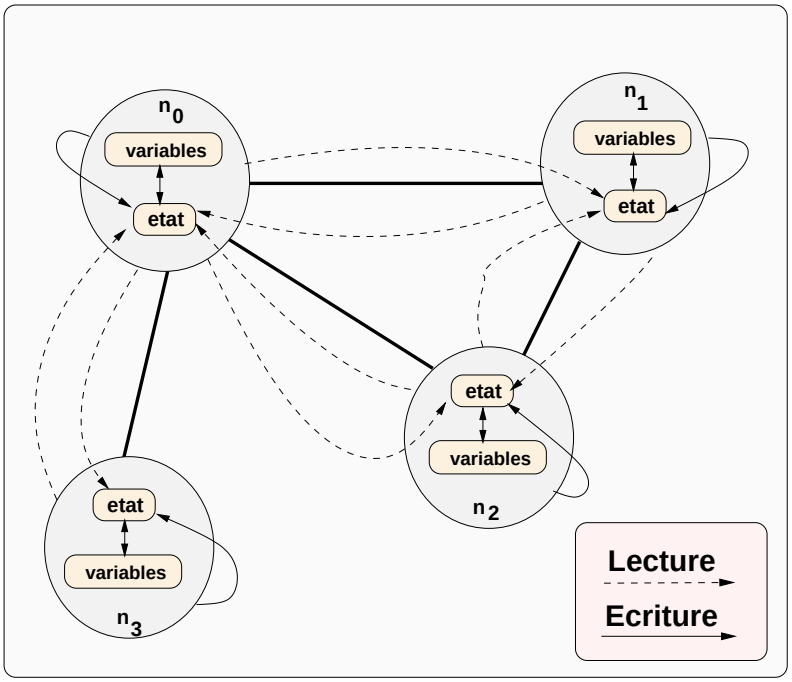


FIGURE 1.3 – Modèle à états

déclenchables, le démon en choisit un seul qui aura l'exclusivité d'exécuter une règle.

Le démon est dit distribué (*Distributed Daemon*) si et seulement si $1 < |\mathcal{Q}| \leq |\mathcal{N}|$. C'est-à-dire à chaque configuration γ_i, un sous ensemble de l'ensemble des nœuds déclenchables est sélectionné par le démon pour exécuter une règle applicable. Dans ce cas, l'exécution s'effectue de manière synchrone. Tous les nœuds sélectionnés exécutent leurs règles en même temps.

Un démon totalement distribué (*Fully Distributed Daemon*) est semblable à un démon distribué à la différence que l'exécution des règles est asynchrone. Ainsi, dès lors que l'autorisation est donnée par le démon, chaque nœud est libre d'exécuter sa règle quand bon lui semble.

Pour un démon centralisé, distribué ou totalement distribué, un problème peut subsister. Si un nœud continuellement déclenchable n'est jamais choisit par le démon, alors l'exécution de l'algorithme peut ne pas se dérouler correctement. Pour palier ce problème, la notion d'équité à été introduite. L'équité du démon peut être forte ou faible.

Le démon peut alors être *faiblement équitable* (*unfair*). Dans ce cas, si un nœud u est continuellement déclenchable ($u \in \mathcal{N}$) à partir d'une configuration γ_i alors il existe une configuration γ_j telle que $\gamma_i \longmapsto \gamma_j$ où le démon est contraint de le choisir ($u \in \mathcal{Q}$) pour exécuter une règle.

Par contre, le démon est dit *fortement équitable* (*fair*) si pour tout nœud u infiniment souvent déclenchable à partir d'une configuration γ_i, alors il existe une configuration γ_j telle que $\gamma_i \longmapsto \gamma_j$, où u est choisi par le démon pour exécuter une règle.

Si aucune des deux hypothèses n'est prise en compte alors le démon est considéré comme inéquitable.

Modèle à registres

Le modèle à registres constitue avec le modèle à états le deuxième modèle de communication par mémoires partagées. Il présente les mêmes spécificités que le modèle à états à la différence que les échanges d'informations entre les nœuds s'effectuent à l'aide de registres associés aux canaux de communication. Ainsi, pour tout couple (u, v) de nœuds communiquant, il est associé un couple de registres $\{R_{(u,v)}; R_{(v,u)}\}$ tel que u dispose des droits d'écriture et de lecture sur son registre $R_{(u,v)}$ et uniquement du droit de lecture sur le registre $R_{(v,u)}$ du nœud v (respectivement v dispose des droits d'écriture et de lecture sur son registre $R_{(v,u)}$ et uniquement du droit de lecture uniquement sur le registre $R_{(u,v)}$ du nœud u).

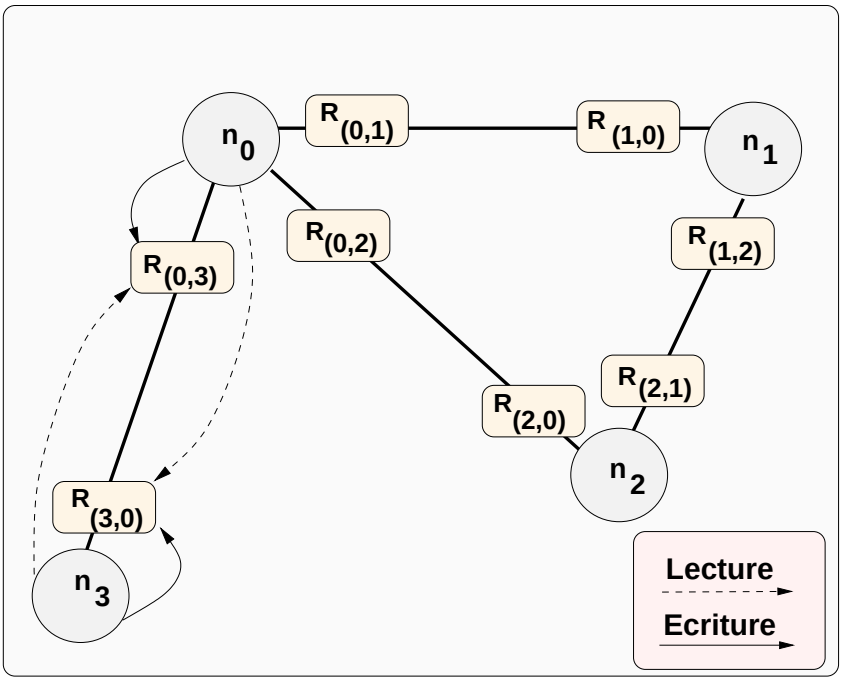


FIGURE 1.4 – Modèle à registres

La figure 1.4 montre le nœud n_0 qui écrit sur son registre $R_{(0,3)}$ pour envoyer une information au nœud n_3 et lit sur le registre $R_{(3,0)}$ les informations du nœud n_3. Les algorithmes s'écrivent de façon similaire à ceux du modèle à états mais avec la particularité de considérer des registres.

Le modèle à passage de messages

Dans un modèle à passage de messages, les nœuds communiquent entre eux par émission et réception de messages via des canaux de communication. Dans un tel modèle, comme illustré dans la figure 1.5, chaque couple de nœuds (u, v) est relié par un canal de communication. Les canaux de communication d'un nœud u sont étiquetés par un identifiant l pris entre 1 et δ où δ représentant le degré du nœud (nombre de voisins). Plusieurs hypothèses peuvent être faites sur un tel modèle. La taille des canaux de communication peut être bornée, finie mais aussi non bornée, infinie. Dans le cas où la taille des canaux est bornée alors l'écriture sur un canal est bloquante si la file d'attente est remplie et la réception d'un message permet de libérer

de la place. De plus, pour tout couple de nœuds communiquant dans un modèle à passage de messages, l'ordre de réception des messages peut être identique à leur ordre de transmission (*pas de déséquencement*). Dans ce cas, les transmissions sont FIFO [1]. L'ordre des messages reçus peut être éventuellement différent de l'ordre des messages transmis (*il peut y avoir déséquencement*). Dans cette hypothèse, les transmissions le long des canaux de communications ne sont pas FIFO.

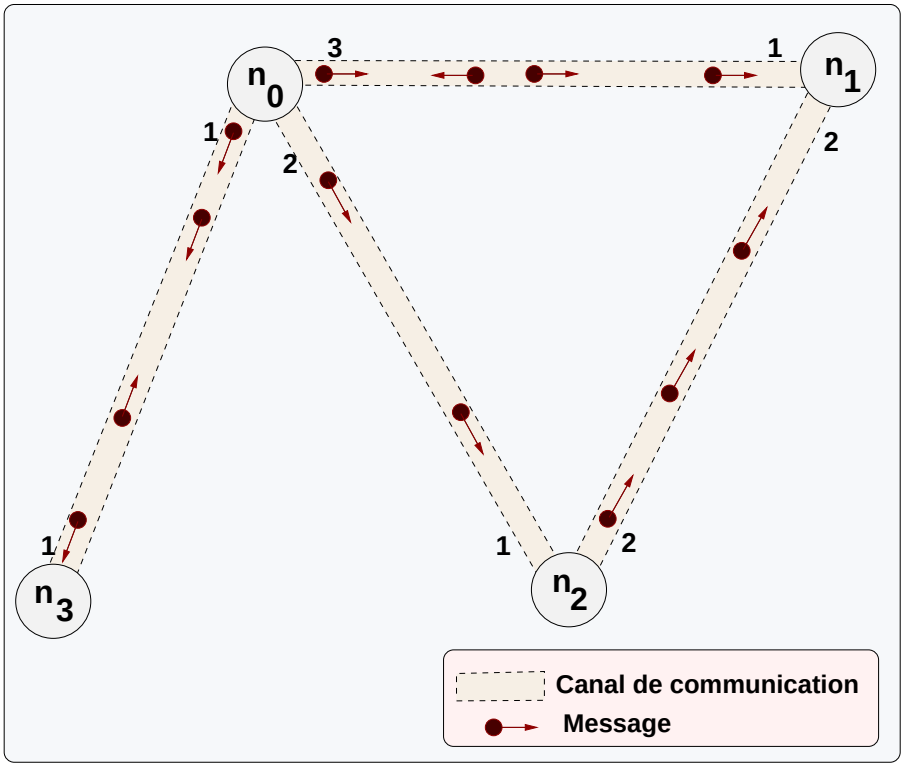


FIGURE 1.5 – Modèle à passage de messages

Un modèle à passage de messages peut être synchrone ou asynchrone. Dans un modèle à passage de messages synchrone, tous les nœuds envoient simultanément leurs messages au signal du synchroniseur d'horloge. Tout message envoyé par un nœud à temps t à un voisin est reçu et traité par ce voisin au temps $t + 1$, le temps *global du système*, commun aux deux nœuds. Ce qui ramène à supposer que l'horloge globale du système synchronise cycle par cycle chaque événement et chaque action de tout nœud du système. Par contre, un modèle à passage de messages est dit asynchrone si et seulement si les délais d'acheminement des messages le long des canaux de communication sont finis mais non bornés. Un message prend alors un temps fini mais arbitrairement long dans un canal de communication reliant un nœud u à un voisin v.

Discussions et choix du modèle à passage de messages

Chacun des trois modèles que nous venons de présenter ci-dessus est une abstraction qui peut être utilisée comme modèle de communication dans la conception et l'écriture d'algorithmes

1. ***F**irst-**I**n **F**irst-**O**ut*

distribués. Il présente chacun des propriétés qui leurs sont propres et peuvent être adéquates pour une situation donnée.

Le modèle à états permet de faire abstraction des problèmes liés à la communication et d'éviter l'utilisation de protocoles de communication. Un tel modèle offre un avantage remarquable : tout nœud peut lire la mémoire de ses voisins. Le modèle à registres permet une première approche vers l'utilisation et la gestion de messages ainsi que l'échange d'informations entre les nœuds d'un système distribué. Contrairement à un modèle à états, dans un modèle à registres, un nœud choisit les données qu'il souhaite transmettre à un voisin et les met à la disposition de ce dernier grâce au registre qu'ils partagent ensemble. Ces deux modèles qui constituent la communication en mémoires partagées (modèle à états et modèle à registres) modélisent parfaitement des systèmes distribués comme les machines parallèles et sont adaptés et utilisables dans le contexte de la programmation parallèle multi-threads.

Le modèle à passage de messages prend en considération l'ensemble des spécificités relatives aux échanges de messages entre différents nœuds du système : incohérence (due à l'asynchronisme dans le cas d'un système asynchrone à passage de messages) entre les données reçues et celles présentes dans les mémoires de l'émetteur, l'altération possible du message lors de sa transmission, la perte ou la réplications de messages. Nous nous intéressons à ce modèle car il est plus réaliste et plus adapté aux contraintes des systèmes distribués que nous traitons dans nos travaux de recherche. De plus, comme nous cherchons à nous rapprocher le plus possible des conditions réelles, nous optons pour un modèle asynchrone à passage de messages.

Toutefois, il faut souligner que la transformation d'un modèle de communication à un autre reste possible bien que cela entraine des surcoûts supplémentaires. En effet, la mémoire partagée peut être vue comme un lien de communication acceptant un seul message à fois. Ainsi, de nombreux travaux proposent des méthodes pour adapter un algorithme conçu dans un modèle particulier vers un autre [GM91, KP93, DIM97].

1.1.6 Algorithmes classiques

Dans le cadre des systèmes distribués, il existe plusieurs algorithmes classiques qui constituent les briques de base pour l'écriture d'autres algorithmes plus complexes. Plusieurs ouvrages traitent de ces problèmes classiques parmi lesquels nous pouvons citer [Lav95, Ray85, Tel00, AW04]. A titre indicatif, nous listons quelques algorithmes classiques.

- **Les algorithmes à vagues** : ils sont utilisés pour la diffusion d'une information, la synchronisation, l'envoi d'ordre, des calculs locaux, etc. dans le réseau. Un algorithme à vague peut être constitué d'une seule vague ou d'une succession de vagues. Dans cette dernière, l'algorithme fonctionne de façon répétitive : à chaque fois une exécution se termine, ou nouvelle commence. Une instruction particulière appelée « décision » est appelée pour terminer une exécution. Le concept de vague a été exprimé en tant que tel tout d'abord par Chang dans [Cha82] puis particulièrement approfondie par Schneider et Lamport dans [SL85] et par Helary et *al.* dans [HMPR86].

- **La détection de propriété et par extension de terminaison** : c'est un mécanisme de contrôle distribué qui permet de vérifier une propriété de stabilité globale (la terminaison). Les algorithmes de détection de terminaison montrent ainsi qu'un ensemble de nœuds vérifie le prédicat de terminaison. Les solutions de référence proposées dans [DS80, DFVG83, Mis83, Ran83] commencent par structurer cet ensemble de façon à pouvoir le parcourir pour vérifier la propriété de terminaison. Les structures de base utilisées sont l'anneau unidirectionnel [DFVG83], un arbre couvrant [DS80] ou un circuit quelconque tiré du graphe d'origine [Mis83, Ran83]. Pour la structure d'arbre couvrant, le mode de parcours utilisé est le calcul diffusant alors que pour l'anneau ou le circuit quelconque un jeton circulant est utilisé pour parcourir la structure.
- **L'exclusion mutuelle** : c'est le paradigme des problèmes de concurrence dans un système distribué. Elle a pour but d'éviter l'interférence des accès des différents nœuds du système à une ressource partagée. Elle consiste donc à attribuer équitablement une ressource critique (et ne peut donc être possédé indéfiniment par un même nœud du système). Les algorithmes de référence d'exclusion mutuelle qui ont été proposés se classent en deux grandes familles : les algorithmes fondés sur les permissions et ceux fondés sur l'utilisation de jeton. Dans la première famille, à tout nœud u est associé un ensemble de nœuds auxquels u doit demander la permission avant d'utiliser la section critique (ressource partagée) ; ces demandes sont véhiculées à l'aide des messages de type *requête*. Dans cette famille d'algorithmes, nous pouvons citer entre autres [RA81, Mae85, Sin91]. Pour la seconde famille d'algorithmes, la propriété de sureté (*à tout instant, il y a au plus un seul nœud du système disposant de la ressource partagée*) est assurée trivialement par l'unicité du jeton qui circule dans le système : le nœud possédant le jeton dispose du droit d'utiliser la section critique. Pour assurer la vivacité (*tout autre nœud demandant la ressource partagée est servit au bout d'un temps fini*), un déplacement séquentiel du jeton entre les sites demandeurs est imposé soit par un mouvement perpétuel, soit par des requêtes explicites. Dans cette famille d'algorithmes, nous pouvons citer entre autres [LeL77, LT87, HPR88].
- **Élection de leader** : l'objectif est de distinguer un nœud afin de le doter d'un certain nombre de privilèges. Ainsi, dans une configuration initiale du système, tous les nœuds actifs sont dans le même état (*candidat*). Et dans une configuration finale, un seul nœud est dans un état prédéterminé (*élu*), tous les autres sont dans un état prédéterminé différent du précédant (*battu*). Le nœud dans un état élu à la fin d'une exécution est appelé *leader* et est dit élu par élection. Le problème de l'élection a été posé et résolu pour la première fois en 1977 par LeLann dans [LeL77] puis par Gallager dans [Gal77]. Nos travaux de recherche que nous avons mené et que nous présentons à partir du Chapitre 3 s'inscrivent dans cette famille d'algorithmes.

1.2 La tolérance aux pannes dans les systèmes distribués

Dans cette section, nous abordons la tolérance aux pannes dans les systèmes distribués.

Le nombre important de composants qui collaborent pour l'accomplissement de la tâche d'un système distribué augmente la probabilité que certains d'entre eux soient sujets à des

dysfonctionnements. Également, un système distribué peut subir des modifications du graphe d'interconnexion. En effet, les nœuds du système peuvent être itinérants, mobiles. Ainsi, des nœuds peuvent se connecter ou se déconnecter à tout moment et des liens de communications peuvent être ajoutés ou supprimés. Tous ces aspects entrainent des fautes qui peuvent conduire à des erreurs dans le déroulement global de l'activité du système. Par conséquent, ces erreurs occasionnent des pannes, encore appelées perturbations ou dysfonctionnements, lors de l'exécution des algorithmes distribués [Lap88].

Il est donc nécessaire qu'un algorithme distribué s'exécutant dans un système distribué offre des mécanismes de résistance à d'éventuels dysfonctionnements pouvant survenir au cours de l'accomplissement de la tâche globale du système ; on parle alors de la tolérance aux pannes. Plusieurs ouvrages traitent de la problématique de la tolérance aux pannes dans les systèmes distribués. Parmi ceux-ci, nous pouvons citer [Lap88, Ray91, AH93, AK98, Tel00].

1.2.1 Notion de panne et tolérance aux pannes

Notion de panne

Dans [Lap88], Laprie définit une panne comme étant « *la défaillance temporaire ou définitive d'un ou de plusieurs composants du système* ». Un composant est défaillant lorsqu'il ne remplit plus ses spécifications requises. Une panne peut être classée suivant son origine, sa cause ou sa durée.

- **Origine de la panne** : elle vient du type de composant responsable de la panne. Elle peut venir des liens de communication qui peuvent subir des modifications (la rupture d'un lien causée par la mobilité des nœuds, l'ajout ou la suppression de nœuds) ou des données traitées par le nœud exécutant le code de l'algorithme distribué qui peuvent être erronées.
- **Cause de la panne** : par omission ou byzantine. Les pannes d'omission regroupent les défaillances des composants alors que les pannes byzantines concernent des comportements anormaux des composants eux-mêmes de sorte qu'ils ne replissent plus les spécifications qui les définissent.
- **Durée de la panne** : panne définitive ou transitoire. La panne est dite définitive (permanente) lorsque sa présence n'est pas reliée à des conditions ponctuelles, internes (traitement d'instructions) ou externes (environnement). Par contre, une panne est dite transitoire (intermittente) lorsqu'elle est présente pour une durée limitée, c'est-à-dire sa durée est inférieur au temps d'exécution de l'algorithme.

Tolérance aux pannes

La tolérance aux pannes est un mécanisme permettant d'assurer le bon fonctionnement du système et de remplir les spécifications requises malgré la présence de dysfonctionnement dans ses composants [Lap88].

Les solutions de tolérance aux pannes proposées se classent en deux familles : les algorithmes robustes qui abordent le problème de tolérance aux pannes selon une approche pessimiste et les algorithmes auto-stabilisants qui, eux, l'abordent selon une approche optimiste.

Les algorithmes robustes assurent en permanence que les spécifications du système, qui sont souvent définies comme étant des invariants, sont vérifiées. En d'autres termes, les algo-

rithmes robustes assurent la continuité des fonctionnalités du système même en présence de perturbations. Dans ce cas, ils sont dits algorithmes masquant les défaillances. Ils tolèrent ainsi des dysfonctionnements frappant continuellement le système. Il s'agit d'une garantie très importante qui est coûteuse et complexe à mettre en œuvre. Une telle garantie est proposée et vérifiée, par exemple, pour les systèmes critiques dont le dysfonctionnement peut mettre en danger la vie d'êtres humains (par exemple, les systèmes de contrôle des avions, les centrales nucléaires, etc.). De plus, ces algorithmes ne tolèrent qu'une classe restreinte de défaillances [Lap88, Ray91, AH93, AK98, Tel00].

Les algorithmes auto-stabilisants tolèrent un dysfonctionnement intermittent dès lors qu'ils assurent au système de retrouver un comportement correct en un temps fini. En d'autres termes, les algorithmes auto-stabilisants ne garantissent pas la continuité du service, mais ils assurent la reprise correcte du service à la fin des défaillances. Dans ce cas, les algorithmes auto-stabilisants sont dits non masquants. Ce type d'algorithmes permet de tolérer toutes les perturbations sur un laps de temps fini mais non borné. À la fin d'une série de perturbations, le système va s'auto-corriger pour reprendre un fonctionnement correct après une phase transitoire durant lequel le système ne fonctionne pas encore correctement (les spécifications de système peuvent ne pas être respectées), bien qu'il n'y ait plus de perturbations [Lap88, Ray91, AH93, AK98, Tel00].

Dans les deux sections qui suivent, nous allons donner les définitions formelles et les caractéristiques des systèmes robustes et auto-stabilisants.

1.2.2 Algorithmes robustes

Dans cette section, nous décrivons les algorithmes robustes. Nous donnons la définition d'un algorithme robuste avant de présenter ses caractéristiques.

Définition des algorithmes robustes

Comme nous l'avons décrit précédemment, les algorithmes robustes sont l'une des solutions apportées au problème de pannes dans les systèmes distribués. Un algorithme robuste est un algorithme qui est capable de continuer son exécution de façon valide malgré l'apparition de pannes dans le système.

Définition 1.10. *Un algorithme est dit robuste lorsque, malgré l'occurrence de pannes dans le système, son comportement ne cesse d'assurer sa spécification.*

Caractéristiques des algorithmes robustes

Dans [MW87], Moran et Wolfsahl ont montré que les algorithmes robustes sont similaires au problème de *décision commune* autrement dit au *problème du consensus* : tous les nœuds du système doivent s'accorder sur le choix d'une valeur binaire et celle-ci doit obligatoirement être la valeur initiale d'au moins l'un des nœuds du système. Toutefois, Fischer et *al.* ont prouvé dans [FLP85] qu'il est impossible d'aboutir à un consensus (prise de décision commune) dans le cas d'un système asynchrone où un seul nœud est susceptible de pannes bénignes définitives. Pour palier ce problème d'insolvabilité d'une terminaison dans le problème de consensus des systèmes asynchrones, trois hypothèses supplémentaires ont été principalement introduites. Dans un premier temps, Fischer et *al.* proposent de considérer qu'un protocole fiable se termine

en un temps fini. Mais cette hypothèse est réfutée dans [BT85] où Bracha et Toueg proposent deux algorithmes qui peuvent ne jamais se terminer. Dans un deuxième temps, Dolev et *al.* proposent dans [DDS87] de considérer un niveau minimal de synchronisation requis pour la résolution du problème consensus. Enfin, dans un troisième temps, Chandra et Toueg ont introduit dans [CT92] la notion de détecteurs de pannes non fiables pour résoudre le problème de consensus.

1.2.3 Algorithmes auto-stabilisants

Ici, nous abordons les algorithmes auto-stabilisants. Nous donnons la définition d'un algorithme auto-stabilisant, ses caractéristiques et les critères de performances à évaluer.

Définition des algorithmes auto-stabilisants

Le concept d'auto-stabilisation a été introduit par Dijkstra en 1974 dans [Dij74].

Définition 1.11. *Quelle que soit sa configuration initiale, un algorithme auto-stabilisant recouvre un comportement correct en un temps fini.*

Dijkstra classe l'ensemble des configurations possibles du système en deux sous-ensembles [Dij74].

1. Configurations légitimes :
   (1.1) Dans chaque configuration légale, au moins une règle est applicable ;
   (1.2) À partir d'une configuration légale, toute action mène à une configuration légale ;
   (1.3) Chaque nœud du système dispose d'une règle applicable dans au moins une configuration légale ;
   (1.4) Pour tout couple de confirmations légales, il existe une succession d'actions menant l'une vers l'autre.
2. Confirmations illégales : toutes configurations autres que légales.

Un état de l'art détaillé sur les différents algorithmes auto-stabilisants a été proposé par Schneider dans [Sch93].

Caractéristiques des algorithmes auto-stabilisants

Dijsktra caractérise l'auto-stabilisant comment étant un algorithme pour lequel : *(i)* toute configuration légale ou non est admise comme pouvant être une configuration initiale correcte, *(ii)* toute exécution dans un tel système contient au moins une configuration légale, *(iii)* une action autorisée dans une configuration légale mène toujours vers un autre configuration légale. De la caractéristique *(i)*, il en découle une propriété fondamentale d'un système l'auto-stabilisant : *l'absence d'initialisation.* En effet, quel que soit l'état initial de ses variables et canaux de communication, il est capable de retrouver une configuration légale après un nombre fini mais non borné d'étapes. Comme illustré dans la figure 1.6 (par exemple, c_0 étant une configuration de départ), le système tend vers un fonctionnement correcte même si initialement aucun nœud ne se trouve dans un configuration légale. Ceci est désigné par la *convergence* et elle est rendue possible par la propriété *(ii)*. Pendant la convergence (phase de stabilisation), le système peut

osciller entre différentes configurations illégales (états $c_0 \rightsquigarrow c_9$) mais finit par atteindre une configuration légale en l'absence de pannes transitoires. De plus, comme illustré toujours dans la figure 1.6 ($c_{10} \rightsquigarrow c_{16}$), une fois la phase stabilisée atteinte, l'auto-stabilisation assure un fonctionnement correcte en l'absence de panne transitoire et passe d'une configuration légale à une autre. Ceci est assuré par la propriété *(iii)* appelée *clôture*. Cependant, suite à l'occurrence d'une panne transitoire, le système peut potentiellement basculer dans une configuration illégale ($c_{16} \rightsquigarrow c_{17}$) et traverse ainsi une nouvelle phase de stabilisation au cours de laquelle les spécifications du système ne sont plus respectées ($c_{17} \rightsquigarrow c_{22}$). Le retour à une configuration légale se fait, par définition, au bout d'un nombre fini d'étapes.

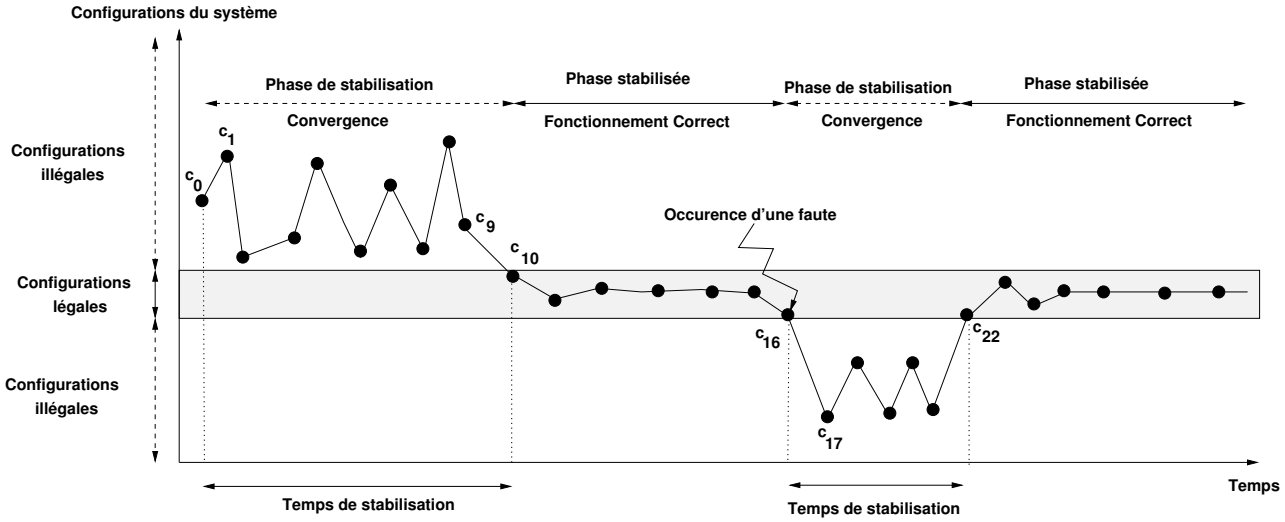


FIGURE 1.6 – Exécution d'un algorithme auto-stabilisant

L'auto-stabilisation a été donc définie pour résister aux pannes transitoires. Celles-ci sont provoquées soit par des ruptures de liens de communication soit par des corruptions de données. Les ruptures des liens de communication peuvent être causées par la mobilité de nœuds, l'ajout ou la suppression de nœuds. Les corruptions de données concernent l'état de la mémoire locale d'un nœud, des messages ou de données échangés entre nœuds durant l'exécution de l'algorithme. Dans le cas où l'algorithme auto-stabilisant est écrit suivant un modèle à états, la corruption de données concernent uniquement celles contenues dans la mémoire de chaque nœud alors que pour un modèle à registres, cette corruption concerne les données présentes dans les registres qui peuvent subir des modifications chaotiques. Par contre, pour un modèle à passage de messages, la corruption concerne aussi bien l'état des messages. Soit par la modification du contenu de ces derniers les rendant erronés ou inexploitables, soit par la destruction d'un message durant son transfert ou sa réplication.

Performance des algorithmes auto-stabilisants

La validation d'un algorithme auto-stabilisant consiste à montrer que les propriétés de convergence et de clôture sont vérifiées. L'évaluation de ses performances consiste à déterminer le temps de convergence (temps de stabilisation) ainsi que l'occupation mémoire nécessaire à son exécution.

Montrer la propriété de convergence d'un algorithme auto-stabilisant revient également à trouver son temps de stabilisation. Dans [DIM91, DIM97], Dolev et *al.* ont montré que pour la plupart des algorithmes auto-stabilisants fondés sur un modèle à passage de messages, notamment ceux utilisant la circulation de jeton, il est souvent difficile voire même impossible de trouver la borne minimale du temps nécessaire pour atteindre une configuration légale. Ainsi, trouver le temps de stabilisation consiste à déterminer au bout de combien de temps au plus (borne maximale) une configuration légale est atteinte en partant d'une configuration quelconque. Prouver la clôture consiste à montrer que toute exécution à partir d'une configuration légale et sans occurrence de pannes transitoires mènera toujours vers une configuration légale. Soit $\mathcal{A}$ un algorithme auto-stabilisant suivant une exécution d'un système de transitions et soit $\mathcal{L}_c$ l'ensemble de toutes les configurations légales de $\mathcal{A}$, prouver formellement la convergence et la clôture de $\mathcal{A}$ consiste à vérifier les deux propriétés suivantes [CDDL10] :

1. **Convergence** : une configuration légale δ est atteinte depuis une configuration γ, notée $\gamma \rightsquigarrow \delta$, s'il existe une exécution finie $\mathcal{E}$ telle que : $\gamma = \gamma_0, \gamma_1, \gamma_2, \cdots, \gamma_k = \delta$ avec $\gamma_i \mapsto \gamma_{i+1}$ pour tout $0 \leq i < k$.
2. **Clôture** : si une exécution $\mathcal{E}$ démarre par une configuration γ membre de $\mathcal{L}_c$ alors, sans occurrence de pannes transitoires, toute configuration γ_i de $\mathcal{E}$ est aussi membre de $\mathcal{L}_c$.

Montrer l'occupation en mémoire d'un algorithme auto-stabilisant consiste à déterminer tous les états possibles pour chaque nœud du système ou bien déterminer la taille (en bits) nécessaire pour stocker l'ensemble des valeurs possibles de toutes les variables utilisées localement par un nœud.

1.2.4 Discussions : Robustesse vs Auto-stabilisation

Les approches robustes proposent une solution de tolérance aux pannes et garantissent que le comportement global du système respecte les spécifications du but à atteindre. Les pannes considérées sont des pannes définitives ou des pannes byzantines. Par contre, elles n'offrent pas de solution au problème de l'apparition ou de l'ajout de nouveaux nœuds dans le système. En effet, si un nouveau nœud est ajouté au système, son état peut ne pas forcement respecter la spécification requise et l'état global du système qui en découle peut ne pas être correct. Ainsi, comme les algorithmes robustes n'apportent pas de réponse à l'ajout de nouveaux nœuds dans le système, ils ne sont donc pas adaptés aux systèmes dynamiques.

Les solutions auto-stabilisantes ne nécessitent aucune initialisation et s'adaptent aux systèmes dynamiques. En effet, elles sont capables de gérer l'apparition ou la disparition de nœuds dans le système. Pour ce faire, elles supposent toute modification de la topologie comme une altération du système qui mène fatalement vers un état illégitime, et par définition, un nouvel état sera retrouvé au bout d'un temps fini. En outre, contrairement aux solutions robustes, les approches auto-stabilisantes ne requièrent pas la détection de la panne. En d'autres termes, elles n'utilisent pas de mécanisme de détection de pannes pour évaluer l'état global du système. Pour cette raison, elles sont sans terminaison. Il en découle une exécution continuelle et il n'est pas nécessaire de détecter une panne pour la corriger. Cependant, les algorithmes auto-stabilisants

considèrent toute altération du système comme étant une faute pouvant mener vers une configuration illégale. La principale faiblesse de tels algorithmes réside durant l'apparition de ces cas de situation. Ils sont vulnérables aux périodes de perturbations et donc difficilement utilisables pour des d'applications « sensibles » ou sujettes à des fautes de fréquence « élevées ». Si la fréquence des pannes est très grande, l'algorithme peut ne jamais se stabiliser.

Notons que des solutions visant à combiner la robustesse et l'auto-stabilisation ont été proposées. Cette combinaison, bien que difficile et induisant des coûts supplémentaires, apporte des solutions qui sont pleinement dynamiques et ne nécessitant aucune initialisation. De plus, elles sont capables de résister aux pannes transitoires et définitives tout en assurant un comportement respectant la spécification requise malgré une disparition d'un nœud ou d'un lien de communication. Parmi ces solutions connues sous le nom de FTSS (Fault-Tolerant Self-Stabilizing), nous pouvons citer [AG93, AH93, CT92].

1.3 Les réseaux ad hoc

Dans cette section, nous abordons les réseaux ad hoc qui sont un type particulier de système distribué [Ily02, CCL03, Agg04, BK07a, SBP11, Fri13] et qui représentent le centre d'intérêt de nos travaux de recherche. Nous nous intéressons à ces types de réseaux pour leurs propriétés et les nombreux avantages qu'ils offrent. C'est pourquoi nous allons les présenter en détails.

1.3.1 Réseaux avec infrastructure vs Réseaux sans infrastructure

Dans les réseaux informatiques, nous distinguons deux grandes familles de réseaux : (i) les réseaux avec infrastructure et (ii) les réseaux sans infrastructure.

Les réseaux avec infrastructure

Dans les réseaux avec infrastructure, il existe une entité centrale, appelée coordinateur ou station de base qui a pour rôle d'orchestrer toute l'activité du réseau [Hec07, IMM08, Ini09].

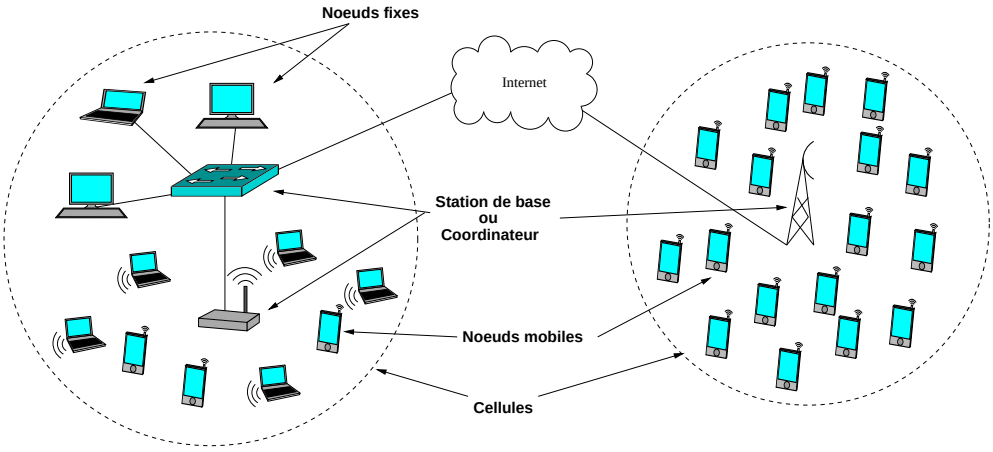


FIGURE 1.7 – Réseaux avec infrastructure

Comme illustré dans la figure 1.7, les réseaux avec infrastructure sont partitionnés en cellules, encore appelées zones. Chaque zone dispose d'un coordinateur appelé station de base qui joue le rôle de l'entité centrale. Toutes les communications passent par la station de base. Les nœuds peuvent être fixes ou mobiles, connectés par une interface filaire ou sans fil. La particularité de cette famille de réseau est que toutes les communications passent obligatoirement par la station de base. Le système de communication est fondé essentiellement sur l'utilisation de réseaux filaires mais aussi de station de base, avec une longue portée, pour couvrir l'ensemble des nœuds mobiles. Ainsi, les réseaux avec infrastructure nécessitent une importante logistique pour leur déploiement.

Les réseaux sans infrastructure

Dans les réseaux sans infrastructure, il n'existe aucune entité centrale [Ily02]. Les nœuds communiquent directement entre eux. L'absence d'infrastructure fixe oblige les nœuds du réseau à participer à la découverte et à la maintenance des routes dans le but d'acheminer les informations. Un réseau sans infrastructure peut être constitué de nœuds mobiles avec des interfaces sans fil. Ainsi, comme illustré dans la figure 1.8, tout nœud peut communiquer avec l'ensemble des autres nœuds se trouvant dans sa zone de couverture (portée radio). Les nœuds du réseau peuvent être de même type, dans ce cas on parle de réseau homogène. Mais aussi le réseau peut être constitué de nœuds de types différents. Dans ce cas, le réseau est dit hétérogène.

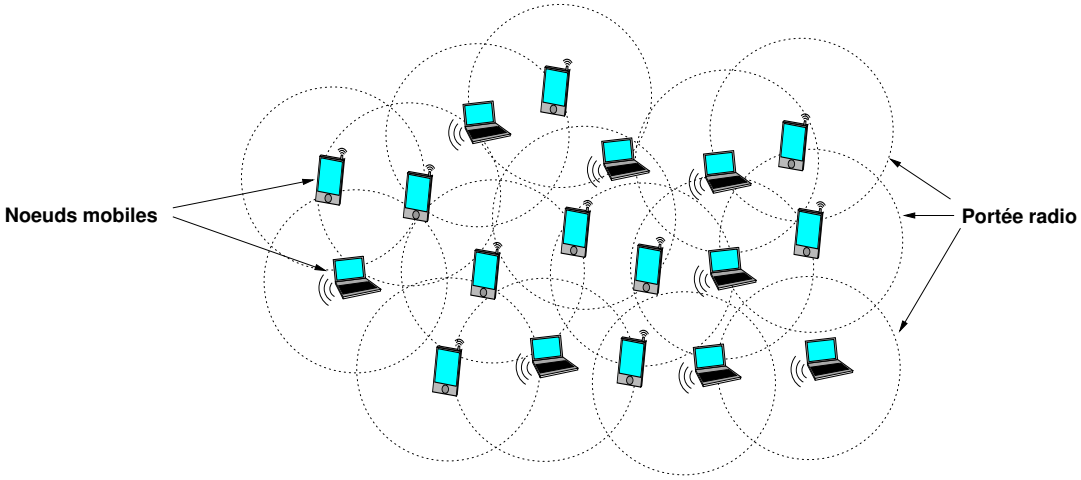


FIGURE 1.8 – Réseaux sans infrastructure

Les réseaux sans infrastructure sont plus contraignants que les réseaux avec infrastructure. En effet, la mobilité des nœuds conduit à de fréquents changements topologiques, d'où la nécessité de recalculer fréquemment les tables de routage. De plus, les nœuds d'un réseau sans infrastructure peuvent avoir des ressources limitées (capacité de calcul, de stockage et énergétique). Toutes ces contraintes rendent difficile l'établissement de protocoles de communication.

1.3.2 Spécificités des réseaux ad hoc

Les réseaux ad hoc sont des réseaux sans infrastructure [Ily02, CCL03, Agg04, BK07a, SBP11, Fri13]. Ce sont des systèmes distribués complexes regroupant un important nombre de nœuds qui peuvent être mobiles et équipés d'interfaces sans fil pour les communications. Dans le cas où les nœuds sont tous mobiles et munis d'interfaces de communication sans fil, on parle alors de réseau MANET (*Mobile Ad hoc NETwork*).

Différentes techniques de communication

Dans les réseaux ad hoc, il existe plusieurs techniques de communication : le *broadcast*, le *multicast* ou l'*unicast* [Ily02]. Dans le *broadcast*, encore dit de technique de la diffusion, un message envoyé par un nœud est reçu par tous ses voisins. Dans une communication en *multicast*, un message envoyé par un nœud est reçu par un groupe de nœuds parmi ses voisins. Et dans la technique de *unicast*, encore dite communication point à point, un message envoyé par un nœud est destiné à un seul de ses voisins. Cependant, dans chacune de ces différentes techniques, les communications entre les nœuds ne nécessitent aucune infrastructure fixe. De ce fait, les réseaux ad hoc représentent une alternative aux réseaux avec infrastructure, notamment pour des applications qui requièrent un déploiement dynamique du réseau.

Caractéristiques des réseaux ad hoc

Les réseaux ad hoc présentent plusieurs spécificités qui rendent difficile l'établissement des protocoles de communication. Parmi ces spécificités, nous pouvons citer :

- **La volatilité du médium de communication** : la communication dans un environnement sans fil diffère de celle des réseaux câblés (réseau avec infrastructure) en deux points. D'une part, les ondes radio ne sont pas "canalisables" comme dans un médium câblé, elles ont donc un comportement variable et imprévisible. D'autre part, le médium sans fil est un médium de diffusion, c'est-à dire, tous les nœuds qui sont dans la portée de transmission de l'émetteur peuvent recevoir le message [Ily02, CCL03, BK07a, Fri13].
- **La topologie dynamique** : la topologie d'un réseau ad hoc est dynamique. Les nœuds peuvent se déplacer de façon libre et arbitraire. Ainsi, un nœud peut se connecter ou se déconnecter du réseau à tout instant. Par conséquent, la topologie du réseau peut changer à des instants imprévisibles, d'une manière rapide et aléatoire. Cela affecte fortement la disponibilité des chemins de routage, et les protocoles de routage doivent s'adapter à la mobilité des nœuds [Ily02, CCL03, SBP11, Fri13].
- **La bande passante limitée** : dans un réseau ad hoc, la bande passante est limitée. En effet, tous les nœuds du réseau se partagent le médium de communication pour les échanges et routages d'informations. Ce partage fait que la bande passante réservée à un hôte est réduite [CCL03, Agg04, Fri13].
- **L'hétérogénéité des nœuds** : les nœuds ad hoc peuvent correspondre à une multitude d'équipements, par exemple des ordinateurs portables, des PDA, téléphones mobiles, etc. Ainsi, ces nœuds peuvent avoir des différences en termes de capacité de traitement (CPU, mémoire), de mobilité (lent, rapide) et de logiciel, etc. Cependant, quelle que soit l'hétérogénéité des nœuds, ils doivent être en mesure de communiquer et de coopérer pour accomplir les tâches requises dans le réseau [Ily02, CCL03, Fri13].

1.3.3 Le cas des réseaux de capteurs sans fil (RCSF)

Après avoir présenté les réseaux ad hoc, nous abordons dans cette section les réseaux de capteur sans fil, leurs caractéristiques et leurs contraintes.

Concept de réseaux de capteurs

Les réseaux de capteurs sans fil sont un type particulier de réseau ad hoc [ASSC02, AY07, KW07, DMCA06, SDT07, Chi08, Dre08, LP09, XCL10, Sha11]. Ils consistent en un grand nombre de dispositifs, déployés dans une zone géographique (zone d'étude), dotés de la capacité de collecter des informations (telles que température, lumière, pression, etc.) sur l'environnement physique, de faire des traitements sur les données recueillies puis de coopérer entre-eux pour leur acheminement vers un centre de contrôle. Ces dispositifs sont appelés *capteurs* et ont connus un essor important dû aux avancées récentes dans la technologie de la micro-électromécanique (Micro Electro-Mechanical Systems (MEMS)), de la micro-électronique et des communications sans fil.

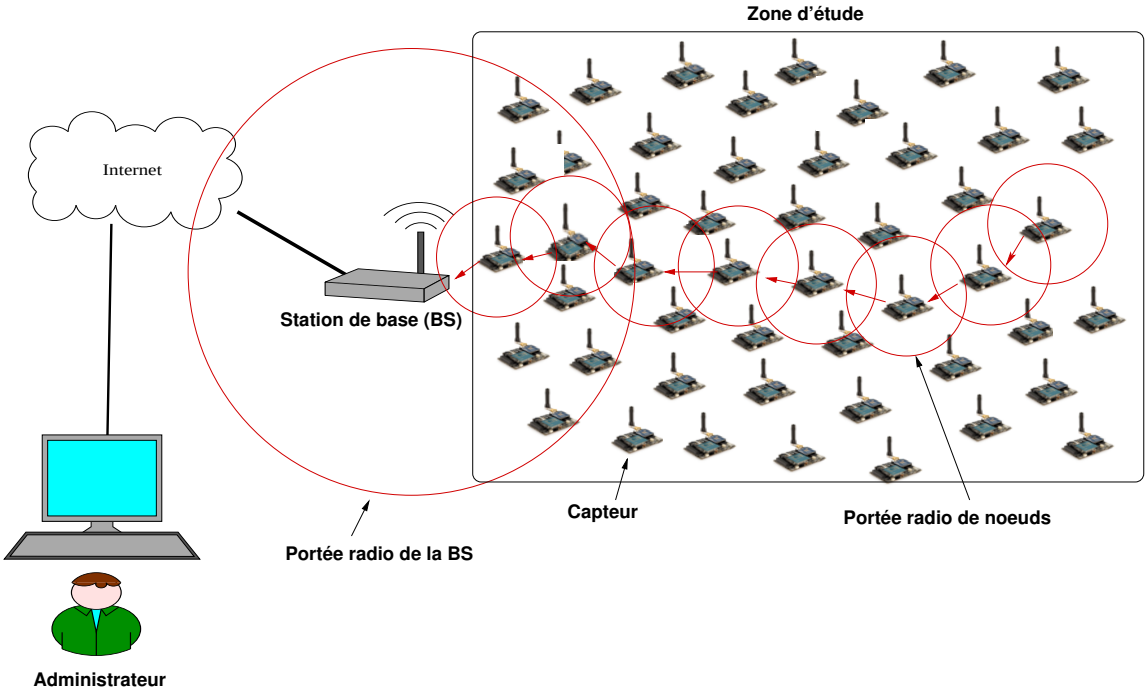
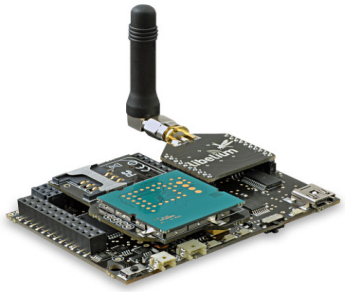


FIGURE 1.9 – Réseaux de capteurs sans Fil

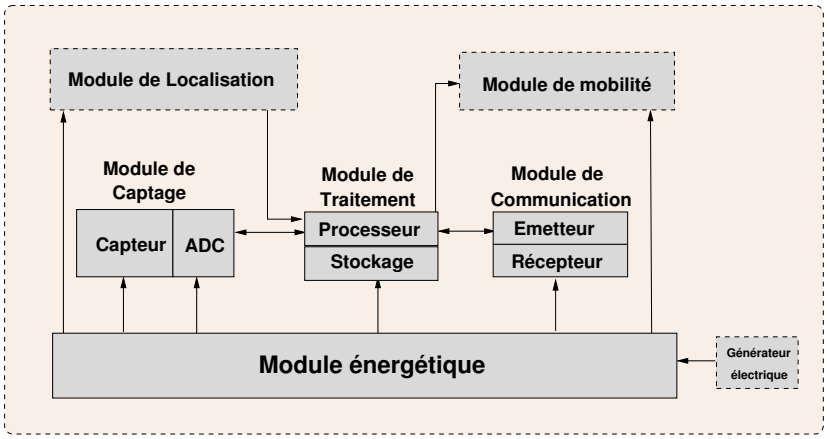
Dans un réseau de capteurs, comme illustré dans la figure 1.9, les données collectées par les nœuds capteurs sont acheminées, avec un routage multi-sauts de proche en proche, vers un centre de collecte appelé station de base (*Base Station* (BS)). Cette dernière est aussi désignée sous le nom de Puits (*Sink*) et est connecté via Internet ou par satellite à un centre de commande où un administrateur visualise et contrôle l'activité du réseau.

Composition d'un capteur

Il existe plusieurs types de capteurs fournis par divers constructeurs. Nous pouvons citer l'exemple des capteurs de type MICA, TelosB, Iris, Cricket, etc. [MEM13]. La figure 1.10(a) illustre un capteur de type Waspmote de la marque Libelium [Lib13].



(a) Nœud de type Waspmote de la marque Libelium



(b) Nomenclature d'un capteur

FIGURE 1.10 – Composants d'un capteur

Dans [ASSC02], Akyildiz et *al.* ont illustré un schéma simplifié d'un capteur. Comme montré dans la figure 1.10(b), un capteur est composé de quatre (4) modules principaux.

- **Module de captage** : il est composé de deux unités : l'unité de captage et l'unité ADC (*Analog-to-Digital Converter*). L'unité de captage a pour fonction de mesurer une grandeur physique à observer sur l'environnement d'étude. L'unité ADC est un convertisseur analogique-numérique et a pour rôle de transformer la grandeur physique captée par l'unité de captage en signaux numériques traitables par l'unité de traitement. Par exemple, les nœuds de type IRIS sont dotés d'un module de captage de type MTS400 pouvant mesurer la température, la pression, l'humidité et les secousses sismiques.
- **Module de traitement** : il a pour rôle d'effectuer les traitements requis par les autres modules ainsi que ceux nécessaires pour la collaboration avec les autres nœuds capteurs. Par exemple, les nœuds de type MICA MPR2400CB sont dotés d'un processeur Atmel ATMega 128L, 8bit à 8 - 16 MHz.
- **Module de communication** : il assure les échanges de messages entre les nœuds du réseau. Il envoie les données collectées par le nœud dans le réseau mais aussi relaye les messages des autres nœuds du réseau. Par exemple, les nœuds de type TelosB sont munis d'une antenne qui peut émettre dans la bande 2.4 - 2.48 GHz ISM [MEM13]
- **Module énergétique** : il constitue la réserve énergétique du nœud et alimente les autres modules en énergie nécessaire pour leur bon fonctionnement. Vu, la taille réduite des capteurs, ils sont souvent équipés de batterie avec une faible quantité énergétique. Par exemple, les nœuds de type Waspmote sont équipés de batteries 2X AA de voltage compris entre 3.3 V - 4.2V [Lib13].

Un capteur peut aussi être équipé de trois (3) modules optionnels. Le module de localisation

permet au nœud de connaitre sa localisation. Il s'agit souvent d'un module GPS (Global Positioning System). Le module de mobilité permet d'assurer le déplacement du capteur dans la zone d'étude. Le capteur peur être également équipé d'un générateur énergétique afin recharger la batterie.

Les nœuds capteurs, vu leurs faibles capacités de stockage (par exemple, les nœuds de type MICA MPR2400CB disposent de $4KB$ en RAM) ainsi que de traitement, ne peuvent supporter que des systèmes d'exploitation « légers ». Pour tenir compte de ces contraintes, plusieurs systèmes d'exploitation ont été proposés dans la littérature. Parmi ces derniers, nous pouvons citer TinyOS [HSW+00] ou MANTIS OS [BCD+05]. Un état de l'art sur les systèmes d'exploitation proposés pour les capteurs a été proposé dans [FW08].

De même, la norme 802.15.4 TG4 a été standardisée par IEEE afin de prendre en compte les faibles ressources des capteurs dans l'établissement des protocoles de communications [Gro14]. Le protocole Zigbee, fondé sur la norme IEEE 802.15.4 TG4, a été proposé pour les réseaux de type LR WPAN (*Low Rate Wireless Personal Area Network*) c'est-à-dire pour des transmissions radio à faible consommation avec une transmission de données à faible débit (250 Kbit/s) sur une faible étendue [Far08, Zig13].

Parmi les quatre modules principaux d'un capteur que nous avons présentés ci-dessus, nous nous focalisons sur le module de communication dans l'étude consommation énergétique tel que nous le présentons dans le Chapitre 4. En effet, les communications sont considérées comme étant la principale source de consommation énergétique [PK00, HCB00].

Applications de réseaux de capteurs

Les RCSF, du fait de la taille réduite des capteurs et leur faible coût de production, de l'absence d'infrastructure fixe, etc. offrent de nombreuses applications pratiques. Les opérations tactiques comme les opérations de secours, militaires ou d'explorations de zones hostiles trouvent dans les réseaux de capteur un candidat efficace. La technologie des capteurs intéresse également la recherche d'applications civiles.

- **Applications militaires** : l'absence d'infrastructure fixe et la rapidité de déploiement sont entres autres deux aspects qui font des RCSF une solution efficace pour la mise en place de réseaux militaires dans les théâtres d'opérations ;
- **Les services d'urgence** : Dans le cas de tremblement de terre, de feux de forêts, d'inondation, de secours de personnes, etc. où la mise en place d'infrastructure demande un temps énorme ou est même quasi impossible, les réseaux de capteurs offrent une solution alternative pour une mise en place et un déploiement rapide ;
- **Applications médicales** : Dans le domaine médical, les réseaux de capteurs peuvent être utilisés pour assurer une surveillance permanente ou temporaire des organes vitaux grâce des micro-capteurs qui peuvent être avalés ou implantés sur le patient ;
- **Applications environnementales** : Pour la surveillance de zones tempérées, des profondeurs océaniques, de la qualité de l'air atmosphérique, le suivi des migrations des animaux, les réseaux de capteurs offrent d'intéressantes applications ;
- **Application domestiques** : Les réseaux de capteurs présentent aussi des applications pratiques pour la surveillance et le contrôle d'équipement domestiques à distance tel que les appareils de chauffage.

Topologie des réseaux de capteurs

Le déploiement des réseaux de capteurs peut suivre des topologies régulières ou quelconques. Dans le cas de topologie quelconques, plusieurs travaux, comme ceux proposés dans [BC03, HLS05, MFGLT05, CKF10, NQL11], ont montré que des graphes aléatoires suivant une loi de Poisson offrent une meilleure représentation de la distribution des nœuds dans un RCSF. Ainsi, dans le Chapitre 4, nous considérons de pareilles topologies réseaux dans l'étude des RCSF. Dans un réseau suivant une loi Poisson homogène d'intensité $\lambda > 1$, chaque nœud u a un nombre de voisins aléatoire δ prenant la valeur k une probabilité $\mathbb{P}$ telle que :

$$\mathbb{P}(\delta = k) = \frac{\lambda^k}{k!} \exp(-\lambda) \tag{1.1}$$

Dans l'équation 1.1, le paramètre λ de la loi de Poisson représente alors le degré moyen du réseau. Il est également supposé que la loi de Poisson est homogène (intensité constante dans un plan 2D) et isotrope (propriété invariante par rotation). Les coordonnées (x_u, y_u) de chaque nœud u dans le plan $2D$ sont définies par des nombres pseudo-aléatoires tirés d'une loi uniforme. Un lien de communication existe entre deux nœuds u et v si la distance $dist(u, v)$ entre ces deux nœuds est inférieure ou égale à la porté r de la communication ($dist(u, v) \leq r$).

Modèle énergétique pour le module radio

Dans les RCSF, le modèle énergétique de référence a été proposé par Heinzelman et *al.* dans [HCB00]. Actuellement, c'est le modèle le plus couramment utilisé dans les travaux de recherche sur la consommation énergétique des capteurs. Pour cette raison, nous utilisons ce modèle dans le Chapitre 4 lors de nos mesures énergétiques.

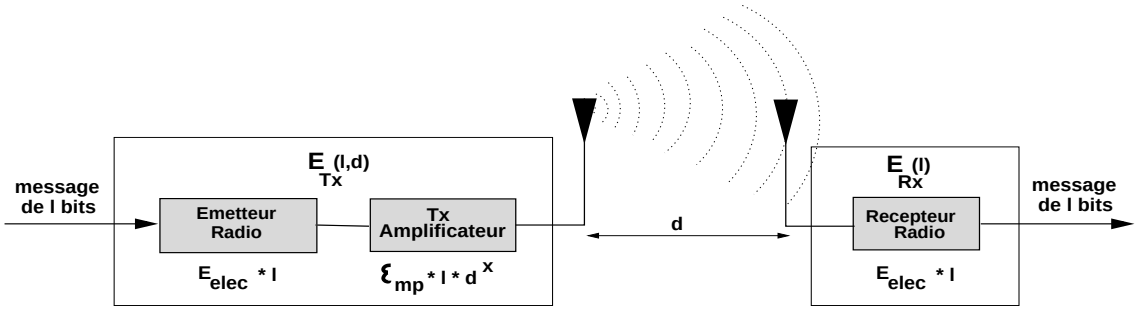


FIGURE 1.11 – Schéma du module radio de Heinzelman et *al.* [HCB00]

Heinzelman *et al.* décrivent un modèle radio, illustré dans la figure 1.11, où un nœud capteur consomme une quantité d'énergie E_{Tx} pour transmettre un message de longueur l bits sur une distance d (en mètre). Comme décrit dans l'équation 1.2, quand la distance dépasse un certain seuil d_0, le modèle de consommation est diffèrent car l'énergie nécessaire est plus importante.

$$E_{Tx}(l, d) = \begin{cases} l \times E_{elec} + l \times \varepsilon_{fs} \times d^2, & \text{si } d < d_0; \\ \\ l \times E_{elec} + l \times \varepsilon_{mp} \times d^4, & \text{si } d \geq d_0. \end{cases} \tag{1.2}$$

A chaque réception de message, un nœud consomme une quantité d'énergie E_{Rx} comme illustré dans l'équation 1.3.

$$E_{Rx}(l) = l \times E_{elec} \tag{1.3}$$

Les valeurs des paramètres utilisés dans les équations 1.2 et 1.3 sont mentionnées dans le tableau 1.1. E_{elec} représente l'énergie nécessaire pour activer les circuits électroniques du module radio lors des communications au niveau de l'émetteur et du récepteur. En plus de cette quantité E_{elec}, l'émetteur consomme une énergie supplémentaire dépendant de la longueur l du message et de la distance de transmission d.

Paramètre	définition	Unité
E_{elec}	Énergie pour faire marcher la radio	$50nJ/bit$
ε_{fs}	Modèle d'espace libre de l'amplificateur de l'émetteur	$10pJ/bit/m^2$
ε_{mp}	Modèle multi-path de l'amplificateur de l'émetteur	$0.0013\ pJ/bit/m^4$
l	Taille d'un message	2000 bits
d_0	Seuil de distance	$\sqrt{\varepsilon_{fs}/\varepsilon_{mp}}$

TABLE 1.1 – Paramètres du module radio

1.3.4 Contraintes dans les réseaux de capteurs sans fil

En plus d'hériter de toutes les spécificités des réseaux ad hoc, les réseaux de capteurs présentent deux contraintes fondamentales : la faible capacité énergétique et la vulnérabilité aux pannes.

Contrainte énergétique

Comme nous l'avons dit précédemment, du fait de sa taille réduite, un capteur est doté de batterie avec une quantité énergétique très limitée. De plus, l'hostilité qui caractérise souvent les zones de déploiement fait qu'il est difficile, voire même impossible de recharger ou de changer la batterie des capteurs. À cela s'ajoute que l'énergie d'un capteur est consommée principalement par trois opérations ; le captage de données, les traitements CPU internes et les communications. Néanmoins, les communications sont la source majeure de la consommation énergétique par rapport aux instructions CPU pour les traitements internes et les opérations de captage. En effet, dans [PK00], Pottie et Kaiser ont montré que la consommation énergétique requise pour la transmission d'un message de $1KB$ sur une distance de 100 mètres est équivalente à 3 millions d'instructions CPU dans un processeur 100 $MIPS/W$.

Or, avec toutes ces contraintes, il s'avère logique que les techniques de communication classiques ne s'adaptent pas aux réseaux de capteurs. Ainsi, vu la nature critique et sensible des applications qu'offrent les RCSF d'une part et des faibles capacités énergétiques des capteurs d'autre part, il est plus que nécessaire de minimiser les communications dans le but d'optimiser la consommation énergétique afin de prolonger au maximum la durée de vie du réseau. Pour réduire les communications, une solution efficace consiste à structurer le réseau [EWB87, AKY91, Bas99, BCV03, DLV08, BPBR11].

Vulnérabilité aux pannes

Optimiser seulement la consommation énergétique ne suffit pas dans les RCSF. Une autre spécificité essentielle dans les réseaux de capteurs est la vulnérabilité aux pannes. En effet, les nœuds capteurs d'un RSFC sont susceptibles de tomber en panne du fait de l'épuisement total de leurs énergies ou du dysfonctionnement d'un de leurs composants. Également la mobilité des nœuds peut entrainer des ruptures de liens de communication conduisant ainsi à des changements topologiques.

Donc, comme démontré dans plusieurs études (Liu et *al.* [LWJ05], Zhang et *al.* [WXM07], Hao et *al.* [BJX04]), la tolérance aux pannes transitoire est une problématique capitale dans les RCSF. Dans [LNS09], Hai Liu et *al.* définissent la tolérance aux pannes dans les RCSF comme étant la capacité à assurer le service requis même en présence de pannes transitoires dans le réseau. Dans un système distribué comme les RCSF, Johnen et Mekhaldi ont montré dans [JM11] que, l'auto-stabilisation que nous avons présenté dans la Section 1.2.1 est une solution efficace de tolérance aux pannes transitoires. De plus, Drabkin et *al.* [DFG06] ont présenté un modèle de fautes qui peut occasionner des pannes transitoires dans les réseaux de capteur sans fil et que l'auto-stabilisation peut gérer efficacement. Ces fautes sont les suivantes :

1. l'état d'un nœud ou la configuration du système peut subir des corruptions quelconques. Néanmoins, le programme (code) de l'algorithme est supposé sûr ;
2. tout nœud du réseau peut tomber à l'arrêt total ;
3. les canaux de communications peuvent subir des corruptions quelconques et contenir des données erronées ;
4. la topologie du réseau peut changer à tout instant : mobilité de nœud, ajout ou rupture de liens de communication.

Cependant, l'occurrence de ces fautes est finie mais non bornée. Elle peut suivre n'importe quel ordre, n'importe quelle fréquence et peut survenir à n'importe quel moment.

Pour toutes les raisons que nous venons d'évoquer dans cette section, la réduction des communications et la tolérance aux pannes transitoires dans les réseaux ad hoc (et par extension dans les réseaux de capteurs) ont inspiré bon nombre de chercheurs ces dernières années. C'est dans ce que contexte que s'inscrivent les travaux de recherche que nous avons menés et que nous présentons dans ce document.

1.4 Conclusion

Dans ce chapitre, nous avons commencé par poser les bases de nos travaux de recherche que sont les systèmes distribués en présentant sa représentation par les graphes, son exécution par un système de transitions et ses différents modèles de communication, à savoir, le modèle à états, à registres ou à passages de messages. Ensuite, nous avons présenté brièvement quelques algorithmes classiques en systèmes distribués puis nous avons décrit deux approches de tolérances aux pannes que sont les algorithmes robustes et les algorithmes auto-stabilisants. Enfin, nous avons abordé les réseaux ad hoc qui nous intéressent particulièrement dans notre travail de recherche. Après avoir présenté ce type de réseau et soulevé ses spécificités, nous avons introduit

les réseaux de capteurs sans fil (RCSF). Ces derniers présentent deux contraintes fondamentales qu'il faut tenir en compte pour satisfaire les exigences applicatives : (*i*) l'énergie limitée des capteurs et (*ii*) les pannes transitoires. Pour gérer ces deux contraintes, dans le but d'améliorer les performances du réseau et de prolonger sa durée de vie, les chercheurs se sont penchés massivement sur la structuration et l'auto-stabilisation dans les réseaux ad hoc.

Ainsi, dans le prochain chapitre, nous étudierons les différentes approches de structuration, auto-stabilisantes comme non auto-stabilisantes, proposées dans la littérature.

Chapitre 2

Structuration des réseaux ad hoc

Résumé : *Dans ce chapitre nous abordons la structuration des réseaux ad hoc en clusters et nous donnons une classification des solutions de structuration existantes dans la littérature. Ensuite, nous étudions les solutions de structuration non auto-stabilisantes puis nous ferons de même pour leurs homologues auto-stabilisantes. Nous clôturons cette étude de l'existant en soulevant notre problématique de recherche.*

Sommaire

2.1 Motivations et objectifs

Dans un réseau ad hoc, comme il n'existe pas d'infrastructure fixe qui coordonne les communications, tout nœud du réseau participe à l'acheminement de l'information. Devant cette situation, la solution de communication la plus courante est la diffusion [1]. Cette approche a été initialement introduite par Chang dans [Cha82] et est connue sous le nom de l'algorithme

1. *Broadcast*

d'echo. C'est une technique simple et facile à mettre en œuvre. Elle consiste, comme illustré dans la figure 2.1, à inonder le réseau : un nœud Source **(S)** qui souhaite communiquer avec un nœud Destinataire **(D)** diffuse son message dans son voisinage. Puis, chaque nœud qui reçoit le message pour la première fois le diffuse à son tour. Et cela, jusqu'à ce que le message atteigne la destination.

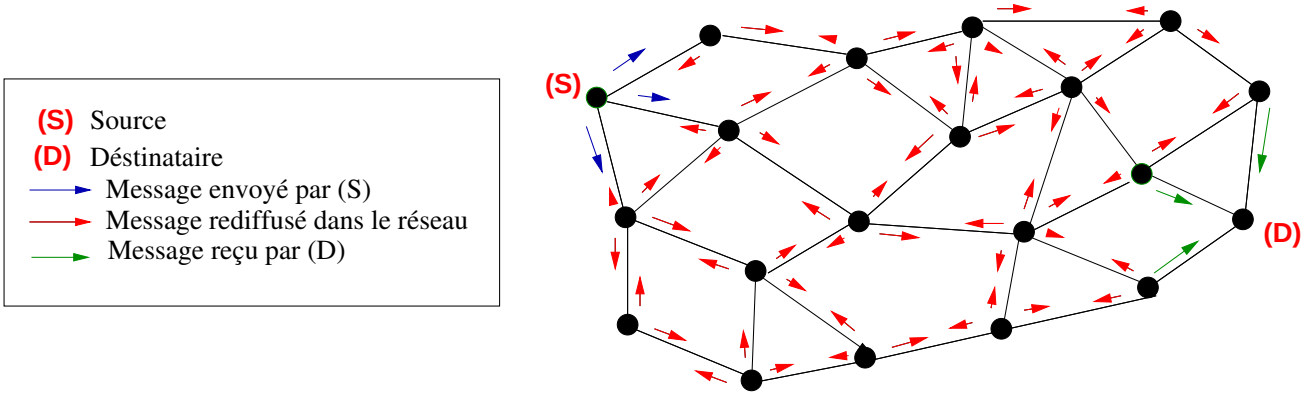


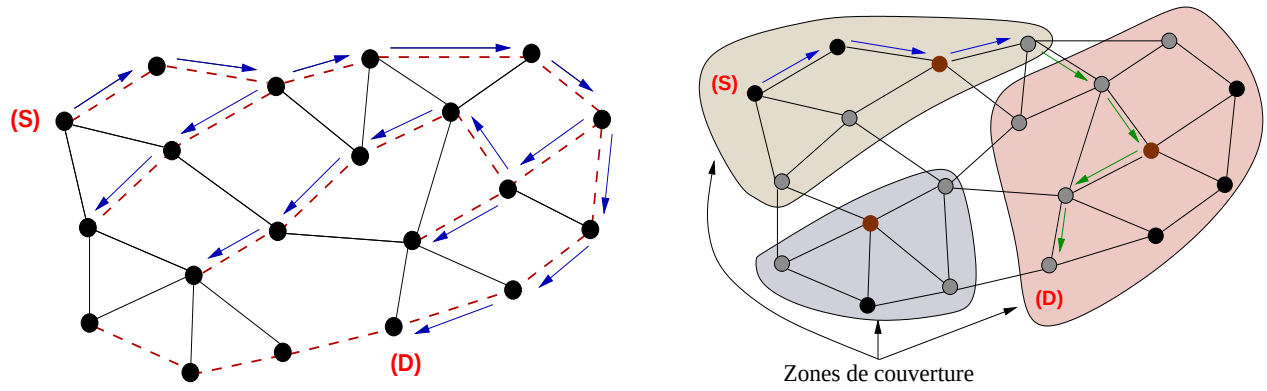
FIGURE 2.1 – Communication par la technique de la diffusion

Cependant, la technique de diffusion conduit à une explosion du nombre de messages qui transitent dans le réseau. En effet, elle a un coût en messages de $2m$, avec m étant le nombre de liens de communication dans le réseau. Ce qui entraine donc la saturation ainsi que la surcharge du réseau. Afin de réduire les communications dans le but de minimiser la consommation des ressources du réseau, une autre solution consiste à structurer le réseau en arbre [AKY91, Lav00, BFG+03, BCV03, BPBR11] ou en *clusters* [EWB87, Bas99, JN06, DLV08, OHN09].

La structuration en arbre d'un réseau quelconque connexe consiste à construire un arbre couvrant [2] sur le réseau de sorte qu'il n'existe pas de boucle dans l'arbre et qu'il existe un chemin entre n'importe quel couple (u, v) de nœuds du réseau en passant par l'arbre couvrant [AKY91, BLB95, HL01, Gae03, BK07b]. En d'autres termes, tout nœud du graphe est voisin d'au moins un nœud membre de l'arbre couvrant. Dans l'exemple de la figure 2.2(a), l'arbre couvrant est illustré par les arêtes en pointillées matérialisées par la couleur rouge. La structuration en arbre permet de limiter les communications en diffusant les messages le long des chemins de l'arbre depuis une source **(S)** jusqu'à une destination **(D)**.

La structuration en *clusters* consiste à construire sur un réseau quelconque connexe, des zones (groupes) de couverture contenant un certain nombre de nœuds qui partagent une même politique de communication. Dans ces zones, les nœuds peuvent avoir des rôles différents afin de bien orchestrer les communications. La structuration en *clusters*, permet de limiter les communications en effectuant des communications intra et/ou inter zones. Plusieurs travaux comme [EWB87, Bas99, HCB00, YF04, OHN09, YQWG12] ont montré que la structuration en *clusters* est plus efficace et permet une meilleure réduction des communications.

2. *Spanning tree*



(a) Structuration en arbres (b) Structuration en *clusters*

FIGURE 2.2 – Structuration en arbres ou en *clusters*

Les réseaux ad hoc peuvent être également dynamiques. Un nœud peut se connecter ou se déconnecter du réseau à tout instant et peut donc induire des pannes transitoires dans le réseau, causées par des modifications topologiques. De plus, les variables qui définissent les états des nœuds peuvent contenir, à l'initialisation comme lors des communications, des données erronées. Il en est de même pour les canaux de communication. Par conséquent, la conception de solutions tolérantes aux pannes transitoires est plus que nécessaire pour des applications de réseaux ad hoc souvent critiques [LWJ05, WXM07, BJX04]. Comme nous l'avons étudié dans la Section 1.2 du Chapitre 1, l'auto-stabilisation, introduit par Dijsktra en 1974 dans [Dij74], est une approche intéressante de tolérance aux pannes transitoires dans les réseaux. Rappelons qu'un système est dit auto-stabilisant si quelle que soit sa configuration de départ, il existe une garantie d'arriver à comportement correct en un nombre fini de transitions mais non borné. Suite à l'apparition d'une panne transitoire, le système retrouve progressivement, de lui même, une nouvelle configuration légale. Ainsi, comme montré dans [DLV08, OHN09, JM11], pour un système distribué comme les réseaux ad hoc, les algorithmes de structuration auto-stabilisants sont une approche efficace de structuration du réseau pour optimiser les communications et tolérer les pannes transitoires qui peuvent survenir.

Nous nous proposons d'étudier dans ce chapitre les solutions de structuration en *clusters*, auto-stabilisantes tout comme leurs homologues non auto-stabilisantes, proposées dans la littérature afin de pouvoir positionner les travaux de recherche que nous menons.

La structuration en *clusters* est communément appelée *clustering*. Ainsi, tout au long de ce document, nous utilisons indifféremment ces deux expressions. De même, nous emploierons indifféremment les termes de structure ou *cluster* pour désigner les zones (groupes) créées avec le *clustering*.

La suite de ce chapitre est organisée comme suit. Dans la Section 2.2, nous donnons d'abord la définition, les caractéristiques et les propriétés du *clustering*. Puis, nous présentons une classification des différentes techniques de structuration en *clusters*. La Section 2.3 décrit les solutions de structuration en *clusters* non-auto-stabilisantes puis une synthèse comparative

entre elles. Les approches de structuration auto-stabilisantes et une synthèse comparative sont présentées au niveau de la Section 2.4. Dans la Section 2.5, nous soulevons la problématique de recherche qui fait l'objet de nos travaux. Ce chapitre est conclu par un résumé au niveau de la Section 2.6.

2.2 Présentation du *clustering*

Dans cette section, nous présentons d'abord la notion de *cluster* et de *clustering*. Ensuite, les objectifs, propriétés et cas d'utilisation du *clustering*. Puis, nous donnons une classification des différentes approches de *clustering* proposées dans la littérature.

2.2.1 Notions de *cluster* et *clustering*

Le *clustering* est issu de la théorie de partitionnement des graphes [KL70, Fie73]. Soit $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ un graphe connexe et non-orienté tel que défini dans la Section 1.1.2 du Chapitre 1. On entend par partitionnement d'un graphe $\mathcal{G}$ la répartition de l'ensemble des sommets du graphe en sous-graphes. Il appartient aux problèmes de la classe NP-difficile.

Définition 2.1. ***Cluster***
Soit $\mathcal{R}$ un réseau modélisé par un graphe connexe et non-orienté $\mathcal{G}$ de diamètre $\mathcal{D}$. Un cluster $\mathcal{C}$ dans le réseau $\mathcal{R}$ est un sous-graphe g connexe du graphe $\mathcal{G}$ de diamètre $\delta \leq \mathcal{D}$.

Définition 2.2. ***Diamètre d'un cluster***
Le diamètre d'un cluster représente la valeur du plus long des plus courts chemins entre deux nœuds du cluster.

Définition 2.3. ***Clustering***
Le clustering consiste à découper le réseau en groupes de nœuds appelés clusters donnant ainsi au réseau une structure hiérarchique [EWB87].

Dans un réseau structuré en *clusters*, il peut y exister jusqu'à trois types de nœuds jouant des rôles différents. Ces différents types de nœuds sont : le *cluster-head*, le nœud de passage et le nœud membre.

Définition 2.4. ***Cluster-Head***
Le cluster-head est le nœud principal du cluster et il a pour rôle d'organiser et de coordonner les communications au sein du cluster.

Le *cluster-head*, noté CH, est élu selon une métrique particulière telle que l'identifiant des nœuds, le degré, l'énergie, la densité, etc. par l'algorithme de *clustering*. Dans les exemples de la figure 2.3(c), les *cluster-heads* sont matérialisés par une étoile.

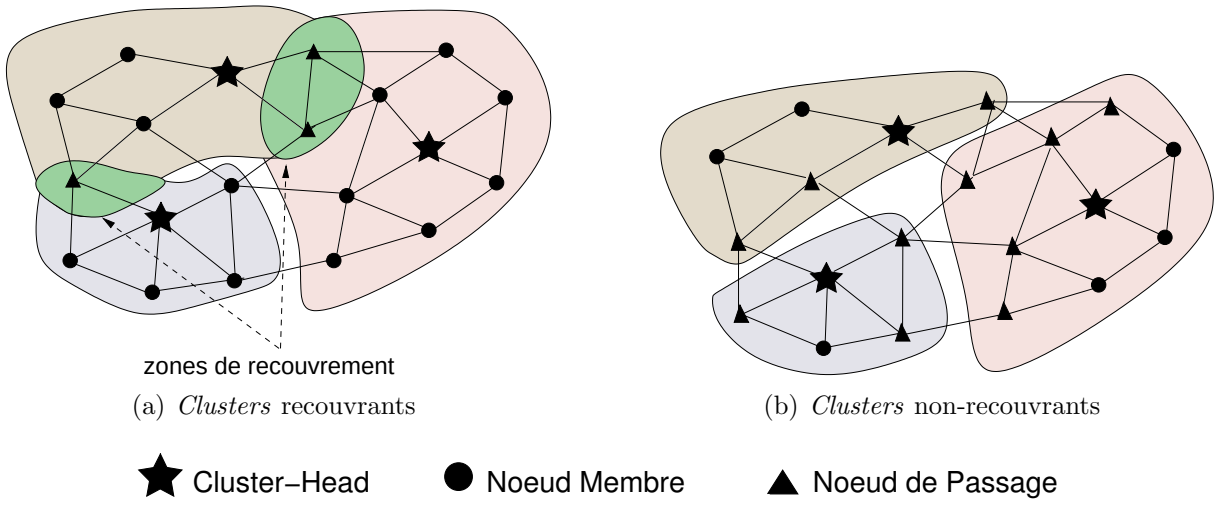
Définition 2.5. ***Nœud de Passage***
Un nœud de passage est un nœud du cluster qui a pour fonction d'assurer l'inter-connexion entre deux ou plusieurs clusters.

Les nœuds de passage, notés NP, constituent les points de jonctions entre plusieurs *clusters* et relayent ainsi les communications inter-*clusters*. Un nœud de passage est pris parmi ceux se trouvant à la frontière du *cluster*. Dans les exemples de la figure 2.3(c), les nœuds de passages sont matérialisés par un triangle.

Définition 2.6. ***Nœud Membre***
Un nœud au sein d'un cluster qui n'est ni cluster-head ni nœud de passage est qualifié de nœud membre.

Les nœuds membres, notés NM, sont encore appelés nœuds ordinaires ou nœuds simples. Dans les exemples de la figure 2.3(c), les nœuds membres sont matérialisés sous forme de cercle.

Dans le *clustering*, il existe deux techniques de construction différentes. Il est possible de construire des *clusters* recouvrants ou non-recouvrants.



(a) *Clusters* recouvrants

(b) *Clusters* non-recouvrants

(c) Différents type de nœuds

FIGURE 2.3 – *Clusters* recouvrants vs *Clusters* non-recouvrants

Définition 2.7. ***Clusters recouvrants***
Un cluster est dit recouvrant lorsqu'un nœud peut appartenir à au moins deux clusters adjacents.

Dans la construction de *clusters* recouvrants, comme illustré dans la figure 2.3(a), les nœuds de passage sont choisis parmi ceux se trouvant dans les zones de recouvrement. Ainsi, un ou plusieurs nœuds de passage constituent le pont entre les *clusters* adjacents. De plus, un nœud dans une zone de recouvrement jouant le rôle de nœud de passage aura au moins deux *cluster-heads*.

Définition 2.8. ***Clusters non-recouvrants***
Un cluster est dit non-recouvrant lorsqu'un nœud appartient à un et un seul cluster.

Dans la construction de *clusters* non-recouvrants, comme illustré dans la figure 2.3(b), les nœuds de passages sont ceux situés aux frontières de *clusters*. Un pont entre deux *clusters* adjacents est alors constitué d'au moins d'une paire de deux nœuds de passage pris dans chaque *cluster*.

2.2.2 Objectifs, propriétés et cas d'utilisation du *clustering*

Il existe dans la littérature une multitude de *survey* sur le *clustering* dans les réseaux ad hoc qui mettent en avant les objectifs à atteindre et les utilisations qu'il est possible d'en faire [ASSC02, CCL03, YC05, DYY08, AM09, CR09, JYZ09, SHN11, VBK12].

Les objectifs du *clustering*

La structuration du réseau en *clusters* a pour objectifs de réduire les communications. Cet objectif général englobe plusieurs sous objectifs. Nous allons énumérer quelques uns des plus importants. Cependant, avant de lister ces objectifs, rappelons que dans [SHN11], les auteurs précisent que le *clustering* comme tous les algorithmes d'infrastructure doit avoir un coût minimal. Cela est d'autant plus important que le processus de *clustering* peut être ramené à s'exécuter à plusieurs reprises pour réorganiser les structures suite à la mobilité des nœuds ou suite à d'autres pannes transitoires.

- **Équilibrer la charge** : le *clustering* cherche à répartir équitablement les tâches les plus coûteuses dans le réseau afin d'éviter des points de congestion ou une consommation de ressources déséquilibrée entre les nœuds du réseau [GSS07] ;
- **Prolonger la durée de vie du réseau** : par la réduction des communications, le *clustering* vise à retarder l'épuisement des ressources des nœuds du réseau. Cela aura pour conséquence de prolonger la durée du vie du réseau [YF04, AKKUM04] ;
- **Assurer une connectivité totale et une réduction des délais** : le *clustering* vise aussi à assurer une connectivité totale dans le réseau. Malgré le fait que les *clusters* constituent des zones de cloisonnement, n'importe quel couple de nœuds, se trouvant dans le même *cluster* ou dans des *clusters* différents, doit pouvoir communiquer [DW05]. De même, la communication entre n'importe quel couple de nœuds doit être assurée avec un délai satisfaisant selon les besoins applicatifs [NGS03] ;
- **Optimiser la bande passante** : en minimisant les communications et en évitant les duplications de messages et les retransmissions inutiles, le *clustering* vise à optimiser l'utilisation de la bande passante du réseau [CCL03] ;
- **Assurer une qualité de service (QoS)** : la mobilité des nœuds ou d'autres fautes transitoires peuvent entrainer des ruptures de liens de communications entrainant ainsi un arrêt momentané du service fournit par l'applicatif du réseau. Il est donc nécessaire d'assurer une certaine qualité de service, même en présence de ruptures de liens de communication, afin d'assurer le service requis [CCL03].

Propriétés du *clustering*

Pour atteindre les objectifs décrits ci-dessus, un algorithme de *clustering* doit vérifier un certain nombre de propriétés de construction. Dans [Bas99], Basagni présente trois propriétés qu'un algorithme de *clustering* doit satisfaire à la terminaison de l'algorithme, c'est-à-dire à la fin de l'exécution de la procédure de structuration.

Propriété 2.1. ***Dominance***
Chaque nœud du réseau a au moins un cluster-head dans son voisinage.

Dans la propriété 2.1, la notion de voisinage est à prendre au sens large selon le diamètre des *clusters*. En effet, dans un algorithme de *clustering* de diamètre d'au plus 2 c'est-à-dire à 1 saut, chaque nœud du réseau doit obligatoirement être voisin d'au moins un *cluster-head*. Dans le cas de *clusters* de diamètre $2k$, chaque nœud du réseau a au moins un *cluster-head* se trouvant au plus à une distance de k sauts.

Propriété 2.2.
Chaque nœud du réseau s'associe avec le cluster-head ayant le poids le plus plus fort dans son voisinage.

Dans la propreté 2.2, Basagni désigne par poids le plus fort la métrique utilisée pour élire les *cluster-heads*. De plus, tout comme dans la propriété 2.1, la notion de voisinage est à prendre au sens large selon que le diamètre des *clusters* est égal à 2 ou $2k$.

Propriété 2.3. ***Indépendance***
Deux cluster-heads ne peuvent pas être voisins.

La propriété 2.3 assure une bonne répartition des *cluster-head* dans l'ensemble du réseau. En d'autres termes, elle permet d'éviter que les clusters-heads se concentrent dans une zone donnée du réseau.

Cas d'utilisation possibles du *clustering*

Les objectifs et les propriétés décrits ci-dessus, une fois satisfaits, permettent à la structuration en *clusters* d'offrir quelques cas d'utilisation qui ont favorisé l'attention portée sur elle ces dernières décennies.

- **Coordination des communications** : la structure hiérarchique offerte par le *clustering* permet de mieux coordonner les communications dans le réseau. Effet, il est possible, par exemple, de faire orchestrer les communications au sein de *clusters* par le *cluster-heads* mais aussi d'établir des politiques de communications entre *clusters* adjacents [SHN11] ;
- **Routage hiérarchique** : dans les *clusters*, il est possible de mettre en place un routage hiérarchique en établissant une politique de routage au sein des *clusters* et d'autres schémas de routages spécifiques aux échanges d'informations entre *clusters*. Cela permet d'avoir un minimum d'informations à stocker dans les tables de routage conduisant ainsi à un routage plus efficace [SM04] ;
- **Agrégation de données** : comme la structuration en *clusters* offre une hiérarchisation du réseau, Il est donc possible de faire une agrégation de données par niveau de hiérarchie. Associée au routage hiérarchique, cette agrégation peut permettre de réduire le nombre de messages envoyés depuis les *clusters*, diminuant ainsi le trafic dans le réseau. De plus, cela permet d'envoyer un volume de données plus significatif fournissant une meilleure vue sur les *clusters* [CR09].
- **Ré-utilisation des ressources** : la hiérarchisation permet aussi une ré-utilisation et une redistribution des ressources du réseau. En effet, les mêmes fréquences ou codes d'accès au médium peuvent être utilisés dans deux *clusters différents* à condition que les *clusters* soient non-recouvrants (pour éviter les collisions et interférences) [Bas99, TML07] ;
- **Déploiement d'intergiciel** : le *clustering* peut aussi être utilisé pour un déploiement efficace d'intergiciel [CDDL10].

- **Passage à l'échelle** : un avantage important qu'offre le *clustering* est le passage à l'échelle. En effet, une structuration efficace du réseau permet de garder les performances satisfaisantes même avec l'augmentation de la taille du réseau [ASSC02, SM04].

2.2.3 Classification des solutions de *clustering*

Dans cette section, nous présentons une classification des solutions de *clustering* qui ont été proposées dans la littérature. Nous les classons d'abord suivant la famille à laquelle elles appartiennent, puis selon le diamètre des *clusters* et enfin selon la métrique utilisée pour élire le *cluster-head*.

Classification selon la famille d'algorithmes

Nous classons les solutions de *clustering* existantes en deux grandes familles : (*i*) les algorithmes non auto-stabilisants pour structurer le réseau et (*ii*) les algorithmes auto-stabilisants pour structurer le réseau et tolérer les pannes transitoires.

Les solutions de structuration non auto-stabilisantes ont fait l'objet d'un plus grand nombre de propositions et sont caractérisées par une initialisation bien déterminée. En effet, un algorithme doit démarrer avec des états bien définis et corrects. De plus, ces solutions considèrent les aspects les plus fondamentaux tels que la consommation énergétique ou la mobilité des nœuds, la densité du réseau, etc. Elles visent à améliorer la stabilité du réseau, à optimiser l'utilisation de la bande passante, à réduire la surcharge du réseau, etc. Cependant, les solutions de structuration non auto-stabilisantes sont souvent vulnérables aux pannes transitoires. En effet, elles ne sont pas en mesure de contrôler la cohérence des données présentes dans la mémoire des nœuds, dans les canaux de communication ainsi que de détecter et corriger les pannes transitoires.

Les solutions auto-stabilisantes ne nécessitent aucune initialisation des contenus des variables des nœuds et des canaux de communication ni aucune intervention extérieur. Les nœuds sont en mesure de gérer leurs états de façon autonome et distribuée, de détecter et de corriger les pannes transitoires en un nombre fini de transitions. Comme montré par Johnen et Mekhaldi dans [JM11], l'auto-stabilisation est une technique efficace de *clustering* et de tolérance aux pannes transitoires dans un système distribué comme les réseaux ad hoc. Pour ces raisons, de nombreux chercheurs se sont focalisés sur l'élaboration de solutions de *clustering* auto-stabilisantes. Dans le domaine du *clustering* auto-stabilisants, selon le modèle de communication utilisé, il existe deux approches différentes : (*i*) les algorithmes de *clustering* fondés sur un modèle à états et (*ii*) leurs homologues utilisant un modèle à passages de messages. Nous avons présenté ces deux modèles de communication dans la Section 1.1.5 du Chapitre 1.

Classification selon le diamètre des *clusters*

Dans chacune de ces deux familles de *clustering* que nous avons présenté ci-dessus (auto-stabilisante comme non auto-stabilisante), nous pouvons classer les solutions existantes en deux groupes : *(i)* les solutions à 1 saut et celles *(ii)* à k sauts. Dans les premières, chaque nœud du *cluster* se trouve à une distance de 1 saut du *cluster-head* et le diamètre maximal des

clusters est donc égal à 2. Les solutions à 1 saut, du fait du faible diamètre maximal qui est égal à 2, construisent un nombre important de *clusters* de petite taille. La conséquence qui en découle est que les *clusters* sont sensibles aux pannes transitoires favorisant ainsi de fréquentes reconstructions. Pour palier ce problème, les solutions à k sauts ont été proposées. Dans celles-ci, un nœud peut se situer jusqu'à une distance de k du *cluster-head*. Ainsi, le diamètre des *clusters* est au plus égal à $2k$. Notons que quel que soit le diamètre du *cluster*, il est possible de construire un arbre couvrant de *clusters* donnant ainsi un niveau de hiérarchisation supplémentaire.

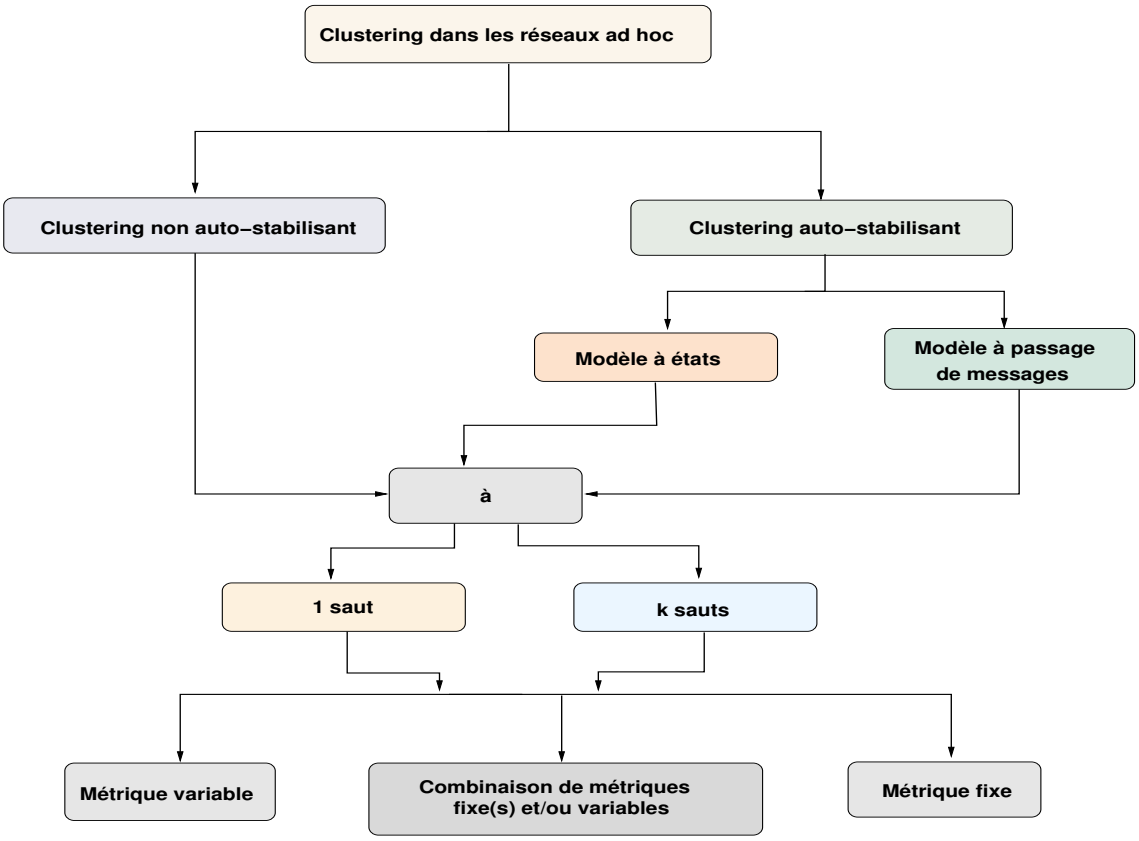


FIGURE 2.4 – Classification des solutions de *clustering*

Classification selon la métrique d'élection du *cluster-head*

De même dans les deux familles de *clustering*, pour élire les *cluster-heads*, trois types de métriques différentes peuvent être utilisées : (*i*) les métriques fixes, (*ii*) les métriques variables et (*iii*) une combinaison de métriques fixe(s) et/ou variable(s). Dans la première catégorie, une métrique fixe, c'est-à-dire constant dans le temps, est utilisée pour élire les *cluster-heads*. Parmi ces métriques fixes, nous pouvons citer l'identifiant des nœuds, le degré des nœuds, un poids fixe attribué aux nœuds, etc. Avec la deuxième catégorie, pour choisir les *cluster-heads*, une métrique variable, c'est-à-dire pouvant évoluer au cours du temps, est utilisée. Comme métriques variables, nous pouvons donner l'exemple de la mobilité, de l'énergie, de la pertinence, de densité, etc. Pour la troisième catégorie, une combinaison d'une ou de plusieurs métriques fixe(s) et/ou variable(s) est utilisée comme critère discriminatoire pour la forma-

tion des *clusters*. Par exemple, nous pouvons citer une combinaison de " identifiant + degré + énergie", " identifiant + densité + énergie", " identifiant + énergie" etc. Notons aussi que des coefficients de normalisation sont généralement associés aux différentes métriques dans une combinaison.

En résumé, en combinant les trois classifications que nous venons de présenter ci-dessus, nous donnons une classification générale, illustrée dans la figure 2.4, des différentes solutions de *clustering* existantes. Dans les deux prochaines sections, nous décrivons les solutions issues de cette classification.

2.3 Structuration non auto-stabilisante

Dans cette section, nous présentons les algorithmes de *clustering* non auto-stabilisantes dans les réseaux ad hoc. Pour ce faire, nous illustrons d'abord les solutions en 1 saut ensuite celles à k sauts.

2.3.1 Solutions à 1 saut

Dans cette partie, nous présentons les algorithmes de *clustering* non auto-stabilisants à 1 saut selon la métrique utilisée et suivant un ordre chronologique. Pour ce faire, décrivons d'abord les solutions qui utilisent une métrique fixe puis celles fondées sur une métrique variable pour terminer sur celles sur une combinaison de métriques.

Algorithmes à 1 saut utilisant une métrique fixe pour l'élection du *cluster-head*

Nous décrivons ici les algorithmes de *clustering* à métrique d'élection du *cluster-head* fixe.

LCA : algorithme du plus petit identifiant de Ephremides et *al.* [EWB87]

L'un des tout premiers algorithmes de *clustering* a été proposé par Ephremides et *al.* dans [EWB87]. Cet algorithme, nommé LCA pour *Linked Cluster Architecture*, construit à la fois des *clusters* recouvrants et non-recouvrants. Les auteurs supposent que chaque nœud du réseau porte un unique identifiant noté ID. Le déroulement de LCA consiste en deux étapes logiques : (i) construction des *clusters* et *(ii)* liaison des *clusters* (*Linkage*). Dans la première étape, en utilisant uniquement la comparaison des identifiants, un nœud u se déclare *cluster-head* s'il possède le plus petit identifiant (*Lowest ID*) parmi tous ses voisins, puis u informe son voisinage de son nouveau statut. Ainsi, tous les nœuds à distance 1 du *cluster-head* u avec un identifiant plus grand le rejoignent dans son *cluster*. Cette procédure se répète jusqu'à ce que tout nœud choisit comme *cluster-head* son voisin ayant le plus petit identifiant. Ensuite, commence l'étape de liaison des *clusters*. Deux situations sont possibles durant cette étape ; soit les *clusters* sont recouvrants, soit ils sont non-recouvrants. Dans le dernier cas, au moins une paire de nœuds est nécessaire, un dans chaque *cluster*, pour constituer le pont entre deux *clusters* adjacents. Dans le cas de *clusters* recouvrants, le nœud avec le plus petit identifiant se trouvant dans la zone de recouvrement est désigné comme passerelle. Au final, comme illustré dans le figure 2.5, LCA considère trois types de nœuds ; le *cluster-head*, les nœuds passerelles

(*gateway node*), et les nœuds ordinaires (*ordinary node*). LCA fonctionne en mode synchrone et utilise la méthode d'accès TDMA [3] [NK85] afin d'éviter les collisions. Donc, à chaque nœud est alloué un intervalle de temps durant lequel il a l'accès exclusif au canal de communication pour transmettre ses données.

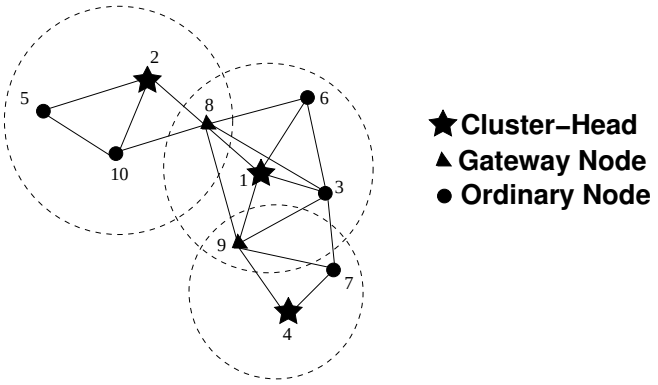


FIGURE 2.5 – Illustration du *clustering* dans le LCA [EWB87]

Cependant, dans LCA, la synchronisation nécessaire pour le protocole TDMA est coûteuse en temps et en message. En effet, LCA présente une complexité en temps et en messages de $O(n)$, avec n étant le nombre de nœuds dans le réseau. De plus, LCA est très sensible à la mobilité des nœuds qui entraine une reconstruction totale du réseau.

HCC : algorithme du plus grand degré de Gerla et Tsai [GTCT95]

Un algorithme de *clustering*, nommé HCC pour *High-Connectivity Clustering*, a été proposé par Gerla et Tsai dans [GTCT95]. Dans HCC, les auteurs mettent l'accent sur la connectivité pour le choix du *cluster-head*. Gerla et Tsai considèrent qu'un nœud ayant un degré (nombre de voisins) plus élevé présente une meilleure connectivité et est donc préférable à la fonction de *cluster-head*. Ainsi, les nœuds avec les degrés les plus élevés dans un voisinage à 1 saut deviennent *cluster-heads*.

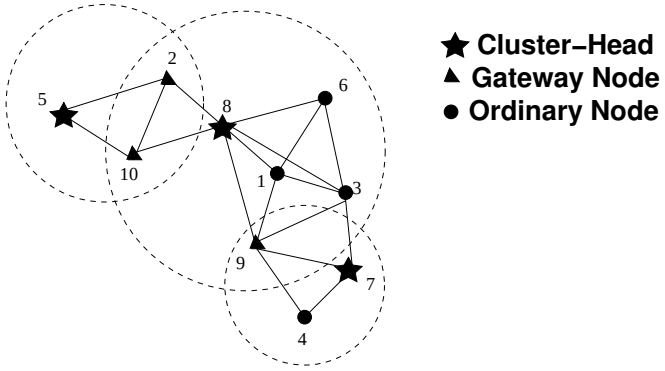


FIGURE 2.6 – Illustration du *clustering* dans le HCC [GTCT95]

3. *Time Disivion Multiple Access*

Comme LCA, HCC construit à la fois des *clusters* recouvrants comme non-recouvrants (cf. Figure 2.6). De même, pour l'accès au canal de communication, la méthode d'accès TDMA [NK85] est utilisé afin d'éviter les collisions.

Cependant, du fait du critère du degré, HCC construit des *clusters* très denses et en petit nombre. Cela constitue le point faible de HCC. En effet, les *clusters* sont très sensibles à la mobilité des nœuds. Cela a pour conséquence d'entrainer des reconstructions de la structure des *clusters*. HCC présente une complexité en temps et en messages de $O(n)$, avec n étant le nombre de nœuds dans le réseau.

LCC : algorithme du plus petit identifiant ou du plus grand degré de Gerla et *al.* [LCWG97]

Afin de palier le problème lié à la mobilité dans LCA et HCC et d'améliorer la stabilité des *clusters*, Gerla et *al.* ont proposé un algorithme de *clustering* nommé LCC (*Least Cluster Change*) dans [LCWG97]. LCC combine la technique de LCA et HCC avec deux conditions supplémentaires qui doivent être l'une ou l'autre satisfaite pour entrainer une reconstruction des *clusters*. Ainsi, une reconstruction est déclenchée lorsque (i) deux *cluster-heads* deviennent voisins du fait de leur mobilité ou (ii) lorsqu'un nœud sort de la zone de couverture de son *cluster-head*. Ces conditions (i) et (ii) constituent les améliorations apportées par LCC comparé à LCA et HCC qui eux reconstruisent les *clusters* dès le moindre changement topologique. LCC présente les cinq opérations suivantes :

1. Initialement l'algorithme LCA ou HCC est utilisé pour construire les *clusters* ;
2. Lorsqu'un nœud ordinaire se déplace d'un *cluster* i vers un *cluster* j, alors aucun des deux *cluster-heads* dans les *clusters* i et j ne changent ;
3. Lorsqu'un nœud ordinaire sort de la zone de couverture de son *cluster-head* et n'atteint aucune autre zone de couverture d'un autre *cluser-head*, alors il devient un *cluster-head* formant ainsi un *cluster* singleton ;
4. Lorsqu'un *cluster-head* $C(i)$ d'un *cluster* i se déplace dans un autre *cluster* j, alors l'un des deux va abandonner son statut de *cluster-head* par exécution de LCA ou HCC ;
5. Lorsque plusieurs nœuds sortent d'un *cluster*, alors ils vont construire un autre *cluster* en exécutant soit LCA, soit HCC.

Cette approche présente une vulnérabilité en cas de forte mobilité. En effet, dans une telle situation, la totalité des structures est à reconstruire même avec les deux contraintes ajoutées par Gerla et *al.*. LCC, tout comme LCA et HCC, présente une complexité en temps et en messages de $O(n)$, avec n étant le nombre de nœuds dans le réseau.

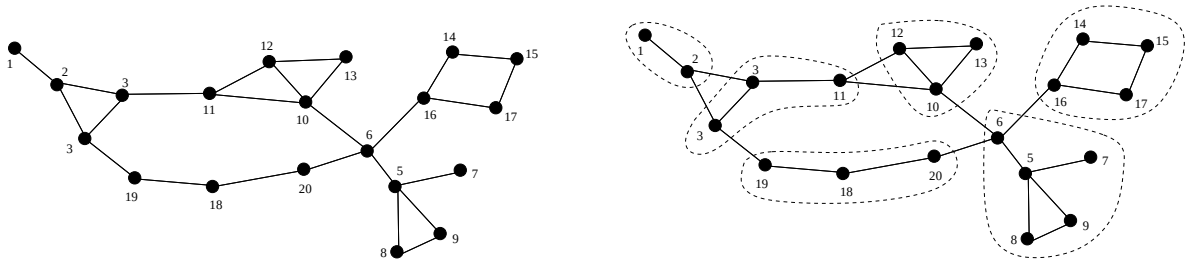
Algorithme non-recouvrant avec une non existence de *cluster-heads* de Lin et Gerla [LG97]

Dans [LG97], Lin et Gerla ont proposé un algorithme de *clustering* (*Adaptive Clustering for Mobile Wireless Networks*) où la notion de *cluster-head* n'existe que durant la phase de *clustering*. Une fois les *clusters* construits, tous les nœuds sont équivalents et jouent le même

rôle (cf. Figure 2.7). Lin et Gerla ont estimé que les *cluster-heads* peuvent représenter des goulots d'étranglement dans le réseau. Et cela peut constituer un handicap pour des applications en temps réel nécessitant une certaine qualité de service (QoS) sur la bande passante ou sur les délais. Ainsi, en utilisant comme critère l'identité des nœuds, Lin et Gerla ont proposé un algorithme adaptatif aux changements topologiques qui construit des *clusters* non-recouvrants. Pour cela, les auteurs émettent les trois hypothèses suivantes :

1. Chaque nœud u connait son identifiant noté ID ainsi que l'ensemble des identifiants de ses voisins. Le tout est maintenu dans un ensemble Γ_u ;
2. Chaque message envoyé par un nœud est correctement reçu au bout d'un temps fini par tous ses voisins à distance 1 ;
3. La topologie du réseau ne change pas durant la phase de *clustering*.

Pour construire les *clusters*, un nœud u se déclare *cluster-head* si son identifiant est le plus petit parmi tous ceux qui se trouvent dans son ensemble Γ_u. Ensuite, u envoie son nouveau statut à tous les nœuds de son voisinage. Tout nœud v ayant reçu un message de u le supprime de son ensemble Γ_v puis l'adapte comme *cluster-head* s'il n'en avait pas encore adopté un ou si u possède un identifiant plus petit que celui *cluster-head* actuel de v. Cette procédure se répète jusqu'à ce que l'ensemble Γ_u de chaque nœud u devienne vide.



(a) Réseau de départ (b) Réseau structuré en *clusters*

FIGURE 2.7 – Illustration du *clustering* dans l'algorithme de Lin et Gerla [LG97]

Pour la maintenance des *clusters* en cas de modifications topologiques, Lin et Gerla partent du principe qu'il est préférable de laisser les nœuds ayant une forte connectivité ainsi que leurs voisins dans le *cluster* de départ et de supprimer les autres nœuds (avec une faible connectivité) du *cluster*. Pour cela, ils supposent que le déplacement d'un nœud u ne doit pas l'éloigner à une distance de plus 2 sauts de tout autre nœud v du *cluster*. Si tel est le cas, alors le nœud v est supprimé du *cluster*. Ainsi, v cherchera à joindre un autre *cluster* ou à former un *cluster* singleton.

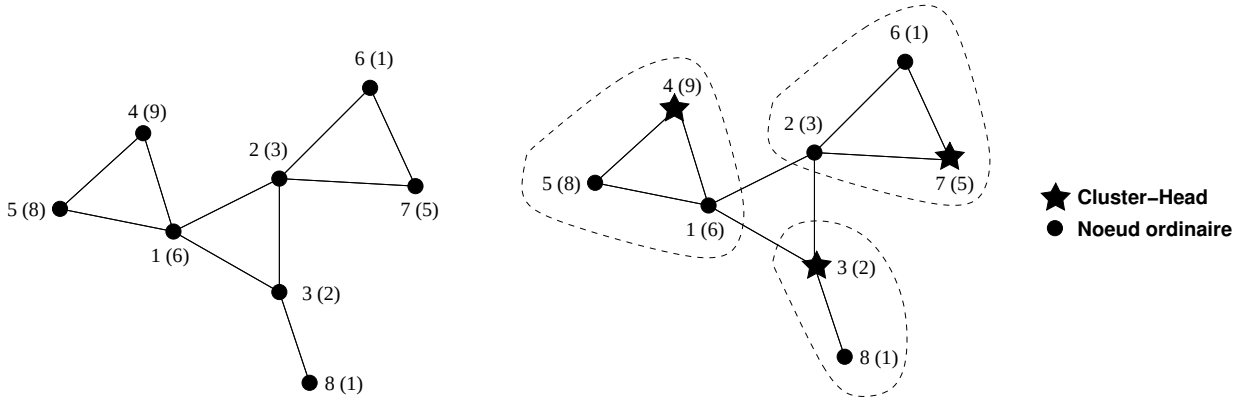
L'approche de Lin et Gerla souffre du fait qu'elle engendre beaucoup de messages pour la maintenance des *clusters*. Le fait que les nœuds doivent avoir une vision permanente du voisinage à distance de 2 sauts requiert beaucoup d'échanges de messages. Cette situation est beaucoup plus forte en cas de mobilité. L'algorithme de Lin et Gerla présente une complexité en temps et en messages de $O(n)$, avec n étant le nombre de nœuds dans le réseau.

DCA et DMAC : algorithmes non-recouvrants à métrique de poids de Basagni [Bas99]

Dans [Bas99], Basagni propose deux algorithmes, l'un nommé DCA pour *Distributed Clustering Algorithm* et l'autre appelé DMAC pour *Distributed and Mobility-Adaptive Clustering*, dans le but de structurer des réseaux statiques ou dynamiques. Basagni modélise le réseau comme un graphe connexe et non orienté où chaque nœud est identifié par un unique identifiant. Le critère d'élection dans les approches proposées par Basagni est un poids unique attribué à chaque nœud du graphe. Ainsi, chaque nœud u est caractérisé par le couplet $(ID_u; w_u)$ où ID_u représente l'identifiant du nœud et w_u le poids qui lui est associé. Un nœud avec un poids élevé est préférable à la fonction de *cluster-head*. De plus, DCA et DMAC construisent des *clusters* non-recouvrants et deux types de nœuds sont considérés : les *cluster-heads* et les *nœuds ordinaires*. Basagni considère les propriétés suivantes dans ses deux algorithmes de *clustering* :

1. Chaque nœud ordinaire a au moins un *cluster-head* dans son voisinage (propriété de dominance) ;
2. Chaque nœud ordinaire s'associe avec le voisinage du *cluster-head* ayant le poids le plus élevé ;
3. Deux *cluster-heads* ne peuvent pas être voisins (propriété d'indépendance).

La propriété (1) assure que chaque nœud ordinaire a directement accès à au moins un *cluster-head* permettant ainsi les communications intra-*clusters* et inter-*clusters*. La propriété (2) assure que chaque nœud ordinaire restera dans le voisinage du *cluster-head* qui lui assurera un meilleur service. La propriété (3) permet d'assurer une bonne couverture dans le réseau. La figure 2.8 montre un exemple de *clustering* où les trois propriétés sont vérifiées. La figure 2.8(a) schématise les nœuds du réseau à l'état initial avec leurs poids $ID_u(w_u)$. La figure 2.8(b) montre les structures obtenues à la fin de la procédure de *clustering*. Pour construire les *clusters* vérifiant les trois propriétés, Basagni a proposé deux algorithmes que sont DCA et DMAC.



(a) Réseau de départ (b) Réseau structuré en clusters

FIGURE 2.8 – Illustration du *clustering* dans DCA et DMAC de Basagni [Bas99]

Dans DCA, Basagni considère un réseau "quasi-static" c'est-à-dire un réseau dont le déplacement des nœuds est considéré comme "lent". Pour cela, il suppose que durant la phase de *clustering*, la topologie du réseau ne change pas. Il suppose aussi que initialement chaque nœud u

connait son identifiant ID_u et son poids w_u, ainsi que l'identifiant ID_v et le poids w_v de chaque nœud v appartenant à son voisinage. Ainsi, seuls les nœuds avec les poids les plus élevés dans le voisinage à distance 1 diffusent un message pour se déclarer *cluster-head* indiquant ainsi le démarrage de la phase de *clustering*. De ce fait, chaque nœud aura la libre décision de choisir comme *cluster-head* le voisin ayant le poids le plus élevé. Basagni a montré dans DAC, qu'à la fin de la procédure de *clustering*, tout nœud u du réseau aura soit un statut de *cluster-head* soit un statut de nœud ordinaire.

Dans DMAC, Basagni considère un réseau dynamique. À la différence de DCA, durant la phase de *clustering*, les nœuds peuvent se déplacer et changer de position. Pour supporter la mobilité, DMAC autorise les nœuds à être "réactif's' non seulement à la réception d'un message d'un voisin mais aussi à la "cassure" d'un lien de communication due à une panne ou à un déplacement (départ) d'un nœud et à la "présence" (établissement) d'un nouveau lien de communication due à l'arrivée d'un nœud. Le mécanisme d'élection reste le même que dans DCA. Cependant, dans DMAC, dès l'arrivée d'un nœud u dans le réseau, u exécute une procédure $Init$ pour déterminer son rôle à jouer. Si dans le voisinage de u il existe un *cluster-head* v avec un poids plus élevé, alors u rejoint le *cluster* de v. Sinon, u se déclare *cluster-head*. Lorsqu'un nœud u détecte la défaillance d'un lien de communication causée par un nœud v, alors u versifie le statut de v. Si v est un nœud ordinaire, alors u le supprime de son *cluster*. Par contre, si u est un nœud ordinaire tandis que v était son *cluster-head*, alors u va déterminer son nouveau rôle dans le réseau. Si u dispose du poids le plus élevé dans son voisinage alors il se déclare *cluster-head*. Sinon, il rejoint un autre *cluste-head* avec un poids plus élevé.

Cependant, dans les deux algorithmes de Basagni, il existe une contrainte forte ; initialement tout nœud doit connaitre les identifiants et les poids de tous ses voisins. Cela suppose un déploient initial bien particulier. Or, les réseaux ad hoc sont souvent caractérisés par un déploiement aléatoire et à priori, les nœuds ne connaissent pas leurs voisins à l'avance. De plus, DCA et DMAC présentent une complexité en temps de $O(D+1)$ et en message de $O(n)$, avec D étant le diamètre du réseau et n le nombre de nœuds.

Algorithmes à 1 saut utilisant une métrique variable pour l'élection du *cluster-head*

Dans cette section, nous présentons les algorithmes de *clustering* non auto-stabilisants à 1 saut utilisant une métrique variable pour l'élection du *cluster-head*.

MBC : algorithme non-recouvrant à métrique de mobilité de Beongku et Papavassiliou [BP01]

Une métrique variable, dite de mobilité, est utilisée dans [BP01] par Beongku et Papavassiliou pour proposer un algorithme nommé MBC (*Mobility-Based Clustering*) qui construit des *clusters* non-recouvrants. Dans BMC, la mobilité relative des nœuds entre eux est utilisée comme critère d'élection du *cluster-head*. L'objectif est de favoriser les nœuds les moins mobiles à la fonction de *cluster-head*. Pour cela, les nœuds du réseau sont équipés de GPS (*Global Position System*) afin qu'ils puissent déterminer leurs coordonnées ainsi que celles de leurs voisins. L'idée de base de MBC est de combiner un partitionnement physique et logique du réseau.

C'est-à-dire, d'associer la proximité géographique avec une fonction de relation entre les nœuds tel un canevas (*pattern*) de mobilité pour aboutir au concept de mobilité relative. Cette mobilité relative permettra, selon Beongku et Papavassiliou, d'améliorer la stabilité des *clusters*. Dans cette optique, tout nœud m du réseau calcule sa mobilité relative par rapport à chacun de ses voisins n en fonction de leurs vitesses relatives de déplacement. Soit $v(m,t)$ le vecteur de vitesse du nœud m au temps t et $v(n,t)$ celui du nœud n au même temps t. La vitesse relative entre les nœuds m et n est calculée à partir de l'équation 2.1 suivante :

$$v(m,n,t) = v(m,t) - v(n,t) \tag{2.1}$$

À partir de cette vitesse relative $v(m,n,t)$, la mobilité relative, notée $M_{m,n,T}$, entre toute paire de nœuds $(m;n)$ durant une période de temps T est obtenue comme suit :

$$M_{m,n,T} = \frac{1}{N}\sum_{i=1}^{N} |v(m,n,t)| \tag{2.2}$$

Dans l'équation 2.2, N représente le nombre de fois que les nœuds m et n ont changé de positions et que la vitesse relative $v(m,n,t)$ a été recalculée durant la période T.

Lors de la phase de *clustering*, les nœuds s'échangent leurs vitesses relatives afin de pouvoir calculer leurs mobilités relatives correspondantes. Ainsi, le nœud avec la plus faible mobilité relative devient *cluster-head*. Ceci permet de réduire les fréquences de réélections des *cluster-heads*.

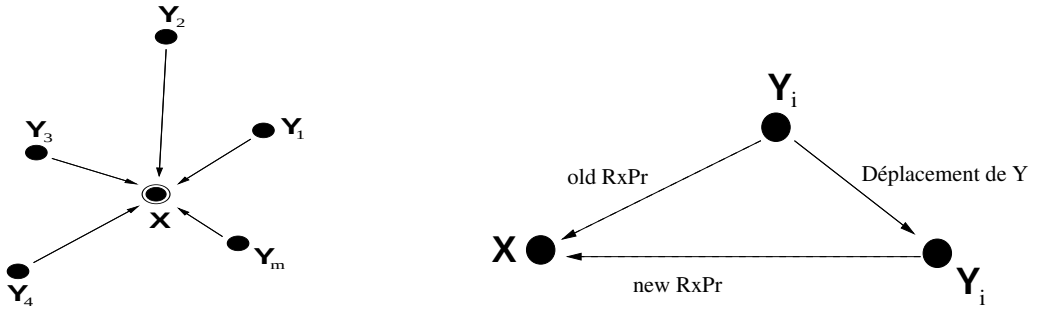
L'algorithme de Beongku et Papavassiliou, bien qu'il apporte une certaine stabilité dans la structure des *clusters*, souffre principalement des périodes de réévaluation de la mobilité relative. En effet, à chaque série de déplacements, les nœuds s'échangent leurs vitesses afin de réévaluer leurs mobilités relatives. Or, cette procédure requiert d'importants échanges de messages. MBC présente une complexité en messages de $O(n)$, avec n étant le nombre de nœuds dans le réseau.

MOBIC : algorithme non-recouvrant à métrique de mobilité de Basu et *al.* [BKL01]

Dans [BKL01], Basu et *al.* ont présenté un algorithme de *clustering*, nommé *Lowest Relative Mobility Clustering Algorithm* (MOBIC), où une métrique de mobilité est utilisée dans le but d'apporter une meilleure stabilité. A la différence de BMC [BP01], MOBIC ne nécessite pas de GPS pour la localisation de nœuds. En effet, la métrique de mobilité dans MOBIC ne se fonde pas sur les coordonnées géographiques et les vitesses de déplacements mais sur la force des signaux des messages échangés par les nœuds. MOBIC reprend le principe d'exécution de LCA décrit dans [EWB87] excepté qu'à la place du critère d'identité minimale (*Lowest-ID*) pour choisir les *cluster-head*, la mobilité est utilisée. Pour cela, chaque nœud X du réseau calcule, pour chacun des nœuds Y_i de son voisinage à distance 1 (cf. Figure 2.9(a)), la mobilité relative, notée $M_X^{rel}(Y_i)$, en se fondant sur la force des signaux de deux messages successifs reçus de Y_i comme suit :

$$M_X^{rel}(Y_i) = 10 \times \log_{10}\left(\frac{RxPr^{new}_{Y_i \mapsto X}}{RxPr^{old}_{Y_i \mapsto X}}\right) \tag{2.3}$$

Dans l'équation 2.3, $RxPr^{new}_{Y_i \mapsto X}$ représente la force du signal du message "Hello" nouvellement reçu par le nœud X et venant du voisin Y_i. De même, $RxPr^{old}_{Y_i \mapsto X}$ correspond à celle du signal du message "Hello" précédemment reçu par le nœud X et venant de Y_i.



(a) Voisinage du nœud X

(b) Illustration du déplacement relatif d'un voisin Y_i par rapport à X

FIGURE 2.9 – Illustration de la mobilité dans MOBIC [BKL01]

Si $RxPr^{new}_{Y_i \mapsto X} < RxPr^{old}_{Y_i \mapsto X}$, alors $M^{rel}_X(Y_i) < 0$. Dans ce cas, une valeur négative de la mobilité relative indique que les nœuds X et son voisin Y_i s'éloignent mutuellement l'un de l'autre (cf. Figure 2.9(b)). Sinon, si $RxPr^{new}_{Y_i \mapsto X} > RxPr^{old}_{Y_i \mapsto X}$, alors $M^{rel}_X(Y_) > 0$. Dans ce cas, les nœuds X et Y_i se rapprochent mutuellement l'un de l'autre lors de leurs déplacements.

Afin d'élire les *cluster-heads*, tout nœud X du réseau détermine sa mobilité relative générale M^{rel}_X. Cette mobilité relative générale du nœud X est calculée comme étant la variance résultante des mobilités relatives établies avec chacun des voisins Y_i à l'aide de l'équation 2.3. Ainsi, les nœuds avec les plus petites valeurs de mobilités relatives générales dans une distance de 1 saut deviennent *cluster-heads*.

En cas de mobilité, MOBIC reprend le principe de LCC pour départager deux *cluster-heads* u et v qui deviennent voisins avec une contrainte supplémentaire. Soit v le *cluster-head* avec le plus petit identifiant. Le nœud v abandonnera son statut de *cluster-head* au profit du nœud u si et seulement si il demeure dans le voisinage du *cluster-head* u au delà d'une durée τ. Selon Basu et *al.*, cette contrainte évitera des fréquences de réélections élevées de *cluster-heads*. Les simulations menées par Basu et *al.* dans [BKL01] ont montré que MOBIC améliore de 30% la stabilité par rapport à LCA décrit dans [EWB87].

Cependant, tout comme BMC [BP01], MOBIC nécessite beaucoup d'échanges de messages. De plus, à chaque déplacement des nœuds, même pour les *cluster-heads* où une attente d'une durée de τ a été introduite, une réévaluation de toutes les mobilités relatives est générale et nécessaire. Et cela entraine d'importants échanges de messages. MOBIC présente une complexité en temps et en messages de $O(n)$, avec n étant le nombre de nœuds du réseau.

LEACH : algorithme non-recouvrant à métrique probabiliste de Heinzelman et *al.* [HCB00]

Dans [HCB00] Heinzelman et *al.* ont proposé un des algorithmes de *clustering*, connu sous le nom de *Low Energy Adaptive Clustering Hierarchy* (LEACH), où l'élection des *cluster-heads* se fonde sur une génération d'un nombre aléatoire. L'objectif de LEACH est de permettre une faible consommation énergétique lors de la phase de *clustering* ainsi que dans le routage des informations. Pour cela, il procède par *rounds* et ces derniers ont approximativement le même intervalle de temps (durée) déterminé préalablement. Comme illustré dans la figure 2.10,

chaque round est composé de deux phases successives. La première est la phase d'initialisation, nommée "Set-Up Phase", durant laquelle le réseau s'auto-organise en *clusters*. Et la seconde représente la phase de transmission, nommée "Study-State Phase", lors de laquelle les données sont collectées et routées dans le réseau hiérarchisé.

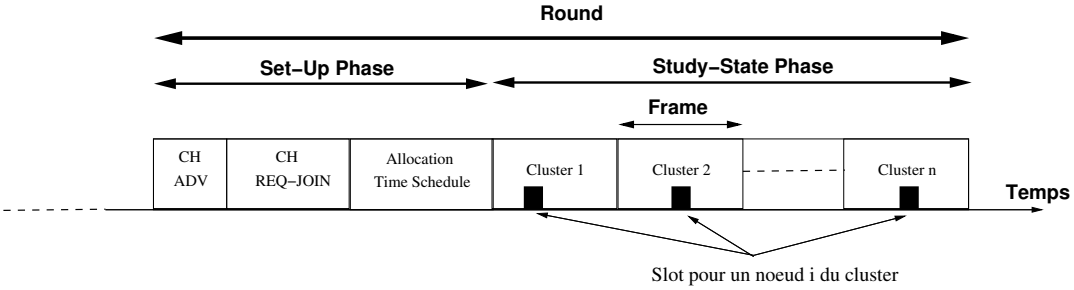


FIGURE 2.10 – Détails d'un round dans LEACH [HCB00]

La "Set-Up Phase" est composée de trois sous phases (cf. Figure 2.10) ; une phase d'annonce, une phase d'organisation des *clusters* et une phase d'ordonnancement.

La phase d'annonce est déclenchée par la station de base en donnant à tous les nœuds du réseau le pourcentage p de *cluster-heads* souhaité durant un round d'une durée τ. Tout nœud u tire un nombre aléatoire $\chi(u)$ telle que $0 < \chi(u) < 1$. Puis, compare $\chi(u)$ avec un seuil $\mathcal{S}(u)$ calculé comme suit :

$$\mathcal{S}(u) = \begin{cases} \frac{p}{1-p\times(\tau \times mod(\frac{1}{p}))} & \text{si } u \in G_{CH} \\ \\ 0 & \text{sinon.} \end{cases} \tag{2.4}$$

Dans l'équation 2.4, G_{CH} représente l'ensemble des nœuds qui n'ont pas encore été élus comme *cluster-head* lors des $\frac{1}{p}$ rounds prétendants. En d'autres termes, seuls les nœuds qui n'ont pas encore été élus *cluster-heads* lors des $\frac{1}{p}$ rounds précédents peuvent prétendre l'être dans le round courant. La probabilité p est préférentiellement choisie ente 5% et 15%.

Ensuite, lors de la phase d'organisation des *clusters*, si $\chi(u) < \mathcal{S}(u)$ alors le nœud u se déclare *cluster-head* et envoie un message *ADV* contenant son identifiant pour avertir son voisinage de son nouvel statut. Sinon, il attend les messages *ADV* venant des autres *cluster-heads*. Pour éviter les collisions avec les autres messages *ADV*, la méthode d'accès CSMA [4] [MR83, NZD99] est utilisée. Les nœuds qui ne sont pas élus *cluster-heads* vont s'appuyer sur le force du signal des messages *ADV* reçus pour adopter un *cluster-head*. Un nœud v va ainsi adhérer au *cluster-head* ayant le plus fort signal (en cas d'égalité, v choisit aléatoirement un des deux). Puis, v diffuse un message *REQ-JOIN* à son *cluster-head* pour lui notifier son adhésion à son *cluster*.

Pour terminer la "Set-Up Phase", les *cluster-heads* vont créer un ordonnancement au sein de leurs *clusters*. Pour cela, un utilisant le protocole TDMA [NK85], les *cluster-heads* attribuent à chaque nœud de leurs *clusters* un *slot* de temps durant lequel les nœuds vont envoyer leurs données collectées. Pour éviter les interférences entre *clusters* adjacents, les *cluster-heads* choisissent aussi un code à l'aide du protocole CDMA [5] [GJP+91, Pra96] utilisé lors des com-

4. *Carrier Sense Multiple Access*
5. *Code Division Multiple Access*

munications des *cluster-heads* vers la station de base. La "Set-Up Phase" se termine ainsi puis démarre la "Study-State Phase".

Lors de la "Study-State Phase" (phase de transmission), plus longue que la "Set-Up Phase"', les nœuds dans les *clusters* collectent leurs données puis les envoient à leurs *cluster-heads* durant leurs *slots* de temps obtenus précédemment à l'aide du protocole TDMA. Chaque *cluster-head* agrège toutes les données issues de son *cluster* puis l'envoie à la station de base en utilisant son code obtenu précédemment à l'aide du protocole CDMA.

Cependant, le protocole LEACH présente plusieurs limitations. Il suppose que les *cluster-heads* puissent transmettre leurs données sur une longue portée afin d'atteindre la station de base et que chaque nœud dispose de la puissance nécessaire pour supporter chacun des protocoles CSMA, TDMA et CDMA utilisés. Ainsi, LEACH n'est pas adapté pour un déploiement sur un grand nombre de nœuds et dans de vastes étendues. Également, LEACH ne pose aucune condition sur l'emplacement des *cluster-heads* de même que sur leurs niveaux énergétiques. Cela fait qu'un certain nombre de *cluster-heads* peuvent se retrouver dans une zone commune ou ayant de faibles niveaux d'énergies.

Afin d'améliorer LEACH, d'autres propositions ont été faites dans la littérature telles que TEEN [MA01], LEACH-C [HCB02], APTEEN [MA02], PEGASIS [LR02], HEED [YF04], et M-LEACH [MR04]. Cependant, toutes ces approches précitées, tout comme LEACH sont coûteuses en termes d'échanges de messages et nécessitent une forte synchronisation. En effet, LEACH présente une complexité en message de $O(n)$, avec n étant le nombre de nœuds du réseau.

MPGC : algorithme recouvrant à métrique de puissance de transmission de Leu et *al.* [LTCH06]

Dans [LTCH06], Leu et *al.* ont proposé un algorithme de structuration qui construit des *clusters* recouvrants, nommé MPGC (*Multicast Power Greedy Clustering Algorithm*), afin de réduire la consommation énergétique. Pour cela, Leu et *al.* adoptent une heuristique ("*greedy heuristic*") en s'inspirant des travaux de Tang et Raghavendra décrits dans [TR04]. Ainsi, dans MPGC, chaque nœud contrôle son niveau d'énergie et dispose de plusieurs puissances de transmissions qu'il utilise en fonction de son énergie résiduelle. L'algorithme de MPGC se déroule en trois phases ; "*Beacon phase*", "*Greedy phase*" (algorithme glouton) et "*Recruting phase*". Dans le "*Beacon phase*", chaque nœud envoie un signal de *beacon*, contenant son identifiant et son niveau de batterie. Les auteurs supposent qu'un message *beacon* doit être envoyé avec une forte puissance afin que le signal puisse atteindre tous les nœuds du voisinage. Ainsi, chaque nœud collectionne les identifiants et les niveaux d'énergie de tous ses voisins. Dans la "*Greedy phase*", tout nœud du réseau envoie une déclaration de *cluster-head* à tous ses voisins dont il a reçu un *beacon* dans la phase précédente. Pour ce faire, il procède étape par étape, en augmentant successivement la force du signal pour atteindre ses voisins les plus proches au plus lointains. Cette phase permet à chaque nœud de déterminer sa puissance de transmission la plus adaptée pour le "*greedy heuristic*" et la puissance résiduelle de chacun de ses voisins. Enfin, dans le "*Recruting phase*", le *cluster-head* est élu. Grâce aux informations recueillies dans les deux précédentes phases, les nœuds avec les plus grandes puissances résiduelles parmi leurs voisins deviennent *cluster-head*.

Cet algorithme, bien qu'il prolonge la durée du vie du réseau, se déroule en plusieurs phases (3 phases). Or, ces trois phases nécessitent d'importants échanges de messages afin d'élire le

nœud ayant la plus forte puissance résiduelle dans le voisinage comme *cluster-head*. En effet, MPGC présente une complexité en temps de $O(m+1)$ et en messages de $O(n \times (m+1))$, avec m étant le nombre de niveaux de puissance de transmission et n étant le nombre de nœuds du réseau.

Algorithmes à 1 saut utilisant une combinaison de métriques pour l'élection du *cluster-head*

Dans cette section, nous présentons les solutions de *clustering* non auto-stabilisantes à 1 saut qui utilisent une combinaison de métriques fixes et/ou variables comme critère de choix pour l'élection du *cluster-head*.

WCA : algorithme non-recouvrant à combinaison de métriques variables de Chatterjee et *al.* [CKT02]

Dans [CKT02], Chatterjee et *al.* ont proposé WCA pour *Weighted Clustering Algorithm*. WCA construit des *clusters* non-recouvrants à 1 saut. Dans cet algorithme, le critère d'élection du *cluster-head* est un poids noté $\mathcal{W}_u$, calculé de façon distribuée au niveau de chaque nœud u et qui est fonction de quatre métriques que sont : la différence de degré notée Δ_u, la somme des distances entre u et chacun de ses voisins notée $\mathcal{D}_u$, la mobilité relative du nœud u notée $\mathcal{M}_u$ et le temps de service durant lequel le nœud u est resté en tant que *cluster-head* noté $\mathcal{P}_u$. Chacune de ces métriques est pondérée par un coefficient w_i et le poids $\mathcal{W}_u$ est calculé comme suit :

$$\mathcal{W}_u = w_1 \times \Delta_u + w_2 \times \mathcal{D}_u + w_3 \times \mathcal{M}_u + w_4 \times \mathcal{P}_u \tag{2.5}$$

avec

$$w_1 + w_2 + w_3 + w_3 = 1$$

La différence de degré Δ_u est calculée comme étant la différence entre le degré réel du nœud, c'est-à-dire le nombre de voisins, et un degré idéal δ fixé comme étant le nombre optimal de nœuds autour d'un *cluster-head*. En effet, Chatterjee et *al.* supposent qu'un *cluster-head* ne peut générer qu'un certain nombre de nœuds δ au-delà duquel il devient surchargé et épuise rapidement ses ressources. Pour calculer la métrique de mobilité $\mathcal{M}_u$, WCA reprend la méthode de MOBIC proposée par Basu et *al.* dans [BKL01]. La somme des distances $\mathcal{D}_u$ est calculée à partir des coordonnées des nœuds obtenues par GPS. Le temps de service $\mathcal{P}_u$ est obtenu en faisant la somme des différentes durées durant lesquelles le nœud u a été *cluster-head*. Ainsi, à l'initialisation tous les nœuds du réseau s'échangent leurs informations, calculent leurs poids et s'échangent ces poids à nouveau. Le nœud avec la plus petite valeur de poids dans un voisinage à distance 1 devient *cluster-head*.

Dans WCA, dès qu'un nœud sort de la zone de couverture de tous les *cluster-heads*, la procédure de *clustering* est déclenchée à nouveau. Ce procédé est très couteux en cas de forte mobilité. Également, le calcul des poids est très couteux et peut entrainer une surcharge du réseau. En effet, WCA présente une complexité en temps de $O(D)$ et en messages de $O(n)$, avec D étant le diamètre du réseau et n le nombre de nœuds.

SBCA : algorithme non-recouvrant à combinaison de métriques variables de Chatterjee et *al.* [AJRA08]

Adabi et *al.* ont proposé dans [AJRA08] un algorithme appelé SBCA : *Score Based Clustering Algorithm*. SBCA est une solution de *clustering* non-recouvrante distribuée qui a été proposée dans le but de prolonger la durée de vie du réseau et de minimiser le nombre de *clusters* comparé à WCA [CKT02]. Pour l'élection du *cluster-head*, SBCA se fonde sur une métrique appelée *Score* qui est calculée de façon distribuée au niveau de chaque nœud u et est fonction de quatre paramètres que sont : l'autonomie de la batterie du nœud u notée $\mathcal{B}_\tau(u)$, le degré de u noté $\mathcal{N}_n(u)$, le nombre de membre de u noté $\mathcal{N}_m(u)$ et la stabilité de u notée $\mathcal{S}(u)$. Le *Score* est déterminé comme suit :

$$Score(u) = c_1 \times \mathcal{B}_\tau(u) + c_2 \times \mathcal{N}_n(u) + c_3 \times \mathcal{N}_m(u) + c_4 \times \mathcal{S}(u) \tag{2.6}$$

Dans l'équation 2.6, les éléments c_1, c_2, c_3 et c_4 sont des constantes. L'autonomie de la batterie $\mathcal{B}_\tau(u)$ représente l'énergie résiduelle disponible au niveau de la batterie du nœud u à l'instant où le *Score* est calculé. Le degré $\mathcal{N}_n(u)$ est le nombre de nœuds dans le voisinage à distance 1. Le nombre de membres $\mathcal{N}_m(u)$ désigne le nombre de nœuds généré par un *cluster-head*. La stabilité $\mathcal{S}(u)$ représente la période de temps où les voisins du nœuds u sont restés dans sa zone de couverture et est calculée comme suit :

$$\mathcal{S}(u) = \sum_{i=1}^{n} \mathcal{T}_{RF} - \mathcal{T}_{RL} \tag{2.7}$$

Dans l'équation 2.7, $\mathcal{T}_{RF}$ est le temps où le nœud u a reçu un premier message venant du voisin i, $\mathcal{T}_{RF}$ le temps où le dernier message du voisin i a été reçu par u et n le nombre total de voisins du nœud u.

Pour la structuration du réseau avec l'algorithme SBCA, tout nœud u calcule son *Score* d'après l'équation 2.6 et le diffuse à ses voisins. Donc, les nœuds maintiennent à jour une table contenant les Scores de tous leurs voisins. Ainsi, le nœud ayant le Score le plus élevé devient *cluster-head*.

Cependant, bien que SBCA génère moins de *clusters* et offre de meilleurs délais de bout en bout que WCA, Adabi et *al.* n'ont pas abordé le comportement de SBCA en cas de mobilité. De plus, SBCA présente une complexité en messages de $O(n)$, avec n le nombre de nœuds du réseau.

2.3.2 Solutions à k sauts

Après avoir décrit les algorithmes de *clustering* non auto-stabilisants à 1 saut, nous abordons maintenant ceux à k sauts. Nous les décrivons également suivant la métrique utilisée pour élire les *cluster-heads*.

Algorithmes utilisant une métrique fixe pour l'élection du *cluster-head*

Nous présentons ici les algorithmes de *clustering* à k sauts qui utilisent une métrique fixe.

Algorithme non-recouvrant à métrique non spécifique de Lin et Chu [LC00]

Lin et Chu ont présenté dans [LC00] un algorithme de structuration non-recouvrant où l'élection du *cluster-head* ne se fonde pas sur une métrique spécifique. Pour ce faire, Lin et Chu supposent qu'un nœud u qui se déplace dans le réseau ou qui vient de se réveiller, déclenche une phase d'initialisation. Ce nœud u envoie alors un message pour découvrir son voisinage et annoncer de sa présence. Si les nœuds de son voisinage sont eux aussi en phase d'initialisation, alors le nœud u se proclame *cluster-head* et diffuse son statut. Ainsi, tout nœud dans la voisinage situé au plus à distance de R sauts et qui n'a pas encore adhéré à un *cluster-head* devient membre du *cluster* du nœud u. Sinon, si les nœuds du voisinage de u étaient déjà en phase d'initialisation, alors u se rattachera au *cluster-head* le plus porche situé au plus à R sauts. Si tous les *cluster-heads* déjà élus se trouvent à une distance de plus de R sauts, alors u se déclare *cluster-head* et attire dans son *cluster* tous les nœuds membres de *cluster-heads* plus éloignés. L'algorithme de Lin et Chu suppose que deux *cluster-heads* doivent être obligatoirement séparer d'une distance de $R+1$ sauts. Ainsi, si avec la mobilité des nœuds, deux *cluster-heads* devenaient voisins à moins de R sauts, celui avec le plus petit identifiant abandonne son statut et tous les nœuds de son *clusters* chercheront à élire un autre *cluster-head*.

L'algorithme de Lin et Chu est très couteux du fait des échanges de messages pour construire les *clusters* et leurs maintenances en cas de modifications topologiques. En effet, il présente une complexité en messages de $O(n)$, avec n étant la taille du réseau.

Max-Min : algorithme non-recouvrant sur une métrique fixe à base d'identifiant de Amis et *al.* [APVH00]

Dans [APVH00], Amis et *al.* ont proposé une heuristique, connue sous le nom de Max-Min, qui construit des *clusters* non-recouvrants à d sauts où d est un paramètre fournit à l'algorithme. Max-Min utilise le critère de l'identifiant pour l'élection du *cluster-head* et se déroule en quatre phases logiques. La première consiste à propager les identifiants des nœuds les plus grands dans le *d-neighborhood* c'est-à-dire le voisinage jusqu'à une distance de d sauts via une technique dite de *FloodMax*. La deuxième consiste à propager les identifiants les plus petits dans le *d-neighborhood* grâce à une technique dite de *FloodMin*. Dans la troisième phase, les *cluster-heads* sont élus. Enfin, dans la quatrième phase, les différents *clusters* construits sont reliés (*Linkage*) entre eux. Le *FloodMax* se déroule en d rounds où tous les identifiants des nœuds sont propagés jusqu'à d sauts. Durant chaque round, chaque nœud compare les identifiants reçus et désigne comme "WINNER" le nœud ayant le plus grand des identifiants dans une distance de d sauts. La *FloodMin* est similaire à la *FloodMax* excepté que les plus petits identifiants des "WINNER' sont propagés à distance d sauts. Et les plus petits identifiant parmi les plus grands sont retenus dans cette deuxième phase. Dans la troisième phase, les *cluster-heads* sont élus en se fondant sur les identifiants des "WINNER" collectés lors des deux précédentes phases. Un nœud est élu comme *cluster-head* s'il était sorti vainqueur, c'est-à-dire retenu comme "WINNER", lors de la deuxième phase. Enfin, dans la dernière phase, des passerelles sont désignées pour la liaison des différents *clusters*. Pour cela, les nœuds situés aux périphériques des *clusters* envoient vers leurs *cluster-heads* un message contenant leurs identifiants ainsi que ceux de leurs voisins dans les périphéries des autres *clusters*.

Les auteurs de Max-Min ont comparé, par des simulations, leur approche à LCA [EWB87].

Ils ont notamment montré que Max-Min construit moins de *clusters*, de plus grandes tailles et les *cluster-heads* ont une plus longue durée de vie comparé à LCA. Cependant, Max-Min nécessite un important trafic pour construire les *clusters* et cela augmente fortement la surcharge et la latence du réseau. En effet, Max-Min présente une complexité en temps de $O(k)$ et en messages de $O(k \times n)$, avec k étant le rayon maximal des *clusters* et n la taille du réseau.

Algorithme non-recouvrant sur une métrique fixe à base d'identifiant virtuel de Amis et Prakash [AP00]

Dans [AP00] Amis et Prakash ont proposé une amélioration de Max-Min afin d'apporter plus d'équité entre les nœuds dans l'élection du *cluster-head*. Les auteurs estiment que, comme les *cluster-heads* sont les nœuds du réseau les plus sollicités, un nœud occupant longtemps la fonction de *cluster-head* épuise très rapidement ses ressources énergétiques. Ainsi, pour permettre une rotation équitable de la fonction de *cluters-head* entre tous les nœuds du réseau, Amis et Prakash utilisent une file d'attente circulaire associée à un principe d'identifiant virtuel (VID) attribué à chaque nœud. Chaque nœud u initialise la valeur de son identifiant virtuel à son propre identifiant ($VID_u = id_u$). Au premier tour, les nœuds possédant les plus grandes valeurs de VID sur la file d'attente circulaire situés dans une distance de d sauts sont élus comme *cluster-head*. Ils resteront *cluster-heads* durant une période τ. Tous les autres nœuds non *cluster-heads* incrémentent d'une unité leurs valeurs VID. Amis et Prakash ont fixé une limite maximale MAX_COUNT qui une fois atteinte, la VID_u de tout nœud u est ré-initialisée à 1 au prochain round. A l'issue de la durée τ, les *cluster-heads* ré-initialisent leurs valeurs de VID à 0. Puis, les autres nœuds possédant les plus grandes valeurs de VID sur la file d'attente circulaire dans une distance de d sauts sont élus *cluster-heads* pour une nouvelle durée τ. En cas d'égalité de valeur de VID ou lorsque deux *cluster-heads* sont voisins alors le nœud avec le plus grand identifiant l'emporte au détriment de l'autre. Amis et Prakash proposent aussi d'utiliser le degré pour départager deux nœuds avec des valeurs de VID égales.

L'algorithme de Amis et Prakash, bien qu'offrant une équilibrage de charge, nécessite une forte synchronisation pour le fonctionnement de la file d'attente circulaire. Or, cette synchronisation requiert beaucoup d'échanges de messages. En effet, comme Max-Min, l'algorithme de Amis et Prakash présente une complexité en temps de $O(k)$ et en messages de $O(k \times n)$, avec k étant le rayon maximal des *clusters* et n la taille du réseau.

Algorithme non-recouvrant sur une métrique fixe à base *spanning tree* de Fernandess et Malkhi [FM02]

Fernandess et Malkhi ont présenté dans [FM02] un algorithme de structuration qui construit des *clusters* non-recouvrants à k sauts. Cet algorithme est constitué de deux phases. Dans la première phase, partant d'un graphe connexe, les auteurs construisent un arbre couvrant du réseau (*spanning tree*) en s'inspirant de l'algorithme de construction de MCDS (*Minimum Connected Dominating Set*) proposé par Alzoubi et *al.* dans [AWF02]. Un MCDS d'un graphe représente le plus petit sous-ensemble de nœuds du graphe tel que tout autre nœud du graphe n'appartenant pas au MCDS est voisin d'au moins un membre du MCDS. Dans la seconde phase, Fernandess et Dahlia partitionnent l'arbre couvrant en sous arbres *k-subtree*, de diamètre au plus $2k$. Et chaque sous arbre est considéré comme un *cluster*. Cet algorithme présente une

complexité en temps de $O(n)$ et en message de $O(n \times \log(n))$ avec n étant le nombre de nœuds dans le graphe.

Fernandess et Dahlia n'ont pas abordé la maintenance des *clusters* en cas de modification topologique. De plus, la construction de l'arbre couvrant nécessite un important trafic.

k-lowestID : algorithme recouvrant sur une métrique fixe à base d'identifiant ou du degré de Nocetti et *al.* [NGS03]

Dans [NGS03], Nocetti et *al.* ont proposé une solution, nommé *k-lowestID*, qui généralise de l'algorithme de Lin et Gerla décrit dans [LG97] pour construire des *clusters* recouvrants à k sauts. Pour cela, Nocetti et *al.* supposent que tous les nœuds du réseau ont une vision du voisinage jusqu'à une distance de k sauts. Ainsi, tous les nœuds avec un identifiant plus petit dans un voisinage de k sauts déclenchent la procédure de *clustering* en se proclamant *cluster-heads* et en transmettant leurs nouveaux statuts aux voisins situés à une distance de 1 saut. Ensuite, ces derniers relayent les nouveaux statuts de proche en proche jusqu'à ce que la distance des k sauts soit atteinte. Un nœud u peut décider alors de se proclamer *cluster-head* ou de s'attacher à un autre *cluster-head* que lorsque tous ses voisins situés jusqu'à une distance de k sauts ayant un plus petit identifiant que lui ont diffusé leurs décisions d'être *cluster-heads* ou d'être membre d'un autre *cluster*. Si aucun nœud v ne se déclare *cluster-head* dans le voisinage jusqu'à une distance de k sauts du nœud u, alors u se déclare lui même *cluster-head*. Sinon, le nœud s'attache au *cluster-head* ayant le plus petit identifiant situé au plus à une distance de k sauts. Cependant, afin de réduire le nombre de *clusters* construits, c'est-à-dire dans le but de créer des *clusters* plus denses et en petit nombre, Nocetti et *al.* ont proposé d'utiliser le degré à la place de l'identifiant comme critère d'élection du *cluster-head*. Ainsi, un nœud ayant le degré le plus élevé dans son voisinage à distance de k sauts est élu *cluster-head*. En cas d'égalité des degrés entre deux nœuds, alors celui ayant le plus petit identifiant l'emporte au détriment de l'autre.

Cependant, le *k-lowestID* proposé par Nocetti et *al.*, du fait de la vision à distance de k sauts nécessaire pour construire les *clusters*, génère beaucoup de trafic. En effet, les statuts des *cluster-heads* sont retransmis jusqu'à une distance de k sauts. De plus, en cas de modifications topologiques, la totalité des structures est reconstruite. *k-lowestID* présente une complexité en temps et en messages de $O(n \times k)$, avec k étant le rayon maximal des *clusters* et n la taille du réseau.

Algorithmes utilisant une métrique variables pour l'élection du *cluster-head*

A-NCR : algorithme non-recouvrant sur une métrique variable à base de priorité de Yang et *al.* [YWC05]

Dans [YWC05], Yang et *al.* ont proposé un algorithme de *clustering* nommé A-NCR pour *Adjacency-based Neighbor Clusterhead selection Rule* qui construit des *clusters* non-recouvrants à k sauts. Cette solution étend l'algorithme de Wei et Lou décrit dans [WL03] et se fonde sur la priorité des nœuds pour élire les *cluster-heads*. Les nœuds qui ont la plus forte priorité dans une

distance de k sauts (ne sont pas concernés les nœuds déjà membres d'un *cluster*) se déclarent *cluster-heads* et diffusent un message de déclaration de *cluster-heads*. Les autres nœuds non prioritaires collectionnent tous les messages de déclaration de *cluster-heads* afin d'en choisir un et devenir membre de son *cluster*. Si un nœud u non prioritaire reçoit plusieurs déclarations de *cluster-head* situés dans une distance de k sauts alors u peut se fonder sur trois conditions pour en adopter un. (i) *ID-based* : choisir le *cluster-head* avec le plus petit identifiant. (ii) *Distance-based* : choisir le *cluster-head* le plus proche. En cas d'égalité de distance, la condition (i) fait la différence. (iii) *Size-based* : choisir le *cluster* de plus petit taille afin d'équilibrer la taille des structures. En cas d'égalité de taille, la condition (ii) puis (i) feront la différence.

Cependant, dans A-NCR, en cas de mobilisations topologiques, la totalité des structures est à reconstruire. De plus, A-NCR présente une complexité en temps et en messages de $O(n \times k)$, avec k étant le rayon maximal des *clusters* et n la taille du réseau.

Algorithmes utilisant une combinaison de métriques variables pour l'élection du *cluster-head*

Ici, nous présentons les algorithmes de *clustering* à k sauts utilisant une combinaison de métriques variables pour l'élection du *cluster-head*

ESAC,CES et ECFS : algorithmes non-recouvrants sur une combinaison de métriques variable de Lehsaini et *al.* [LGF08a]

Lehsaini et *al.* ont proposé dans [LGF08a] un algorithme de *clustering* nommé ESAC pour *Efficient Self-Organization Algorithm for Clustering* qui construit des *clusters* à k sauts (avec $k = 2$). Pour élire les *cluster-heads*, ESAC utilise un critère de poids dépendant de trois métriques variables : la *k-density* ($\mathcal{P}_{k-Density}$), l'énergie résiduelle ($\mathcal{P}_{Res-Energy}$) et la mobilité $\mathcal{P}_{Mobility}$. Ainsi, chaque nœud u calcule son poids $Weight(u)$ comme suit :

$$Weight(u) = \alpha \times \mathcal{P}_{k-Density} + \beta \times \mathcal{P}_{Res-Energy} + \gamma \times \mathcal{P}_{Mobility} \tag{2.8}$$

Avec

$$\alpha + \beta + \gamma = 1$$

Dans l'équation 2.8, les coefficients α, β et γ sont des constantes et représentent le degré d'influence de la métrique correspondante dans le calcul du poids $Weight(u)$ du nœud u. Selon les auteurs, ces coefficients dépendent de la nature de l'application pour laquelle le réseau est dédié. La métrique $\mathcal{P}_{k-Density}$ représente la densité du nœud u dans un voisinage à 2 sauts et est calculée comme étant le nombre de liens de communications dans un voisinage à 2 sauts sur le nombre de nœuds dans le voisinage à 2 sauts.

Dans ESAC, chaque nœud calcule son poids de façon distribuée. À l'initialisation, un nœud u quelconque pris aléatoirement dans le réseau initie le processus de *clustering* en diffusant un message *Hello*. Ainsi, tous les nœuds diffusent leurs poids dans leur voisinage à 2 sauts et collectent aussi ceux de leurs voisins à distance 2. Les nœuds avec les poids le plus élevé à distance 2 deviennent *cluster-heads*. Ensuite, les *cluster-heads* diffusent des messages *ADV_CH* afin d'avertir leurs voisinages à distance 2 de leur présence. C'est d'abord aux nœuds situés à une

distance de 1 saut de demander l'adhésion aux *cluster-heads* ayant les poids les plus élevés de leurs voisinages. Ces derniers vérifient la taille de leurs *clusters* avant de confirmer les demandes d'adhésion. En effet, ESAC construit des *clusters* de tailles comprises $Thres_{Lower}$ et $Thres_{Upper}$. Si la borne maximale $Thres_{Upper}$ n'est pas atteinte, alors les *cluster-heads* confirment aux nœuds situés à une distance de 1 saut leurs adhésions. Puis, cette même procédure se répète pour les nœuds à distance 2.

Pour la maintenance des *clusters*, Lehsaini et *al.* proposent qu'en cas de modifications topologiques, afin d'éviter une réaction en chaine, seuls les nœuds du *clusters* concernés participent à la procédure de *re-clustering*.

Les auteurs ont comparé, par des simulations, ESAC avec WCA [CKT02] et LCC [LCWG97]. Ils ont montré que ESAC présente une meilleure performance en termes de nombre moyen de *clusters* formés par rapport à WCA. De même, comparé à LCC, ESAC présente des changements de *cluster-heads* moins fréquents.

Dans [LGF08b], Lehsaini et *al.* ont étudié la consommation énergétique de ESAC [LGF08a]. Lors de cette étude, intitulée CES *(Cluster-based Energy-efficient Scheme for Mobile Wireless Sensor Networks)*, les auteurs ont comparé, par des simulations, la consommation énergétique et le volume de données transmis à la station de base de leur solution à ceux de LEACH [HCB00] et LEACH-C [HCB02]. Ils ont montré que leur algorithme présente de meilleurs performances en termes de consommation énergétique et de volume de données transmis. De plus, dans [LFG12], Lehsaini et *al.* ont proposé deux mécanismes (ECFS-1 et ECFS-2 (Efficient Cluster-based Fault-tolerant Schemes)) pour tolérer les fautes sur les canaux de communication et sur les nœuds.

Cependant, dans les solutions proposées dans [LGF08a, LGF08b, LFG12], pour construire les *clusters* à 2 sauts, les nœuds doivent retransmettre leurs informations dans tout le voisinage à 2 sauts. Cela entraine un trafic non négligeable dans le réseau. En effet, pour un réseau de n nœuds, ESAC,CES et ECFS présentent une complexité en messages de $O(n)$.

2.3.3 Synthèse

Nous venons de présenter, dans la Section 2.3, les solutions de *clustering* non auto-stabilisantes. Nous les avons classés et décrit en algorithmes de *clustering* en 1 saut et en k sauts respectivement dans la Section 2.3.1 et la Section 2.3.2. De plus, nous les avons étudiés selon que la métrique considérée pour élire les *cluster-heads* est fixe, variable ou une combinaison de critères fixe(s) et/ou variable(s). Le tableau 2.1 donne un récapitulatif de ces différentes solutions.

Les solutions de *clustering* non auto-stabilisantes offrent d'intéressantes techniques de structuration en construisant des *clusters* recouvrants et/ou non-recouvrants. Elles considèrent les aspects les plus fondamentaux tels que la consommation énergétique des nœuds, la mobilité de nœuds ou la densité du réseau, etc. Elles visent à améliorer la stabilité du réseau, à optimiser l'utilisation de la bande passante, à réduire la surcharge du réseau, etc.

Cependant, elles engendrent d'importants échanges de messages lors la construction et la maintenance des structures. Notamment, les algorithmes à k sauts requièrent d'avoir une vision et une connaissance sur le voisinage jusqu'à une distance de k sauts. Cela conduit à beaucoup d'échanges de messages dans le réseau. Or, les communications constituent la principale source de consommation de ressources et de saturation du réseau. En outre, les solutions de *clustering* non auto-stabilisantes nécessitent une initialisation correcte de nœuds et des canaux de communication afin de mener à bien la procédure de *clustering*. De même, elles ne gèrent pas les

	Solutions	Métrique	Critère d'élection	Recouvrant	Temps	Messages
1 saut	LCA [EWB87]	Fixe	ID	OUI et NON	$O(n)$	$O(n)$
	HCC [GTCT95]	Fixe	$\mathcal{D}eg$	OUI et NON	$O(n)$	$O(n)$
	LCC [LCWG97]	Fixe	ID ou $\mathcal{D}eg$	OUI et NON	$O(n)$	$O(n)$
	Lin et Gerla [LG97]	Fixe	ID	NON	$O(n)$	$O(n)$
	DCA et DMAC [Bas99]	Fixe	$Poids$	NON	$O(D+1)$	$O(n)$
	MBC [BP01]	Variable	$\mathcal{M}ob$	NON	–	$O(n)$ *
	MOBIC [BKL01]	Variable	$\mathcal{M}ob$	OUI	$O(n)$ *	$O(n)$ *
	LEACH [HCB00]	Variable	$\mathcal{P}rob$	NON	–	$O(n)$ *
	MPGC [LTCH06]	Variable	$\mathcal{E}ner$	OUI	$O(m+1)$ *	$O(n \times (m+1))$ *
	WCA [CKT02]	Combinaison	$\mathcal{D}eg$+$\mathcal{D}ist$+$\mathcal{M}ob$+$\mathcal{T}serv$	NON	$O(D)$ *	$O(n)$ *
	SBCA [AJRA08]	Combinaison	$\mathcal{E}ner$+$\mathcal{D}eg$+$\mathcal{S}tab$+ $\mathcal{M}em$	NON	–	$O(n)$ *
k sauts	Lin et Chu [LC00]	Non spécifique	–	NON	–	$O(n)$ *
	Max-Min [APVH00]	Fixe	ID	NON	$O(k)$	$O(k \times n)$ *
	Amis et Prakash [AP00]	Variable	Max VID	NON	$O(k)$	$O(k \times n)$ *
	MCDS [FM02]	Fixe	Spanning tree [AWF02]	NON	$O(n)$	$O(n \log(n))$
	Nocetti et *al.* [NGS03]	Fixe	ID ou Deg	OUI	$O(k \times n)$	$O(k \times n)$
	A-NCR [YWC05]	Variable	Priorité	NON	$O(k \times n)$ *	$O(k \times n)$ *
	ESAC [LGF08a]	Combinaison	$\mathcal{D}ens$+$\mathcal{E}ner$+$\mathcal{M}ob$	NON	–	$O(n)$ *
	CES [LGF08b]	Combinaison	$\mathcal{D}ens$+$\mathcal{E}ner$+$\mathcal{M}ob$	NON	–	$O(n)$ *
	ESFS [LFG12]	Combinaison	$\mathcal{D}ens$+$\mathcal{E}ner$	NON	–	$O(n)$ *

TABLE 2.1 – Récapitulatif des algorithmes de *clustering* non auto-stabilisants

Notice de lecture :

ID = Identifiant ; $\mathcal{D}eg$ = Degré ; $\mathcal{M}ob$ = Mobilité ; $\mathcal{D}ens$ = Densité ; $\mathcal{E}ner$ = Énergie ; VID = Identifiant virtuel ; $\mathcal{T}serv$ = Temps de service ; $\mathcal{S}tab$ = stabilité ; * Nous avons estimé ces complexités à partir du contenu des articles cités.

corruptions mémoires et les pannes transitoires pendant et après la phase de *clustering*.

Les solutions de *clustering* auto-stabilisantes ont été proposées pour une structuration du réseau dans le but d'optimiser les communications et de tolérer les pannes transitoires. Ainsi, dans la prochaine section, nous allons décrire les algorithmes de *clustering* auto-stabilisants proposés dans la littérature.

2.4 Structuration auto-stabilisante

Cette section aborde le *clustering* auto-stabilisant dans les réseaux ad hoc. Nous étudions d'abord les solutions fondées sur un modèle à états en les scindant en solutions à 1 saut et à k sauts. Ensuite, nous ferons de même pour celles utilisant un modèle à passage de messages. Enfin, nous présentons un résumé sur les approches de structuration auto-stabilisantes. Rappelons que nous avons présenté les deux modèles de communication (modèle à états et modèle à passages de messages) utilisés dans le *clustering* auto-stabilisant au niveau de la Section 1.1.5.

2.4.1 Solutions sur un modèle à états

Dans cette section, nous décrivons les solutions de *clustering* fondées sur un modèle à états. Pour cela, nous les classons en algorithmes de *clustering* à 1 saut et à k sauts. Nous présentons d'abord les solutions à 1 sauts, puis celles à k sauts.

Clustering auto-stabilisant sur un modèle à états à 1 saut

Johnen et Nguyen ont proposé dans [JN08] un algorithme auto-stabilisant fondé sur un modèle asynchrone à états pour construire des *clusters* à 1 saut de taille fixe. Afin de limiter la charge du *cluster-head*, les auteurs fixent un paramètre *SizeBound* qui représente le nombre maximal de nœuds dans un *cluster*. Le critère d'élection du *cluster-head* dans cet algorithme est un poids attribué à chaque nœud. Les auteurs supposent que ce poids peut être considéré comme pouvant être la puissance de transmission des nœuds, le niveau d'énergie, la bande passante, etc. Un nœud ayant le poids le plus élevé devient *cluster-head* et collecte dans son *cluster* jusqu'à *SizeBound* nœuds. Cet algorithme utilise un démon fortement équitable (*fair scheduler*). Johnen et Nguyen ont prouvé que partant d'une configuration quelconque, une configuration légale est atteinte en au plus $O(n)$ rounds (transitions) et nécessite une occupation mémoire de $O(\log(3n+5))$ bits par nœud, avec n étant le nombre de nœuds dans le réseau.

Dans [JN09], Johnen et Nguyen ont étendu leur proposition décrite dans [JN08] pour apporter la notion de robustesse pour améliorer la stabilité lors de la phase de *clustering* et d'accélérer la stabilisation. La robustesse est une propriété qui assure que partant d'une configuration quelconque, le réseau est partitionné après un round asynchrone. Ainsi, durant la phase de convergence, le réseau demeure toujours partitionné et vérifie un prédicat de sûreté. L'algorithme décrit dans [JN09] se fonde toujours sur un modèle asynchrone à états et fonctionne avec un démon fortement équitable (*fair scheduler*). Les auteurs ont prouvé que cet algorithme se stabilise en $O(D)$ rounds asynchrones et nécessite une occupation mémoire de $O(\log(2n+5))$ bits par nœud, avec D étant le diamètre du réseau et n étant le nombre de nœuds.

Dans [KM06], Kakugawa et Masuzawa ont proposé une version auto-stabilisante de l'algorithme DMAC de Basagni décrit dans [Bas99] afin de tolérer les fautes transitoires pouvant survenir dans ce dernier. L'algorithme de Kakugawa et Masuzawa est déterministe, fonctionne sous un modèle synchrone à états et utilise le critère de l'identité maximale pour la formation des *clusters*. En outre, un démon fortement équitable est utilisé afin de résoudre les problèmes d'accès concurrents. Ainsi, tous les nœuds mettent à jour leurs états simultanément dès qu'un seul d'entre eux entame une mise à jour de ses propres variables. Cet algorithme construit des *clusters* à 1 saut. Pour cela, il construit des *minimal dominating set* (*1-dominating*) définis comme étant les plus petits ensembles possibles dans lesquels deux *cluster-heads* ne peuvent pas être voisins. Kakugawa et Masuzawa ont prouvé qu'avec leur algorithme le système tend d'abord vers une configuration de *dominating set* en une étape puis converge vers une configuration optimale de *minimal dominating set* c'est-à-dire vers une structuration complète et optimale du réseau en $O(D)$ rounds et une occupation mémoire de $O(\log(2n+2))$ bits par nœud où D est le diamètre du réseau et n étant le nombre de nœuds.

Drabkin et *al.* ont proposé dans [DFG06] un algorithme de *clustering* qui construit des zones de couverture connexes (*connected overlays*). Les auteurs visent à construire des zones de couverture (*cluster*) contenant le moins de nœuds possible mais suffisant pour assurer une connectivité totale afin de permettre des échanges d'informations à l'intérieur de la zone couverte. Pour ce faire, ils ont adopté deux méthodologies de construction. La première consiste en deux actions parallèles ; l'une consiste à construire les *Maximal Independent Set* (MIS) qui vont constituer les zones de couverture et la seconde consiste à construire des ponts entre les différentes MIS. Pour la deuxième méthodologie, Drabkin et *al.* déterminent les *dominating set* en partant du principe que les dominants sont les ponts entre les nœuds qui ne partagent pas de zones de couverture. L'algorithme de Drabkin et *al.* utilise un modèle asynchrone à états comme technique de communication avec un démon faiblement équitable et distribué (*weakly fair and distributed scheduler*). Les auteurs ont prouvé que pour chacune de leurs méthodologies, l'algorithme se stabilise en $O(n)$ rounds et a une occupation mémoire de $O(\log(2n+2))$ bits par nœud avec n étant le nombre de nœuds du réseau.

Clustering auto-stabilisant sur un modèle à états à k sauts

Dans [DDL09], utilisant le critère de l'identité minimale, Datta et *al.* ont proposé *MINIMAL* qui est un algorithme auto-stabilisant et distribué de *clustering* à k sauts. *MINIMAL* utilise un modèle à états et fonctionne avec un démon faiblement équitable (*unfair scheduler*). Il considère un graphe $\mathcal{G}$ quelconque de n nœuds avec un identifiant unique attribué à chacun. Dans *MINIMAL* chaque nœud peut lire les états de ses voisins situés jusqu'à une distance k et est le seul à disposer des privilèges d'écriture sur ses propres états. *MINIMAL* procède en deux étapes. Il construit, dans un premier temps, un ensemble $\mathcal{D}$ qui représente un *k-dominating* set. D est défini tel que tout nœud n'appartenant pas à $\mathcal{D}$ se situe au plus à distance k d'au moins d'un membre de $\mathcal{D}$. Datta et *al.* ont prouvé que *MINIMAL* requiert $O(n)$ rounds pour la construction de $\mathcal{D}$. Dans un deuxième temps, considérant $\mathcal{D}$ comme l'ensemble de *cluster-heads*, *MINIMAL* structure le graphe $\mathcal{G}$ en *clusters* de rayon k. Les auteurs ont montré que, partant d'une configuration quelconque, MINIMAL converge vers une configuration terminale en $O(n^2)$ rounds. De plus, il requiert $O(\log(n))$ bits pour tout nœud du graphe $\mathcal{G}$.

Caron *et al.* ont proposé dans [CDDL10] un algorithme de *clustering* auto-stabilisant nommé *K-CLUSTERING*. Cet algorithme utilise un modèle à états et fonctionne avec un démon faiblement équitable (*unfair scheduler*). Les auteurs considèrent comme métrique un unique identifiant attribué à chaque nœud d'un graphe $\mathcal{G}$ et associé avec un poids attribué à chaque arrête du graphe. Caron *et al.* ont défini le *K-CLUSTERING* dans un graphe graphe $\mathcal{G}$ comme étant le partitionnement en *clusters* disjoints dans lesquels chaque nœud se situe au plus à distance k du *cluster-head*. Cette solution s'inspire partiellement de l'algorithme de Amis et *al.* proposé dans [APVH00]. Dans *K-CLUSTERING*, chaque nœud peut lire les états de ses voisins situés jusqu'à une distance de $k+1$ sauts et est le seul à disposer des privilèges d'écriture sur ses propres états. Les auteurs ont montré que cet algorithme s'exécute en $O(n \times k)$ rounds et nécessite $O(log(n) + log(k))$ bits par nœud, où n est le nombre de nœuds du réseau.

Un algorithme de *clustering* auto-stabilisant, asynchrone et distribué nommé *FLOOD* a été proposé dans [DLV10] par Datta et *al.* Le principe de *FLOOD* est similaire à celui de Max-Min proposé par Amis et *al.* proposé dans [APVH00]. *FLOOD* se fonde sur un modèle à états et fonctionne avec un démon faiblement équitable. Il construit des *k-dominating* set en utilisant la comparaison des identités des nœuds. Pour cela, les auteurs considèrent un graphe G de nœuds avec un identifiant unique attribué à chaque nœud. De plus, chaque nœud peut lire les états de ses voisins situés jusqu'à une distance $k+1$ et est le seul à disposer des privilèges d'écriture sur ses propres états. Datta et *al.* ont montré que FLOOD se stabilise en $O(k)$ rounds et nécessite une occupation mémoire de $O(k \times \log(n))$ bits pour chaque nœud du réseau.

2.4.2 Solutions sur un modèle à passage de messages

Dans cette section, nous décrivons les solutions de *clustering* auto-stabilisantes utilisant un modèle à passage de messages. Pour cela, nous les présentons suivant le fait qu'elles sont à 1 saut ou à k sauts.

Clustering auto-stabilisant sur un modèle à passages de messages à 1 saut

Dans [MBF04] Mitton et *al.* ont proposé un algorithme de *clustering* auto-stabilisant et distribué qui construit des *clusters* à 1 saut pour les réseaux ad hoc. La solution de Mitton et *al.* utilise un modèle asynchrone à passage de messages et construit des *clusters* disjoints. L'objectif visé par les auteurs est de minimiser les reconstructions des *clusters* (*re-clustering*) en cas de faible modification topologique. Pour cela, Mitton et *al.* ont introduit une nouvelle métrique d'élection du *cluster-head* appelée 1-*density* qui représente la densité de chaque nœud du réseau et est notée $\rho_1(u)$. Ils calculent cette 1-density comme suit :

$$\rho_1(u) = \frac{|e = (v, w) \in E, v \in \{u, \Gamma_1(u)\}, w \in \Gamma_1(u)|}{\delta_1(u)} \tag{2.9}$$

Dans l'équation 2.9, $\Gamma_1(u)$ est le nombre de liens de communication dans le voisinage du nœud et $\delta_1(u)$ le degré du nœud u. Donc, la 1-density $\rho_1(u)$ de chaque nœud u est calculée comme étant le nombre total de liens de communication dans le voisinage à distance 1 divisé par le nombre de voisins de u. Afin de calculer $\rho_1(u)$, chaque nœud diffuse son identifiant. Puis,

chaque nœud diffuse la liste de ses voisins. Ainsi, tout nœud calcule sa densité $\rho_1(u)$ et la diffuse dans son voisinage. Le nœud ayant la plus grande valeur de $\rho_1(u)$ devient *cluster-head*. En cas d'égalité, la sélection est faite selon les identifiants des nœuds.

Un deuxième algorithme auto-stabilisant et distribué de *clustering* à 1 saut a été proposé par Flauzac et *al.* dans [OHN09]. Cet algorithme utilise un modèle asynchrone à passage de messages et construit des *clusters* disjoints. Les auteurs modélisent le réseau comme un graphe $G = (E, V)$ où V désigne l'ensemble des nœuds du réseau dont à chacun est attribué un unique identifiant et E représente l'ensemble des arêtes. Le critère d'élection du *cluster-head* est l'identité maximale. Pour cela, les nœuds s'échangent un seul type de message appelé *hello* contenant trois informations qui déterminent l'état du nœud. Ces trois informations sont : l'identité du nœud, l'identité du *cluster-head* auquel appartient le nœud et son statut. En effet, Flauzac et *al.* proposent trois statuts de nœuds différents dans leurs réseaux à savoir les *cluster-heads* qui sont chargés d'orchestrer les communications au sein des *clusters*, les nœuds passerelles qui permettent les communications entre *clusters* voisins et les nœuds membres. L'avantage de cet algorithme est qu'il combine la découverte de voisinage et la construction des *clusters* en une seule phase. Sur la base des messages *hello*, les nœuds de plus grands identifiants deviennent *cluster-heads*. Flauzac et *al.* ont prouvé que cet algorithme se stabilise en $D + 2$ rounds (transitions) et nécessite $\log(2n + 3)$ bits par nœud.

Dans [KYSI11], Kuroiwa et *al.* proposent une version dans un modèle synchrone à passage de messages de l'algorithme de Johnen et Nguyen décrit dans [JN08]. Les auteurs visent à apporter une meilleure stabilité dans les *clusters*. La stratégie de Kuroiwa et *al.* consiste à prédire le nœud le plus approprié à rester le plus longtemps possible *cluster-head* en fonction de sa mobilité. Pour cela, ils considèrent un modèle de mobilité où les nœuds se déplacent en groupe. Dans ce cas, si chaque groupe forme un *cluster*, alors un *cluster-head* choisit parmi les nœuds de ce groupe va demeurer le plus longtemps possible inchangé. Pour élire un *cluster-head*, Kuroiwa et *al.* utilisent une métrique qui est fonction du centre de gravité du nœud dans le groupe auquel il appartient, de la taille du groupe ainsi que du vecteur de mouvement du nœud. La méthode proposée par Kuroiwa et *al.* améliore de 50% la stabilité des *clusters* comparé à l'algorithme de Johnen et Nguyen décrit dans [JN08] .

Clustering auto-stabilisant sur un modèle à passages de messages à k sauts

Dans [MFGLT05], Miton et *al.* reprennent le même principe qu'ils décrivent dans [MBF04] pour proposer un algorithme robuste de *clustering* auto-stabilisant, déterministe et distribué pour construire des *clusters* disjoints à k sauts. Mitton et *al.* utilisent toujours un modèle asynchrone à passage de messages. Pour élire le *cluster-head*, les auteurs utilisent comme métrique la *k-density* qui est une généralisation de la 1-*density* proposé sans [MBF04]. Cette *k-density* est déterminée à l'aide de l'équation 2.10 suivante :

$$\rho_k(u) = \frac{|e = (v, w) \in E, v \in \{u, \Gamma_k(u)\}, w \in \Gamma_k(u)|}{\delta_k(u)} \tag{2.10}$$

En d'autres termes, la *k-density* représente le nombre total de liens de communication dans le voisinage à une distance de k sauts du nœud u divisé par le nombre total de voisins qui

se trouvent à une distance de k sauts. Pour calculer la k-density, chaque nœud du réseau a besoin d'une vision de son voisinage à distance $k+1$ sauts ($\{k+1\}$-*Neighborhood view*). Par conséquent, les informations sur chaque nœud sont propagées jusqu'à $k+1$ sauts. Ensuite, la *k-density* calculée par chaque nœud est ensuite diffusée dans le voisinage. Ainsi, les nœuds avec la plus grande valeur de *k-density* deviennent *cluster-heads*. Mitton et *al.* ont comparé leur solution à l'algorithme de Max-Min proposé par Amis et *al.* dans [APVH00] et à celui de *Highest-Connectivity* proposé par Nocetti et *al.* dans [NGS03]. Ils ont montré l'efficacité de l'auto-stabilisation dans le *clustering* comparé aux solutions non auto-stabilisantes proposées dans [APVH00, NGS03] (cf. Section 2.3.2). Ils ont notamment montré que leur approche de *clustering* auto-stabilisante apporte plus de stabilité, un meilleur passage à l'échelle et est plus robuste contre les pannes transitoires comparé à [APVH00, NGS03]. Cependant, la solution de Mitton et *al.*, du fait de la vision du voisinage à distance $k+1$ sauts ($\{k+1\}$-*Neighborhood view*) requise pour calculer la *k-density*, nécessite un important échange de messages.

Larsson et Tsigas ont proposé dans [LT11] le premier algorithme distribué, auto-stabilisant avec multiples chemins, nommé (x,k)-clustering. Cet algorithme assigne, si possible, x *cluster-heads* dans une distance de k sauts à tout nœud du réseau. Les auteurs utilisent un modèle synchrone à passage de messages et procèdent par rounds. Ils ont montré que leur algorithme attribue, si possible, x *cluster-heads* à chaque nœud en $O(k)$ rounds. Puis, l'ensemble des *cluster-heads* se stabilise, avec une forte probabilité, en un minimum local de $O(g \times k \times \log(n))$ rounds. De plus, il nécessite $O(|G_u^k| \times (\log(n) + \log(k)))$ bits pour chaque nœuds u du réseau, où n représente la taille du réseau, k le nombre maximal de sauts et g une borne supérieure du nombre de nœuds dans un *cluster*. Cette approche génère beaucoup de messages. En effet, les messages de chaque nœuds sont retransmis jusqu'à une distance k.

2.4.3 Synthèse

Nous venons de présenter, dans la Section 2.4, les algorithmes de *clustering* auto-stabilisants proposés dans la littérature. Ces derniers, contrairement à leurs homologues non auto-stabilisants présentés dans la Section 2.3, ne nécessitent aucune initialisation des variables de nœuds et des canaux de communication. Les nœuds d'un système auto-stabilisant sont capables, de façon autonome et distribuée, de s'auto-organiser, de vérifier la cohérence de leurs données et états, de détecter et corriger les fautes transitoires. C'est pour cela qu'un algorithme auto-stabilisant est une solution efficace pour la structuration et la tolérance aux pannes transitoires dans un système distribué comme les réseaux ad hoc [JM11].

Comme nous venons de le décrire, dans le domaine du *clustering* auto-stabilisant, il existe deux classes d'approches différentes selon le modèle de communication utilisé : celle qui utilise un modèle à états [DFG06, KM06, JN08, JN09, DDL09, CDDL10, DLV10] et celle qui se fonde sur un modèle à passages [MBF04, MFGLT05, OHN09, LT11, KYSI11]. Le tableau 2.2 résume les caractéristiques de ces différents algorithmes selon le modèle de communication, le diamètre des *clusters* construits, la métrique utilisée pour élire les *cluster-heads* et le voisinage considéré.

Dans chaque modèle de communication (modèle à états et modèle à passage de messages), contrairement aux solutions à 1 saut, les solutions à k ont été proposées afin d'éviter de

	Solutions	Modèle	Diamètre	Métrique	Voisinage
Structuration auto-stabilisante	Johnen et Nguyen [JN08]	à états	2	*Poids*	1 saut
	Johnen et Nguyen [JN09]	à états	2	*Poids*	1 saut
	Kakugawa et *al.* [KM06]	à états	2	ID	1 saut
	Drabkin et *al.* [DFG06]	à états	2	*Poids*	1 saut
	MINIMAL [DDL09]	à états	$2k$	ID	k sauts
	FLOOD [DLV10]	à états	$2k$	ID	k sauts
	K-CLUSTERING [CDDL10]	à états	$2k$	ID+*Poids*	$k+1$ sauts
	Mitton et *al.* [MBF04]	à passage de messages	2	1−*density*	1 saut
	Flauzac et *al.* [OHN09]	à passage de messages	2	ID	1 saut
	Kuroiwa et *al.* [KYSI11]	à passage de messages	2	$\mathcal{M}$*obilté*	1 saut
	Larsson et Tsigas [LT11]	à passage de messages	$2k$	ID	k sauts
	Mitton et *al.* [MFGLT05]	à passage de messages	$2k$	k-*density*	$k+1$ sauts

TABLE 2.2 – Récapitulatif des algorithmes de *clustering* auto-stabilisants

Notice de lecture du tableau 2.2 :
ID : Identifiant des nœuds.
k : Distance maximale dans les *clusters*.
n : Nombre de nœuds dans le réseau.
1-density : Densité dans un voisinage à distance 1.
k-*density* : Densité dans un voisinage à distance k.

construire un grand nombre de *clusters* de tailles petites. En effet, des *clusters* de petite taille sont sensibles aux pannes transitoires et notamment à la mobilité des nœuds. Dans des *clusters* à 1 saut, le déplacement des nœuds, surtout celui des *cluster-heads*, peut occasionner une reconstruction des structures. Cependant, les solutions de structuration à k sauts, utilisant un modèle à états comme un modèle à passage de messages, nécessitent une vision large du voisinage pour la construction des *clusters*. Plus précisément, la solution dans un modèle à états décrite dans [CDDL10] requiert un rayon de lecture jusqu'à une distance de $k+1$ sauts et celles décrites dans [DDL09, DLV10] demandent un rayon de lecture à une distance de k sauts. Quant aux solutions sur un modèle à passage de messages proposées dans [MFGLT05] et [LT11], lors de la phase de structuration, les informations sur les nœuds sont retransmises respectivement jusqu'à une distance de $k+1$ et k sauts. Cela entraine une importante surcharge et consommation des ressources du réseau.

S'agissant de l'évaluation des performances dans les pires cas, pour les approches décrites dans [KM06, JN08, DFG06, JN09, OHN09, DDL09, CDDL10, DLV10, LT11], les auteurs ont établi la preuve formelle de la propriété de convergence. Dans [OHN09, DDL09, CDDL10, DLV10, LT11], les auteurs ont également établi la preuve formelle de l'occupation mémoire requise pour l'exécution de l'algorithme. Quant aux approches [KM06, DFG06, JN08, JN09], à partir des différentes variables utilisées dans des algorithmes, nous avons déduit l'occupation mémoire nécessaire pour chaque nœud. Nous résumons dans le tableau 2.3 les différents temps de stabilisation comme étant le nombre de transitions nécessaire pour la stabilisation du réseau en partant d'une configuration quelconque et les occupations mémoires comme étant le nombre de bits nécessaire pour encoder les différentes variables utilisées pour un nœud durant l'exécution

de l'algorithme.

Nous remarquons que les solutions auto-stabilisantes qui ont été proposées présentent souvent des temps de stabilisation élevés notamment celles à k sauts décrites dans [DDL09, CDDL10, LT11]. De même, le temps de stabilisation des algorithmes de *clustering* à k sauts proposés dans [CDDL10, DLV10, LT11] est fonction du paramètre k. Pour les solutions décrites dans [DDL09, LT11], étant donné que les auteurs construisent d'abord l'ensemble des *cluster-heads* puis forment les *clusters*, il existe deux temps de stabilisation. Nous pouvons souligner que la solution de *clustering* à k sauts décrite dans [LT11] qui se fonde sur un modèle synchrone à passage de messages nécessite une importante occupation mémoire par nœud. En effet, chaque nœud stocke les informations sur l'ensemble de ses voisins situés jusqu'à une distance de k sauts.

	Solutions	**Temps de stabilisation***	**Occupation mémoire****	**Voisinage**
1 saut	Kakugawa et *al.* [KM06]	$O(D)$	$O(\log(2n+2))$	1 saut
	Drabkin et *al.* [DFG06]	$O(n)$	$O(\log(2n+2))$	1 saut
	Johnen et Nguyen [JN08]	$O(n)$	$O(\log(3n+5))$	1 saut
	Johnen et Nguyen [JN09]	$O(D)$	$O(\log(2n+5))$	1 saut
	Flauzac et *al.* [OHN09]	$D+2$	$\log(2n+3)$	1 saut
k sauts	MINIMAL [DDL09]	$O(n)\,;\, O(n^2)$	$O(\log(n))$	k sauts
	FLOOD [DLV10]	$O(k)$	$O(k \times \log(n))$	k sauts
	Larsson et Tsigas [LT11]	$O(k)\,;\, O(g \times k \times \log(n))$	$O(\lvert G_u^k \rvert \times (\log(n)+\log(k)))$	k sauts
	K-CLUSTERING [CDDL10]	$O(n \times k)$	$O(log(n)+log(k))$	k+1 sauts

TABLE 2.3 – Comparaison formelle des temps de stabilisation et des occupations mémoires

Notice de lecture du tableau 2.3.
k : Distance maximale dans les *clusters*.
D : Diamètre du graphe.
n : Nombre de nœuds dans le réseau.
g : Borne maximale du nombre de nœuds dans les *clusters*.
G_u^k : Voisinage du nœud u.
* Le temps de stabilisation est exprimé en *rounds* aussi dit *transitions* ou *étapes*.
** Il s'agit l'occupation mémoire pour encoder les variables d'un nœud. Elle est exprimée en bits.

Les auteurs des approches [OHN09], [CDDL10] et [LT11], en plus d'établir par preuve formelle des temps de stabilisation et occupations mémoires de leurs algorithmes dans les pires cas, ont aussi mené des simulations dans le but d'évaluer les performances moyennes de leurs algorithmes. Cependant, pour les solutions décrites dans [MBF04, MFGLT05, KYSI11], les performances des algorithmes ont été évaluées par des simulations.

En résumé, nous pouvons retenir, d'une part, que les approches fondées sur un modèle à états permettent de se dégager complètement des problèmes liés aux communications et d'éviter l'utilisation de protocoles de communication. Cependant, dans les réseaux ad hoc, il est primordiale de s'intéresser aux problèmes liés aux communications. D'autre part, les solutions sur un modèle à passage de messages, notamment celles à k sauts, du fait du voisinage à distances de k ou $k+1$ sauts considéré, peuvent engendrer d'importants échanges de messages

dans le réseau causant ainsi une saturation et une sûr-consommation de ressources. Partant de ces observations, nous allons introduire notre problématique de recherche dans la prochaine section.

2.5 Problématique de recherche

Partant des constatations sur les solutions de *clustering* non auto-stabilisantes (cf. Section 2.3.3) comme celles auto-stabilisantes (cf. Section 2.4.3), nous nous proposons dans le cadre de nos travaux de recherche la problématique suivante :

Problématique

Élaborer une nouvelle solution simple et efficace de structuration des réseaux ad hoc dans le but d'optimiser les communications et de tolérer les pannes transitoires.

Pour répondre à notre problématique, la solution que nous élaborons présente les spécificités suivantes.

Structuration auto-stabilisante, distribuée, déterministe et fondée sur un modèle asynchrone à passage de messages

Dans le but de tolérer les pannes transitoires, de ne pas avoir recours à aucune initialisation ni intervention extérieure, nous optons pour une approche auto-stabilisante. Comme notre solution vise à être réaliste, nous adoptons un modèle asynchrone à passage de messages pour les communications. De même, afin de garantir un choix déterministe et sans équivoque lors de l'élection du *cluster-head*, nous optons principalement comme métrique les identités associées aux nœuds.

Structuration non-couvrante à k sauts

Partant du constat que les solutions à 1 saut construisent un nombre assez conséquent de *clusters* de petite taille, très sensibles aux changements topologiques et autres pannes transitoires, nous optons pour une construction de *clusters* à k sauts, avec k étant le rayon maximal des *clusters* et fixé au préalable. Donc, chaque nœud se trouvant dans un *cluster* est situé au plus à une distance de k sauts de son *cluster-head*. Également, pour une prise de décision simple, efficace et rapide lors de l'acheminement des données nous optons pour la construction de *clusters* non-recouvrants. Ainsi, chaque nœud est membre de un et un seul *cluster* et sait vers quel *cluster-head* envoyer ses données sans aucune ambigüité.

En résumé, l'approche de *clustering* que nous proposons est distribuée, auto-stabilisante, fondée sur un modèle asynchrone à passage de messages et construit des *clusters* non-recouvrants à k sauts. La figure 2.11 illustre schématiquement le positionnement de nos travaux de recherche dans le vaste domaine du *clustering* dans les réseaux ad hoc.

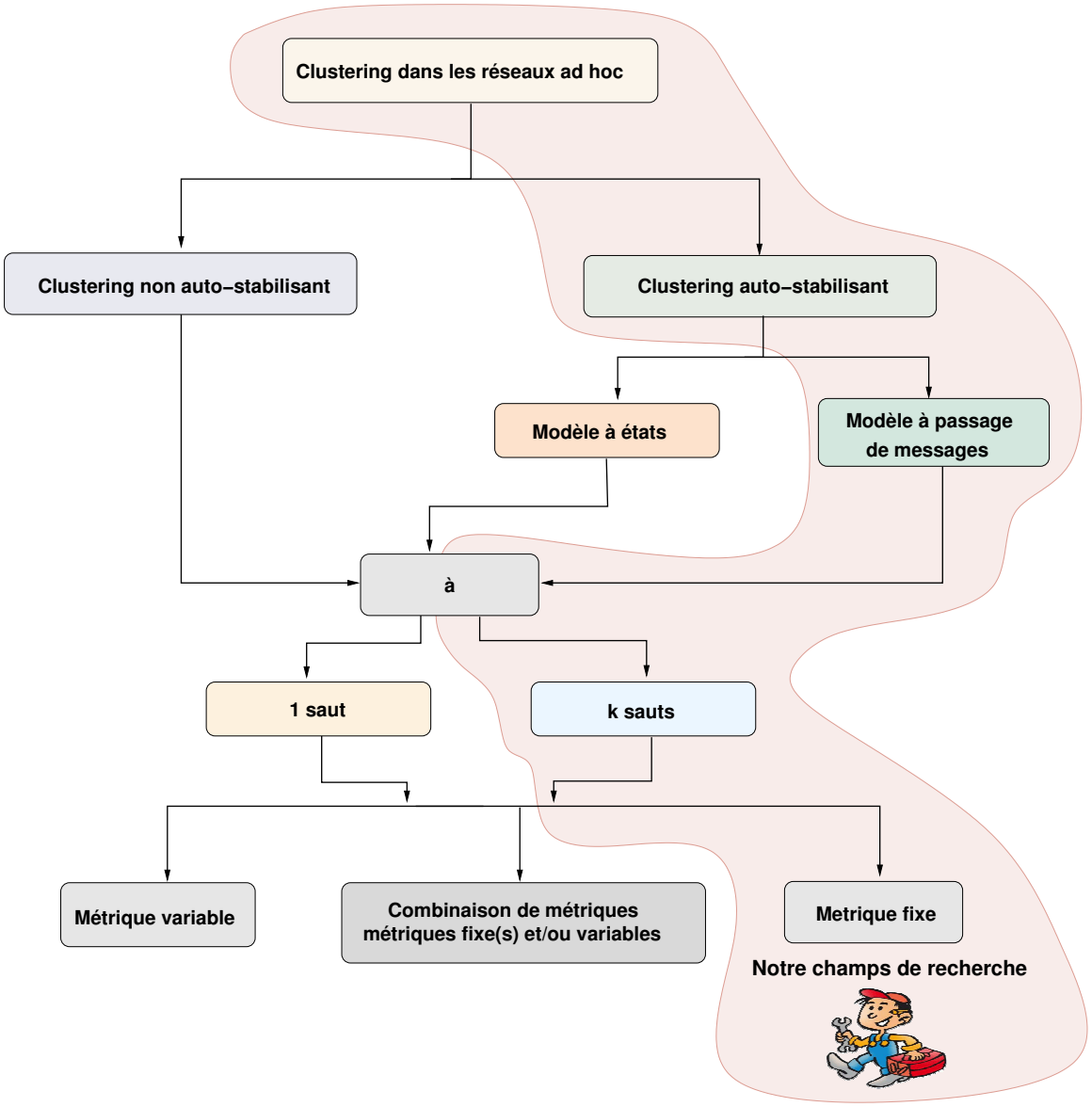


FIGURE 2.11 – Illustration schématique du positionnement de nos travaux de recherche

Vers les réseaux de capteurs sans fils avec contrainte énergétique

Toujours dans un souci d'élaborer une solution adaptée à la réalité et aux conditions d'un réseau ad hoc, nous appliquons notre solution de *clustering* dans le contexte des réseaux de capteurs sans fil (RCSF) avec contrainte énergétique. Dans de tels réseaux, nous étudions la consommation énergétique de notre solution aussi bien lors de la procédure de *clustering* qu'en présence de pannes de transitoires. Nous faisons cette étude suivant le critère d'élection du *cluster-head*. En d'autres termes, nous étudions les performances de notre approche de structuration en considérant d'autres critères d'élection des *cluster-heads* tels que le degré ou l'énergie des nœuds.

Utilisation des *clusters* pour l'acheminement de l'information

Comme notre finalité est l'acheminement de l'information dans le réseau, en partant d'un réseau que nous structurons en *clusters*, nous proposons plusieurs techniques de routage. L'idée est d'arriver à différentes manières d'acheminer l'information en intégrant différents niveaux d'agrégation.

2.6 Conclusion

Dans ce chapitre, nous avons étudié des solutions de structuration des réseaux ad hoc existantes en les classifiant en deux catégories : (*i*) les algorithmes de *clustering* non auto-stabilisants pour structurer le réseau et (*ii*) leurs homologues auto-stabilisants pour structurer le réseau et tolérer les pannes transitoires. Partant des constatations soulevées dans l'étude de ces différentes solutions, nous nous sommes proposé d'élaborer une nouvelle approche simple et efficace de structuration des réseaux ad hoc dans le but d'optimiser les communications et de tolérer les pannes transitoires. Ainsi, dans le prochain chapitre, nous présenterons la nouvelle approche auto-stabilisante pour une structuration des réseaux ad hoc que nous proposons.

Chapitre 3

SDEAC : une approche de structuration auto-stabilisante des réseaux ad hoc

Résumé.

Ce chapitre présente SDEAC (self-Stabilizing Distributed Energy-Aware k-hops Clustering), un algorithme original de structuration auto-stabilisant des réseaux ad hoc que nous proposons dans le but de réduire les communications et de tolérer les pannes transitoires. Nous commençons d'abord par exposer nos motivations, les spécifications et notations de SDEAC, son principe d'exécution et sa description formelle. Ensuite, nous établissons la preuve formelle des propriétés de convergence et de clôture ainsi que l'occupation mémoire nécessaire. Puis, nous étudions par observation le comportement de SDEAC suite à l'occurrence de pannes transitoires causées par une modification topologique. Enfin, par une campagne de simulations sous OMNeT++, nous évaluons le temps de stabilisation moyen de SDEAC ainsi que l'impact de pannes transitoires.

Les travaux que nous exposons dans ce chapitre ont fait l'objet de plusieurs publications dans des revues et conférences avec actes et comités de sélection : une (1) revue internationale [BFH+13a], deux (2) revues francophones [BFH+12a, BFH+14], deux (2) articles conférences en internationales [BFH+12b, BFH+13b] avec un ***Best Paper Award*** *dans [BFH+13b] et deux (2) articles en conférences francophones [BFH+12c, BFH+12d].*

Sommaire

3.1 Motivations et objectifs

Nous proposons un algorithme original de structuration des réseaux ad hoc qui construit des *clusters* non-recouvrants de diamètre au plus $2k$. Notre approche de *clustering*, que nous appelons *SDEAC* (self-Stabilizing Distributed Energy-Aware k-hops Clustering), est auto-stabilisante, déterministe, distribuée et utilise un modèle asynchrone à passage de messages. Elle ne nécessite aucune initialisation ni intervention extérieure et se fonde principalement sur le critère de l'identité maximale des nœuds pour l'élection des *cluster-heads*. La particularité de SDEAC est qu'elle combine la découverte du voisinage et la procédure de *clustering* en une seule phase. De plus, elle utilise uniquement une vision de voisinage à distance 1 (informations locales) pour construire des *clusters* à k sauts.

Nous présentons d'abord les détails de SDEAC c'est-à-dire la définition de ses différents concepts et notations, son principe d'exécution et sa description formelle. Ensuite, nous validons SDEAC par preuve formelle en montrant que les propriétés de convergence et de clôture sont vérifiées. Dans cette preuve, nous évaluons le temps de convergence de SDEAC ainsi que l'occupation mémoire requise pour son exécution. L'objectif est de comparer les performances théoriques (temps de stabilisation et occupation mémoire) de SDEAC avec celles des solutions décrites [DDL09, CDDL10, DLV10, LT11] bien que ces dernières utilisent des modèles de communication différents. Puis, par observation, nous étudions le comportement de SDEAC en présence de pannes transitoires causées par des modifications topologiques. Enfin, nous évaluons, par simulation sous OMNeT++ [1], les performances moyennes de SDEAC pour des topologies réseaux quelconques. Par performances moyennes, nous entendons le temps de stabilisation nécessaire pour atteindre une configuration légale en partant d'une configuration quelconque et sans occurrence de pannes transitoires. Nous mesurons également ce temps de stabilisation dans un environnement sujet à des pannes transitoires. Lors de nos évaluations de performances moyennes, nous considérons les paramètres les plus usuels dans les évaluations des

1. http ://www.omnetpp.org

algorithmes de *clustering* auto-stabilisants à savoir la taille du réseau, sa densité de connexité et le paramètre k.

La suite de ce chapitre est organisée comme suit. Nous commençons par présenter les spécifications de SDEAC et ses principaux concepts dans la Section 3.2. Ensuite, dans la Section 3.3, nous donnons le principe d'exécution de SDEAC ainsi que sa description formelle. Dans la Section 3.4, nous établissons la preuve formelle de la validité de SDEAC. Le comportement de SDEAC en présence de pannes transitoires (apparition ou disparation de nœuds) est étudié au niveau la Section 3.5. Dans la Section 3.6, nous menons une campagne de simulations avec le simulateur OMNeT++ dans le but d'évaluer les performances moyennes de SDEAC. Enfin, au niveau la Section 3.7, nous synthétisons l'ensemble des travaux présentés dans ce chapitre.

3.2 Spécifications de SDEAC

Dans cette section, nous présentons les spécifications de SDEAC et la définition de ses principaux concepts ainsi que ses principales notations.

3.2.1 Modèlisation

Nous considérons un réseau ad hoc comme un système distribué modélisé par un graphe avec identités connexe et non-orienté tel que nous l'avons présenté dans la Section 1.1.2 du Chapitre 2. Chaque nœud dispose d'une identité unique et l'ensemble des identités des nœuds du graphe est muni d'un ordre total. Nous optons pour un modèle de communication asynchrone à passage de messages. En effet, comme nous l'avons dit dans la Section 1.1.5 du Chapitre 1, notre choix est motivé par la validité de ce modèle ; validité due à sa similarité avec la réalité.

3.2.2 Définitions et notations

Ici, nous définissons les principaux concepts et notations utilisés dans SDEAC.

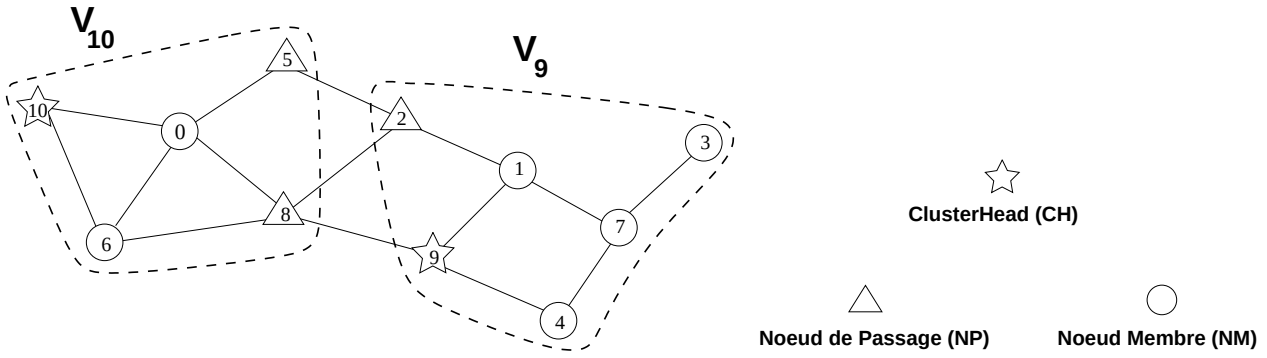
Nous définissons les notions de *cluster* et le *clustering* telles que considérées dans littérature et telles que nous l'avons présenté dans la Section 2.2.1 du Chapitre 2. Cependant, dans notre spécification, nous considérons que chaque *cluster* possède un unique identifiant définit comme suit.

Définition 3.1. ***(Identifiant du cluster)***
Chaque cluster possède un unique identifiant qui correspond à la plus grande identité de tous les nœuds du cluster. L'identifiant d'un cluster auquel appartient le nœud u est notée cl_u.

Dans nos *clusters*, nous avons trois types de nœuds avec des statuts différents. Le statut d'un nœud u est noté $statut_u$ et il définit le rôle que joue le nœud dans le *cluster*. Ainsi, un nœud peut être soit *Cluster-Head* (CH), soit *Nœud Membre* (NM), soit *Nœud de Passage* (NP). Nous définissions ces trois types de nœuds comme suit :

- **Cluster-Head** : il est le nœud possédant la plus grande identité dans le *cluster*. Donc l'identifiant du *cluster* (cf. Définition 3.1) correspond à celui du *cluster-head*. Le *cluster-head* est élu par l'algorithme de *clustering* et a pour rôle d'orchestrer les communications au sein du *cluster* ;
- **Nœuds de passage** : ils assurent l'interconnexion entre *clusters* adjacents et permettent les communications entre eux.
- **Nœud membre** : un nœud qui n'est ni *cluster-head* ni nœud de passage est alors qualifié de nœud membre d'un *cluster*.

Dans la suite ce document, comme illustré dans les figures 3.1(a) et 3.1(b), nous matérialisons un *cluster-head* par une *étoile*, un nœud de passage par un *triangle* et un nœud membre par un *cercle*.



(a) Structure en *clusters* (b) Statut des nœuds

FIGURE 3.1 – Structure des *clusters* et statuts des nœuds

Avant de donner la définition formelle des différents statuts des nœuds, nous définissons la notion de distance notée $d_{(u,CH_u)}$ qui représente le nombre minimal de sauts (plus court chemin) entre un nœud u et le *cluster-head* du *cluster* auquel il appartient. Comme illustré dans l'équation 3.1, cette distance est définie telle que :

$$\begin{cases} d_{(u,CH_u)} = 0, & \text{si } statut_u = CH. \\ 1 \leq d_{(u,CH_u)} \leq k, & \text{si } (statut_u = NM) \vee (statut_u = NP) \end{cases} \tag{3.1}$$

De plus, chaque nœud adopte un voisin $v \in N_u$, noté np_u, par lequel il passe pour atteindre son *cluster-head* situé au plus à distance k.

Après avoir défini la distance d'un nœud vis à vis de son *cluster-head*, nous définissons formellement les trois différents statuts comme suit. Rappelons que le statut d'un nœud dépend de son état qui est fonction de ses variables locales ainsi que celles de ses voisins.

Définition 3.2. *(**Statut des nœuds**)*

- *Cluster-Head (**CH**) : un nœud u a le statut de CH s'il possède le plus grand identifiant parmi tous les nœuds de son cluster :*
  - $statut_u = CH \iff \forall v \in V_{cl_u}, (id_u > id_v) \wedge (dist_{(u,v)} \leq k)$.
- *Nœud Membre (**NM**) : un nœud u a le statut de NM s'il n'est pas cluster-head et que tous ses voisins appartiennent au même cluster que lui :*

– $statut_u = NM \Longleftrightarrow (\forall v \in N_u, cl_v = cl_u) \wedge (\exists w \in V \mid (statut_w = CH) \wedge (dist_{(u,w)} \leq k))$.

- *Nœud de Passage (**NP**) : un nœud u a le statut de NP s'il existe au moins un nœud v dans son voisinage appartenant un autre cluster :*
  – $statut_u = NP \Longleftrightarrow \exists v \in N_u, (cl_v \neq cl_u)$.

Dans l'exemple la figure 3.1(a), le nœud 10 du *cluster* V_{10} possède le statut de CH. De même, le nœud 9 joue le rôle de cluster-head dans le *cluster* V_9. Les nœuds 0 et 6 sont des nœuds membres (statut NM) du *cluster* V_{10}. Pareillement, les nœuds 1, 3, 4 et 7 ont un statut de NM dans le *cluster* V_9. Le nœud 2, membre du *cluster* $V9$, sert de passerelle entre le *clusters* $V9$ et V_{10}. Les nœuds 5 et 8 assurent ce même rôle dans le *cluster* V_{10}.

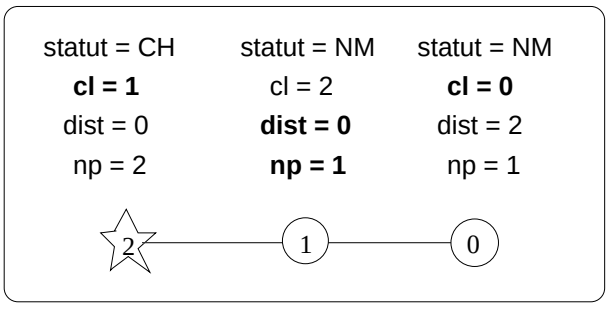
3.2.3 Cohérence des nœuds

La cohérence des nœuds est une notion fondamentale de notre spécification. Elle empêche les nœuds d'avoir des valeurs incorrectes dans leurs variables qui définissent leurs états. En effet, comme le *cluster-head* est le nœud avec la plus grande identité au sein du *cluster*, si un nœud u a le statut de CH alors son identifiant de *cluster* doit obligatoirement être égal à son propre identifiant, sa distance égale à 0 et la valeur de sa variable np_u égale à son propre identifiant. Par contre, pour un nœud u de statut NM ou NP, son identifiant de *cluster* et la valeur de sa variable np_u sont différents par rapport à la valeur de son propre identifiant. En outre, sa distance doit être différente de 0.

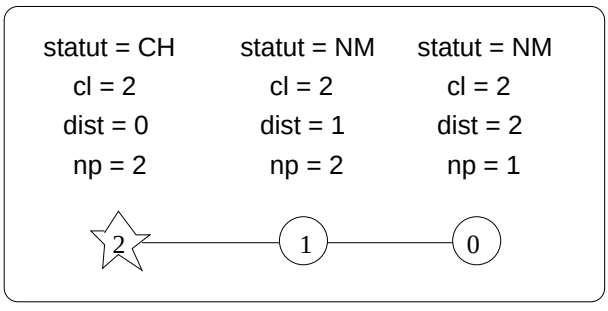
Définition 3.3. ***(Nœud cohérent)***
Un nœud u est cohérent si et seulement si, il est dans l'un des états suivants.

- *Si $statut_u = CH$ alors $(cl_u = id_u) \wedge (dist_{(u,CH_u)} = 0) \wedge (np_u = id_u)$.*
- *Si $statut_u \in \{NM, NP\}$ alors $(cl_u \neq id_u) \wedge (dist_{(u,CH_u)} \neq 0) \wedge (np_u \neq id_u)$.*

La figure 3.2(a) montre un exemple de nœuds dans des états incohérents. Cette incohérence est détectée et corrigée dans la première phase de notre algorithme que nous appelons *Gestion de la Cohérence*. Nous expliquerons cette dernière dans la section suivante. La figure 3.2(b) montre les nœuds respectant la définition 3.3, c'est-à-dire se trouvant dans un état cohérent.



(a) Nœuds incohérents



(b) Nœuds cohérents

FIGURE 3.2 – Notion de cohérence des nœuds

3.2.4 Stabilité des nœuds et du réseau

Dans notre spécification, tout nœud du réseau, quel que soit son statut, doit satisfaire un certain nombre de conditions pour être considéré comme stable. Donc, la stabilité des nœuds se définit comme suit.

Définition 3.4. *(**Nœud stable**)*
Un nœud u est dans un état stable si et seulement si, il est cohérent et satisfait à l'une des conditions suivantes :

- *Si* $statut_u = CH$ *alors* $\forall v \in N_u, (statut_v \neq CH) \wedge \{((cl_v = cl_u) \wedge (id_v < id_u)) \vee ((cl_v \neq cl_u) \wedge (dist_{(v,CH_v)} = k))\}$.
- *Si* $statut_u = NM$ *alors* $\forall v \in N_u, (cl_v = cl_u) \wedge (dist_{(u,CH_u)} \leq k) \wedge (dist_{(v,CH_v)} \leq k)$.
- *Si* $statut_u = NP$ *alors* $\exists v \in N_u, (cl_v \neq cl_u) \wedge \{((dist_{(u,CH_u)} = k) \wedge (dist_{(v,CH_v)} \leq k)) \vee ((dist_{(v,CH_v)} = k) \wedge (dist_{(u,CH_u)} \leq k))\}$.

De cette stabilité des nœuds dépend la stabilité du réseau. La stabilité du réseau est définie donc comme suit :

Définition 3.5. *(**Réseau stable**)*
Le réseau est stable si et seulement si tous les nœuds sont stables.

La figure 3.3 illustre un exemple de réseau stable avec le contenu des variables de chaque nœud. Une fois le réseau stable, alors une configuration légale est atteinte.

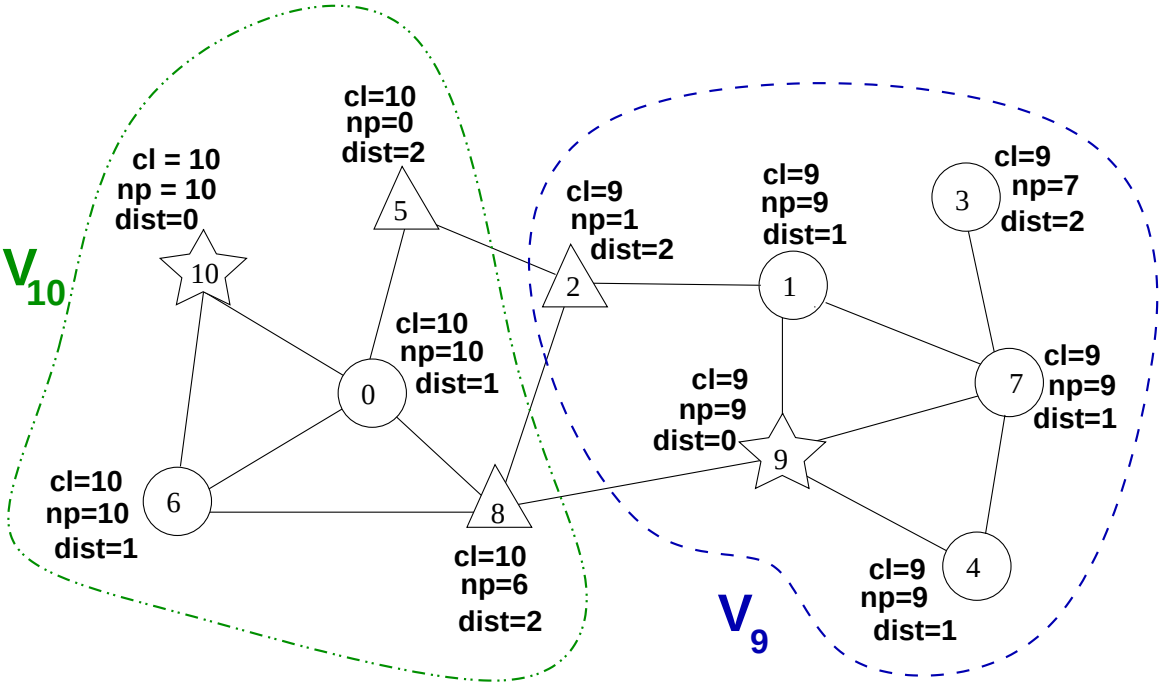


FIGURE 3.3 – Réseau stable

3.3 Présentation de SDEAC

Dans cette section, nous présentons d'abord le principe d'exécution SDEAC, puis nous donnons sa description formelle.

3.3.1 Principe d'exécution

SDEAC est auto-stabilisant, il ne nécessite ainsi aucune initialisation. Partant d'une configuration quelconque, avec seulement l'échange d'un seul type de message, les nœuds s'auto-organisent en *clusters* non-recouvrants au bout d'un nombre fini d'étapes. Ce message, dit *hello*, est échangé périodiquement entre chaque paire de nœuds voisins. Il contient les quatre informations suivantes : l'identité du nœud u expéditeur (id_u), son identifiant de *cluster* (cl_u), son statut ($statut_u$) et la distance ($dist_{(u,CH_u)}$) qui sépare le nœud u de son *cluster-head*. Rappelons que l'identifiant du *cluster* d'un nœud est égal à l'identifiant de son *cluster-head* (cf. Définition 3.1). Donc, la structure des messages *hello* est la suivante : $hello(id_u, cl_u, statut_u, dist_{(u,CH_u)})$. De plus, chaque nœud maintient une table de voisinage, notée $StateNeigh_u$, contenant l'ensemble des états de ses voisins. $StateNeigh_u[v]$ contient les états du nœud v voisin du nœud u. SEDAC se déroule comme suit.

Dès la réception d'un message *hello*, un nœud u exécute l'algorithme 1. Durant cet algorithme, u exécute trois étapes consécutives :

1. mise à jour de la table de voisinage ;
2. vérification de la cohérence des variables locales ;
3. procédure de *clustering*.

Étape de mise à jour et de vérification de la cohérence

À chaque réception d'un message, les nœuds effectuent une mise à jour après comparaison des données du message et celles présentes dans leurs tables de voisinage. Après la phase de mise à jour, chaque nœud vérifie sa cohérence. Comme le *cluster-head* est le nœud avec la plus grande identité au sein du *cluster*, si un nœud a le statut de CH alors son identifiant de *cluster* doit obligatoirement être égal à son propre identifiant. Dans la figure 3.2(a), le nœud d'identité 2 est *cluster-head*, son identifiant de *cluster* est égal à 1 donc le nœud 2 n'est pas cohérent. Tout comme les nœuds 1 et 0, ils ne vérifient pas la définition 3.3. Chaque nœud détecte et corrige son incohérence durant la phase de vérification de cohérence telle que définit dans l'algorithme 1. La figure 3.2(b) montre les nœuds dans des états cohérents.

Étape de *clustering*

Durant l'étape de *clustering*, chaque nœud compare son identité à celle de chacun de ses voisins. Ainsi :

- Un nœud u s'élit *cluster-head* s'il possède la plus grand identité parmi tous les nœuds de son *cluster* ;

- Si un nœud u découvre un voisin v avec une plus grande identité que la sienne, alors il devient membre du même *cluster* que v avec un statut de NM ;
- Si un nœud reçoit un message provenant d'un voisin membre d'un autre *cluster*, alors il devient un nœud de passage avec un statut de NP ;
- Comme les messages *hello* contiennent la distance entre chaque nœud u et son *cluster-head*, alors u peut déterminer si le diamètre maximal du *cluster* est atteint. Ainsi, u pourra donc choisir un autre *cluster* si les k *sauts* sont atteints ;
- À la fin de cette exécution, u envoie un message *hello* à ses voisins pour les informer de son changement d'état.

Illustration d'une transition

La figure 3.4 décrit une transition τ_i c'est-à-dire le passage d'une configuration γ_i à γ_{i+1}. Dans cet exemple, $k = 2$. A γ_i, chaque nœud envoie à ses voisins un message *hello*. γ_{i+1} est une configuration stable. Dès la réception de messages venant des voisins, chaque nœud met à jour sa table de voisinage puis exécute l'algorithme 1. Dans cet exemple, le nœud n_1 qui est un nœud membre du *cluster* de n_3 détecte le nœud n_0 comme un *cluster* voisin. Il devient nœud de passage avec un statut de NP et envoie un message *hello* à ses voisins pour une mise à jour. Les nœuds n_3, n_2 et n_0, en fonction de leurs états actuels ainsi que ceux de leurs voisins, ne changent pas d'état. γ_{i+i} correspond à un état stable.

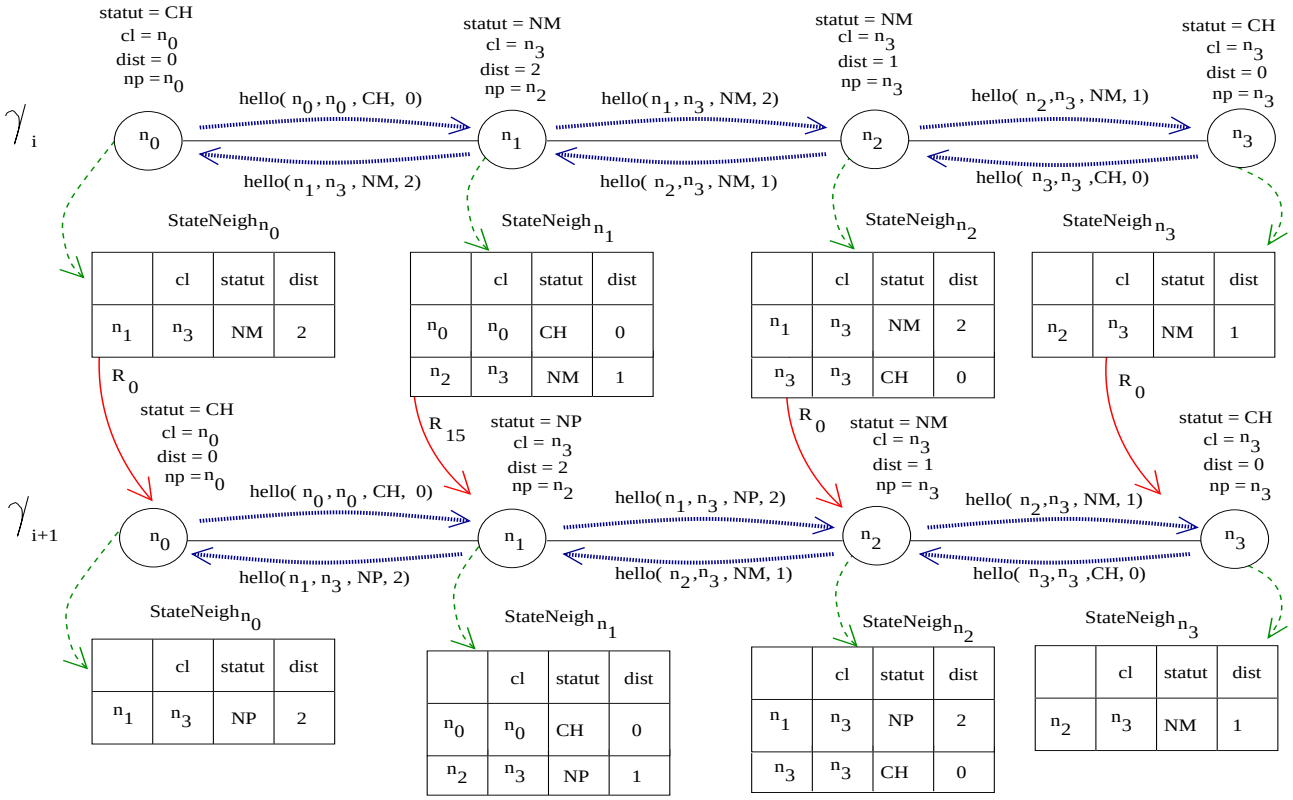


FIGURE 3.4 – Passage d'une configuration γ_i à γ_{i+1}

3.3.2 Description formelle

Après avoir présenté le principe d'exécution de SDEAC, nous passons maintenant sa description formelle. Chaque nœud u du réseau connaît le paramètre k (avec $k \leq n$ et n étant le nombre total de nœuds dans le réseau) qui représente le nombre maximal de sauts dans un *cluster*. De plus, tout nœud u possède les variables locales et *macros* qui sont définis dans le tableau 3.1. L'exécution de l'algorithme 1 (cf. Page 80) est déclenchée par la réception d'au moins un message venant d'au moins un voisin v ou par le déclenchement d'un « *timeout* ». Suite à cette exécution, chaque nœud u envoie un message à ses voisins pour les informer de son changement d'état.

Définition des constantes, variables et macros de SDEAC
Paramètres : $\|\mathcal{V}\| = n$: *nombre total de nœuds dans le réseau.* k : *nombre maximal de sauts dans un cluster.*
Variables locales du nœud u : id_u : *identifiant du nœud u.* N_u : *ensemble des nœuds voisins de u.* cl_u : *identifiant du cluster-head du nœud u.* $dist_{(u,CH_u)}$: *distance entre un nœud u et son cluster-head* CH_u. np_u : *identifiant du voisin du nœud u lui permettant d'atteindre son cluster-head.* $statut_u \in \{CH, NM, NP\}$: statut du nœud u. $StateNeigh_u$: *table de voisinage contenant l'ensemble des états des voisins de u.* $StateNeigh_u[v]$ *contient les états du nœud v voisin de u.*
Macros : $NeighCH_u = \{id_v \mid v \in N_u \wedge statut_v = CH \wedge cl_u = cl_v\}$. $NeighMax_u = (Max\{id_v \mid v \in N_u \wedge statut_v \neq CH \wedge cl_u = cl_v\}) \wedge (dist_{(v,CH_u)} = Min\{dist_{(x,CH_u)}, x \in N_u \wedge cl_x = cl_v\})$.

TABLE 3.1 – Constantes, variables et macros de SDEAC

3.4 Preuve formelle de SDEAC

Dans cette section, nous prouvons d'abord les propriétés de convergence et la clôture de SDEAC puis nous étudions son occupation mémoire requise.

3.4.1 Preuve des propriétés de convergence et de clôture

Nous étudions d'abord la propriété de convergence en montrant que partant d'une configuration quelconque et sans occurrence de pannes transitoires, SDEAC converge en un nombre fini d'étapes (temps de stabilisation) vers une configuration légale où les spécifications du problème sont respectées. Puis, nous montrons que toute exécution à partir d'une configuration légale et sans occurrence de pannes transitoires mènera toujours vers une configuration légale.

Algorithme 1: Algorithme de *clustering* auto-stabilisant à k *sauts*

/* A la réception d'un message *hello* ou au déclenchement d'un *timeout* */
/* Prédicats */
1 $P_1(u) \equiv (statut_u = CH)$
2 $P_2(u) \equiv (statut_u = NM)$
3 $P_3(u) \equiv (statut_u = NP)$
4 $P_{10}(u) \equiv (cl_u \neq id_u) \vee (dist_{(u,CH_u)} \neq 0) \vee (np_u \neq id_u)$
5 $P_{20}(u) \equiv (cl_u = id_u) \vee (dist_{(u,CH_u)} = 0) \vee (np_u = id_u)$
6 $P_{40}(u) \equiv \forall v \in N_u, (id_u > id_v) \wedge (id_u \geq cl_v) \wedge (dist_{(u,v)} \leq k)$
7 $P_{41}(u) \equiv \exists v \in N_u, (statut_v = CH) \wedge (cl_v > cl_u)$
8 $P_{42}(u) \equiv \exists v \in N_u, (cl_v > cl_u) \wedge (dist_{(v,CH_v)} < k)$
9 $P_{43}(u) \equiv \forall v \in N_u/(cl_v > cl_u), (dist_{(v,CH_v)} = k)$
10 $P_{44}(u) \equiv \exists v \in N_u, (cl_v \neq cl_u) \wedge \{(dist_{(u,CH_u)} = k) \vee (dist_{(v,CH_v)} = k)\}$
/* Règles */
/* Mise à jour de ta table de voisinage */
11 $StateNeigh_u[v] := (id_v, cl_v, statut_v, dist_{(v,CH_v)})$;
/* Cluster-1: Gestion de la cohérence */
12 $R_{10}(u) : P_1(u) \wedge P_{10}(u) \longrightarrow cl_u := id_u; np_u = id_u; dist_{(u,CH_u)} = 0;$
13 $R_{20}(u) : \{P_2(u) \vee P_3(u)\} \wedge P_{20}(u) \longrightarrow$
$statut_u := CH; cl_u := id_u; np_u = id_u; dist_{(u,CH_u)} = 0;$
/* Cluster-2: Clustering */
14 $R_{11}(u) : : \neg P_1(u) \wedge P_{40}(u) \longrightarrow statut_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; np_u := id_u;$
15 $R_{12}(u) : : \neg P_1(u) \wedge P_{41}(u) \longrightarrow$
$statut_u := NM; cl_u := id_v; dist_{(u,v)} := 1; np_u := NeighCH_u;$
16 $R_{13}(u) : : \neg P_1(u) \wedge P_{42}(u) \longrightarrow$
$statut_u := NM; cl_u := cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; np_u := NeighMax_u;$
17 $R_{14}(u) : : \neg P_1(u) \wedge P_{43}(u) \longrightarrow statut_u := CH; cl_u := id_v; dist_{(u,CH_u)} := 0; np_u := id_u;$
18 $R_{15}(u) : : P_2(u) \wedge P_{44}(u) \longrightarrow statut_u := NP;$
19 $R_{16}(u) : : P_1(u) \wedge P_{41}(u) \longrightarrow statut_u := NM; cl_v := id_v; dist_{(u,v)} := 1; np_u := NeighCH_u;$
20 $R_{17}(u) : : P_1(u) \wedge P_{42}(u) \longrightarrow$
$statut_u := NM; cl_u := cl_v; dist_{(u,CH_u)} := dist_{(v,CH_v)} + 1; np_u := NeighMax_u;$
/* Envoi d'un message *hello* */
21 $R_0(u) : hello(id_u, cl_u, statut_u, dist_{(u,CH_u)});$

Preuve de convergence

Pour montrer la convergence de notre algorithme, nous allons nous appuyer sur la notion de nœud *fixé* que nous définissons de la façon suivante.

Définition 3.6. ***(Nœud fixé)***
Un nœud u est dit fixé à partir d'une configuration γ_i si et seulement si $\forall j \geq i$, la valeur de sa variable cl_u à γ_j est égale à la valeur de sa variable cl_u à γ_i.

L'ensemble des nœuds fixés à la configuration γ_i est noté $\mathcal{F}_i$ (cf. Figure 3.5).

Définition 3.7. ***(Nœud non fixé)***
Un nœud est dit non fixé s'il ne respecte pas la définition 3.6 ci-dessus.

L'ensemble des nœuds non fixés à la configuration γ_i est noté par $\overline{\mathcal{F}_i}$ (cf. Figure 3.5).

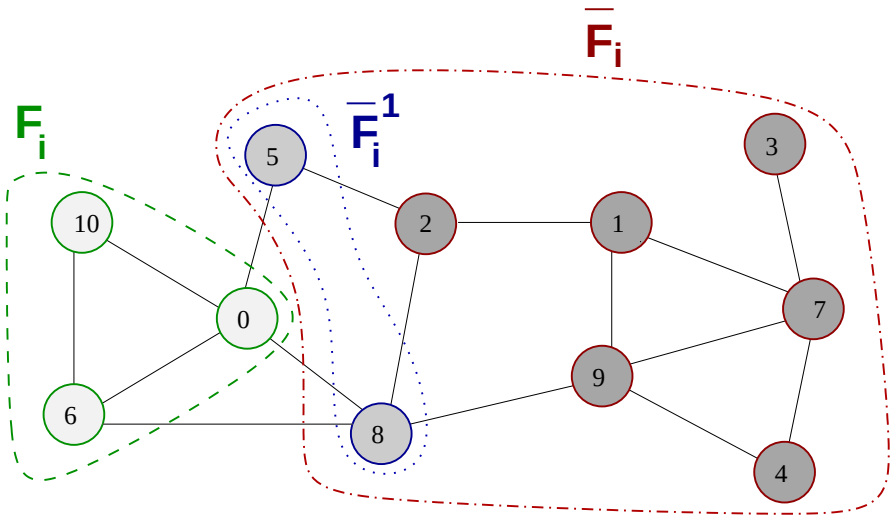


FIGURE 3.5 – Ensemble des nœuds fixés et non fixés à γ_i

Définition 3.8. ***(Nœud non fixé à distance 1)***
Un nœud non fixé à distance 1 à la configuration γ_i est un nœud non fixé situé à distance 1 des nœuds fixés à la configuration γ_i.

L'ensemble des nœuds non fixés à distance 1 à la configuration γ_i est noté $\overline{\mathcal{F}}_i^1$ (cf. Figure 3.5). Ainsi, cet ensemble est tel que $\forall u \in \overline{\mathcal{F}}_i^1 \implies u$ est non fixé et $dist_{(u,v)} = 1$ avec $v \in \mathcal{F}_i$.

Définition 3.9. ***(Cluster fixé)***
Un cluster est dit fixé à γ_i si et seulement si tous ses nœuds sont fixés. Un cluster fixé est aussi dit cluster stable.

En utilisant cette notion de nœud fixé, nous adoptons la démarche suivante pour établir la preuve de la propriété de convergence de SDEAC. Nous étudions la valeur de $|\mathcal{F}_i|$ et nous montrons que $|\mathcal{F}_i| = n$ si $\gamma_i = \delta$ avec δ étant une configuration légale et n le nombre de nœuds dans le réseau. En d'autres termes :

- nous prouvons que le nombre de nœuds fixés croit strictement à chaque transition $\gamma_i \mapsto \gamma_{i+1}$ et nous avons $|\mathcal{F}_i| = n$ si $\gamma_i = \delta$ c'est-à-dire que tous les nœuds du réseau sont fixés une fois la configuration légale atteinte.
- nous déterminons pour quelle valeur de i au plus nous avons $|\mathcal{F}_i| = n$ et $\gamma_i = \delta$ c'est-à-dire pour quelle valeur de i au plus, une configuration légale δ est atteinte depuis une configuration quelconque γ_0. La valeur de i représente alors le temps de stabilisation.

Lemme 3.1. *Soit γ_0 une configuration quelconque. A γ_1, $\forall u \in \mathcal{V}$, u est cohérent.*

Preuve.

Dès la réception de message *hello* d'un voisin, $\forall u \in \mathcal{V}$, u a exécuté l'algorithme 1 durant la transition $\gamma_0 \mapsto \gamma_1$. Deux cas sont possibles à l'exécution de l'étape *Cluster-1* de l'algorithme 1 (cf. Page 80) :

1. **Cas 1** : $statut_u = CH$. Donc, le prédicat P_1 est vrai. Ainsi :
   - si $(cl_u = id_u) \wedge (dist_{(u,CH_u)} = 0) \wedge (np_u = id_u)$, alors le prédicat P_{10} est faux et u est déjà cohérent. Donc, un nœud u est déjà cohérent et reste toujours cohérent à la fin de l'exécution de l'étape *Cluster-1*.
   - si $(cl_u \neq id_u) \vee (dist_{(u,CH_u)} \neq 0) \vee (np_u \neq id_u)$, alors le prédicat P_{10} est vrai et u n'est pas cohérent. Donc, la règle R_{10} est exécutée et le nœud u devient cohérent à la fin de l'exécution de l'étape *Cluster-1*.
2. **Cas 2** : $statut_u \in \{NM, NP\}$. Donc, le prédicat P_2 ou P_3 est vrai. Ainsi :
   - si $(cl_u \neq id_u) \wedge (dist_{(u,CH_u)} \neq 0) \wedge (np_u \neq id_u)$, alors le prédicat P_{20} est faux et u est déjà cohérent. Donc, un nœud u est déjà cohérent et reste toujours cohérent à la fin de l'exécution de l'étape *Cluster-1*.
   - si $(cl_u = id_u) \vee (dist_{(u,CH_u)} = 0) \vee (np_u = id_u)$, alors le prédicat P_{20} est vrai et u n'est pas cohérent. Donc, la règle R_{20} est exécutée et le nœud u devient cohérent à la fin de l'exécution de l'étape *Cluster-1*.

Donc, partant d'une configuration quelconque γ_0, à la configuration γ_1, $\forall u \in \mathcal{V}$, alors le nœud u est cohérent.

□

Corollaire 3.1. *À la configuration γ_1, un moins un nœud du réseau est fixé : $|\mathcal{F}_1| \geq 1$.*

Preuve.

- D'après le lemme 3.1, tous les nœuds sont cohérents à la fin de l'exécution de l'étape *Cluster-1* de l'algorithme 1 (cf. Page 80) durant $\gamma_0 \mapsto \gamma_1$.
- Or, $\exists\ u$ tel que $\forall\ v \in \mathcal{V}$, $id_u > id_v$. Alors, pour le nœud u, trois cas sont possibles durant l'exécution de l'étape *Cluster-2* de l'algorithme 1 (cf. Page 80) :

1. **Cas 1** : soit le nœud u tel que $statut_u = CH$. Donc, le prédicat P_1 est vrai.
   Comme P_1 est vrai, alors supposons que la règle R_{16} ou R_{17} s'exécute. Puisque $\forall\ v \in \mathcal{V}$, v est cohérent d'après le lemme 3.1 et de plus, $id_u > id_v$ donc les prédicats P_{41} et P_{42} sont faux. Ainsi, ni la règle R_{16}, ni la règle R_{17} est exécutée à la fin de l'étape *Cluster-2*. Seule la règle R_0 est exécutée par le nœud u.
2. **Cas 2** : soit le nœud u tel que $statut_u = NM$. Donc, le prédicat P_2 est vrai.
   Comme $statut_u = NM$, nous avons P_1 faux et seule R_{11}, R_{12} où R_{13} peut s'exécuter. Or, $\forall\ v \in \mathcal{V}$, v est cohérent d'après le lemme 3.1 et de plus, $id_u > id_v$. Donc, le prédicat P_{40} est vrai. Si $statut_v = CH$ alors $cl_v = id_v \Rightarrow cl_u > cl_v$. Ainsi, les prédicats P_{41} et P_{42} sont faux. D'où, seule le règle R_{11} est exécutée par le nœud u durant l'étape *Cluster-2*.
3. **Cas 3** : soit le nœud u tel que $statut_u = NP$. Donc, le prédicat P_3 est vrai.
   Comme $statut_u = NP$, nous avons le prédicat P_1 qui est faux. Donc, nous retrouvons le **Cas 2** ci-dessus.

Donc, à la configuration γ_1, ***au moins*** le nœud u tel que $\forall\ v \in \mathcal{V}$, $id_u > id_v$ est fixé et nous avons $(statut_u = CH) \wedge (cl_u = id_u) \wedge (dist_{(u,CH_u)} = 0) \wedge (np_u = id_u)$. D'où, $u \in \mathcal{F}_1$ et cela implique que $|\mathcal{F}_1| \geq 1$.

□

Définition 3.10. *Nous appelons CH_{Max_1} le nœud u tel que $\forall v \in \mathcal{V} \setminus \{u\}$, alors $id_u > id_v$. D'après le lemme 3.1 et le corolaire 3.1, à configuration γ_1, CH_{Max_1} est cohérent et son statut est CH (c.f Figure 3.6).*

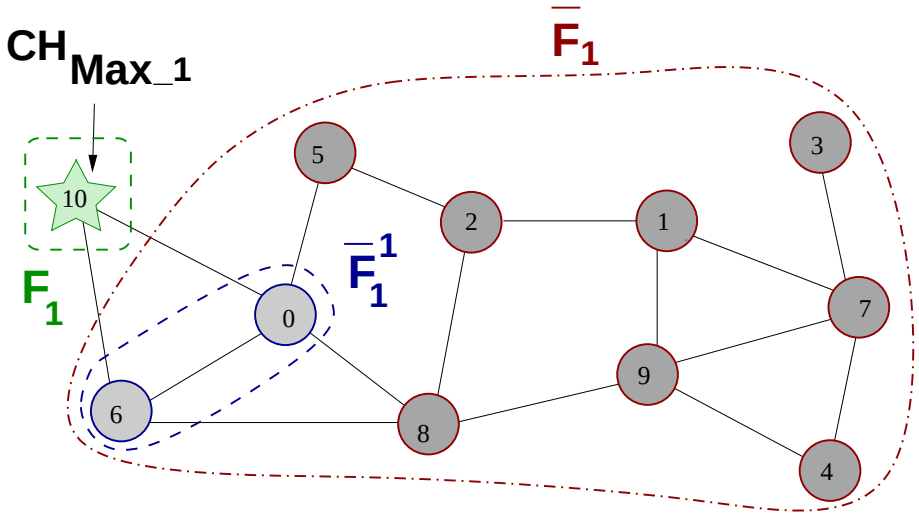


FIGURE 3.6 – Illustration du nœud CH_{Max_1}

Corollaire 3.2. *$\forall\ i > 1$, à la configuration γ_i le nœud u (CH_{Max_1}) tel qu'il est défini dans le corolaire 3.1 n'exécutera que la règle R_0.*

Preuve.

Le nœud u tel qu'il est défini dans le corolaire 3.1 est cohérent et $statut_u = CH$ à la configuration γ_1. Donc, $\forall\ i > 1$, à l'absence de l'occurrence de pannes transitoires, nous avons toujours le **cas 1** du corolaire 3.1. Seule la règle R_0 est exécutée par le nœud u à la configuration γ_i.

□

Théorème 3.1. *Jusqu'à k sauts, le nombre de nœuds fixés croit strictement à chaque transition : $\forall i < k + 1$, $|\mathcal{F}_{i+1}| > |\mathcal{F}_i|$.*

Preuve.

Pour démontrer ce théorème, nous adoptons une démarche par récurrence.

- D'après le corolaire 3.1, nous avons $|\mathcal{F}_1| \geq 1$. Donc, $|\mathcal{F}_1| > |\mathcal{F}_0|$ d'où l'hypothèse de récurrence est vraie au premier rang.
- Supposons l'hypothèse vraie $\forall i$, $i < k+1$ et prouvons que $|\mathcal{F}_{i+1}| > |\mathcal{F}_i|$:
  - ◇ Soit le nœud $w \in \mathcal{F}_i$ à la configuration γ_i, $\Rightarrow dist_{(w,CH_{Max\_1})} = i-1$,
  - ◇ Soit $v \in \overline{\mathcal{F}}_i^1$ à γ_i, $\Rightarrow dist_{(v,w)} = 1$.
  - ◇ D'après le lemme 3.1, tous les nœuds sont cohérents. De plus, durent $\gamma_i \mapsto \gamma_{i+1}$, tout nœud v a reçu au moins un message de w et a exécuté l'algorithme 1. Ainsi, trois cas sont possibles pour le nœud v :

    1. **Cas 1** : Soit $statut_v = CH$, alors le prédicat P_1 est vrai. Comme P_1 est vrai, seule la règle R_{16} ou R_{17} peut s'exécuter. Ainsi :
       - ∗ si $v \in N_{CH_{Max\_1}}$ avec CH_{Max_1} tel que définit dans le corollaire 3.1, alors le prédicat P_{41} est vrai. Donc, la règle R_{16} est exécutée durant la transition $\gamma_i \mapsto \gamma_{i+1}$ et nous avons $v \in \mathcal{F}_{i+1}$ à la configuration γ_{i+1}.
       - ∗ si $v \notin N_{CH_{Max\_1}}$ avec CH_{Max_1} tel que définit dans le corolaire 3.1, alors le prédicat P_{41} est faux et la règle R_{16} n'est pas exécutée. De plus, comme nous avons $(dist_{(w,CH_{Max\_1})} = i-1)$ et $(cl_w = id_{CH_{Max\_1}}) \Rightarrow (dist_{(w,CH_{Max\_1})} < k)$ et $(cl_w > cl_v)$. Donc, le prédicat P_{42} est vrai. Ainsi, la règle R_{17} est exécutée durant la transition $\gamma_i \mapsto \gamma_{i+1}$ et nous avons $v \in \mathcal{F}_{i+1}$ à la configuration γ_{i+1}.
    2. **Cas 2** : Soit $statut_v = NM$, alors le prédicat P_2 est vrai $\Rightarrow$ P_1 est faux. Comme $v \neq CH_{Max\_1}$, alors le prédicat P_{40} est faux et la règle R_{11} n'est pas exécutée. Seule la règle R_{12} ou R_{13} peut être exécuter :
       - ∗ si $v \in N_{CH_{Max\_1}}$ avec CH_{Max_1} est tel que définit dans le corolaire 3.1, alors le prédicat P_{41} est vrai et la règle R_{12} est exécutée durant la transition $\gamma_i \mapsto \gamma_{i+1}$ et $v \in \mathcal{F}_{i+1}$ à la configuration γ_{i+1}.
       - ∗ si $v \notin N_{CH_{Max\_1}}$ avec CH_{Max_1} est tel que définit dans le corolaire 3.1, alors le prédicat P_{41} est faux et la règle R_{12} n'est pas exécutée. De plus, comme nous avons $(dist_{(w,CH_{Max\_1})} = i-1)$ et $(cl_w = id_{CH_{Max\_1}}) \Rightarrow (dist_{(w,CH_{Max\_1})} < k)$ et $(cl_w > cl_v)$. Donc, le prédicat P_{42} est vrai. Ainsi, la règle R_{13} est exécutée durant la transition $\gamma_i \mapsto \gamma_{i+1}$ et nous avons $v \in \mathcal{F}_{i+1}$ à la configuration γ_{i+1}.
    3. **Cas 3 :** Soit $statut_v = NP$, alors le prédicat P_3 est vrai :
       - ∗ Comme $statut_v = NP$, le prédicat P_1 est faux, nous retrouvons alors le **Cas 2**.

Donc, $\forall\ v \in \overline{\mathcal{F}}_i^1$ à la configuration γ_i, à la configuration γ_{i+1} nous avons $v \in \mathcal{F}_{i+1}$ (c'est le nœud v est fixé). De plus, $(statut_v = NM) \wedge (cl_v = id_{CH_{Max\_1}}) \wedge (dist_{(v,CH_{Max\_1})} = i) \wedge (np_v = NeighMax_v)$. D'où, $\forall i < k+1$, $|\mathcal{F}_{i+1}| > |\mathcal{F}_i|$.

□

Lemme 3.2. *À la configuration γ_{k+1}, le premier cluster est fixé et ne change plus en l'absence de pannes transitoires.*

Preuve.

D'après le théorème 3.1, nous avons $|\mathcal{F}_k| > |\mathcal{F}_{k-1}|$.

- Or, nous savons que $\exists\ w \in \mathcal{F}_k$ à la configuration $\gamma_k \Rightarrow dist_{(w,CH_{Max\_1})} = k-1$.

- Soit $v \in N_w$ et $v \in \overline{\mathcal{F}}_k^1$ à $\gamma_k \Rightarrow dist_{(v,w)} = 1$.

Durant la transition $\gamma_k \mapsto \gamma_{k+1}$, tout nœud v a reçu un message d'un nœud w et a exécuté l'algorithme 1. Trois cas sont possibles pour le nœud v :

1. **Cas 1** : $statut_v = CH$, nous retrouvons le cas 1 du théorème 3.1 ;
2. **Cas 2** : $statut_v = NM$, nous retrouvons le cas 2 du théorème 3.1 ;
3. **Cas 3** : $statut_v = NP$, nous retrouvons le cas 3 du théorème 3.1.

Ainsi, $\forall v \in \overline{\mathcal{F}}_k^1$ à la configuration γ_k, à la configuration γ_{k+1} nous avons $v \in \mathcal{F}_{k+1}$ et $dist_{(v,CH_{Max\_1})} = k$. Donc, $|\mathcal{F}_{k+1}| > |\mathcal{F}_k|$. D'où, le premier *cluster* est fixé (c'est-à-dire tous le nœuds du premiers *cluster* son fixés).

- Soit le nœud $x \in \overline{\mathcal{F}}_{k+1}^1$ tel que $x \in N_v \backslash \{w\}$ et que le nœud $w \in \mathcal{F}_k$ depuis la configuration γ_k. Donc, nous avons $dist_{(x,v)} = 1$.

Durant la transition $\gamma_{k+1} \mapsto \gamma_{k+2}$, le nœud x a reçu un message de v et a exécuté l'algorithme 1 et vice-versa. Deux cas sont possibles :

1. **Cas 1** : v a reçu un message de x. Comme $cl_v = id_{CH_{Max\_1}} > cl_x$ et $dist_{(v,CH_{Max\_1})} = k$ alors quel que soit l'état du nœud x, le nœud v reste toujours dans $V_{CH_{Max\_1}}$. De plus, $(cl_v \neq cl_x) \Rightarrow P_{44}$ est vrai et v exécute la règle R_{15}. D'où, le statut du v devient NP.
2. **Cas 2** : x a reçu un message de v. Comme $dist_{(v,CH_{Max\_1})} = k$, le nœud x ne peut pas se rattacher au c*luster* $V_{CH_{Max\_1}}$ même si $(cl_v > cl_x)$.

Donc, à la configuration γ_{k+1}, le premier *cluster* est fixé et ne change plus en l'absence de fautes transitoires.

□

L'ensemble des nœuds non fixés à la configuration γ_{k+1} est $\overline{\mathcal{F}}_{k+1} = \mathcal{V} - \mathcal{F}_{k+1}$. Dans l'exemple la figure 3.7, où $k = 2$, nous montrons l'ensemble des nœuds non fixés à la configuration γ_3 et ceux déjà fixés à cette même configuration. La figure 3.7 illustre aussi un exemple du CH_{Max_1} et du CH_{Max_2} que nous définissons de la façon suivante.

Définition 3.11. *Nous définissons CH_{Max_2} le nœud u tel que $u \in \overline{\mathcal{F}}_{k+1}$ et $\forall v \in \overline{\mathcal{F}}_{k+1} \setminus \{u\}$, alors $id_u > id_v$ (c.f Figure 3.7)*

Théorème 3.2. ***Généralisation du théorème** 3.1*

À chaque transition, le nombre de nœuds fixés croit strictement : $\forall i < n$, $|\mathcal{F}_{i+1}| > |\mathcal{F}_i|$.

Preuve.

Nous faisons une preuve par récurrence.

- D'après le lemme 3.2, $V_{CH_{Max\_1}}$ est fixé à la configuration $\gamma_{k+1} \Rightarrow \forall v \in \mathcal{F}_{k+1}$, $(cl_v = id_{CH_{Max\_1}}) \wedge (dist_{(v,CH_{Max\_1})} \leq k)$. De plus, $\forall w \in N_{CH_{Max\_2}}$ tel que $w \in \mathcal{F}_{k+1}$ et CH_{Max_2} décrit dans la définition 3.11, nous avons $(cl_w > id_{CH_{Max\_2}}) \wedge (dist_{(w,CH_{Max\_1})} = k) \Rightarrow P_{43}$ est vrai. Donc, durant la transition $\gamma_{k+1} \mapsto \gamma_{k+2}$ le nœud CH_{Max_2} exécute la règle R_{14} et devient *cluster-head* $\Rightarrow |\mathcal{F}_{k+2}| > |\mathcal{F}_{k+1}|$. D'où, l'hypothèse de récurrence est vraie au premier rang.

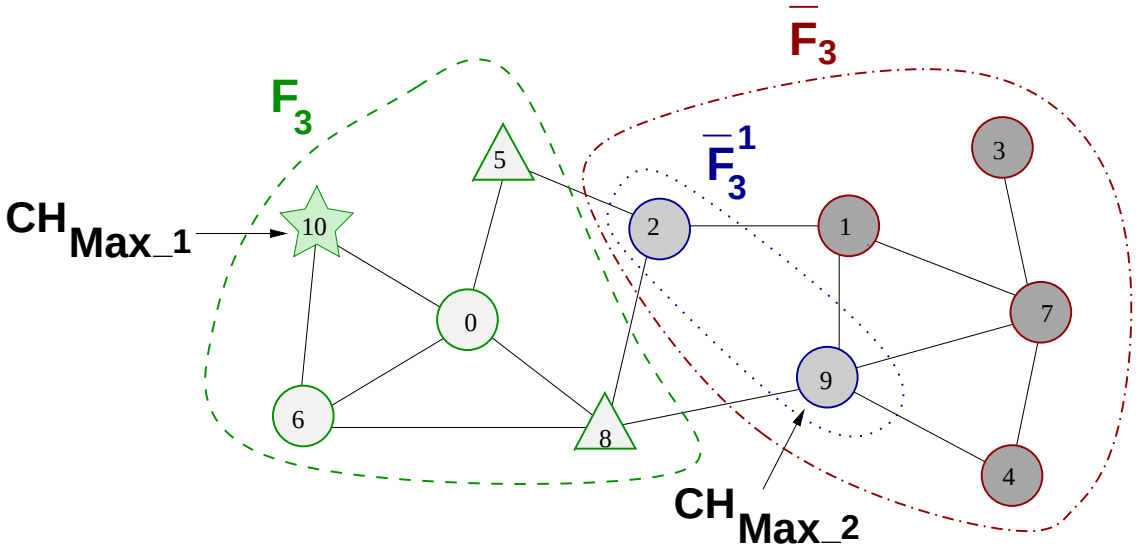


FIGURE 3.7 – CH_{Max_1} et CH_{Max_2} dans un *clustering* à 2 sauts

- Supposons l'hypothèse de récurrence vraie $\forall i < n$ et prouvons que $|\mathcal{F}_{i+1}| > |\mathcal{F}_i|$:
  - $\forall i < n$, durant la transition $\gamma_i \mapsto \gamma_{i+1}$ pour chaque $u \in \overline{\mathcal{F}}_i^1$ trois cas sont possibles :
    1. **Cas 1** : $statut_u = CH$: nous retrouvons le cas 1 du théorème 3.1 ;
    2. **Cas 2** : $statut_u = NM$: nous retrouvons le cas 2 du théorème 3.1 ;
    3. **Cas 3** : $statut_u = NP$: nous retrouvons le cas 3 du théorème 3.1.

Donc, $\forall i < n$, $|\mathcal{F}_{i+1}| > |\mathcal{F}_i|$.

□

Théorème 3.3. ***(Convergence)***
Partant d'une configuration quelconque γ_0, sans occurrence de fautes transitoires, une configuration légale δ est atteinte au plus en $n+2$ transitions.

Preuve.

- D'après le lemme 3.1, partant d'une configuration quelconque γ_0, durant $\gamma_0 \mapsto \gamma_1$ tous les nœuds sont cohérents à γ_1 ;
- D'après le corolaire 3.1, nous avons $|\mathcal{F}_1| > |\mathcal{F}_0| \geq 1$;
- D'après le théorème 3.2, nous avons $\forall i < n$, $|\mathcal{F}_{i+1}| > |\mathcal{F}_i|$;
- Pour $i = n$, nous avons $|\mathcal{F}_{n+1}| > |\mathcal{F}_n| \geq n$. Comme $|\mathcal{V}| = n \Rightarrow |\mathcal{F}_{n+1}| = n$;
- Il faut ensuite une transition supplémentaire pour que les mises à jour s'effectuent et que les états des nœuds ne changent plus en l'absence de pannes transitoires.

Par conséquence, au plus en $n+2$ transitions, une configuration légale δ est atteinte en partant d'une configuration quelconque γ_0. Nous avons $|\mathcal{F}_{n+2}| = n$ et seule la règle R_0 est exécutée par les nœuds.

□

Preuve de la clôture

Dans cette section, nous montrons la propriété de clôture de SDEAC. En d'autres termes, depuis une configuration légale et sans occurrence de pannes transitoires, toute exécution conduit toujours à nouvelle configuration légale.

Théorème 3.4. ***(Clôture)***
A partir d'une configuration légale γ_i et sans occurrence de pannes transitoires, chaque nœud reste dans une configuration légale à γ_{i+1}.

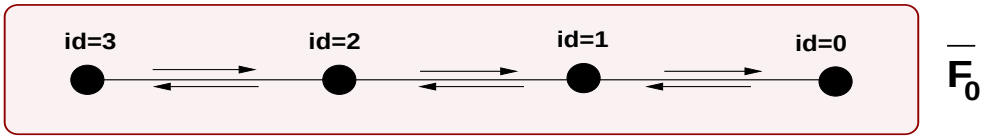
Preuve.
- D'après le théorème 3.3, tous les nœuds sont stables au plus en $n + 2$ transitions ;
- Soit γ_i une configuration légale. $\forall\ u \in \mathcal{V}$, le nœud u est fixé à la configuration γ_i et seule la règle R_0 est exécutée ;
- Donc, $\forall j > i$, à l'absence de pannes transitoires, nous avons toujours une configuration légale à γ_j ;
- L'état des nœuds ne change plus entre γ_j et γ_{j+1} $\forall j$ sans occurrence de pannes transitoires seule la règle R_0 est exécutée.

□

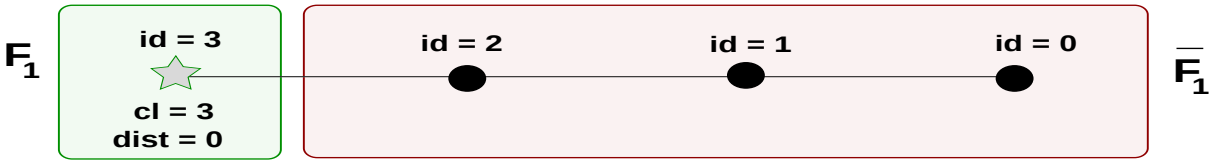
3.4.2 Temps de stabilisation : le cas de la chaîne ordonnée

Nous venons de montrer que partant d'une configuration quelconque γ_0 et sans occurrence de fautes transitoires, une configuration légale δ est atteinte au plus en $n+2$ transitions. Ce temps de stabilisation représente le « pire des cas » et correspond à une topologie bien particulière qui est la chaîne ordonnée. En effet, comme illustré dans l'exemple de la figure 3.8, dans une topologie en chaîne ordonnée, les nœuds se fixent de la plus grande à la plus petite identité.

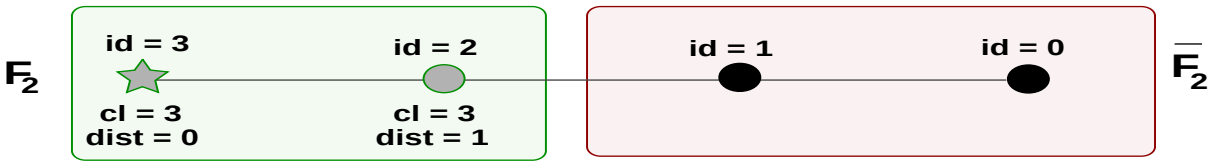
Cependant, étant donné le caractère distribué et asynchrone de notre algorithme, dans le cas de topologies de réseaux quelconques, les nœuds possédant les identités les plus grandes se fixent en premier puis initient la construction de leurs *clusters*. Ainsi, la construction des différents *clusters*, comme illustré dans l'exemple de la figure 3.9, s'effectue en parallèle et nous observons des temps de stabilisation très inférieurs à celui du pire des cas ($n + 2$ transitions). Dans le cas de topologies quelconques, nous évaluons donc le temps stabilisation par le biais de simulation.



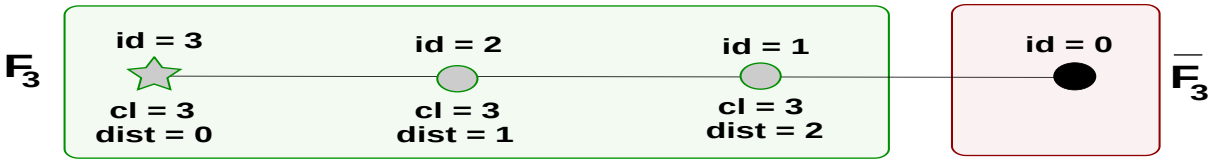
(a) Configuration γ_0 de départ : aucun nœud de la chaîne ordonnée n'est fixé.



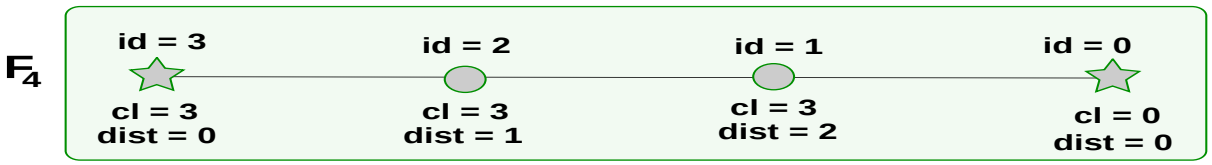
(b) Configuration γ_1 : le nœud d'identité maximale ($id = 3$) de la chaîne ordonnée se fixe.



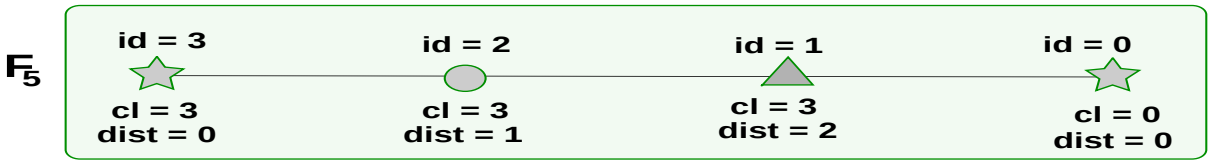
(c) Configuration γ_2 : le nœud à distance 1 se fixe au nœud d'identité maximale de la chaîne.



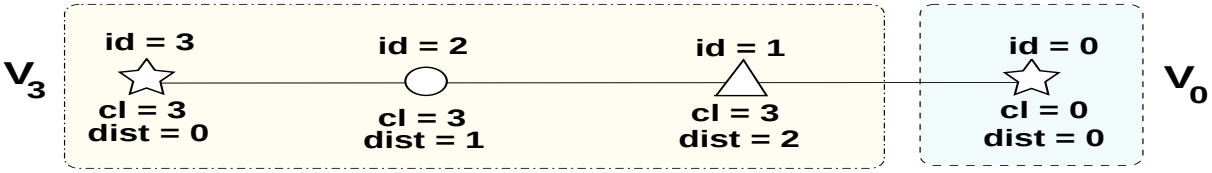
(d) Configuration γ_3 : le nœud à distance 2 se fixe au nœud d'identité maximale de la chaîne.



(e) Configuration γ_4 : comme $k = 2$, le nœud à distance 3 du nœud maximal devient CH.

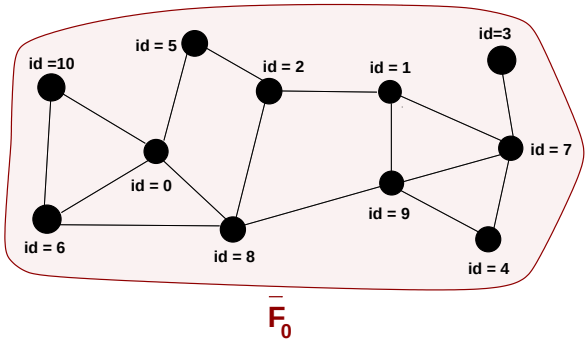


(f) Configuration γ_5 : le nœud 1, bien que déjà fixé au nœud 3, devient nœud passerelle.

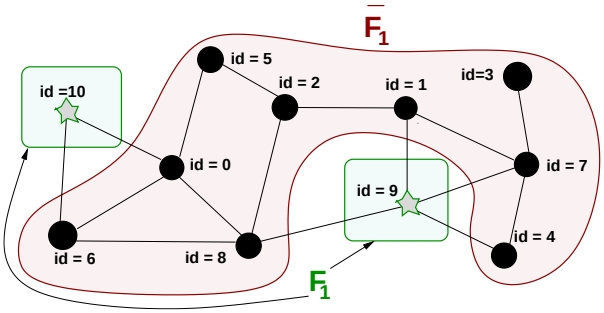


(g) Configuration γ_6 : $\gamma_5 \mapsto \gamma_6$ représente la transition supplémentaire nécessaire aux nœuds 0 et 1 pour les mises à jour. γ_6 est la configuration finale.

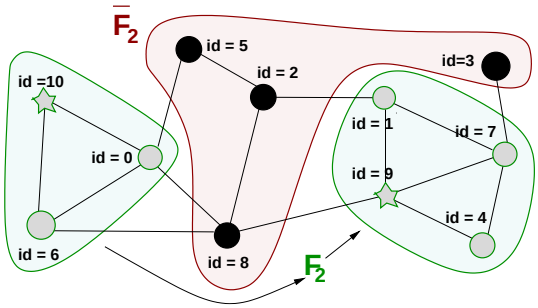
FIGURE 3.8 – Pire des cas : stabilisation dans une chaîne ordonnée



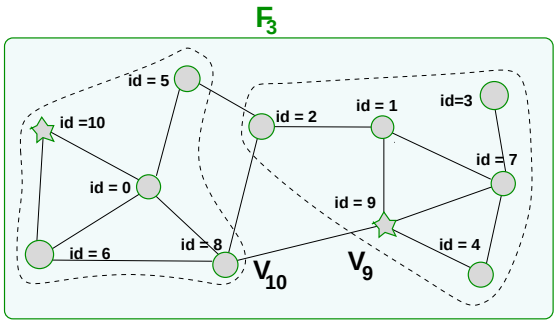
(a) Configuration γ_0 : aucun nœud du réseau n'est fixé.



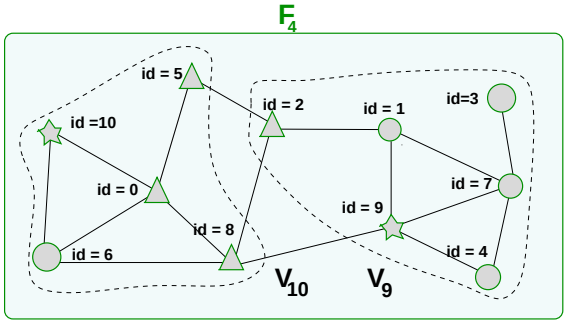
(b) Configuration γ_1 : les nœuds les plus grands du réseau (10 et 9) se fixent.



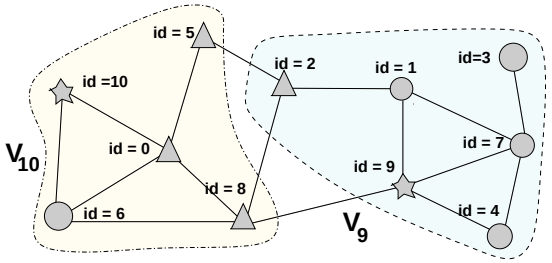
(c) Configuration γ_2 : les nœuds à distance 1 des nœuds les plus grands du réseau se fixent.



(d) Configuration γ_3 : les nœuds à distance 2 des nœuds les plus grands du réseau se fixent.



(e) Configuration γ_4 : comme $k = 2$, tous les nœuds sont déjà fixés. Seuls les statuts des passerelles sont déterminés.



(f) Configuration γ_5 : les dernières mises à jour s'effectuent et les *clusters* sont stables.

FIGURE 3.9 – Stabilisation dans un graphe quelconque

3.4.3 Occupation mémoire

Dans cette section, nous montrons l'occupation mémoire maximale nécessaire pour l'exécution de SDEAC.

Lemme 3.3. *L'espace mémoire nécessaire pour chaque voisin est de* $\log(2n + k + 3)$ *bits.*

Preuve.

Pour fonctionner, chaque nœud u du réseau a besoin de connaître pour chacun de ses voisins $v \in N_u$ son l'identifiant (id_v), son identifiant de *cluster* (cl_v), son statut ($statut_v$) et sa distance $dist_{(v,CH_v)}$. Ainsi, pour un réseau de n nœuds, nous avons au plus :

- n valeurs possibles pour l'identifiant du nœud ;
- n valeurs possibles pour l'identifiant de *cluster* ;
- k valeurs possibles pour la distance ;
- trois (3) valeurs de statut différentes que sont CH, NM et NP.

Or, il est nécessaire d'avoir $\log(n)$ bits pour encoder une variable avec n valeurs possibles. Donc, pour chaque nœud voisin, l'algorithme a besoin d'au plus de $\log(2n + k + 3)$ bits.

□

Corollaire 3.3. *Chaque nœud u du réseau a besoin de* $(\Delta_u + 1) \times \log(2n + k + 3)$ *bits.*

Preuve.

D'après le lemme 3.3, SDEAC requiert pour chaque voisin v d'un nœud u un espace mémoire de $\log(2n + k + 3)$ bits. Soit N_u le voisinage du nœud u. Posons $\Delta_u = |N_u|$ comme étant le degré du nœud u c'est-à-dire le nombre total de nœuds dans le voisinage de u. Pour l'exécution de SDEAC, chaque nœud u a besoin de connaitre l'état de tous ses Δ_u voisins. De plus, u mémorise aussi les informations qui définissent ses propres états.

Donc, chaque nœud u du réseau a besoin de $(\Delta_u + 1) \times \log(2n + k + 3)$ bits.

□

Corollaire 3.4. *Chaque nœud u du réseau a besoin d'au plus* $n \times \log(2n + k + 3)$ *bits.*

Preuve.

Pour un réseau de n nœuds, le voisinage N_u de chaque nœud u peut contenir au maximum $n - 1$ voisins (pour un graphe complet). Dans ce cas précis, pour fonctionner, SDEAC a besoin de $n \times \log(2n + k + 3)$ bits par nœud.

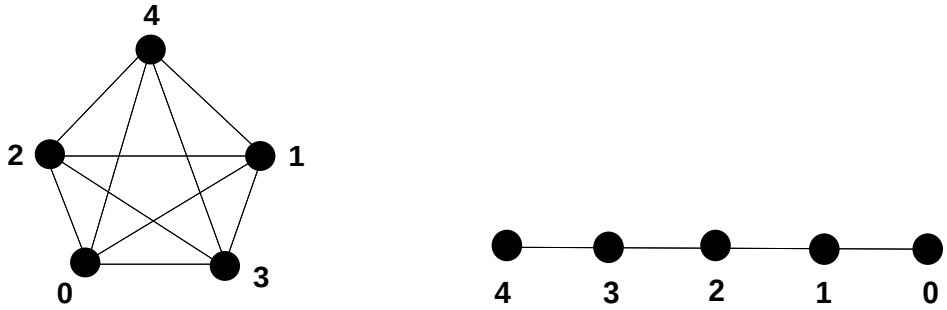
□

3.4.4 Temps de stabilisation vs Occupation mémoire

Nous venons de montrer avec le corollaire 3.4 l'occupation mémoire la plus importante dans le cas d'un graphe complet où chaque nœud a besoin de $n \times \log(2n + k + 3)$ bits. Or, comme dans un graphe complet tous les nœuds sont voisins (c.f Figure 3.10(a)), nous pouvons aisément remarquer qu'en 2 transitions tous les nœuds se fixent. En effet, dans un graphe complet, partant d'une configuration quelconque γ_0, à la configuration γ_1, le nœud possédant la plus grande identité se fixe en premier. Puis, à γ_2 tous les autres nœuds se fixent au nœud

d'identité maximale, et nous obtenons un seul *cluster*. Dans ce cas, l'occupation mémoire est maximale et le temps de stabilisation est minimal.

Cependant, comme nous l'avons soulevé dans la Section 3.4.2, pour l'autre critère déterminant de SDEAC, à savoir le temps de stabilisation, nous observons le pire des cas dans une topologie en chaîne ordonnée. Or, comme illustré dans l'exemple de la chaîne ordonnée de la figure 3.10, les nœuds aux deux extrémités de la chaîne ont un (1) seul voisin alors que ceux à l'intérieur de la chaîne n'ont que deux (2) voisins. Dans ce cas, l'occupation mémoire est minimale et le temps de stabilisation est maximal.



(a) Graphe complet (b) Chaîne ordonnée

FIGURE 3.10 – Graphe complet vs Chaîne Ordonnée

En résumé, comme illustré dans le tableau 3.2, dans le cas où nous observons un temps de stabilisation au maximal de $n+2$ transitions pour une chaîne ordonnée de n nœuds, l'occupation mémoire est la moindre et est de $3 \times \log(2n + k + 3)$ bits pour les nœuds à l'intérieur de la chaîne ordonnée (pour les deux nœuds situés aux deux extrémités, l'occupation mémoire est $2 \times \log(2n + k + 3)$ bits). Inversement, dans le cas ou l'occupation mémoire est maximale de $n \times \log(2n + k + 3)$ bits pour un graphe complet, le temps de stabilisation est au plus bas et est de 2 transitions.

Pour des topologies quelconques qui ne sont ni une chaîne ordonnée ni un graphe complet, nous avons recours à une campagne de simulation pour évaluer les performances de notre solution.

	Temps de stabilisation	**Occupation mémoire**
Chaîne ordonnée	$n + 2$ transitions	$3 \times \log(2n + k + 3)$ bits
Graphe complet	2 transitions	$n \times \log(2n + k + 3)$ bits

TABLE 3.2 – Graphe complet vs Chaîne Ordonnée : temps de stabilisation et occupations mémoires

3.4.5 Comparaison analytique

Dans les Sections 3.4.1 et 3.4.3, nous avons respectivement montré que le temps de stabilisation de SDEAC est au plus de $n+2$ transitions et l'espace mémoire nécessaire pour chaque voisin est $\log(2n+k+3)$ bits. Le tableau 3.3 illustre une comparaison du temps de stabilisation, de l'espace mémoire par voisin et du voisinage de notre algorithme avec ceux des meilleures solutions de *clustering* auto-stabilisantes fondées soit sur un modèle à états [DDL09, CDDL10, DLV10] soit sur un modèle synchrone à passage de messages [LT11] et construisant des *clusters* à k sauts. Précisons que nous donnons les comparaisons en faisant abstraction du modèle communication et que la transformation d'un modèle de communication à un autre reste possible bien que cela entraine des sur-coûts supplémentaires [GM91, KP93, DIM97].

	Solutions	Voisinage	Temps de stabilisation	Occupation mémoire
***Clusters* de diamètre $2k$**	**SDEAC**	1 saut	$n+2$	$\log(2n+k+3)$
	MINIMAL [DDL09]	k sauts	$O(n)$; $O(n^2)$	$O(\log(n))$
	FLOOD [DLV10]	k sauts	$O(k)$	$O(k \times \log(n))$
	Larsson et Tsigas [LT11]	k sauts	$O(k)$; $O(g \times k \times \log(n))$	$O(\|G_u^k\| \times (\log(n) + \log(k)))$
	K-CLUSTERING [CDDL10]	k+1 sauts	$O(n \times k)$	$O(log(n) + log(k))$

TABLE 3.3 – Performances des solutions auto-stabilisantes dans les pires cas

Dans un premier temps, remarquons que parmi les approches résumées dans le tableau 3.3, la solution proposée dans [DLV10] présente le temps de stabilisation le plus bas. En effet, les auteurs reprennent, dans un modèle à états moins contraignant, le principe de l'algorithme Max-Min proposé par Amis et *al.* [APVH00]. Donc, FLOOD présente la même complexité en temps que Max-Min. Hormis FLOOD, SDEAC présente un temps de stabilisation plus bas que [DDL09, CDDL10, LT11].

Dans un deuxième temps, nous pouvons noter que le temps de stabilisation de SDEAC n'est pas fonction du paramètre k contrairement aux solutions proposées dans [CDDL10, DLV10, LT11]. De plus, étant donné que notre approche combine la découverte de voisinage et la procédure de *clustering* en une seule phase, elle présente un unique temps de stabilisation contrairement à la solution de Datta *et al.* [DDL09] et de Larsson and Tsigas [LT11]. Dans ces deux solutions [DDL09, LT11], les auteurs commencent d'abord par construire l'ensemble de *cluster-heads*. Puis, partant de cet ensemble de *cluster-heads*, ils structurent le réseaux en *clusters* de rayon au plus k.

Dans un troisième temps, nous pouvons aussi remarquer que SDEAC considère un voisinage à distance 1 pour construire des *clusters* à k sauts. Alors que les algorithmes proposés dans [DDL09, CDDL10, DLV10, LT11] nécessitent un voisinage à distance k ou $k+1$. Plus précisément, pour les solutions sur un modèle à états présentées dans [DDL09, DLV10, CDDL10], les nœuds peuvent lire les états de leurs voisins sur une distance de k ou $k+1$

sauts. Quant à la solution de Larsson et Tsigas [LT11], les nœuds ont une vision du voisinage à distance de k sauts. Devant cette situation, les messages de chaque nœuds sont retransmis jusqu'à k sauts.

Enfin, dans un quatrième temps, s'agissant de l'occupation mémoire, notons que SDEAC requiert moins de mémoire que la solution de Larsson et Tsigas [LT11] qui utilise un modèle de communication assez semblable c'est-à-dire un modèle synchrone à passage de messages. En effet, dans la solution de Larsson et Tsigas [LT11], comme les messages sont retransmis jusqu'à une distance k, alors chaque nœud stocke les informations sur tout son voisinage à distance k. Alors dans SDEAC, comme les échanges de messages s'effectuent uniquement entre voisins de distance 1, un nœud ne mémorise que les informations sur ses voisins à distance 1.

3.5 Comportement de SDEAC face aux pannes transitoires

Dans cette section, nous étudions par observation le comportement de SDEAC en présence de pannes transitoires causées par une modification topologique (apparition ou disparition de nœuds dans le réseau).

3.5.1 Disparition de nœuds dans le réseau

En cas de disparition d'un nœud u, comme illustré dans la figure 3.11, deux cas sont possibles : soit le nœud est un *cluster-head* ($statut_u = CH$) soit le nœud u est nœud membre ou nœud passerelle ($statut_u \in \{NM, NP\}$).

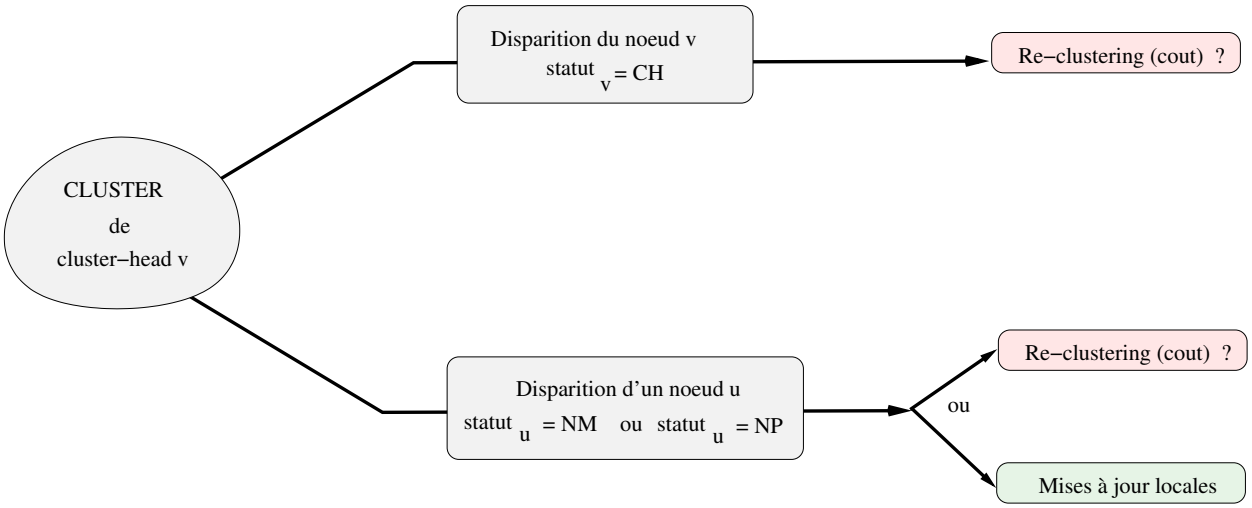
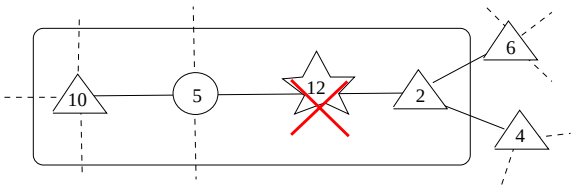


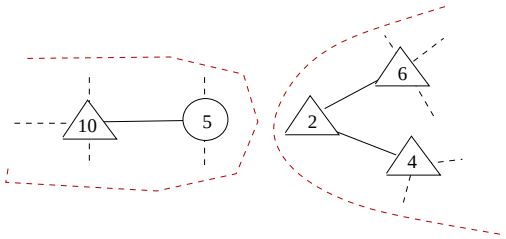
FIGURE 3.11 – Disparition d'un nœud du réseau au sein d'un *cluster*

Disparition d'un *cluster-head*

Si le nœud qui disparait du *cluster* est de statut CH, alors le *cluster* est détruit et une procédure de *re-clustering* est déclenchée. Le coût de cette procédure de *re-clustering* dépend de la topologie du réseau et de la répartition des identifiants des nœuds. Donc, dans le pire des cas cela peut entrainer une reconstruction totale du réseau (cas d'une chaîne ordonnée par exemple avec la disparition du nœud d'identifiant maximal). Comme nous venons de le montrer, une construction de *clusters* dans une topologie en chaine ordonnée se fait au plus en $n + 2$ transitions. Pour des topologies réseaux quelconques, comme illustré dans la figure 3.12, le coût de cette reconstruction ne peut pas être connu d'avance.



(a) Dans cet exemple, le *cluster-head* 12 disparait du *cluster*. Le *cluster* est détruit et une reconstruction est déclenchée

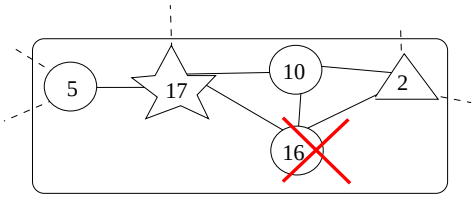


(b) Cette reconstruction dépend de la répartition des identifiants de nœuds. Comme dans SDEAC les nœuds n'ont qu'une vision à distance 1 du voisinage, le coût de cette reconstruction ne peut pas être connu à l'avance.

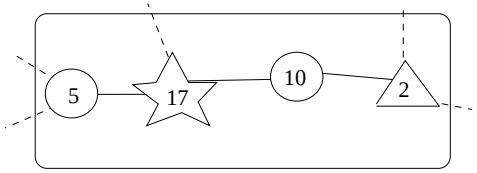
FIGURE 3.12 – Disparition d'un *cluster-head* ($k = 2$)

Disparition de NM ou NP

Si le nœud disparu est de statut NM ou NP, alors au mieux nous avons uniquement des mises à jour locales (cf. figure 3.12,).



(a) Dans cet exemple, le 16 de statut NM disparait du réseau.

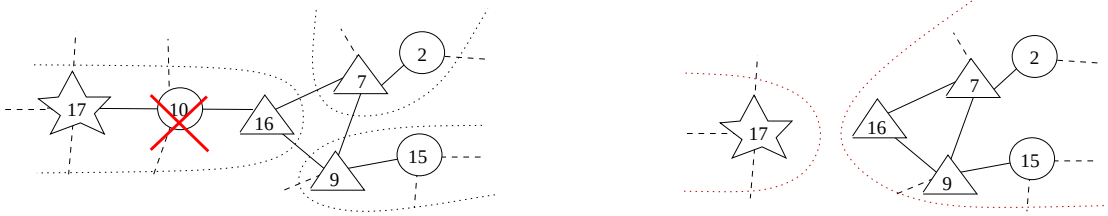


(b) Aucune modification ne s'opère dans le *cluster* mais uniquement des mises à jours locales.

FIGURE 3.13 – Disparition de NM ou NP : mises à jour locales

Cependant, selon la topologie du réseau et la répartition des identifiants, la disparition d'un nœud de statut NM ou NP peut entrainer dans le pire des cas une reconstruction totale.

La figure 3.14 illustre un exemple de disparition d'un nœud de statut NP où le coût de la reconstruction ne peut pas être connu d'avance.



(a) Disparition du nœud du nœud 10 de statut NM.

(b) En fonction de l'identité des nœuds 2, 7, 9 et 15, la reconstruction peut ne pas être locale.

FIGURE 3.14 – Disparition de NM ou NP : reconstruction non locale

3.5.2 Apparition de nœud dans le réseau

En cas d'apparition d'un nœud u dans le réseau, comme illustré dans la figure 3.15, selon sa distance d'un *cluster-head* v, deux cas sont possibles : soit le nœud u arrive à une distance inférieure à k sauts de v ($dist_{(u,v)} < k$) soit le nœud u arrive à distance $k+1$ saut de v ($dist_{(u,v)} = k+1$). Dans les deux cas, selon que l'identifiant de u est supérieur celui de v ou non, deux cas sont également possibles.

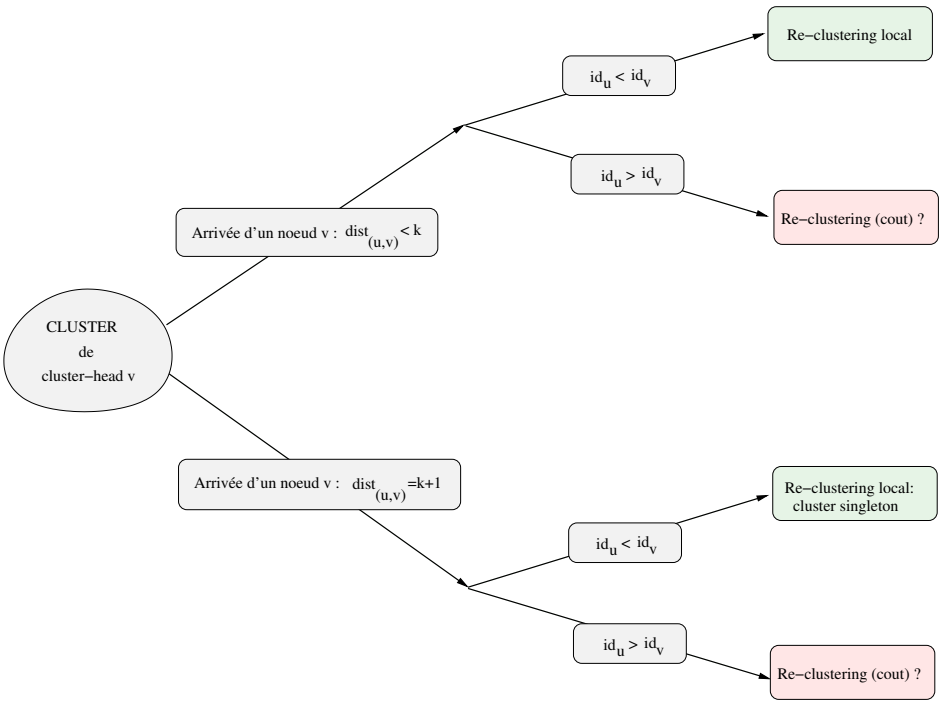


FIGURE 3.15 – Apparition d'un nœud dans le réseau au sein d'un *cluster*

Arrivée d'un nœud à distance inférieure à k d'un *cluster-head*

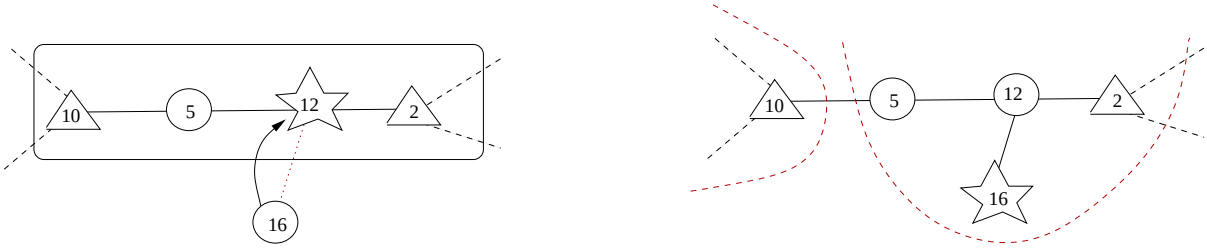
Lorsque que le nœud u arrive à une distance inférieure à k de v et possède un plus petit identifiant $\{(dist_{(u,v)} < k) \wedge (id_u < id_v)\}$, alors seule une reconstruction locale est effectuée (cf. Figure 3.16). Le nœud u rejoint tout simplement au *cluster* de v.

(a) Le nœud 7 arrive à 2 sauts du *cluster-head* 12.

(b) Le nœud 7 rejoint le *cluster* du *cluster-head* 12 comme ce dernier possède un identifiant plus grand et est à moins de k sauts.

FIGURE 3.16 – Arrivée d'un nœud à une distance inférieure à k sauts et d'identifiant plus petit que celui du *cluster-head* : mise à jour locale ($k = 2$)

Par contre, lorsque le nœud u arrive à une distance inférieure à k de v et possède un plus grand identifiant $((dist_{(u,v)} < k) \wedge (id_u > id_v))$, alors une reconstruction du *cluster* est entamée. Au pire des cas, cela peut impacter jusqu'à la totalité des *clusters* selon la topologie du réseau et la répartition des identifiants (cf. Figure 3.17).



(a) Le nœud 16 arrive à 1 saut du *cluster-head* 12

(b) Le nœud 12 perd son statut de *cluster-head* et une reconstruction du *cluster* est déclenchée.

FIGURE 3.17 – Arrivée d'un nœud à une distance inférieure à k sauts et d'identifiant plus grand que celui du *cluster-head* : reconstruction non locale

Arrivée d'un nœud à distance à $k+1$ d'un *cluster-head*

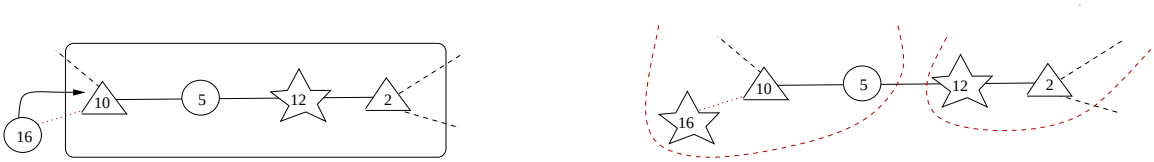
Si le nœud u arrive à une distance de $k+1$ de v et possède un plus petit identifiant $\{(dist_{(u,v)} = k+1) \wedge (id_u < id_v)\}$, alors comme les k sauts sont atteints dans le *cluster* de v, le nœud ne peut pas rejoindre le *cluster* de v. Le nœud devient donc *cluster-head* et forme un *cluster* singleton (cf. Figure 3.18). Dans une telle situation, seule une reconstruction locale est effectuée.

(a) Le nœud 7 arrive à distance 3 sauts du *cluster-head* 12.

(b) Comme les k sauts sont atteints, alors le nœud 7 forme un *cluster* singleton.

FIGURE 3.18 – Arrivée d'un nœud d'identifiant plus petit à une distance à $k+1$ sauts d'un *cluster-head*

Sinon, si le nœud u arrive à une distance de $k+1$ sauts de v et possède un identifiant plus grand $\{(dist_{(u,v)} = k+1) \wedge (id_u > id_v)\}$, alors le *cluster* de v est « bouleversé » (cf. Figure 3.19). Le nœud u attire tous les nœuds avec des identifiants plus petits pour former un nouveau *cluster*. Cela peut conduire dans le pire à cas à une reconstruction de la totalité des *clusters*.



(a) Le nœud 16 arrive à distance 3 saut du *cluster-head* 12.

(b) Comme le 16 possède un identifiant plus grand, alors un reconstruction est déclenchée.

FIGURE 3.19 – Arrivée d'un nœud d'identifiant plus grand à une distance de $k+1$ sauts d'un *cluster-head*.

3.5.3 Observation générale

Nous venons d'illustrer par observation le comportement de SDEAC en présence de pannes transitoires dues à des modifications topologiques (apparition et disparition de nœuds dans le réseau). Dans certains cas de disparition de nœuds de statut NM ou NP ou l'arrivée d'un nœud d'identifiant plus petit dans le réseau, nous avons observé une reconstruction locale. Cependant, dans d'autres cas en l'occurrence la disparition d'un CH d'un *cluster* ou l'arrivée d'un nœud d'identifiant supérieur dans un *cluster* (distance inférieure à k sauts du *cluster-head*) ou aux abords d'un *cluster* (distance $k+1$ sauts du *cluster-head*), le *cluster* en question est reconstruit. Dans le pire des cas, en fonction de la topologie du réseau et de la répartition des identifiants, une reconstruction de la totalité des *clusters* du réseau peut se faire. Par exemple, nous pouvons remarquer que dans le cas de la chaîne ordonnée, l'arrivée d'un nœud d'identité plus grande à l'extrémité où se trouve le nœud d'identité maximale de la chaîne ou la disparition du nœud d'identité maximale de la chaîne entraine forcement une reconstruction qui impacte toute la chaîne ordonnée. Comme il existe une multitude de topologies quelconques ainsi qu'une infinité de façon de répartition des identifiants dans de telles topologies, dans la prochaine section, nous avons recours à une campagne de simulation pour évaluer l'impact moyen de pannes transitoires.

3.6 Validation de SDEAC par simulation

Dans cette section, nous présentons une campagne de simulation dans le but d'évaluer les performances moyennes de SDEAC.

Par une preuve formelle dans la Section 3.4, nous avons montré que partant d'une configuration quelconque, SDEAC structure le réseau en *clusters* disjoints de diamètre au plus $2k$ au plus en $n + 2$ transitions. Nous avons aussi montré que ce temps de stabilisation représente le « pire des cas » et correspond à une topologie bien particulière qui est la chaîne ordonnée. Ainsi, pour évaluer le temps de stabilisation de SDEAC en moyenne c'est-à-dire dans les cas les plus fréquents correspondants à des topologies quelconques, nous avons recours à une campagne de simulations.

3.6.1 Environnement et paramètres de simulation

Environnement de simulation

Dans les évaluations de performances moyennes de SDEAC que nous présentons dans cette section et dans toutes les autres mesures de performances que nous effectuons dans nos travaux de recherche et que nous présenterons dans la suite de ce document, nous utilisons le simulateur OMNeT++ [2] (*Objective Modular Network Test-bet in C++*) [VH]. C'est un simulateur à événements discrets orienté objet et écrit en C++. Il a été conçu pour simuler des systèmes réseaux de communication, des systèmes multi-processeurs, et d'autres systèmes distribués. C'est un projet open source dont le développement a commencé en 1992 par Andras Vargas à l'université de Budapest.

Pour nos topologies réseaux, nous générons des graphes aléatoires à l'aide de la libraire open source SNAP [Les13] (Stanford Network Analysis Platform) développé à l'Université de Stanford. Toutes nos simulations sont exécutées sous Grid'5000 [BCC+06] en utilisant les nœuds de Saint Rémi du Centre de Calcul de Champagne-Ardenne ROMEO [3].

Paramètres de simulation et métriques d'évaluation

Dans toutes nos simulations et les mesures que nous effectuons, nous considérons une variation de trois paramètres : la taille du réseau, la densité de connexité et le paramètre k qui représente le rayon maximal des *clusters*. Pour la taille du réseau, nous considérons des valeurs allant de 100 à 1000 nœuds par pas de 100 afin d'étudier le passage à l'échelle de notre solution. Pour étudier l'impact de la connexité du réseau, nous utilisons d'une part des topologies réseaux suivant un modèle de Erdos Renyi [ER61], qui représente un modèle de base dans les graphes aléatoires, où nous faisons varier la valeur de son paramètre de densité. Nous utilisons également des graphes $\delta-$réguliers, avec δ étant le degré des nœuds (nombre de voisins), où nous faisons varier la valeur de δ pour étudier son impact sur les performances de SDEAC. Pour le paramètre k, nous prenons des valeurs comprises entre 2 et 10 afin de déterminer également son impact dans les performances de notre algorithme.

2. http ://www.omnetpp.org

3. Le Centre de Calcul de Champagne-Ardenne ROMEO est une plateforme technologique de l'Université de Reims Champagne-Ardenne soutenue par la région Champagne-Ardenne. https ://romeo.univ-reims.fr

Pour les métriques à mesurer dans les évaluations de performances de SDEAC, nous nous focalisons d'abord, dans un environnement sans faute, sur le temps de stabilisation comme étant le nombre de transitions nécessaire pour atteindre la stabilisation du réseau en partant d'une configuration quelconque. Nous évaluons ce temps de stabilisation en fonction de la taille du réseau, de sa densité de connexité et en fonction du paramètre k. Ensuite, en partant d'une configuration stable, nous introduisons des pannes transitoires dans le réseau puis nous évaluons le coût nécessaire pour leurs corrections c'est-à-dire le nombre de transitions supplémentaires pour retrouver une confirmation légale du réseau. Dans chacune de nos simulations, pour chaque paramètre fixé nous faisons une série de 100 simulations. Puis, nous mesurons la métrique correspondante comme étant la moyenne résultante sur 100 exécutions.

Dans la suite de cette section, nous présentons les résultats d'évaluations. Nous nous limitons aux mesures décrites ci-dessus. Précisons déjà que d'autres métriques telles que la consommation énergétique de SDEAC ainsi que le pourcentage de *clusters* construit seront évalués dans le prochain chapitre.

3.6.2 Impact de la taille du réseau et du degré des nœuds

Nous entamons les évaluations des performances moyennes de SDEAC en étudiant l'impact de la densité et du nombre de nœuds du réseau sur le temps de stabilisation du réseau. Dans la première série d'expériences présentée au niveau la figure 3.20, nous fixons le paramètre $k = 2$ et nous faisons varier le nombre de nœuds de 100 à 1000 par pas de 100. Pour chaque taille de réseau fixée, à l'aide de la librairie SNAP, nous générons des graphes aléatoires $\delta-$réguliers en fixant δ à 3, 5 et 7. Nous calculons le temps de stabilisation moyen comme étant le nombre de transitions obtenus jusqu'à la formation des *clusters* stables.

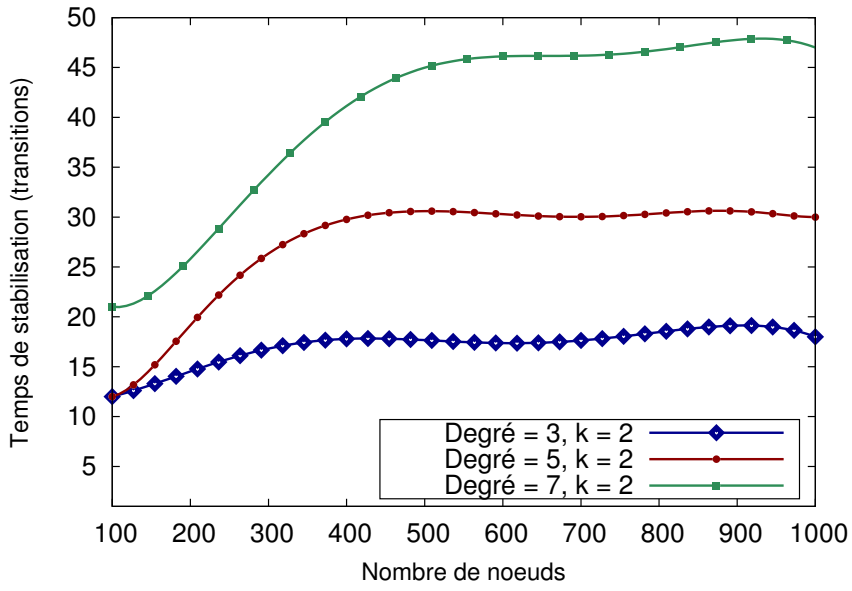


FIGURE 3.20 – Impact de la taille du réseaux sur le temps de stabilisation

Ainsi, au niveau de la figure 3.20, nous remarquons que le temps de stabilisation moyen croit légèrement avec l'augmentation du nombre de nœuds mais varie peu à partir d'une certaine taille du réseau. De plus, nous notons que pour des topologies totalement aléatoires, le temps de stabilisation moyen est très en dessous de $n + 2$ transitions (valeur formelle prouvée dans le pire des cas).

Pour mieux observer l'impact du degré des nœuds sur le temps de stabilisation, comme illustré avec la figure 3.21, nous fixons la taille du réseau et nous faisons varier le degré de nœuds. Nous observons que pour des tailles de réseaux fixées à 100, 200 et 400 nœuds, 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 du réseau ont plus de voisins. Donc, ils attirent plus de nœuds dans leurs *clusters* durant chaque transition. Ainsi, avec notre approche, plus le degré des nœuds augmente, plus le temps de stabilisation diminue. Or, les réseaux *ad hoc* sont souvent caractérisés par une forte densité.

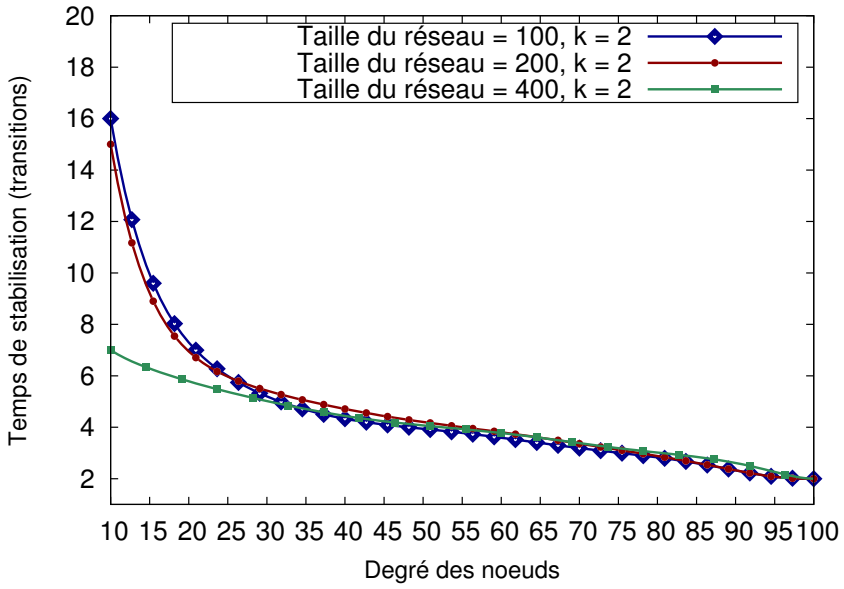


FIGURE 3.21 – Impact du degré des nœuds sur le temps de stabilisation

3.6.3 Étude du passage à l'échelle

Pour étudier le passage à l'échelle de SDEAC, nous faisons varier le nombre de nœuds dans le réseau en même temps que la densité de connexité du réseau. Pour $k = 2$, nous faisons évoluer le nombre de nœuds de 100 à 1000 par pas de 100. Pour chaque taille de réseau fixée, nous faisons varier la densité du réseau de 10% à 100% par pas de 10 en générant des graphes aléatoires suivant le modèle de Erdos Renyi [ER61] à l'aide de la librairie SNAP. Nous obtenons la courbe *3D* de la figure 3.22.

Nous remarquons que sauf pour de faibles densités (10% et 20%), le temps de stabilisation varie légèrement avec l'augmentation du nombre de nœuds. Et en cas de faibles densités, nous observons un pic. Mais avec l'augmentation du nombre de nœuds, le temps de stabilisation diminue et nous observons le même phénomène que la figure 3.21. Avec cette série de simulations, nous pouvons soulever deux observations. *(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 chaîne ordonnée comme nous l'avons prouvé dans la Section 3.4.

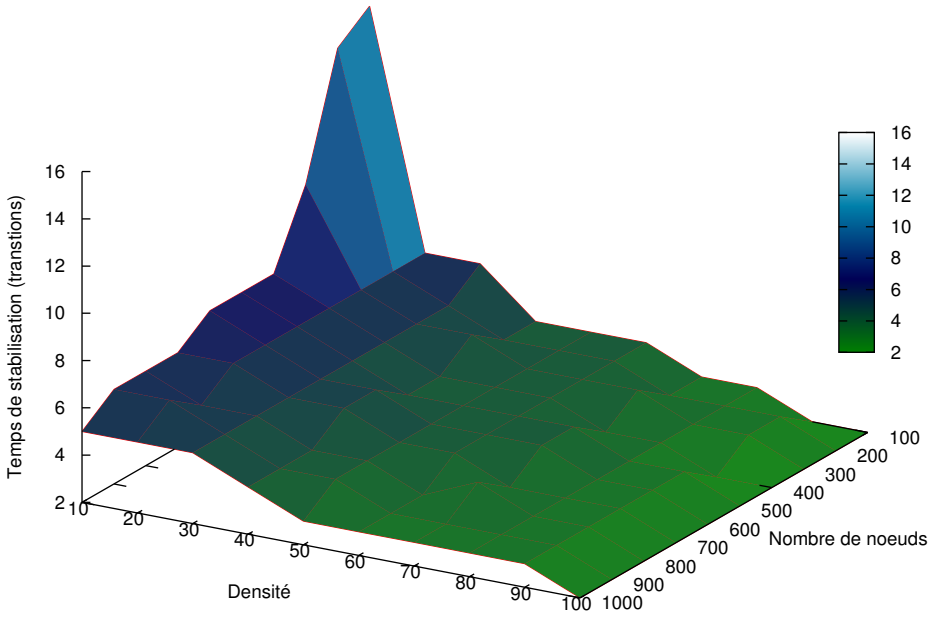


FIGURE 3.22 – Passage à l'échelle

3.6.4 Impact du rayon des *clusters* : le 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 3.23 montre le temps de stabilisation en fonction de la valeur de 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 champ d'influence des nœuds avec une plus grande identité augmente. Les nœuds effectuent moins de transitions pour se *fixer* à un *cluster-head*. 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).

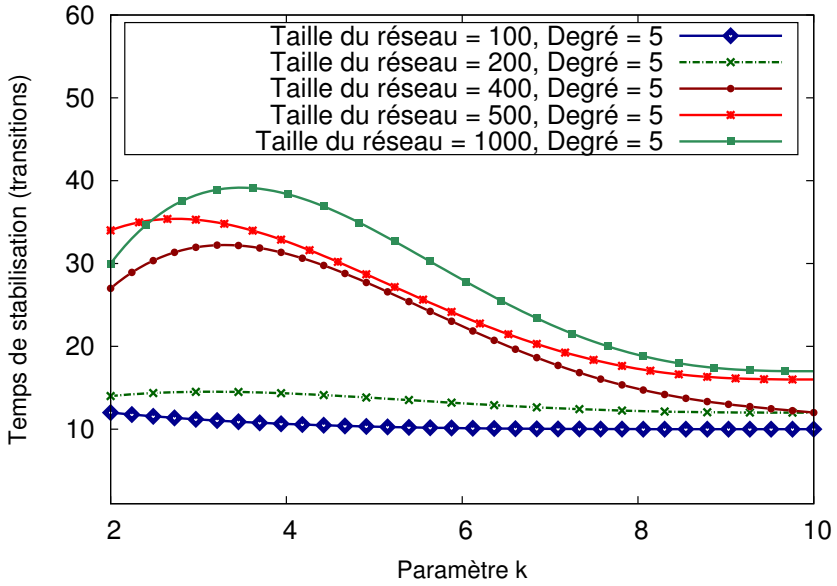


FIGURE 3.23 – Impact du paramètre k

3.6.5 Impact de pannes transitoires

Dans la Section 3.5, nous avons illustré par observation le comportement de SDEAC en présence de pannes transitoires. Ainsi, nous présentons ici une campagne de simulation pour évaluer l'impact de pannes transitoires. Pour ce faire, nous exécutons SDEAC jusqu'à la stabilisation (c'est-à-dire jusqu'à ce que le réseau soit entièrement structurer en *cluster*). Ensuite, nous introduisons une panne transitoire en modifiant la topologie du réseau par la disparition d'un certain nombre de nœuds. Ainsi, nous évaluons le nombre de transitions supplémentaires requis pour retrouver un nouvel état stable dans l'ensemble du réseau. Nous adoptons les deux scénarios suivants :

(i) Dans un premier temps, nous considérons un réseau de taille allant de 100 à 1000 nœuds. Pour chaque taille de réseau fixée, nous faisons disparaitre 1, 3, et 5 nœuds du réseau ;

(ii) Dans un second temps, nous fixons un réseau de taille 1000 nœuds et nous faisons disparaitre 1% à 5% des nœuds du réseau.

Impact de la disparition de 1, 3 et 5 nœuds du réseau

Ici, nous considérons une variation du nombre de nœuds du réseau de 100 à 1000 par pas de 100. Puis, pour chaque taille de réseau, nous faisons disparaitre 1, 3 et 5 nœuds. Précisons que dans le cas de la disparition de 1 nœud, c'est celui avec la plus grande identité que nous faisons disparaitre. Pour 3 nœuds, nous faisons disparaitre le nœud d'identité maximale et 2 autres choisis aléatoirement. Et pour le cas 5 nœuds, nous faisons toujours disparaitre celui d'identité maximale avec 4 autres nœuds choisis aléatoirement. Pour chaque taille de réseau en

fixant arbitrairement le degré moyen du réseau 6 et le paramètre k à 2. La figure 3.24 donne le nombre de transitions supplémentaires pour retrouver un nouvel état légal dans l'ensemble du réseau.

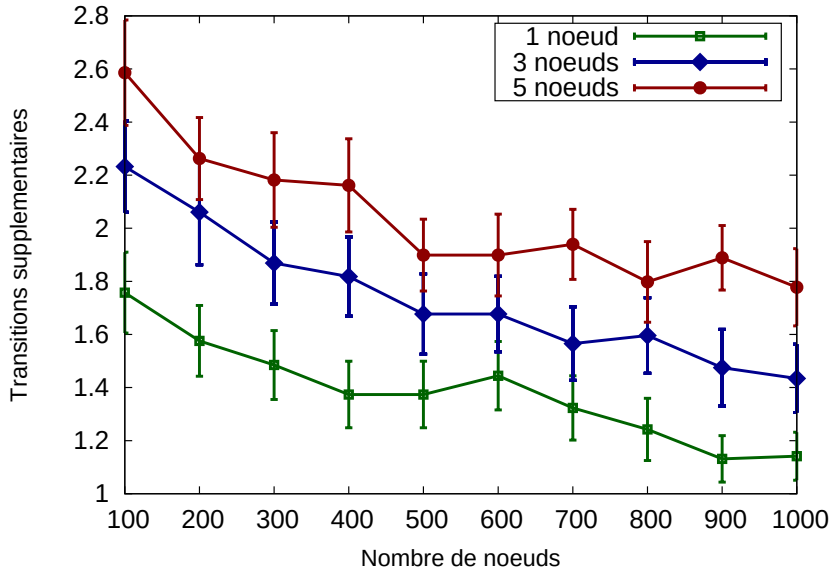


FIGURE 3.24 – Disparition de 1, 3 et 5 nœuds : transitions supplémentaires

D'une part, nous remarquons que pour chaque nombre de nœuds disparus fixé, le nombre de transitions supplémentaires requises diminue avec l'augmentation de la taille du réseau. Ceci s'explique par le fait qu'avec l'augmentation de la taille du réseau, le nombre de nœuds impactés diminue. En effet, la figure 3.25 montre que le pourcentage de nœuds impactés diminue avec l'augmentation de la taille du réseau.

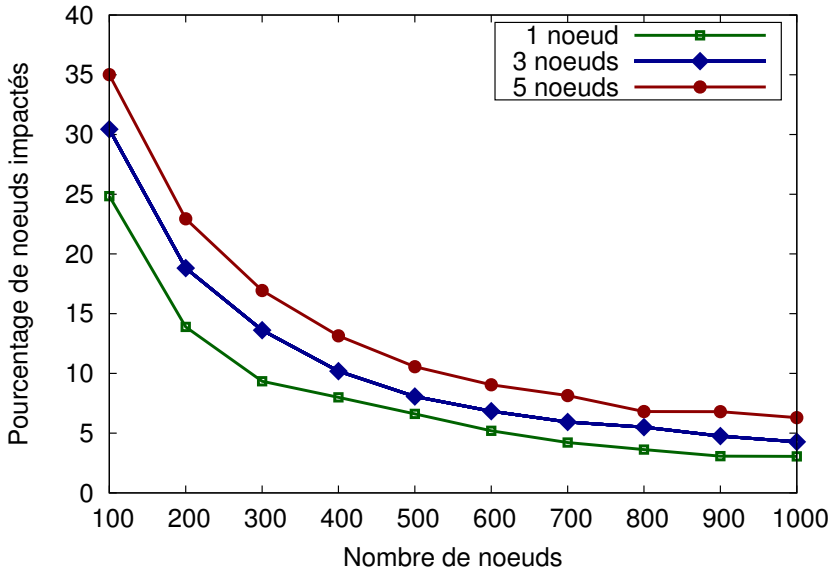


FIGURE 3.25 – Disparition de 1, 3 et 5 nœuds : pourcentage de nœuds impactés

D'autre part, l'observation majeure que nous faisons est que, suite à l'occurrence de fautes transitoires, le nombre de transitions supplémentaires requis pour que le réseau retrouve un état stable est minimal comparé au temps de stabilisation partant d'une configuration quelconque (cf. Figure 3.20, Figure 3.21 et Figure 3.22). En effet, une fois qu'un état stable est atteint dans le réseau, suite à l'occurrence de pannes transitoires qui modifient la structure des *clusters*, les nœuds réutilisent les informations déjà présentes dans leurs tables de voisinage durant la procédure de *re-clustrering*. Ce qui diminue ainsi le nombre de transitions nécessaire pour retrouver un nouvel état stable.

Impact de la disparition de 1 à 5 % des nœuds du réseau

Ici, pour mieux mesurer l'impact de pannes transitoires, nous fixons un réseau de 1000 nœuds avec un degré moyen du réseau fixé à 6 et le paramètre k à 2, ensuite nous exécutons SDEAC jusqu'à la stabilisation. Puis, nous introduisons des pannes transitoires en faisant disparaitre aléatoirement 1 à 5 % des nœuds du réseau. Nous calculons le temps de stabilisation moyen nécessaire pour retrouver un nouvel état légal comme étant le nombre de transitions supplémentaires.

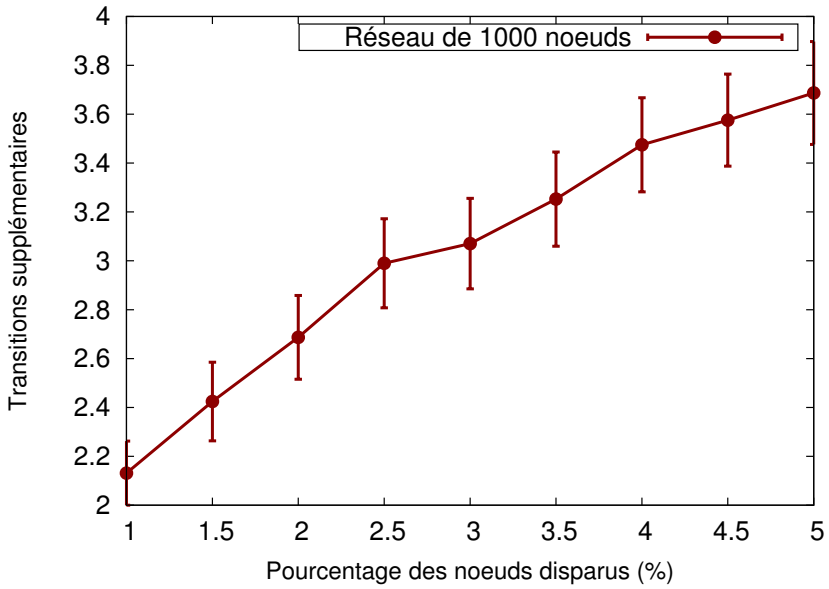


FIGURE 3.26 – Disparition de 1 à 5 % de nœuds : transitions supplémentaires

Dans la figure 3.26, nous observons que le nombre de transitions nécessaires pour corriger les pannes transitoires augmente en suivant l'augmentation du taux de nœuds disparus. En effet, comme illustré dans la figure 3.27, l'augmentation du pourcentage de nœuds disparus fait croitre logiquement le nombre nœuds impactés dans le réseau. De ce fait, il nécessite plus de transitions pour que le réseau devienne stable à nouveau. La figure 3.27 montre qu'une disparition jusqu'à 5% des nœuds du réseau n'impacte environ que 1/4 de la taille du réseau. Ceci montre que les pannes transitoires n'impactent que les *clusters* dans lesquels elles se sont produites et les *clusters* adjacents. Également, nous constatons que le nombre de transitions

supplémentaires requises pour que le réseau retrouve un état stable est minimal comparé au temps de stabilisation partant d'une configuration quelconque (cf. Figure 3.20, Figure 3.21 et Figure 3.22).

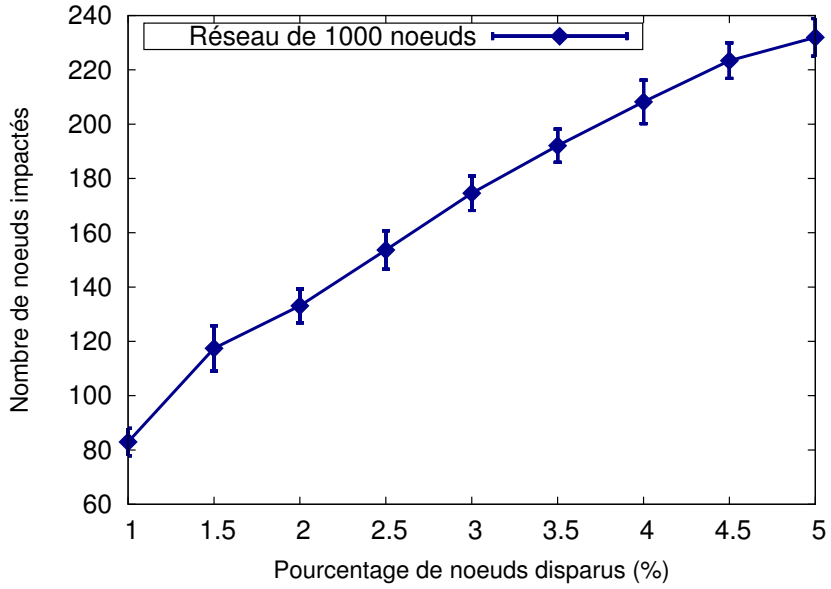


FIGURE 3.27 – Disparition de 1 à 5 % de nœuds : nombre de nœuds impactés

En résumé, les résultats expérimentaux illustrés dans les figures 3.24, 3.25, 3.26 et 3.27 montrent que la correction de pannes transitoires nécessites un nombre minimal de transitions. Les pannes transitoires ne conduisent pas à une reconstruction totale du réseau mais impactent les *cluters* dans lesquels elles se sont produites et les *clusters* adjacents. Les nœuds réutilisent les informations déjà présentes dans leurs tables de voisinage pour la correction de pannes transitoires. Ainsi, SDEAC s'adapte bien aux pannes transitoires dues à des modifications topologiques.

3.7 Conclusion

À travers ce chapitre, nous avons présenté un algorithme original appelé SDEAC qui structure le réseau en *clusters* non-recouvrants de diamètre au plus $2k$ en au plus $n+1$ transitions (cas de la chaîne ordonnée). De plus, l'exécution de SDEAC nécessite une occupation mémoire de $(\Delta_u+1)\times\log(2n+k+3)$ bits pour chaque nœud u du réseau soit au plus $n*\log(2n+k+3)$ bits pour un graphe complet. Avec Δ_u étant le degré - nombre de voisins - de chaque nœud u, k le rayon maximal des *clusters* et n la taille du réseau. Par une campagne de simulation sous OMNeT++, nous avons évalué les performances moyennes de SDEAC. Les résultats ont montré que pour des topologies réseaux quelconques, le temps de stabilisation moyen est très inférieur à celui du pire des cas. Nous avons également constaté par les simulations que, pour des topologies réseaux quelconques, après la stabilisation du réseau, le nombre de transitions requis pour corriger une panne transitoire causée par une modification topologique est minimal

comparé à celui du *clustering*.

Les travaux que nous avons présentés dans ce chapitre ont fait l'objet d'une (1) revue internationale [BFH+13a], deux (2) revues francophones [BFH+12a, BFH+14], deux (2) articles conférences internationales [BFH+12b, BFH+13b] avec un **Best Paper Award** dans [BFH+13b] et deux (2) articles en conférences francophones [BFH+12c, BFH+12d].

Dans le Chapitre 4, nous étudions la consommation énergétique SDEAC dans le contexte des réseaux de capteurs sans fil puis nous la comparons avec la solution de *clustering* opérant dans le même modèle.

Chapitre 4

Étude de la consommation énergétique de SDEAC dans les réseaux de capteurs sans fil

Résumé.

Dans ce chapitre, nous étudions dans un premier temps la consommation énergétique de SDEAC dans le contexte des réseaux de capteurs sans fils. Pour cela, nous proposons une généralisation du critère d'élection des cluster-heads. Nous montrons que SDEAC est une approche générique et qu'elle peut s'utiliser aisément avec d'autres critères tels que le degré ou l'énergie des nœuds pour l'élection des cluster-heads. Puis, dans un second temps, nous menons une étude comparative des coûts et performances de SDEAC avec la solution de structuration auto-stabilisante, distribuée, déterministe, fondée sur un modèle asynchrone à passages de messages et construisant des clusters à k sauts.

*Les travaux que nous exposons dans ce chapitre ont fait l'objet de plusieurs publications dans des revues et conférences avec actes et comités de sélection : une (1) revue internationale [BFM+13d], deux (2) articles en conférences internationales [BFM+13a, BFM+13c] dont un **Best paper Award** dans [BFM+13c] et un (1) article en conférence d'audience francophone [BFM+13b].*

Sommaire

4.1 Motivations et objectifs

Dans la première partie de ce chapitre, nous étudions d'abord les performances de SDEAC dans le contexte des RCSF en évaluant sa consommation énergique et le pourcentage de *clusters* construits. Dans cette étude, nous proposons une généralisation du critère d'élection des *cluster-heads* de SDEAC. En d'autres termes, outre le critère principal d'élection des *cluster-head* qui est l'identité maximale, nous montrons que SDEAC est une solution de structuration générique qui peut s'utiliser avec d'autres critères d'élection parmi les plus usuels tels que le degré ou l'énergie résiduelle des nœuds. Ensuite, nous évaluons le comportement de SDEAC dans un environnement de RCSF sujet à des pannes transitoires causées par des modifications topologiques. Pour ce faire, nous mesurons la consommation énergétique nécessaire pour retrouver un état stable dans le réseau suite à l'occurrence de pannes transitoires. Pour mesurer la consommation énergétique de SDEAC, nous implémentons le modèle énergétique de Heinzelman et *al.* [HCB00] (cf. Section 1.3.3 du Chapitre 1)

Dans la seconde partie de ce chapitre, nous comparons SDEAC avec la seule solution à notre connaissance opérant dans les mêmes hypothèses et modèles : celle de Mitton et *al.* [MFGLT05]. Ainsi, en se plaçant dans les mêmes topologies réseaux c'est-à-dire des graphes aléatoires suivant une loi de Poisson (cf. Section 1.3.3 du Chapitre 1) et en utilisant le modèle énergétique de Heinzelman et *al.* [HCB00], nous comparons les coûts et performances de SDEAC avec la solution de Mitton et *al.* [MFGLT05].

La suite de ce chapitre est organisée comme suit. Dans la Section 4.2, nous présentons la généralisation du critère d'élection des *cluster-heads* de SDEAC et les mesures de performances effectuées. Ensuite, au niveau de la Section 4.3, nous étudions le comportement de SDEAC dans un environnement sujet à des pannes transitoires. Dans la Section 4.4, nous menons une étude comparative des performances de SDEAC avec la solution de Mitton et *al.* [MFGLT05]. Nous clôturons ce chapitre par une synthèse dans la Section 4.5.

4.2 Étude de SDEAC dans le cadre des RCSF

Ici, nous étudions SDEAC dans le contexte des RCSF en évaluant ses performances suivant plusieurs critères d'élection des *cluster-heads*.

4.2.1 Généralisation du critère d'élection des *cluster-heads*

Il existe plusieurs critères pour élire les *cluster-heads*. Pour mieux choisir l'un d'eux, il est important d'étudier l'influence de chacun des critères d'élection sur les performances de l'algorithme de *clustering*. Nous proposons une généralisation de SDEAC en considérant pour l'élection des *cluster-heads* les critères les plus utilisés à savoir l'identité, le degré et l'énergie résiduelle des nœuds [CKT02, CW06, BAR07, AD08, LGF08b, BAKA$^+$13, WZX$^+$13]. Pour chacun de ces trois critères, nous mesurons et comparons leurs consommations énergétiques totales et leurs pourcentages de *clusters* qu'ils construisent. Le pourcentage de *clusters* est défini comme étant le nombre de *cluster-heads* divisé par le nombre total de nœuds dans le réseau [YF04, CS10, AVX$^+$12]. Après avoir présenté SDEAC sur la base du critère d'identité

maximale dans le Chapitre 3, dans les deux prochaines sections, nous allons décrire le critère du degré et de l'énergie résiduelle.

Critère du degré des nœuds

Dans cette approche, la décision concernant le choix du nœud le plus adéquat pour devenir *cluster-head* se fait en fonction du degré D (nombre de voisins) des nœuds. Il existe deux types d'approches se fondant sur le degré. Celle privilégiant le nœud ayant le degré maximal et celle privilégiant le nœud ayant le degré relatif $\triangle_d$ qui se rapproche le plus du degré idéal ρ. Le degré idéal (ρ) représente le nombre optimal de nœuds que peut générer efficacement un *cluster-head* sans être saturer [CW06]. En effet, des études précédentes, comme [CKT02, CW06, BAR07], ont montré que lors de la phase de routage où les messages sont routés via les *cluster-heads*, un *cluster-head* qui a un degré très élevé consomme son énergie très rapidement à cause du nombre de messages qu'il recoit/envoie à ses voisins. Pour mieux répartir la consommation d'énergie, il est important de limiter le nombre de voisins des *cluster-heads*. Le meilleur candidat est alors 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 *cluster-head*, le choix se fait en fonction d'un critère secondaire qu'est l'identité maximale des nœuds. Ce critère est discriminant dans la mesure où chaque nœud possède une identité unique.

Critère de l'énergie résiduelle

Dans cette approche, le choix des *cluster-heads* se fait en fonction de l'énergie résiduelle (restante) des nœuds. En effet, le *cluster-head* est le nœud le plus sollicité et a donc besoin d'une réserve d'énergie plus importante que celle des autres nœuds afin d'éviter la reconstruction fréquente des *clusters*. Cependant, un candidat à la fonction de *cluster-head* avec initialement le maximum d'énergie résiduelle en consomme une partie au fur et à mesure de la procédure de *clustering*. Après quelques 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 comme *cluster-head*. Afin d'éviter de changer fréquemment de candidat à cause d'un gain en énergie pouvant être négligeable et afin de garantir plus de stabilité à la procédure de *clustering*, nous considérons une variante de cette approche se fondant sur un seuil d'énergie $\mathcal{E}_\mathcal{S}$ [BAKA$^+$13, WZX$^+$13]. Ainsi, tant que la différence d'énergie entre le candidat initial i et le nouveau j ($\triangle_e = |\mathcal{E}_i - \mathcal{E}_j|$) est inférieure au seuil d'énergie $\mathcal{E}_\mathcal{S}$, le candidat initial conserve son statut de candidat idéal pour devenir *cluster-head*. Notons que pour le critère de l'énergie résiduelle comme pour le seuil d'énergie, l'identité maximale des nœuds est utilisée comme critère secondaire pour départager deux nœuds candidats à la fonction de *cluster-head* ayant le même niveau d'énergie.

4.2.2 Métriques d'évaluation et paramètres de simulation

Après avoir décrit les critères d'élection des *cluster-heads*, nous présentons dans cette section les métriques d'évaluation et les paramètres de simulation que nous avons considéré lors des mesures de leurs performances.

Métriques d'évaluation

Nous évaluons les différents critères d'élection des *cluster-heads* en considérant, dans les mêmes conditions et environnements de tests, deux métriques : la consommation énergétique totale et le pourcentage de *clusters* construits. Ces deux métriques sont définies de la façon suivante.

- **Consommation énergique totale** : nous définissons la consommation énergétique totale, notée $\mathcal{E}_{totale}$, comme étant la quantité totale d'énergie consommée dans tout le réseau jusqu'à la formation de *clusters* stables.
  La consommation énergétique totale dépend fortement des communications c'est-à-dire des échanges de messages dans le réseau. En d'autres termes, plus il y a d'échanges de messages dans le réseau, plus la consommation énergétique est importante. En effet, comme la consommation énergétique pour les traitements CPU et les opérations de captage est très négligeable comparée à celle des communications (envois/réceptions de messages), elle est négligée dans le calcul de $\mathcal{E}_{totale}$ [HCB00, PK00, SDT07, ACDFP09, KEAC14]. Ainsi, sur la base du modèle énergétique de Heinzelman et *al.* [HCB00], la consommation énergétique totale est obtenue comme suit :

$$\mathcal{E}_{totale} = \sum_{i=0}^{n-1} \mathcal{M}_i^{env} \times E_{Tx}(l,d) + \sum_{i=0}^{n-1} \mathcal{M}_i^{rec} \times E_{Rx}(l) \tag{4.1}$$

  Dans l'équation 4.1, le terme $E_{Tx}(l,d)$ représente la quantité d'énergie consommée pour envoyer un message de longueur l bits sur une distance d exprimée en mètre. Cette quantité est calculée à partir de l'équation 1.2 de Heinzelman et *al.* (cf. Page 29). $E_{Rx}(l)$ est la quantité d'énergie consommée pour réceptionner un message de longueur l bits et elle est calculée à partir de l'équation 1.3 de Heinzelman et *al.* (cf. Page 30). $\mathcal{M}_i^{env}$ représente le nombre total de messages envoyés et $\mathcal{M}_i^{rec}$ désigne le nombre total de messages reçus par un capteur i. n désigne le nombre total de nœuds dans le réseau.
- **Pourcentage de *clusters*** : nous définissons le pourcentage de *clusters* construits, noté $\mathcal{P}_{CH}$, comme étant le nombre de *cluster-heads* (noté N_{CH}) sur le nombre total de nœuds du réseau [YF04, CS10, AVX+12]. Le pourcentage de *clusters* est calculé à partir de l'équation 4.2.

$$\mathcal{P}_{CH} = \frac{N_{CH}}{n} \times 100 \tag{4.2}$$

Paramètres de simulation

Les paramètres de toutes les simulations que nous avons effectuées sous le simulateur OMNeT++ sont résumés dans le tableau 4.1. Comme nous considérons des RCSF hétérogènes, nous dotons à chaque nœud du réseau une énergie initiale ($\mathcal{E}_i^{init}$) de 1, 2 ou 3 joules. Ces valeurs représentent les plus petites utilisées dans la littérature [YP12, LZXG12, GJS+12, GYP13]. En outre, cela permet de prendre en compte les faibles capacités énergétiques des capteurs. Nous considérons aussi des messages de 2000 bits de longueur et une distance entre les nœuds de 100 mètres [HCB00]. Nous faisons varier la valeur du paramètre k de 1 à 10 et celle du paramètre λ (degré moyen du réseau) de 2 à 11. Nous considérons, dans les simulations, des réseaux de taille comprise entre 100 et 1000 nœuds par pas de 100 afin de prendre en compte le passage à l'échelle dans les mesures de performances.

Afin d'évaluer avec précision les mesures de simulation à travers de multiples exécutions (100 exécutions), nous calculons chacune des valeurs obtenues avec un intervalle de confiance de 99% [Jai91]. Les intervalles de confiance sont matérialisés dans les courbes par des barres d'erreurs. Dans certaines mesures, elles ne sont peu ou pas visibles ce qui atteste de la précision des mesures. Dans le cas où elles sont visibles, plus les courbes passent au milieu des barres d'erreurs, plus les mesures sont précises

	Paramètres	**Valeurs**
Constantes	Taille d'un message (l)	2000 bits
	Distance entre 2 nœuds (d)	100 mètres
	Modèle de graphes	Distribution de Poisson
	Nombre de simulations pour chaque taille du réseau	100
	Intervalle de confiance	99%
Variables	Énergie initiale	{1,2,3} Joules
	Paramètre k	[1,10]
	Degré moyen du réseau	[2,11]
	Degré idéal	{6,10,12,20}
	Seuil d'énergie	{0.1%,0.05%}
	Nombre de nœuds	[100,1000]

TABLE 4.1 – Paramètres de simulation pour l'évaluation de SDEAC dans les RCSF

4.2.3 Résultats d'évaluation des critères d'élection des *cluster-heads*

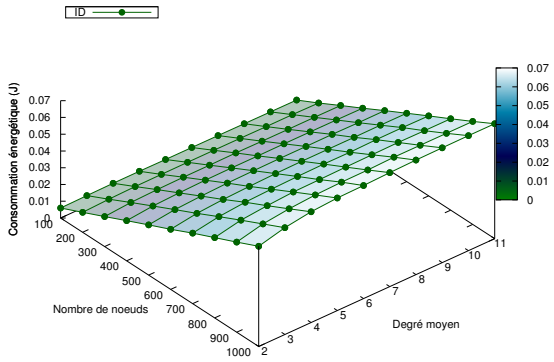
Dans cette section, nous présentons les résultats obtenus lors de l'évaluation de la consommation énergétique totale et du pourcentage de *clusters* de chaque critère d'élection des *cluster-heads* (identité maximale, degré maximal et énergie résiduelle).

Consommation énergétique totale

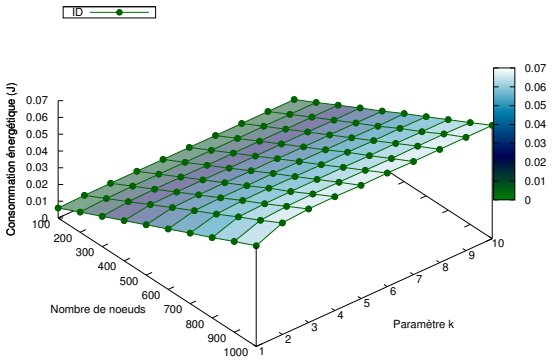
Pour chaque critère d'élection des *cluster-heads*, nous mesurons la consommation énergétique totale (en joule) nécessaire à la formation de *clusters* stables en fonction du degré moyen du réseau, du paramètre k et de la taille du réseau.

Pour évaluer l'énergie consommée en fonction du degré moyen du réseau, nous fixons la valeur du paramètre k à 2. Puis, nous faisons varier la valeur du degré moyen du réseau (paramètre λ) de 2 à 11. Pour chaque valeur du degré moyen fixée, nous considérons des réseaux avec un nombre total de nœuds variant de 100 à 1000. Les figures 4.1(a), 4.1(c) et 4.1(e) représentent respectivement la consommation énergétique totale du critère de l'identité maximale, du degré maximal et de l'énergie résiduelle en fonction du degré moyen et de la taille du réseau.

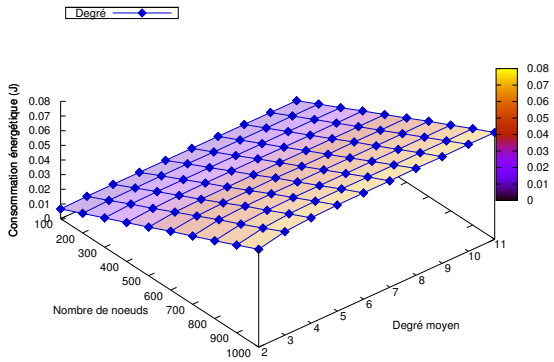
Pour évaluer la consommation énergétique en fonction du paramètre k, le degré moyen du réseau est fixé à 6 et le paramètre k varie de 1 à 10. Pour chacune des valeurs du paramètre k, nous considérons des réseaux de taille variant de 100 à 1000 nœuds. Les figures 4.1(b), 4.1(d) et 4.1(f) représentent respectivement la consommation énergétique totale du critère de l'identité



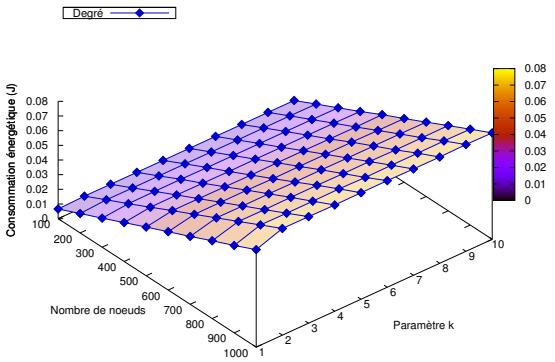
(a) Consommation énergétique du critère de l'identité maximale en fonction du degré moyen du réseau



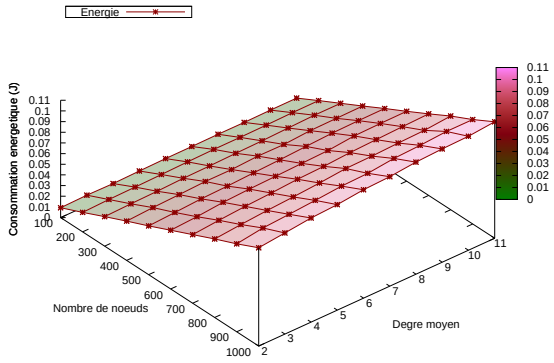
(b) Consommation énergétique du critère de l'identité maximale en fonction du paramètre k



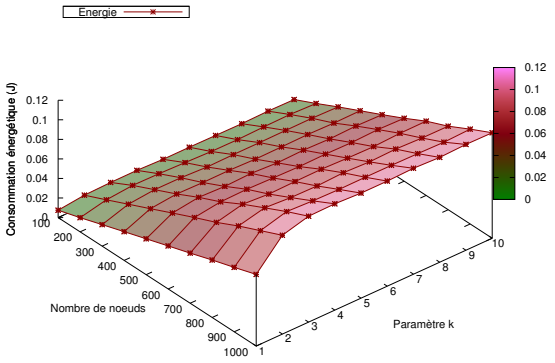
(c) Consommation énergétique du critère du degré maximal en fonction degré moyen du réseau



(d) Consommation énergétique du critère de degré maximal en fonction du paramètre k



(e) Consommation énergétique du critère de l'énergie résiduelle en fonction du degré moyen du réseau



(f) Consommation énergétique du critère de l'énergie résiduelle en fonction du paramètre k

FIGURE 4.1 – Consommation énergétique totale (en joule) de chaque critère d'élection des *cluster-heads* en fonction de la taille du réseau, du degré moyen et du paramètre k

maximale, du degré maximal et de l'énergie résiduelle en fonction du paramètre k et de la taille du réseau.

Pour chaque critère d'élection, nous observons d'une part que, pour chacune des valeurs du degré moyen du réseau (cf. Figures 4.1(a), 4.1(c) et 4.1(e)) et du paramètre k (cf. figures 4.1(b), 4.1(d) et 4.1(f)), la consommation énergétique totale suit linéairement l'augmentation de la taille du réseau. En effet, l'augmentation du nombre de nœuds du réseau fait croître les échanges de messages dans le réseau. Donc, la consommation énergétique nécessaire à la formation de *clusters* stables suit logiquement l'évolution de la taille du réseau.

Et d'autre part, nous remarquons que, pour chaque taille de réseau considérée, l'augmentation du degré moyen du réseau (cf. Figures 4.1(a), 4.1(c) et 4.1(e)) n'entraine pas une croissance significative de la consommation énergétique totale. En effet, l'évolution du degré moyen du réseau se traduit par une augmentation du nombre de voisins se trouvant dans le rayon de couverture de chaque nœud. Ce qui implique une augmentation du nombre de messages reçus. Or, l'énergie nécessaire pour la réception d'un message de longueur l bits est minimale par rapport à l'énergie de l'émission (cf. Équations 1.2 et 1.3 à la page 29). L'augmentation du paramètre k (cf. Figures 4.1(b), 4.1(d) et 4.1(f)) entraine une légère augmentation de la consommation énergétique totale. Lorsque la valeur du paramètre k fixée augmente, le diamètre des *clusters*. Nous reviendrons sur la conséquence majeure de l'augmentation du paramètre k dans nos prochaines mesures sur le pourcentage de *clusters* construits.

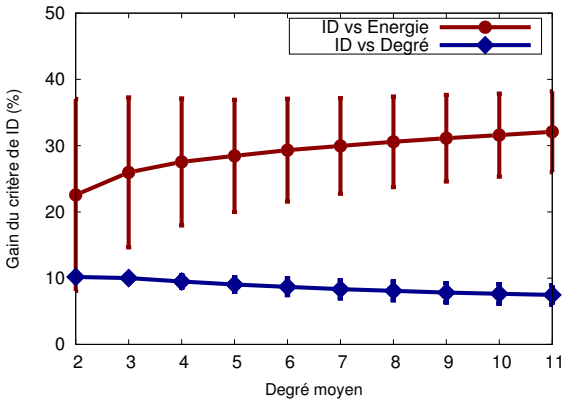
Ainsi, avec les résultats illustrés dans figure 4.1, nous pouvons noter que pour chacun des critères d'élection des *cluster-heads*, SDEAC permet le passage à l'échelle dans la consommation énergétique totale du réseau.

Gain dans la consommation énergétique entre les critères d'élection des *cluster-heads*

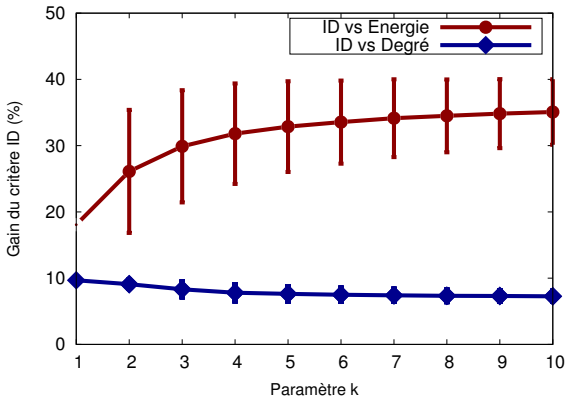
Pour mieux évaluer les critères d'élection des *cluster-heads*, nous comparons leurs consommations énergétiques. Dans la figure 4.2(a), nous évaluons le gain énergétique du critère de l'identité maximale par rapport au degré maximal et à l'énergie résiduelle en fonction du degré moyen du réseau. Dans la figure 4.2(b), nous mesurons ce même gain en fonction du paramètre k.

Nous notons que le critère de l'identité maximale, comparé à celui du degré maximal, réduit la consommation énergétique totale de 7% à $10,2\%$ en fonction du degré moyen du réseau et de $7,3\%$ à 9% en fonction du paramètre k. En effet, pour le critère du degré maximal, les nœuds doivent recevoir un message de tous leurs voisins pour pouvoir déterminer leurs propres degrés avec exactitude. Ensuite, pour élire le *cluster-head*, ils cherchent à localiser le nœud avec le degré le plus élevé situé au plus à k sauts. Alors que pour le critère de l'identité maximale, comme nous l'avons présenté dans le Chapitre 3, les nœuds comparent uniquement leurs identités pour déterminer leur *cluster* d'appartenance.

Nous observons également que le critère de l'identité maximale, comparé à celui de l'énergie



(a) Gain énergétique du critère de l'identité maximale en fonction du degré moyen du réseau



(b) Gain énergétique du critère de l'identité maximale en fonction du paramètre k

FIGURE 4.2 – Réduction de la consommation énergétique par le critère de l'identité maximale

résiduelle, réduit la consommation énergétique de $22,6\%$ à $32,1\%$ en fonction du degré moyen du réseau et de $18,1\%$ à $35,1\%$ en fonction du paramètre k. En effet, comme l'énergie est un facteur qui décroit continuellement dans le temps, un candidat à la fonction de *cluster-head* avec initialement le maximum d'énergie résiduelle en consomme une partie au fur et à mesure de la procédure de *clustering*. Avant l'achèvement de la procédure de *clustering*, 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 comme *cluster-head*. Il en découle que le critère de l'énergie résiduelle nécessite plus d'échanges de messages pour la stabilisation du réseau.

Degré maximal vs Degré idéal

S'agissant du critère du degré maximal, lors de la phase de routage où les messages sont routés via les *cluster-heads*, un *cluster-head* possédant un degré maximal consomme son énergie très rapidement à cause du nombre de messages qu'il reçoit/envoie à ses voisins. Pour optimiser la consommation énergétique des *cluster-heads*, une solution consiste à fixer un degré idéal qui représente le nombre optimal de nœuds qu'un *cluster-head* peut gérer sans être surchargé [CKT02, CW06, BAR07]. Ainsi, nous évaluons les performances de SDEAC en comparant l'usage du degré idéal par rapport au degré maximal.

Dans la figure 4.3, nous fixons un degré moyen du réseau à 6 et le paramètre k à 2. Puis, nous considérons un réseau de taille allant de 100 à 1000 nœuds. Nous choisissons un degré idéal fixé à 10, 12 et 20. Puis, pour chaque valeur du degré idéal, nous comparons la consommation énergétique totale nécessaire à la formation des *clusters* par rapport à celle du degré maximal.

Dans un premier temps, nous observons que pour chaque degré idéal fixé, la consommation énergétique totale suit linéairement l'évolution de la taille du réseau, comme pour le degré maximal. En effet, l'augmentation de la taille du réseau fait croître logiquement les communications et donc la consommation énergétique augmente proportionnellement. Dans un second temps, nous notons que plus le degré idéal choisi est élevé, plus la consommation énergétique totale,

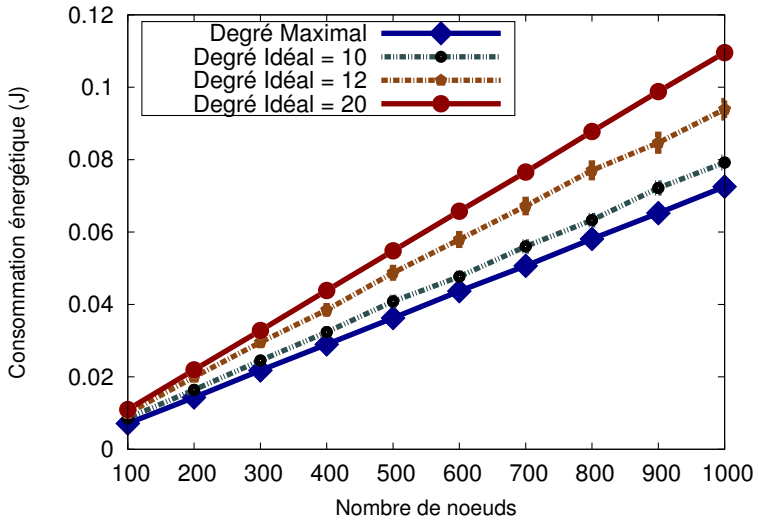


FIGURE 4.3 – Consommation énergétique : Degré maximal vs Degré idéal

pour la formation de *clusters* stables, est importante. En effet, les nœuds vont chercher à élire comme *cluster-head* le candidat minimisant au plus la distance $\triangle_d$, calculée selon l'équation suivante : $\triangle_d = |D - \rho|$. Et ceci nécessite plus d'échanges de messages. Néanmoins, le degré idéal, bien qu'étant souvent plus coûteux lors de la phase de *clustering*, offre l'avantage de paramétrer le nombre de voisins des *cluster-heads* qui se verront ainsi moins solliciter lors de la phase de routage.

Énergie résiduelle vs Seuil d'énergie

Comme l'énergie des nœuds est un facteur qui décroit continuellement dans le temps suivant les communications, le critère de l'énergie résiduelle est plus instable lors de la phase de *clustering*. Cela peut conduire à de fréquents changements de *cluster-heads* candidats. Pour palier ce phénomène, une solution consiste à fixer un seuil d'énergie en dessous duquel les nœuds changent de *cluster-heads* candidats. Nous fixons donc ce seuil et nous comparons la consommation énergétique totale du réseau par rapport au cas où l'énergie résiduelle est considérée comme critère d'élection *cluster-head*.

Dans la figure 4.4, nous fixons le degré moyen du réseau à 6 et le paramètre k à 2. Puis, nous considérons un réseau de taille allant de 100 à 1000 nœuds. Nous choisissons deux seuils d'énergie de $0,1\%$ et $0,05\%$. Puis, pour chacun d'eux, nous comparons la consommation énergétique totale dans le réseau par rapport à celle du critère de l'énergie résiduelle.

D'une part, nous observons que, pour chacun des deux seuils d'énergie comme pour le critère de l'énergie résiduelle, la consommation énergétique totale suit linéairement l'évolution de la taille du réseau. En effet, l'augmentation de la taille du réseau fait croitre logiquement les communications et donc la consommation énergétique suit cette tendance. D'autre part, nous remarquons que, pour les deux seuils d'énergie fixés, la consommation énergétique nécessaire

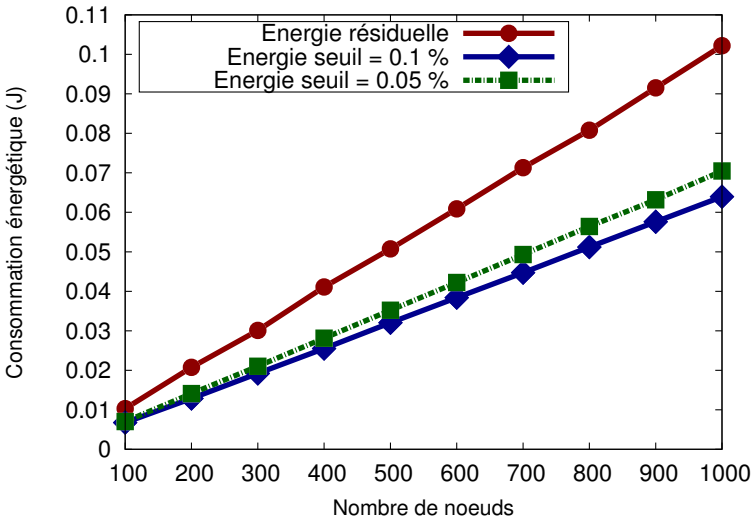


FIGURE 4.4 – Consommation énergétique : Énergie résiduelle vs Seuil d'énergie

à la formation de *clusters* stables est inférieure à celle du critère d'énergie résiduelle. En effet, un seuil d'énergie permet d'améliorer la stabilité lors de la phase de *clustering* en empêchant de fréquents changements de *cluster-heads* candidats à cause d'une infime baisse du niveau d'énergie.

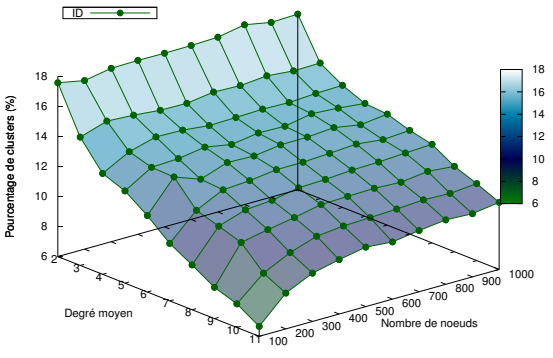
Pourcentage de *clusters* construits

Dans cette section, nous évaluons le pourcentage de *clusters* construits par chaque critère d'élection des *cluster-heads* en fonction du degré moyen du réseau, du paramètre k et de la taille du réseau.

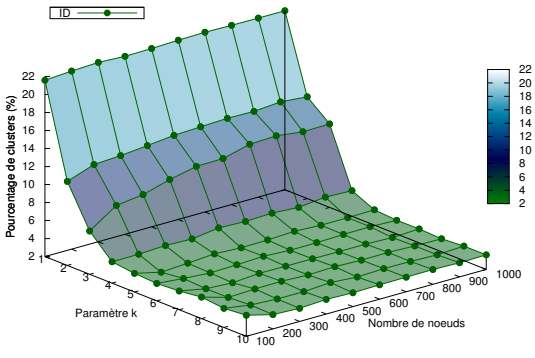
Pour évaluer le pourcentage de *clusters* en fonction du degré moyen du réseau, nous fixons la valeur du paramètre k à 2. Puis, nous faisons varier le degré moyen du réseau de 2 à 11. Pour chaque valeur du degré moyen, nous considérons des réseaux avec un nombre total de nœuds variant de 100 à 1000. Les figures 4.5(a), 4.5(c) et 4.5(e) représentent respectivement le pourcentage de *clusters* construits par le critère de l'identité maximale, du degré maximal et de l'énergie résiduelle en fonction du degré moyen et de la taille du réseau.

Pour évaluer le pourcentage de *clusters* en fonction du paramètre k, le degré moyen du réseau est fixé à 6 et le paramètre k varie de 1 à 10. Pour chaque valeur du paramètre k, nous considérons des réseaux de taille variant de 100 à 1000 nœuds. Les figures 4.5(b), 4.5(d) et 4.5(f) représentent respectivement le pourcentage de *clusters* construits suivant le critère de l'identité maximale, du degré maximal et de l'énergie résiduelle en fonction du paramètre k et de la taille du réseau.

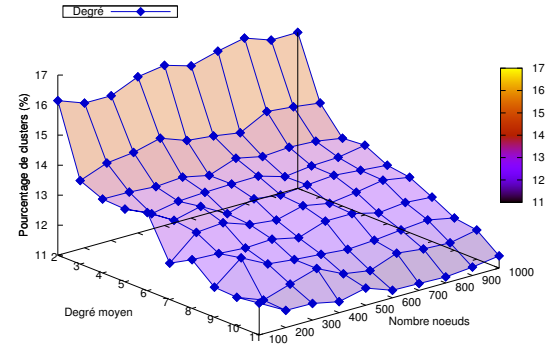
Comme nous pouvions l'imaginer, les résultats de la figure 4.5 montrent que lorsque le paramètre k ou le degré moyen du réseau augmente, le pourcentage de *clusters* construits



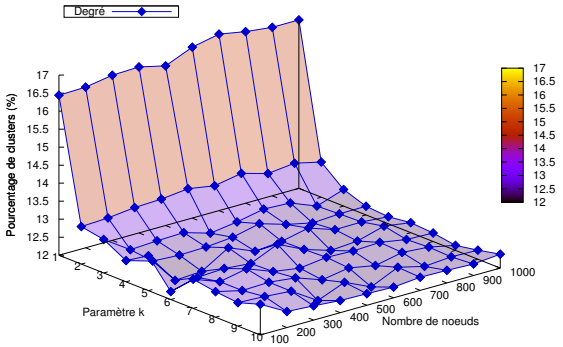
(a) Pourcentage de *clusters* du critère de l'identité maximale en fonction du degré moyen du réseau



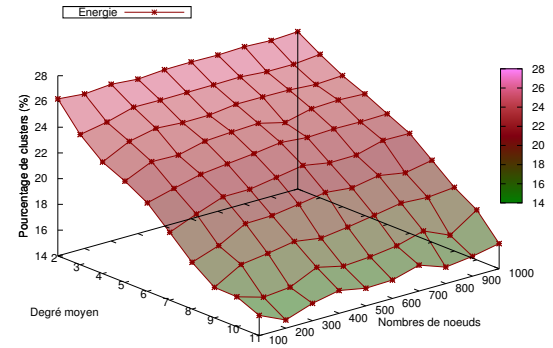
(b) Pourcentage de *clusters* du critère de l'identité maximale en fonction du paramètre k



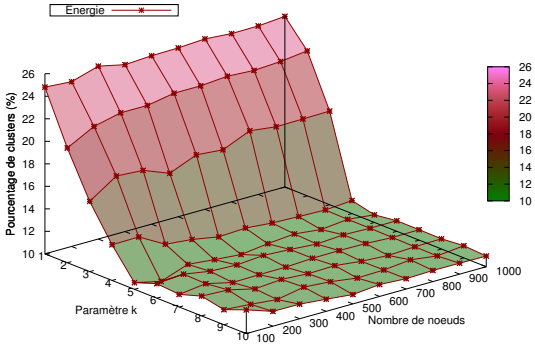
(c) Pourcentage de *clusters* du critère du degré maximal en fonction degré moyen du réseau



(d) Pourcentage de *clusters* du critère du degré maximal en fonction du paramètre k



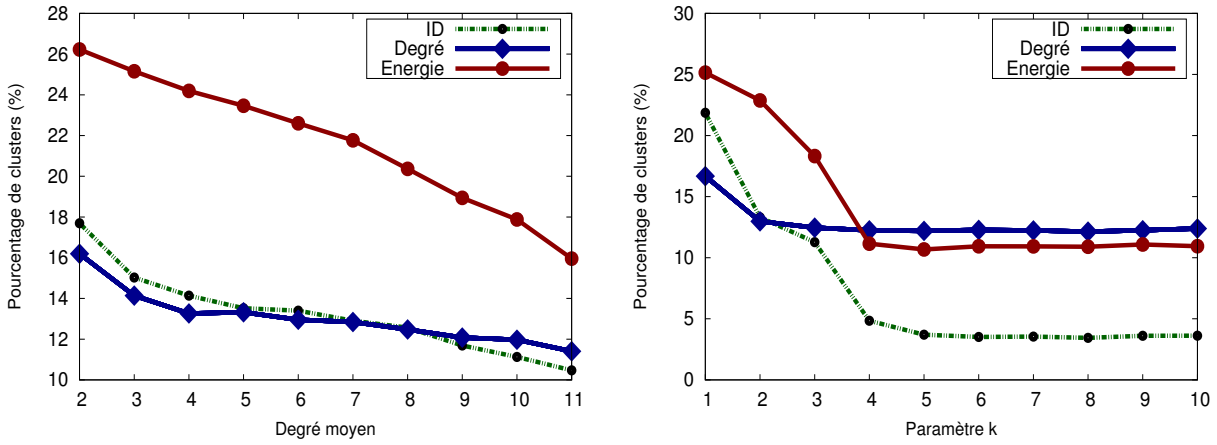
(e) Pourcentage de *clusters* du critère de l'énergie résiduelle en fonction du degré moyen du réseau



(f) Pourcentage de *clusters* du critère de l'énergie résiduelle en fonction du paramètre k

FIGURE 4.5 – Pourcentage de *clusters* construits par chaque critère d'élection des *cluster-heads* en fonction de la taille du réseau, du degré moyen et du paramètre k

décroit. Ce qui est logique car plus le degré moyen du réseau augmente, plus nous tendons vers des topologies réseaux correspondants à des graphes complets. Quand le paramètre k augmente, le diamètre des *clusters* augmente également et provoque une diminution de leur nombre. Par contre, nous constatons que lorsque la taille du réseau augmente, le pourcentage de *clusters* reste relativement constant quelque soit la valeur considérée pour le paramètre k et le degré moyen du réseau. Ce qui montre le passage à l'échelle de SDEAC en termes de pourcentage de *clusters* construits. Il serait donc intéressant d'analyser de plus prés le pourcentage de *clusters* obtenu par rapport aux critères d'élection. Dans ce sens, comme illustré dans la figure 4.6, pour un réseau 1000 nœuds, nous montrons le pourcentage de *clusters* construits par chaque critère d'élection.



(a) En fonction du degré moyen du réseau (b) En fonction du paramètre k

FIGURE 4.6 – Pourcentage de *clusters* construits sur un réseau de 1000 nœuds

Nous pouvons observer que, suivant le degré moyen du réseau, SDEAC construit un pourcentage de *clusters* de $10,47\%$ à $17,67\%$ pour le critère de l'identité maximale, de $11,41\%$ à $16,20\%$ pour le critère du degré maximal et de 14% à 26% pour le critère de l'énergie résiduelle. Pour un meilleur dimensionnement du réseau évitant ainsi une surcharge des *cluster-heads* et afin d'éviter de construire un faible nombre de *clusters* très denses ou un nombre conséquent de *clusters* de petite taille, plusieurs études préconisent un nombre optimal de *cluster-heads* compris entre $15,95\%$ et $26,23\%$ du nombre total de nœuds dans le réseau [YF04, CS10, AVX+12].

S'agissant du paramètre k, nous pouvons alors remarquer que les valeurs qui cadrent au mieux avec cette recommandation sont 2 et 3. En effet, pour l'exemple d'un réseau de 1000 nœuds et pour des valeurs de k de 2 et 3, SDEAC fournit un taux de *cluster-heads* respectivement de $12,98\%$ et $12,46\%$ pour le critère de l'identité maximale, de $12,97\%$ et $12,46\%$ pour le critère du degré maximal et de $22,88\%$ et $18,32\%$ pour le critère de l'énergie résiduelle.

4.3 Impact énergétique de pannes transitoires

Après avoir proposé une généralisation du critère d'élection des *cluster-heads*, nous nous intéressons à l'impact de pannes transitoires sur la consommation énergétique du réseau en cas

de reconstruction des *clusters*. Pour ce faire, nous considérons le critère de l'identité maximale étant donné qu'il procure la meilleure stabilité durant la phase de *clustering* et présente la consommation énergétique la plus basse. Donc, nous reprenons les mêmes expériences que nous avons menées dans la Section 3.6.5 du Chapitre 3 mais en mesurant cette fois-ci la consommation énergétique totale pour corriger les pannes transitoires.

4.3.1 Cas de la disparition de 1, 3 ou 5 nœuds

Nous commençons l'évaluation de l'impact de pannes transitoires sur la consommation énergétique du réseau en considérant un réseau de taille variant de 100 à 1000 nœuds où le degré moyen est fixé à 6 et le paramètre k à 2. Ensuite, nous exécutons SDEAC jusqu'à la formation de *clusters* stables dans tout le réseau. Puis, nous provoquons des pannes transitoires par modification topologique en faisant disparaitre aléatoirement 1, 3 et 5 nœuds du réseau. Ainsi, nous mesurons la consommation énergétique nécessaire pour la reconstruction de *clusters* stables.

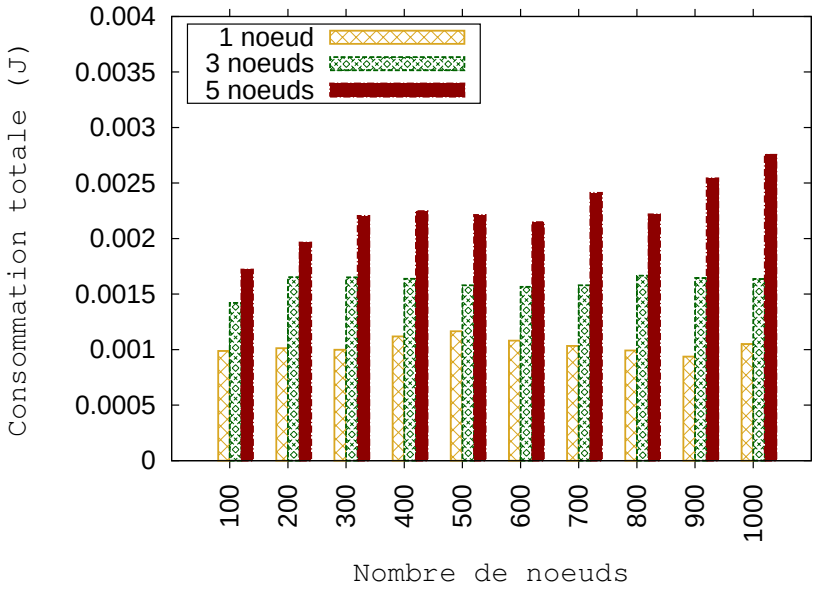


FIGURE 4.7 – Disparition de 1, 3, et 5 nœuds : consommation énergétique supplémentaire

La figure 4.7 illustre la consommation énergétique supplémentaire requise pour corriger les disparitions de 1, 3 et 5 nœuds du réseau. Nous constatons que, pour chaque nœud disparu, la consommation énergétique supplémentaire provoquée par la reconstruction des *clusters* varie peu en suivant l'évolution de la taille du réseau. En effet, comme nous l'avons montré avec les résultats de la figure 3.24 (voir Page 102), le nombre de transitions supplémentaires pour corriger une panne transitoire est moindre. Donc, un nombre moindre de messages est échangé et par conséquence une faible consommation énergétique.

4.3.2 Cas de la disparition de 1% à 5% de nœuds

Pour mieux étudier l'impact énergétique de pannes transitoires, nous considérons un réseau de 1000 nœuds avec un degré moyen fixé à 6 et le paramètre k à 2. Nous exécutons SDEAC jusqu'à la formation de *clusters* stables dans tout le réseau. Ensuite, nous provoquons des pannes transitoires en faisant disparaitre aléatoirement 1% à 5% des nœuds du réseau et nous mesurons la consommation énergétique nécessaire pour la reconstruction de *clusters* stables.

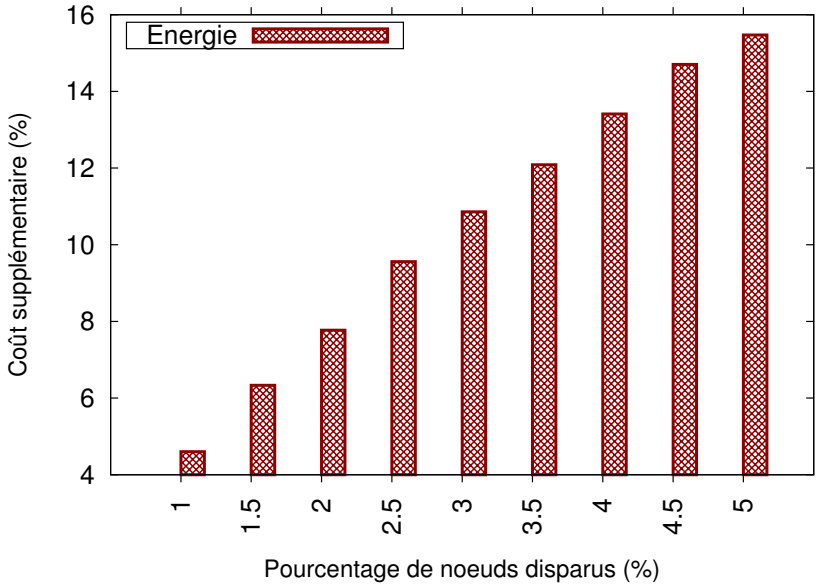


FIGURE 4.8 – Impact de la disparition 1% à 5% de nœuds : coût supplémentaire en énergie

Dans la figure 4.8, nous avons calculé, pour chaque pourcentage de nœuds disparus, le coût supplémentaire dans la consommation énergétique induit par la procédure de *re-clustering* comparé à celle du *clustering*. Nous observons que le coût supplémentaire requis pour retrouver un état stable dans le réseau croit linéairement avec l'évolution du pourcentage de nœuds disparus. Néanmoins, les résultats montrent que la disparition jusqu'à 5% des nœuds du réseau ne conduit environ qu'à $15,5\%$ de consommation énergétique supplémentaire. La raison principale est que, comme nous l'avons montré avec les résultats de la Section 3.6.5 du Chapitre 3, même avec la disparition de 5% des nœuds du réseau, le temps de stabilisation lors du *re-clustering* est minimal comparé à celui du *clustering*. Ceci atteste que, pour des topologies réseaux quelconques, les pannes transitoires n'entrainent pas une reconstruction de la totalité du réseau. En effet, les nœuds réutilisent les informations déjà présentes dans leurs tables de voisinages durant les phases de *re-clustering*. Ce qui réduit les échanges de messages nécessaires pour corriger les pannes transitoires.

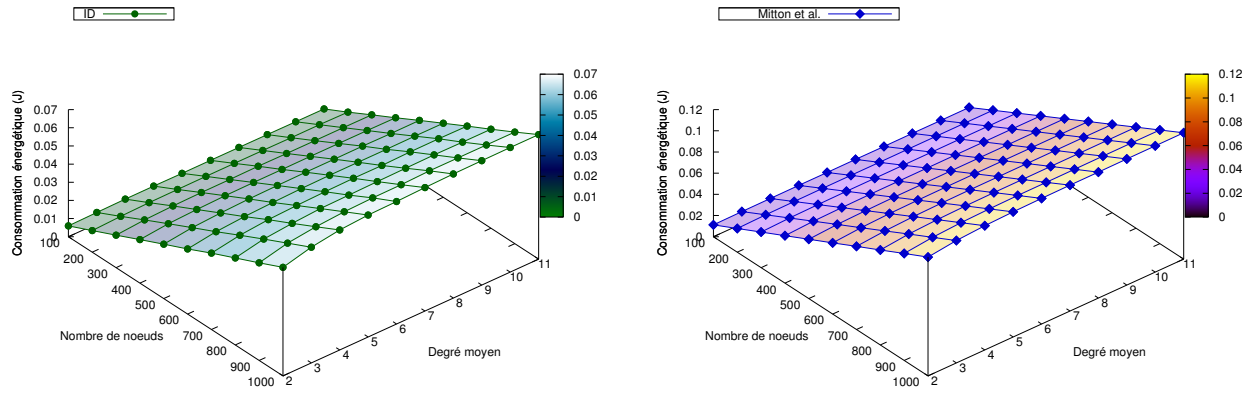
4.4 Comparaison : SDEAC vs Mitton et *al.* [MFGLT05]

Maintenant, nous comparons la consommation énergétique de SDEAC et son pourcentage de *clusters* construits avec ceux de la solution de Mitton et *al.* [MFGLT05] qui opère dans le

même modèle que la notre c'est-à-dire auto-stabilisante, déterministe, distribuée, utilisant un modèle asynchrone à passage de messages et construisant des *clusters* à k sauts. Tout comme SDEAC, nous implémentons l'algorithme de Mitton et *al.* dans OMNeT++. Nous effectuons les comparaisons sur des topologies réseaux quelconques générés suivant une loi de Poisson comme celles utilisées par Mitton et *al.* dans [MFGLT05]. De plus, pour mesurer la consommation énergétique dans les deux approches, nous utilisons le modèle énergétique Heinzelman et *al.* [HCB00]. Dans les simulations, nous considérons des réseaux de taille allant de 100 à 1000 nœuds. Pour chaque taille de réseau, nous fixons le paramètre k à 2 et nous faisons varier le degré moyen du réseau de 2 à 11.

4.4.1 Consommation énergétique

Nous comparons les consommations énergétiques de SDEAC et celle de Mitton et *al.* qui sont illustrées respectivement dans la figure 4.9(a) et la figure 4.9(b). Nous constatons que la consommation énergique totale de la solution de Mitton et *al.* présente la même tendance déjà observée et expliquée lors de la mesure de la consommation énergétique du critère de l'identité maximale des *cluster-heads* de SDEAC (voir Page 112). Pour chaque valeur du degré moyen fixée, la consommation énergétique totale nécessaire à la formation de *clusters* stables croit linéairement avec l'évolution de la taille du réseau. Pour chaque taille de réseau fixée, la consommation énergétique varie peu avec l'augmentation du degré moyen du réseau.



(a) SDEAC

(b) Mitton et *al.*

FIGURE 4.9 – Consommation énergétique totale : SDEAC vs Mitton et *al.*

Ainsi, d'après les résultats illustrés dans les figures 4.9(a) et 4.9(b), nous pouvons noter que SDEAC comme la solution de Mitton et *al.* permettent le passage à l'échelle en termes de consommation énergétique.

Gain apporté par SDEAC dans la consommation énergétique

Pour mieux faire ressortir la réduction apportée par SDEAC dans la consommation énergétique, nous calculons le gain, noté $\mathcal{G}_\mathcal{E}$, de SDEAC par rapport à l'approche de Mitton et *al.*. Pour chaque taille de réseau et pour chaque valeur du degré moyen, $\mathcal{G}_\mathcal{E}$ est calculé comme suit :

$$\mathcal{G}_{\mathcal{E}} = \frac{\mathcal{E}_{totale}(Mitton) - \mathcal{E}_{totale}(SDEAC)}{\mathcal{E}_{totale}(Mitton)} \times 100 \tag{4.3}$$

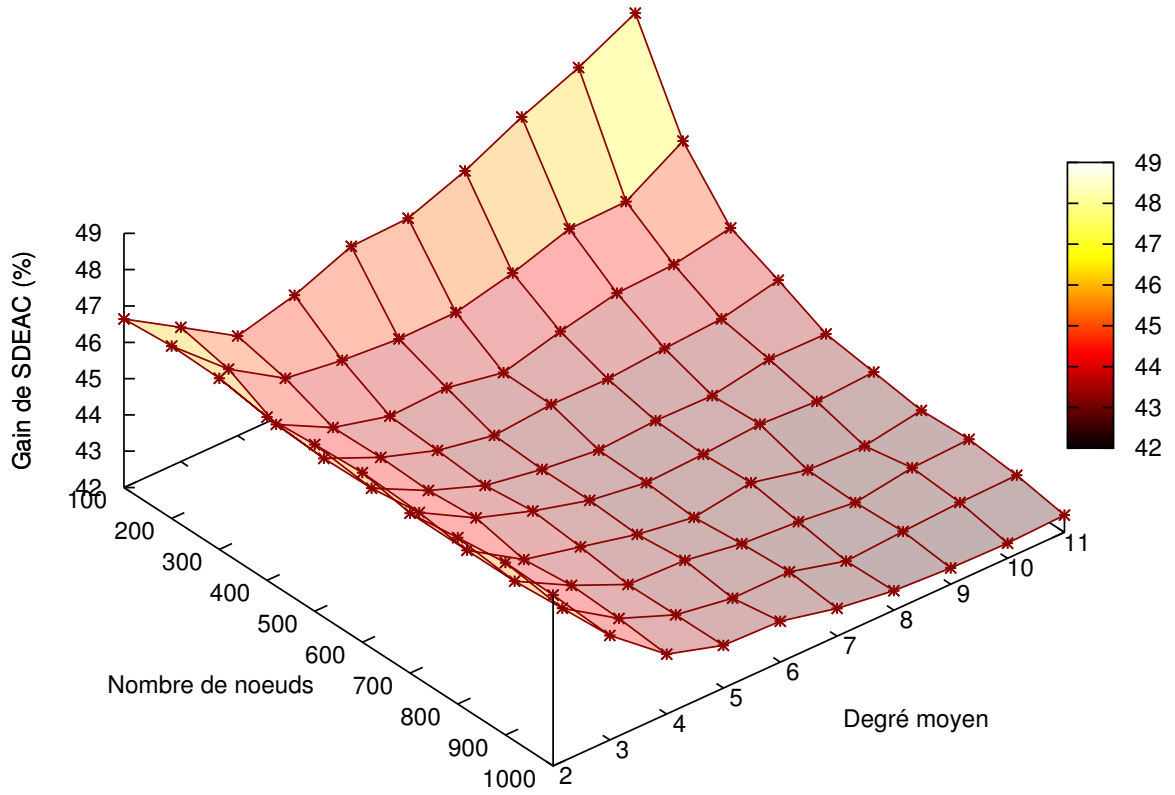


FIGURE 4.10 – Gain apporté par SDEAC dans la consommation énergétique

D'après la figure 4.10, nous observons que SDEAC réduit la consommation énergétique entre 42% et 49% par rapport à la solution de Mitton et *al.*. En effet, dans [MFGLT05], Mitton et *al.* utilisent la *k-density* comme critère d'élection des *cluster-heads*. Pour chaque nœud u du réseau, la *k-density* est calculée à partir de l'équation 2.10 telle que nous l'avons présenté dans la Section 2.4.2 du Chapitre 2 (voir Page 63). Elle représente le nombre de liens de communication dans un voisinage à distance k sauts divisé par le nombre de voisins à distance k sauts. Pour effectuer ce calcul, chaque nœud a besoin d'avoir une vue sur son voisinage à distance $k+1$ sauts ($\{k+1\}$-*Neighborhood*). Ensuite, chaque nœud diffuse sa *k-density* calculée dans son voisinage et la procédure d'élection des *cluster-heads* démarre. L'exemple de la figure 4.11 illustre la 2-*density* du nœud 11 avec sa 3-*Neighborhood*. Le nœud 11 a 5 voisins (nœuds 1, 2, 4, 7 et 9) situés à distance de 2 sauts. Pour connaitre toutes les connexions existantes entre lui et ses 5 voisins, il se fonde sur sa 3-*Neighborhood*. Comme il existe 8 liens de communications entre eux matérialisés par les arêtes en gras ((1,11) ;(1,4) ;(4 ;11) ;(4,7) ;(7,11) ;(2,11) ;(2,9) ;(7,9)), alors sa 2-*density* est égale à 1,6. Le calcul de la *k-density* qui nécessite une $\{k+1\}$-*Neighborhood view*

est très couteux en termes d'échanges de messages. Il faut noter que dans SDEAC, comme nous l'avons illustré dans le Chapitre 3, les nœuds se fondent uniquement sur l'information des voisins directs (voisinage à distance 1) pour structurer le réseau en *clusters* à k sauts.

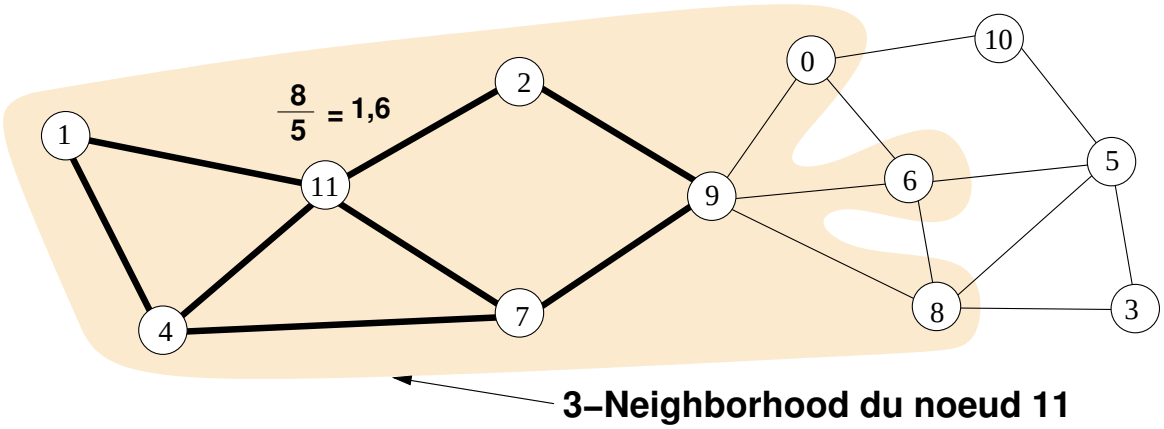
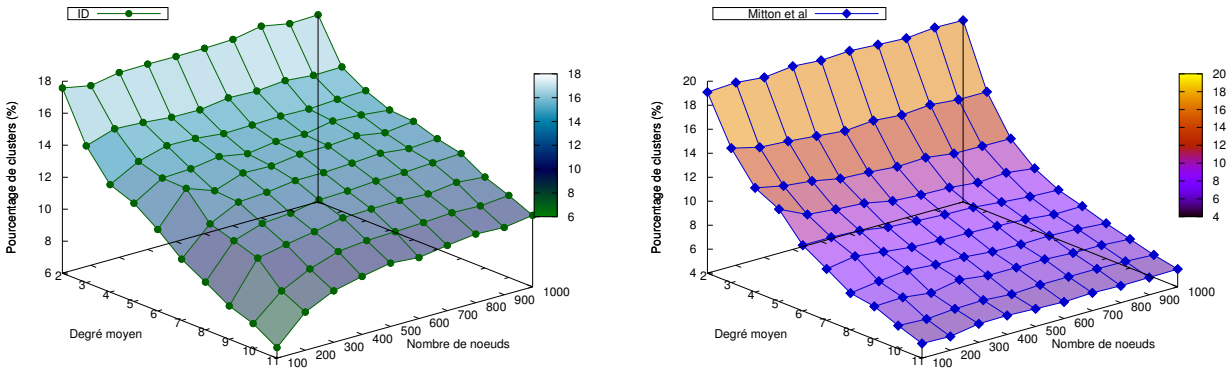


FIGURE 4.11 – Exemple de calcul de la 2-*density* de Mitton et *al.*

4.4.2 Pourcentage de *clusters*

Après avoir comparé les consommations énergétiques des deux approches, nous nous penchons maintenant sur le pourcentage de *clusters* qu'elles construisent. Avec les résultats présentés dans la figure 4.12, le pourcentage de SDEAC est donné dans la figure 4.12(a) et celui de la solution de Mitton et *al.* est illustré dans la figure 4.12(b).



(a) SDEAC (b) Mitton et *al.*

FIGURE 4.12 – Pourcentage de clusters de SDEAC et Mitton et *al.*

D'abord, nous observons que dans les deux approches, pour chaque valeur du degré moyen du réseau fixé, le pourcentage de *clusters* varie peu avec l'augmentation de la taille du réseau.

Ceci atteste que les deux approches permettent le passage à l'échelle. Cependant, pour chaque taille de réseau fixée, nous notons une baisse du pourcentage de *clusters* avec l'augmentation du degré moyen du réseau. En effet, la densité du réseau suit l'augmentation du degré moyen du réseau. Ainsi, les *clusters* deviennent plus denses. D'où une diminution du nombre de *clusters*.

S'agissant des pourcentages de *clusters* construits, nous notons que SDEAC fournit en moyenne un pourcentage de *clusters* entre 6% et 18% et l'algorithme de Mitton et *al.* donne un pourcentage entre 4% et 20% ce qui cadrent bien avec le pourcentage de *clusters* recommandé entre 5% et 20% [YF04, CS10, AVX+12].

Remarque 4.1.
Nous avons effectué les comparaisons en fixant le paramètre k à 2 qui est l'une des valeurs qui donnent les pourcentages de clusters les plus intéressants. Cependant, nous pouvons remarquer qu'avec l'augmentation de la valeur du paramètre k, la zone de diffusion de la solution de Mitton et al. qui se fonde sur un voisinage à distance $k+1$ sauts augmente. Or, l'augmentation du voisinage entraine plus d'échanges de messages conduisant ainsi à une augmentation de la consommation énergétique totale dans le réseau. Notons que, quelle que soit la valeur du paramètre k, SDEAC utilise toujours l'information sur le voisinage à distance 1 pour construire des clusters à k sauts.

4.5 Conclusion

Dans ce chapitre, nous avons étudié la consommation énergétique de SDEAC dans le contexte des RCSF en fonction de trois critères d'élection des *cluster-heads* que sont l'identité maximale, le degré maximal et l'énergie résiduelle des nœuds. Les résultats de simulation ont montré que les trois critères d'élection permettent le passage à l'échelle. Cependant, le critère d'identité maximale, du fait de sa simplicité d'utilisation et de sa stabilité lors de la phase de *clustering*, présente la consommation énergétique la plus basse suivit du critère du degré maximal puis de l'énergie résiduelle des nœuds. Nous avons aussi évalué l'impact de pannes transitoires dans la consommation énergétique du réseau. Les résultats ont montré que la consommation énergétique nécessaire pour corriger les pannes transitoires est très inférieure comparé à celle du *clustering*.

Nous avons également mené une étude comparative des coûts et performances entre SDEAC avec la solution de Mitton et *al.* proposée dans [MFGLT05] qui opère dans le même modèle c'est-à-dire auto-stabilisante, déterministe, distribuée, fondée sur un modèle asynchrone à passage de messages et construisant des *clusters* à k sauts. Les résultats de comparaison ont montré que SDEAC comme la solution de Mitton et *al.* permettent le passage à l'échelle dans la consommation énergétique. Cependant, SDEAC offre l'avantage de réduire la consommation énergétique de 42% à 49% comparé à la solution de Mitton et *al.* [MFGLT05].

Les travaux que nous avons présentés dans ce chapitre ont fait l'objet de plusieurs publications dans des revues et conférences avec actes et comités de sélection : une (1) revue internationale [BFM+13d], deux (2) articles en conférences internationales [BFM+13a, BFM+13c] dont un **Best paper Award** dans [BFM+13c] et un (1) article en conférence francophone [BFM+13b].

Dans le prochain chapitre, nous présentons trois approches qui permettent d'utiliser les *clusters* construits à l'aide de SDEAC pour l'acheminement de l'information dans un RCSF.

Chapitre 5

Utilisations de SDEAC pour acheminer l'information dans les réseaux capteurs sans fil

Résumé. *Dans ce chapitre, partant des clusters construits à l'aide de SDEAC, nous présentons trois scénarios de routage pour l'acheminement de l'information depuis un nœud source jusqu'à la station de base. Ces trois scénarios sont le Routage Sans Agrégation (RSA) pour un transfert de l'information sans attente, le Routage avec Agrégation Partielle (RAP) pour une agrégation des données de chaque cluster et le Routage avec Agrégation Totale (RAT) pour une agrégation d'un maximum d'information par un questionnement du voisinage avant transfert. Les résultats que nous présentons dans ce chapitre ont fait l'objet d'une publication en conférence à audience internationale [BFM+14].*

Sommaire

5.1 Motivations et objectifs

Dans les RCSF, pour économiser l'énergie des capteurs, il est nécessaire de limiter les échanges de messages dans le réseau [PK00, SDT07, ACDFP09, KEAC14]. Or, le routage de l'information depuis un nœud source jusqu'à la BS nécessite la transmission de messages à travers le réseau. Pour réduire les messages qui transitent dans le réseau, une solution consiste à agréger les données lors de leurs acheminements [AYY05, FRWZ07, WAP14]. Cependant, l'agrégation de données nécessite une attente au niveau des nœuds pour recevoir les informations venant de leurs voisins. Cette attente dure un temps prédéterminé et implique une augmentation du délai de bout en bout. Par contre, l'agrégation permet de limiter le nombre de messages transmis et donc d'économiser l'énergie des capteurs. Devant cette situation, nous proposons trois scénarios de routage avec différent niveau d'agrégation. Ces trois scénarios sont : *(i)* le Routage Sans Agrégation (RSA), *(ii)* le Routage avec Agrégation Partielle (RAP) et *(iii)* le Routage avec Agrégation Totale (RAT).

Dans le scénario RSA, les capteurs envoient les données collectées à leur *cluster-head*. Puis, ces derniers routent automatiquement toute donnée reçue vers la BS. Cette technique de routage a pour but d'offrir un acheminement rapide des données.

Avec le scénario RAP, les *cluster-heads* collectent toutes les données captées par les nœuds de leurs *clusters* et les fusionnent en un seul agrégat. Ainsi, seuls les agrégats sont routés depuis les *cluster-heads* jusqu'à la BS. Cette approche permet de réduire les communications et les consommations énergétiques dans le but de prolonger la durée du réseau.

Pour le troisième scénario RAT, nous optons pour une technique où l'agrégation se fait au niveau de chaque nœud en mettant en œuvre une approche d'agrégation de données fondée sur des agents coopératifs. Pour cela, nous intégrons les travaux sur l'agrégation de données dans les RCSF à base d'agents coopératifs proposés par Merghem-Boulahia et *al.*[1] [SRAMBG09a, SRAMBG09c, SRAMBG09b, SMMB$^+$10, Sar10]. Initialement, Merghem-Boulahia et *al.* ont proposé une approche d'agrégation de données pour un RCSF non structuré. Ainsi, en adaptant cette technique sur un RCSF structuré, nous mettons en œuvre une approche d'agrégation totale et décentralisée qui se fonde sur un Système Multi-Agents (SMA) (*Multi-Agent Systems* (MAS)). L'objectif est de permettre aux nœuds d'utiliser un mécanisme de coopération piloté par un agent implémenté dans sa "couche application" pour agréger ses données avec celles de ses voisins lors des phases d'acheminement des données. En d'autres termes, le RAT permet l'acheminement d'un maximum de données agrégées après consultation du voisinage.

Pour mettre en œuvre les trois scénarios de routage que nous proposons, il est nécessaire de construire les routes depuis tous les nœuds du réseau jusqu'à la BS. Pour cela, nous proposons une adaptation et une optimisation du protocole *Dynamic Source Routing* (DSR) proposé par

1. Nos travaux de recherche et ceux de Merghem-Boulahia et *al.* effectués au sein de l'équipe ICD-ERA (UMR CNRS 6279) de l'Université de Technologies de Troyes (UTT) s'inscrivent dans le cadre du projet régional CPER CapSec ROFICA. Ce dernier chapitre est donc le lieu de faire la jonction des travaux réalisés dans le cadre du projet CPER CapSec ROFICA.

Johnson et *al.* [JMB01] et normalisé par IETF [2] (RFC 4728 [3]). Notre choix est motivé d'une part par la simplicité de DSR et d'autre part, par les travaux de Merghem-Boulahia et *al.*, où ce même protocole est utilisé pour la construction des routes.

La suite de ce chapitre est organisée comme suit. Dans la Section 5.2, nous faisons un rappel sur les principaux protocoles de routages normalisés par l'IETF puis nous présentons notre mécanisme de construction des routes. Ensuite, dans la Section 5.3, nous décrivons l'approche d'agrégation décentralisée à base d'agents coopératifs que nous mettons en œuvre dans un réseau structuré en *clusters*. Puis, dans la Section 5.4, nous détaillons les trois scénarios de routage proposés pour l'acheminement de l'information. Nous présentons une évaluation des coûts et performances des trois techniques de routage dans la Section 5.5. Enfin, nous résumons les travaux présentés dans ce chapitre au niveau de la Section 5.6.

5.2 Approche de routage pour la construction des routes

Dans cette section, nous faisons un aperçu sur les principaux protocoles de routage normalisés par l'IETF puis nous présentons la démarche que nous allons adopter pour la construction des routes.

5.2.1 Vue d'ensemble sur les protocoles de routage standards

Il existe plusieurs protocoles de routage dans la littérature. Nous pouvons les classer en trois catégories : proactif, réactif et hybride.

Dans un protocole de routage proactif, les routes sont créées à l'avance sur la base d'échanges périodiques de table de routage. Chaque nœud maintient à jour les informations de routage concernant tous les autres nœuds du réseau [TSV11, PV10]. Du fait de l'aspect dynamique de la topologie des réseaux ad hoc, la maintenance des tables de routage nécessite l'envoie périodique par chaque nœud d'un message de signalisation indiquant sa présence à tous ses voisins. L'idée majeure est de conserver dans chaque nœud des informations de routage vers tous les autres nœuds du réseau pour accélérer le routage des paquets par la suite. Les changements topologiques du réseau sont gérés par propagation à chaque voisin des mises à jour des routes, afin que chacun puisse maintenir une vue consistante du réseau. Parmi les protocoles de routage proactifs, nous pouvons citer *Optimized Link State Routing Protocol* (OLSR) [JMC+01] RFC 3626 [4], *The Babel Routing Protocol* (BRP) RFC 6126 [5] etc.

Avec un protocole de routage réactif, les routes sont créées à la demande. Ce type de protocole ne garde que les routes en cours d'utilisation pour le routage. A la demande, le protocole va chercher une route pour atteindre une destination. Ainsi, lorsqu'un nœud souhaite communiquer avec une station distante, il est obligé de déterminer une route dynamiquement.

2. The Internet Engineering Task Force. http ://www.ietf.org
3. http ://tools.ietf.org/html/rfc4728
4. http ://tools.ietf.org/html/rfc3626
5. http ://tools.ietf.org/html/rfc6126

Cette technique permet de ne pas inonder le réseau par des paquets de contrôle et de ne pas conserver les routes non utilisées. Mais elle nécessite en contrepartie un certain temps pour établir une route avant de pouvoir la transmettre. Parmi les protocoles de routage réactifs, nous pouvons citer *Dynamic Source Routing* (DSR) [JMB01], *Ad hoc On-demand Distance Vector* (AODV) [PR99] RFC 3561 [6] etc.

Il existe aussi des protocoles hybrides qui combinent les deux approches précédentes afin de tirer les avantages de chacune d'elle tout en réduisant leurs inconvénients. Ils utilisent un protocole proactif pour apprendre le proche voisinage (voisinage à deux ou trois sauts) et un protocole réactif pour atteindre les nœuds situés au-delà de cette zone de voisinage prédéfinie. Comme protocoles hybrides, nous pouvons citer *Zone Routing Protocol* (ZRP) [Haa97] [7], *Cluster Based Routing Protocol* (CBRP) [8] etc.

5.2.2 Approche de construction des routes

Dans cette section, nous présentons l'algorithme de construction des routes qui sont utilisées lors de l'acheminement des données collectées dans le réseau.

Utilisation du protocole DSR dans un réseau structuré en *clusters*

Pour construire les différentes routes depuis tous les nœuds jusqu'à la BS, nous utilisons le principe du protocole standard nommé DSR [JMB01]. DSR se fonde sur une technique d'inondation pour découvrir l'itinéraire d'une source vers une destination. Or, l'inondation peut être coûteuse en termes d'échanges de messages. Nous proposons donc d'adapter DSR dans nos structures en *clusters* afin de réduire les messages lors de la phase de construction des routes. L'approche que nous mettons en œuvre fonctionne en deux phases : (1) une phase de *découverte des routes* durant laquelle les nœuds capteurs obtiennent des informations sur leurs voisinages ainsi que leurs nœuds de passage (i.e. le premier voisin direct d'un nœud sur son chemin vers la BS) 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 des routes.

Découverte des routes

Dans le but de découvrir les routes qui mènent vers la BS tout en limitant les échanges de messages, seuls les *cluster-heads*, une fois qu'ils se considèrent stables (cf. Définition 3.5 au niveau de la Section 3.2.4 du Chapitre 3), initient la construction des routes en envoyant un message de découverte des routes vers la BS (cf. Figure 5.1). Ces messages vont être retransmis à destination de la BS et vont récolter dans un vecteur toutes les identités de nœuds sur le chemin. La BS maintient une table de routes contenant toutes les routes vers tous les *cluster-heads* du réseau. Dans la figure 5.1, la table $Route_{BS}$ illustre un exemple de quelques routes entre la BS et chaque *cluster-head* du réseau.

6. http ://tools.ietf.org/html/rfc3561
7. http ://tools.ietf.org/html/draft-ietf-manet-zone-zrp-04
8. http ://tools.ietf.org/html/draft-ietf-manet-cbrp-spec-01

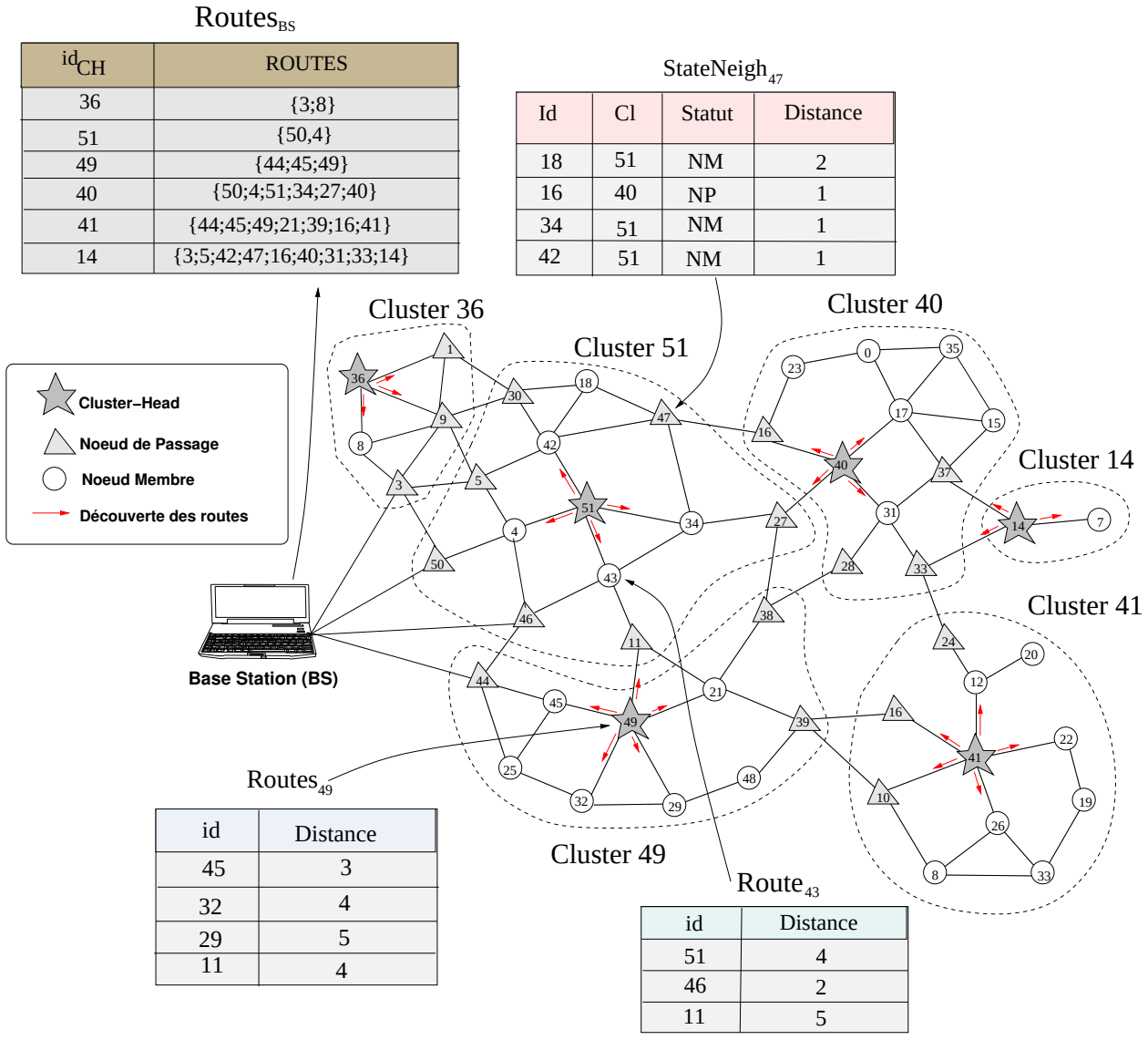


FIGURE 5.1 – Construction des routes

Une fois arrivée au niveau de la BS, les messages de découverte des routes reprennent les chemins inverses vers les *cluster-heads* initiateurs. En récupérant les messages revenus de la BS, chaque *cluster-head* du réseau maintient une table de routes contenant toutes les routes connues vers la BS. Ainsi, pour envoyer les données, les *cluster-heads* pourront par exemple choisir les routes les plus courtes. Dans la figure 5.1, $Routes_{49}$ illustre quelques routes du *cluster-head* 49.

Chaque nœud u qui n'est pas *cluster-head* (c'est-à-dire de statut NM ou NP) connait déjà ses voisins directs et son nœud de passage vers son *cluster-head*. Ces informations sont obtenues lors de la structuration du réseau par SDEAC et sont présentes dans sa table de voisinage $StateNeigh_u$. Dans la figure 5.1, $StateNeigh_{47}$ donne l'exemple de la table de voisinage du nœud 47 appartenant au **Cluster 51**. Ainsi, durant la phase de construction des routes, un nœud de statut NM ou NP recueille les informations sur les nœuds de passage qu'il peut

contacter pour envoyer ses informations en direction de la BS. Se sont les seules informations qui sont collectées par un NM ou NP lors de la construction des routes. Dans la figure 5.1, $Routes_{43}$ montre quelques routes du nœud 43 appartenant au **Cluster 51**.

Mise à jour des routes

Les nœuds mettent à jour leurs tables de routage avec une technique de communication d'informations pro-active de type *push*. Cette opération est déclenchée lorsqu'un capteur n'a plus suffisamment d'énergie pour participer à l'opération de routage. Le seuil d'énergie critique est laissé à l'appréciation de l'administrateur du réseau : une méthode consisterait à de fixer le seuil à 25% du niveau initial. 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 décrit la figure 5.2, la mise à jour des routes se fait au moyen de quatre types de messages :

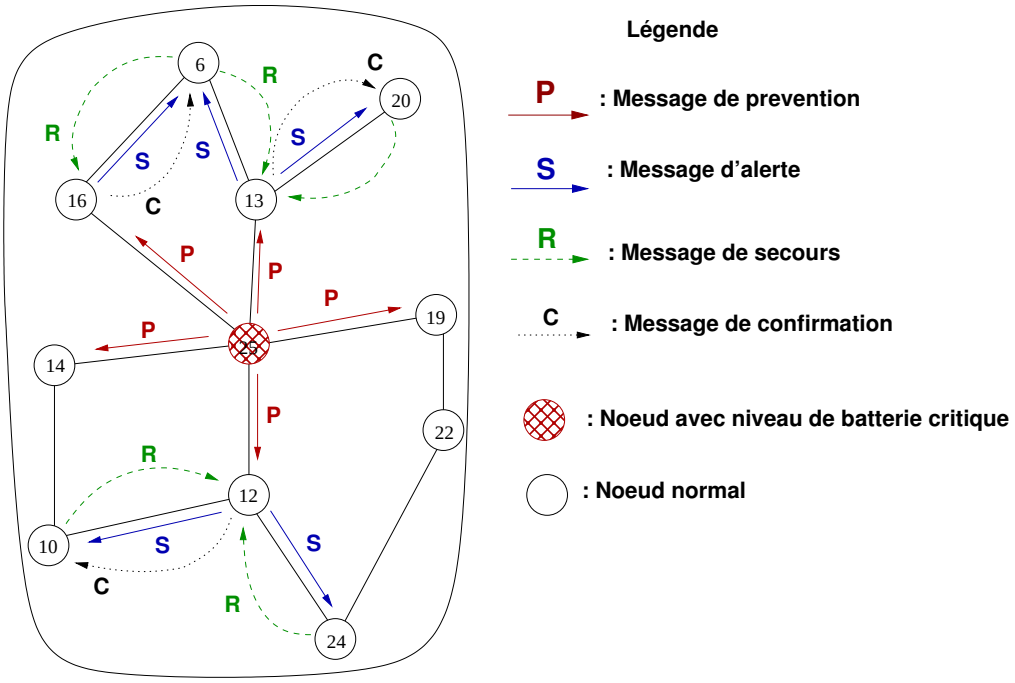


FIGURE 5.2 – Maintenance du voisinage

- Messages de prévention (P) : un 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 *cluster-head* ou la station de base ;
- Messages d'aide (S) : un nœud 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 (R) : lorsqu'un nœud 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 (C) : après la réception de messages d'assistance, le nœud en quête de passerelle choisit le voisin qui a la plus grande quantité d'énergie résiduelle comme

passerelle puis informe celui-ci du changement en lui envoyant 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.

5.3 Agrégation de données à base d'agents coopératifs

En partant du fait que la communication est de loin la première source de consommation d'énergie, nous proposons un traitement local des données avant leur envoi vers la station de base. Pour cela, nous appliquons les travaux de Merghem-Boulahia et *al.* [SRAMBG09a, SRAMBG09c, SRAMBG09b, SMMB$^+$10, Sar10] dans le réseau que nous structurons avec SDEAC. Le but est d'arriver à une approche décentralisée d'agrégation de données se fondant sur un système multi-agents afin d'envoyer un plus gros volume de données vers BS pour une meilleure vue sur la zone de captage et prise de décision.

5.3.1 Scénarios 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 décrite à la section suivante. 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.

5.3.2 Coopération entre agents

Nous définissons 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, la position des nœuds dans le réseau et le degré des nœuds. Ainsi, en fonction de la valeur de ces paramètres, chaque agent calcule un coefficient $\mathcal{R}$ 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 5.1. 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 :

- Énergie résiduelle ($\mathcal{E}$) : 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 $\mathcal{E}_r^t$ à un instant t et l'énergie maximale $\mathcal{E}_i$ du capteur au moment de son déploiement, comme illustré dans cette équation : $\mathcal{E} = \frac{\mathcal{E}_r^t}{\mathcal{E}_i}$;
- Degré ($\mathcal{D}$) : 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 ($\mathcal{P}$) : 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*. Dans ce cas, les nœuds de statut NP ou CH vont être des nœuds critiques. Un nœud est dit de bordure s'il est à "l'extrémité" d'un *cluster*. Tout autre nœud occupe une position "normale". Celui-ci est entouré par plusieurs voisins et ne représente pas une passerelle. Les nœuds normaux ou de bordures sont de statut SN. A titre d'exemple, dans la figure 5.3, les nœuds 46 et 50 du **Cluster 51** sont des nœuds critiques, les 19, 22 et 33 du **Cluster 41** sont des nœuds de bordures et les nœuds 4, 34, 42 et 43 du **Cluster 51** sont des nœuds normaux.

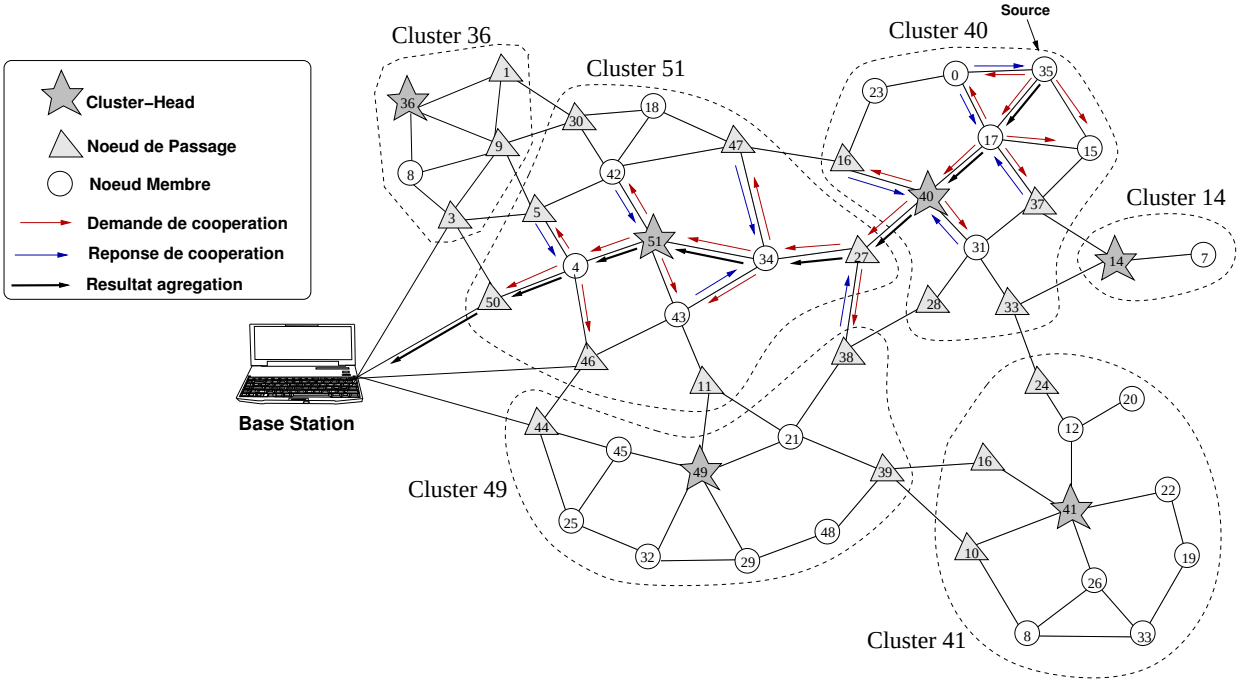


FIGURE 5.3 – Routage avec agrégation de données

- Importance des informations ($\mathcal{I}$) : Ce paramètre dépend surtout du type d'application cible. Une information est jugée importante dans le cas où par exemple 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 $\mathcal{E}$, $\mathcal{D}$, $\mathcal{P}$, $\mathcal{I}$ et de leurs poids respectifs w_e, w_d,w_p, w_i, nous calculons le coefficient de coopération selon l'équation 5.1.

$$\mathcal{R} = \mathcal{E} \times w_e + \frac{1}{\mathcal{D}} \times w_d + \mathcal{P} \times w_p + \mathcal{I} \times w_i \tag{5.1}$$

La figure 5.3 illustre l'exemple d'un nœud source d'identité 35 du **Cluster 40** qui détecte un événement important, demande à ses voisins directs 0, 15 et 17 de coopérer, agrège les données reçues avec les siennes puis envoie l'agrégat qui en résulte à son nœud de passage d'identité 17. Dans cet exemple, le nœud 46 du **cluster 51** décide de ne pas coopérer étant donnée sa position critique (voisin direct de la BS et passerelle) et ses réserves d'énergie qui s'épuisent vite à cause de son emplacement.

5.4 Scénarios d'acheminement de l'information proposés

Après avoir présenté notre mécanisme de construction des routes et décrit notre scénario d'agrégation à base d'un système multi-agent coopératif, nous proposons trois scénarios de routage pour l'acheminement des informations collectées depuis les nœuds du réseau jusqu'à la BS. Ces trois scénarios sont le Routage Sans Agrégation (RSA), le Routage avec Agrégation Partielle (RAP) et le Routage avec Agrégation Totale (RAT).

5.4.1 Routage Sans Agrégation (RSA)

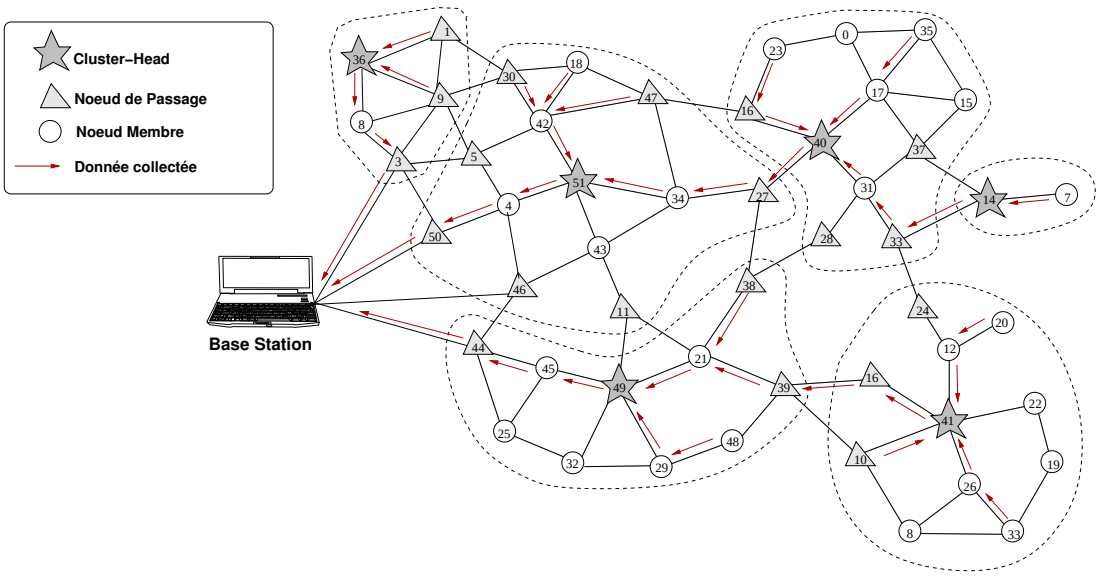


FIGURE 5.4 – Scénario de Routage Sans Agrégation (RSA)

Dans le RSA, comme illustré dans la figure 5.4, les données collectées par les nœuds sont directement envoyées à la BS. Un nœud capteur u qui collecte une donnée l'envoie à destination

de son *cluster-head* en passant par sa passerelle ($np_{(u,CH_u)}$) dans le *cluster*. Tout *cluster-head* qui reçoit un message venant de l'un des nœuds de son *cluster* le transfère directement vers la BS via sa passerelle $np_{(CH,BS)}$. Ainsi, tout nœud v du chemin transfère le message vers la BS via sa passerelle $np_{(v,BS)}$. Cette procédure se répète jusqu'à ce que le message arrive au niveau de la BS.

5.4.2 Routage avec Agrégation Partielle (RAP)

Pour le RAP, comme montré dans la figure 5.5, nous proposons un routage avec une agrégation partielle (agrégation au niveau des *cluster-heads* uniquement). Tout u dans un *cluster* qui collecte une donnée l'envoie vers son *cluster-head* via sa passerelle ($np_{(u,CH_u)}$). Ainsi, chaque *cluster-head* collecte et fusionne toutes les données émanant de son *cluster* en un seul agrégat. Puis, l'agrégat est envoyé par le *cluster-head* en direction de la BS via la passerelle $np_{(CH,BS)}$. Et tout nœud v sur le chemin retransmet à son tour l'agrégat vers la BS via sa passerelle $np_{(v,BS)}$. Cette procédure se répète jusqu'à ce que le message arrive au niveau de la BS.

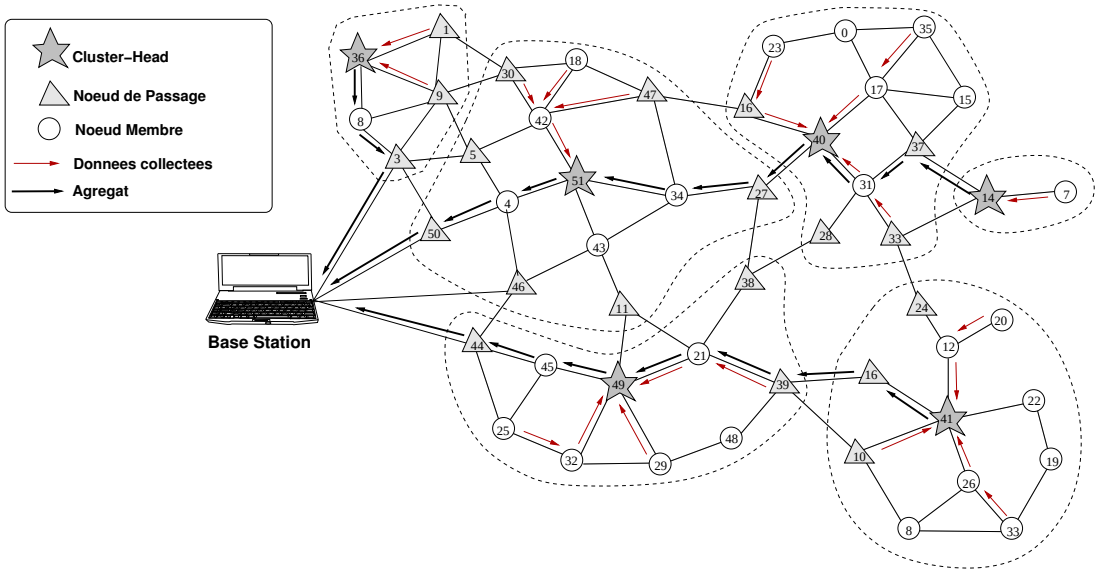


FIGURE 5.5 – Scénario de Routage avec Agrégation Partielle (RAP)

5.4.3 Routage avec Agrégation Totale (RAT)

Pour le troisième scénario de routage, illustré dans la figure 5.6, nous proposons un routage avec une agrégation totale (RAP) en mettant en œuvre le mécanisme d'agrégation de données à base d'agents coopératifs décrit dans la Section 5.3. Comme précédemment décrit, un nœud qui capture une donnée pertinente demande à ses voisins leurs données puis fusionne partiellement toutes les données reçues du voisinage et envoie l'agrégat partiel à destination de la BS. Cette procédure se répète au niveau de chaque nœud le long du chemin jusqu'à ce que le message arrive au niveau de la BS.

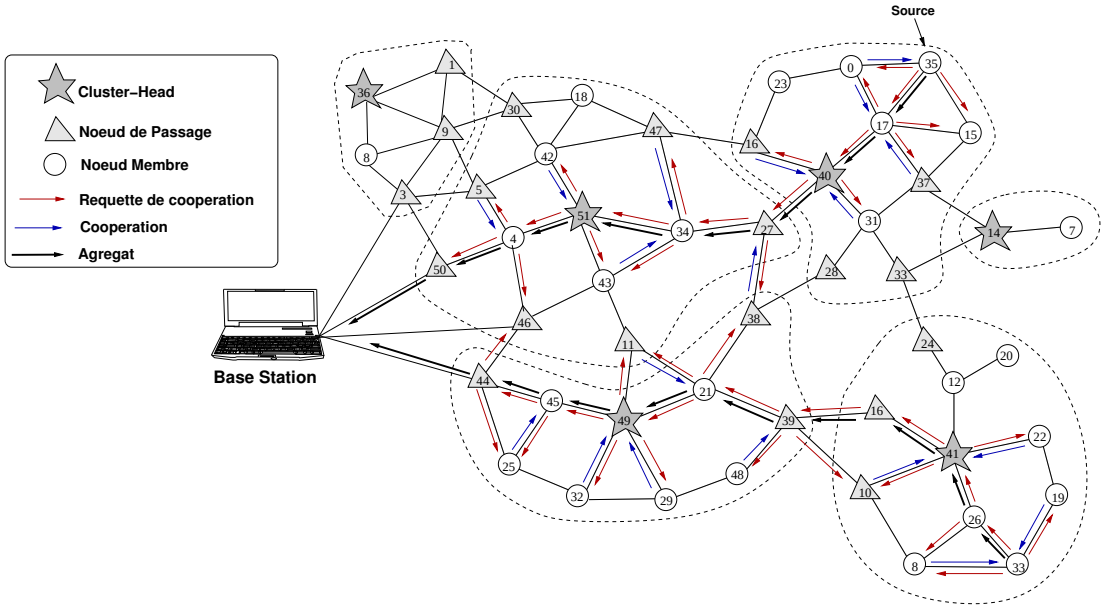


FIGURE 5.6 – Scénario de Routage avec Agrégation Totale (RAT)

5.5 Évaluation de l'acheminement de l'information

Dans cette section, nous présentons une évaluation des performances des trois approches que nous proposons pour l'acheminement de l'information depuis un capteur source jusqu'à la BS. Pour ce faire, tout comme nous avons implémenté SDEAC dans OMNeT++, nous implémentons le mécanisme de construction des routes et les trois scénarios de routage (RSA, RAP et RAT). Ensuite, nous comparons les coûts et performances des trois scénarios de routage.

5.5.1 Métriques d'évaluation

Nous proposons d'évaluer les coûts et performances des trois scénarios de RSA, de RAP et de RAT en mesurant et comparant trois métriques : le délai de bout en bout, la consommation énergétique et la durée de vie du réseau.

Délai de bout en bout

Le délai de bout en bout est défini comme étant le temps d'acheminement de l'information depuis un capteur source jusqu'à la BS [JHRC08, KLSS10, KLS11, BSM12]. Plusieurs facteurs influent sur ce délai. Nous avons le temps de propagation (δ_{prop}) qui est le temps nécessaire pour la transmission du message entre deux nœuds. Le temps de traitement (δ_{trait}) qui représente le temps nécessaire à un nœud pour réceptionner un message (démodulation), faire le traitement requis (récupérer l'adresse de destination et chercher la route dans le table de routage) puis retransmettre (modulation) vers la destination. Et le temps d'agrégation δ_{agreg} représente le temps d'attente pour fusionner les données. En fonction de ces différents facteurs et du nombre de liens de communication $\mathcal{N}$ qui est égal à $\alpha + 1$ avec α le nombre de nœuds traversés par un message, le délai de bout en bout pour chaque message délivré à la BS est calculé comme suit :

$$\Delta T = \mathcal{N} \times (\delta_{prop} + \delta_{trait} + \delta_{agreg}) \tag{5.2}$$

Dans le RSA, δ_{agreg} est nul car aucun nœud du réseau n'effectue d'agrégation. Pour le scénario de RAP, chaque *cluster-head* va attendre δ_{agreg} pour agréger les données de son *cluster*. Et pour le scénario de RAT, tout nœud attend un temps δ_{agreg} pour la coopération avec ses voisins. Les valeurs des paramètres δ_{prop}, δ_{trait} et δ_{agreg} seront données au niveau la Section 5.5.2. Dans chaque scénario, nous supposons que les temps de propagation, de traitement et d'agrégation sont identiques sur tous les nœuds réseau.

Consommation énergétique

Pour la consommation énergétique, nous reprenons la définition et la formule que nous avons présenté au niveau de la Section 4.2.2 du Chapitre 4. La consommation énergétique totale, notée $\mathcal{E}_{totale}$, est définie ici comme étant la quantité totale d'énergie consommée dans tout le réseau durant la période d'observation considérée (temps de simulation).

$$\mathcal{E}_{totale} = \sum_{i=0}^{n-1} \mathcal{M}_i^{env} \times E_{Tx}(l,d) + \sum_{i=0}^{n-1} \mathcal{M}_i^{rec} \times E_{Rx}(l) \tag{5.3}$$

Durée de vie du réseau

La durée de vie du réseau est définie comme étant le *temps qui s'écoule* ou le *nombre de cycles de captages* depuis le déploiement des nœuds c'est-à-dire le démarrage de l'activité du réseau jusqu'au moment où le réseau devient non-fonctionnel [BGC01, CZ05, DD09, PC10]. Le réseau est considéré comme non-fonctionnel selon les spécificités de l'application pour laquelle il a été déployé. Pour certaines applications critiques, le réseau est considéré comme non-fonctionnel dès la mort du premier nœud dans le réseau. Un nœud u est considéré comme mort dès l'épuisement de son énergie ($\mathcal{E}_u^{disp} = 0$). Dans ce cas, le nœud ne peut plus émettre ni réceptionner de messages. Pour d'autres applications, le réseau est considéré comme non-fonctionnel à partir de la mort d'un certain pourcentage de nœuds. Le réseau peut également être considéré comme non-fonctionnel dès la perte de la connexité totale (perte de couverture) c'est-à-dire à partir du moment où la BS n'est plus atteignable depuis n'importe quel nœud du réseau.

Nous évaluons la durée de vie du réseau dans chacun des trois scénarios de routage (RSA, RAP et RAT) en mesurant d'abord le temps qui s'écoule et le nombre de cycles de captages depuis le déploiement des nœuds jusqu'à ce que le premier nœud disparait ($(\mathcal{E}_u^{disp} = 0)$). Ensuite, sans refaire de *re-clustering*, nous évaluons la durée de vie maximale du réseau jusqu'à ce qu'à la perte de connexité totale. L'objectif est d'évaluer le nombre de nœuds i dont l'épuisement totale de la batterie conduit à une perte de connexité totale.

5.5.2 Environnement et paramètres de simulation

De la même manière que les mesures de performances effectuées lors des évaluations de SDEAC dans le Chapitre 3 et le Chapitre 4, nous utilisons le simulateur OMNeT++ pour

l'implémentation et les mesures des coûts et performances des trois scénarios de routage (RSA, RAP et RAT). Le tableau 5.1 résume les différents paramètres de simulations utilisés.

Nous considérons toujours des topologies réseaux suivant une loi de Poisson [BC03, HLS05, MFGLT05, CKF10, NQL11]. Pour les mesures de la consommation énergétique, nous utilisons toujours le modèle énergétique de Heinzelman et *al.* [HCB00]. De même, nous effectuons toutes nos mesures avec un intervalle de confiance de 99%.

	Paramètres	**Valeur**
Constantes	Taille d'un message	2000 bits
	Portée radio	100 mètres
	Modèle de graphe	Distribution de Poisson
	Nombre de simulations pour chaque taille du réseau	100
	Intervalle de confiance	99%
	Paramètre k	2
	Degré moyen du réseau	6
	Intervalle de captage τ_s	1 s
	Délais de propagation δ_{prop}	0.3 μ_s
	Délais de traitement δ_{trait}	15,36 ms
	Délais d'agrégation δ_{agreg}	40 ms
	Poids pour la stratégie des agents (w_e, w_p, w_d, w_i)	$0,25$
Variables	Énergie initiale	{1,2,3} Joules
	Seuil de Pertinence $\mathcal{R}$	$[0,5; 0,99]$
	Nombre de nœuds	[100,1000]
	Temps de simulation $\mathcal{T}$	1000s, 10 jours

TABLE 5.1 – Paramètres de simulation pour l'évaluation des scénarios de routage

Dans le but de considérer un RCSF hétérogène avec des nœuds dotés d'une faible capacité énergétique, nous attribuons à chaque nœud une valeur énergétique initiale prise aléatoirement à 1, 2 ou 3 joules [YP12, LZXG12, GJS+12, GYP13]. Pour étudier le passage à l'échelle, nous considérons des réseaux de taille allant de 100 à 1000 nœuds par pas de 100.

Pour le délai de propagation δ_{prop} et de traitement δ_{trait}, nous utilisons respectivement les valeurs 0.3 μs et 15,36 ms telles que définies dans la norme 802.15.4 [Gro14] et utilisées dans plusieurs travaux [KAT06, MH09, Woo13]. Et pour le délai d'agrégation δ_{agreg}, nous utilisons la valeur de 40 ms telle que utilisée dans la littérature [CKYL06, SRAMBG09a, Sar10].

Dans la stratégie de coopération des agents, nous attribuons équitablement un poids de $0,25$ à chacun des critères pour le calcul du coefficient de pertinence de la coopération. Pour favoriser un ou plusieurs critère (s) au détriment des autres, il est possible de jouer sur la valeur de leurs poids. Nous fixons le seuil de pertinence $\mathcal{R}$ (cf. Section 5.3.2) dans l'intervalle $[0,5; 0,99]$. Dès qu'un agent calcule une valeur de pertinence comprise dans cette intervalle alors il décide de coopérer. Nous fixons cet intervalle assez large dans le but simuler le réseau pour favoriser plus d'échanges possibles et de simuler au mieux des conditions extrêmes de consommation

énergétique. Pour cette même raison, nous fixons également l'intervalle de captage à une (1) seconde.

Nous supposons le scénario suivant dans nos simulations. Nous avons un réseau de capteurs où les nœuds sont équipés d'un capteur de température. Ainsi, durant la période d'observation considérée (temps de simulation), à toutes les secondes (cycle de captage), tous les nœuds recueillent la température environnant et l'envoie à destination de la BS.

5.5.3 Résultats

Dans cette section, nous présentons les résultats des mesures du délai de bout en bout, de la consommation énergétique et de la durée du vie du réseau.

Délais de bout en bout

Nous entamons les évaluations de performances du RSA, RAP et RAT en mesurant leurs délais de bout en bout. Avec les résultats de simulation illustrés au niveau de la figure 5.7, nous considérons un temps de simulation fixé à 1000 secondes, un réseau de taille allant de 100 à 1000 nœuds avec le paramètre k fixé à 2 et le degré moyen du réseau à 6. La figure 5.7 montre le délai moyen de bout en bout du RSA, RAP et RAT.

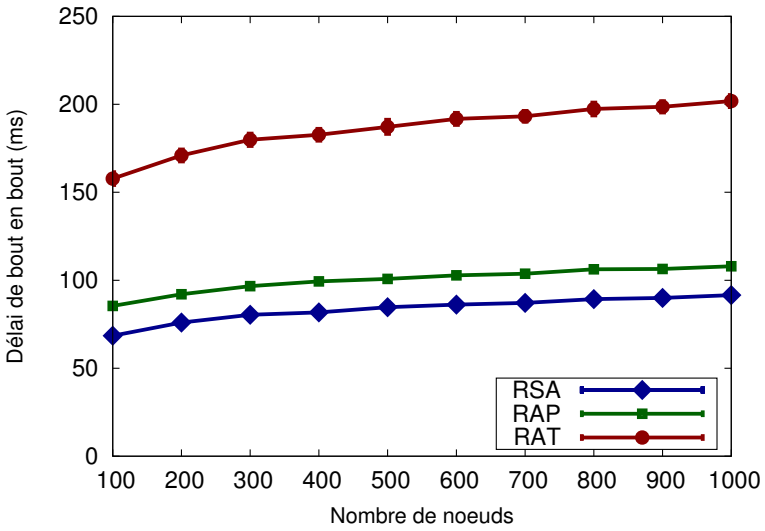


FIGURE 5.7 – Délais moyen de bout en bout

Nous observons, pour chacun des scénarios de routage, une évolution logarithmique du délai de bout en bout en fonction de la taille du réseau. Cela montre que les trois scénarios permettent le passage à l'échelle en termes de délai de bout en bout. Nous remarquons aussi que le RSA présente un meilleur délai de bout en bout suivit du scénario de RAP puis arrive en dernière position le RAT.

Pour mieux évaluer la rapidité du RSA par rapport aux deux autres techniques, nous avons calculé, comme illustré dans la figure 5.8, le gain en termes de délai. Les résultats montrent que le RSA est plus rapide de $15,18\%$ à $19,85\%$ par rapport au RAP et de $54,64\%$ à $56,61\%$ plus rapide que le RAT. En effet, dans le RAS, il y a aucune attente. Les données collectées sont relayées directement de proche en proche jusqu'à la BS via les *cluster-heads*. Alors que dans le RAP, chaque *cluster-head* attend un temps δ_{agreg} pour procéder à la fusion de toutes les données émanant des nœuds de son *cluster* en un seul agrégat. Au bout du δ_{agreg}, le *cluster-head* envoie l'agrégat à destination du centre de commande. Cette attente au niveau des *cluster-heads* fait que le RAP est $15,18\%$ à $19,85\%$ plus lent que le RSA. Dans le cas du RAT, à chaque fois qu'un nœud détecte une température, l'agent que nous avons implémenté dans sa "couche application" initie une session coopération avec les nœuds du voisinage. Cela nécessite une attente de δ_{agreg} afin de fusionner toutes les données venant éventuellement des nœuds du voisinage qui ont accepté de coopérer. Comme ce procédé se répète tout au long du chemin jusqu'à la BS, le RAT est donc $54,64\%$ à $56,61\%$ plus lent que le RSA.

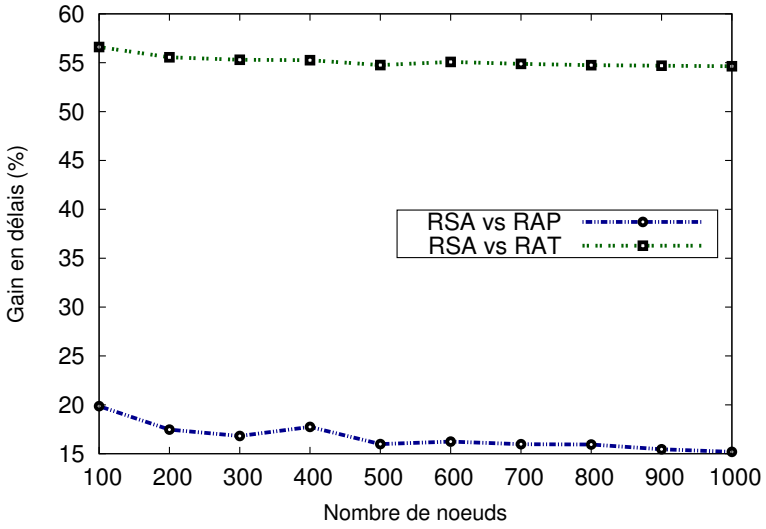


FIGURE 5.8 – Rapidité de RSA comparé au RAP et RAT

Consommation énergétique

Pour la deuxième série de mesures, nous évaluons la consommation énergétique durant une période d'observation commune (temps de simulation) pour le RSA, le RAP et le RAT. Pour ce faire, de la même manière que pour l'évaluation du délai de bout en bout, nous considérons un réseau de taille allant de 100 à 1000 nœuds et un temps de simulation à 1000 secondes avec le paramètre k fixé à 2 et le degré moyen du réseau à 6. La figure 5.9 illustre la consommation énergétique du RSA, RAP et du RAT.

Nous remarquons que les consommations énergétiques des trois scénarios lors de la phase d'acheminement des données captées suivent une évolution logarithmique avec l'évolution du nombre de nœuds dans le réseau. Cela montre le passage à l'échelle des trois techniques de routage en termes de consommation énergétique. Nous constatons également que le RAP consomme

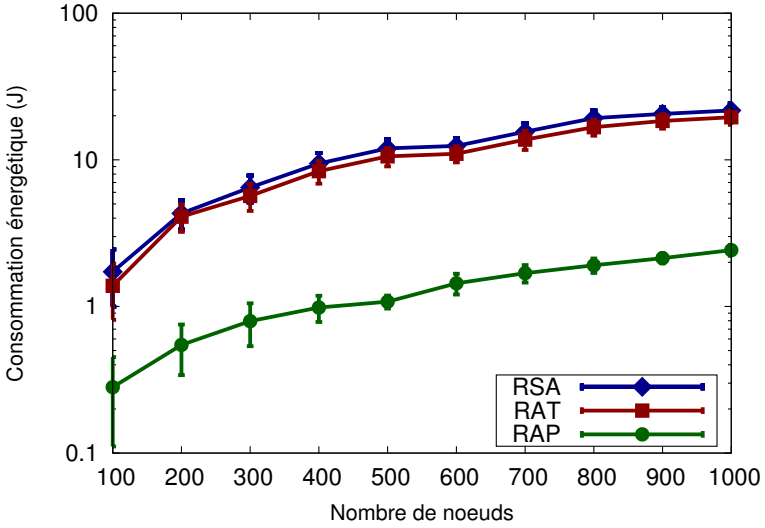
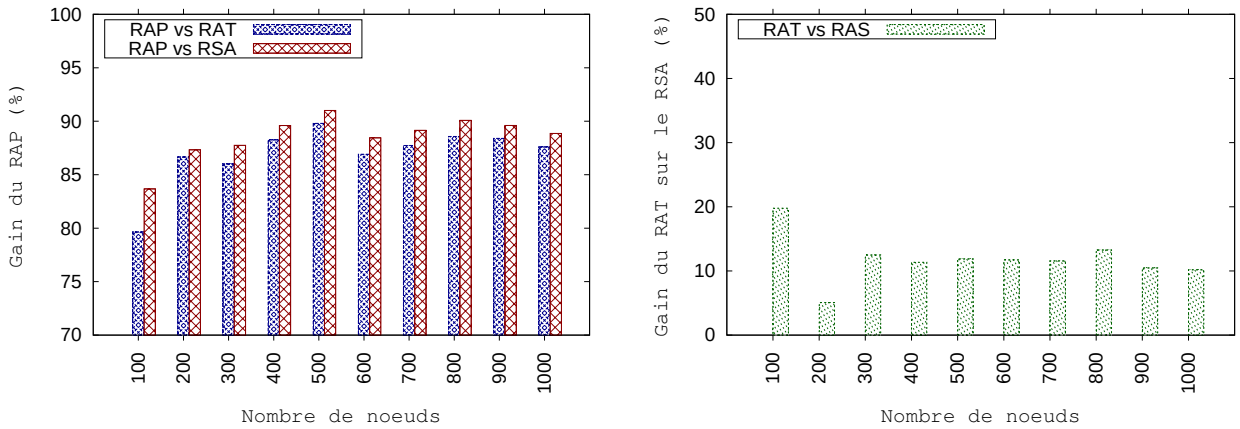


FIGURE 5.9 – Consommation énergétique totale du RSA, RAP et RAT (échelle logarithme)

moins d'énergie comparé aux deux autres scénarios. Nous notons aussi que le RAT consomme moins d'énergie que le scénario du RSA. Pour plus de précision, nous avons calculé le taux de réduction des consommations énergétiques entre les différentes approches de routage dans la figure 5.10.



(a) RAP vs RSA - RAP vs RAT (b) RAT vs RAS

FIGURE 5.10 – Comparaison des consommations énergétiques : RAP vs RSA et RAP vs RAT

La figure 5.10(a) montre que la technique du RAP réduit la consommation énergétique lors de la phase d'acheminement des données de $83,67\%$ à $91,01\%$ par rapport au scénario de RSA et de $79,64\%$ à $89,80\%$ par rapport à l'approche de RAT. L'avantage du RAP est que, comme durant chaque cycle de captage, un seul message contenant toutes les données collectées au sein du *cluster* est envoyé par chaque *cluster-head* à destination de la BS. La consommation énergétique totale du réseau résultante en est considérablement réduite.

La figure 5.10(b) montre que le RAT réduit la consommation énergétique de 5% à 19% comparé au RSA. Le RAT, même avec les messages de coopération échangés entre agents qui entrainent aussi un surplus de consommation énergétique, présente une meilleure consommation énergétique que le RSA. En effet, dans le scénario RSA, engendre plus de trafic car toutes les données captées sont directement acheminées vers la BS. Par conséquent, cela entraine une consommation plus élevée en énergie. Cependant, le RSA offre de meilleurs délais de bout en bout (cf. Figure 5.7).

Durée de vie du réseau

Comme dernière série de mesures, nous évaluons la durée de vie du réseau dans ce cas précis : nous partons d'un réseau quelconque que nous structurons d'abord en *clusters* puis nous exécutons les trois scénarios de routage (RSA, RAP et RAT). Ensuite, nous évaluons le temps de disparition progressive des nœuds en fonction de leur épuisement énergétique ($\mathcal{E}_u^{disp} = 0$). Pour ce faire, nous considérons un réseau de taille fixée à 100 nœuds [HCB00, LR02, DD09, PC10, PSHL10]. À toutes les secondes, chaque nœud du réseau capte une température et l'envoie à la BS. La figure 5.11 montre la durée vie du réseau pour chacun des trois scénarios de routage comme étant le nombre de nœuds en vie ($\mathcal{E}_u^{disp} > 0$) en fonction du nombre de cycles de captages.

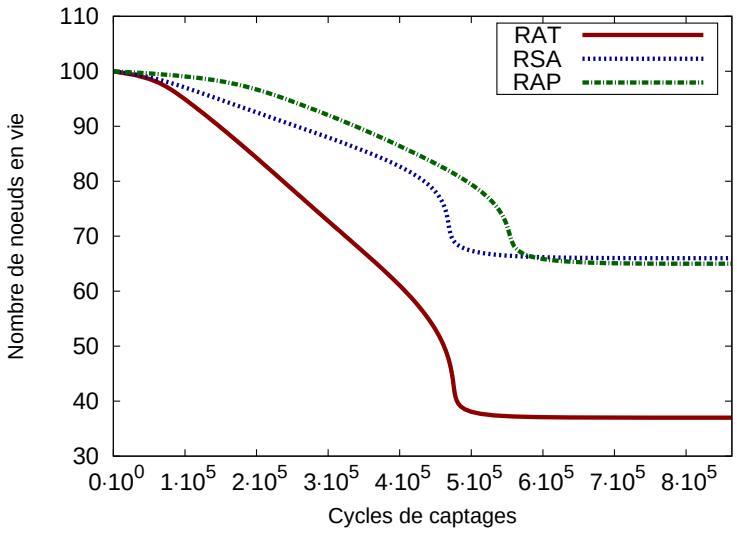


FIGURE 5.11 – Durée de vie du réseau pour le RSA, le RAP et le RAT

Nous remarquons que les nœuds restent en vie pendant un certain temps puis nous constatons une baisse progressive du nombre de nœuds en vie au fil des cycles de captage. Intéressons nous au temps qui s'écoule jusqu'à ce que le premier nœud du réseau disparaisse ($\mathcal{E}_u^{disp} = 0$) pour chaque scénario. Comme résumé dans le tableau 5.2, nous constatons que le premier nœud disparait plus rapidement dans le RAT puis le RSA et enfin le RAP. Ceci est causé par les échanges de messages nécessaires pour la coopération des agents dans le RAT. Dans le RSA, les données captés sont systématiquement routées vers la BS. Par conséquent, en considérant que

le réseau est non-fonctionnel dès la disparition des premiers nœuds, il est nécessaire d'effectuer une reconstruction des *clusters* pour chaque scénario aux instants indiqués dans le tableau 5.2. Dans ce cas, le RAP présente la meilleure durée de vie comparée au RSA et au RAT. En outre, il procure un meilleur compromis entre la réduction de la consommation énergétique totale et les délais de bout en bout.

Toutefois, nous constatons que le RAT (que nous avons implémenté dans un réseau structuré et initialement proposé par Merghem-Boulahia et *al.* pour un réseau plat) présente de moins bonnes performances en termes de consommation énergétique totale à cause des échanges de messages nécessaires à la coopération entre les agents.

	Disparition du premier nœud
RAP	41h 56min 57s
RSA	17h 41min 44s
RAT	15h 42min 57s

TABLE 5.2 – Temps de disparition du premier nœud dans le réseau

5.6 Conclusion

Dans ce chapitre, nous avons proposé une utilisation des *clusters* construits à l'aide de SDEAC pour la collecte et l'acheminement de données depuis des nœuds capteurs sources jusqu'à la station de base (centre de contrôle). Pour ce faire, nous avons proposé trois scénarios que sont le Routage Sans Agrégation (RSA), le Routage avec Agrégation Partielle (RAP) et le Routage avec Agrégation Totale (RAT).

Les résultats d'évaluation ont montré que pour le délai de bout en bout, le scénario de RSA, bien qu'engendrant d'importants échanges de messages, offre une meilleure performance comparé au RAP et au RAT. Pour la consommation énergétique totale dans le réseau, comme le RAP réduit au maximum les messages transitant dans le réseau, il présente la consommation énergétique la plus faible par rapport au RSA et RAT. Pour cette même raison, il offre la meilleure durée vie du réseau.

Nous avons publié les travaux présentés dans ce chapitre dans une conférence internationale avec actes et comité de sélection [BFM+14].

Dans le prochain chapitre, nous tirerons un bilan général de nos travaux de thèse et soulèverons nos perspectives de recherche.

Conclusion et Perspectives

Conclusion

Dans cette thèse, nous nous sommes intéressés à la structuration auto-stabilisante des réseaux ad hoc dans le but d'optimiser les communications et de tolérer les pannes transitoires.

Nous avons proposé un algorithme de structuration nommé SDEAC qui est auto-stabilisant, distribué, déterministe et fondé sur un modèle asynchrone à passage de messages. SDEAC construit des *clusters* non-recouvrants à k sauts en au plus $n+2$ transitions. De plus, l'exécution de SDEAC nécessite une occupation mémoire de $(\Delta_u + 1) \times \log(2n + k + 3)$ bits pour chaque nœud u, avec Δ_u étant le degré - nombre de voisins - de u, k le rayon maximal des *clusters* et n la taille du réseau.

Dans un contexte de réseau ad hoc, nous avons évalué le temps de stabilisation moyen de SDEAC par simulation sous OMNeT++. Les résultats ont montré que, pour des topologies réseaux quelconques et quels que soient la densité, la taille du réseau ainsi que le paramètre k, le temps de stabilisation est très inférieur à celui du pire des cas ($n + 2$ transitions dans le cas de la chaîne ordonnée). Nous avons également constaté à travers les simulations que pour des topologies quelconques, après la stabilisation du réseau, le nombre de transitions supplémentaires pour corriger des pannes transitoires causées par une modification topologique est minimal comparé à celui du *clustering*.

En se plaçant dans le cadre particulier des réseaux de capteurs sans fils (RCSF), nous avons étudié la consommation énergétique de SDEAC en fonction de trois critères d'élection des *cluster-heads* que sont l'identité maximale, le degré et l'énergie résiduelle des nœuds. Les résultats ont montré que les trois critères d'élection permettent le passage à l'échelle. Néanmoins, le critère de l'identité maximale, du fait de sa simplicité d'utilisation et de sa stabilité lors de la phase de *clustering*, présente une plus faible consommation énergétique. Les mesures ont également montré qu'en présence de pannes transitoires, l'énergie consommée lors du *re-clustering* est inférieure à celle du *clustering*.

Nous avons comparé la consommation énergétique de SDEAC avec celle de la solution de Mitton et *al.* [MFGLT05] qui à notre connaissance est la seule à opérer dans le même modèle que la notre. En considérant les mêmes topologies de réseaux que celles utilisées à la base par Mitton et *al.*, les résultats ont montré que SDEAC réduit la consommation énergétique 42% à 49% comparé à la solution de Mitton et *al.* [MFGLT05].

Enfin, pour l'utilisation des *clusters* construits par SDEAC dans l'acheminement de l'information, nous avons d'abord présenté un routage sans agrégation qui minimise les délais de communication. Ensuite, un routage avec agrégation partielle qui réduit les messages transitant dans le réseau ainsi que la consommation énergétique totale. Pour cette même raison, il offre la meilleure durée vie du réseau. Enfin, nous avons montré que le routage avec agrégation totale présente de moins bonnes performances en termes de consommation énergétique totale à cause des échanges de messages nécessaires à la coopération entre les agents.

Bilan

Les travaux de recherche que nous avons mené ont abouti à plusieurs résultats validés au sein de la communauté scientifique par le biais de publications en revues comme en conférences avec actes et comité de sélection :

- deux (2) revues à audience internationale [BFH$^+$13a, BFM$^+$13d] ;
- deux (2) revues à audience francophone [BFH$^+$12a, BFH$^+$14] ;
- cinq (5) conférences à audience internationale [BFH$^+$12b, BFH$^+$13b, BFM$^+$13a, BFM$^+$13c, BFM$^+$14] dont deux (2) "**Best Paper Award**" dans [BFH$^+$13b] et [BFM$^+$13c] ;
- trois (3) conférences à audience francophone [BFH$^+$12c, BFH$^+$12d, BFM$^+$13b].

Perspectives de recherche

Les travaux que nous avons mené et les résultats obtenus ont permis de soulever plusieurs pistes de réflexions, autant sur le plan théorique et scientifique que sur le plan technique, vers lesquelles nous aimerions axer nos futures recherches.

Vers une meilleure tolérance à la mobilité

Dans [DMH13], Ducrocq et *al.* optent pour un critère de choix du *cluster-head* fondé sur une métrique appelée k-densité qui a été proposé par Mitton et *al.* [MFGLT05]. Rappelons que, pour le *clustering* à 2 sauts par exemple, la 2-densité est calculée en fonction au nombre de voisins d'un nœud mais aussi de ses nœuds voisins à distance 2. Comme nous l'avons montré dans le **Chapitre 4**, cette approche nécessite une importante consommation énergétique comparée à la méthode de *clustering* que nous avons proposé (entre 42% à 49% de plus). 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. En revanche, en cas de disparition du *cluster-head*, de nouveaux *clusters* doivent être construits.

L'objectif est maintenant de vérifier cette stabilité et de la concilier avec notre approche qui nécessite peu de messages dans le but d'aboutir à une nouvelle solution afin de garantir une meilleure stabilité du réseau. En effet, bien que le coût de la procédure de *re-clustering* que nous avons évalué est minimal comparé à celui du *clustering*, nous jugeons que cela peut être améliorer en évitant de déclencher une reconstruction dès la moindre modification topologique (notamment du *cluster-head*).

L'idée intuitive est de ne plus construire un seul *cluster-head* mais d'anticiper la disparition du *cluster-head* en effectuant une élection d'un second *cluster-head*, pour chaque *cluster*. La difficulté réside à savoir s'il faut faire une redondance de *cluster-heads* 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 *cluster-head* actif et l'autre dans un état passif ?

Vers une combinaison de critères d'élection des *cluster-heads*

Dans [LGF08a, LGF08b, LFG12], Lehsaini et *al.* ont proposé une approche de *clustering* où l'élection des *cluster-heads* s'effectue selon une combinaison de plusieurs critères tels que l'énergie des nœuds, la densité, la mobilité, etc. L'idée d'une combinaison est de tirer profil des avantages de plusieurs critères. Dans le Chapitre 4, nous avons étudié SDEAC en fonction de trois métriques d'élection (identité, degré et énergie résiduelle) prises séparément. Nous aimerions donc proposer une combinaison de ces trois métriques.

L'idée intuitive est de choisir un *cluster-head* en fonction de deux critères combinés à savoir l'énergie et le degré puis de réserver l'identité comme critère discriminant. Ainsi, un nœud u est choisi comme *cluster-head* s'il maximise une valeur U_u 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 comptons nous fonder sur une méthode de prise de décision MADM (Multiple Attribute Decision-Making) nommée SAW (Simple Additive Weighting) [Zha04, SNW06, CCXW11]. 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 agrégation additive des critères par une somme pondérée, comme le montre l'équation suivante :

$$U_u = w_e \times E_r + w_d \times \triangle_d$$

Les coefficients w_e et w_d représentent les poids (importance) attribués à chaque critère, avec $\sum_{j=1}^{m} w_j = w_e + w_d = 1$. 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 qu'est l'identité des nœuds.

Vers une structuration en arbre de *clusters*

Dans le **Chapitre 5**, afin de construire les routes à travers le réseau, nous avons repris le principe du protocole DSR [JMB01]. Cependant, nous jugeons non négligeable le coût de construction des routes. Donc, une solution serait de construire un arbre de *clusters* (arbre couvrant) dans lequel la racine serait la station de base et tous les fils dans l'arbre seraient constitués de *cluster-heads*.

L'objectif est de mettre œuvre, en même temps que la construction des *clusters* et de façon distribuée et asynchrone, un arbre couvrant du réseau, puis, utiliser les chemins le long de l'arbre pour l'acheminement de l'information vers la station de base qui se trouvera être la racine de l'arbre.

L'idée intuitive consiste à utiliser la distance entre les *cluster-heads* et la station de base comme critère de construction de l'arbre. L'identité des *cluster-heads* peut être associée à la distance pour assurer l'unicité du critère de construction de l'arbre. De plus, comme pour

le *clustering*, les *cluster-heads* doivent se fonder sur un "voisinage local" des *clusters* pour construire l'arbre couvrant. Par voisinage local, nous entendons les informations issues uniquement des *clusters* directement adjacents. Comment s'effectuera alors la construction de l'arbre ? Chaque *cluster-head* aura pour père le *cluster-head* voisin ayant une plus petite distance et une plus grande identité ! Les nœuds ayant un statut NP (Nœuds de Passage) seront utilisés pour propager les informations entre *cluster-heads*.

Vers une meilleure adaptation et prise en compte des contraintes de réseaux de capteurs

Dans [EAA13], El-Aaasser et Ashour ont fourni une nouvelle classification des algorithmes de *clustering* et de routage en fonction de leurs consommations énergétiques. De plus, il est connu que l'émission est fortement consommatrice d'énergie. La première idée est donc de minimiser l'émission de messages. Une solution étant l'agrégation de données. L'approche que nous avons utilisée dans le **Chapitre 5** en est une, et consiste en l'agrégation des données à base de système multi-agents coopératifs.

Un problème fonctionnel, que nous n'avons pas pu étudier dans cette thèse, peut apparaître en pratique avec la technique d'agrégation. 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. Par exemple, pour des algorithmes réactifs nécessitant que la route soit présente dans le paquet transmis, comme actuellement sur les réseaux de capteurs existants les adresses MAC sont sur 64 bits, nous obtenons rapidement des paquets de tailles importantes. Sachant que, pour les nœuds de type Libelium DigiPro [Lib13] par exemple, les portées maximales obtenues par expérimentation en milieu urbain sont de 150 mètres en 868 Mhz avec 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 données à transporter. La limite des paquets étant de 1500 octets, il n'est donc pas envisageable d'utiliser tout type d'agrégation et tout type d'algorithme de routage. A notre connaissance, cette problématique n'a pas encore été étudiée par la communauté scientifique.

Nous aimerions donc explorer cette voie sans dissocier les algorithmes de *clustering* et d'acheminement de l'information. Comme nous l'avons 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 *clustering* 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.

Bibliographie

[ACDFP09] G. Anastasi, M. Conti, M. Di Francesco et A. Passarella : Energy Conservation in Wireless Sensor Networks : A Survey. *Ad Hoc Networks*, 7(3): 537–568, 2009.

[AD08] H. M. Ammari et S. K. Das : Clustering-based Minimum Energy Wireless M-connected K-covered Sensor Networks. *In Proceedings of the 5th European Conference on Wireless Sensor Networks*, EWSN'08, pages 1–16, 2008.

[AG93] A. Arora et M. Gouda : Closure and Convergence : A Foundation of Fault-Tolerant Computing. *IEEE Transactions on Software Engineering - Special issue on software reliability*, 19(11):1015–1027, 1993.

[Agg04] G. Aggelou : *Mobile Ad Hoc Networks : From Wireless LANs to 4G Networks*. 2004.

[AH93] E. Anagnostou et V. Hadzilacos : Tolerating Transient and Permanent Failures (Extended Abstract). *In Proceedings of the 7th International Workshop on Distributed Algorithms*, WDAG'93, pages 174–188, 1993.

[AJRA08] S. Adabi, S. Jabbehdari, A.M. Rahmani et S. Adabi : SBCA : Score Based Clustering Algorithm for Mobile AD-hoc Networks. *In Proceedings of the the 9th International Conference for Young Computer Scientists*, ICYCS'08, pages 427–431, 2008.

[AK98] A. Arora et S. S. Kulkarni : Designing Masking Fault-Tolerance via Nonmasking Fault-Tolerance. *IEEE Transactions on Software Engineering*, 24(6):435–450, 1998.

[AKKUM04] J. N. Al-Karaki, A. E. Kamal et R. Ul-Mustafa : On the Optimal Clustering in Mobile Ad Hoc Networks. *In Porceedings of the 1th First IEEE Consumer Communications and Networking Conference*, CCNC'04, pages 71–76, 2004.

[AKY91] Y. Afek, S. Kutten et M. Yung : Memory-Efficient Self-Stabilizing Protocols for General Networks. *In Distributed Algorithms*, volume 486 de *Lecture Notes in Computer Science*, pages 15–28. 1991.

[AM09] R. Agarwal et M. Motwani : Survey of Clustering Algorithms for MANET. *International Journal on Computer Science and Engineering*, 1(2):98–104, 2009.

[AP00] A. D. Amis et R. Prakash : Load-Balancing Clusters in Wireless Ad Hoc Networks. *In Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, ASSET '00, pages 25–33, 2000.

[APVH00] A. D. Amis, R. Prakash, T. H. P. Vuong et D. T. Huynh : Max-Min D-Cluster Formation in Wireless Ad Hoc Networks. *In Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM'00, pages 32–41, 2000.

[Arn92] A. Arnold : *Systèmes de transitions finis et sémantique des processus communicants*. E.R.I. Etudes et recherches en informatique. Masson, 1992.

[ASSC02] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam et E. Cayirci : Wireless sensor networks : a survey. *Computer Networks : The International Journal of Computer and Telecommunications Networking*, 38(4):393–422, 2002.

[AVX+12] N. Amini, A. Vahdatpour, W. Xu, M. Gerla et M. Sarrafzadeh : Cluster size Optimization in Sensor Networks with Decentralized Cluster-Based Protocols. *Computer Communications*, 35(2):207–220, 2012.

[AW04] H. Attiya et J. Welch : *Distributed Computing : Fundamentals, Simulations, and Advanced Topics*. Wiley series on Parallel and Distributed Computing. 2004.

[AWF02] K. Alzoubi, P. J. Wan et O. Frieder : New distributed algorithm for connected dominating set in wireless ad hoc networks. *In Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, HICSS'02, pages 297–304, 2002.

[AY07] A. A. Abbasi et M. Younis : A Survey on Clustering Algorithms for Wireless Sensor Networks. *Computer Communications*, 30(14-15):2826–2841, 2007.

[AYY05] K. Akkaya, M. Younis et M. Youssef : Efficient Aggregation of Delay-Constrained Data in Wireless Sensor Networks. *In Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications*, AICCSA'05, pages 904–909, 2005.

[BAKA+13] M. Bsoul, A. Al-Khasawneh, A. Abdallah, E.. Abdallah et 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 et S. Rothkugel : WACA : A Hierarchical Weighted Clustering Algorithm Optimized for Mobile Hybrid Networks. *In Proceedings of the 3rd International Conference on Wireless and Mobile Communications*, ICWMC'07, pages 23–32, 2007.

[Bas99] S. Basagni : Distributed Clustering for Ad Hoc Networks. *In Proceedings of the 4th International Symposium on Parallel Architectures, Algorithms, and Networks*, I-SPAN'99, pages 310–315, 1999.

[BC03] S. Bandyopadhyay et E. J. Coyle : An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks. *In Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, INFOCOM'03, pages 1713–1723, 2003.

[BCC+06] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, T. El-Ghazawi et I. Touche : Grid'5000 : A Large Scale And Highly Reconfigurable Experimental Grid Testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[BCD+05] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson et R. Han : MANTIS OS : An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. *Journal Mobile Networks and Applications*, 10(4):563–579, 2005.

[BCV03] L. Blin, A. Cournier et V. Villain : An improved snap-stabilizing pif algorithm. *In Self-Stabilizing Systems*, volume 2704 de *Lecture Notes in Computer Science*, pages 199–214. 2003.

[BFG+03] H. Baala, O. Flauzac, J. Gaber, M. Bui et T. El-Ghazawi : A Self-Stabilizing Distributed Algorithm for Spanning Tree Construction in Wireless Ad Hoc Networks. *Journal of Parallel and Distributed Computing*, 63(1):97–104, 2003.

[BFH+12a] Mandicou Ba, Olivier Flauzac, Bachar Salim Haggar, Florent Nolot et Ibrahima Niang : Clustering auto-stabilisant à k sauts dans les réseaux ad hoc. *Revue URED (Université Recherche Développement), Presses Universitaires de l'Université Gaston Berger de Saint-Louis du Sénégal*, 2012. To appear.

[BFH+12b] Mandicou Ba, Olivier Flauzac, Bachar Salim Haggar, Florent Nolot et Ibrahima Niang : Clustering auto-stabilisant à k sauts dans les réseaux ad hoc. *In Proceedings of the 11th Colloque Africain sur la Recherche en Informatique et Mathématiques appliquées*, CARI'12, pages 420–427, Alger, Algérie, 2012.

[BFH+12c] Mandicou Ba, Olivier Flauzac, Bachar Salim Haggar, Florent Nolot et Ibrahima Niang : Clustering auto-stabilisant à k sauts dans les réseaux ad hoc. *In Actes de la 9e Manifestation des Jeunes Chercheurs en Sciences et Technologies de l'Information et de la Communication*, MajecSTIC'12, pages 1–8, Lile, France, 2012.

[BFH+12d] Mandicou Ba, Olivier Flauzac, Bachar Salim Haggar, Florent Nolot et Ibrahima Niang : Clustering auto-stabilisant à k sauts dans les réseaux ad hoc. *In Actes du 4e Colloque Nationale sur la Recherche en Informatique et ses Applications*, CNRIA'12, Thies - Bambaye, Sénégal, 2012.

[BFH12e] A. Bretto, A. Faisant et F. Hennecart : *Eléments de théorie des graphes.* Collection IRIS. Springer, 2012.

[BFH+13a] Mandicou Ba, Olivier Flauzac, Bachar Salim Haggar, Rafik Makhloufi, Florent Nolot et Ibrahima Niang : Energy-Aware Self-Stabilizing Distributed Clustering Protocol for Ad Hoc Networks : the case of WSNs. *KSII Transactions on Internet and Information Systems*, 7(11):2577–2596, 2013.

[BFH+13b] Mandicou Ba, Olivier Flauzac, Bachar Salim Haggar, Florent Nolot et Ibrahima Niang : Self-Stabilizing k-hops Clustering Algorithm for Wireless

Ad Hoc Networks. *In Proceedings of the 7th ACM International Conference on Ubiquitous Information Management and Communication*, IMCOM'13, pages 38 :1–38 :10, Kota Kinabalu, Malaysia, 2013. **Best Paper Award**.

[BFH+14] Mandicou Ba, Olivier Flauzac, Bachar Salim Haggar, Rafik Makhloufi, Florent Nolot et Ibrahima Niang : Vers une structuration auto-stabilisante de réseaux ad hoc. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées (ARIMA)*, 17:119–141, 2014.

[BFM+13a] Mandicou Ba, Olivier Flauzac, Rafik Makhloufi, Florent Nolot et Ibrahima 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] Mandicou Ba, Olivier Flauzac, Rafik Makhloufi, Florent Nolot et Ibrahima Niang : Étude comparative entre solutions de clustering auto-stabilisantes à k sauts dans les réseaux ad hoc. *In Actes du 5e Colloque Nationale sur la recherche en Informatique et ses Applications*, CNRIA'13, pages 86–93, Zinguinchor, Sénégal, 2013.

[BFM+13c] Mandicou Ba, Olivier Flauzac, Rafik Makhloufi, Florent Nolot et Ibrahima Niang : Evaluation Study of Self-Stabilizing Cluster-Head Election Criteria in WSNs. *In Proceedings of the 6th International Conference on Communication Theory, Reliability, and Quality of Service*, CTRQ'13, pages 64–69, 2013. **Best Paper Award**.

[BFM+13d] Mandicou Ba, Olivier Flauzac, Rafik Makhloufi, Florent Nolot et Ibrahima Niang : Fault-Tolerant and Energy-Efficient Generic Clustering Protocol for Heterogeneous WSNs. *International Journal On Advances in Networks and Services*, 6(3-5):231–245, 2013.

[BFM+14] Mandicou Ba, Olivier Flauzac, Rafik Makhloufi, Leïla Merghem-Boulahia, Florent Nolot et Ibrahima Niang : A Novel Aggregation Approach Based on Cooperative Agents and Self-Stabilizing Clustering for WSNs. *In Proceedings of the 7th International Conference on Communication Theory, Reliability, and Quality of Service*, CTRQ'14, pages 23–26, 2014.

[BGC01] M. Bhardwaj, T. Garnett et A. P. Chandrakasan : Upper Bounds on the Lifetime of Sensor Networks. *In Proceedings of the IEEE International Conference on Communications*, ICC'01, pages 785–790, 2001.

[BJX04] H. Bin, T. Jian et G. Xue : Fault-Tolerant Relay Node Placement in Wireless Sensor Networks : formulation and approximation. *In Proceedings of the Workshop on High Performance Switching and Routing,*, HPSR'04, pages 246–250, 2004.

[BK07a] M. Barbeau et E. Kranakis : *Principles of Ad-hoc Networking.* Wiley Publishing, 2007.

[BK07b] J. Burman et S. Kutten : Time Optimal Asynchronous Self-stabilizing Spanning Tree. *In Distributed Computing*, volume 4731 de *Lecture Notes in Computer Science*, pages 92–107. 2007.

[BKL01] P. Basu, N. Khan et T. D. C. Little : A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks. *In Proceedings of the International Workshop on Wireless Networks and Mobile Computing*, WNMC'01, pages 413–418, 2001.

[BLB95] F. Butelle, C. Lavault et M. Bui : A Uniform Self-Stabilizing Minimum Diameter Spanning Tree Algorithm. *In Distributed Algorithms*, volume 972 de *Lecture Notes in Computer Science*, pages 257–272. 1995.

[BP01] A. Beongku et S. Papavassiliou : A Mobility-based Clustering Approach to Support Mobility Management and Multicast Routing in Mobile Ad-hoc Wireless Networks. *International Journal of Network Management*, 11(6):387–395, 2001.

[BPBR11] L. Blin, M. G. Potop-Butucaru et S. Rovedakis : Self-stabilizing Minimum Degree Spanning Tree within one from the Optimal Degree. *Journal of Parallel and Distributed Computing*, 71(3):438–449, 2011.

[BSM12] S. Balamurugan, S. Saraswathi et A. Mullaivendhan : Delay Sensitive Optimal Anycast Technique to Maximize Lifetime in Asynchronous WSN's. *Procedia Engineering*, 38(0):3351–3361, 2012.

[BT85] G. Bracha et S. Toueg : Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM*, 32(4):824–840, 1985.

[CCL03] I. Chlamtac, M. Conti et J. J. N. Liu : Mobile Ad Hoc Hetworking : Imperatives and Challenges. *Ad Hoc Networks*, 1(1):13–64, 2003.

[CCXW11] Y Chen, H. Chen, L. Xie et K. Wang : A Handoff Decision Algorithm in Heterogeneous Wireless Networks with Parallel Transmission Capability. *In Proceedings of the IEEE Vehicular Technology Conference*, VTC'11, pages 1–5, 2011.

[CDDL10] E. Caron, A. K. Datta, B. Depardon et L. L. Larmore : A Self-Stabilizing k-Clustering Algorithm for Weighted Graphs. *Journal of Parallel and Distributed Computing*, 70(11):1159–1173, 2010.

[Cha82] E.J.H. Chang : Echo Algorithms : Depth Parallel Operations on General Graphs. *IEEE Transactions on Software Engineering*, 8(4):391–401, 1982.

[Chi08] M. H. Chiang : *Energy Optimization in Wireless Sensor Networks : A Study of Power Consumption and Energy Optimization*. VDM Publishing, 2008.

[CKF10] J. Chen, C. S Kim et S. Fu : A Distributed Clustering Algorithm for Voronoi Cell-Based Large Scale Wireless Sensor Network. *In Proceedings of the International Conference on Communications and Mobile Computing*, CMC'10, pages 209–213, 2010.

[CKT02] M. CHATTERJEE, DAS S. K. et D. TURGUT : WCA : A Weighted Clustering Algorithm for Mobile Ad Hoc Networks. *Cluster Computing*, 5(2): 193–204, 2002.

[CKYL06] M. Chen, T. Kwon, Y. Yuan et V. Leung : Mobile Agent Based Wireless Sensor Networks. *Journal of Computers*, 1(1), 2006.

[CR09] S. Chinara et S. K. Rath : A Survey on One-Hop Clustering Algorithms in Mobile Ad Hoc Networks. *Journal of Network and Systems Management*, 17(1-2):183–207, 2009.

[CS10] A. Chamam et P. Samuel : A Distributed Energy-Efficient Clustering Protocol for Wireless Sensor Networks. *Computers & Electrical Engineering*, 36(2):303–312, 2010.

[CT92] T. D. Chandra et S. Toueg : Unreliable Failure Detectors for Asynchronous Systems. *In Proceedings of the 10th annual ACM symposium on Principles of distributed computing*, PODC'91, pages 325–340, 1992.

[CW06] W. Choi et M. Woo : A Distributed Weighted Clustering Algorithm for Mobile Ad Hoc Networks. *In Proceedings of the International Conference on Internet and Web Applications and Services/Advanced International Conference on Telecommunications*, AICT-ICIW '06, pages 73–73, 2006.

[CZ05] Y. Chen et Q. Zhao : On The Lifetime of Wireless Sensor Networks. *IEEE Communications Letters*, 9(11):976–978, 2005.

[DD09] I. Dietrich et F. Dressler : On the Lifetime of Wireless Sensor Networks. *ACM Transactions on Sensor Networks*, 5(1):5 :1–5 :39, 2009.

[DDL09] A.K. Datta, S. Devismes et L. L. Larmore : A Self-Stabilizing O(n)-Round k-Clustering Algorithm. *In Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems*, SRDS'09, pages 147–155, 2009.

[DDS87] D. Dolev, C. Dwork et L. Stockmeyer : On the Minimal Synchronism Needed for Distributed Consensus. *Journal of the ACM*, 34(1):77–97, 1987.

[DFG06] V. Drabkin, R. Friedman et M. Gradinariu : Self-stabilizing Wireless Connected Overlays. *In Principles of Distributed Systems*, volume 4305 de *Lecture Notes in Computer Science*, pages 425–439. 2006.

[DFVG83] E.W. Dijkstra, W. H. J. Feijen et A. J. M. Van Gasteren : Derivation of a Termination Detection Algorithm for Distributed Computations. *Information Processing Letters*, 16(5):217–219, 1983.

[Dij74] E. W. Dijkstra : Self-Stabilizing Systems in Spite of Distributed Control. *Communications of the Association of the Computing Machinery*, pages 643–644, 1974.

[DIM91] S. Dolev, A. Israeli et S. Moran : Resource bounds for self stabilizing message driven protocols. *In Proceedings of the 30th annual ACM symposium on Principles of distributed computing*, PODC'91, pages 281–293, 1991.

[DIM97] S. Dolev, A. Israeli et S. Moran : Resource Bounds for Self-Stabilizing Message-Driven Protocols. *SIAM Journal on Computing*, 26(1):273–290, 1997.

[DLV08] A. Datta, L. Larmore et P. Vemula : Self-Stabilizing Leader Election in Optimal Space. *In Stabilization, Safety, and Security of Distributed Systems*, volume 5340 de *Lecture Notes in Computer Science*, pages 109–123. 2008.

[DLV10] A. K. Datta, L. L. Larmore et P. Vemula : A Self-Stabilizing O(k)-Time k-Clustering Algorithm. *The Computer Journal*, 53(3):342–350, 2010.

[DMCA06] C. De Morais Cordeiro et D. P. Agrawal : *Ad Hoc & Sensor Networks : Theory and Applications*. World Scientific Publishing Company, 2006.

[DMH13] T. Ducrocq, N. Mitton et M. Hauspie : Energy-Based Clustering for Wireless Sensor Network Lifetime Optimization. *In Proceedings of the IEEE Wireless Communications and Networking Conference*, WCNC'13, pages 968–973, 2013.

[Dre08] F. Dressler : *Self-Organization in Sensor and Actor Networks*. Communications Networking & Distributed Systems. Wiley, 2008.

[DS80] E.W. Dijkstra et C. S. Scholten : Termination detection for diffusing computations. *Information Processing Letters*, 4(1):1–4, 1980.

[DW05] F. Dai et J. Wu : On Constructing k-Connected k-Dominating Set in Wireless Networks. *In Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, IPDPS'05, pages 81–91, 2005.

[DYY08] B. P. Deosarkar, N. S. Yadav et R. P. Yadav : Clusterhead Selection in Clustering Algorithms for Wireless Sensor Networks : A survey. *In Proceedings of the International Conference on Computing, Communication and Networking*, ICCCN'08, pages 1–8, 2008.

[EAA13] M. El-Aaasser et M. Ashour : Energy Aware Classification for Wireless Sensor Networks Routing. *In Proceedings of the 15th International Conference on Advanced Communication Technology*, ICACT'13, pages 66–71, 2013.

[ER61] P. Erdos et A. Rényi : On the Evolution of Random Graphs. *Bulletin de l'Institut international de statistique*, 38(4):343–347, 1961.

[EWB87] A. Ephremides, J. E. Wieselthier et D. J. Baker : A Design Concept for Reliable Mobile Radio Networks With Frequency Hopping Signaling. *Proceedings of the IEEE*, 75(1):56–73, 1987.

[Far08] S. Farahani : *ZigBee Wireless Networks and Transceivers*. Newnes, 2008.

[Fie73] M. Fiedler : Algebraic Connectivity of Graphs. *Czechoslovak Mathematical Journal*, 23(98):298–305, 1973.

[FLP85] M. J. Fischer, N. A. Lynch et M. S. Paterson : Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[FM02] Y. Fernandess et D. Malkhi : K-Clustering in Wireless Ad Hoc Networks. *In Proceedings of the 2nd ACM International Workshop on Principles of Mobile Computing*, POMC'02, pages 31–37, 2002.

[Fri13] M. Frikha : *Ad Hoc Networks : Routing, Qos and Optimization*. Wiley, 2013.

[FRWZ07] E. Fasolo, M. Rossi, J. Widmer et M. Zorzi : In-Network Aggregation Techniques for Wireless Sensor Networks : a survey. *IEEE Wireless Communications*, 14(2):70–87, 2007.

[FW08] A. A Fr'ohlich et L. F Wanner : Operating System Support for Wireless Sensor Networks. *Journal of Computer Science*, 4(4):272–281, 2008.

[Gae03] F. C. Gaertner : A Survey of Self-Stabilizing Spanning-Tree Construction Algorithms. Rapport technique, 2003.

[Gal77] R. G. Gallager : Finding a Leader in a Network with O(E) + O(N log(N)) Messages. *In Internal Memo., MIT, Combridge, MA*, 1977. Unpublished Note.

[Gib85] A. Gibbons : *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[GJP+91] K.S. Gilhousen, I.M. Jacobs, R. Padovani, A.J. Viterbi, Jr. Weaver, L.A. et III Wheatley, C.E. : On the Capacity of a Cellular CDMA System. *IEEE Transactions on Vehicular Technology*, 40(2):303–312, 1991.

[GJS+12] T. Gao, R. Jin, J. Song, T. Xu et L. Wang : Energy-Efficient Cluster Head Selection Scheme Based on Multiple Criteria Decision Making for Wireless Sensor Networks. *Wireless Personal Communications*, 63(4):871–894, 2012.

[GM91] M. G. Gouda et N. J. Multari : Stabilizing Communication Protocols. *IEEE Transactions on Computers*, 40(4):448–458, 1991.

[GM95] M. Gondran et M. Minoux : *Graphes et algorithmes*. Eyrolles, 1995.

[Gro14] IEEE Working Group : 802.15 WPAN Task Group 4 (TG4). http ://www.ieee802.org/15/pub/TG4.html, 2014.

[GSS07] V. Gayathri, E. Sabu et T. Srikanthan : Size-Restricted Cluster Formation and Cluster Maintenance Technique for Mobile Ad Hoc Networks. *International Journal of Network Management*, 17(2):171–194, 2007.

[GTCT95] M. Gerla et J. Tzu-Chieh Tsai : Multicluster, Mobile, Multimedia Radio Network. *Wireless Networks*, 1(3):255–265, 1995.

[GYP13] D. Gong, Y. Yang et Z. Pan : Energy-Efficient Clustering in Lossy Wireless Sensor Networks. *Journal of Parallel and Distributed Computing*, 73(9):1323–1336, 2013.

[Haa97] Z. J. Haas : A New Routing Protocol for the Reconfigurable Wireless Networks. *In Proceedings of the 6th IEEE International Conference on Universal Personal Communications*, ICUPC'97, pages 562–566, 1997.

[HCB00] W. R. Heinzelman, A. Chandrakasan et H. Balakrishnan : Energy-Efficient Communication Protocol for Wireless Microsensor Networks. *In Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, HICSS '00, pages 8020–8013, 2000.

[HCB02] W. B. Heinzelman, A. P. Chandrakasan et H. Balakrishnan : An Application-Specific Protocol Architecture for Wireless Microsensor Networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002.

[Hec07] O. M. Heckmann : *The Competitive Internet Service Provider : Network Architecture, Interconnection, Traffic Engineering and Network Design*. Communications Networking & Distributed Systems. Wiley, 2007.

[HL01] L. Higham et Z. Liang : Self-stabilizing Minimum Spanning Tree Construction on Message-Passing Networks. *In Distributed Computing*, volume 2180 de *Lecture Notes in Computer Science*, pages 194–208. 2001.

[HLS05] J. C. Hou, N. Li et I. Stojmenovic : *Topology Construction and Maintenance in Wireless Sensor Networks*, pages 311–341. 2005.

[HMPR86] J.M. Hélary, A. Maddi, N. Plouzeau et M. Raynal : Parcours et apprentissage dans un réseau de processus communicants. Rapport de recherche RR-0543, INRIA, 1986.

[HPR88] J. M. Helary, N. Plouzeau et M. Raynal : A Distributed Algorithm for Mutual Exclusion in an Arbitrary Network. *The Computer Journal*, 31(4):289–295, 1988.

[HSW+00] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler et K. S. J. Pister : System Architecture Directions for Networked Sensors. *ACM SIGPLAN Notices*, 35(11):93–104, 2000.

[Ily02] M. Ilyas : *The Handbook of Ad Hoc Wireless Networks*. Electrical Engineering Handbook. Taylor & Francis, 2002.

[IMM08] K. Iniewski, C. McCrosky et D. Minoli : *Network Infrastructure and Architecture : Designing High-Availability Networks*. Wiley, 2008.

[Ini09] K. Iniewski : *Internet Networks : Wired, Wireless, and Optical Technologies*. Devices, Circuits, and Systems. Taylor & Francis, 2009.

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

[JHRC08] B Jiang, K. Han, B. Ravindran et H. Cho : Energy Efficient Sleep Scheduling Based on Moving Directions in Target Tracking Sensor Network. *In Proceedings of the 6th IEEE International Symposium on Parallel and Distributed Processing*, IPDPS'08, pages 1–10, 2008.

[JM11] C. Johnen et F. Mekhaldi : Self-Stabilization versus Robust Self-Stabilization for Clustering in Ad-Hoc Network. *In Proceedings of the 17th international conference on Parallel processing - Volume Part I*, Euro-Par'11, pages 117–129, 2011.

[JMB01] D. B. Johnson, D. A. Maltz et J. Broch : Ad hoc networking. chapitre DSR : The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks, pages 139–172. 2001.

[JMC+01] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum et L. Viennot : Optimized link state routing protocol for ad hoc networks. *In Proceedings of IEEE International Multi Topic Conference Technology for the 21st Century*, INMIC'01, pages 62–68, 2001.

[JN06] C. Johnen et L. Nguyen : Self-stabilizing Weight-Based Clustering Algorithm for Ad Hoc Sensor Networks. *In Algorithmic Aspects of Wireless Sensor Networks*, volume 4240 de *Lecture Notes in Computer Science*, pages 83–94. 2006.

[JN08] C. Johnen et L. Nguyen : Self-Stabilizing Construction of Bounded Size Clusters. *In Proceedings of the International Symposium on Parallel and Distributed Processing with Applications*, ISPA'08, pages 43–50, 2008.

[JN09] C. Johnen et L. Nguyen : Robust Self-Stabilizing Weight-Based Clustering Algorithm. *Theoretical Computer Science*, 410(6-7):581–594, 2009.

[JYZ09] C. Jiang, D. Yuan et Y. Zhao : Towards Clustering Algorithms in Wireless Sensor Networks : A Survey. *In Proceedings of the 2009 IEEE Conference on Wireless Communications and Networking Conference*, WCNC'09, pages 2009–2014, 2009.

[KAT06] A. Koubaa, M. Alves et E. Tovar : GTS Allocation Analysis in IEEE 802.15.4 for Real-Time Wireless Sensor Networks. *In Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 176–176, 2006.

[KEAC14] A. Karahan, I. Erturk, S. Atmaca et S. Cakici : Effects of Transmit-Based and Receive-Based Slot Allocation Strategies on Energy Efficiency in WSN MACs. *Ad Hoc Networks*, 13(0):404–413, 2014.

[KL70] B. W. Kernighan et S. Lin : An Efficient Heuristic Procedure for Partitioning Graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[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.

[KLS11] J. Kim, X. Lin et N. B. Shroff : Optimal Anycast Technique for Delay-sensitive Energy-constrained Asynchronous Sensor Networks. *IEEE/ACM Transactions on Networking*, 19(2):484–497, 2011.

[KLSS10] J. Kim, X. Lin, N. B. Shroff et P. Sinha : Minimizing Delay and Maximizing Lifetime for Wireless Sensor Networks with Anycast. *IEEE/ACM Transactions on Networking*, 18(2):515–528, 2010.

[KM06] H. Kakugawa et T. Masuzawa : A Self-Stabilizing Minimal Dominating Set Algorithm with Safe Convergence. *In Proceedings of the 20th International Symposium on Parallel and Distributed Processing*, IPDPS'06, pages 8–16, 2006.

[KP93] S. Katz et K. Perry : Self-Stabilizing Extensions for Meassage-Passing Systems. *Distributed Computing*, 7(1):17–26, 1993.

[KW07] H. Karl et A. Willig : *Protocols and Architectures for Wireless Sensor Networks*. Wiley, 2007.

[KYSI11] J. Kuroiwa, Y. Yamauchi, Weihua Sun et M. Ito : A Self-Stabilizing Algorithm for Stable Clustering in Mobile Ad-Hoc Networks. *In Proceedings of the 4th IFIP International Conference on New Technologies, Mobility and Security*, NTMS'11, pages 1–7, 2011.

[Lap88] J. C. Laprie : Surete de fonctionnement et tolerance aux fautes : concepts de base. Rapport technique LAAS-88287, Laboratoire d'automatique et d'analyse des systèmes (Toulouse), 1988.

[Lav95] C. Lavault : *Evaluation des algorithmes distribués : analyse, complexité, méthodes*. Collection Informatique. Hermès, 1995.

[Lav00] I. Lavallée : Self-stabilizing distributed spanning tree and leader election algorithm. *ACIS International Journal of Computer and Information Science*, 1(3):119–125, 2000.

[LC00] H. C. Lin et Y. H. Chu : A Clustering Technique for Large Multihop Mobile Wireless Networks. *In Proceedings of the 51st IEEE Vehicular Technology Conference Proceedings*, VTC'00, pages 1545–1549, 2000.

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

[LeL77] G. LeLann : Distributed Systems : Towards a Formal Approach. *In IFIP Congress*, pages 155–160, 1977.

[Les13] J. Leskovec : SNAP : Stanford Network Analysis Platform. http ://snap.stanford.edu, 2013.

[LFG12] M. Lehsaini, M. Feham et H. Guyennet : Efficient Cluster-Based Fault-Tolerant Schemes for Wireless Sensor Networks. *In Proceedings of the 5th International Conference on New Technologies, Mobility and Security*, NTMS'12, pages 1–5, 2012.

[LG97] C. R. Lin et M. Gerla : Adaptive Clustering for Mobile Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[LGF08a] M. Lehsaini, H. Guyennet et M. Feham : A Novel Cluster-Based Self-Organization Algorithm for Wireless Sensor Networks. *In International Symposium on Collaborative Technologies and Systems*, CTS'08, pages 19–26, 2008.

[LGF08b] M. Lehsaini, H. Guyennet et M. Feham : CES : Cluster-based Energy-efficient Scheme for Mobile Wireless Sensor Networks. *In Wireless Sensor and Actor Networks II*, volume 264 de *IFIP - The International Federation for Information Processing*, pages 13–24. 2008.

[Lib13] Libelium : Waspmote node. http ://www.libelium.com, 2013.

[LNS09] H. Liu, A. Nayak et I. Stojmenovi : Fault-Tolerant Algorithms/Protocols in Wireless Sensor Networks. *In Guide to Wireless Sensor Networks*, Computer Communications and Networks, pages 261–291. 2009.

[LP09] T. P. Lambrou et C. G. Panayiotou : A Survey on Routing Techniques Supporting Mobility in Sensor Networks. *In Proceedings of the 5th International Conference on Mobile Ad-hoc and Sensor Networks*, MSN '09, pages 78–85, 2009.

[LR02] S. Lindsey et C. S. Raghavendra : PEGASIS : Power-Efficient Gathering in Sensor Information Systems. *In Proceedings of the IEEE Aerospace Conference*, pages 1125–1130, 2002.

[LT87] N. A. Lynch et M. R. Tuttle : Hierarchical Correctness Proofs for Distributed Algorithms. *In Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, PODC'87, pages 137–151, 1987.

[LT11] A. Larsson et P. Tsigas : A Self-stabilizing (k,r)-clustering Algorithm with Multiple Paths for Wireless Ad-hoc Networks. *In Proceedings of the 31st International Conference on Distributed Computing Systems*, ICDCS'11, pages 353–362, 2011.

[LTCH06] J. J. Y. Leu, M. H. Tsai, T. C. Chiang et Y. M. Huang : Adaptive Power-aware Clustering and Multicasting Protocol for Mobile Ad Hoc Networks. *In Proceedings of the 3th International Conference on Ubiquitous Intelligence and Computing*, UIC'06, pages 331–340, 2006.

[LWJ05] H. Liu, P. J. Wan et X. Jia : Fault-Tolerant Relay Node Placement in Wireless Sensor Networks. *In Computing and Combinatorics*, volume 3595 de *Lecture Notes in Computer Science*, pages 230–239. 2005.

[Lyn96] N. A. Lynch : *Distributed Algorithms*. Elsevier Science, 1996.

[LZXG12] Z. Liu, Q. Zheng, L. Xue et X. Guan : A Distributed Energy-Efficient Clustering Algorithm With Improved Coverage in Wireless Sensor Networks. *Future Generation Computer Systems*, 28(5):780–790, 2012.

[MA01] A. Manjeshwar et D. P. Agrawal : TEEN : A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks. *In Proceedings of the 15th IEEE International Parallel & Distributed Processing Symposium*, IPDPS '01, pages 189–196, 2001.

[MA02] A. Manjeshwar et D. P. Agrawal : APTEEN : A Hybrid Protocol for Efficient Routing and Comprehensive Information Retrieval in Wireless Sensor Networks. *In Proceedings of the 16th IEEE International Parallel & Distributed Processing Symposium*, IPDPS '02, pages 48–56, 2002.

[Mae85] M. Maekawa : A N algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, 1985.

[MBF04] N. Mitton, A. Busson et E. Fleury : Self-Organization in Large Scale Ad Hoc Networks. *In Proceedings of the Mediterranean ad hoc Networking Workshop*, MedHocNet'04, 2004.

[MEM13] MEMSIC : The memsic solution. http ://www.memsic.com, 2013.

[MFGLT05] N. Mitton, E. Fleury, I. Guerin Lassous et S. Tixeuil : Self-Stabilization in Self-Organized Multihop Wireless Networks. *In 25th IEEE International Conference on Distributed Computing Systems Workshops*, ICDCSW '05, pages 909–915, 2005.

[MH09] F. Meng et Y. Han : A New Association Scheme of IEEE 802.15.4 for Real-Time Applications. *In Proceedings of the 5th International Conference on Wireless communications, networking and mobile computing*, WiCom'09, pages 3432–3436, 2009.

[Mis83] J. Misra : Detecting Termination of Distributed Computations Using Markers. *In Proceedings of the second annual ACM symposium on Principles of distributed computing*, PODC'83, pages 290–294, 1983.

[MR83] M.A. Marsan et D. Roffinella : Multichannel Local Area Network Protocols. *IEEE Journal on Selected Areas in Communications*, 1(5):885–897, 1983.

[MR04] V. Mhatre et C. Rosenberg : Design Guidelines for Wireless Sensor Networks : Communication, Clustering and Aggregation. *Ad Hoc Networks*, 2(1):45–63, 2004.

[MW87] S. Moran et Y. Wolfstahl : Extended Impossibility Results for Asynchronous Complete Networks. *Information Processing Letters*, 26(3):145–151, 1987.

[NGS03] F. G. Nocetti, J. S. Gonzalez et I. Stojmenovic : Connectivity Based k-Hop Clustering in Wireless Networks. *Telecommunication Systems*, 22(1-4):205–220, 2003.

[NK85] R. Nelson et L. Kleinrock : Spatial TDMA : A Collision-Free Multihop Channel Access Protocol. *Communications, IEEE Transactions on*, 33(9):934–944, 1985.

[NQL11] M. Nasim, S. Qaisar et S. Lee : An Energy Efficient Cooperative Hierarchical MIMO Clustering Scheme for Wireless Sensor Networks. *Sensors*, 12(1):92–114, 2011.

[NZD99] A. Nasipuri, J. Zhuang et S.R. Das : A Multichannel CSMA MAC Protocol for Multihop Wireless Networks. *In Proceedings of the IEEE Wireless Communications and Networking Conference*, WCNC'99, pages 1402–1406, 1999.

[OHN09] F. Olivier, B. S. Haggar et F. Nolot : Self-Stabilizing Clustering Algorithm for Ad Hoc Networks. *In Proceedings of the Fifth International Conference on Wireless and Mobile Communications*, ICWMC '09, pages 24–29, 2009.

[PC10] T. V. Padmavathy et M. Chitra : Extending the Network Lifetime of Wireless Sensor Networks Using Residual Energy Extraction - Hybrid Scheduling Algorithm. *International Journal of Communications, Network and System Sciences*, 3(1):98–106, 2010.

[PK00] G. J. Pottie et W. J. Kaiser : Wireless Integrated Network Sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[PR99] C. E. Perkins et E. M. Royer : Ad-hoc On-Demand Distance Vector Routing. *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, WMCSA'99, pages 90–100, 1999.

[Pra96] R. Prasad : *CDMA for Wireless Personal Communications*. Artech House, 1st édition, 1996.

[Pri94] C. Prins : *Algorithmes de graphes*. Eyrolles, 1994.

[PSHL10] L. Peng, G. Song, J. Hai et V. Leung : Maximum Lifetime Broadcast and Multicast Routing in Unreliable Wireless Ad-Hoc Networks. *In Proceedings of the IEEE Global Telecommunications Conference*, GLOBECOM'10, pages 1–5, 2010.

[PV10] K. Parveen et S. Vikas : Traffic Pattern based Performance Comparison of Reactive and Proactive protocols of Mobile Ad-hoc Networks. *International Journal of Computer Applications*, 5(10):16–20, 2010.

[RA81] G. Ricart et A. K. Agrawala : An Optimal Algorithm for Mutual Exclusion in Computer Networks. *Communications of the ACM*, 24(1):9–17, 1981.

[Ran83] S. P. Rana : A Distributed Solution of the Distributed Termination Problem. *Information Processing Letters*, 17(1):43–46, 1983.

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

[Ray91] M. Raynal : *La communication et le temps dans les réseaux et les systèmes répartis : tome 1 d'une introduction aux principes des systèmes répartis*. Collection de la Direction des Études et Recherches d'Électricite de France. Editions Eyrolles, 1991.

[Ray92] M. Raynal : *Synchronisation et état global dans les systèmes répartis*. Collection de la Direction des études et recherches d'Électricité de France. Editions Eyrolles, 1992.

[Ray13] M. Raynal : *Distributed Algorithms for Message-Passing Systems*. Springer, 2013.

[RH88] M. Raynal et J. M. Helary : *Synchronisation et contrôle des systèmes et des programmes répartis*. Editions Eyrolles, 1988.

[Sar10] A. Sardouk : *Agrégation de donnée dans les réseaux de capteurs sans fil à base d'agents coopératifs*. Thèse de doctorat, 2010.

[SBP11] S. K. Sarkar, T. G. Basavaraju et C. Puttamadappa : *Ad Hoc Mobile Wireless Networks : Principles, Protocols and Applications*. Auerbach Publications, 2nd edition édition, 2011.

[Sch93] M. Schneider : Self-Stabilization. *ACM Computing Surveys (CSUR)*, 25(1): 45–67, 1993.

[SDT07] K. SOHRABY, M. DANIEL et Z. TAIEB : *Wireless Sensor Networks Technology Protocols and Applications*. Wiley, 2007.

[Sha11] M. Shakir : *Wireless Sensor Networks*. Lambert Academic Publishing, 2011.

[SHN11] I. G. Shayeb, A. H. Hussein et A. B. Nasoura : A Survey of Clustering Schemes for Mobile Ad-Hoc Network (MANET). *American Journal of Scientific Research*, 20(2011):135–151, 2011.

[Sin91] M. Singhal : A Class of Deadlock-Free Maekawa-Type Algorithms for Mutual Exclusion in Distributed Systems. *Distributed Computing*, 4(3):131–138, 1991.

[SL85] F. B. Schneider et L. Lamport : Paradigms for Distributed Programs. *In Distributed Systems : Methods and Tools for Specification, An Advanced Course*, pages 431–480, 1985.

[SM04] J. Sucec et I. Marsic : Hierarchical Routing Overhead in Mobile Ad Hoc Networks. *IEEE Transactions on Mobile Computing*, 3(1):46–56, 2004.

[SMMB+10] A. Sardouk, M. Mansouri, L. Merghem-Boulahia, D. Gaïti et R. Rahim-Amoud : Multi-Agent System Based Wireless Sensor Network for Crisis Management. *In IEEE Global Telecommunications Conference*, GLOBECOM'10, pages 1–6, 2010.

[SNW06] E. Stevens-Navarro et V. W. S. Wong : Comparison between Vertical Handoff Decision Algorithms for Heterogeneous Wireless Networks. *In Proceedings of the IEEE Vehicular Technology Conference*, VTC'06, pages 947–951, 2006.

[SRAMBG09a] A. Sardouk, R. Rahim-Amoud, L. Merghem-Boulahia et D. Gaïti : A Multi-criterion Data Aggregation Scheme for WSN. *In Proceedings of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, WIMOB '09, pages 30–35, 2009.

[SRAMBG09b] A. Sardouk, R. Rahim-Amoud, L. Merghem-Boulahia et D. Gaïti : Data Aggregation Scheme for a Multi-Application WSN. *In Wired-Wireless Multimedia Networks and Services Management*, volume 5842 de *Lecture Notes in Computer Science*, pages 183–188. 2009.

[SRAMBG09c] A. Sardouk, R. Rahim-Amoud, L. Merghem-Boulahia et D. Gaïti : Information-Importance Based Communication for Large-Scale WSN Data Processing. *In Wireless and Mobile Networking*, volume 308 de *IFIP Advances in Information and Communication Technology*, pages 297–308. 2009.

[Tel00] G. Tel : *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.

[TML07] F. D. Tolba, D. Magoni et P. Lorenz : A Stable Clustering Algorithm for Highly Mobile Ad Hoc Networks. *In Proceedings of the 2nd International Conference on Systems and Networks Communications*, ICSNC '07, pages 11–17, 2007.

[TR04] C. Tang et C. S. Raghavendra : Energy Efficient Adaptation of Multicast Protocols in Power Controlled Wireless Ad Hoc Networks. *Mobile Networks and Applications*, 9(4):311–317, 2004.

[TSV11] P. S. Tanu, R. K Singh et J. Vats : Routing Protocols in Ad Hoc Networks : A Review. *International Journal of Computer Applications*, 25(4):30–35, 2011.

[VBK12] S. Vodopivec, J. Bester et A. Kos : A Survey on Clustering Algorithms for Vehicular Ad-Hoc Networks. *In Proceedings of th 35th International Conference on Telecommunications and Signal Processing*, TSP'12, pages 52–56, 2012.

[VH] A. Varga et R. Hornig : An overview of the omnet++ simulation environment.

[WAP14] Y. F. Wen, T. A. F. Anderson et D. M. W. Powers : On Energy-Efficient Aggregation Routing and Scheduling in IEEE 802.15.4-Based Wireless Sensor Networks. *Wireless Communications and Mobile Computing*, 14(2):232–253, 2014.

[WL03] J. Wu et W Lou : Forward-Node-Set-Based Broadcast in Clustered Mobile Ad Hoc Networks. *Wireless Communications and Mobile Computing*, 3(2):155–173, 2003.

[Woo13] K. Wook : Short Clear Channel Assessment in Slotted IEEE 802.15.4 Networks. *Wireless Personal Communications*, 71(1):735–744, 2013.

[WXM07] Z. Weiyi, G. Xue et S. Misra : Fault-Tolerant Relay Node Placement in Wireless Sensor Networks : Problems and Algorithms. *In Proceedings of the 26th IEEE International Conference on Computer Communications*, INFOCOM'07, pages 1649–1657, 2007.

[WZX+13] J. Wang, Z. Zhang, F. Xia, W. Yuan et S. Lee : An Energy Efficient Stable Election-Based Routing Algorithm for Wireless Sensor Networks. *Sensors*, 13(11):14301–14320, 2013.

[XCL10] Y. Xiao, H. Chen et Frank H. Li : *Handbook on Sensor Networks*. World Scientific, 2010.

[YC05] J. Y. Yu et P. H. J. Chong : A Survey of Clustering Schemes for Mobile Ad Hoc Networks. *IEEE Communications Surveys & Tutorials*, 7(1):32–48, 2005.

[YF04] O. Younis et S. Fahmy : HEED : a Hybrid, Energy-Efficient, Distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3(4):366–379, 2004.

[YP12] Z. Yong et Q. Pei : A Energy-Efficient Clustering Routing Algorithm Based on Distance and Residual Energy for Wireless Sensor Networks. *Procedia Technology*, 29(0):1882–1888, 2012.

[YQWG12] J. Yu, Y. Q, G. Wang et X. Gu : A Cluster-Based Routing Protocol for Wireless Sensor Networks with Nonuniform Node Distribution. *AEU - International Journal of Electronics and Communications*, 66(1):54–61, 2012.

[YWC05] S. Yang, J. Wu et J. Cao : Connected k-hop Clustering in Ad Hoc Networks. *In Proceedings of the International Conference on Parallel Processing*, ICPP'05, pages 373–380, 2005.

[Zha04] W. Zhang : Handover Decision Using Fuzzy MADM in Heterogeneous Networks. *In Proceedings of the IEEE Wireless Communications and Networking Conference*, WCNC'04, pages 653–658, 2004.

[Zig13] ZigBee : ZigBee Alliance. http ://www.zigbee.org, 2013.

Vers une structuration auto-stabilisante des réseaux ad hoc: cas des réseaux de capteurs sans fil

Résumé Nous proposons un algorithme original de structuration des réseaux ad hoc nommé SDEAC dans le but d'optimiser les communications et de tolérer les pannes transitoires. SDEAC est auto-stabilisant, distribué et déterministe. Il utilise un modèle asynchrone à passage de messages et se fonde sur un voisinage à distance 1 pour construire des *clusters* non-recouvrants à k sauts. Nous montrons que partant d'une configuration quelconque et sans occurrence de pannes transitoires, SDEAC structure le réseau dans le pire des cas en $n + 2$ transitions. En outre, son exécution nécessite une occupation mémoire de $(\Delta_u + 1) \times \log(2n + k + 3)$ bits pour chaque nœud u, avec Δ_u étant le degré de u, k le rayon maximal des *clusters* et n la taille du réseau. Par le biais de simulation sous OMNeT++, nous observons pour un réseau quelconque un temps de stabilisation très inférieur à celui du pire des cas d'une part. D'autre part, suite à l'occurrence de pannes transitoires après la stabilisation, nous constatons un temps de stabilisation inférieur à celui du *clustering*. Dans le contexte des RCSF, nous étudions la consommation énergétique de SDEAC suivant trois critères d'élection des *cluster-heads* (identité, degré et énergie résiduelle des nœuds) puis nous la comparons avec celle de la solution de Mitton et *al.* opérant dans le même modèle. Les résultats montrent que SDEAC permet le passage à l'échelle et réduit la consommation énergétique de 42% à 49%. Enfin, pour l'utilisation de SDEAC dans l'acheminement de l'information, nous proposons deux approches efficaces : *(i)* un routage sans agrégation qui minimise les délais de bout en bout et *(ii)* un routage avec agrégation partielle qui réduit la consommation énergétique totale offrant ainsi une meilleure durée de vie du réseau.

Mots-clés : Systèmes distribués, réseaux ad hoc, auto-stabilisation, structuration, économie d'énergie, tolérance aux pannes.

Towards a self-stabilizing structuring of ad hoc networks: the case of wireless sensor networks

Abstract We propose SDEAC, a self-Stabilizing Distributed Energy-Aware and fault-tolerant Clustering algorithm. SDEAC uses an asynchronous message-passing model and it is based on 1-hop neighboring to build non-overlapping k-hops clusters. We prove that, starting from an arbitrary configuration, SDEAC structures the network after at most $n+2$ transitions and requires $(\Delta_u + 1) \times \log(2n + k + 3)$ memory space for each node u, where n is the number of network nodes, Δ_u is the degree of u and k represents the maximum hops number. Through simulations under OMNeT++, we observe that over arbitrary network, the stabilization time is far below the worst case scenario. Furthermore, we remark that after faults, the re-clustering cost is significantly lower than the clustering cost. In the context of Wireless Sensor Networks (WSNs), we evaluate the energy consumption of SDEAC according to multiple criteria in the election of cluster-heads, such as nodes' identity, residual energy or degree and we compare it with the well-known message-passing based self-stabilizing clustering algorithm proposed by Mitton et *al.*. The obtained results show that SDEAC is scalable and reduces energy consumption between 42% and 49%. Afterwards, we propose efficient scenarios in order to transfer information : *(i)* the non-aggregation scenario that provides a better end-to-end delay and *(ii)* the partially-decentralized aggregation scenario that reduces the total energy consumption and prolongs the network lifetime.

Keywords : Distributed systems, ad hoc networks, self-stabilization, clustering, energy-aware, fault-tolerant.

Centre de Recherche en STIC (CReSTIC) - Équipe Systèmes Communicants (SysCom)