République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderahmane Mira de Béjaia

Faculté des Sciences Exactes
Département d'Informatique
Ecole Doctorale Réseaux et Systèmes Distribués

Mémoire de Magistère
En Informatique

Option
Réseaux et Systèmes Distribués

Thème

Tolérance aux défaillances dans les réseaux de capteurs sans fil

Présenté par

ZEMMAR Ammar

Devant le jury composé de :

Président :	**Mohand Ouamer BIBI**	**Professeur**	**Université de Béjaia**
Directeur du Mémoire :	**Ali BELMEHDI**	**Professeur**	**Université de Béjaia**
Examinateurs :	**Boubekeur MENDIL**	**Professeur**	**Université de Béjaia**
	Abdelouahab MOUSSAOUI	**Maître de conférences**	**Université de Sétif**
Invité :	**Hamouma MOUMEN**	**M.A.**	**Université de Béjaia**

Juin 2008

Remerciement

Pour m'avoir, dans un premier temps, fait confiance et partagé avec moi sa vision du domaine de tolérance aux fautes et les réseaux de capteurs, et dans un second temps, avoir été un directeur de mémoire actif et patient, je tiens à remercier tout particulièrement Monsieur BELMEHDI Ali, professeur à l'université A/Mira. Je le remercie pour son soutien, remarques et conseils très constructifs.

J'exprime ma profonde reconnaissance à Monsieur MOUMEN Hamouma, co-promoteur, maître assistant au département d'informatique de l'université A/Mira, qui a participé dans la direction de mémoire et pour m'avoir encadré et soutenu tout au long de ce mémoire.

Je tiens à remercier monsieur ALOUI, maître assistant au département d'informatique, pour ses conseils, ses remarques pertinentes et son soutien moral, qui ont fortement contribué au bon déroulement des travaux présentés dans ce mémoire.

Je remercie Monsieur Mohand Ouamer BIBI, Professeur à l'université de Béjaia, pour l'honneur qu'il m'a fait en présidant mon jury de mémoire.

Je remercie tout particulièrement Monsieur Boubekeur MENDIL, Professeur à l'Université de Béjaia, et Monsieur Abdelouahab MOUSSAOUI, maître de conférences à l'université de Sétif, pour avoir accepté la charge d'être examinateurs.

Mes remerciements s'adressent également à tous mes professeurs, qui ont contribué dans notre formation de première année magistère à l'école doctorale d'informatique de l'université Abderrahmane Mira de Béjaïa, et surtout à monsieur *Mr* TARI Kamel pour ses contributions à l'école doctorale (ReSyd). Je désire également remercier mes collègues de l'école doctorale d'informatique ReSyD 2 et 3 pour leur soutien moral.

Mes remerciements vont à ceux qui ont participé, de près ou de loi à l'élaboration de ce mémoire, surtout Monsieur Mekki qui essaye toujours de m'offrir les meilleures conditions pour vivre et étudier à Béjaia dès le premier jour, pour son soutien dans les moments les plus difficiles, pour son encouragement.

Un grand merci exceptionnel à ma mère, mes frères et ma sœur, pour m'avoir donné tout ce qui est beau et bon.

JEMMAR.A

Ce mémoire est dédié spécialement

A la plus belle femme dans ce monde, ma mère

Sommaire

Introduction générale 1

Chapitre 1 : Introduction aux réseaux de capteurs sans fil 4

1.1 Introduction 4

1.2 Réseau de capteurs sans fil "Wireless Sensors Network" 5

1.2.1 Facteurs influant la conception des RCSF 5

1.2.2 Applications des réseaux de capteurs 7

1.3 Architecture de réseau de capteurs 9

1.3.1 Le nœud capteur 9

1.3.2 Communication dans les réseaux de capteurs sans fil 10

1.3.3 Liaisons sans fil 11

1.3.4 Modèle en couches 11

1.3.5 Architecture des réseaux de capteurs sans fil 12

1.3.5.1 Les réseaux de capteurs sans fil plats 12

1.3.5.2 Les réseaux de capteurs hiérarchiques 12

1.4 Les services 13

1.4 .1 La Synchronisation 13

1.4 .2 La localisation 13

1.4 .3 Agrégation De Données 14

1.4 .4 Stockage De Données 15

1.4 .5 Gestion de topologie 16

1.4 .5.1 Auto configuration 16

1.4 .5.2 Clusterisation 16

1.4 .5.3 L'auto adressage 17

1.4 .6 Routage de messages 18

1.5 Conclusion 19

Chapitre 2 : Les techniques de tolérance aux fautes dans les réseaux de capteurs sans fil 20

2.1 Introduction 20

2.2 Sûreté de fonctionnement de RCSF 21

2.3Les fautes et les défaillances dans RCSF 22

2.3.1. Notions de faute, d'erreur et de défaillance 23

2.3.2. Les sources de fautes dans les réseaux de capteurs 24

2.3.3. Classification de fautes et défaillances 26
2.4Tolérance aux fautes dans les réseaux de capteurs 28
2.4.1. La Tolérance aux fautes 28
2.4. 1 .1 Les mécanismes de tolérance aux fautes 28
2.4.2. La détection de fautes et le recouvrement dans les réseaux de capteurs 29
2.4.3. Tolérance aux fautes dans les différents niveaux de réseau de capteurs 30
2.5 Modèle d'intégration Tolérant aux fautes 32
2.5.1 Modèle De Marzullo 32
2.5.2 Modèle d'Iyengar 33
2.5.3. Comparaison de modèle de Marzullo et d'Iyengar 33
2.6 Tolérance aux fautes dans le routage 34
2.6.1. Le routage multi chemins (multipath routing) 34
2.6.2. Le problème de trou dans le routage 35
2.6.2.1 Définition formelle de poids dans WEAR 39
2.6.2.2. Calcul de l'information de trou 39
2.6.2.2 .a. L'identification du trou 41
2.6.2.2. b. Maintenance du trou 42
2.6.2.2.b.1. Protocole de maintenance du trou 42
2.6.2.2.b.2. Agrandissement du trou 43
2.6.2.2.b.3. Fusion du trou 44
2.7 Tolérance aux fautes dans l'agrégation de données 45
2.7.1. Les approches de corrélations 45
2.7.2. L'impact de perte de lien du réseau 46
2.7.3. Tolérance aux fautes du schéma d'agrégation 47
2.8 Tolérance aux fautes dans le contrôle de topologie 50
2.9 Tolérance aux fautes dans la détection de cible et d'événement 52
2.9.a Détection de cible 52
2.9.b Détection d'événement 53
2. 10 Tolérance aux fautes dans les applications de surveillance 54
2.11 Conclusion 55
Chapitre 3 : Le service d'élection de leader pour une agrégation tolérante aux fautes en réseaux de capteurs sans fil 56
3.1 Introduction 56
3.2 Le consensus 57
3.2.1. Paradigme du consensus 58
3.2.2. Les Détecteurs de défaillances 58

3.2.2.1. Propriétés de détecteurs de défaillances 59
3.2.2.2. Les classes de détecteurs de défaillances 59
3.3 Élection de leader 60
3.3.1. Protocoles d'élection basés sur les détecteurs de défaillances 61
3.3.1.1 Le détecteur de défaillance Ω 61
3.4 Protocole d'élection pour le réseau de capteurs 62
3.4.1. Protocole WWLE (Wirless Wave Leader Election) 63
3.4.2. Election d'agrégateur en RCSF par implémentation de détecteur de défaillance oméga..... 64
3.4.2.1. Implémentation d'oméga pour le niveau local (Intra- région) 67
3.4.2.2 Implémentation hiérarchique d'oméga pour le niveau global (Inter-région) 73
3.5 Conclusion..... 76
Chapitre 4: Implémentation de détecteur de défaillance Ω pour l'élection des agrégateurs dans un réseau de capteurs hiérarchiques 77
4.1 Introduction 77
4.2 Modèle du système 78
4.3 Détecteur de défaillances Ω..... 79
4.4 Un algorithme implémentant Ω dans un RCSF 81
4.5 La preuve de l'algorithme 84
4.6 Un algorithme pour l'agrégation de données en utilisant Ω……85
4.7 Conclusion..... 87
Conclusion générale 88
Références 89

Liste de figures

Figure 1 : composantes d'un nœud capteur 9
Figure 2 : exemples de quelques nœuds capteurs 10
Figure 3 : dispersement des nœuds capteurs dans un terrain 10
Figure 4: La pile de protocole d'un réseau de capteurs 12
Figure 5 : de la faute à la défaillance 23
Figure 6 : Récursivité des fautes, erreurs et défaillances 23
Figure 7: modèle de Marzullo 32
Figure 8 : modèle d'Iyengar 33
Figure 9: agrandissement de trou dans GPSR 37
Figure 10: exemple de protocole WEAR 38
Figure 11 : Exemple d'identification et propagation du Trou 41
Figure 12: exemple de maintenance du trou 43
Figure 13: fusion de deux trous 44
Figure 14 : exemple de diffusion de synopsis 48
Figure 15 : schéma d'agrégation qui traite les valeurs dupliquées 49
Figure 16 : Classification des détecteurs de défaillances. 60
Figure 17 : Implémentation de Ω dans un réseau de capteurs sans fil 56
Figure 18 : comparaison de durée de vie de batterie d'un capteur 67
Figure 19 : modification de l'algorithme (seuil) 67
Figure 20 : le deuxième algorithme 2 d'implémentation de Ω dans un réseau de capteurs sans fil pour le niveau local 69
Figure 21 : le troisième algorithme 3 d'implémentation de Ω dans un réseau de capteurs sans fil pour le niveau local 71
Figure 22 : algorithme de niveau global (partie 1) 74
Figure 23 : algorithme de niveau global (partie 2) 75
Figure 24 : schéma de l'implémentation hiérarchique de Ω dans un réseau de capteurs sans fil 81
Figure 25: implémentation hiérarchique de Ω dans un réseau de capteurs sans fil 83
Figure 26: Agrégation de données en utilisant Ω 86

Introduction Générale

La révolution des réseaux de communication est dûe à l'évolution de la technologie sans fil, et les derniers progrès sont couronnés par l'apparition des réseaux mobiles sans infrastructure, appelés les réseaux Ad-hoc, qu'il s'agit d'unités mobiles qui communiquent sans fil et sans infrastructure préexistante. Le réseau de capteurs sans fil, un type particulier de réseaux Ad-hoc, a vu le jour grâce au développement de la micro-électronique et la miniaturisation croissante de composants électroniques, ces capteurs traitent et communiquent les informations après les avoir capté à partir d'un environnement souvent hostile et dur.

Les réseaux de capteurs sans fil sont basés sur une interaction entre des capteurs, processeurs et actionneurs pour que le monde virtuel soit capable de percevoir et d'agir sur son environnement réel, à travers une variété d'applications, militaires, environnementales, de santé et commerciales. Ce domaine se dirige donc vers un grand essor et promet d'être la nouvelle révolution de la technologie d'information (IT).

Les réseaux de capteurs sans fil imposent de nouveaux défis conceptuels et techniques aux chercheurs de tolérance aux fautes. A cause de la communication sans fil, les ressources limitées en énergie et le déploiement dans des environnements durs et parfois dangereux (les volcans, la mer, etc.), les capteurs sont enclins à une variété de fautes qui peuvent endommager la fonctionnalité du réseau et causent la perturbation des applications et les services de réseaux comme le routage et l'agrégation de données, où la panne des agrégateurs dans un schéma d'agrégation de données mène à la perte des données collectées par les nœuds.

Le déploiement dense et aléatoire et la topologie dynamique des réseaux de capteurs exigent l'autonomie de gestion de réseau, et les nœuds capteurs peuvent choisir par eux mêmes les agrégateurs de données et tolèrent les pannes de ceux-ci, par utilisation de service d'élection de leader.

Larrea & al [89] utilisent le détecteur de défaillance de type oméga pour l'élection de leader (agrégateur de données) en réseau de capteurs. Ils présentent un algorithme qui adapte l'implémentation de détecteur de défaillance de type oméga (Ω) dans le modèle de panne / reprise, pour coordonner l'agrégation de données dans un réseau de capteurs sans fil hiérarchique et divisé en régions. L'algorithme présenté par Larrea est destiné à la partie de réseaux de capteurs où l'intervention humaine est permanente afin de remplacer les batteries.

Comme la plupart des déploiements de réseaux de capteurs sont faits dans des environnements hostiles, difficiles et parfois dangereux, ce qui rend le remplacement des batteries, une tâche difficile voir impossible, et afin d'augmenter l'autonomie dans la gestion de réseaux, notre travail consiste à écrire un algorithme d'élection d'agrégateurs, basé sur le détecteur de défaillance Ω pour la construction d'un schéma d'agrégation destiné à ce type de déploiement et sera basé sur l'hypothèse de l'impossibilité de remplacement des batteries.

Organisation de mémoire :

- Le premier chapitre donne un aperçu général sur les réseaux de capteurs sans fil, présente ses caractéristiques, ses applications et ses architectures.

- Le deuxième chapitre est consacré à définir les fautes et les défaillances qui peuvent survenir dans un réseau de capteurs et ses origines, et présente une étude des techniques de tolérance aux fautes dans les réseaux de capteurs sans fil, qui existent dans la littérature.

- Le troisième chapitre présente l'utilisation de service d'élection de leader pour coordonner l'agrégation de données. Il présente un rappel sur le problème d'élection, le consensus et les détecteurs des défaillances, ainsi qu'une présentation de détecteur de défaillance de type oméga. Les algorithmes d'élection de leader dans les réseaux de capteurs existants sont aussi présentés.

- Le dernier chapitre constitue la contribution de ce travail. Il présente notre algorithme, avec lequel nous avons implémenté le détecteur de défaillance oméga pour coordonner l'agrégation de données dans des réseaux de capteurs hiérarchiques, où l'intervention

humaine est limitée dans le déploiement. Une preuve d'exactitude de l'algorithme est donnée dans ce chapitre.

En fin, notre mémoire s'achève par une conclusion générale résumant les grands points qui ont été abordés dans ce mémoire, ainsi que des perspectives.

Chapitre 1 :

Introduction aux réseaux de capteurs sans fil

1.1 Introduction:

Les réseaux sans fil sont deux catégories: réseaux à infrastructure fixe et d'autres sans infrastructure préétablie, la première classe représente le modèle cellulaire alors que la deuxième est celle des réseaux ad hoc.

Les réseaux ad hoc sont connus sous le nom MANET "Mobile Adhoc NETwork" [1], il s'agit d'un ensemble relativement dense, d'unités mobiles qui se déplacent dans une région quelconque, sans infrastructure préexistante ou une administration centralisée, dont le seul moyen de communication est l'utilisation des interfaces sans fil.

Les avances enregistrées dans la technologie sans fil, la microélectronique et le besoin de collecter les informations dans des endroits hostiles et difficiles, ont donné naissance à une nouvelle génération de réseau de type MANET, des nœuds capteurs collectent des informations, les traitent et communiquent les données vers une station de base lointaine d'un champ de captage. C'est ce qu'on appelle réseau de capteurs sans fil, en anglais WSN (Wireless Sensors Network).

1.2 Réseau de capteurs sans fil "Wireless Sensors Network":

Un réseau de capteurs sans fil (RCSF) est composé d'un grand nombre de nœuds capteurs densément déployés à l'intérieur du phénomène ou tous près [1]. Interconnectés via un lien sans fil (radio, infrarouge, optique); ces noeuds ont pour fonctions de capturer, mémoriser, traiter et de communiquer les données vers une station de base (nœud qui collecte les données capturées). Le réseau de capteurs peut contenir des actionneurs, ce qui étend le contrôle du monde virtuel au monde physique.

Les réseaux de capteurs sans fil (RCSF) sont une instance particulière de la classe des réseaux ad hoc [1], ils héritent des caractéristiques et partagent beaucoup de concepts existants avec ceux-ci, par contre il existe un certain nombre de différences et de défis spécifiques tels que :

- ✓ Le nombre de nœuds dans le réseau de capteurs est beaucoup plus élevé par apport à celui d'un réseau ad hoc.
- ✓ Les noeuds capteurs déployés densément.
- ✓ La topologie du réseau de capteurs change fréquemment.
- ✓ La communication entre les nœuds capteurs, repose sur la diffusion, alors que la plupart des réseaux ad hoc utilisent une communication point à point.
- ✓ Les nœuds capteurs ont des ressources limitées (puissance faible en transmission et calcul, énergie et capacité mémoire limitées).
- ✓ Les nœuds capteurs peuvent ne pas avoir des identificateurs globaux comme dans le protocole IP.

1.2.1 Facteurs influant la conception des RCSF :

Tandis que le réseau de capteurs partage beaucoup de concepts avec le réseau ad hoc, il y a aussi un certain nombre de différences et de défis spécifiques, intervenant comme facteurs qui influencent sur la conception de réseaux de capteurs sans fil, à savoir, la tolérance aux fautes, la scalabilité, le coût de production, l'environnement, la topologie du réseau, les contraintes sur le matériel et la consommation d'énergie.

1. Tolérance aux fautes : est la capacité de maintenir le réseau de capteurs sans fil en fonctionnement en cas de défaillance, même s'il fonctionne en mode dégradé. Quelques

capteurs peuvent tomber en panne à cause du manque d'énergie, dommages physiques ou interférence avec l'environnement. L'échec des capteurs ne doit pas affecter le fonctionnement du réseau.

2. Scalabilité : Les réseaux de capteurs peuvent contenir des centaines ou des milliers de noeuds capteurs et par conséquent le réseau doit être capable de fonctionner avec un grand nombre de capteurs tout en permettant l'évolution de ce nombre.

3. Contraintes matérielles : La principale contrainte matérielle est liée aux faibles dimensions d'un nœud capteur, et la taille exigée peut être plus petite qu'un centimètre cube, Ce qui nécessite l'emploi de technologies de pointe pour disposer de circuits électroniques à haut niveau d'intégration et de batteries de petites tailles.

4. Topologie des réseaux de capteurs sans fil : le déploiement d'un grand nombre et la mobilité continue des nœuds nécessite une maintenance de la topologie.

5. l'environnement : les noeuds de capteurs fonctionnent habituellement sans surveillance dans des régions géographiques éloignées. ils peuvent fonctionner :

- à l'intérieur de grandes machines,
- au fond d'un océan, ou sur la surface d'un océan pendant une tornade,
- dans des réacteurs chimiques
- dans un champ de bataille,
- à l'intérieur des bâtiments,
- attachés aux animaux,
- fixés aux véhicules,
- dans un fleuve se déplaçant avec le courant, etc.

6. Média de transmission : Les nœuds communiquant sont reliés de manière sans fil. Ce lien peut être réalisé par radio, signal infrarouge ou un média optique.

7. La consommation d'énergie : C'est actuellement le plus grand défi de conception dans les applications de réseau de capteurs. Les nœuds, étant des dispositifs microélectroniques, ne peuvent être équipés que par une source énergétique limitée dont dépend la durée de vie du nœud, et par conséquent, la durée de vie du réseau. De ce fait, la gestion et la conservation d'énergie prennent une grande importance pour prolonger la durée de vie du réseau.

8. coût de production : Les réseaux de capteurs doivent être à faible coût, afin de pouvoir être déployés à grande échelle, et ceci en minimisant la complexité des composants, des protocoles, des exigences de mémoires et le réseau doit aussi être tolérant aux pannes et auto configurable « Self configuration », qui est défini comme étant la capacité des noeuds du réseau de détecter la présence d'autres nœuds, et de s'organiser en réseau fonctionnel structuré, sans intervention humaine, et auto maintenue « Self maintenance », qui est défini comme étant la capacité du réseau à détecter les fautes ou les pannes des nœuds, sans intervention humaine.

1.2.2 Applications des réseaux de capteurs :

Cette nouvelle technologie promet dans beaucoup de domaines de nouvelles applications [1] :

A) Applications militaires :

Dans les applications de sécurité en général, la réduction de latence d'une transmission d'alarme, d'une manière significative, est plus importante que la réduction de coût énergétique des transmissions, la communication immédiate et fiable des messages d'alarme est la condition primaire de système, parce que les événements d'alarme sont rares et importants. Dans les applications militaires, les nœuds capteurs devraient fournir des services comme:

- surveillance du champ de bataille
- Reconnaissance des forces ennemies
- repérage des cibles
- évaluation des dommages de la bataille
- détection et reconnaissance d'attaque nucléaire, biologique et chimique.

B) Applications environnementales:

L'application de collection de données dans un environnement est caractérisée par le grand nombre de nœuds, qui captent et transmettent les données à la station de base, d'une manière continue et sous des topologies relativement statiques. Ces réseaux exigent généralement des débits très bas, une synchronisation précise et une durée de vie extrêmement longue.

Les paramètres typiques d'environnement surveillé, comme la température, l'intensité de la lumière et l'humidité, ne changent pas rapidement, et par conséquent, n'exigent pas des taux de transmission élevés. Pour beaucoup de scénarios, l'intervalle entre les transmissions peut être de l'ordre de quelques minutes (1 et 15 minutes). En plus, les applications de contrôle de l'environnement n'ont pas de conditions strictes de latence, les échantillons de données peuvent être retardés à l'intérieur du réseau pour des périodes modérées, selon le besoin et pour améliorer l'efficacité du réseau, sans affecter de manière significative l'exécution d'application. En général, les données sont rassemblées pour une future analyse et pas pour des opérations en temps réel. Les applications de contrôle d'environnement incluent:

- Le repérage des mouvements des oiseaux, des petits animaux et des insectes;
- La surveillance des conditions environnementales qui affectent les récoltes et le bétail;
- La détection de feu de forêt;
- La détection d'inondation;
- Etude de pollution … etc.

C) Applications dans la santé :

Les réseaux de capteurs sont appliqués aussi dans le domaine médical, cette classe inclut des applications comme :

- fourniture d'interfaces pour les handicapés;
- repérage et surveillance des médecins et des patients dans les hôpitaux;
- le diagnostic et la télésurveillance des données physiologiques humaines;

D) Applications domestiques :

C'est une autre classe d'applications où plusieurs concepts sont déjà conçus par les chercheurs et les architectes, comme la maison intelligente (domotique), oû des nœuds capteurs sont intégrés à l'intérieur des dispositifs domestiques permettant l'interconnexion et l'interactivité. Le calcul et le captage dans cet environnement doivent être fiables et persistants.

1.3 Architecture d'un réseau de capteurs:

1.3.1 Le nœud capteur :

Un nœud capteur est composé de 4 unités :

- ***Unité de détection (sensing unit)*** : est composée de deux sous unités, capteur (sensor) et ADC (Analog to Digital Converters) qui permet de convertir le signal produit par le capteur, sur la base du phénomène observé, en un signal digital qui est par la suite fourni à l'unité de calcul.
- ***Unité de calcul (processing unit):*** qui est généralement associé à une petite unité de stockage, gère les procédures permettant au noeud de collaborer avec les autres noeuds pour accomplir la requête assignée.
- ***Unité de transmission****:* connecte le noeud au réseau.
- ***Unité d'énergie (power unit):*** la batterie, composante importante du nœud capteur, contient l'énergie électrique.

Selon les applications, il peut y avoir d'autres unités telles que *Location finding system* qui fournit des informations sur la localisation requise par les techniques de routage et le *mobiliser* qui est nécessaire si le nœud capteur doit être déplacé pour accomplir la requête assignée.

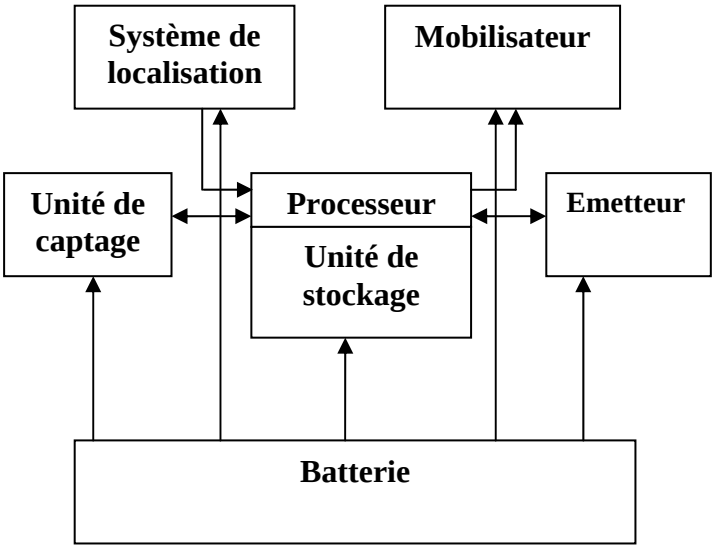
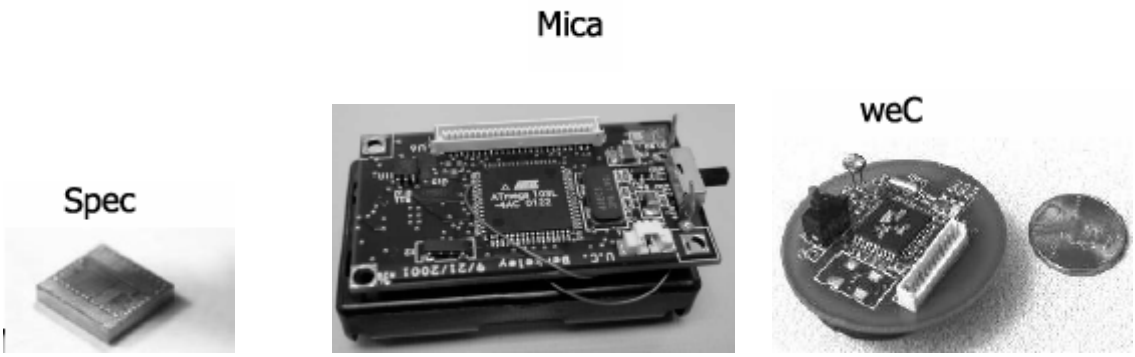


Figure 1 : composantes d'un nœud capteur.

La figure 2 présente quelques plateformes existantes, mica (b) inclut un émetteur-récepteur à basse consommation énergétique, un sous-ensemble de gestion d'énergie et un microcontrôleur incorporé. Une autre plateforme de matériel plus avancée est un dispositif à microplaquette unique de CMOS qui intègre les possibilités de traitement, de stockage et de communication pour former un système complet de nœud. Ce nœud simple – appelé Spec - a des mesures de 2.5 millimètres X 2.5 millimètres, contient un microcontrôleur, émetteur, ADC, mémoire et moteur de chiffrement. L'image (c) présente le capteur de Berkeley (Crossbow sensor mote).



(a) (b) (c)

Figure 2 : exemples de quelques nœuds capteurs

1.3.2 Communication dans les réseaux de capteurs sans fil

Les nœuds des capteurs sont habituellement dispersés dans un terrain de captage comme le montre la figure 3. Les noeuds de capteurs ont des possibilités à rassembler des données et de router ces données vers les utilisateurs. Ces données sont routées vers l'utilisateur par une architecture multi saut. La station de base (Sink) peut communiquer avec l'utilisateur final par l'intermédiaire de l'Internet ou Satellite.

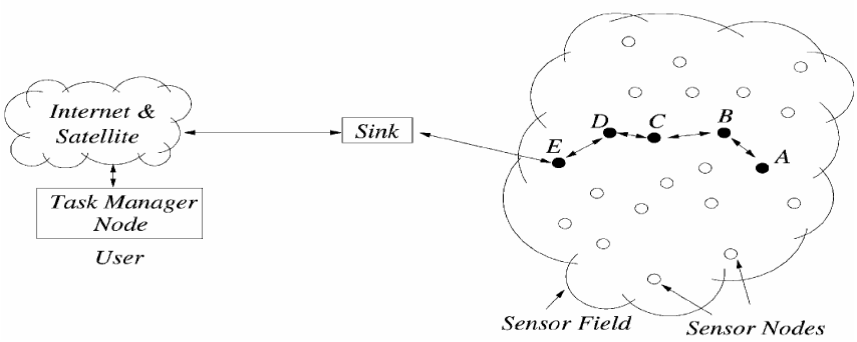


Figure 3 : dispersement des nœuds capteurs dans un terrain

1.3. 3 Liaisons sans fil

Les moyens de transmission entre les capteurs et la station de base sont trois : optique, infrarouge et radio [2].

a) Liaisons optiques (laser) : La communication optique a comme avantages une consommation énergétique moins élevée que la radio et une sécurité de transmission élevée, puisque il n'y a pas de diffusion. La technologie optique ne nécessite pas l'emploi d'une antenne, mais nécessite une visibilité directe entre l'émetteur et le récepteur. Un autre inconvénient est que cette communication est sensible aux conditions atmosphériques. La communication par liaison optique est directionnelle et elle n'est pas bien adaptée aux réseaux de capteurs, qui sont souvent déployés dans des environnements hostiles.

b) Liaisons infrarouges : Une liaison infrarouge est en général directionnelle, mais des projets récents ont permis de développer des communications omnidirectionnelles. Le principal inconvénient de ce type de liaison est sa courte portée (environ 1m), et son principal avantage est qu'elle ne nécessite pas d'antenne.

c) Liaisons sur fréquences radio : La communication radio est basée sur les ondes électromagnétiques. Se distinguant par sa facilité d'utilisation, et très répandue sur le marché des outils de communications, la technologie radio présente une solution bien adaptée pour les réseaux de capteurs sans fil.

1.3.4 Modèle en couche

La pile de protocoles comprend la couche application, couche de transport, couche réseau, couche liaison de données, la couche physique. Ainsi qu'un module de gestion de l'énergie, module de gestion de mobilité, et un module de gestion des tâches (Figure 4).

Différents types d'applications peuvent être établis et employés sur la couche application. La couche réseau prend le soin de router les données fournies par la couche de transport, la couche MAC doit être capable de réduire au minimum la collision avec l'émission des voisins et de minimiser les retransmissions qui consomment beaucoup d'énergie. La couche physique offre une modulation simple et robuste, et des techniques de transmission et de réception. En outre, les modules de surveillent la puissance, le mouvement et la distribution des tâches.

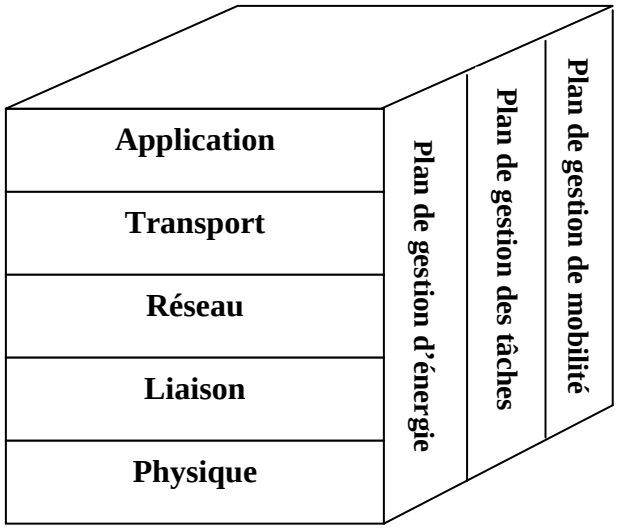


Figure 4: La pile de protocole d'un réseau de capteurs

1.3.5 Architecture des réseaux de capteurs sans fil :

Selon [3], il existe deux types d'architectures pour les réseaux de capteurs :

1.3.5.1 ***Les réseaux de capteurs sans fil plats :*** Un réseau de capteurs plat est un réseau homogène, où tous les nœuds sont identiques en termes de batterie et de complexité du matériel, excepté la station de base (sink) qui joue le rôle d'une passerelle et qui est responsable de la transmission de l'information collectée à l'utilisateur final.

Selon le service et le type de capteurs, une densité de capteurs élevée (plusieurs nœuds capteurs/m^2) ainsi qu'une communication multi saut peut être nécessaire pour l'architecture plate. En présence d'un très grand nombre de nœuds capteurs, la scalabilité devient critique. Le routage et le contrôle d'accès au médium (MAC) doivent gérer et organiser les nœuds d'une manière très efficace en termes d'énergie.

1.3.*5.2 Les réseaux de capteurs hiérarchiques :* l'architecture hiérarchique réduit le coût et la complexité de la plus part des nœuds capteurs en introduisant un ensemble de nœuds capteurs peu coûteux et plus puissants, ceci en créant une infrastructure qui décharge la majorité des nœuds simples à faible coût de plusieurs fonctions du réseau. L'architecture hiérarchique est composée de multiples couches: une couche de capteur, une couche de transmission et une couche de point d'accès.

1.4 Les services:

La plupart des applications de réseaux de capteurs à grande échelle partagent des caractéristiques communes. Des services tels que la synchronisation, la découverte d'endroit, l'agrégation de données, le stockage de données, la gestion de topologie, et le routage de messages sont utilisés par ces applications.

1.4.1 La Synchronisation

La synchronisation est un service essentiel dans les réseaux de capteurs sans fil [4]. Afin de coordonner correctement leurs opérations pour réaliser le captage, des nœuds capteurs doivent être synchronisés.

Une horloge globalement synchronisée permet aux nœuds d'estampiller correctement les événements détectés. La chronologie, la durée, et la période appropriée entre ces événements peuvent alors être déterminées. En cas d'estampillage incorrect, dûe aux facteurs tels que la dérive d'horloge, la station de base va réunir les paquets reçus dans un ordre chronologique incorrect.

Les noeuds peuvent conserver la vie des batteries en se mettant hors tension. Une fois correctement synchronisés, les noeuds peuvent s'allumer simultanément. Une autre fois mis sous tension, les noeuds peuvent transmettre des messages à la station de base et plus tard se mettre hors tension encore pour conserver l'énergie. Les noeuds non synchronisés ont comme conséquence l'augmentation du délai, tandis qu'ils attendent des noeuds voisins pour allumer leurs radios, et dans le plus mauvais cas, des messages transmis peuvent être perdus.

1.4.2 La localisation :

La découverte d'endroit est de trouver l'information de position des capteurs, exprimée comme des coordonnées globales. Elle sert de base fondamentale aux services de réseau de capteurs additionnels où la connaissance d'endroit est exigée, comme le routage de messages. En outre, dans les applications telles que la détection du feu, il n'est pas suffisant de déterminer si un feu est présent car le lieu de l'incendie dans ce cas est très important.

1.4.3 Agrégation De Données

L'agrégation de données et la diffusion de requêtes ont une grande importance dans les réseaux de capteurs [5]. Les noeuds ont typiquement la contrainte d'énergie. Par conséquent, il est souhaitable de réduire au minimum le nombre de messages transmis, parce que les transmissions par radio peuvent rapidement consommer la puissance de batterie. Une approche naïve pour rapporter des phénomènes sentis est que toutes les informations captées sont transmises à une station de base pour l'analyse et le traitement à l'extérieur. Cependant, puisque les capteurs dans la même proximité détectent souvent la même chose, il est probable qu'une certaine redondance dans les informations captées se produira [6]. La collaboration locale permet à des noeuds voisins de filtrer et traiter les informations captées avant de les transmettre à la station de base. En conséquence, ce processus peut réduire le nombre de messages transmis à la station de base.

Divers types d'agrégation de données sont possibles, selon le niveau d'amélioration désiré. Le traitement au niveau du réseau peut être conçu pour effectuer un ou plusieurs des opérations suivantes :

- Agréger les données dans une valeur binaire simple. (c.-à-d., vrai ou faux). Une valeur booléenne serait suffisante pour indiquer par exemple si un animal était détecté ou pas.
- Agréger les données collectées dans une zone, à base de coordonnées qui définissent la région.
- Agréger les données rassemblées en appliquant une agrégation spécifique ou une fonction de filtrage. Comme exemple, la moyenne, le maximum, le minimum, ou la somme de valeurs captées, peuvent être calculés en route avant l'expédition de n'importe quelle information reçue.

La conservation d'énergie, en raison de l'agrégation de données, est un intérêt particulier pour les capteurs près de la station de base. Sans n'importe quelle forme d'agrégation de données, un plus grand nombre de messages est transmis. En conséquence, leurs batteries sont épuisées rapidement. Par la suite, quand des noeuds qui communiquent directement avec la station de base s'arrêtent de fonctionner, le réseau de capteurs est devenu inutile, indépendamment de la puissance restante d'autres nœuds, puisque aucun message ne peut atteindre la station de base.

L'agrégation de données cherche à combiner des données arrivant de différentes sources en route. Dans [6], les auteurs étudient l'épargne d'énergie et les différences de latence provoquées par l'agrégation de données, et comment les facteurs tels que la densité de réseau et l'emplacement de la source (c.-à-d., événement) et la station de base affectent cette différence. Une analyse de complexité de l'agrégation optimale de données dans des réseaux de capteurs est également réalisée, et elle a bien montré que l'agrégation optimale de données est NP- complexe.

1.4.4 Stockage De Données

Le stockage de données présente un défi unique aux réalisateurs. L'information d'événement rassemblée par différents noeuds doit être stockée à certain endroit.

Dans certains cas, où une zone de stockage off line n'est pas disponible, les données doivent être stockées dans les réseaux de capteurs. Trois paradigmes de stockage de données employables dans les réseaux de capteurs sont répertoriés [7, 8]:

1. **Stockage Externe** : quand un noeud détecte un événement, les données correspondantes sont transmises à une station du stockage externe situé en dehors du réseau. L'avantage de cette approche est que les requêtes posées au réseau n'encourent aucune dépense énergétique puisque toutes les données sont déjà extérieurement stockées.
2. **Stockage Local** : Dans ce modèle, quand un noeud détecte un événement, l'information d'événement est stockée localement au noeud. L'avantage de cette approche est qu'aucun coût initial de communication n'est encouru. Les requêtes sont diffusées à tous les noeuds. Les noeuds qui ont l'information désirée transmettent leurs données à la station de base.
3. **Stockage Donnée- Central** : l'information d'événement est routée à un endroit prédéfini, indiqué par une fonction d'informations géographique (GHT), dans le réseau de capteurs. Les requêtes sont dirigées vers le noeud qui contient l'information appropriée, qui transmet la réponse à la station de base pour une transformation plus ultérieure.

1.4.5 Gestion de topologie :

Selon Tilak et al [9], le déploiement dense des nœuds rend le réseau plus efficace et si la topologie n'est pas soigneusement contrôlée, ceci peut mener à un grand nombre de collisions et à un encombrement potentiel du réseau. En conséquence, il y a une plus grande latence et une réduction de l'efficacité énergétique globale du réseau.

1.4.5.1 Auto configuration

L'auto configuration consiste à élire un chef, à créer une topologie du réseau, effectuer les opérations nécessaires à la mise en œuvre de l'objectif du réseau de capteurs. Le premier choix à effectuer est certainement celui de l'adressage.

Plusieurs techniques en vue d'optimisation de l'énergie, peuvent intervenir dans la mise en place d'un réseau de capteurs : clusterisation, auto-configuration et auto-adressage.

1.4.5.2 Clusterisation

L'agrégation de nœuds en clusters permet de réduire la complexité des algorithmes de routage, de faciliter l'agrégation des données, de simplifier la gestion du réseau et en particulier l'affectation d'adresses, d'optimiser les dépenses d'énergie et enfin de rendre le réseau plus scalable.

L'utilisation de clusters permet aussi de stabiliser la topologie si les tailles de clusters sont grandes par rapport aux vitesses des nœuds.

On peut classer les algorithmes de formation de clusters en :

- Implicites (les nœuds se constituent en groupes), ou explicites (les nœuds se constituent en groupes autour d'un chef) ;
- Actifs (les clusters sont le résultat de l'exécution d'un protocole dédié), ou passifs (les clusters se forment en déduisant des informations sur la topologie du réseau et en écoutant les messages MAC utilisés pour transmettre le trafic de données) ;
- Hiérarchiques (clusters de clusters), ou non hiérarchiques ;
- Centralisés ou distribués, les algorithmes distribués étant éventuellement émergents s'ils permettent d'obtenir un résultat global prévisible de manière déterministe ou stochastique.

1.4.5.3 L'auto adressage

L'attribution d'une adresse unique à un nœud doit se faire de manière automatique. Une classification de différents mécanismes d'auto adressage était proposée [10], elle les sépare en deux grandes familles : les protocoles de types stateful et ceux de type stateless.

Les protocoles de type stateful se répartissent en trois sous-ensembles :

1) ***CAC (Centralized Auto Configuration) :*** Le protocole du premier type le plus connu est certainement le protocole DHCP (Dynamic Host Configuration Protocol), et il n'est pas applicable aux réseaux ad hoc parce que le serveur DHCP doit être directement atteint par un nœud. Dans [11], les auteurs proposent un protocole d'auto configuration qui utilise une table d'allocation centralisée, où l'initiateur joue le rôle d'agent d'adressage (AA), qui est élu d'une manière dynamique. Il maintient la table d'allocation qui contient la liste des adresses déjà affectées, et la correspondance avec l'adresse MAC.

2) DAC (Distributed Auto configuration): Assure l'unicité d'une adresse en maintenant une table d'allocation des adresses en cours d'attribution au niveau de chaque nœud.

3) ***DAT (Disjoint Allocation Tables):*** Le principe est de proposer à chaque nœud de gérer lui même une partie de la table d'allocation globale. De ce fait, il dispose d'adresse à affecter sans avoir à attendre la permission d'un autre nœud et sans avoir à mettre en place un mécanisme de synchronisation d'affectation des adresses.

Les protocoles stateless n'utilisent pas la table d'allocation et leur fonctionnement est décentralisé. Ils utilisent à la place une adresse choisie aléatoirement ou basée par exemple sur un numéro de série.

La différence entre les différents protocoles de ce type se fait donc sur la performance de la procédure DAD (Duplicated Address Detection), qui détecte la collision d'adresses et garantie l'unicité de l'adresse choisie. Trois sous-familles sont généralement citées : QDAD, WDAD, PDAD.

1) QDAD (Query DAD)

Un nœud désirant obtenir une adresse, commence par s'en attribuer une de manière aléatoire. Et interroge par la suite les autres nœuds, si cette adresse est déjà prise ou non (message AREQ). Si c'est le cas, le nœud qui l'utilise répondra avec un message AREP en mode unicast.

2) WDAD (Weak DAD)

Proposé par Vaidya [12], chaque nœud génère une clé d'initialisation et la distribue en même temps que son adresse dans tous les paquets de routage. Si un nœud reçoit un paquet de routage qui contient une adresse qui fait partie de sa table de routage, il compare les deux clés. Et si elles sont différentes, une collision d'adresse est détectée.

3) PDAD (Passive DAD)

Il s'agit d'arriver à déduire qu'il y a eu une collision d'adresse, en observant uniquement les tables de routages. L'avantage de cette méthode est qu'il n'y a pas de surplus dû au mécanisme DAD.

1.4.6. Routage de messages :

Les réseaux de capteurs peuvent contenir des centaines ou des milliers de nœuds. Des protocoles de routage doivent être conçus pour réaliser un degré acceptable de tolérance aux fautes à la présence des défaillances de nœuds, tout en réduisant au minimum la consommation d'énergie.

Plusieurs protocoles de routage sont discutés dans la littérature, Iwata et al [13] classent trois catégories de protocoles de routages dans les réseaux de capteurs :

- Le routage global pré-calculé où tous les chemins sont pré-calculés et maintenus par les mises à jour périodiques.
- routage sur demande où les routes sont calculées seulement quand il est nécessaire.
- inondation (Flooding) où un paquet est envoyé à toutes les destinations, s'assurant qu'il atteint la destination prévue.

Le routage global pré-calculé peut être plat ou hiérarchique, Il inclut la plupart des techniques du vecteur de distance et d'état de lien, comprenant DSDV [14] et WRP [15]. Des techniques hiérarchiques sont utilisées pour permettre le passage à l'échelle. Dans le routage hiérarchique, les nœuds agissent en tant que commutateurs ou points finaux et s'organisent en clusters. Le routage hiérarchique réduit considérablement la taille des tables de routage et les rend scalables. Cependant, elles font face à beaucoup de problèmes tels que la nécessité d'assurer que les identifications de clusters, qui se produisent dynamiquement, sont uniques. Les algorithmes de routage sur demande incluent LMR [16] et TORA [17]. Ce type de routage est basé sur une approche de requête /réponse.

1.5 Conclusion :

Ce chapitre présente un état de l'art sur les réseaux de capteurs sans fil, en passant en revue les concepts principaux, les domaines d'application, les architectures, ainsi que les services utilisés par les applications.

Des propriétés comme flexibilité, prix réduit et facilité de déploiement offrent de nombreuses possibilités de développement dans tous les domaines d'applications, et la nature de ce type de réseau présente un domaine vaste contenant plusieurs défis et axes de recherches.

Chapitre 2 :

Les techniques de tolérance aux fautes dans les réseaux de capteurs sans fil

2.1 Introduction :

Un aspect fondamental dans le design de réseaux de capteurs sans fil (RCSF) est la fonctionnalité [18]. Un RCSF serait fonctionnel s'il y a un chemin de communication entre n'importe quelle paire de capteurs non défectueux dans le réseau. La notion de la fonctionnalité de réseau et la tolérance aux fautes dépendent ainsi fortement de la connectivité de réseau. Avec précision, un RCSF serait tolérant aux fautes s'il demeure fonctionnel malgré l'occurrence des défaillances des capteurs.

2.2 Sûreté de fonctionnement de RCSF :

Un système défaille lorsque le service qu'il délivre diverge du service attendu. La sûreté de fonctionnement cherche donc à éviter les défaillances, ou au moins à prévenir les plus catastrophiques. De ce fait, la caractéristique tolérance aux fautes est fortement liée avec la sûreté de fonctionnement de système [19]. La sûreté de fonctionnement d'un système est définie comme la propriété de ce système qui permet aux utilisateurs de placer une confiance justifiée dans la qualité service qu'il leur délivre, et s'articule autour de quatre propriétés [19,20, 21] : Disponibilité, et elle est définie comme propriété qu'un système est prêt à être employé immédiatement. La fiabilité, est la propriété qu'un système peut fonctionner de manière continue sans défaillances. La maintenabilité, qui est l'aptitude d'un système à être réparé et la sécurité qui assure la confidentialité et l'intégrité des informations. Pour les applications de RCSF, les données captées doivent être livrées à la station de base d'une manière permanente et dans les délais.

a) La fiabilité : Le réseau de capteur est fiable si :

- il est robuste contre la défaillance de nœud (manque d'énergie, destruction physique …)

- le transfert fiable de données (par exemple limiter ou tolérer les pertes des paquets)

La fiabilité d'un réseau de capteurs est liée avec sa durée de vie, et comme cette durée dépend de l'application, le réseau devrait accomplir ses tâches aussi longtemps que possible pour satisfaire les besoins de l'application.

Les défis à assurer la fiabilité sur les réseaux de capteurs sans fil, peuvent être divisés en trois catégories principales [22]; les premiers problèmes sont liés à la communication sans fil. L'asymétrie des liens rend l'évaluation de qualité de lien difficile, et les pertes corrélées dûes aux obstacles, interférence, peuvent mener à pertes consécutives, diminuant l'efficacité du code d'effacement (erasure code). La deuxième sorte de problèmes vient des ressources limitées en énergie. Les capteurs ont également une capacité limitée de traitement et de stockage. A la fin, et d'un point de vue de génie logiciel, la couche réseau ajoute plus de défis. Et à cause de contrainte d'énergie, les réseaux de capteurs doivent utiliser des protocoles de routage optimisés et à basse consommation.

b) Disponibilité :

La disponibilité d'un réseau de capteurs est fortement liée avec sa qualité de service, le service de RCSF doit être "bon", c-à-d la bonne réponse au bon temps.

c) Maintenabilité:

- le réseau de capteurs doit s'adapter aux changements, par l'auto-contrôle, auto-guérison et auto-stabilisation.

- le réseau doit permet l'incorporation des ressources additionnelles, par exemple, des nœuds nouvellement déployés.

d) Sécurité (au sens safety) :

Pour certaines applications (systèmes de contrôle), le réseau de capteurs est lié avec des déclencheurs, et dans ce cas, cette propriété devient très importante et un mauvais fonctionnement, calcul incorrect par exemple, peut avoir des conséquences catastrophiques.

e) Sécurité :

Le réseau de capteurs doit assurer la protection de la collection et la diffusion de données (résilience aux attaques).

2.3 Les fautes et les défaillances en RCSF :

Lorsqu'on augmente le nombre de composants d'un système réparti, la possibilité qu'un ou plusieurs de ces composants tombe en panne augmente également, et lorsqu'on réduit le coût de fabrication des composants pour obtenir des économies d'échelle, on accroît également le taux de défauts potentiels. Enfin, lorsqu'on déploie les composants du système dans un environnement que l'on ne contrôle pas nécessairement, les risques de pannes deviennent impossibles à négliger.

Dans les réseaux de capteurs, ces trois facteurs principaux sont combinés. Aussi, il devient certain que des pannes se produiront au cours de la vie du système. En effet, les pannes peuvent avoir des répercussions importantes sur la durée de vie du réseau et sur l'application qu'on exécute sur le système. Ces répercussions dépendent de la sévérité de la panne.

2.3.1 Notions de faute, d'erreur et de défaillance :

Prévenir les défaillances requiert d'introduire deux nouvelles notions : faute et erreur (fig 5,6).

Faute : Toute cause (événement, action, circonstance) [23] pouvant provoquer une erreur menant à une défaillance de système tel qu'il ne se comporte plus d'une manière pré-spécifiée [24].

Erreur : L'erreur est la partie de l'état du système susceptible d'entraîner la défaillance, qui est causée par une faute [20,25]. S'il y a une erreur dans l'état de système, alors il existe une séquence d'actions qui peut être exécutée par le système et qui mènera à la défaillance du système [26].

Défaillance : La défaillance dénote l'incapacité d'un composant d'exécuter sa fonction en raison des erreurs dans l'élément ou dans son environnement [27]. Quatre sources de fautes peuvent causer la défaillance d'un système :

- Une spécification inadéquate
- Des erreurs de conception dans le logiciel
- Une défaillance de processeur
- Une interférence sur le sous-système de communication

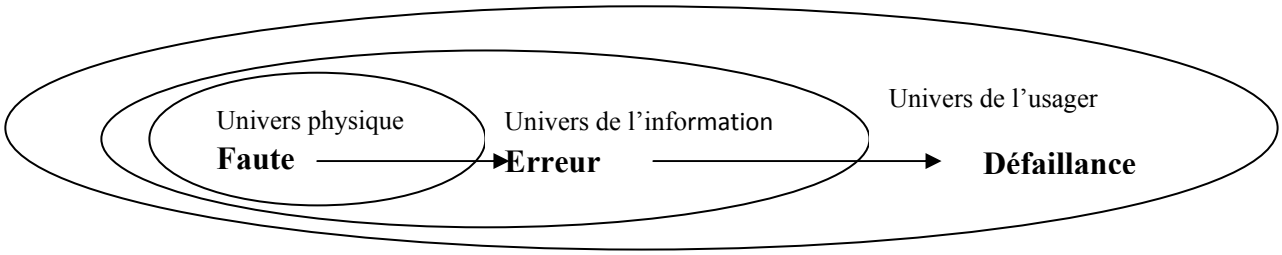


Figure 5 : de la faute à la défaillance

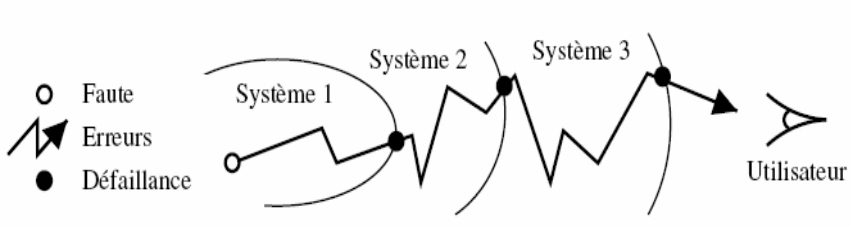


Figure 6 : Récursivité des fautes, erreurs et défaillances

2.3.2 Les sources de fautes dans les réseaux de capteurs :

Les réseaux de capteurs sont sujets à une grande variété de fautes et à un manque de fiabilité. Le matériel peu coûteux, les ressources limitées, et les extrêmes conditions environnementales contribuent tous à causer ces fautes. Plusieurs sources peuvent endommager le bon fonctionnement d'un réseau de capteurs [28] :

a) Les phénomènes atmosphériques

Les changements du climat modifient la propagation des signaux, qui peut alternativement causer des erreurs de communication pendant que la force de signal diminue [29]. Plusieurs conditions environnementales telles que l'humidité, la température, entre autres, modifient la propagation de signal. Pendant que le climat change constamment, la qualité de communication change avec le temps.

b) Les sources mobiles d'interférence

D'autres dispositifs fonctionnant aux fréquences semblables, où même les véhicules, les animaux et les humains peuvent interférer les nœuds communicants [30]. Pendant que les RCSF utilisent habituellement les fréquences d'ISM (industriel, scientifique et médical), qui n'exigent pas l'autorisation pour l'opération, les RCSF sont exposées aux interférences d'autres dispositifs fonctionnant dans ces fréquences. Afin de diminuer le coût de communication, les nœuds capteurs utilisent habituellement un seul canal radio, avec un schéma fixe de modulation. Ces contraintes gênent l'utilisation du choix dynamique des fréquences et la modulation, la détection des canaux de base interférence et les méthodes du saut de fréquence [31, 32].

***g) Les catastrophes naturelles* :**

Les nœuds capteurs peuvent être déployés dehors ou dans des endroits de désastre, ainsi ils peuvent être exposés aux éboulements, aux inondations et aux tremblements de terre. Ces événements peuvent causer la destruction massive des nœuds capteurs en endommageant de manière permanente des composants matériels. À la différence des défaillances provoquées par des phénomènes atmosphériques, les nœuds échoués dus aux désastres naturels seront de manière permanente non- opérationnels.

***h) Accidents de rupture* :**

Les nœuds de capteur peuvent être accidentellement détruits, par exemple en raison des animaux piétinant ces nœuds, ou des arbres en chute. Puisque les nœuds seront plusieurs mètres partis de l'un l'autre, seulement un nœud tend à échouer à un moment donné.

e) La panne de processeur :

Le logiciel d'application peut contenir des erreurs de programmation, qui pourraient mener le processeur aux situations de panne, et à cause de fautes de synchronisation, le logiciel défectueux peut bloquer le processeur. Pour éviter de telles situations, les capteurs (microcontrôleurs) utilisent watchdogs2, qui reprise le processeur si une erreur de fonctionnement de logiciel se produit. Ainsi, les nœuds seront indisponibles pour une durée de temps finie, puis reviendront à l'opération normale.

f) Les défaillances malveillantes :

Les RCSF sont enclins aux défaillances malveillantes dûes aux attaques destinées à perturber l'opération des réseaux, provoquée par un étranger ou par un nœud corrompu. Cependant, quelques attaques de sécurité peuvent être évitées ou partiellement récupérées avec l'utilisation des techniques de tolérance aux fautes qui évitent les routes passantes par des zones sous l'attaque [33]. Il y a aussi des techniques de tolérance aux fautes pour éviter les attaques de déni de service, comme les attaques d'interférence, attaques de collision, et attaques de trou de la station de base (sinkhole). Ces attaques se comportent comme des fautes silencieuses.

g) L'épuisement d'énergie :

La diminution d'énergie peut produire des pannes de communication. Habituellement, les batteries ne seront pas remplacées, puisque les réseaux de capteurs sont utilisés dans les environnements durs où la présence d'un opérateur est interdite, ainsi que le grand nombre de nœuds déployés fait du remplacement de batterie une tâche difficile.

Puisque les capteurs ont des ressources limitées, les nœuds courants ne permettent pas une mesure fiable de la quantité d'énergie restante stockée dans les batteries; par conséquent, les nœuds ne peuvent pas informer leurs voisins quand leur réserve d'énergie est presque complètement consommée.

2.3.3 Classification de fautes et défaillances :

Deux points de vue [34], pour classer les fautes dans un système réparti, sont définies sur l'occurrence et l'origine de fautes. La première se base sur la localisation dans le temps, et selon ce critère, trois types de fautes sont possibles: permanentes, intermittentes et transitoires. De point de vue de l'origine de fautes, il y a les fautes physiques qui proviennent des phénomènes physiques internes, comme le changement de seuil, ou les changements externes tels que l'électromagnétisme ou les vibrations, ainsi que les fautes humaines dùes aux actions involontaires ou intentionnelles [35]. Selon le comportement du système après l'occurrence d'une faute, des défaillances ont été classées [36, 25, 37] comme suit :

Fail – stop : se produit quand un processus cesse son exécution et alerte d'autres processus.

Défaillances d'omission : soit un processus émetteur envoyant une séquence de messages à un processus récepteur. Si le récepteur ne reçoit pas certains messages envoyés par l'émetteur, alors une défaillance d'omission se produit [36].

Défaillance de synchronisation : des systèmes temps réel exigent des actions d'être terminées dans un temps spécifique [38].

Défaillances de calculs incorrects : se produit quand un processus ne produit pas le résultat correct en réponse à des entrées correctes [39]

Les types de fautes qui peuvent affecter l'exactitude et l'exécution d'un réseau de capteurs sont la panne de nœud, et la faute de matériel, les fautes de communication et les fautes de logiciel [40].

a) Panne de nœud et fautes de matériel :

Pendant le déploiement, les capteurs peuvent être lâchés à des altitudes élevées et quelques nœuds peuvent ne pas survivre après la chute. Dans certains cas, l'unité de captage peut devenir décollée du nœud, en raison de l'impact au sol et cause des fausses alarmes continues, ou pertes, ayant pour résultat le comportement apparemment byzantin. Quelques nœuds peuvent manquer de puissance à cause des ressources énergétiques limitées et surtout s'il n'y a pas de possibilité de recharger ou de remplacer la batterie. Des nœuds peuvent être déplacés de leurs positions originales par les cibles elles-mêmes ou en raison des facteurs

environnementaux. Les capteurs peuvent devenir désensibles, à cause de la chaleur ou de l'humidité et rapporter des fausses lectures.

Alors, nous voyons que la panne de nœud et l'épuisement d'énergie mènent à la défaillance Fail – stop, et les fautes de matériel peuvent causer des défaillances byzantines.

b) Fautes de communication :

Le Broadcast ou la diffusion, le primitif de base de communication dans le réseau de capteurs sans fil, mène aux pertes de messages à partir des collisions quand deux nœuds dans le même espace les transmettent simultanément. Même si les nœuds ne sont pas dans l'espace de transmission d'un autre nœud, des messages peuvent se heurter et perdus au niveau du récepteur en raison de l'effet de terminal de réception caché. Même en l'absence des collisions, des messages peuvent encore être perdus en raison de l'effacement pendant la propagation au-dessus du milieu sans fil. La distance entre les nœuds, la différence d'altitude, la polarisation d'antenne, les conditions environnementales, et la présence des obstacles sont tous des facteurs qui contribuent aux caractéristiques d'effacement d'un lien sans fil. De ce fait, cette classe de fautes mène à la défaillance d'omission.

a) Fautes de logiciel :

Les ressources limitées du calcul, disponibles sur un nœud, imposent quelques restrictions à la quantité de traitement qui peut être effectuée avec succès. Si cette limite est dépassée, le traitement des tâches peut ne pas achever, causant un comportement non déterministe et divers genres de défaillances. Les pointeurs et les emplacements de mémoire peuvent devenir corrompus, les tampons des messages peuvent être débordés, et en conséquence certaines informations captées ou traitées pourraient se perdre.

On peut conclure que les défaillances qui peuvent être causées par cette catégorie sont les défaillances de synchronisation, de calculs incorrects et byzantins.

2.4 Tolérance aux fautes dans les réseaux de capteurs :

Un grand effort de recherche est consacré aujourd'hui à la fiabilité et la tolérance aux fautes dans les réseaux de capteurs sans fil: dans le déploiement, la construction de topologie, l'agrégation de données, le routage, car les différents services et applications devraient être robustes et tolérants aux fautes.

La tolérance aux défaillances dans les réseaux de capteurs inclut des issues comme: l'intégration d'information en temps réel, la transmission et l'agrégation des données sans encourir de lourdes pertes; la tolérance aux fautes de la topologie de réseau, de schéma de routage et de schéma d'agrégation en présence de nœuds morts.

2.4.1 La Tolérance aux fautes :

La tolérance aux fautes est l'ensemble de techniques de conception qui permet à un système de continuer à fonctionner et à fournir ses services même en présence de fautes.

La tolérance aux fautes consiste à protéger l'utilisateur externe des conséquences des défaillances de composants internes du système [23]. Cette protection peut prendre plusieurs formes sur la base de masquer complètement les fautes à l'utilisateur, à assurer une continuité de service en acceptant une dégradation temporaire de sa qualité, ou à prévenir les conséquences les plus catastrophiques d'une défaillance en mettant le système dans un état sûr.

La tolérance aux fautes vient de la redondance d'espace, de temps et de valeur [41, 42] :

1- **redondance d'espace** : plusieurs copies du même composant (nœuds capteurs)
2- **redondance de temps** : répéter l'action (envoyer des copies multiples du message)
3- **redondance de valeur** : ajouter des données supplémentaires (codes corrigeant des erreurs, signatures)

2.4.1.1 Les mécanismes de tolérance aux fautes :

Les mécanismes de tolérance aux fautes font typiquement intervenir les deux primitives suivantes [23]:

a) La détection d'erreur : permet au système d'identifier un état erroné au cours de son fonctionnement. La détection s'accompagne éventuellement d'un diagnostic pour évaluer le degré de propagation de l'erreur, et l'importance des dommages causés.

b) Le recouvrement d'erreur : consiste à ramener le système vers un fonctionnement sûr (mais éventuellement dégradé) en remplaçant l'état erroné et donc potentiellement dangereux, par un état sans erreur.

On distingue trois formes de recouvrement :

b.1) Recouvrement par reprise : le système est ramené dans un état antérieur à partir duquel il reprend son fonctionnement (un exemple de la vie quotidienne consiste à rallumer son ordinateur de bureau après qu'il se soit bloqué, et à travailler à partir de la dernière version sauvegardée de son travail)

b.2) Recouvrement par poursuite : le système est amené dans un nouvel état, connu à priori, à partir duquel il peut continuer son fonctionnement (souvent de manière dégradée). Ce mode de recouvrement est souvent très dépendant de l'application.

b.3) Recouvrement par compensation : la redondance contenue dans l'état erroné suffit à éliminer l'erreur et à poursuivre l'exécution.

2.4.2 La détection de fautes et le recouvrement dans les réseaux de capteurs :

Pour traiter les fautes dans un réseau de capteurs, le système devrait suivre deux étapes principales. La première l'étape est la détection de fautes, elle doit détecter qu'une fonctionnalité spécifique est défectueuse, et à prévoir qu'elle continuera à fonctionner correctement dans un proche avenir. Après que le système détecte une faute, le recouvrement de cette faute est la deuxième étape.

Il y a deux types de techniques de détection fondamentaux: l'auto-diagnostic et la diagnostic coopératif. Quelques fautes qui peuvent être déterminées par un nœud capteur lui-même peuvent adopter la détection d'auto-diagnostic. Par exemple, des fautes provoquées par épuisement de batterie peuvent être détectées par un nœud capteur lui-même. La charge de batterie restante peut être prévue en mesurant la tension courante de batterie. Un autre

exemple est la détection de défaillance des liens, il peut détecter que certain lien à un de ses voisin est défectueux si le nœud ne reçoit aucun message du voisin dans un intervalle prédéterminé. Cependant, il y a quelques genres de fautes qui exigent le diagnostic coopératif parmi un ensemble des nœuds capteurs. Une grande partie des fautes dans les réseaux de capteurs sans fil sont dans cette catégorie. Par exemple, la méthode de détection proposée dans [43] est d'identifier des nœuds capteurs défectueux dans l'application de détection d'événement. La méthode de détection est fondée sur l'hypothèse que les nœuds capteurs dans la même région devraient avoir des valeurs captées semblables. La méthode prend des mesures de tous les voisins d'un nœud et emploie les résultats à calculer la probabilité du nœud défectueux.

La technique la plus généralement utilisée pour le recouvrement d'une faute est la réplique ou la redondance des composants qui sont enclins pour être défectueux. Par exemple, les RCSF sont habituellement utilisés pour surveiller périodiquement une région et envoyer les données captées à une station de base, si quelques nœuds ne fournissent pas des données, la station de base réceptionne toujours des données suffisantes. Le routage multi chemins est un autre exemple; dans le cas d'une seule route, une requête ou les données ne peuvent pas être routées si quelques nœuds ou liens le long de la route échouent. Garder un ensemble des chemins de réserve fournit la fiabilité élevée des routes pour le routage.

2.4.3Tolérance aux fautes dans les différents niveaux de réseau de capteurs:

La tolérance aux fautes dans les réseaux de capteurs peut être étudiée sur quatre niveaux d'abstraction [44] à savoir : matériel, logiciel, réseau de communication et application, et dans quatre axes qui sont : les composants d'un capteur, le nœud capteur individuel, le réseau et le système distribué dans lequel nous contribuons dans cette thèse.

1. Niveau matériel :

Les défaillances au niveau matériel peuvent être provoquées par le mauvais fonctionnement de n'importe quel composant matériel d'un nœud capteur, tel que la mémoire ou l'unité de stockage, la batterie, le microprocesseur, l'unité de captage, et l'interface de réseau (radio sans fil). Il y a trois raisons principales qui causent les défaillances de matériel des nœuds capteurs. La première est que les réseaux de capteurs sont habituellement conçus pour des utilisations commerciales et les nœuds capteurs ont un coût sensible. Par conséquent,

la conception d'un capteur n'emploiera pas toujours les composants de la plus haute qualité. La seconde est la contrainte stricte d'énergie qui limite la fiabilité et la durée de vie des nœuds capteurs; à titre d'exemple, les informations captées par un capteur peuvent devenir incorrectes quand la batterie de celui-ci atteint un certain niveau [45]. Et la troisième est que ces systèmes sont souvent déployés dans des environnements durs et dangereux.

2. Niveau logiciel :

Le logiciel d'un nœud capteur contient deux composants : logiciel système, qui est le système d'exploitation, et middleware, qui assure la communication, le routage, et l'agrégation. Les défaillances de logiciels ont la même source d'erreur dans les réseaux de capteurs, et une méthode prometteuse est la diversité de logiciel, où chaque programme est mis en application dans différentes versions, dans l'espoir que les différentes versions n'auront pas les bogues identiques [46, 47]. Et puisqu'il est difficile de fournir la tolérance de fautes d'une manière économique au niveau de matériel d'un capteur, de nombreuses approches tolérantes aux défaillances sont prévues au niveau de middleware.

3. Niveau réseau de communication :

Les fautes au niveau de la couche de communication de réseau sont les fautes sur des liaisons sans fil. En supposant qu'il n'y a aucune erreur sur le matériel, les défauts de lien dans les réseaux de capteurs sont habituellement reliés aux environnements sans fil. En outre, les fautes de lien peuvent également être causées par l'interférence par radio des nœuds de capteurs. Par exemple, le nœud A ne peut pas recevoir un message du nœud B, avec succès, si le nœud A est dans le champ d'interférence des autres nœuds qui transmettent des messages en même temps. La manière standard pour augmenter la performance de communication sans fil est d'employer les schémas de correction d'erreur et la retransmission.

4. Niveau application :

La tolérance aux fautes peut être destinée également au niveau d'application. Par exemple, si on veut identifier une personne particulière, on peut essayer de mesurer en utilisant les capteurs de diverses métriques de cette personne. Chaque métrique, et probablement une combinaison de métriques, sera suffisante pour identifier cette personne. Cependant, une approche pour la tolérance aux fautes dans une application ne peut pas être directement appliquée à d'autres applications, et chaque application exigera une manière

adaptée pour aborder correctement la question, et la tolérance aux fautes dans ce niveau dépend fortement du type d'application, et à titre d'exemple, dans une application pour maintenir le niveau d'humidité et de la température dans une chambre, les capteurs ne peuvent pas être endommagés facilement ou interférés par l'environnement, et par conséquent, la condition de tolérance aux fautes est faible. Par contre, dans une application militaire pour la surveillance d'un champ de bataille, les données captées sont critiques et les capteurs peuvent être détruits par les ennemis, et dans ce cas, la condition de tolérance aux fautes est élevée.

2.5 Modèle d'intégration Tolérant aux fautes :

2.5.1. Modèle De Marzullo :

Dans son travail pour tolérer les mesures erronées de capteur [48], Keith Marzullo présente l'idée d'un capteur abstrait qui incorpore l'incertitude aux mesures d'un ensemble associé de capteurs. Son but est alors de construire un capteur abstrait de telle manière qu'il soit tolérant aux défaillances.

Dans ce modèle, un processeur reçoit des entrées de plusieurs capteurs dont les sorties sont les intervalles reliés, et donne un l'algorithme tolérant aux fautes qui prend ces intervalles comme entrées et donne le rendement comme intervalle relié représentant les valeurs captées. Si nous supposons que f capteurs sont défectueux d'un total de n capteurs, en conséquence on a au moins n – f capteurs corrects. Marzullo considère toutes les intersections non vides possibles (de n - f) des n capteurs et un capteur qui n'appartient à aucune de ces intersections (de n – f) est considéré défectueux. Un capteur correct chevauché avec au moins (n-f-1) autres capteurs corrects. Le plus petit intervalle relié contenant toutes les intersections (de n - f) est considéré comme le rapport de processeur et nous pouvons d'une manière concluante dire que cet intervalle contient la valeur physique réelle (actuelle) (Figure 7).

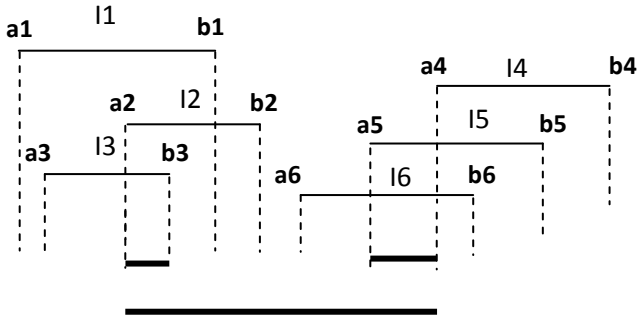


Figure 7: modèle de Marzullo

2.5.2. Modèle d'Iyengar :

Le modèle de Marzullo sert comme base pour le modèle d'Iyengar qui va plus loin en réduisant considérablement l'intervalle de sortie. Iyengar présente le concept de fautes irrégulières et régulières dans son modèle. Il déclare qu'un capteur peut défaillir d'une manière extravagante s'il n'y a aucune corrélation entre la valeur physique réelle mesurée et l'intervalle de capteur défectueux. D'autre part, la faute de capteur est régulière, bien que l'intervalle ne contient la valeur physique réelle, si l'intervalle de sortie se trouve de manière significative près de la valeur correcte qui doit être mesurée, à cause des vibrations et des phénomènes environnementaux transitoires ou permanents qui peuvent induire des fautes régulières.

Iyengar considère le cas où le nombre de capteurs intégrés (embarqués) est grand et la plupart des fautes sont régulières et que le nombre d'intervalles soit relativement grand. Des poids sont attachés à chaque intervalle, et le poids maximum donné à l'intervalle qui a eu la probabilité maximum de contenir la valeur physique. L'intervalle avec le poids maximum est lié par le plus petit intervalle possible et pris pour être l'intervalle de sortie (figure 8).

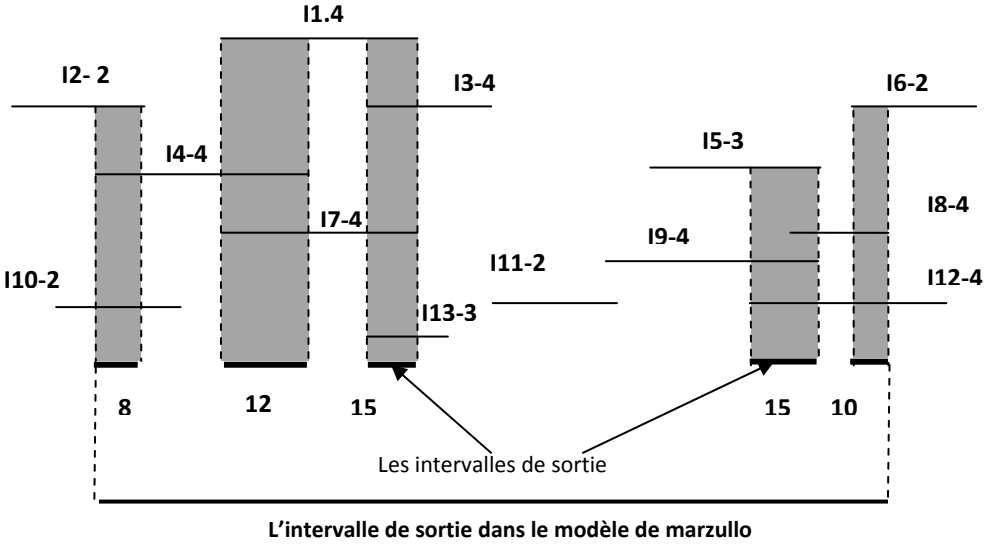


Figure 8 : modèle d'Iyengar

2.5.3 *Comparaison de modèle de Marzullo et d'Iyengar :*

Avec un nombre très grand de capteurs et en prenant les fautes régulières des capteurs en considération, le modèle d'Iyengar réduit les largeurs d'intervalle de sorties considérablement comparées à l'intervalle de sortie de Marzullo. En conséquence, la largeur

de l'intervalle de sortie est réduit de manière significative, où le nombre des capteurs impliqués est grand et la plupart des fautes sont régulières.

2.6 Tolérance aux fautes dans le routage :

Il y a plusieurs issues et contraintes qui gèrent la construction de n'importe quel routage dans un réseau de capteur sans fil. Ces issues résultent des limitations des capteurs elles-mêmes, la nature de la topologie, et l'environnement de déploiement.

Les capteurs ont des capacités limitées en énergie et traitement, une topologie grande, dense et dynamique à cause de défaillances des capteurs, ainsi qu'un environnement hostile et il n'y a pas de surveillance sur les capteurs, et il est difficile de remplacer les capteurs défaillants. Parmi ces contraintes, comme la fiabilité, Gafni and Bertsekas [49] proposent que les nœuds doivent avoir des chemins multiples vers la station de base, qui permettent à ces nœuds de choisir parmi ces chemins en cas où la route actuelle défaille.

2.6.1 Le routage multi chemins (multipath routing) :

La tolérance aux fautes d'un protocole de routage est mesurée par la probabilité qu'un autre chemin de routage existe entre une source et une destination quand le chemin primaire échoue [50]. Ceci peut être augmenté en utilisant des chemins multiples entre la source et la destination. Ces chemins de secours restent vivants en envoyant des messages périodiques.

Chang et Tassiulas [51] ont proposé un algorithme qui route les données par un chemin dont les nœuds ont la plus grande énergie résiduelle. Le chemin change toutefois lorsqu'un meilleur chemin est découvert. Le chemin primaire est employé jusqu'à ce que son énergie tombe au-dessous de l'énergie du chemin de secours pour lequel ce dernier est utilisé. En employant cette approche, les nœuds dans le chemin primaire n'épuisent pas leurs ressources énergétiques par l'utilisation permanente du même chemin, réalisant par conséquent la plus longue durée de vie.

Rahul et Rabaey [52] ont proposé l'utilisation d'un ensemble de chemins optimaux pour augmenter la durée de vie du réseau. Ces chemins sont choisis au moyen de probabilité, qui dépend de la consommation d'énergie de chaque chemin. Notant que le chemin avec la plus grande énergie résiduelle, si utilisé pour router les données dans un réseau, peut consommer beaucoup d'énergie.

Dans le travail de Dulman et al [53], le routage par chemins multiples a été employé pour augmenter la fiabilité d'un réseau de capteurs. Le schéma proposé est utile pour délivrer des données dans des environnements difficiles, et la fiabilité du réseau peut être augmentée par l'emploi de plusieurs chemins de source à la destination, et en envoyant le même paquet sur chaque chemin. Cependant, en utilisant cette technique, le trafic augmentera de manière significative.

Par conséquent, il y a une relation inversement proportionnelle (Tradeoff Relation) entre la quantité de trafic et la fiabilité du réseau. Cette relation est étudiée [53] en employant une fonction de redondance qui dépend du degré de chemins multiples et les probabilités des défaillances des chemins disponibles. L'idée est de couper le paquet original de données en sous-paquets et puis envoyer chaque sous-paquet par un des chemins disponibles. En conséquence, même si certains de ces sous-paquets étaient perdus, le message original peut être encore reconstruit.

La diffusion dirigée [54] est un bon choix pour un robuste routage par chemins multiples. Basé sur le paradigme de diffusion dirigée, un schéma de routage par chemins multiples qui trouve un ensemble de chemins partiellement disjoints, est étudié par D. Ganesan et al dans [55], ils ont trouvé que l'utilisation du routage par chemins multiples fournit le choix fiable pour le recouvrement efficace en énergie d'une défaillance dans un réseau de capteurs. La motivation d'utiliser ces chemins tressés est dûe à la nécessité de garder le coût de maintien des chemins multiples bas.

Les coûts de chemins de secours sont comparables au chemin primaire parce qu'ils tendent à être beaucoup plus près du chemin primaire.

2.6.2 Le problème de trou dans le routage:

Kewei Sha et al [56] présentent un protocole de routage appelé WEAR (Weighted Energy-Aware Routing Protocol) qui prend en compte les quatre facteurs qui affectent la politique de routage, à savoir *la distance à la destination*, *le niveau d'énergie de capteur*, *l'information globale d'endroit*, et *l'information locale de trou*. Tous ces facteurs sont mélangés et intégrés dans la notion du poids (Weight) dans le protocole WEAR. Kewei Sha & al [56] ont défini quatre conditions fondamentales pour tout protocole de routage :

a. Efficacité en Énergie : la différence la plus significative entre le réseau de capteur et le réseau traditionnel est la contrainte d'énergie des batteries des capteurs. Et les capteurs parfois sont déployés dans des endroits dangereux de sorte qu'il est impossible de recharger ces batteries. De ce fait, la consommation d'énergie a une grande considération et le protocole de routage efficace en énergie est exigé pour prolonger la durée de vie du réseau de capteurs.

***b. Equilibrage de la charge (load balancing)*:** la fonction principale du réseau de capteurs est la collecte d'information intéressante du champ de surveillance. Certaines applications comme la surveillance d'environnement ont besoin d'une grande durée de vie du réseau. Sans équilibrage de la charge, l'énergie de certains capteurs s'épuisera très rapidement ayant pour résultat une courte vie du réseau, et un bon protocole de routage devrait avoir un dispositif d'équilibrage de charge pour prolonger la vie du réseau de capteurs.

c. La tolérance aux fautes: à cause de la difficulté de remplacement des capteurs défaillants, et après que quelques capteurs tombent en panne, quelques trous se produisent dans le réseau et bloquent le routage; le protocole de routage devrait ainsi dévier du trou et empêcher le trou d'agrandir.

d. Scalabilité : une autre caractéristique du réseau de capteurs est son déploiement à grande échelle, qui peut se composer des milliers des nœuds. Et seul l'algorithme localisé peut aborder la question de scalabilité. Ainsi, un bon protocole de routage devrait employer seulement l'information localisée pour être scalable.

Kewei Sha et al [56] comparent plusieurs protocoles typiques de routage avec ces quatre conditions. Le plus court chemin est une manière de réaliser l'efficacité en énergie. *Le routage géographique* [57], comme GPSR (Geographical and energy aware routing), est une approche attrayante dans les réseaux de capteurs pour réaliser l'efficacité énergétique en utilisant le chemin le plus court et obtenir la scalabilité en employant seulement l'information locale. Cependant, il prend toujours le chemin local le plus court, ce qui fait un problème d'épuisement d'énergie des capteurs de ce chemin, et en cas d'exploitation du même chemin le plus court à coté de trou afin de le dévier, GPSR agrandit le trou très rapidement comme représenté dans la figure 9.

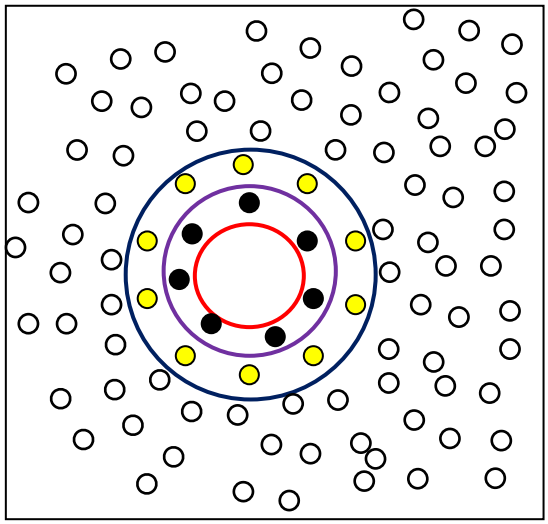


Figure 9: agrandissement de trou dans GPSR.

Dans la figure 9, les points vides dénotent les capteurs vivants, et les points foncés représentent les capteurs défaillants ou morts. Le plus petit cercle montre la forme du trou initial. Après la manipulation de quelques requêtes, les capteurs montrés en tant que points noirs défaillent, et le trou se prolonge au deuxième cercle et par la suite au plus grand cercle après que les capteurs jaunes tombent en panne.

Les protocoles de Routage Géographique GEAR (greedy perimeter stateless routing) [58] et GPSR (Geographical and energy aware routing) [59] sont deux protocoles de routage géographiques qui sont près de WEAR. Les deux ne considèrent pas l'information globale et l'information locale de trou. Ainsi, ils ne sont pas tolérants aux fautes et ne satisfont pas l'équilibrage de charge, et ils ne peuvent pas contrôler l'agrandissement de trou.

Le routage tolérant aux fautes exige que le réseau trouve un chemin de la source à la destination quand il y a quelques capteurs défaillants et contrôle également l'agrandissement du trou.

Le protocole WEAR réalise le but de la tolérance aux fautes par deux manières, déviant le trou par l'emploi de la règle right-hand proposée dans [60], et identifiant l'information de trou et la publier pour que les capteurs puissent éviter de router les messages vers le trou.

Le WEAR est construit sur plusieurs hypothèses. D'abord, l'information d'endroit est disponible par les dispositifs physiques comme GPS [61] ou l'algorithme de découverte de topologie [62], [63]. En second lieu, les endroits des capteurs et de station de base sont fixés.

Troisièmement, le routage multi-sauts est exigé pour livrer le message, et l'information d'énergie des voisins est disponible.

Le WEAR peut fonctionner de deux manières. Dans le premier cas, le routage est totalement basé sur la valeur du poids. Quand un capteur reçoit un message, il vérifie s'il est la destination ou pas. Si ce n'est pas la destination, il choisit la prochaine étape de sa liste de voisins avec le plus petit poids et fait passer le paquet. Pas à pas, le message est fourni de la source à la destination. Dans l'autre cas, le routage a deux modes, le mode avide et le mode déviant. En mode avide, il doit y avoir des voisins qui sont plus proches de la destination que le capteur courant. Alors le capteur courant fait passer le message au voisin ayant la plus petite valeur de poids. S'il n'y a aucun voisin plus près de la destination que le capteur courant, le routage entre dans le mode déviant. En mode déviant, le routage suit la règle de main droite (right-hand) proposée dans [60] pour expédier le message, jusqu'à ce que le message atteint un nœud plus près de la destination que l'endroit où le mode déviant commence. Cette deuxième approche garantit la livraison à la destination et elle est efficace en énergie.

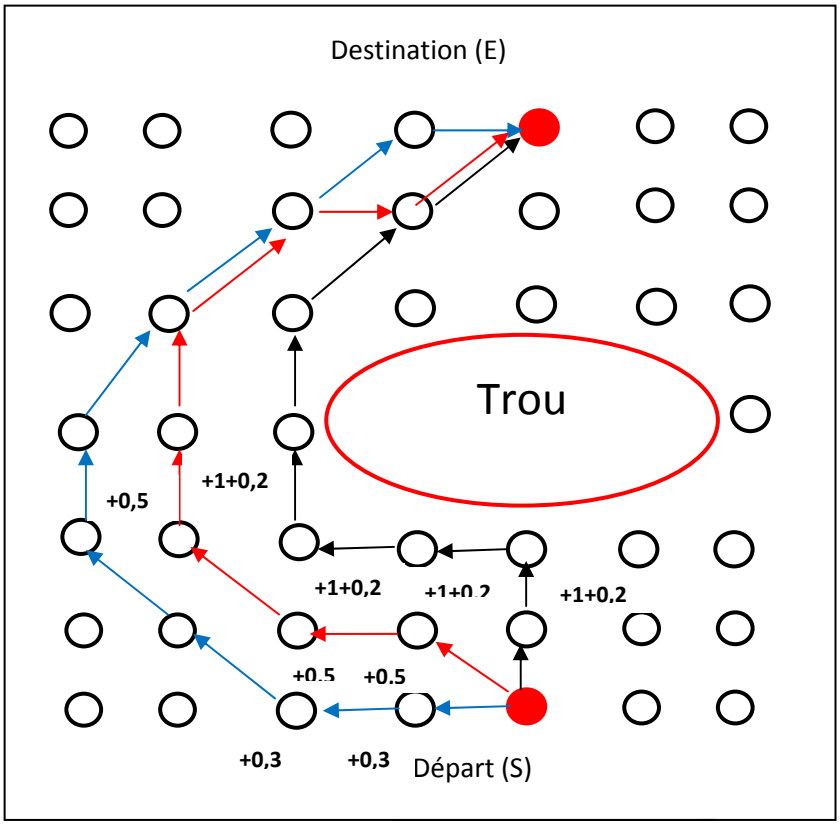


Figure 10: exemple de protocole WEAR

L'idée fondamentale de WEAR est illustrée par la figure 10, qui a un trou au centre de groupe de capteurs. Dans la figure, un message est envoyé de départ (S) à la destination (E), et les numéros près des nœuds représentent l'augmentation du poids, par exemple, +1 + 0,2 signifie qu'un trou près de capteur augmente le poids par 1, et la consommation d'énergie augmente le poids par 0,2. On peu voir que le protocole WEAR prend trois chemins différents pour le routage entre la même pair du départ et de destination.

Au début, Le WEAR prend le chemin dénoté par la ligne noire, comme un protocole purement géographique avide. Après que l'information de trou soit connue, les poids des capteurs qui sont proches du trou s'augmentent par 1, 0.5, ou 0.3, et la diminution d'énergie des capteurs sur le chemin noir, augmente la valeur du poids.

Après que les poids de la ligne noire s'augmentent, le WEAR suit le chemin représenté par la ligne rouge, et quand l'énergie des capteurs de cette ligne diminue, les poids de ces nœuds s'augmentent encore, et le WEAR emploiera automatiquement le chemin montré dans la ligne bleue. En conséquence, le WEAR essais d'éviter le routage vers le trou aussi bien que distribuer la charge aux chemins alternatifs.

Comparé à WEAR, GPSR prendra seulement le chemin représenté par la ligne noire parce qu'il choisit toujours le plus court chemin basé sur la distance à la destination et dévie le trou.

2.6.2.1 Définition formelle de poids dans WEAR :

Le poids est la clef pour prendre la décision pendant le routage et il contient quatre facteurs, la distance à la destination, le niveau d'énergie des capteurs voisins, l'information globale d'endroit, et l'information locale de trou.

Le poids est défini formellement comme suit :

$$Wj = \alpha\, G_j + \beta\, L_j + \gamma\, R\, E_j + \lambda\, D_{jd}$$

Wj est la valeur de poids de capteur ***j*** ; L_j est la valeur d'information locale de trou de capteur ***j***; G_j est la valeur l'information de l'endroit global de capteur ***j*** ; $R\,E_j$ est l'énergie restante de capteur ***j*** ; et D_{jd} est la distance entre capteur ***j*** et la destination, et les quatre paramètres ***α, β, γ, λ*** dénotent l'importance des quatre facteurs.

- L'information globale d'endroit G: Le nœud le plus proche de la station de base est le nœud le plus important. Ainsi G est calculé comme suit :

$$G = c \frac{d_{max} - d_{js}}{d_{max}}$$

Où ***c*** est une constante et d_{js} est la distance entre le capteur ***j*** et la station de base. d_{max} est la distance maximum de tout pair de nœuds.

- L'énergie restante *R* E_j : C'est l'énergie initiale sans l'énergie consommée pendant le message :

$$R\,E_j = c \frac{E_0 - C\,E_j}{E_0}$$

Où E_0 est l'énergie initiale des sondes. $C\,E_j$ est l'énergie consommée de capteur.

- L'information locale de trou L_j : l'influence du trou est liée à la distance de capteur au trou et l'espace occupé par le trou. Pour la distance, il y a deux manières de la mesurer, une, est basée sur les sauts entre ce capteur et la frontière de trou, et l'autre est basée sur la distance géométrique entre le capteur et le centre de trou. Elles sont calculées par les formules suivantes respectivement:

$$Lj = \sum_{h=0}^{n} A_h \left(1 - \frac{Hops_i}{Max_{hop}}\right)$$

Ou :

$$Lj = \sum_{h=0}^{n} A_h \left(1 - \frac{C}{d_{jh}}\right)$$

Où Lj est l'impact du trou au capteur ***j***; A_h est l'espace du trou avec l'identification h; $Hops_i$ signifie le nombre de sauts entre le capteur ***j*** et la frontière de trou; Max_{hop} est le nombre maximum des sauts; c est une constante ; n est le nombre des trous; et d jh est la distance entre le capteur ***j*** et le centre h du trou.

- La distance à la destination D_{jd} : Le facteur D jd est défini comme distance géométrique entre le capteur j et la destination.

Le poids contient quatre composants, leurs significations sont impliquées par leurs coefficients correspondants ***α, β, γ, λ***. De ce fait, les applications peuvent choisir la configuration appropriée pour répondre à leurs exigences spécifiques. Par exemple, si on prend ***α, β, γ*** égales à zéro, WEAR se confond à GPSR. Et si on prend ***α, γ*** égales à zéro, WEAR est semblable à GEAR.

2.6.2.2. Calcul de l'information du trou : Un des buts importants de WEAR est à dévier le trou et empêcher l'agrandissement de celui-ci. WEAR identifie le trou et propage l'information de trou aux capteurs qui sont près du trou, puis modifier le poids des capteurs.

2.6.2.2 .a. L'identification du trou : Le procédé d'identification de trou se compose de trois étapes, *localisation de trou*, *annoncement de l'information du trou* et *propagation*. L'idée fondamentale de l'identification de trou est illustrée sur la figure 11.

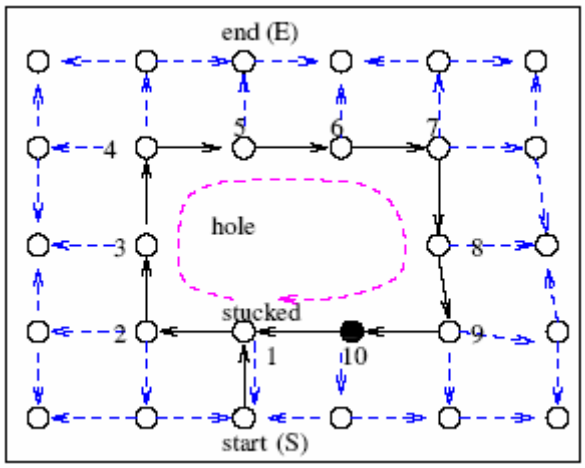


Figure 11 : Exemple d'identification et propagation de Trou

Étape 1 : ***localisation du Trou :*** la première étape d'identification de trou doit trouver le trou dans le réseau, qui est montré comme lignes noires pleines dans la figure. Les auteurs de WEAR utilisent l'algorithme de localisation de trou proposé dans [60]. D'abord quand un message atteint un nœud coincé avec l'identificateur ID 1, il produit un message localisant le trou, plaçant la valeur de maximum et de minimum de coordonnées x et y dans ses propres coordonnées, et se considère comme propriétaire.

Selon la règle right-hand décrite dans [60], le message est routé et modifié si nécessaire au long du bord de trou. Plusieurs sauts plus tard, et quand un capteur voit un message déjà vu, le trou avec l'ID 10 est localisé. Par supposition que les trous sont sous forme d'un rectangle, On peut facilement calculer la surface du trou et du centre du trou en basant sur l'information de trou collecté.

Étape 2 : ***annoncement du trou :*** *A*près avoir trouvé le trou, l'information de trou est distribuée aux capteurs sur la frontière de trou.

Étape 3 : ***propagation du trou*** cette étape commence immédiatement après l'*annoncement de trou*, comme dénoté par les lignes bleues sur le schéma. Après que les capteurs de bord de trou obtiennent le message d'annoncement, ils produisent le message propageant l'information de trou. Les capteurs reçoivent le message de propagation de trou et mettent à jour l'information de trou si le message contient une nouvelle information sur le trou.

2.6.2.2.b. Maintenance du trou : En raison de la dynamique du réseau de capteur, le trou peut agrandir ou se déformer pendant la vie du réseau. Par conséquent, maintient l'information fraîche de trou est également une grande issue. Habituellement, les trous dans un groupe de capteurs se changèrent dans deux modèles : *agrandissement de trou* et *la fusion de trou,* ce dernier se produit quand quelques capteurs qui sont entre deux trous tombent en panne de sorte que ces trous se fusionnent en un seul trou.

2.6.2.2.b.1. Protocole de maintenance du trou:

Deux techniques de maintenance sont utilisées à savoir :

a) la maintenance périodique : le capteur propriétaire mis à jour périodiquement l'information de trou, par exécution des trois étapes d'identification du trou sauf que l'ID de

trou est précédemment connue. Quand d'autres capteurs reçoivent la nouvelle information de trou, elles mettent à jour leur valeur de poids également.

En cas ou les capteurs lancent la procédure d'identification de trou dans deux intervalles de mise à jour, et quand le message d'identification de trou arrive à un capteur sur le bord d'un vieux trou dont la même direction. Le capteur ne prend pas le message en considération et envoie un message de réponse contenant l'information de vieux trou au créateur de message.

b) la maintenance réactive : La maintenance réactive se diffère de la maintenance périodique, quand un capteur sur le bord du trou reçoit un message d'identification pour un vieux trou, il complète le processus d'identification de trou immédiatement mais sans mise à jour périodique. L'avantage de cette approche est qu'elle garde toujours l'information de trou fraîche.

2.6.2.2.b.2. Agrandissement du trou : est causé par la défaillance de nouveaux capteurs sur la frontière du trou. Quand ces capteurs défaillent, une partie de leurs voisins deviennent la nouvelle frontière, ainsi le trou est agrandi et l'information de trou a besoin de mis à jour.

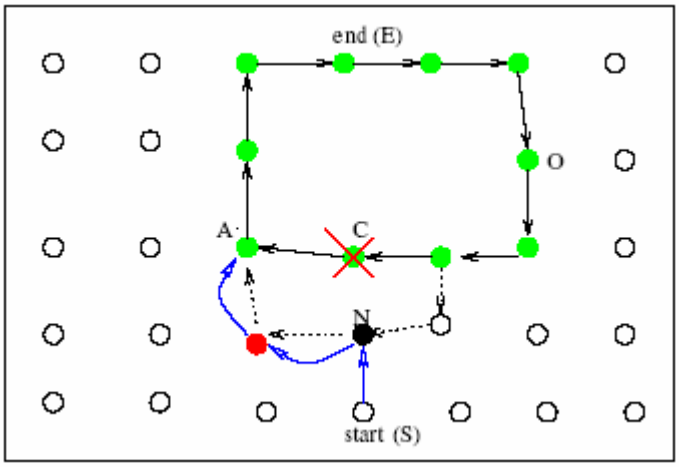


Figure 12: exemple de maintenance de trou

La figure 12 présente un exemple d'agrandissement de trou. Les nœuds verts reliés par les lignes noires désignent l'ancien trou. Quand le noeud C sur le bord du trou défaille, et si le nœud S envoie le message à la destination E, le noeud noir N est le nouveau nœud coincé sur le bord du nouveau trou. Ainsi il commence la procédure d'identification de trou.

Quand le message d'identification de trou, désigné par les flèches discrètes, atteint le nœud A, qui a l'ancienne information de trou. Il trouve que c'est un agrandissement du l'ancien trou, ainsi lui attachera l'ID du trou au message d'identification de trou et l'envoi. Puis le trou est réidentifié et la partie prolongée de trou est désignée par la ligne discrète dans la figure 12, alors la nouvelle information de trou sera annoncée et propagé aux capteurs voisins, qui mettent à jour la valeur de poids.

2.6.2.2.b.3. Fusion de Trou :

Si des capteurs placés sur le bord de deux trous se tombent en panne, ces trous peuvent fusionner à un seul trou. Quand le fusionnement de trou apparaît, le nouveau ID de trou deviendra la combinaison des IDs des trous fusionnés, et l'information de trou telle que la surface sera également mis à jour et propagé aux capteurs près du trou.

La figure 13 est un exemple de fusion des trous. Après que le capteur C défaille la fusion s'apparaît. Si le propriétaire du trou 1 commence une mise à jour de trou. Le message atteindra le capteur A, qui sait qu'il est sur le bord de trou 1, et puisque le capteur C a échoué, le message routera au capteur B, qui sait qu'il est sur le bord du trou 2 et qu'il est a deux sauts de trou 1.

Quand B reçoit le message de mise à jour du trou 1, il identifie une fusion de trou parce qu'il est sur le bord des trous 1 et 2 dans ce temps. Ainsi B combine l'ID de deux trous h1 et h2, et envoi le message. Le trou résultant est identifié comme montré par les flèches noires dans la figure 13.

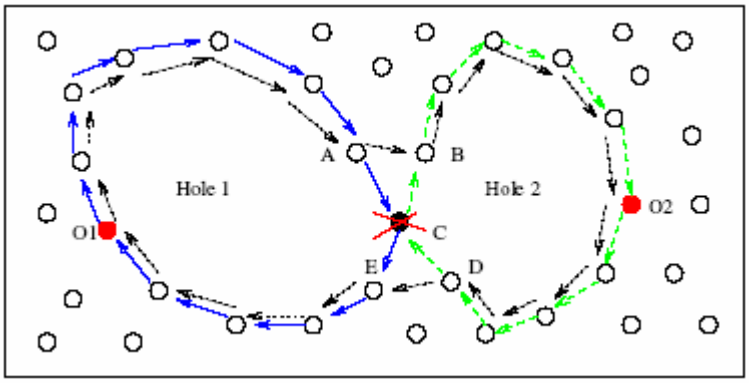


Figure 13: fusion de deux trous.

2.7 Tolérance aux fautes dans l'agrégation de données :

Un réseau de capteurs est considéré comme une plateforme à collecter les données corrélées des nœuds de capteurs et les transfèrent à une station de base a travers un réseau multi sauts. Un défi principal dans ces réseaux est l'efficacité énergétique. Les batteries des nœuds peuvent ne pas être rechargeables quand leur énergie est consommée, et la vie de réseau dépend de la conservation d'énergie. Ainsi le défi de conception est que doit rassembler des données tout en maintenant une longue vie. Récemment, la fusion de données ou l'agrégation a été étudiée comme une stratégie pour économiser de l'énergie en employant la corrélation des données de différents nœuds pour réduire le taux de transmission de données. Et comme les réseaux de capteurs sont caractérisés par la panne fréquente des nœuds, la tolérance de fautes devrait être considérée en concevant les protocoles de la collection et de l'agrégation de données.

2.7. 1 Les approches de corrélations :

L'agrégation de données peut être basée sur le codage commun des échantillons mesurés X, sans ou avec communication explicite. L'ancienne approche est l'emploi de codage de Slepian-Wolf [64]. Ceci, cependant, exige la connaissance centralisée de la structure de corrélation de X. En outre, un tel schéma ne sera pas tolérant aux fautes de lien ou de nœud, ce qui pourrait causer des erreurs dans le décodage de données par beaucoup de nœuds.

Dans la deuxième approche de communication explicite, les nœuds exploitent la corrélation de données seulement en recevant l'information explicite d'autres nœuds. Puisqu'un nœud emploie d'autres nœuds comme relais, les données de premier nœud sont disponibles aux nœuds de relais. Dans ce cas-ci, le codage de données est facile et se fonde sur des données localement disponibles. La nouveauté ici est que les données transmises par le nœud donné dépendent de flux entrant d'autres nœuds qui emploient ce nœud comme relais, et également sur la structure de transmission de ces nœuds. La communication explicite est plus tolérante aux fautes de lien ou de nœud que le codage de Slepian-Wolf.

L'agrégation de données basée sur le codage de Slepian-Wolf suppose la connaissance de la corrélation des données à tous les nœuds. Les encodeurs peuvent alors séparément coder des données à chaque nœud en tant qu'efficacement comme si chaque encodeur pourrait voir les valeurs de données de tous autres nœuds. Cependant, l'incapacité des schémas de tolérer

les défaillances élimine cet avantage, parce que si les bits codés d'un nœud sont perdus, le nœud collecteur peut ne pas pouvoir reconstruire plusieurs valeurs de capteurs.

L'arbre optimal de routage pour la collecte de données pour le codage de Slepian-Wolf est l'arbre de chemin le plus court (SPT) [65].

L'agrégation de données basée sur la communication explicite n'exige pas la connaissance de la structure de corrélation d'échantillons mesurés. Le codage à chaque nœud est simple mais trouver l'arbre optimal de collecte de données est difficile.

Soit le graphe G = (V, E) avec |V| = N+1, le nœud N + 1 est la station de base S. Chaque nœud i doit transmettre au taux R_i par le réseau à la station de base. Le problème est de trouver l'arbre ST (spanning tree) et les taux R_i qui réduisent au minimum $\sum_{I,v} R_i d_{ST}$ (i, S) , où d_{ST} (i, S) est le poids totale de chemin reliant le nœud i à S sur l'arbre ST. C'est difficile parce que le taux R_i dépend des nœuds qui emploient le nœud i comme relais. Le problème est NP –complet [65].

2.7.2 L'impact de perte de lien du réseau :

Soit p la probabilité que les bits codés, ou le paquet formé par ces bits est perdu. Le facteur *L* de perte d'un capteur est la division probable des valeurs de capteurs (paquets) qui ne peuvent pas être reconstruit au collecteur [66].

La structure optimale pour le codage de Slepian-Wolf est SPT. SPT contient le chemin avec le nombre minimum de sauts de chaque capteur à la station de base. Si le lien de X_1 à son parent échoue, aucuns paquets ne peuvent être décodés au niveau de station de base. Si ce lien n'échoue pas mais le lien de X_2 à son parent échoue, alors seulement un paquet peut être décodé. Si tous les deux liens n'échouent pas mais le lien de X_3 à son parent échoue, alors deux paquets sont réussis.

Par conséquent, basé sur la supposition que le coût d'un chemin est le nombre de liens sur ce chemin, le facteur de perte d'un réseau qui utilise le codage de Slepian-Wolf est donné par la formule suivante :

$$L_{SW}(p, N) = \frac{1}{N}\Sigma_{k=0}^{N-1}(N-k)(1-p)^k p$$

Le facteur de perte dans le cas de communication explicite, L_{EC}, dépend de la structure de réseau. Si un nœud ou un lien échoue sur l'arbre ST (spinning tree), tous les nœuds au-dessous de ce nœud ou liens échouent d'envoyer leurs données à S. D'autre part, quand un nœud ou un lien échoue dans Codage Slepian-Wolf, l'ensemble contenant les nœuds d'une distance plus élevée échouent d'envoyer leurs données. Cet ensemble contient donc tout le les nœuds descendants sur SPT. Le L_{EC} est haut lié par L_{SW}. Ils sont égaux si le réseau est linéaire.

S.Coleri & P. Variya comparent en [67] la performance des déférents schémas d'agrégation de données en termes de consommation d'énergie et la tolérance aux fautes, et ils montrent que quand l'agrégation basée sur le codage de Slepian-Wolf est efficace en énergie que la communication explicite, elle est moins tolérante aux défaillances de lien à cause que le décodage des données codées d'un nœud dépendent des données d'autres nœuds. Ils améliorent la tolérance de fautes de communication explicite en transmettant les paquets sur des chemins multiples de chaque source à la destination.

2.7.3 Tolerance aux fautes de schéma d'agrégation:

Une approche générale utilisée pour la collection et l'agrégation de données est de construire un arbre qui mène à la station de base et relie tous les nœuds dans le réseau. Cependant, une topologie d'arbre n'est pas robuste contre aucune défaillance de nœud ou de transmission, Puisque chacun de ces défaillances cause la perte d'un sous-arbre des valeurs captées.

Une approche d'agrégation, appelé *la diffusion de synopsis (synopsis diffusion)*, qui utilise le routage multi chemins efficace en énergie proposé dans [68], il y a trois fonctions de base :

Fonction de génération de *synopsis SG*(.) : prend la valeur captée et produit une combinaison *(synopsis)* représentant ces données.

Fonction de fusion de *synopsis SF*(.;.) : fusionne deux *synopsis* et produire un nouveau *synopsis* .

Fonction d'évaluation de *synopsis SE*(.) : traduit le *synopsis* en réponse finale.

L'algorithme de diffusion de *synopsis* se compose d'une phase de distribution et une autre d'agrégation. Dans la phase de distribution, la requête d'agrégation est diffusée par le réseau et une topologie d'agrégation est construite. Dans la phase d'agrégation, l'agrégation des valeurs captées est routé pas par pas vers le nœud qui envoi sa requête.

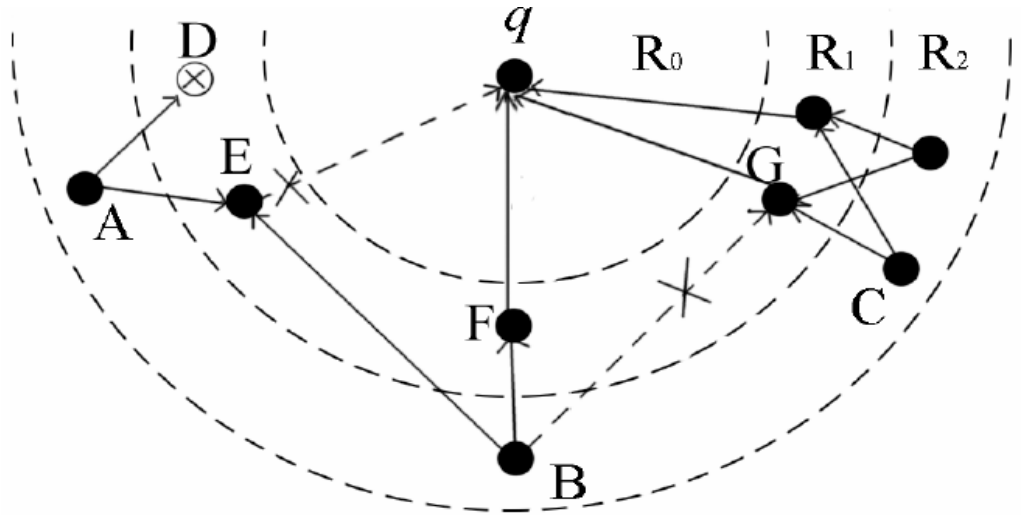


Figure 14 : exemple de diffusion de *synopsis*

Pendant la phase de distribution de la requête, les nœuds dans le réseau forment un ensemble d'anneau autour du nœud qui envoi la requête, dans cet exemple (figure 14) le nœud q dans l'anneau R0, selon leur distance *à q*, et un nœud est dans l'anneau *Ri* si il est de i saut loin *de q*. La durée de requête d'agrégation est divisée en périodes et le processus d'agrégation est exécuté une fois à chaque période. Il ya cinq nœuds dans l'anneau R1 où un de ces nœuds est défaillant, et quatre nœuds dans R2. Au début de chaque période, chaque nœud dans l'anneau extérieur (*R2* dans l'exemple) produit de son synopsis locale *s = SG* (r), où *r* est la valeur captée concernant la réponse à la requête, et diffuse le synopsis a tous ses voisins.

Lors la réception d'un synopsis s' par un nœud il mis à jour sa synopsis locale en tant que *s = SF* (*s ;s), et le diffuse* à la fin de la durée assignée au nœud. Ainsi, les synopses résultantes sont transmît couche par couche vers le nœud *q,* qui renvoie *SE* (s) comme réponse du requête à la fin de la période. Nous pouvons voir que la synopsis du nœud *B* peut atteindre le nœud q même les transmissions $E \rightarrow q$ et $B \rightarrow G$ échouent. Cependant, comme le nœud *D* et la transmission $E \rightarrow q$ sont défectueux, synopsis du nœud *A* ne peuvent pas être reçus par le nœud *q*.

En divisant un réseau en ensemble d'anneaux, l'agrégation de données peut être faite sur des topologies arbitraires. Puisque des chemins multiples sont utilisés pour conduire des données au nœud qui envoi la requête.

Un autre défi de l'algorithme d'agrégation est qu'il doit soutenir les agrégats des valeurs dupliqués, deux schémas tolérants aux fautes pour l'agrégation des valeurs dupliqués dans les réseaux de capteurs étaient présentés dans [69]. Les schémas emploient la redondance de chemin pour livrer un résultat d'agrégation correct à la station de base. L'idée fondamentale est comme suit, quand un paquet est perdu entre deux à cause de défaillance de lien, il est possible qu'un ou plusieurs autre capteurs possèdent le paquet. Si certains d'entre eux n'ont pas encore transmis leurs propres valeurs, ils corrigent l'erreur en agrégeant la valeur absente.

La topologie de réseau est formée dans les couches qui sont semblable à la figure 14, la seule différence est que quelques liens peuvent exister parmi des nœuds dans la même couche

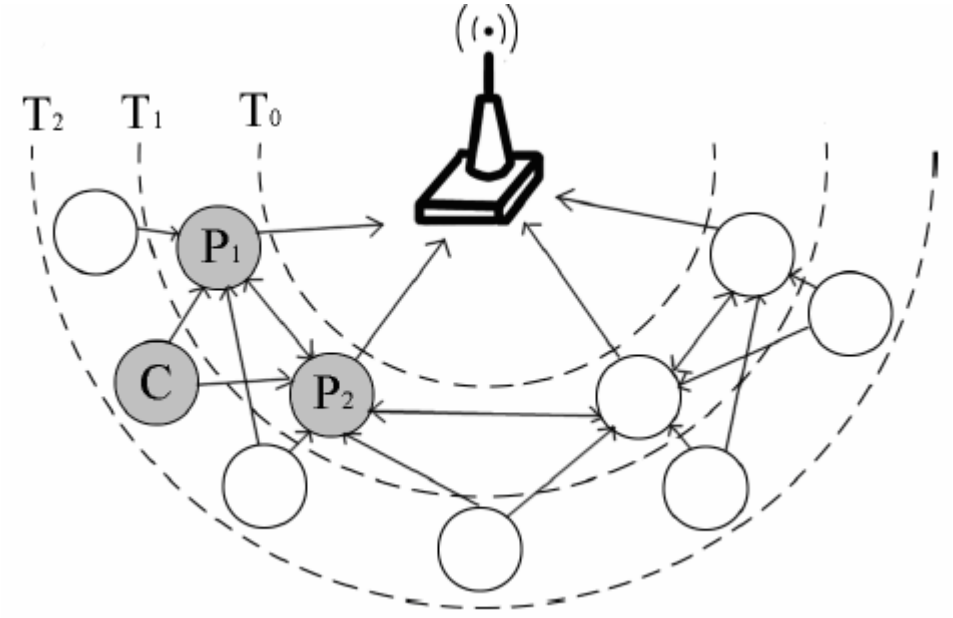


Figure 15 : schéma d'agrégation qui traite les valeurs dupliquées

Dans la figure 15 il y a trois couches, le capteur *C* en couche *T2* a deux parents : parent primaire *P1* et parent de secours *P2*.*C* transmet un message *à P1* et *P2* reçoit aussi le message. S'il n'y a aucune erreur, seulement *P1* qui fait l'agrégation de message. Supposant

qu'il y a une erreur sur le lien primaire *C* → *P1*, *P2* recevra le message *P1* sur le lien *P1* → *P2 et* constate que *C* est absent, et fait l'agrégation de la valeur *du C* pour corriger l'erreur.

2.8 Tolérance aux fautes dans le contrôle de topologie :

La propriété de la tolérance de fautes peut être invalide dû aux mouvements et à l'épuisement d'énergie des nœuds. Par conséquent, le contrôle de topologie est exigé pour construire et maintenir la propriété de la tolérance de fautes dans les réseaux de capteurs.

Un protocole de contrôle de topologie tolérant aux défaillances est proposé par Y. Chen, and S.H. Son [70], le protocole construit un ensemble dominant connecté (CDS: Connected Dominating Set), qui représente l'axe du réseau, et pour chaque nœud dans les CDS, il ajoute des nœuds nécessaires parmi les voisins de nœud au CDS, pour avoir un certain degré de connectivité. Le modèle de off/on est adopté pour réveiller les nœuds dans l'axe pour satisfaire la condition de connectivité et les autres nœuds inutiles sont éteints.

Il y a plusieurs métriques de choix, une est basée sur l'énergie: le nœud dans CDS choisit des nœuds avec plus d'énergie un par un jusqu'à que le graphe résultant soit le *k*-vertex local connecté. Un autre métrique est le degré de connexion, les nœuds avec un degré plus élevé de connexion sont d'abord choisis. Quelques métriques hybrides sont également discutées dans [70].

La majorité des travaux de control de topologie tolérante aux défaillances existants étudient la connectivité *k-vertex,* qui exige l'existence de *k-vertex* chemins distincts entre chaque deux nœud capteurs. La condition est plus appropriée pour les réseaux ad-hoc, où chaque deux nœud peuvent être source et destination. Cependant, la transmission de données dans les réseaux de capteurs est habituellement de la façon de collecte et d'agrégation, il est important pour avoir la tolérance aux fautes dans les chemins des nœuds capteurs à la station de base et aux nœuds de passage, dans le ces des réseaux de capteurs hétérogènes.

Cardei et al étudient [71] le contrôle de topologie tolérant aux défaillances dans les réseaux de capteurs hétérogènes sans fil. Le réseau se compose de deux types de dispositifs sans fil : un grand nombre des nœuds de capteurs déployés aléatoirement et de plusieurs supers nœuds riches en ressources et placés aux endroits connus. Ils modèlent le contrôle de topologie comme un problème de gamme de transmission pour lequel la gamme de

communication de chaque nœud capteur doit être calculée. Le problème est d'ajuster la gamme de transmission de chaque nœud, telle qu'il existe *k*-vertex chemins disjoints de chaque nœud à l'ensemble de supers nœuds, et l'objectif est réduire la puissance de transmission de capteur et minimiser la consommation totale d'énergie par les nœuds capteurs. De cette façon le réseau peut tolérer l'échec de jusqu'à K-1 nœuds capteurs.

Les auteurs de [71] formulent le problème de k-degree Anycast Topology Control (k-ATC) qui consiste à déterminer la gamme de transmission r i de chaque capteur n i tels que :

1. existe k -vertex disjoints chemins de communication de chaque nœud capteur à l'ensemble de super nœuds, c.-à-d, k –vertex super nœud connectivité, et

2. l'énergie totale consommée par tous les capteurs est réduit, c.-à-d, $\sum_{i=1,N} p_i$ = minimum .

Et ils proposent par la suite trois solutions pour résoudre le problème k-ATC :

a) algorithme k –approximation, $MWATC_k$ (Minimum Weight-Based Anycast Topology Control) qui consiste en deux étapes, dans la première, le graphe est réduit à un graphe dirigé où les super nœuds sont fusionnées comme racine. Dans la deuxième étape, une optimale solution pour le problème de Min-Weight *k*-Out Connectivity est adoptée pour calculer la minimum gamme de transmission de chaque nœud [72], ce problème consiste à trouver un sous graphe dirigé couvrent de graphe G tel que :

1. la somme des poids des lignes choisis est réduit au minimum, et

2. il y a k -vertex disjoints chemins entre r et n'importe quel sommet dans le graphe.

b) un algorithme avide centralisé (centralized greedy algorithm) GATCk qui minimise la gamme de transmission de capteur.

c) un algorithme distribué de localisation DATCk (Distributed Anycast Topology Control) qui adapte le niveau d'énergie de chaque nœud tel que la connectivité k-vertex de super nœuds est assurée.

La tolérance aux fautes des réseaux sans fil peut être réalisée par le contrôle de mouvement des nœuds. Bien que le travail dans [73] soit pour les réseaux mobiles de robots,

l'algorithme proposé peut être facilement appliqué aux réseaux de capteurs sans beaucoup de modification. Un algorithme localisé de contrôle de mouvement a été proposé pour former une topologie de réseau tolérante aux fautes bi-connecté a partir d'un réseau connecté. L'objectif est de réduire au minimum la distance totale du mouvement des nœuds.

2.9 Tolérance aux fautes dans la détection de cible et d'événement :

Une des applications importantes de réseau de capteurs est la détection, la classification et la localisation des événements spécifiques, et la détermination de cible dans une région spécifique. Un exemple d'application le déploiement d'un un réseau de capteurs dans un champ de bataille pour détecter les véhicules des ennemis. Une fois qu'un véhicule entre dans le champ, l'information de celui-ci, tel que l'endroit et la vitesse, sera recueilli et rapporté par les capteurs qui peuvent la détecter. Un autre exemple est l'utilisation de réseau de capteurs comme observateurs pour détecter le feu dans une forêt. Plusieurs travaux en tolérance aux fautes dans la détection de trajet et évènements ont été effectués, et dans cette partie nous présentons quelques travaux représentants en détection de cible et la détection d'évènement respectivement.

2.9.a) Détection de cible:

Clouqueur et al proposent en [74] deux algorithmes tolérants aux fautes pour la détection collaborative de cible dans les réseaux de capteurs, dans lequel les nœuds peuvent tombent en panne dûe aux conditions environnementales dures ou avec malveillance. Les deux algorithmes sont basés sur le partage d'information entre les nœuds capteurs pour atteindre le consensus.

Le premier algorithme, appelé *la fusion de valeur*, fonctionne comme suit. Chaque nœud obtient les mesures d'énergie de chaque nœud, calcule la moyenne utilisant la plus grand et la plus petite valeur *de n*, et compare cette moyenne à un seuil pour la décision finale pour *un n* donné. Le deuxième algorithme, appelé *la fusion de décision*, travaille sur la décision locale de chaque nœud capteur.

L'accord exact garantit l'uniformité finale parmi des nœuds, et donc les nœuds défectueux peuvent seulement dégrader l'exactitude du système, et l'échec de système se produit quand la limite sur le nombre de nœuds défectueux acceptables est violée. Les auteurs

ont mesuré la probabilité de détection des deux algorithmes pour différentes probabilités de fausse alarme, nombre de nœuds, puissance maximum et affaiblissement de facteur et ont conclu que la fusion de décision est supérieure à la fusion de valeur dans le taux d'augmentation de nœuds défectueuses. En plus de coût bas de communication, précision élevée et l'exactitude élevée, les deux algorithmes sont efficaces en termes de nombre de capteurs défectueux tolérables dans le réseau.

Ding et al [75] ont récemment proposé des algorithmes tolérants aux défaillances pour détecter la région qui contient des cibles et pour les identifier, dans la région de cible, tout en approvisionnant une faible énergie et à la présence des capteurs défectueuses.

L'idée fondamentale derrière leur approche pour la détection de cible est que chaque capteur calcule la médiane des signaux mesurés, tel que les perturbations des mesures provoquées par les capteurs défectueuses sont filtrées, et si une médiane dépasse certain seuil alors elle implique qu'une cible possible est présente. Mais sans déterminer le nombre de cibles ni où elles sont.

Un algorithme de localisation de cible est employé pour calculer la position de chaque cible. La tâche de communiquer avec la station de base et le calcul de position des cibles est destinée à un capteur particulier, appelée le capteur racine (root). Le capteur racine calcule le centre géométrique des capteurs voisins avec les observations semblables.

L'algorithme de Ding et al [75] travail bien dans les réseaux de capteurs denses car la médiane n'est pas robuste dans la faible densité, et les cibles doivent être distantes pour les identifier en tant que des cibles indépendantes. Les auteurs supposent que chaque capteur calcule son endroit physique en utilisant les techniques GPS, et il n'y a aucune faute dans le traitement, la transmission et la réception des informations captées par les voisines.

2.9.b) Détection d'événement :

Krishnamachari and Iyengar [76] ont proposé une méthode tolérante aux fautes de détection d'événement localisé et distribué pour les réseaux de capteurs sans fil. Basé sur l'observation que les fautes de capteur sont susceptibles d'être stochastiquement non-corrélatifs, tandis que des mesures d'événement sont susceptibles d'être dans l'espace corrélées, ils proposent un algorithme tel que chaque capteur communique avec ses voisins pour rassembler le leur décisions pour corriger sa propre décision. Dans [77], Luo et al ont

proposé une approche distribuée qui traite l'erreur de détection et la faute de capteur simultanément, où ils choisissent la taille appropriée de voisinage afin d'être efficace en énergie et fournir la des erreurs de détection proportionnée. Leur approche est une amélioration du travail de Krishnamachari et Iyengar [76], qui fournit un schéma de vote de majorité pour permettre aux nœuds de corriger leurs décisions en communiquant avec leurs voisins. L'amélioration est l'addition de l'erreur de détection en plus de faute de capteur, ainsi que la détermination d'une taille appropriée de voisinage pour améliorer l'efficacité énergétique.

2.10 Tolérance aux fautes dans les applications de surveillance :

La surveillance se diffère à la détection de cible/événement discutée dans la section précédente. La détection de cible/événement doit détecter la présence et le changement de l'état de cibles et des événements, alors que la surveillance est de surveiller les cibles statiques et les zones intéressantes. Dans l'application de surveillance, la stratégie pour augmenter la fiabilité est le déploiement d'un grand nombre de nœuds. Puisqu'il y a une contrainte d'énergie, généralement seulement une partie des nœuds de capteurs est en activité tandis que d'autres vont au sommeil. Par conséquent, il exige une synchronisation qui détermine qui reste actif et qui peut aller au sommeil. Pour assurer l'opération appropriée du réseau, les nœuds de sommeil devraient fréquemment surveiller les nœuds actifs et les nœuds défaillantes seront remplacés une fois qu'elles sont détectées. D'autre part, les nœuds devraient demeurer dormants autant que possibles pour économiser de l'énergie.

La période de surveillance optimale dans les réseaux de capteurs tolérant aux fautes a été étudiée dans [78]. Un algorithme de synchronisation appelé Sleep-Query-Active (SQA) était proposé, son but est d'assurer la connexion de réseau et maximiser la durée de vie du réseau. Elle suppose que les nœuds se rendent compte de leurs endroits et emploie l'information pour diviser un espace bidimensionnel en grilles. La distance de deux points les plus lointains dans deux grilles adjacentes quelconques doivent être plus petits que la gamme de communication des nœuds capteurs

Le problème d'assurer la couverture de zone dans des réseaux de capteurs a été étudié dans [79]. Le problème est la synchronisation des nœuds capteurs pour qu'ils soient en activité ou en sommeil, tel que la connectivité et la couverture de la zone sont assurées. Les auteurs

ont proposé quatre variantes des protocoles qui transmettent avec cout bas de communication afin d'être appliqué pour les réseaux fortement denses.

La nature de surveillance des réseaux de capteurs exige une longue durée de vie. Le problème de durée de vie maximum dans des systèmes de surveillance a été étudié dans [80].

Puisque les capteurs sont habituellement déployés d'une manière redondante, le problème est de synchroniser un sous-ensemble de capteurs pour être en activité en temps qu'ils observent les cibles et trouver les routes ou les chemins pour que les capteurs actives envoient des données au station de base, tel que chaque cible devrait être observée par k capteurs à tout moment et la durée de vie du réseau est maximisée. La durée de *vie* est le temps jusqu'à où il existe une cible qui ne peut pas être observée par des k capteurs ou les données ne peuvent pas être expédiées à la station de base due à l'épuisement de l'énergie des nœuds.

La solution optimale proposée se compose de trois étapes. Dans la première, elle a formulé le problème avec la technique de programmation linéaire (LP) pour calculer le haut bondir sur la vie maximale et une matrice de charge de travail (workload matrix). L'objectif du la deuxième étape est de trouver la synchronisation détaillé pour que les capteurs observent des cibles basant sur la matrice de charge de travail. L'idée fondamentale est de représenter la matrice de charge de travail en tant que graphe bipartite et applique la technique pour décomposer la matrice de charge de travail en séquence de matrices de synchronisation. La dernière étape à construire les arbres des capteurs de surveillance pour chaque session basée sur les matrices de synchronisation et les données calculées dans la formulation de LP.

2.11 Conclusion :

Nous avons présenté dans ce chapitre les fautes qui peuvent survenir dans un réseau de capteurs ainsi que ses origines, et quelques techniques de tolérance aux fautes étudiées et appliqués au niveau de capteur et celle de réseau, dans les services de routage, l'agrégation de données et le contrôle de topologie, et dans des applications comme la détection de cible et d'évènement et les application de surveillance.

Nous allons étudier dans le prochain chapitre, la tolérance aux fautes dans l'agrégation de données d'un point de vue de système distribué par l'utilisation de service d'élection dans la construction de schéma d'agrégation.

Chapitre 3 :

Le service d'élection du leader pour une agrégation tolérante aux fautes dans les réseaux de capteurs sans fil

3.1 Introduction :

Un système tolérant aux fautes satisfait deux propriétés : la *sûreté* et la *vivacité*. Un algorithme réparti est correct s'il satisfait les propriétés de sûreté et de vivacité. La sûreté est une propriété continue, elle est garantie grâce à la détection des fautes. La vivacité est une propriété éventuelle, elle est assurée grâce à la correction [81].

Durant ces dernières années, plusieurs paradigmes ont été conçus et mis en œuvre pour simplifier ces tâches. Le plus répandu, le *consensus*, qui est une forme générale d'accord. Il permet aux processus de prendre une décision commune, dépendant de leurs valeurs initiales, malgré la présence de défaillances. L'algorithme du consensus peut être utilisé pour résoudre plusieurs problèmes pratiques tels que l'élection d'un leader de groupe, et l'accord sur une variable répartie.

3.2 Le consensus :

Quelque soit le type du système réparti, l'élaboration d'applications fiables passe par la résolution de problème d'accord dans ce système, et les problèmes d'accord sont au cœur de la gestion des systèmes répartis. Nous intéressons aux systèmes dits asynchrones, le modèle temporel asynchrone est le modèle le plus général, autrement dit, il possède les hypothèses temporelles les moins fortes, les moins restrictives. Il convient à la modélisation de réseau de capteurs.

Le consensus est un problème important dans les systèmes répartis, tels que les réseaux de capteurs sans fil (WSNs). Il se produit toutes les fois que les capteurs doivent convenir sur une valeur ou des déclencheurs convient sur une action. Si on prend le cas d'un réseau de capteurs dans un bâtiment pour contrôler la température et conserver l'énergie, en utilisant le capteur et les déclencheurs pour ouvrir ou fermer les sorties d'une salle. Dans un tel système, beaucoup de décisions dépendent par exemple de la température, éclat du soleil et le vent, sont possibles. Les déclencheurs peuvent par exemple accepter de fermer tous les sorties. Ou ils pourraient convenir de fermer en moitié toutes les sorties, ou pour fermer la sortie A, tout en ouvrant B et C. et dans ce cas il doit y a un consensus parmi les déclencheurs.

Cependant, pour les systèmes répartis complètement asynchrones, Fischer et al [82] ont montré qu'aucun accord entre des processus ne peut être résolu d'une manière déterministe et tolérant aux fautes, ils prouvent qu'il n'existe pas de solution déterministe au problème de consensus dans un environnement asynchrone si les processus peuvent tombent en panne et même en présence d'une seule faute de processus, intuitivement, Cette impossibilité est due à l'absence des suppositions sur les délais de transfert et la vitesse d'exécution, qui rend difficile à distinguer un processus très lent d'un qui est réellement défaillant.

Pour palier à ce résultat d'impossibilité, plusieurs approches ont vu le jour : l'utilisation de techniques aléatoires, l'étude de différents modèles de systèmes répartis partiellement synchrones, et l'utilisation de détecteurs de défaillances non fiables.

3.2.1 Paradigme du consensus :

Dans le paradigme du consensus, tous les processus corrects proposent une valeur et doivent prendre une décision irrévocable sur une valeur appartenant à l'ensemble des valeurs proposées [82]. Le consensus utilise les deux primitives proposer (v) et décider (v), où v est une valeur extraite de l'ensemble des valeurs proposables par les processus. Quand un processus exécute la primitive proposer (v), on dit que le processus propose la valeur v. De même, quand il exécute décider (v), on dit qu'il décide la valeur v. Le paradigme du consensus est caractérisé par les quatre propriétés suivantes :

Terminaison : chaque processus correct décide éventuellement une valeur.

Validité uniforme : si un processus décide une valeur v alors v a été proposée par un processus.

Intégrité uniforme : chaque processus décide au plus une fois.

Accord : deux processus corrects ne décident pas deux valeurs différentes.

3.2.2 Les Détecteurs de défaillances

Les détecteurs de défaillances de Chandra et Toueg [83] peuvent être considérés comme des briques de base des systèmes distribués tolérant les fautes. Un détecteur de défaillances ou FD - Oracle (Failler Detector) peut être vu comme un ensemble de modules distribués. À chaque processus est associe un module. Ces modules fournissent aux processus du système une liste de processus suspectés d'être défaillants. Cette liste peut être différente d'un processus à l'autre.

Les détecteurs de défaillances sont habituellement implémentés en utilisant le délai de garde (time out): un processus est suspecté s'il ne s'est pas manifeste avant 1'expiration du délai de garde. Cela signifie qu'un détecteur de défaillances peut commettre des erreurs :

1. Ne pas suspecter un processus alors qu'il est défaillant.
2. Suspecter un processus alors qu'il est correct.

3.2.2.1) Propriétés de détecteurs de défaillances :

Formellement, un détecteur de défaillances est complètement défini par deux propriétés : la *complétude* et la *précision.*

• **La complétude** : définit des contraintes concernant la détection des processus réellement arrêtés. Elle peut être faible ou forte.

- *La complétude forte :* les processus fautifs sont finalement et définitivement suspectés par tous les processus corrects.

- *La complétude faible :* pour tout processus fautif, il existe au moins un processus correct qui le suspecte.

• **La précision** : exprime le degré de fiabilité dans le non suspicion des processus corrects. Idéalement, seuls les processus défaillants doivent être détectés comme tels. Là encore, la précision peut être forte ou faible. Mais aussi ultimement forte ou ultimement faible.

- *La précision forte :* les processus corrects ne sont suspectés à aucun moment.

- *La précision faible :* il existe au moins un processus correct qui n'est jamais suspecté.

- *La précision ultime forte* : il existe un moment à partir duquel aucun processus correct n'est plus jamais suspecté.

- *La précision ultime faible :* il existe un moment à partir duquel il existe un processus correct qui n'est plus jamais suspecté.

3.2.2.2) Les classes de détecteurs de défaillances :

Chandra et Toueg définissent huit classes de détecteurs de défaillances en caractérisant chacune d'entre elles par une propriété de complétude et une propriété d'exactitude [83]. Le tableau de la figure 16 montre ces huit classes.

La classe des détecteurs de défaillances fortes (dénotée S) regroupe tous les détecteurs de défaillances qui ont pour propriétés de suspecter les processus défaillants: (propriété de complétude) mais de ne pas suspecter au moins un des processus non défaillants (propriété d'exactitude) [83].

Complétude	**Precision**			
	Forte	Faible	Ultimement forte	Ultimement faible
Forte	**P**	**S**	◊**P**	◊***S***
Faible	**Q**	**W**	◊**Q**	◊**W**

Figure 16 : Classification des détecteurs de défaillances.

La classe des détecteurs de défaillances faibles (dénotée ◊S) regroupe quant a elle tous les détecteurs de défaillances qui vérifient la même propriété de complétude mais ne satisfont la propriété d'exactitude qu'après un laps de temps indéterminé. De ce fait, les détecteurs de défaillances appartenant à ces deux classes sont intrinsèquement non fiables puisqu'ils peuvent suspecter arbitrairement des processus corrects. Chandra et Toueg [83] ont montré que les classes P et S peuvent être employées pour résoudre le consensus dans les environnements avec le nombre arbitraire des échecs, alors que ◊P et ◊*S* peuvent résoudre le consensus dans les environnements avec une majorité de processus corrects.

3.3 Élection de leader :

Le problème d'élection constitue une brique de base utile dans les systèmes distribués, particulièrement ceux sujets à des défaillances. À titre d'exemple : si la défaillance d'un noeud provoque la perte d'un jeton dans un algorithme d'exclusion mutuelle, les autres noeuds doivent élire un nouveau 'leader' qui sera chargé de régénérer le jeton. L'élection d'un leader est aussi utile: dans les protocoles de communication de groupes afin de choisir un nouveau coordinateur lorsque la composition du groupe change.

Une solution au problème d'élection d'un leader pour un système reparti conventionnel est définie par les deux propriétés suivantes :

- *Accord :* il n'y aura jamais plus d'un leader.

- *Terminaison :* il existe éventuellement un leader.

Le problème d'élection a été intensivement étudié sur une grande variété de modèles et de topologies. En fait, les solutions proposées diffèrent par le mécanisme de communication utilisé (asynchrone ou synchrone). 1'identité des processus (identités uniques ou anonyme), et la topologie du réseau (anneau, graphe complet. topologie arbitraire. arbre. etc.). L'accent a été mis sur 1'étude de la complexité en messages.

3.3.1 Protocoles d'élection basés sur les détecteurs de défaillances :

Sabel et Marzullo [84] ont montré que 1'élection dans un réseau asynchrone sujet à des défaillances est impossible. Ceci et une variété des autres résultats d'impossibilité provenant du résultat de Fischer, Lynch et Paterson (FLP) [82], qui montre *qu'il n'existe pas de solutions déterministe au problème du consensus dans un environnement asynchrone même en présence d'une seul défaillance de processus.* Ceci motive la conception d'une variété de protocoles d'élection basée sur les détecteurs de défaillances.

3.3.1.1 Le détecteur de défaillance Ω :

En [85], Chandra, Hadzilacos et Toueg ont proposé un nouveau type de détecteur de défaillance, dénoté Ω,et ont prouvé que c'est le détecteur de défaillance le plus faible pour résoudre le consensus où une majorité des processus sont corrects. À chaque processus *p,* et à chaque instant *t,* la sortie de Ω est un seul processus, appelé q, et on dit que *p fait confiance* a *q* à l'instant *t.* Ω assure qu'éventuellement tous les processus corrects *font confiance* (trust) au même processus et que ce processus est correct. Ainsi, un détecteur de défaillances Ω peut être vu comme un algorithme d'élection d'un leader : le processus auquel font confiance tous les processus corrects est le processus élu. Notons que Ω est le détecteur de défaillances le plus faible pour résoudre le consensus [83].

Ω est un détecteur de défaillances qui réalise une élection de leader. On dira qu'une implémentation de Ω est stable, si elle assure qu'une fois un leader est élu, il reste le leader, sauf, si ce dernier tombent en panne ou si les canaux de communications avec ce leader n'ont plus un bon comportement. Enfin, une implémentation est efficace en communication, si ultimement, uniquement n canaux de communications sont utilisés.

Aguilera et al [86] ont considéré une forme faible d'élection, dénotée Ω, ou chaque processus *p* a une variable $leader_p$ qui maintient 1'identite d'un processus ou ⊥. Intuitivement, par la suite tous les processus corrects devraient maintenir 1'identité du même processus, et que ce processus devrait être correct. Plus précisément, on a besoin de la propriété suivante :

- II existe un processus correct *l* et un instant après lequel, pour chaque processus correct *p,* $leader_p = l$.

Si a 1'instant *t,* $leader_p$ contient le même processus *l* pour tout processus correct *p,* alors on dit que *l* est le *leader* a 1'instant *t.* Notez qu'un processus *p* ne sait jamais si $leader_p$ est vraiment le *leader* a l'instant *t,* ou pas.

En plus d'être stables, les implémentations proposées dans [86] ont plusieurs propriétés souhaitables. En particulier, ils sont tous robustes et efficace en communication (ils fonctionnent dans des systèmes ou seulement les liens avec quelque processus corrects doivent être ponctuels). De plus, la meilleure implémentation proposée dans [86] tolère les pertes de messages et assure 1'élection en temps constant à partir de 1'instant ou le système est stable.

3.4 Protocole d'élection pour le réseau de capteurs:

L'apparition des réseaux des capteurs sans fil est due à la demande de collecte de données fiable par des applications. Une des issues de réseaux de capteurs est la nécessité de réduire au maximum l'intervention humaine. Les applications peuvent agir de façon autonome au-dessus de réseau de capteurs, récupérant l'information rassemblée par les capteurs périodiquement ou sur demande.

Le service d'élection de leader peut aider les nœuds à choisir les agrégateurs pour la construction d'un schéma d'agrégation tolérant aux fautes, et leur permettre de détecter la panne de celles-ci, et par similarité, les nœuds de réseaux de capteurs sont équivalents aux processus dans un système distribués, et de ce fait, nous essayons, dans ce qui suivre, de présenter quelques algorithmes d'élection de leader dans les réseaux de capteurs sans fil.

3.4.1 Protocole WWLE (Wirless Wave Leader Election):

Dulman et al [87] ont présenté et simulé un protocole efficace d'élection pour les réseaux mobiles de capteurs. Le travail a été effectué au sein du projet EYES, un projet de recherche européen (Ist-2001-34734) sur les réseaux de capteurs auto organisés, collaboratifs et efficaces en terme d'énergie. L'algorithme WWLE proposé est efficace en nombre des " rounds " et des messages échangés. Il tient compte de la mobilité et de la topologie irrégulière du réseau.

Le point de départ de 1'algorithme WWLE était un algorithme décrit dans [88]. L'idée principal est d'attribué des identificateurs uniques, à partir d'un ensemble ordonné, à chaque noeud dans le réseau, et puis élire comme leader le noeud avec 1'identificateur minimal. Chaque nœud maintient 1'identificateur minimal qu'il a vu jusqu'a présent (initialement son identificateur). À chaque "round", chaque noeud propage ce minimum sur tous ses liens sortants. Après un certain nombre de "rounds" égal au diamètre du réseau, un noeud s'élit comme leader si la valeur minimale vue dans ce noeud est le propre identificateur du noeud; sinon c'est un non leader.

Quelques nœuds pourraient ne pas avoir assez de ressources pour devenir des leaders. Un coefficient peut être calculé localement par chaque noeud en fonction de la quantité de ressources disponibles. Une valeur basse signifie un désir (willingness) élevé d'être le leader. D'autres valeurs placent le noeud à des niveaux plus bas du désir d'être leader. La notion d'identificateur sera alors étendue à un identificateur obtenu par la concaténation du désir et de 1'identificateur original. Les caractéristiques particulières du protocole WWLE sont :

- un noeud transmettra 1'identificateur minimal seulement la première fois quand il 1'apprend (le premier "round" et puis seulement s'il le reçoit d'un voisin);

- un noeud décidera de transmettre la plus petite valeur immédiatement dès qu'il 1'a reçue et pas à la fin d'un "round".

- 1'identificateur d'un noeud contient également le coefficient du désir du noeud (la valeur est placée devant 1'identificateur original). Le noeud avec le plus petit identificateur transmettra en premier sur le canal. De cette façon, une chance sera donnée aux plus petites valeurs d'être propagées des le début.

3.4.2 Election d'agrégateur en RCSF par implémentation de détecteur de défaillance oméga :

Larrea & al [89] utilisent le détecteur de défaillance oméga pour l'élection de leader (agrégateur de données) en réseau de capteurs, ils présentent un algorithme (figure 17) qui implémente le détecteur Oméga dans le modèle de panne / reprise pour coordonner l'agrégation de données dans les réseaux de capteurs sans fil. L'algorithme assure l'accord sur un agrégateur commun pour tous les nœuds capteurs d'une région, aussi bien que super-agrégateur parmi l'ensemble d'agrégateurs du réseau, par conséquent fournissant un mécanisme d'agrégation de données hiérarchique efficace en terme d'énergie. Ils introduit également un seuil d'épuisement de batterie pour augmenter la qualité du service du réseau de capteurs sans fil.

Larrea & al considèrent l'épuisement de la batterie de nœud capteur comme un type particulier de panne, tel qu'il est possible de reprendre après remplacement de batterie.

Chaque processus p exécute ce code
procedure *GoToHibernation()*
(1) write (*incarnation* $_p$, *leader* $_p$, *Incarnation* $_{leader}$, *Timeout* $_p$) in stable storage
(2) *schedule_ wakeup* $_p \leftarrow$ True
(3) *hibernate()*
end procedure

Initialization:
(4) read (*incarnation* $_p$) from stable storage
(5) **If** [*scheduled_wakeup* $_p$ = *false*] then
(6) Incarnation $_p \leftarrow$ Incarnation $_p + 1$
(7) write incarnation $_p$ in stable storage
(8) **end if**
(9) scheduled_wakeup p $\leftarrow$ false
(10) read (leader $_p$, Incarnation $_{leader}$, Timeout $_p$) from stable storage
(11) if [leader $_p$ = p] then
(12) start tasks 1 and 2
(13) else
(14) reset timer p to Timeout p + 2 ε
(15) start tasks 2 and 3
(16) end if

Task 1:
(17) wait ε time units
(18) broadcast (I-am-the-leader, p, incarnation p)
(19) receive data from sensors during $\Delta_{\text{DATA ACQUISITION}}$ time
(20) GoToHibernation ()
Task 2:
(21) upon reception of message (i-am-the-leader, q, incarnation q) such that
[Incarnation $_q$ < incarnation $_{leader}$] or [(incarnation $_q$ = incarnation $_{leader}$) and
(q ≤ leader $_p$)] do
(22) leader $_p \leftarrow$ q
(23) incarnation $_{leader} \leftarrow$ incarnation $_q$
(24) send data to leader $_p$
(25) GoToHibernation()
Task 3:
(26) upon expiration of timer $_p$ do
(27) leader $_p \leftarrow$ p
(28) incarnation $_{leader} \leftarrow$ incarnation $_q$
(29) Timeout $_p \leftarrow$ Timeout $_p + \Delta_{\text{TIMEOUT}}$
(30) GoToHibernation()

Figure 17 : Implémentation de Ω dans un réseau de capteurs sans fil

L'algorithme commence par l'initialisation (figure 17), après si le capteur se considère comme agrégateur courant, l'algorithme lance les tâches 1 et 2. Sinon il commence les tâches 2 et 3. La tâche 1 est consacrée pour annoncer l'agrégateur au reste de capteurs et pour rassembler toutes les données de capteur. Tâche 2 est consacrée à l'envoi des données au agrégateur, et pour mettre à jour l'agrégateur s'il y a lieu. Et à la fin, La tâche 3 est consacrée pour proposer le capteur comme agrégateur quand l'annonce courante d'agrégateur n'est pas reçue avant l'expiration du temporisateur, et également faire des incréments de timeout. Toutes les tâches appellent la procédure GoToHibernation() à la fin, qui commence la période de sommeil. Chaque capteur restera dans cet état jusqu'au prochain réveil.

Les variables incluent l'incarnation initialisée à 0, qui est incrémenté pendant l'initialisation et chaque fois qu'il revient après une panne. Chaque capteur a un temporisateur utilisé pour détecter la panne potentielle de nœud agrégateur. Le constant $\Delta_{\text{DATA ACQUISITION}}$ représente le temps maximum passé par l'agrégateur pendant la collection des données fournies par des capteurs

L'agrégateur reste comme leader jusqu'à la fin de sa batterie, tenant compte que la batterie d'un capteur qui agit comme l'agrégateur diminue plus rapidement que la batterie d'un capteur régulier (figure 18), afin d'empêcher l'épuisement total de la batterie des agrégateurs, Larrea et al présentent un seuil d'épuisement de batterie. Quand l'agrégateur détecte un niveau bas de batterie, c.-à-d., un niveau de batterie au-dessous de seuil, il induit le système pour choisir un autre agrégateur. Et le choix d'un certain seuil détermine le QoS du réseau de capteurs, mesuré comme nombre de capteurs qui resteront actifs pendant une période donnée, en existence d'un agrégateur commun.

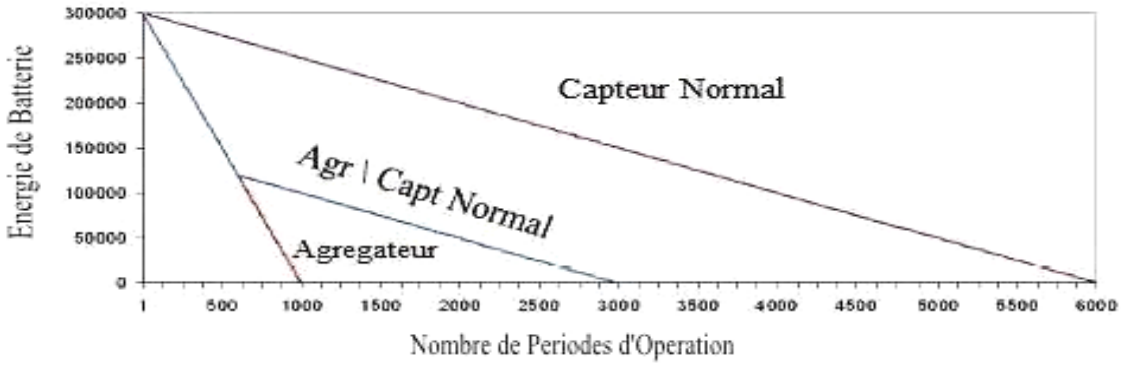


Figure 18 : comparaison de durée de vie de batterie de capteur.

Pour cela, ils introduisent une modification sur l'algorithme (figure 19) pour détecter si le niveau de batterie a diminué au-dessous du seuil ou pas. Dans le cas affirmatif, l'agrégateur augmente son nombre d'incarnation pour forcer par la suite l'algorithme à choisir un autre agrégateur.

```
...
(11)   if [leader_p = p] then
(a1)       if [getBatteryLevel() ≤ BATTERY_THRESHOLD] then
(a2)           incarnation_p ← incarnation_p + 1
(a3)       end if
(12)       start tasks 1 and 2
(13)   else
...
```

Figure 19 : modification de l'algorithme (seuil)

3.4.2.1. Implémentation de oméga pour le niveau local (Intra- région):

Larrea & al ajoutent en [90], en plus de l'algorithme (figure 17) [89], d'autres algorithmes :

Le premier algorithme (figure 17) d'implémentation de oméga pour un niveau local est la même que dans [89], il exige que les identificateurs des capteurs soient totalement ordonnées, mais pas nécessairement consécutifs, et les nœuds n'ont pas besoin de connaître les identificateurs du reste des nœuds à l'avance. Ils supposent aussi que tous les nœuds d'une région peuvent communiquer directement.

Le coût de l'algorithme, mesuré comme nombre de messages pendant une période d'acquisition de données, est linéaire par rapport au nombre de capteurs dans la région, où le noeud agrégateur annonce un message par Tâche 1, et le reste des noeuds envoient un message à l'agrégateur par Tâche 2 (algorithme figure 17).

Le deuxième algorithme (figure 20) est aussi pour l'implémentation de Ω dans un réseau de capteurs sans fil pour le niveau local, mais contrairement au premier algorithme (figure 17), les auteurs supposent que chaque paire de nœuds capteurs d'une région peuvent ne communiquer pas directement, mais il existe un sous-ensemble de nœuds dans cette région qui peuvent atteindre directement le reste de nœuds de la même région, et peuvent également recevoir les messages diffusés par chaque nœud de la région.

Cette hypothèse implique que les nœuds doivent connaître les identificateurs de reste de nœuds à l'avance. Et pour des raisons de clarté de l'algorithme, les auteurs n'ont pas inclus le mécanisme hibernation/ make-up.

Avec cet algorithme, chaque nœud $p \in correct$ à de manière permanente $leader\ p = l$, ou l le nœud le moins suspecté, parmi ceux qui peuvent communiquer directement avec le reste des nœuds de la région.

Every sensor p executes the following:
procedure *updateLeader*()
(1) *leader p* ← *l* such that (*counter p[l], l*) = min{(*counter p[q], q*) : *q* ∈ *candidates p}*
end procedure

Initialization:
(2) increment *incarnation p* by 1 in stable storage
(3) read (*incarnation p*) from stable storage
(4) ∀ *q* ≠ *p* : *Timeout p[q]* ← *η* + *incarnation p*
(5) ∀ *q* ≠ *p*: reset *timer p(q)* to *Timeout p[q]*
(6) ∀ *q* ≠ *p*: *counter p[q]* ← 0
(7) *counter p[p]* ← *incarnation p*
(8) *candidates p* ← *{p}*
(9) *leader p* ← *p*
(10) **start tasks** 1, 2 and 3
Task 1:
(11) **loop forever**
(12) *data p* ← acquire sensed data
(13) broadcast (I-AM-ALIVE , *p*, *counter p*, *data p*)
(14) wait(*η*)
Task 2:
(15) **upon reception of** message (I-AM-ALIVE, *q*, *counter q*, *data q*) **do**
(16) reset *timer p(q)* to *Timeout p[q]*
(17) ∀ *r* : *counter p[r]* ← max{(*counter p[r], counter q[r])}*
(18) *candidates p* ← *candidates p* ∪ *{q}*
(19) *updateLeader*()
(20) **if** [*leader p* = *p*] **then**
(21) collect *data q*
(22) **end if**
Task 3:
(23) **upon expiration of** *timer p*(*q*) **do**
(24) *counter p[q]* ← *counter p[q]* + 1
(25) *candidates p* ← *candidates p* − *{q}*
(26) *Timeout p[q]* ← *Timeout p[q]* + 1
(27) reset *timer p(q)* to *Timeout p[q]*
(28) *updateLeader()*

Figure 20 : le deuxième algorithme 2 d'implémentation de Ω dans un réseau de capteurs sans fil pour le niveau local.

L'algorithme (figure 20) fonctionne comme suit : chaque nœud *p* a un ensemble, *candidates p,* des nœuds considérés comme vivants. Le chef sera choisi, parmi ces nœuds, par *p*. Pour cela, *p* a également *counter p [q]* pour chaque nœud *q,* qui désigne le nombre de fois ou *q* est suspecté. Le nœud *p* choisit son leader le nœud *l* dans *candidates p* avec la plus petite valeur *de counter p [l]*. Afin d'acquérir les données captées, et maintenir à jour les variables *candidates p* et *counter*

p, chaque nœud *p* diffuse dans chaque η unités de temps le message (I-AM-ALIVE, *p*, *counter p*, *data p*), ou η est l'intervalle entre les mesures des capteurs. Si un nœud *p* reçoit un message I-AM-ALIVE, il remet *timer p* (q), pour que quand il reçoit le prochain message (I-AM-ALIVE, *q*, *counter q*, *data q*), il fait la mise à jour de *counter p*, en incluant *q* dans *candidates p* et appelle la procédure *update Leader* (). Si *p* est le leader, il collecte *data q*.

Si *timer p* (*q*) s'expire avant de recevoir un nouveau message (I-AM-ALIVE, *q*, *counter q*, *data q*), *p* incrémente *counter p [q]* , enlève *q* de *candidates p* , incrémente *Timeout p [q]* , remet *timer p (q)* , et également appelle *update Leader ()*. Les prochains messages envoyés par *p* incluront l'incrément de *counter p [q],* et les nœuds de cette manière peuvent connaître que *p* a suspecté *q*.

Le deuxième algorithme (figure 20) inclut un mécanisme pour empêcher les nœuds instables de perturber l'élection de leader. Ce mécanisme est basé sur l'incarnation de nœud. Pendant l'initialisation, chaque nœud initialise son timeout à η + *incarnation p (lignes* 4, figure 20). Et *p* initialise *counter p [p]* à *incarnation p (lignes* 7). Ces initialisations assurent que : chaque nœud p instable ne suspectera jamais un nœud correct *q,* qui peut communiquer directement avec chaque autre nœud, et par conséquence p n'incrémente plus *counter* p*[q]* et chaque noeud instable *p* ne sera jamais élu en tant que leader dans la procédure *updateLeader* ( ). Comme dans le premier algorithme (figure 17), le nombre de messages envoyés pendant une période d'acquisition de données η est linéaire au nombre des capteurs dans la région, puisque chaque capteur annonce un message par Tâche 1.

Le troisième algorithme d'implémentation de Ω dans un réseau de capteurs sans fil pour le niveau local (figure 21), contrairement au deuxième algorithme (figure 20), il n'exige pas une communication directe entre un nœud et le reste, mais seulement l'existence d'un chemin bidirectionnel d'un certains noeuds correct au reste de nœuds. En outre, pareillement au premier algorithme (figure 17), les nœuds n'ont pas besoin de connaître les identificateurs du reste de nœuds à l'avance.

Every sensor p executes the following:
procedure *updateLeader* ()
(1) *leader p* ← sensor in min*{candidates p}* **end procedure**
Initialization:
(2) increment *incarnation p* by 1 in stable storage
(3) read (*incarnation p*) from stable storage
(4) *membership p* ← *{p}*
(5) *candidates p* ← *{(p, incarnation p)}*
(6) *leader p* ← *p*
(7) **start tasks** 1, 2 and 3
Task 1:
(8) wait (*incarnation p*)
(9) **loop forever**
(10) *data p* ← acquire sensed data
(11) broadcast (I-AM-ALIVE, *p*, *candidates p*, *data p*)
(12) wait (κ)
Task 2:
(13) **upon reception of** message (I-AM-ALIVE, *q*, *candidates q*, *data q*)
with $q \neq p$ for the first time **do**
(14) broadcast (I-AM-ALIVE, *q*, *candidates q*, *data q*)
(15) **if** [$q \notin$ *membership p*] **then**
(16) *membership p* ← *membership p* $\cup$ *{q}*
(17) create *timer p*(*q*) and *Timeout p*[*q*]
(18) *Timeout p*[*q*] ← κ + *incarnation p*
(19) **end if**
(20) reset *timer p*(*q*) to *Timeout p*[*q*]
(21) **if** [$\exists$(*q, v*) $\in$ *candidates p*] **then**
(22) replace in *candidates p* (*q, v*) by (*q*, max *{v, v'}*): (*q, v'*) $\in$ *candidates q*
(23) **else**
(24) include in *candidates p* (*q, v'*) : (*q, v'*) $\in$ *candidates q*
(25) **end if**
(26) **if** [(*p*,−) $\notin$ *candidates q*] **then**
(27) replace in *candidates p* (*p, v*) by (*p, v* + 1)
(28) **end if**
(29) *updateLeader*()
(30) **if** [*leader p* = *p*] **then**
(31) collect *data q*
(32) **end if**
Task 3:
(33) **upon expiration of** *timer p*(*q*) **do**
(34) *Timeout p*[*q*] ← *Timeout p*[*q*] + 1
(35) remove (*q*,−) from *candidates p*
(36) *data p* ← acquire sensed data
(37) broadcast (I-AM-ALIVE, *p*, *candidates p*, *data p*)
(38) *updateLeader* ()

Figure 21 : le troisième algorithme 3 d'implémentation de Ω dans un réseau de capteurs sans fil pour le niveau local.

Le nœud choisi comme agrégateur par le noeud *p* est celui avec la valeur minimum associée parmi les candidats à devenir agrégateur (min {candidates p}), L'algorithme utilise une variable *membership p* pour stocker les identificateurs des différents nœuds vus jusqu'ici. Une autre variable *candidates p*, contient un ensemble de tuples (*q, v*), un pour chaque candidat, où *q* est l'identificateur de noeud et *v* est le nombre de fois que les noeuds ont suspectées *q*.

L'algorithme fonctionne comme suit : dans la tâche 1, les nœuds diffusent des messages périodiquement pour essayer de devenir l'agrégateur, aussi bien que pour envoyer leurs données captées, avec une périodicité de κ. Chaque message envoyé par un noeud p contient l'ensemble *candidates p* et *data p*. Dans la tâche 2, si un noeud *p* reçoit un message (I-AM-ALIVE, *q*, *candidates q*, *data q*) avec $q \neq p$ pour la première fois, il rediffuse le message pour essayer d'atteindre tous les noeuds de la région, met à jour *membership p* et crée *timer p* (*q*) et *Timeout p [q]*, remise *timer p (q)* , et met à jour en conséquence *candidates p*. Puis, *p* appel le procédure *updateLeader* (). Et à la fin, si *p* est l'agrégateur, il collecte *data q*.

Dans la tâche 3, si *timer p* (*q*) s'expire avant qu'un nouveau message I-AM-ALIVE de *q* est reçu, *p* "suspecte"*q*. Il incrémente *Timeout p* [*q*], enlève *q* de *candidates p*, envoie un message I-AM-ALIVE avec *candidates p* mis à jour, et appel *updateLeader* (). Pour éviter que les noeuds instables perturbent l'élection de l'agrégateur, pendant l'initialisation chaque noeud *p* initialise *candidates p* avec (*p, incarnation p*) (ligne 5).En outre, dans la tâche 1, *p* attend *incarnation p* unités du temps (ligne 8) avant d'exécuter le loop. Cette attente assure que dans la tâche 2 chaque capteur instable *p*, après la reprise, placera *leader p* à l'agrégateur de sa région avant l'exécution de la ligne 9, y compris l'agrégateur de manière permanente dans *candidates p*. En conséquence, l'agrégateur ne se punira pas soit même à cause de messages I-AM-ALIVE envoyés par *p*.

Le nombre de messages envoyés pendant une période d'acquisition de données (κ) est quadratique par rapport au nombre de capteur dans la région, puisque chaque capteur diffuse un message par tâche1, et les capteurs rediffusent les messages reçus. Et chaque nouvelle suspicion implique un nombre linéaire additionnel de messages.

3.4.2.2 Implémentation hiérarchique d'oméga pour le niveau global (Inter-région):

Les algorithmes présentés jusqu'à ici sont concentrés sur l'acquisition des données captées dans une région donnée d'un réseau de capteurs, et l'application de l'algorithme aux agrégateurs des différentes régions, nous permet de rassembler toutes les données fournies par le réseau de capteurs.

Larrea & al [89] décrivent une adaptation hiérarchique de leur algorithme, et en [90] ils présentent un algorithme d'agrégation de données pour le niveau global (figures 22, 23). Il présente une adaptation du premier algorithme (figure 17) pour le niveau local, mais exécuté seulement parmi les agrégateurs des différentes régions pour choisir le super- agrégateur et pour collecter les données captées par le réseau. Ils supposent que chaque paire d'agrégateurs peut communiquer, directement ou indirectement (par rediffusion). Comme dans le premier algorithme pour le niveau local (figure 17), les auteurs supposent que les agrégateurs n'ont pas besoin de connaître les identificateurs du reste d'agrégateurs à l'avance. Avec cet algorithme (figure 22, 23), tous les agrégateurs choisissent en tant que super –agrégateur, l'agrégateur avec le minimum de nombre d'incarnation. Le plus intéressant dans cet algorithme, et différemment à l'implémentation hiérarchique en [89], est qu'il permet aux différentes régions à exécuter n'importe lequel des trois algorithmes (figures 17, 20, 21) pour le niveau local (d'intra-région).

L'algorithme global (figure 22, 23) emploie les variables *super leader p*, *incarnation super leader* et *Timeout Super p*. Quand l'algorithme I (figure 17) d'intra- région est exécuté, ces variables sont lues dans un support de stockage pendant l'initialisation et écrites dans le support de stockage au temps d'exécution de la procédure *GoToHibernation* ( ); autrement, ils sont initialisées à *p*, *incarnation p* et Δ *OPERATION* + *incarnation p*, respectivement.

L'algorithme hiérarchique proposé par Learra et al fonctionne sous les hypothèses suivantes :

(1) Il y a un chemin opportun (probablement avec les sauts multiples) du super -agrégateur au reste d'agrégateurs, aussi bien que de chaque agrégateur au super -agrégateur;

(2) Δ *OPERATION* est placé à une valeur tels que :

(a) le message I-AM-THE-SUPER-LEADER diffusé par le super- agrégateur atteint le reste d'agrégateurs, et (b) l'émission de message de DIGEST par chaque agrégateur atteint le super - agrégateur ;

(3) Δ *DATA ACQUISITION* ≥ Δ *OPERATION* si cet algorithme est exécuté combinait avec le premier algorithme de niveau local (figure 17).

Every sensor p executes the following:
Initialization (for both intra- and inter-region):
(1) **if** [intra-region algorithm is I] **then**
(2) add the following instruction to *GoToHibernation*() (after Line 1):
(3) write (*super leader p*, *incarnation super leader*, *Timeout Super p*) in stable storage
(4) execute the Initialization of Algorithm I (Lines 4-16 of Figure 1):
(5) read (*super leader p*, *incarnation super leader*, *Timeout Super p*) from stable storage
(6) **else**
(7) **if** [intra-region algorithm is II] **then**
(8) execute the Initialization of Algorithm II (Lines 2-10 of Figure 2):
(9) **else if** [intra-region algorithm is III] **then**
(10) execute the Initialization of Algorithm III (Lines 2-7 of Figure 3):
(11) **end if**
(12) *super leader p* ← *p*
(13) *incarnation super leader* ← *incarnation p*
(14) *Timeout Super p* ← Δ*OPERATION* + *incarnation p*
(15) **end if**
(16) **start tasks** 4, 5, 6 and 7

Figure 22 : algorithme de niveau global (partie 1)

Task 4:
(17) **loop forever**
(18) **if** [*leader p = p*] **then**
(19) *received from super* ← *FALSE*
(20) **if** [*super leader p = p*] **then**
(21) **if** [intra-region algorithm is I] **then**
(22) wait _ time units
(23) **end if**
(24) broadcast (I-AM-THE-SUPER-LEADER, *p*, *incarnation p*)
(25) **else**
(26) reset *timer super p* to *Timeout Super p*
(27) **end if**
(28) **end if**
(29) wait Δ*OPERATION* time units
Task 5:
(30) **upon reception of** message (I-AM-THE-SUPER-LEADER, *q*, *Incarnation q*) with $q \neq p$ for the first time **do**
(31) **if** [*leader p = p*] **then**
(32) **if** [*incarnation q* < *incarnation super leader*] or [(*incarnation q = incarnation super leader*) and ($q \leq$ *super leader p*)] **then**
(33) *super leader p* ← *q*
(34) *incarnation super leader* ← *incarnation q*
(35) broadcast (I-AM-THE-SUPER-LEADER, *q*, *incarnation q*)
(36) broadcast (DIGEST, *p*)
(37) *received from super* ← *TRUE*
(38) **end if**
(39) **end if**
Task 6:
(40) **upon reception of** message (DIGEST, *q*) with $q \neq p$ for the first time **do**
(41) **if** [*leader p = p*] **then**
(42) **if** [*super leader p = p*] **then**
(43) collect DIGEST
(44) **else**
(45) broadcast (DIGEST, *q*)
(46) **end if**
(47) **end if**
Task 7:
(48) **upon expiration of** *timer super p* **do**
(49) **if** [*leader p = p*] **then**
(50) **if** [*received from super = FALSE*] **then**
(51) super leader p ← *p*
(52) *incarnation super leader* ← *incarnation p*
(53) *Timeout Super p* ← *Timeout Super p* + Δ *TIMEOUT*
(54) **end if**
(55) **end if**

Figure 23 : algorithme de niveau global (partie 2)

3.5 Conclusion :

Dans ce chapitre, nous avons présenté comment le service d'élection de leader est utile pour établir la tolérance aux fautes dans un réseau de capteurs, et pour cela on a étudié les algorithmes d'élection de leader dans les réseaux de capteurs, après un aperçu sur le paradigme de consensus, les détecteurs de défaillances, et l'élection de leader.

Chapitre 4 :

Agrégation de données en utilisant un détecteur de défaillance de la classe Ω

4.1 Introduction :

Les réseaux de capteurs sans fil (WSN) fournissent une collection de données fiables par des applications réduisant au maximum l'intervention humaine (auto-organisation et auto-maintenance). Les applications peuvent agir de façon autonome au-dessus d'un réseau de capteurs en configurant les capteurs à distance et récupérant les données collectées par ces capteurs périodiquement ou sur demande. Un réseau de capteurs peut être exposé à plusieurs sources des problèmes tels que la communication, crash et/ou erreurs d'alimentation par énergie.

Les deux facteurs importants à considérer dans les réseaux de capteurs sans fil sont :1) le coût global de déploiement, qui peut être réduit en utilisant des nœuds de capteurs à prix

réduit; et 2) la durée de vie limitée par la consommation d'énergie, qui peut être réduit en minimisant les délais des opérations et en activant l'hibernation de capteurs. De cette façon, dans un réseau de capteurs sans fil, les applications doivent assurer la collecte et l'agrégation de données fiable, tout en satisfaisant les contraintes de telles applications avec un coût réduit et une énergie réduite. Dans ce travail nous nous intéressons à l'agrégation de données avec une prise en charge de contrainte de l'énergie.

Dans [89] Larrea et al ont utilisé le concept de détecteurs de défaillances non fiables introduits par Chandra et Toueg [83]. Ils ont proposé une implémentation du détecteur de défaillances de la classe Omega dans un réseau de capteurs avec le modèle de défaillances crash/reprise. Larrea et al supposent dans [89] qu'un capteur peut reprendre son exécution après l'épuisement de sa batterie par le remplacement ou le rechargement de celle-ci.

La reprise de fonctionnement d'un capteur après l'épuisement de sa batterie dans certaines situations est impossible, due à l'impossibilité de remplacement ou le rechargement de celle-ci lorsque le nombre de capteurs déployés est important ou quand le réseau est déployé dans des zones dangereuses (zones militaires, zones sous-marines).

Dans ce mémoire, nous proposons un modèle du système pour les réseaux de capteurs avec des hypothèses plus réalistes. Nous proposons également un protocole pour coordonner l'agrégation de données dans un réseau de capteurs sans fil en utilisant un détecteur de défaillances de la classe *Omega crash* (Ω). Plus précisément, nous adaptons le protocole de Larrea et al [89] pour un réseau de capteurs sans fil où un capteur peut tombe en panne par arrêt définitif. L'objectif de cette classe est d'assurer un accord sur l'identité d'un et d'un seul capteur qui sera un coordonateur pour l'agrégation de données.

4.2 Modèle de système :

Nous considérons un réseau de capteurs sans fil, où nous collectons les données de capteurs en minimisant la consommation d'énergie. La collection de données peut suivre un modèle adresse-centrale, qui consiste à trouver le chemin le plus court entre un pair de nœuds, ou un modèle data-centric, qui trouve les chemins de multiples sources vers une seule destination [91]. Nous considérons un modèle data-centrale pour effectuer l'agrégation de données.

Nous considérons un réseau de capteurs sans fil divisé en régions différentes. Chaque capteur connaît sa région et ces tâches de collection de données pour les transmettre à un agrégateur responsable de la collection et l'agrégation de toutes les données captées dans cette région.

L'agrégateur est choisi par les capteurs de sa région selon le critère de capacité énergétique, où ils choisissent le nœud avec le plus grand niveau de batterie. Chaque capteur a un identificateur unique de sa région qui est diffusé dans ses messages. Cet identificateur permet aux capteurs de rejeter les messages qui ne sont pas destinés à celle-ci.

Notre implémentation de oméga considère un système *S* composé d'un ensemble fini ∏ de n capteurs (n>1), $\prod = \{s_1, s_2, \ldots\ldots s_n\}$, qui communiquent seulement par échange de messages. Les identificateurs des capteurs sont totalement ordonnés.

Dans ce chapitre , nous considérons un système partiellement asynchrone, c'est-à-dire, il existe, à partir d'un moment non connu mais fini appelé temps global de stabilisation (GST), des bornes non connues mais finis sur les délais de transmission des messages, sur les vitesses relatives des processus et sur les dérives des horloges.

Les capteurs peuvent tomber en panne par crash dont l'épuisement de batterie est un des causes.

4.3 Détecteur de défaillances Ω

Chandra et al ont définis dans [85] la classe de détecteurs de défaillances Ω. Le détecteur de défaillances Ω permet aux processus d'invoquer une primitive *leader* qui retourne une identité d'un processus à chaque fois est invoquée. Ω garantit, après certain temps non connu mais fini, que toutes ces invocations retournent la même identité d'un processus correct du système. Ce type de détecteurs, n'est pas fiable, c'est-à-dire nous ne pouvons pas connaître quand un leader correct sera élu, et nous pouvons avoir plus d'un leader avant la vérification de propriétés de synchronie considérées dans le système.

Nous disons qu'un processus p_i est le *leader* à l'instant *t,* si p_i n'est pas défaillant à l'instant *t* et $leader_{pi} = vrai$. Formellement, nous définissons le problème d'élection d'un leader par les deux propriétés suivantes :

- **Accord** : à l'instant *t* il existe un seul processus leader (deux processus ne peuvent pas être leader en même temps).

- **Terminaison :** à tout moment, il existe finalement un leader.

Sabel et Marzullo [84] ont prouvé que le problème d'élection d'un leader est réductible au problème de consensus. Informellement, une utilisation du consensus pour résoudre l'élection d'un leader consiste à effectuer une décision sur le leader à élire.

Dans [85], Chandra, Hadzilacos et Toueg ont proposé un nouveau type de détecteurs de défaillances, dénoté Ω, et ont montré qu'il est le détecteur de défaillances le plus faible pour résoudre le consensus dans un système réparti asynchrone où la majorité des processus sont corrects. Informellement, le détecteur de défaillances Ω est une entité distribuée qui augmente les processus avec une fonction *leader ()*. Cette fonction retourne une identité d'un processus à chaque fois est invoquée. Formellement, le détecteur de défaillances Ω doit, satisfaire la propriété suivante :

***Propriété 1: Leadership inéluctable* :** il existe un instant *t* et un processus correct *p* tel que, après *t*, chaque invocation de *leader ()* par n'importe quel processus correct retourne *p*.

Cette définition de Ω considère que le leader se tient pour toujours. Dans notre modèle de système, en raison de l'énergie limitée de capteur, cette propriété ne peut pas être satisfaite. Pour cela, Nous définissons une deuxième propriété qui doit être satisfaite pour le capteur leader et nous définissons également un seuil α qui représente l'énergie nécessaire juste pour lancer une nouvelle opération d'élection. Formellement, le détecteur de défaillances Ω doit, satisfaire la propriété suivante :

Propriété 2 : Leadership inéluctable : *il existe un instant t et un processus correct p tel que, après t, chaque invocation de leader () par n'importe quel processus correct retourne p jusqu'à ce que son énergie se diminue sous un seuil α.*

4.4 Un algorithme implémentant Ω dans un RCSF :

Dans cette section, nous présentons notre algorithme qui implémente un détecteur de défaillances de la classe Ω dans un réseau de capteurs avec des défaillances par crash ou par épuisement de batterie. Cet algorithme peut être utilisé par n'importe quelle autre application qui se base sur l'existence de leader. Dans le cadre de notre mémoire nous allons montrer comment cette implémentation de Ω peut être utilisée pour l'agrégation de données.

Après l'élection d'agrégateur dans chaque région, et de même façon, l'algorithme permet aussi l'élection d'un super agrégateur par l'ensemble des agrégateurs (figure 24).

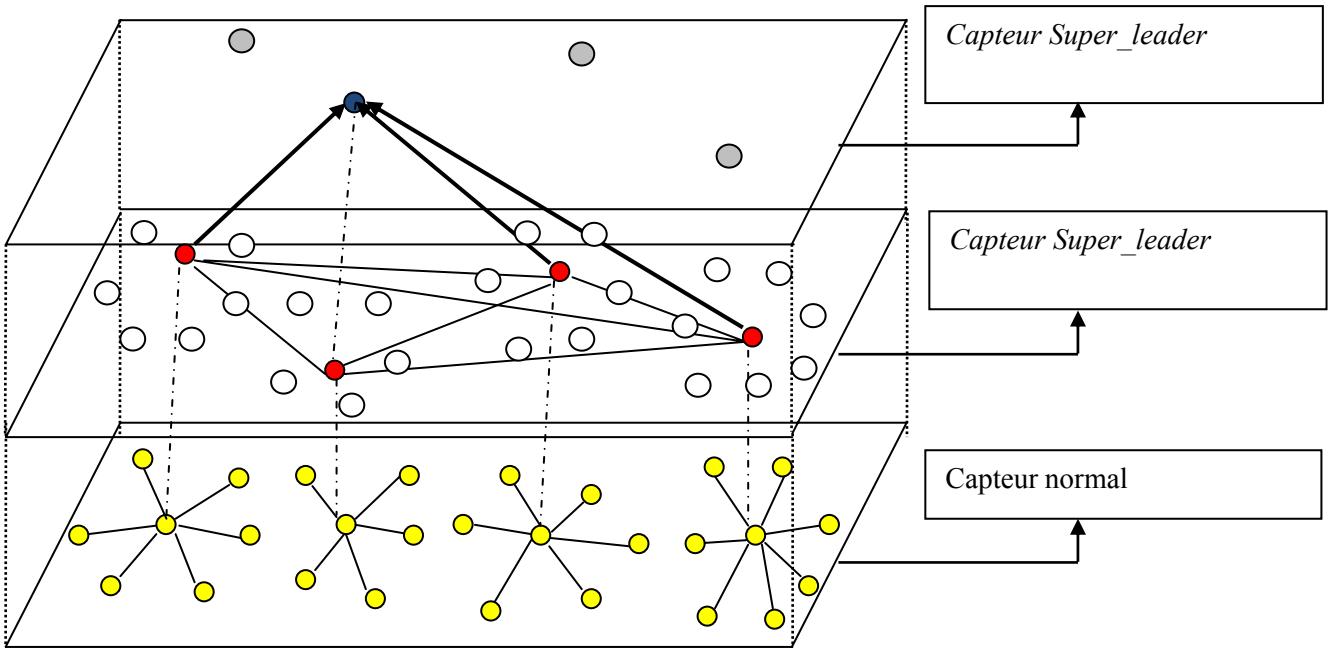


Figure 24: schéma de l'implémentation hiérarchique de Ω dans un réseau de capteurs sans fil.

Principe de l'algorithme :

La figure 25 présente en détail l'algorithme qui implémente Ω dans un réseau de capteurs sans fil. Chaque capteur s_i gère les variables locales suivantes :

R_i: est l'ensemble de capteur d'une région *i*.

k : représente le nombre de régions dans le réseau.

BL_i : est une structure de données pour stocker les valeurs de batteries de capteurs d'une région R_i .

$BL2_i$: est une structure de données pour stocker les valeurs de batteries de capteurs leaders de k régions.

Δ_i: représente les timeouts de capteurs d'une région i par rapport à s_i.

$\Delta 2_i$: représente les timeouts de capteurs leaders du réseau par rapport à s_i.

AG_i: est une structure de données pour stocker les identités des capteurs leaders.

La conception de cet algorithme est simple. chaque capteur s_i utilise deux fonctions : *new_leader()* (lignes 16..22) pour élire un leader pour une région R_i et *new_super_leader()* (lignes 23..29) pour élire un super leader pour tout le réseau.

Dans la fonction *new_leader()* chaque capteur s_i envoie un message INIT() à tous les capteurs de sa région et attend la réception des messages INIT() des autres capteurs de la même région ou l'expiration des timeouts. Une fois le timeout expiré le capteur incrémente le timeout. Une fois s_i reçoit les messages de capteurs corrects il choisit comme leader le capteur qui possède le niveau d'énergie le plus élevé. Dans le cas où plus d'un capteur ont le même niveau de batterie, s_i choisit le capteur avec la plus petite identité.

La fonction *new_super_leader()* suit le même principe à l'exception que l'élection d'un super leader se fait seulement entre les leaders des différentes régions.

Each sensor s_i executes the following:

$R_i \leftarrow$ the set of sensors in the region i
$BL_i[1..|R|] \leftarrow \perp$; $BL2_i[1..k] \leftarrow \perp$; $\Delta_i[1..|R|] \leftarrow 1$; $\Delta 2_i[1..k] \leftarrow 1$ $AG_i[1..k] \leftarrow \perp$

Intialisation
(1) *new_leader(); new_super_leader()*

Repeat forever

---

(2) **If** ($s_i \neq Leader_i$)and ($s_i \neq Super\_Leader_i$) **then**
(3) wait receive leader(IdR_i) from $Leader_i$ **or** $\Delta_{i,leader}$;
(4) **if** times out **then** *new_leader();*
(5) **If** ($s_i = Leader_i$) **and** ($s_i \neq Super\_Leader_i$) **then**
(6) **if** BL_i [i]>α
(7) **broacast** *leader* (s_i, IdR_i) ;
(8) **upon** reception *leader* (s_j, IdR_j) from s_j **then** $AG_i[IdR_j] \leftarrow Leader_j$;
(9) **wait** receive *super_leader* (s_j) from $Super\_Leader_i$ **or** $\Delta 2_{i,superleader}$;
(10) **if** times out **then** *new _ super _leader();*
(11) **else** *new_leader();*
(12) **If** ($s_i = Super\_Leader_i$) **then**
(13) **if** BL_i [i]>α
(14) **send** *super_leader* (s_i) to each sensor in AG_i;
(15) **else** *new _ super _leader();*

End repeat
(16) **Function** *new_leader();*
(17) *Send INIT* (s_i, IdR_i, BL_i) to all sensors in R_i;
(18) For each sensor in R_i **wait** receive *INIT* (S_j , IdR_i, BL_j) or $\Delta_{i,j}$;
(19) **If** receive *INIT*(S_j , IdR_i, BL_j) and $IdR_i = IdR_j$ then **store value** in BL_i;
(20) **else** $\Delta_{i,j} \leftarrow \Delta_{i,j} + 1$;
(21) $Leader_i \leftarrow s_k$ *with* $BL_i[k]$ *is the max in* BL_i ;
(22) **Return *leader ();***

(23) **Function** *new_super_leader*
(24) **Send** INIT2(s_i, BL_i)to all sensors in $AG_i[]$;
(25) For each sensor in AGi[] **wait** receive *INIT2*(S_j , BL_j) or $\Delta 2_{i,j}$;
(26) **If** receive *INIT2*(S_j , BL_j) then **store value** in $BL2_i$;
(27) **Else** $\Delta 2_{i,j} \leftarrow \Delta 2_{i,j} + 1$;
(28) $Super\_Leader_i \leftarrow S_k$ *with* $BL2_i[k]$ *is the max* $BL2_i$;

(29) **Return *Super_Leader ();***

Figure 25: Une implémentation hiérarchique de Ω dans un réseau de capteurs sans fil

À l'initialisation (ligne 1) chaque capteur s_i exécute les deux fonctions précédentes pour sélectionner un leader s'il est un capteur normal ou pour sélectionner un super leader s'il est leader de sa région. Si s_i n'est pas un leader ou super leader (ligne 2 ..4), il attends la réception périodique d'un message *leader()* de $leader_i$ ou le timeout $\Delta_{i,leader}$. Si le timeout est expiré, il détecte que le leader est tombé en panne et il lance une nouvelle élection par l'invocation de la fonction *new_leader()*. Si s_i est le leader de sa région, il envoie périodiquement à tous les capteurs de cette région un message *leader(*IDR_i*)* jusqu'à ce que le niveau de sa batterie atteins un seuil α, après il invoque une nouvelle opération d'élection d'un leader pour la région. s_i attend en même temps la réception d'un messages *super_leader() de* $super\_leader_i$ ou le timeout $\Delta 2_{i,superleader}$. Si le timeout est expiré, il lance une nouvelle opération d'élection d'un super leader pour tout le réseau. Si s_i est le super leader, il envoie périodiquement un message *super_leader()* à tous les leader des différentes régions jusqu'à ce que le niveau de sa batterie atteins un seuil α, après il invoque une nouvelle opération d'élection d'un nouveau super leader pour le réseau.

4.5 La preuve de l'algorithme

Lemme1 : *deux capteurs corrects ne peuvent pas être leaders en même temps de la même région.*

Preuve : Avant le temps globale de stabilisation du système (GST) l'algorithme peut faire des erreurs, mais après GST si un capteur s_i d'une région R_i invoque la fonction leader, automatiquement elle va retourner une seule identité d'un capteur de cette région. Nous pouvons avoir deux capteurs avec le même niveau de batterie et avec des identités déférentes, donc il est impossible de sélectionner le même leader dans une région donnée. Nous avons ajouté le critère de comparaison des identités (il est possible de trouver plus capteur avec le même niveau d'énergie, donc nous prenons le capteur avec l'identité minimale) pour avoir une relation d'ordre totale entre les capteurs de la même région.

Lemme2 : *deux leaders de deux régions différentes ne peuvent pas être super leader en même temps.*

Preuve : la preuve de ce lemme est similaire ou premier, seulement l'élection se fait entre les leaders des différentes régions.

Lemme3 : *si le système est partiellement synchrone, alors chaque invocation de leader () retourne l'identité d'un capteur.*

Preuve : Si les propriétés de synchronisme sont satisfaites, c'est-à-dire, il existe des bornes non connues mais finies sur les délais de communication et les vitesses relatives de processus. Donc il est impossible qu'un capteur se bloque à une étape d'attente (lignes : 3, 18). Chaque capteur attend la réception d'un message pour se débloquer. S'il n'a rien reçu attend l'expiration du timeout pour se débloquer.

Lemme4 : *si le système est partiellement synchrone, alors chaque invocation de super_leader () retourne l'identité d'un capteur.*

Preuve : la preuve de ce lemme est similaire au lemme3.

Théorème : *l'algorithme de la figure 25 implémente hiérarchiquement un détecteur de défaillance de la classe* Ω *dans un réseau de capteur sans fil.*

Preuve : à partir du lemme1, du lemme2, du lemme3 et du lemme 4, nous pouvons facilement déduire qu'après GST :

Tous les capteurs d'une région satisfont la propriété 2

4.6 Un algorithme pour l'agrégation de données en utilisant Ω:

La figure 26 représente un algorithme pour l'agrégation de données dans un réseau de capteurs sans fil. Cet d'algorithme d'agrégation se base sur un détecteur de défaillances hiérarchique Ω. Dans cet algorithme, nous nous intéressons seulement aux communications nécessaires pour l'agrégation de données. Les différentes fonctions utilisées pour l'agrégation de données et la nature de ces données sont ignorées dans cet algorithme. Le but de cet algorithme est de monter que l'agrégation de données peut être résolu en se basant sur la notion de détecteurs de défaillances Ω. L'objectif de l'utilisation de Ω est de minimiser le nombre d'étape de communication et le nombre de messages nécessaires pour l'agrégation de données.

Le principe de l'algorithme est très simple. Un capteur simple envoie périodiquement un message *data ()* au leader de sa région. Si le capteur est leader, il reçoit périodiquement des messages *data ()* de capteurs et procède la tâche d'agrégation Aggregate *data ()* sur les données captées dans sa région, ensuite il envoie le résultat au super leader. Si le capteur est super leader, il reçoit les messages *data ()* envoyées par les leaders de chaque région, et applique la fonction d'agrégation sur les données reçues de différentes régions. La période d'agrégation de données peut être changée selon les besoins de l'application. Dans notre cas, nous avons pris juste un exemple sans faire référence à une application ou à ses exigences.

Each sensor s_i executes the following:
$\Delta_{DATA} \leftarrow 10\Delta_{i,leader}$, $\Delta 1_{DATA} \leftarrow 10\Delta 2_{i,superleader}$;
Repeat forever

If ($s_i \neq Leader_i$)and ($s_i \neq Super\_Leader_i$) **then**
Collecte *data()*
Send *data()* to leader$_i$;
If ($s_i = Leader_i$) **and** ($s_i \neq Super\_Leader_i$) **then**
Receive data from sensors in R_i during Δ_{DATA} ;
Aggregate *data ()* received from corrects sensors;
Send *data ()* to $Super\_leader_i$;
If ($s_i = Super\_Leader_i$) **then**
Receive data from sensors in $AG_i[]$during $\Delta 1_{DATA}$;
Aggregate *data ()* received from corrects leaders;
end repeat

Figure 26: Aggregation de données en utilisant Ω

4.7 **Conclusion:**

Dans ce chapitre, nous avons proposé deux algorithmes. Le premier pour implémenter un détecteur de défaillance Ω dans un réseau de capteurs sans fil dont le modèle de défaillance est par crash. Cet algorithme se base sur le synchronisme partiel contrairement à l'algorithme de Larrea et al [89] qui exige des hypothèses de synchronisation très fortes. Dans cet algorithme nous n'avons pas utilisé des périodes de mise en veille puisque, il est très difficile de synchroniser ces périodes. Les hypothèses de notre algorithme sont plus réalistes que celles de Larrea et al. Le deuxième algorithme présente un algorithme d'agrégation de données en se basant sur le premier algorithme. Contrairement à l'algorithme de Larrea et al qui regroupe l'agrégation et la détection d'un leader, dans notre proposition nous avons séparé entre les deux d'une façon modulaire pour rendre la solution plus lisible, plus facile à comprendre et pour respecter la notion de détecteur de défaillances qui représente une boite noir qui peut être utilisée par n'importe quelle application.

Conclusion Générale

Le travail présenté dans ce mémoire contribue à l'étude de la tolérance aux fautes dans les réseaux de capteurs sans fil et à la mise en œuvre des systèmes répartis pour la coordination d'agrégation de données dans ces réseaux. En effet, nous avons orienté notre étude sur l'utilisation des détecteurs de défaillance pour résoudre le problème d'élection des agrégateurs dans les réseaux de capteurs.

Dans ce mémoire, nous avons présenté un état de l'art sur les réseaux de capteur, ainsi qu'une étude des fautes qui peuvent survenir et les techniques de tolérance aux fautes dans ces réseaux. Nous avons présenté une synthèse sur les protocoles d'élection de leader dans les réseaux de capteurs.

La principale contribution de ce mémoire, après une critique de l'algorithme de Larrea et al, est la proposition de deux algorithmes, le premier pour implémenter un détecteur de défaillance Ω dans un réseau de capteurs sans fil dont le modèle de défaillance est par crash.

Le deuxième algorithme présente un algorithme d'agrégation de données en se basant sur le premier algorithme.

Perspectives :

Les perspectives de ce mémoire sont diverses, nous en citons ci-après deux entre elles.

Utiliser l'implémentation hiérarchique d'Omega.

L'algorithme que nous avons proposé pour implémenter Omega peut être utilisé pour résoudre tous les problèmes qui ont besoin de service d'élection dans un réseau de capteurs sans fil.

Un algorithme pour l'auto-structuration d'un réseau de capteurs sans fil.

Pour que notre implémentation d'Omega soit efficace, elle doit se baser sur un algorithme d'auto-structuration efficace qui consomme le moins d'énergie puisque notre protocole se base sur les résultats d'un algorithme d'auto structuration efficace en énergie.

Références

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci. A survey on sensor networks IEEE Communications Magazine, 40(8):102-114, August 2002.

[2] M. A. M. Vieira and al. Survey on wireless sensor network devices. In 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'03), vol. 1, 537-544. Lisbon, Portugal, 16-19 September 2003.

[3] Imrich Chlamtac, Iacopo Carreras ET Hagen Woesner «From internets to bionets: biological kinetic service oriented networks". The case study of Bionetic Sensor Networks. pp 75-95 , Advances in Pervasive Computing and Networking , Springer US 2005.

[4] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: a survey. IEEE Network, vol.18, no.4, pp.45–50, July/August 2004.

[5] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan. Building efficient wireless sensor networks with low-level naming. In Proceedings of the 18th ACM Symposium on Operating Systems Principles, pages 146–159, ACM Press, 2001.

[6]. B. Krishnamachari, D. Estrin, and S. Wicker. Impact of data aggregation in wireless sensor networks. In International Workshop of Distributed Event Based Systems (DEBS), pages 575-578, July 2002.

[7]. S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. Ght: A Geographic hash table for data-centric storage. In Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, pages 78–87, ACM Press, 2002.

[8]. S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data centric storage in sensor nets with ght, a geographic hash table. Mobile Networks and Applications, 8(4):427–442, 2003.

[9]. S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman. Infrastructure tradeoffs for sensor networks. In Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, pages 49–58, ACM Press, 2002.

[10] K. Weniger and M. Zitterbart. Adress autoconfiguration in mobile ad hoc networks: Current approches and future directions. IEEE Network, 18(4): pages 6–11, July-August 2004.

[11] M. Günes, J. Reibel, An IP address configuration algorithm for zeroconf. mobile multi-hop ad-hoc networks, in: Proceedings of the International Workshop on Broadband Wireless Ad-Hoc Networks and Services, Sophia Antipolis, 2002.

[12] N. H. Vaidya. Weak duplicate address detection in mobile ad hoc networks. In ACM MobiHoc 2002, pages 206–216, June 2002.

[13] Iwata, A., Chiang, C. C., Pei, G., Gerla, M., and Chen, T. W. Scalable routing strategies for ad how wireless networks. IEEE Journal on Selected Areas in Communications pp1369–1379, August 1999.

[14] Perkins, C., and Bhagwat, P. Highly dynamic destination-sequenced distance vector routing (dsdv) for mobile computers. ACM SIGCOMM, pp234–244, October 1994.

[15] Murthy, S., and Garcia-Luna-Aceves, J. J. A routing protocol for packet radio networks. Proc. IEEE Mobicom pp1369–1379, August 1999.

[16] Corson, M. S., and Ephremides, A. A distributed routing algorithm for mobile wireless networks. ACM-Baltzer J. Wireless Networks, pp61–81, January 1995.

[17] Park, V. D., and Corson, M. S. A highly adaptive distributed routing algorithm for mobile wireless networks. Proc. IEEE Infocom , pp1405–1413,1997.

[18] H.M.Ammari S.K.Das. On Computing Conditional Fault-Tolerance Measures for k-Covered Wireless Sensor Networks. MSWiM, pp 309-316 ACM 2006

[19] A.S. Tanenbaum, M.V.Steen. Distributed Systems, Principles and Paradigms Prentice Hall, 2002.

[20] J.-C laprie.” Dependable computing and fault tolerance: concepts and terminology”. Dans proceeding of the 15th International Sympsium on Fault-Tolerant Computing, pages 2_11, Ann Arbor, Michigan, USA, Juin 1985.

[21] J. Arlat, J.-P Blanquart, A. Costes, & all. « Guide de la sûreté de fonctionnement ». Cépaduès, 1995.

[22] S. Kim, R. Fonseca and D. Culler: Reliable Transfer on Wireless Sensor Networks, Proc, pp. 449–459, IEEE SECON, 2004.

[23] F. Taiani. La Réflexivité Dans Les Architectures Multiniveaux: Application Aux Systemes Tolerant Les Fautes. These de doctorat de IT universite Paul Sabatier de Toulouse. 2004.

[24] A. Arora et all. *Gouda: Closure And Convergence: A Foundation Of Fault-Tolerant Computing.* IEEE Transactions on Software Engineering, Vol 19(11) :1015-1027. 1993.

[25] A. Postma, Classes of Byzantine fault-tolerant algorithms for dependable distributed systems, Ph.D. thesis, University of Twente, Enschede, The Netherlands, 1998.

[26] G. L. Lann, P. Minet et D. Powell : *Tolerance Aux Fautes Et Systemes Repartis: Concepts Et Mecanizmes.* Rapport de recherche INRIA n° 2108. 1993.

[27] V. P. Neison. *Fault-Tolerant Computing: Fundamental Concepts. IEEE* Computer, vol. 23, no. 7. 1990.

[28] D.F. Macedo, L.H.A. Correia, A.L. dos Santos, A.A.F. Loureiro, J.M. Nogueira, Evaluating fault tolerance aspects in routing protocols for wireless sensor networks, in: Fourth Annual Mediterranean Ad Hoc Networking Workshop, 2005.

[29] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, pages 307–322, Jan 2004.

[30] N. Reijers, G. Halkes, and K. Langendoen. Link layer measurements in sensor networks. In *1st IEEE Int. Conference on Mobile Ad hoc and Sensor Systems (MASS '04)*, Oct 2004.

[31] B. Walke, N. Esseling, J. Habetha, A. Hettich, A. Kadelka, S. Mangold, J. Peetz, and U. Vornefeld. IP over Wireless Mobile ATM - Guaranteed Wireless QoS by HiperLAN/2. *Proceedings of the IEEE*, 89:21–40, Jan 2001.

[32] Larry L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach, 3rd Edition*. Morgan Kaufmann Publishers Inc., 2003.

[33] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *Computer*, 35(10):54–62, 2002.

[34] J.C Laprie. Dependability - its Attributes, Impairments and Means, *In Predictably Dependable Systems. Springer – Verlag.*1995.

[35] A. Avizienis. *Design Diversity And The Immune System Paradigm: Cornerstones For Information System Survivability.* UCLA Computer Science CA 90095-1596. 2000.

[36] F. Cristian. *Understanding Fault-Tolerant Distributed Systems.* In: Communications of the ACM. Vol.34, no.2. 1991.

[37] M. Barborak, M. Malek et A. Dahbura. *The Consensus Problem In Fault- Tolerant Computing.* In: ACM Computing Surveys. Vol.25, no.2. 1993.

[38] F. Cristian, , H. Aghili, R.Strong, et D. Dolev. *Atomic Broadcast: From Simple Message Diffusion To Byzantine Agreement, In: Information And Computation* Proceedings of the 15th International Symposium on Fault-Tolerant Computing (FTCS-15), Ann Arbor. 1985.

[39] L. Lamport, R. Shostak, et M. Pease. *The Byzantine Generals Problem.* In: ACM Transactions on Programming Languages and Systems, Vol. 4, no. 3. 1982.

[40] A. Arora & all. A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. Computer Networks, Volume 46, Issue 5, 5 December 2004, Pages 605-634

[41] H. Asaad et E. Czeck. *Concurrent Error Correction In Iterative Circuits By Recomputing With Partitining AND Voting.* IEEE Transactions on Software Engineering, Vol. 0-8186-3830-3/93. 1993.

[42] M. Manic et D. Frincke. *Towards the Fault Tolerant Software: Fuzzy Exrension Of Crisp Equivalence Voters.* The 27th Annual Conference of the IEEE Industrial Electronics Society, Page(s): 84-89 vol.1 .2001.

[43] B. Krishnamachari, and S. Iyengar, "Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 241-250, 2004.

[44] Hai Liu, Amiya Nayak, and Ivan Stojmenovic, "Fault Tolerant Algorithms/Protocols in Wireless Sensor Networks," *Handbook of Wireless Ad Hoc and Sensor Networks*, Sudip Misra, Isaac Woungang, Subhas C. Misra (eds.), Springer-Verlag (London), 2008.

[45] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong, "A macroscope in the redwoods," In *Sen- Sys'05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, New York, NY, USA, 2005.

[46] A. Avizienis, J.P.J. Kelly, "Fault Tolerance by Design Diversity: Concepts and *Experiments"*, IEEE Computer, vol. 17 no. 8, August 1984, 67-80.

[47] A. Avizienis, "The N-Version Approach to Fault Tolerant Software," IEEE Trans. Soft. Eng., vol. SE-11, no.12, pp.1491-1501, 1985.

[48] Keith Marzullo. Tolerating Failures of Continuous-Valued Sensors. ACM Transactions on Computer Systems, Vol. 8, No. 4, November 1990, Pages 284-304. 1990

[49] Gafni, E. M., and Bertsekas, D. P. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Transactions on Communications (1981), pp11–18.

[50] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," IEEE Wireless Comm., vol. 11, pp. 6-28, 2004.

[51] J.-H. Chang and L. Tassiulas, “Maximum Lifetime Routing in Wireless Sensor Networks", Proc. Advanced Telecommunications and Information Distribution Research Program (ATIRP2000), College Park, MD, Mar. 2000.

[52] C. Rahul, J. Rabaey, “Energy Aware Routing for Low Energy Ad Hoc Sensor Networks" , IEEE Wireless Communications and Networking Conference (WCNC), vol.1, March 17-21, 2002, Orlando, FL, pp. 350-355.

[53] S. Dulman, T. Nieberg, J. Wu, P. Havinga, “Trade-off between Traffic Overhead and Reliability in Multipath Routing for Wireless Sensor Networks", WCNC Workshop, New Orleans, Louisiana, USA, March 2003.

[54] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," Proceedings of ACM MobiCom '00, Boston, MA, 2000, pp. 56-67.

[55] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, Directed diffusion: a scalable and robust communication paradigm for sensor networks ", ACM SIGMOBILE Mobile Computing and Communications Review, vol. 5, no. 4, pp. 1125, 2001.

[56] Kewei Sha, Junzhao Du, and Weisong Shi. WEAR: A Balanced, Fault-Tolerant, Energy-Aware Routing Protocol for Wireless Sensor Networks. Wayne State University. International journal of Sensor networks 2006-Vol. 1, No.3/4 pp. 156- 168.

[57] G. Xing, C. Lu, R. Pless, and Q. Huang. On greedy geographic routing algorithms in sensing-covered networks. In *the Fifth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, May 2004.

[58] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, UCLA Computer Science Department, May 2001.

[59] B. Karp and H. T. Kung. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual ACM/IEEE International Conference* on Mobile Computing and Networking(MobiCom’00), Aug. 2000

[60] Q. Fang, J. Gao, and L. J. Guibas. Locating and bypassing routing holes in sensor networks. In *Proc. of IEEE Conference on Computer Communications (INFOCOM’04)*, Mar. 2004.

[61] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5), Oct. 2000

[62] N. Bulusu, D. Estrin, L. Girod, and J. Heidemann. Scalable coordination for wireless sensor networks: Selfconfiguring localization systems. In *Proceedings of Internationsl Symposium on Communication Theory and Applications*, July 2001.

[63] D. Moore, J. Leonard, D. Rus, and S. Teller. Robust distributed network localization with noisy range measurements. In *Proc. of ACM SenSys 2004*, Nov. 2004.

[64] D. Slepian and J. Wolf, *Noiseless coding of correlated information sources*, IEEE Transaction Information Theory, no. IT-19, pp.471-480, 1973.

[65] R. Cristescu, B. Beferull-Lozano, and M. Vetterli, *On Network Correlated Data Gathering*, IEEE INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies Volume 4, Issue , 7-11 March 2004 Page(s): 2571 - 2582 vol.4

[66] D. Marco and D. L. Neuhoff, *Reliability and Efficiency in Distributed Source Coding for Field Gathering Sensor Networks*, IPSN 2004, pp. 161-168.

[67] Sinem Coleri and Pravin Varaiya. Fault Tolerance and Energy Efficiency of Data Aggregation Schemes for Sensor Networks. July 2, 2004. California Department of Transportation.

[68] S. Nathy, P.B. Gibbons, S. Seshany, and Z.R. Anderson, “Synopsis Diffusion for Robust Aggregation in Sensor Networks,” *SenSys'04*, November 3-5, 2004, Baltimore, Maryland, USA.

[69] S. Gobriel, S. Khattab, D. Mosse, J. Brustoloni, R. Melhem, “Fault Tolerant Aggregation in Sensor Networks Using Corrective Actions,” *3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, vol. 2, pp. 595-604, 2006.

[70] Y. Chen, and S.H. Son, “A Fault Tolerant Topology Control in Wireless Sensor Networks,” *Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, 2005.

[71] M. Cardei, S. Yang, and J. Wu, “Algorithms for Fault-Tolerant Topology in Heterogeneous Wireless Sensor Networks,” IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 19, NO. 4, APRIL 2008.

[72] A. Frank and E. Tardos, “An Application of Submodular Flows,” Linear Algebra and Its Applications, vol. 114-115, pp. 329-348, 1989.

[73] S. Das, H. Liu, A. Kamath, A. Nayak, and I. Stojmenovic, “Localized Movement Control for Fault Tolerance of Mobile Robot Networks,” *The First IFIP International Conference on Wireless Sensor and Actor Networks* (WSAN 2007), Albacete, Spain, 24-26 Sept. 2007.

[74] T. Clouqueur, K.K. Saluja, and P. Ramanathan, "Fault tolerance in collaborative sensor networks for target detection," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 320-333, 2004.

[75] M. Ding, F. Liu, A. Thaeler, D. Chen, and X. Cheng, "Fault-Tolerant Target Localization in Sensor Networks," *EURASIP Journal on Wireless Communications and Networking*, 2007.

[76] B. Krishnamachari, and S. Iyengar, "Distributed Bayesian Algorithms for Fault-Tolerant Event Region Detection in Wireless Sensor Networks," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 241-250, 2004.

[77] X. Luo, M. Dong, and Y. Huang, "On Distributed Fault-Tolerant Detection in Wireless Sensor Networks," *IEEE Transactions on Computers*, vol. 55, no.1, pp. 58-70 2006.

[78] F. Araújo, and L. Rodrigues. "On the Monitoring Period for Fault-Tolerant Sensor Networks," *LADC 2005*, LNCS 3747, São Salvador da Bahia, Brazil, October 2005.

[79] A. Gallais, J. Carle, D. Simplot-Ryl, and I. Stojmenovic, "Localized Sensor Area Coverage with Low Communication Overhead," In *Proc. of IEEE Sensor'06*, Daegu, Korea, Oct. 2006.

[80] H. Liu, P.J. Wan, and X. Jia, "Maximal Lifetime Scheduling for Sensor Surveillance Systems with *K* Sensors to 1 Target," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 12, Dec. 2006.

[81] F.C. Gärtner. Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments. ACM ComputingSurveys, 31(1):1–26,March 1999.

[82] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374–382, April1985.

[83] Tushar Deepak Chandra and Sam Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. Journalofthe ACM, 43(2),March 1996.

[84] Laura S. Sabel , Keith Marzullo, Election Vs. Consensus in Asynchronous Systems, tech_report TR95-1488, Cornell University, Ithaca, NY, 1995 .

[85] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving Consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[86] MK Aguilera, C Delporte-Gallet, H Fauconnier, and S Toueg. Stable leader election. In DISC01 Distributed Computing 15th International Symposium, Springer LNCS:2180 , 2001.

[87] S. Dulman, P. Havinga and J. Jurink, Wave leader election protocol for wireless sensor networks, MMSA Workshop, Delft, The Netherlands, December 2002.

[88] N. Lynch. Distributedalgorithms.MorganKaufmannPublishers,1997.

[89] M. Larrea, C. Martin, J. Javier Astrain. Coordinated Data Aggregation in Wireless Sensor Networks using the Omega Failure Detector.*PE-WASUN'06,* ACM October 6, 2006, Torremolinos, Malaga, Spain.

[90] M. Larrea, C. Martin, J. Javier Astrain. Hierarchical and fault-tolerant Data Aggregation in Wireless Sensor Networks. IEEE 2007.

[91] B. Krishnamachari, D. Estrin, and S. Wicker. Modelling data centric routing in wireless sensor networks. Technical Report CENG 02-14, USC Computer Engineering, 2002.

Abstract

This document presents the different fault tolerance techniques and algorithms for leader election in Wireless Sensor Networks (WSN). We propose two algorithms, the first one is an implementation of omega (Ω) failures detector with crash failures in wireless sensor networks. This algorithm ensures an agreement on a common leader by all sensors in one region, as well as on a common super-leader among the set of leaders of the network. The second is an algorithm for data aggregation by using the first algorithm to provide a hierarchical energy-efficient data aggregation mechanism.

Keywords :
Wireless sensor network, fault tolerance, data aggregation, failure detector

Résumé

Ce document présente les différentes techniques de tolérance aux fautes et les algorithmes d'élection de leader dans les réseaux de capteurs sans fil. Nous proposons deux algorithmes. Le premier pour implémenter un détecteur de défaillance (Ω) dans un réseau de capteurs sans fil dont le modèle de défaillance est par crash, et le deuxième algorithme présente un algorithme d'agrégation de données en se basant sur le premier algorithme afin de fournir un mécanisme hiérarchique d'agrégation de données efficace en énergie.

Mots clé :

Réseau de capteurs sans fil, tolérance aux fautes, agrégation de données, détecteur de défaillances.