République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed El Bachir El Ibrahimi, Bordj bou Arréridj

Faculté des Mathématiques et d'Informatique

Mémoire

en vue de l'obtention du diplôme de

Master en Informatique

Option : Réseaux et Multimédia

Titre

Routage à délai borné et à basse consommation énergétique, basé-ACO, pour les réseaux de capteurs et d'actionneurs sans fil

Présenté Par

Oussama MANSOURI & Elhadi CHIBANE

Soutenu le ...

Devant le jury composé de :

Président	Boubakeur MOUSSAOUI	MCB, Université de BBA
Rapporteur	Nadjib BENAOUDA	MCB, Université de BBA
Examinateur	Farid NOUIOUA	MCA, Université de BBA

Promotion 2020-2021

Remerciements

En préambule à ce mémoire, nous remercions ALLAH qui nous a aidé et nous a donné la patience et le courage durant la réalisation de ce présent travail et aussi durant ces longues années d'étude.

Nous tenons également, à remercier notre encadrant Dr.Nadjib BENAOUDA pour avoir dirigé ce travail avec une grande rigueur scientifique, et pour sa disponibilité, ses précieux conseils, la confiance qu'il nous a accordé et aussi pour son suivi régulier pendant l'élaboration de ce mémoire.

Nous tenons à remercier aussi les membres de jury pour avoir accepté de lire ce mémoire et évaluer notre travail.

Nos remerciements s'étendent également au Chef de département ainsi qu'à tous nos professeurs de la Faculté des mathématique et d'informatique pour la richesse et la qualité de leurs enseignements et les grands efforts déployés pour assurer à leurs étudiants une formation actualisée.

Nous n'oublions pas nos parents pour leur contribution, leur soutien et leur patience.

Enfin, nous adressons nos plus sincères remerciements à tous nos proches et amis, qui nous ont toujours soutenu et encouragé au cours de la réalisation de ce mémoire.

Nous exprimons notre gratitude aux personnes qui nous ont apporté leur aide, qui ont contribué à l'élaboration de ce mémoire et qui ont accepté de répondre à nos questions avec gentillesse.

Merci à tous et à toutes.

Dédicaces

Je dédie mon travail

À mes chers parents, Mouhamed Aalah et Aldja, qui ont toujours été à côté de moi, qu'Allah, le miséricorde, accueil mon père dans son immense paradis « mes parents qui ont sacrifié les plus belles années de leur vie pour me voir un jour réussir et pour leur soutien moral ».

À mes sœurs Fazia, Linda, Khadédja, Yakout, Ghania, Khalissa, Asma et Dihia, que dieu les garde et les protège.

À mon frère : Sofiane je demande à dieu de le garder.

À mes amis : Mohammed, Ameur, Zohir, Rachid, Abd elbaki, Oussama, Badreddine, Saber.

À mon encadrant qui m'a soutenu tout au long de ce projet

Dr. Nadjib BENAOUDA.

À tous mes Ami(e)s et mon binôme Oussama Mansouri que j'aime tant, Pour leur sincère amitié et confiance, et à qui je dois ma reconnaissance et mon attachement.

Mes dédicaces s'adressent aussi à tous les membres des deux familles, Chibane et Bendifallah.

Chibane Elhadi

Je dédie ce modeste travail

À mes chers parents mais aucune dédicace ne serait témoin de mon profond amour, mon immense gratitude et mon plus grand respect, car je ne pourrais jamais oublier la tendresse et l'amour dévoué par eux, et leur soutien, leur patience, leur encouragement durant mon parcours scolaire.

Je vous aime et qu'Allah vous garde pour nous.

À ma grande mère à qui je souhaite un bon rétablissement.

À mes très chères sœurs et mon cher frère que j'admire beaucoup.

À mes oncles et mes tantes. À mon binôme et sa famille.

À tous mes amis et à tous ceux que j'aime qui m'ont toujours encouragé et à toute ma promotion à qui je souhaite plus de succès.

Mansouri Oussama

Résumé

Dans ce présent travail, nous nous sommes intéressés aux réseaux de capteurs et d'actionneurs sans fil (WSANs). Particulièrement, nous nous sommes intéressés au routage des données prélevées par les nœuds capteurs et leur délivrance au noeud actionneur approprié. Ce routage doit être économe en énergie à cause des ressources énergétiques limitées des nœuds capteurs. Il doit également assurer une latence qui ne dépasse pas une certaine échéance donnée car, les nœuds actionneurs doivent agir rapidement. Étant donnée que la conservation de cette énergie se fait au détriment d'un temps de latence plus important, un tel routage doit assurer un compromis entre la conservation de l'énergie des nœuds et le respect de l'échéance imposée. Pour répondre à ce problème, nous nous sommes basés sur les algorithmes de colonies de fourmis et proposés notre protocole que nous avons baptisé CADER comme acronyme de *« Centralized Ant-based Delay-bounded and Energy-efficient Routing in wireless sensor and actor networks »*. Notre solution a été validée par simulation à l'aide du simulateur J-sim dans lequel nous avons complètement intégré notre protocole. Les résultats qui sont présentés sous forme de courbes et d'instantanés montrent le bon fonctionnement de notre protocole et sa capacité à assurer le compromis visé.

Abstract

In this present work, we are interested in wireless sensor and actor networks (WSANs). In particular, we are interested in the routing of the data sensed by sensor nodes and their transmission to the appropriate actor node. An energy-efficent routing have to be ensured because of the limited energy of sensor nodes. In addition, a real-time communication is needed in order to allow the quik reaction of actor nodes to the sensed event. Given that the conservation of this energy is done at the expense of a greater latency time, a tradeoff between the minimization of energy consumption and deadline respect has to be ensured. To answer this question, we have based on ant colony optimization algorithms and proposed our protocol that we have called CADER as an acronym for « *Centralized Ant-based Delay-bounded and Energy-efficient Routing in wireless sensor and actor networks* ». Our solution has been validated by simulation using J-sim simulator in which we have completely integrated our protocol. The results which are presented in the form of curves and snapshots show the proper operation of our protocol and its capacity to ensure the desired tradeoff.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Introduction générale

Les réseaux de capteurs et d'actionneurs sans fil (WSANs) envisagent un large éventail d'applications dans différents domaines comme celui de la domotique, de l'agriculture, des réseaux urbains, de l'industrie, etc. Ils mettent en coopération deux types de nœuds. Les premiers sont des nœuds capteurs qui sont déployés avec un grand nombre et sont responsables du prélèvement et de la communication des mesures pertinentes à leur environnement immédiat. Les deuxièmes, déployés avec un nombre moins important, récupèrent ces informations afin de prendre les actions appropriées. Les nœuds capteurs sont caractérisés par des ressources de traitement et de stockage restreintes et ils sont généralement alimentés par des batteries qui contraignent leur durée de vie. En contre partie, les nœuds actionneurs sont plus puissants et ont des ressources plus importantes.

Dans ce travail, nous nous intéressons au problème de routage des données prélevées par les nœuds capteurs aux nœuds actionneurs appropriés. Avec ce routage, deux objectifs doivent, principalement, être considérés. Le premier est pertinent à l'énergie restreinte des nœuds capteurs. Dans ce cas, il faut considérer un routage à basse consommation énergétique pouvant augmenter la durée de vie du réseau. Le deuxième considère le besoin à délivrer en temps opportun, et donc, en temps réel, les données prélevées aux nœuds actionneurs afin que ces derniers puissent prendre les actions les plus appropriées au bon moment. En fait, la considération de ces communications en temps réel revient à assurer un compromis entre cette délivrance à délais bornés des données et la minimisation de l'énergie qui est consommée durant cette délivrance. Pouvoir économiser de l'énergie lors du routage requiert l'utilisation de routes avec des nœuds qui utilisent des puissances de transmission réduites et donc, des nœuds qui émettent sur de courtes distances (i.e. vers leurs voisins qui sont proches d'eux). Cela génère à son tour des routes qui ont un grand nombre de sauts, et donc, la latence avec ces routes sera très importante.

D'un autre côté, les algorithmes de colonies de fourmis (ACO)[2] ont été pleinement appliqués à ce problème de routage. Ces algorithmes ont été inspirés du comportement collectif des fourmis recherchant un chemin entre leur colonie et une source de nourriture. En fait, les fourmis, via leur travail collectif, peuvent résoudre des problèmes de routage. Afin de faire survivre leur colonie, les fourmis doivent explorer leur environnement à la recherche de source de nourriture. À la découverte de cette source, elles doivent établir certains chemins qui peuvent être exploités pour que ces fourmis puissent se déplacer entre leur nid et cette source. Ces chemins émergent suite à l'évaluation implicite de multiples trajets et la communication de leurs caractéristiques aux autres fourmis via le dépôt de la phéromone. En résumant, nous pouvons dire que ce processus de collection de la nourriture requière l'exploitation distribuée, la découverte, la détermination et l'utilisation de chemins de routage dans un environnement dynamique. Ces caractéristiques sont en fait, les mêmes fonctions nécessaires au routage de données dans un réseau de communication.

Lorsque ces algorithmes de colonies de fourmis sont appliqués à ce problème de routage, les fourmis sont le plus souvent modélisées sous forme de paquets de contrôle. Ces derniers sont générés de manière répétitive et sont acheminés d'un nœud à un autre dans le but

de trouver de bons chemins. En fait, un bon chemin n'émerge qu'après un grand nombre de phases d'exploration et de mise à jour de tables de routage (e.g. table de phéromone, etc) qui nécessite la génération et la communication d'un grand nombre de paquets de contrôle. Étant donné que les communications représentent la source qui consomme la grande partie de l'énergie des nœuds capteurs, cela peut ne pas justifier l'application répartie de ces algorithmes dans le cas des WSANs car, l'utilisation excessive de ces paquets peut l'emporter sur le gain qu'on peut avoir après l'établissement de bons chemins.

Vue cette contrainte et en exploitant la présence de nœuds actionneurs qui ont des ressources plus importantes, on procède dans ce travail à une application centralisée de ces algorithmes de colonies de fourmis. Avant cette application, on procède à une clusterisation du réseau et aussi à la collection des informations des membres et leurs délivrance à leurs cluster-heads. Chaque cluster est représenté par un nœud actionneur qui joue le rôle du cluster-head avec les nœuds capteurs les plus proches de lui en termes de nombre de sauts comme membres. Lorsqu'un évènement arrive, le nœud capteur le détectant informe son nœud actionneur. Ce dernier lui crée un chemin an appliquant localement l'algorithme de colonie de fourmis sur la base des informations de ses membres et lui renvoie la route formée. Notre protocole proposé est baptisé CADER comme acronyme de : *Centralized Ant-based Delay-bounded and Energy-efficient routing in WSAN*.

Afin de valider notre travail, nous procédons par simulation à l'aide du simulateur J-Sim.[3] Nous menons diverses expérimentations selon différents points de vue. Les résultats obtenus sont présentés sous forme de courbes et d'instantanés et montrent des résultats positifs. En particulier, ils illustrent la capacité de notre protocole à assurer le compromis cité auparavant.

Afin de décrire notre façon de réfléchir, ce mémoire, présente les principaux éléments académiques liés à notre travail ainsi que notre contribution et sa validation. Il est organisé de la façon suivante :

- Le chapitre 1 présente des généralités sur les WSANs. En particulier, toutes les notions pertinentes et nécessaires à la bonne compréhension de notre travail sont présentées.
- Le chapitre 2 présente le principe des algorithmes de colonies de fourmis, la solution que nous avons appliqué pour trouver les routes désirées, et leur application aux WSNs et aux WSANs. Certains travaux de la littérature qui sont connexes à notre travail sont présentés.
- Le chapitre 3 présente une description détaillée de notre protocole proposée.
- Le chapitre 4 présente la méthodologie de validation utilisée et les résultats obtenus.
- Ce mémoire se termine par une conclusion générale récapitulant nos résultats.

Chapitre 1

Généralités sur les Réseaux de capteurs et d'actionneurs sans fil

1.1 Introduction

Dans ce chapitre, nous allons présenter les principales notions liées aux réseaux de capteurs et d'actionneurs sans fil (WSANs). Nous commencerons par définir ce que c'est un réseau de capteurs sans fil (WSN) et un réseau de capteurs et d'actionneurs sans fil. Une anatomie générale d'un noeud capteur est également présentée. Après cette première partie, nous présenterons les différents modes de communication qu'on trouve généralement dans ces réseaux, les objectifs de conception et les contraintes à considérer. Nous présenterons également les différentes formes de dissipation de l'énergie des nœuds capteurs et les principales techniques de conservation de cette énergie. Une brève description des communications temps-réel et du routage dans ces réseaux sera ensuite présentée. Nous terminerons par la présentation de quelques scénarios d'utilisation de ces réseaux.

1.2 Réseaux de capteurs sans fil

Depuis leur émergence, les réseaux de capteurs sans fil n'ont jamais cessé d'être l'un des domaines les plus actifs dans la recherche scientifique à cause de leur potentialité et du nombre important d'applications qu'ils envisagent. En fait, un réseau de capteurs sans fil (WSN) est constitué d'une collection de nœuds qui communiquent via des liens radio, et capables de récolter et de transmettre des données entre eux. Ces nœuds sont déployés sur une zone géographique, appelée champ de captage. Ils prélèvent des grandeurs physiques qui sont pertinentes à l'environnement extérieur (e.g. chaleur, humidité, vibrations, etc.) et les transforment en grandeurs numériques [4]. Ces grandeurs sont ensuite acheminées d'un nœud à un autre jusqu'à un nœud particulier appelé nœud puits (ou Sink). Ce dernier permet de collecter et de stoker les informations reçues des différents capteurs [4]. L'utilisateur peut consulter ces données à travers ce nœud puits via le réseau Internet par exemple. La figure (1.1) illustre l'architecture d'un réseau de capteurs sans fil.

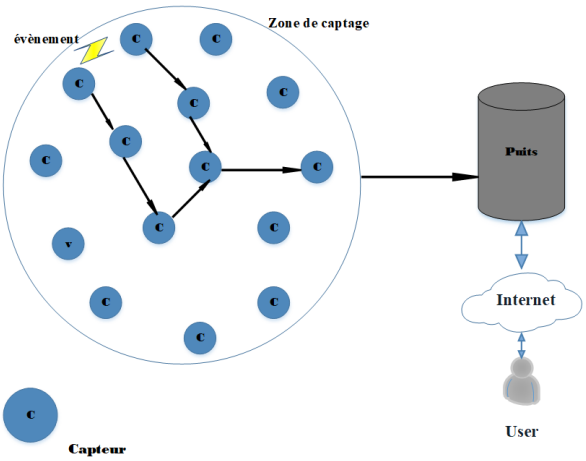


FIGURE 1.1: Architecture d'un réseau de capteurs sans fil.

1.3 Composants d'un nœud de capteur

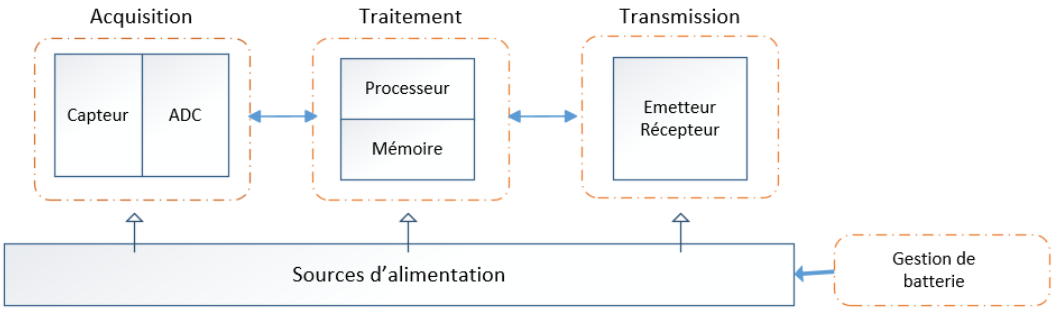


FIGURE 1.2: Architecture d'un nœud capteur.

Un nœud capteur est doté d'une multitude de composants lui permettant d'assurer sa tâche. Principalement, il est doté de capteurs qui lui permettent de prélever les mesures physiques, d'une unité de traitement embarquée à faible coût qui lui permet de faire des calcul, d'une petite mémoire et d'un émetteur-récepteur radio. La capacité d'un capteur est néanmoins limitée à cause, en partie de la miniaturisation de ses composants [5]. En addition, un nœud capteur est alimenté par une batterie qui assure son fonctionnement. Il peut contenir également des modules supplémentaires tels qu'un système de localisation (GPS). La figure (1.2), illustre l'architecture d'un nœud capteur. Comme le montre cette figure, quatre unités peuvent être distinguées dans l'architecture d'un nœud capteur :

- **Unité d'acquisition :** elle est généralement composée de deux sous-unités. Un capteur et un convertisseur Analogique/Numérique. Le capteur est responsable de la fourniture des signaux analogiques en se basant sur le phénomène observé et le convertisseur Analogique/Numérique transforme les grandeurs physiques captées en signaux numériques pour les introduire ensuite dans l'unité de traitement.

- **Unité de traitement :** elle est chargée de faire les traitements appropriés. En particulier, elle permet d'exécuter les protocoles de communications qui permettent de faire collaborer le nœud avec les autres nœuds du réseau et aussi d'analyser et de traiter les données captées. Elle est généralement associée à une petite unité de stockage [4].
- **Unité de transmission radio :** cette unité est responsable d'effectuer toutes les émissions et réceptions des données sur un medium sans fil [4].
- **Unité d'alimentation :** Un capteur est muni d'une ressource énergétique (généralement une batterie) pour alimenter tous ses composants. Cette ressource énergétique dont il dispose est limitée et ne peut généralement pas être compensée. Par conséquent, l'énergie est la ressource la plus précieuse d'un réseau de capteurs, car elle affecte directement la durée de vie. L'unité d'alimentation est donc responsable de répartir l'énergie disponible sur les autres modules et réduire les dépenses en mettant en sommeil les composants actifs par exemple.

1.4 Réseaux de capteurs et d'actionneurs sans fil

Avec les développements récents dans la micro-électronique et les télécommunications, les nœuds capteurs ont connu des améliorations en ce qui concerne leurs différentes capacités. Comme nous avons vu dans la section précédente, un réseau de capteurs sans fil consiste en un grand nombre de nœuds capteurs distribués qui s'auto-organisent en un réseau sans fil multi-sauts.

La technologie moderne a évoluée, menant à l'émergence de d'autres réseaux ad hoc qui considèrent, non pas uniquement des nœuds capteurs, mais aussi d'autres nœuds plus puissants capables d'effectuer des actions sur la base de ce que les nœuds capteurs perçoivent. Ces réseaux sont appelés réseaux de capteurs et d'actionneurs sans fil (WSANs) [6].

1.4.1 Rôles des nœuds dans un WSAN

Généralement, un WSAN se compose d'un nombre important de nœuds capteurs et de quelques actionneurs comment le montre la figure (1.3). Outre les tâches de traitement et de communication qui sont effectuées par tous les nœuds du réseau, les rôles suivants sont à distinguer dans un WSAN [7] :

- **Perception :** comme nous avons déjà vu, chaque noeud capteur est doté d'un certain nombre de capteurs lui permettant de capter des mesures sur l'environnement.
- **Actionnement :** la particularité des WSANs est leur capacité à faire des actions sur l'environnement. Cela est assuré par les nœuds actionneurs responsables de cette tâche. Ces derniers nœuds coopèrent généralement pour couvrir toute la zone d'évènement. Pour cela, ils communiquent ensemble soit via les nœuds capteurs ou directement entre eux à travers leur réseau point-à-point dédié.
- **Puits de données :** les nœuds puits jouent un rôle très important, principalement, ils constituent l'intermédiaire entre le réseau et le monde extérieur. En fait, c'est à travers ces nœuds que l'utilisateur peut contrôler le réseau et récupérer toute information pertinente soit à l'environnement soit au réseau.

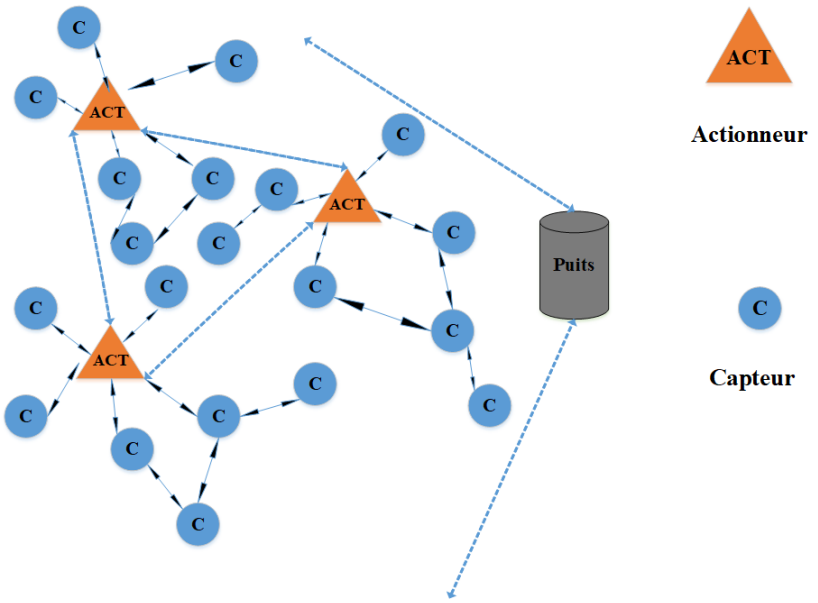


FIGURE 1.3: Architecture d'un réseau de capteurs et d'actionneurs sans fil (WSAN).

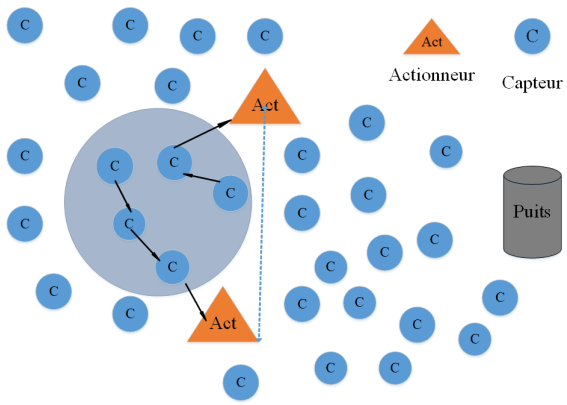
1.4.2 Architectures de communication dans les réseaux WSANs

Dans un WSAN et comme nous venons de voir, le rôle des nœuds capteurs consiste à prélever des données pertinentes à notre environnement et celui des nœuds actionneurs consiste à réaliser les actions les plus appropriées sur la base de ces données collectées. Ainsi et comme le montre la figure (1.3), ces différents nœuds, capteurs et actionneurs, sont répartis sur le champ avec le nœud puits qui représente principalement la passerelle du réseau vers le monde extérieur [8]. Dans les WSANs, les trois types d'architecture de communication suivantes peuvent être distinguées [8] :

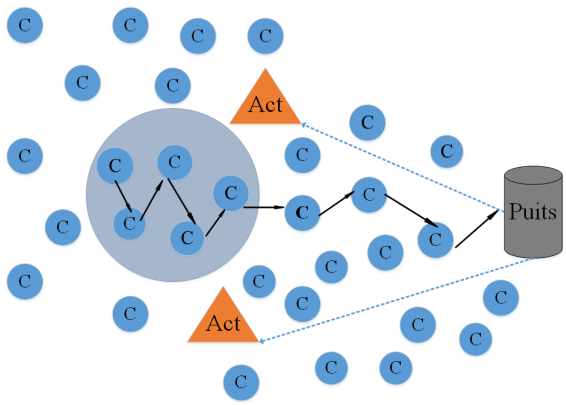
- **Architecture automatique :** avec cette architecture, les nœuds capteurs envoient directement leurs données prélevées vers les nœuds actionneurs. Ces derniers et sur la base de ces données coopèrent afin de réaliser les actions appropriées (voir figure (1.4a)).
- **Architecture semi-automatique :** avec cette architecture, les nœuds capteurs envoient leurs données vers la station de base ou nœud puits. cette dernière choisit ensuite l'actionneur le plus approprié (e.g. le plus proche) pour lui envoyer ces données (voir figure (1.4b)).
- **Architecture coopérative :** avec cette architecture, les nœuds capteurs transmettent leurs données vers les nœuds actionneurs. Ces derniers analysent les données et peuvent consulter le nœud puits avant de prendre toute action. Ils peuvent utiliser également leur réseau point-à-point pour prendre des décisions sur ce qu'ils doivent prendre comme actions ou tout simplement informer le nœud Puits et attendre ce qu'il leur ordonne de faire (voir figure (1.4c)).

1.4.3 Coordination dans les WSANs

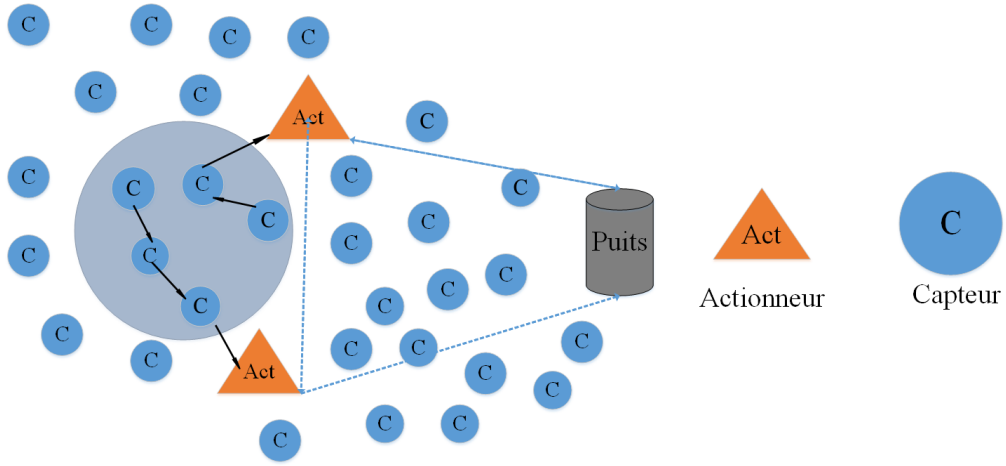
La coordination est un problème fondamental dans les WSANs. La coordination utilisée, cependant, dépend de l'architecture du réseau. En particulier, avec une architecture semi-automatique, cette coordination est assurée par le nœud puits alors qu'avec une architecture automatique, nous distinguons entre deux types de coordination : capteur-actionneur et actionneur-actionneur. Avec le premier mode, les nœuds capteurs doivent



(a) Architecture automatique.



(b) Architecture semi-automatique.



(c) Architecture coopérative.

FIGURE 1.4: Les architectures de communication dans les WSANs.

déterminer le ou les nœuds actionneurs appropriés auxquels ils doivent envoyer leurs données et les routes de routage entre ces nœud capteurs et ces nœuds actionneurs doivent être formées. Dans le deuxième, les nœuds actionneurs interagissent ensemble pour prendre les bonnes décisions d'action.

Coordination capteur-actionneur

Avec une telle coordination, la question suivante surgisse : *comment les nœuds capteurs puissent déterminer auxquels nœuds actionneurs ils doivent émettre leurs données ?,* cela est très important car, les données prélevées doivent arriver correctement et en temps opportun aux nœuds actionneurs afin que ces derniers puissent avoir suffisamment du temps pour décider des actions à réaliser. Lorsqu'un événement se produit, les données peuvent être transmises à un seul nœud actionneur comme l'illustre la figure (1.5a), comme elles peuvent être transmises à plusieurs nœuds actionneurs comme l'illustre la figure (1.5b) [9][10].

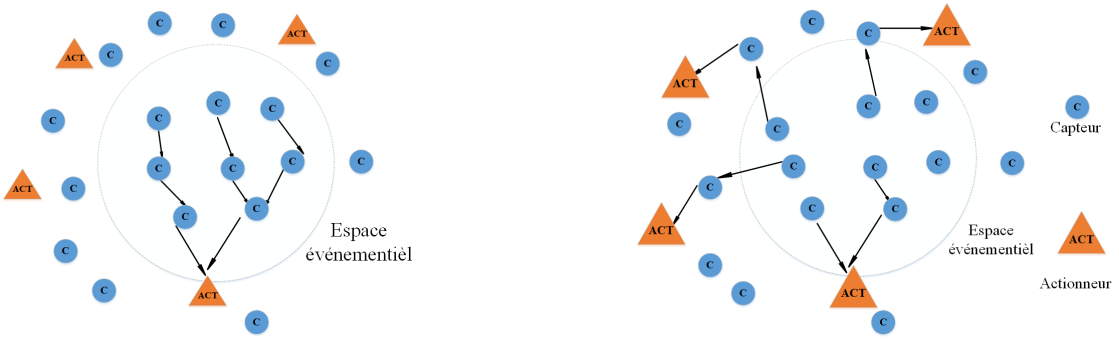
Dans le cas d'un seul nœud actionneur, les nœuds capteurs dans la zone d'événement communiquent ensemble pour trouver le nœud actionneur le plus approprié, c.à.d. le plus proche et qui a suffisamment d'énergie pour réaliser les actions nécessaires pouvant couvrir cette zone. L'avantage ici est que les nœuds actionneurs sont exclus de cette

coordination qui est assurée complètement par les nœuds capteurs. Cependant, les nœuds capteurs autour du nœud actionneur sélectionné vont connaître une certaine charge de communication qui peut épuiser leur énergie.

Dans le deuxième cas, chaque capteur sélectionne indépendamment le nœud actionneur auquel il doit émettre ses données. Dans ce cas, aucun nœud actionneur ne sera surchargé et donc, il n'y aura pas de nœuds capteurs autour d'un nœud actionneur donné qui vont connaître une charge excessive de communication.

Afin que les nœuds capteurs puissent décider de leurs actionneurs correspondants, il faut un moyen approprié pour les notifier de la localisation de ces nœuds actionneurs surtout si ces derniers sont mobiles.

Après la sélection de ou des nœuds actionneurs appropriés, il est nécessaire d'établir les bonnes routes permettant aux nœuds capteurs de faire parvenir leurs données à ces nœuds actionneurs. Le processus d'établissement de ces routes doit tenir en compte des capacités limitées, en particulier énergétique, des nœuds capteurs et aussi du besoin à faire parvenir les données prélevées à temps.



(a) Données envoyées vers un seul actionneur.

(b) Données envoyées vers de multiple actionneurs.

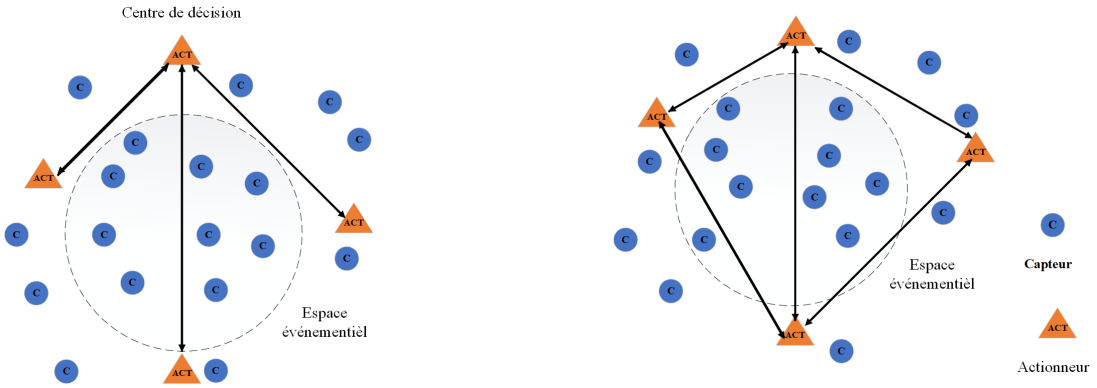
FIGURE 1.5: Approches d'une coordination Capteur-Actionneur.

Coordination actionneur-actionneur

Avec une coordination actionneur-actionneur, les nœuds actionneurs doivent se coordonner entre eux pour qu'au moins, un d'entre eux peut répondre à l'événement survenu. Les actions à prendre peuvent être décidées, comme l'illustre la figure (1.6), selon deux approches : approche centralisée et approche distribuée [9][10].

Dans le premier cas, avec une approche centralisée, et à chaque fois qu'un nœud actionneur reçoive un événement d'un nœud capteur donné, il envoie, comme l'illustre la figure (1.6a), l'information à un nœud actionneur prédéterminé qui est responsable de la prise des décisions (centre de décision). Ce nœud de décision choisit le meilleur groupe de nœuds actionneurs pour effectuer le tâche requise. Dans le deuxième cas, les décisions d'actionnement sont prises de manière répartie.

L'avantage de l'approche centralisée est la capacité de sélection des meilleurs nœuds actionneurs pour accomplir la tâche nécessaire. Cependant, cela s'accompagne avec une augmentation dans le temps de réponse comme les nœuds actionneurs doivent envoyer leurs données à une nœud central. Cela n'est le cas de l'approche répartie car, chaque nœud actionneur est capable de prendre une décision locale rapide basée sur les données reçues.



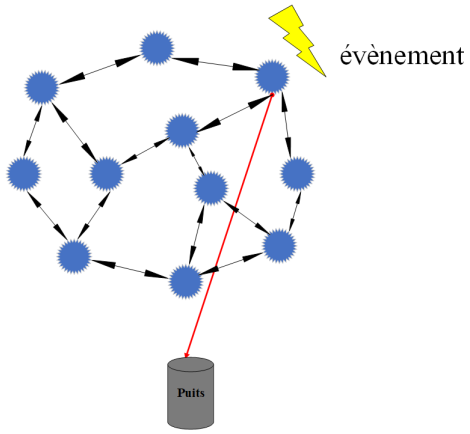
(a) Approche centralisée. (b) Approche distribuée.

FIGURE 1.6: Approches d'une coordination Actionneur-Actionneur.

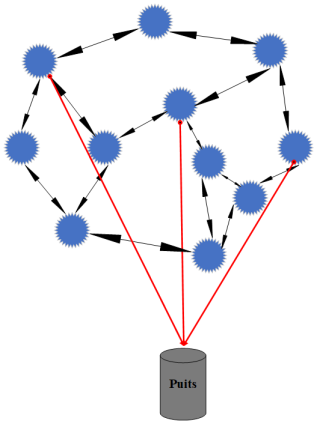
1.5 Modes de communication

Dans les WSNs et les WSANs, la communication peut se faire, comme l'illustre la figure (1.7), selon les trois principaux modes suivants [11] :

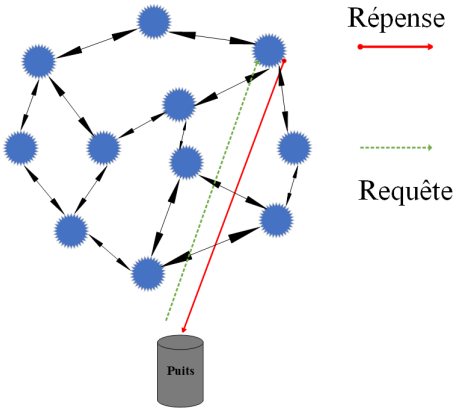
- **Communication événementielle :** avec ce premier mode et comme l'illustre la figure (1.7a), les nœuds capteurs notifient leurs nœuds puits/actionneurs correspondants dès qu'un évènement arrive. Ce mode de communication est propre aux applications qui connaissent une occurrence rare d'évènements. Nous pouvons citer à titre d'exemple, une application de détection de feu dans des forêts, des applications de bâtiments et de structures intelligents, etc. En fait, les réseaux déployés pour ce dernier type d'applications peuvent avoir plusieurs objectifs. Ils peuvent être dédiés à l'étude de structures pour détecter des fissures (qui peuvent se produire lors de séismes) ou détecter l'impact de chantiers sur d'anciennes constructions. Ils peuvent également être dédiés à l'étude des conditions ambiantes d'une maison, d'une industrie ou de navires.
- **Communication périodique :** avec ce deuxième mode de communication, les données sont envoyées, à intervalles réguliers, aux nœuds puits/actionneurs après leur prélèvement. Les données sont transmises à plusieurs reprises par les nœuds capteurs, par exemple, à chaque seconde ou à chaque minute. Ce type de trafic constitue le trafic le plus utilisé dans les WSNs. À la différence des applications évènementielles où les nœuds capteurs détectant des évènements sont les seuls qui sont censés envoyés des données, avec les applications périodiques et comme l'illustre la figure (1.7b), tous les nœuds participent généralement à la collection des données. Comme exemple, nous pouvons citer un exemple lié au monde agricole où les nœuds capteurs transmettent périodiquement des données pertinentes à l'état du sol pour des fins de d'optimisation des apports d'eaux et de nutriments.
- **Communication à la demande :** avec ce dernier mode et comme l'illustre la figure (1.7c), des demandes sont envoyées sous forme de requêtes par l'utilisateur ou par le système à certains nœuds capteurs particuliers pour leur demander par exemple, leurs mesures prélevées ou des informations sur leurs états. Ces nœuds capteurs répondent en leur envoyant les données ou les informations demandées. Par exemple, avec une application de suivi de patients, des relevés à distance et en temps réel peuvent être effectués (pression sanguine, oxygénation, glycémie, etc.) par le personnel médical et qui sont obtenus par les nœuds capteurs sans fil.



(a) Mode de communication événementiel.



(b) Mode de communication périodique.



(c) Mode de communication à la demande.

FIGURE 1.7: Les différents modes de communication dans les WSANs.

1.6 Objectifs et contraintes

Dans cette section, nous présentons les principaux objectifs de conception auxquels les WSANs doivent répondre et les contraintes qui doivent être considérées. En s'intéresse particulièrement à la communication capteur-actionneur, car c'est elle qui est pertinente à notre travail.

1.6.1 Les objectifs

Avec une communication capteur-actionnneur, deux principaux objectifs doivent être visés lors de la conception de toute solution réseau. En particulier :

- Il faut envisager des solutions économes en énergie. En fait et comme nous avons déjà mentionné, les nœuds capteurs sont alimentés avec des batteries de capacités limitées. Pour cela, tout concepteur réseau doit tenir en compte de cette contrainte énergétique et proposer des solutions qui minimisent, non pas uniquement la consommation de ces nœuds, mais aussi qui assurent un équilibrage de cette consommation sur l'ensemble des nœuds réseau afin de prolonger sa durée de vie.
- Il faut assurer également des communication en temps réel. Dans les WSANs, les nœuds actionneurs doivent réagir aux évènements qui arrivent rapidement et intervenir en temps opportun afin de ne pas laisser l'évènement s'aggraver. Par exemple, dans un forêt et lorsque les nœuds capteurs détectent un départ de feu, ils doivent

rapidement informer les nœuds actionneurs pour que ces derniers puissent intervenir et éteindre ce feu. Comme l'illustre cet exemple, les nœuds actionneurs réagissent sur la base de ce que les nœuds capteurs leur envoient. Pour cette raison, les données doivent arriver aux nœuds actionneurs à temps. En outres, ces données doivent arriver correctement sans aucune alternance. On parle dans ce cas de communications fiables qui doivent également être assurées.

1.6.2 Les contraintes

L'accomplissement des deux objectifs cités auparavant doit être achevé avec la considération de quelques contraintes. En particulier, nous pouvons citer les contraintes suivantes [7] :

- **Les ressources énergétiques limitées :** comme nous l'avons déjà cité, il faut utiliser l'énergie d'un nœud capteur judicieusement et éviter toutes les manipulations qui consomment inutilement cette ressource.
- **Les ressources de traitement limitées :** un nœud capteur est doté d'un microcontrôleur de capacité limitée. Par conséquent, il faut éviter les solutions qui ont une complexité algorithmique importante.
- **Les ressources de stockage limitées :** l'espace mémoire dont les nœuds capteurs disposent est limité.
- **Bande passante et débit :** l'émetteur-récepteur d'un nœud capteur est optimisé pour que sa consommation énergétique soit minimale. Pa conséquent, la bande passante qui est mise à la disposition des applications est restreinte.
- **La mobilité :** avec un WSAN, les nœuds actionneurs peuvent être mobiles. Cela augmente la complexité des solutions envisagées à cause du besoin à suivre ces nœuds et à mettre au courant continuellement les nœuds capteurs de leurs positions.

À la différence des WSNs, la présence de nœuds actionneurs dans le réseau doit être exploitée à cause de leurs ressources importantes par rapport à celles des nœuds capteurs. En fait, c'est en se basant sur cette présence que notre contribution a été proposée.

1.7 Formes de dissipation de l'énergie d'un nœud capteur

Habituellement, les nœuds capteurs sont alimentés par des batteries, et ces dernières ont une durée de vie spécifique. Par conséquent, l'énergie de ces nœuds doit être consommée avec prudence afin de prolonger la durée de vie du réseau et donc celle de l'application. Comme nous avons déjà vu, l'efficacité énergétique est un objectif de conception ultime dans les WSANs.

Pour pouvoir concevoir de telles solutions économes en énergie, il est indispensable de comprendre les différentes formes de dissipation de l'énergie des nœuds capteurs. Nous distinguons particulièrement entre les trois formes suivantes que nous présentons dans cette section [4] : l'énergie qui est consommée par le capteur proprement dit, celle qui est consommée par le microcontrôleur et dernièrement celle qui est consommée par la radio.

1.7.1 Le capteur

Le capteur est un dispositif de détection, qui convertit l'état d'une grandeur physique telle que la température ou la pression, etc., en une grandeur mesurable où le signal analogique continu produit par les capteurs, est numérisé puis transféré au microcontrôleur afin qu'il soit traiter. Tout ce processus consomme de l'énergie [4].

1.7.2 Le microcontrôleur MCU (Microcontroller Unit)

Un microcontrôleur transite entre différents modes de fonctionnement dans le but d'économiser de l'énergie. Trois modes sont distingués : actif, inactif et sommeil. L'énergie qui est consommée varie selon le mode. En addition, la transition entre ces modes consomme également de l'énergie [4]. Prenons à titre d'exemple, le microcontrôleur ATMega 128L, que nous trouverons dans les capteurs Micaz, qui consomme 16.5mW en mode actif et seulement 3µW en mode inactif [4].

1.7.3 La radio

La radio est le composant qui consomme la plus grande partie d'énergie. Comme le microcontrôlleur, la radio fonctionne selon différents modes. On trouve particulièrement les quatre modes suivants : transmission, réception, veille et sommeil. Parmi ces modes, ce sont les transmissions et les réceptions qui consomment le plus d'énergie. Même en mode veille, une consommation importante a été remarquée à cause de l'écoute à vide du canal. Dans ce cas, il est préférable de mettre la radio en sommeil lorsqu'il n'y a aucune transmission ou réception afin d'économiser de l'énergie. Il est à noter que le basculement entre ces différents modes consomme également de l'énergie [4]. En fait, il est largement admis que l'énergie nécessaire pour transmettre un seul bit peut être utilisée pour effectuer un nombre important d'opérations arithmétiques [12].

1.8 Techniques de conservation d'énergie

Nous avons vu dans la section précédente que l'énergie des nœuds capteurs est consommée principalement par les opérations suivantes : la détection, le traitement et la communication. Conserver de l'énergie revient à savoir mettre en œuvre ces opérations. Trois classes de techniques de conservation d'énergie peuvent principalement être distinguées [4] : Duty cycling, approches orientées données, et mobilité.

1.8.1 Duty cycling

Le principe de base de cette technique consiste à mettre en mode sommeil les nœuds qui ne connaissent aucune activité réseau (i.e. transmission ou réception) et de les réveiller uniquement lorsqu'il est nécessaire, et donc, lorsqu'il y a des données à envoyer ou à recevoir. Cette alternance entre mode sommeil et actif dépend de l'activité réseau et nécessite un algorithme d'ordonnancement sommeil/réveil sur lequel les différents nœuds se basent pour coordonner leurs instants de sommeil/réveil.

La réalisation de cette technique du duty cycling peut être faite en se basant sur les deux techniques complémentaires suivantes : le contrôle de topologie et la gestion d'énergie. Avec la première technique, la redondance importante des nœuds est exploitée en choisissant un sous-ensemble de nœuds qui restent actifs et assurent la connectivité

du réseau. Les autres peuvent être mis en mode sommeil. Ces mêmes nœuds sélectionnés peuvent basculer entre les deux modes sommeil et réveil suivant l'activité du réseau. Dans ce dernier cas, on parle de gestion d'énergie.

1.8.2 Approches orientées données

Avec ces approches, l'objectif consiste à réduire soit, les coûts qui sont dus aux communications en réduisant la quantité de données qui circulent dans le réseau, soit, ceux qui sont dus à la détection. Dans le premier cas, on parle de technique de réduction de données et dans le deuxième, on parle de techniques d'acquisition de données efficaces en énergie.

La première technique de réduction de données comprend principalement, les méthodes de prédiction et celles d'agrégation de données. Avec une méthode de prédiction, le nœud puits peut prédire une réponse à une requête qui lui arrive d'un utilisateur sans faire recours aux nœuds capteurs. Cela se fait en se basant sur certains modèles. En contre partie, avec une méthode d'agrégation de données, la redondance des nœuds est exploitée en envoyant uniquement un agrégat de ce que certains nœuds perçoivent au lieu de laisser chaque nœud envoyer ses données au nœud puits indépendamment.

Avec la deuxième technique d'acquisition de données efficace en énergie, le fonctionnement du module de détection est optimisé afin de minimiser l'énergie qui sera consommée lors de la détection [4].

1.8.3 Mobilité

L'idée avec cette technique consiste à déployer des nœuds de collection mobiles qui se déplacent entre les différents nœuds capteurs pour récupérer leurs mesures. Cela permet de réduire la consommation énergétique des nœuds capteurs car, ces derniers sont complètement ou partiellement déchargés de l'acheminement de données. Par conséquent, les coûts de communication seront grandement réduits par rapport aux nœuds puits statiques. Cette mobilité peut prendre principalement deux formes. Elle peut être déterministe en ayant des nœuds mobiles contrôlables déployés spécifiquement pour cette tâche de collection. Dans le deuxième cas, les nœuds de collection sont déployés sur d'autres entités mobiles dont leur mobilité ne peut être contrôlée. Par exemple, sur des soldats dans une application militaire ou bien sur des animaux dans le cas d'une application environnementale.

1.9 Les communications temps-réel dans les WSANs

Comme nous avons vu dans le section (1.6.1), la communication qui se fait entre les nœuds capteurs et les nœuds actionneurs doit se faire en temps réel. On distingue généralement entre deux types de systèmes temps-réel [13] : les systèmes temps-réel strict ou dur (*hard* real-time) et les systèmes temps-réel souple ou mou (*soft* real-time). Avec des systèmes temps-réel dur, la réception tardive d'une donnée peut avoir de graves conséquences sur la vie humaine ou sur l'environnement [13][14]. Donc, les messages envoyées ne doivent pas arriver après leurs échéances. Par exemple, le début d'un feu doit être notifier à temps sinon il va s'élargir et sa maîtrise deviendra difficile. De la même façon, une fuite d'un produit toxique dans une usine peut causer de graves dégâts si sa notification ne se faisait pas rapidement. Pour les applications temps-réel souple, un certain dépassement de la borne temporelle imposée est toléré sans de graves conséquences. Par exemple, avec

une application de diffusion de vidéo, la perte ou le retard de certains messages n'a pas de grave conséquence à part une petite dégradation dans la qualité de la vidéo.

En fait, beaucoup de facteurs affectent le temps de bout en bout d'un message donné dans un réseau sans fil comme celui des WSNs ou des WSANs. Nous citons principalement [13] :

- le temps d'émission,
- le temps de propagation,
- le temps d'accès au médium (accès et reprise après collision),
- la longueur des routes.
- les délais d'interaction entre les couches protocolaires.

Parmi cette liste, ce sont les deux facteurs suivants qui ont une grande conséquence sur ce temps de bout en bout : le temps d'accès au médium et la longueur des routes.

Le temps d'accès au médium ou au canal peut être très élevé lorsque plusieurs nœuds émettent simultanément aux mêmes nœuds car, la probabilité d'occurrence de collisions est importante. Cette occurrence de collisions se traduit par une retransmission de tout message concerné par ces collisions qui va retarder l'acheminement de ces messages [13].

Au niveau routage, les données sont acheminées via des routes constituées d'une suite de nœuds dont le premier nœud est la source du message et le dernier est le destinataire. La latence des messages dépend de la longueur des routes empruntées [13]. Une longue route a une latence importante car, il y aura plus de durées de transmission, de propagation, de réception et d'accès au canal au contraire d'une courte route. Cependant, il est important de noter qu'une route courte avec des lignes congestionnés va connaître une latence importante par rapport à une autre route plus longue qui ne connaît rien ou peu de congestions.

1.10 Routage dans les réseaux de capteurs et d'actionneurs sans fil

Le routage est un processus qui permet de calculer et de maintenir des chemins dans un réseau pour transmettre des données depuis un émetteur jusqu'à un ou plusieurs récepteurs. Ce routage est indispensable dans beaucoup de domaines comme celui des réseaux téléphoniques ou des réseaux informatiques comme celui des les WSNs et les WSANs.

1.10.1 Défis de routage dans les WSANs

La transmission des données entre les nœuds capteurs et leurs nœuds actionneurs correspondants se base sur des protocoles de routage qui établissent les routes nécessaires et optimales à cette transmission. Dans un WSANs et pour une communication capteur-actionneur, l'objectif principal d'un protocole de routage est d'établir des routes économes en énergie qui assurent une latence qui ne dépasse pas l'échéance imposée. En plus des contraintes citées dans la section (1.6.2), les contraintes suivantes sont à considérer lors de la conception et l'implémentation de tels protocoles de routage :

- **La tolérance aux pannes :** durant le fonctionnement du réseau, les nœuds peuvent tomber en panne soit à cause de la fin de leurs batteries ou à cause d'une panne physique. Cela entraîne des problèmes de communication et d'acheminement des données car, ces nœuds ne participeront plus dans à acheminement. Par conséquent,

un protocole de routage doit envisager d'autres routes qui permettent la continuité de cet acheminement et donc la continuité du fonctionnement de l'application [4].

- **La dynamicité du réseau :** la mobilité des nœuds actionneurs rend cette tâche de routage plus complexe. Pour cela, il faut des algorithmes de routage qui s'adaptent au changement continu dans la topologie réseau causé par cette mobilité [4].
- **Le mode de communication** : la conception d'un protocole de routage dans un WSN et dans un WSAN dépend du mode de communication utilisé. Un protocole de routage qui suppose un mode de communication évènementiel est complètement différent d'un autre qui suppose un mode de communication périodique.

1.10.2 Protocoles de routage dans les WSANs

Diverses classifications de protocoles de routage ont été proposées dans la littérature. Dans ce présent travail, nous reprenons très brièvement la classification proposée dans [13] qui classifie ces protocoles en quatre classes : protocoles hiérarchiques, protocoles à la demande, protocoles aléatoires et protocoles greedy.

- **Les protocoles hiérarchiques :** avec ces protocoles, les nœuds du réseau sont organisés selon une certaine hiérarchie. On distingue principalement entre une organisation en clusters et une organisation en arbre. Dans le premier cas, les nœuds du réseau sont regroupés en clusters. Pour chaque cluster, un chef est élu appelé cluster-head qui assure la collection des données à partir de ses membres et leur transmission au nœud puits/actionneur. Avec une structure en arbre, un arbre de routage ayant comme nœud racine le nœud puits/actionneur est construit. Cet arbre connecte les autres nœuds du réseau avec ce nœud racine et chaque nœud choisit un nœud parent, selon une certaine métrique, auquel il envoie ses données [13][9].
- **Routage à la demande :** avec un tel routage, la création du chemin de routage entre une source et une destination est faite uniquement sur demande et lorsqu'il est nécessaire. Par exemple, lorsqu'un évènement arrive [13]. En fait, c'est à cette classe que notre protocole proposé dans ce présent travail appartient.
- **Routage aléatoire :** avec un routage aléatoire, chaque nœud sélectionne, comme prochain saut, un nœud aléatoire parmi la liste des voisins disponibles. Ce processus est répété par chaque nœud sélectionné jusqu'à ce que le message arrive au nœud puits/actionneur ou lorsque la valeur du champ TTL [1] du paquet qui est décrémentée à chaque saut, atteigne la valeur 0. L'avantage de ce routage est sa résilience vis-à-vis de la dynamicité du réseau. Cependant, la latence des messages peut ne pas être contrôlée car, la longueur des routes n'est pas du tout maîtrisée [13].
- **Routage greedy :** ce type de routage se base sur des décisions locales qui sont prises localement au niveau de chaque nœud. Le principe consiste à choisir à chaque fois et comme prochain saut, le meilleur voisin qui mène vers le nœud puits/actionneur. Par exemple, en choisissant le nœud voisin qui assure un avancement vers ce nœud puits/actionneur, c.à.d. le nœud voisin qui est plus proche de ce dernier nœud par rapport au nœud courant. Dans ce dernier cas, on parle de routage géographique [13].

1. Time To Live.

1.11 Quelques scénarios d'application

L'émergence des WSNs et des WSANs et leur popularité a été principalement du au nombre important d'applications que ces réseaux envisagent. Voici dans ce qui suit quelques scenarios d'utilisation des WSANs :

- **Détection et l'extinction du feu :** c'est un exemple typique des WSANs avec lequel les nœuds capteurs et les nœuds actionneurs coopèrent ensemble dans la détection et l'extinction du feu. En particulier, les nœuds capteurs détectent tout départ du feu et informent les nœuds actionneurs (des robots par exemple) qui agissent pour éteindre ce feu. Ces nœuds capteurs peuvent contribuer au guidage de ces nœuds actionneurs durant leur opération d'extinction étant donné que l'intervention humaine peut ne pas être possible [7].
- **Détection d'intrusion :** avec ce scénario, les nœuds capteurs observent passivement toute occurrence d'évènement. Dans le cas d'une détection positive, ils informent d'autres nœuds actionneurs qui agissent afin d'identifier l'intrus ou tout simplement pour lui faire peur [7].
- **Contrôle de température :** c'est un exemple simple avec lequel les nœuds capteurs et actionneurs coopèrent pour garder la température ambiante à un certaine niveau [7].
- **Arrosage automatique :** avec un tel scénario, les nœuds capteurs sont dispersés dans le champ afin de prélever toute mesure pertinente à l'environnement ambiant et à l'état du sol (e.g. température atmosphérique, humidité du sol, heures d'ensoleillement, etc.) et les nœuds actionneurs sont responsable d'arroser le sol.

1.12 Conclusion

Dans ce chapitre, nous avons présenté brièvement le principe des réseaux de capteurs et d'actionneurs sans fil avec toutes les notions y sont pertinentes à savoir : les modes de communication, les objectifs de conception visés, les contraintes à considérer, les formes de consommation de l'énergie des nœuds capteurs, les techniques principales de la conservation de cette énergie, la communication temps-réel, les protocoles de routage dans ces réseaux et dernièrement quelques scénarios d'application.

Après ce premier chapitre, nous passons vers un autre domaine qui est pertinent à notre travail. Nous nous intéresserons particulièrement aux algorithmes de colonies de fourmis et à leur application au problème de routage dans les WSNs et les WSANs. Cela fera l'objet du deuxième chapitre.

Chapitre 2

Les algorithmes de colonies de fourmis et leur application aux WSANs

2.1 Introduction

Dans ce chapitre, nous allons présenter, brièvement, le principe des algorithmes de colonies fourmis et leur application au problème de routage dans les réseaux informatiques en général et dans les WSANs en particulier. Nous allons tout d'abord décrire brièvement le comportement des fourmis dans la nature pour mettre l'accent sur leur particularité qui a mené à leur imitation. Ensuite, nous passons à une présentation brève de l'algorithme *Ant System* représentant la première application d'un algorithme de colonie de fourmis à un problème d'optimisation qui est celui du voyageur du commerce. Le principe d'application de ces algorithmes à celui du routage dans les réseaux informatiques est ensuite présenté ainsi que quelques travaux connexes à cette application dans les WSNs et les WSANs.

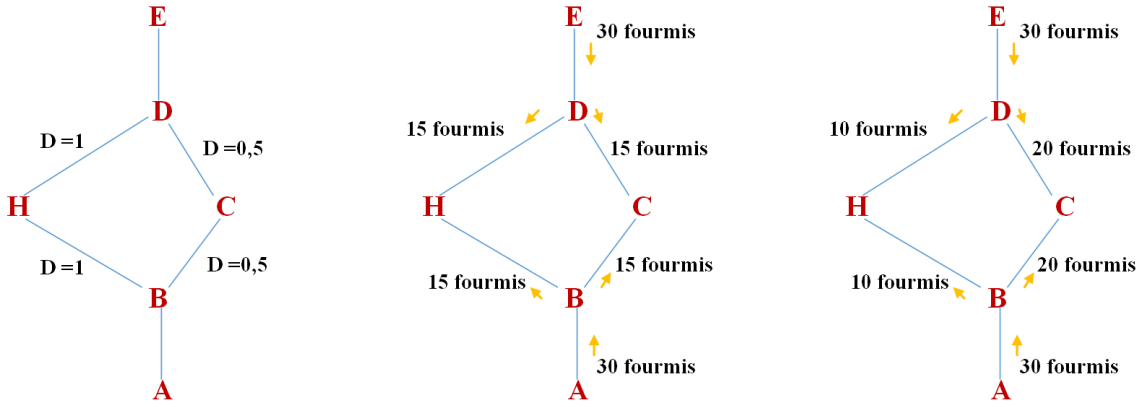
2.2 Les fourmis dans la nature

Les colonies de fourmis sont des organisations sociales hautement organisées. En fait, les fourmis se comportent collectivement et peuvent accomplir des tâches complexes dont la capacité de leur accomplissement dépasse de loin les capacités individuelles d'une seule fourmi [2, 15].

La communication entre ces fourmis se fait via l'environnement à travers le dépôt et la sensation d'une certaine substance chimique volatile appelée phéromone. Une fourmi en mouvement, place de la phéromone en quantités variables sur le sol et donc, elle trace un chemin à travers ce produit. Une autre fourmi se déplace aléatoirement et rencontre cette trace de phéromone déjà déposée qu'elle peut découvrir et décide de la suivre avec une forte probabilité, renforçant ainsi le chemin avec sa propre phéromone [16, 17].

Particulièrement, c'est au travers ce processus que les fourmis puissent déterminer le plus court chemin entre la source de nourriture et leur nid. Pour illustrer le processus d'émergence de ce plus court chemin, nous reprenons l'exemple présenté dans [16] qui est illustré par la figure (2.1). Supposons qu'à chaque unité de temps, 30 fourmis arrivent à D de E et 30 autres fourmis arrivent à B de A. Nous supposons également qu'une telle unité de temps leur permet de faire une distance de longueur 1. Supposons qu'à l'instant $t = 0$, 30 fourmis sont au point D et 30 autres sont au point B. Étant donné que leur choix du chemin à emprunter est complètement aléatoire (la phéromone n'a pas été

encore déposée), elles ont donc la même probabilité d'aller vers l'un des deux points H ou C. Nous supposons comme l'illustre la figure (2.1b), qu'à partir de chaque noeud D et B, au moyenne 15 fourmis choisissent d'aller vers H et les 15 autres fourmis vers C.



(a) (b) (c)

FIGURE 2.1: Processus naturel de l'émergence du plus court chemin entre deux points D et B via le processus de recherche des fourmis.

À $t = 1$, les 30 fourmis qui arrivent à B à partir de A, vont trouver une concentration de 30 sur le chemin qui mène vers C, une concentration qui a été obtenue en prenant la somme des quinze fourmis passant par B et des quinze fourmis passant par D. En revanche, elles vont trouver une concentration de 15 sur le chemin qui mène vers H qui correspond uniquement aux 15 fourmis qui ont emprunté ce dernier chemin. De toute évidence, la probabilité d'emprunter l'un des deux chemins n'est pas la même et donc, celui avec la plus grande concentration de phéromone, celui le plus court, va connaître un nombre plus important de fourmis. Cela est illustré par la figure (2.1c) où, parmi les 30 fourmis arrivant à B à $t = 1$, 20 d'entre elles vont choisir le chemin qui mène vers C, les 10 autres vont emprunter celui qui mène vers H [16]. Avec le temps, davantage de fourmis vont emprunter le chemin BCD et le long chemin BHD finira par attirer moins de fourmis étant donné que la phéromone s'évapore avec le temps.

Cette capacité à faire émerger ce plus court chemin à partir de simples comportements a inspiré les scientifiques à en imiter pour créer des algorithmes pouvant résoudre des problèmes d'optimisation similaires [2, 15].

2.3 Algorithmes de colonies de fourmis et problème du voyageur de commerce

Les algorithmes de colonies de fourmis (abrégé en ACO) sont des algorithmes inspirés par les études sur le comportement des fourmis réelles. Ils ont été initialement proposés par Marco Dorigo dans les années 1983, pour comprendre comment les animaux presque aveugles comme les fourmis peuvent trouver les chemins les plus courts du nid à la source de nourriture [16, 17]. Ces algorithmes forment une classe particulière de méta-heuristiques pouvant être appliquées à une certaine classe de problèmes s'intéressant à la détermination du plus court chemin.

Le premier algorithme de colonie de fourmis proposé est appelé Ant System (AS). Il a été proposé par Dorigo dans [16] pour résoudre le problème de voyageur de commerce

(PVC). Ce problème qui consiste à trouver le plus court chemin permettant de relier un ensemble de villes données avec la condition que chaque ville soit visitée une et une seul fois [17].

Dans l'algorithme Ant System (AS), chaque fourmi est placée dans une ville choisie aléatoirement. Elle possède une mémoire stockant l'ensemble des villes qui seront visitées en commençant par cette première ville de départ. Une fourmi k qui est sur une certaine ville i, choisit d'aller vers une autre ville j qu'elle n'a jamais visitée selon une certaine règle de transition qui dépend de l'intensité de phéromone sur l'arête qui connecte i à j reflétant le nombre de fourmis qui ont emprunté l'arrête (i, j), et aussi de la distance entre ces deux villes (appelée également valeur de visibilité) qui favorise la sélection des villes proches de i. Cette règle est comme suit [15, 17] :

$$p_{i,j}^{k}(t) = \begin{cases} \frac{[\tau_{i,j}(t)]^{\alpha} \cdot [\eta_{i,j}]^{\beta}}{\sum_{l \in J_i^k} \left[\tau_{i,l}(t)\right]^{\alpha} \cdot \left[\eta_{i,l}\right]^{\beta}} & si\ j \in N_i^k \\ 0 & sinon \end{cases} \tag{2.1}$$

Dans cette équation (2.1), nous définissons N_i^k comme l'ensemble des villes non encore visitées par la fourmi k, α et β sont deux paramètres qui représentent l'importance relative entre phéromone et de l'information heuristique, $\tau_{i,j}(t)$ représente la quantité de phéromone sur l'arête (i, j) à l'instant t, $\eta_{i,j} = 1/d_{i,j}$ où $d_{i,j}$ est la distance entre la ville i et la ville j.

Lorsque la fourmi termine la visite de toutes les villes, elle met à jour les valeurs de phéromones selon la règle suivante [17] :

$$\tau_{i,j}(t+1) = (1-\rho)\tau_{i,j}(t) + \Delta\tau_{i,j}(t) \tag{2.2}$$

Dans cette équation (2.2), $(1-\rho)$ correspond à la fraction de phéromone qui est évaporée. $\Delta\tau_{i,j}(t)$ est calculée selon la formule suivante :

$$\Delta\tau_{i,j}(t) = \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}(t) \tag{2.3}$$

Dans cette formule (2.3), m correspond au nombre de fourmis participant à cette itération courante de recherche et $\Delta\tau_{i,j}^{k}(t)$ correspond à la quantité de phéromone qui sera déposée par la fourmi k. Cette dernière valeur est calculée comme suit :

$$\Delta\tau_{i,j}^{k}(t) = \begin{cases} \frac{Q}{L^k(t)} & si\ e_{i,j} \in T^k(t) \\ 0 & sinon \end{cases} \tag{2.4}$$

Dans cette formule (2.4), L est la longueur du tour de la fourmi k, Q une constante de l'algorithme et $T^k(t)$ est la liste des villes déjà visitées.

Ce que nous venons d'expliquer est qu'une seule itération de recherche des m fourmis. En fait, ce même processus sera répété un certain nombre de fois jusqu'à l'émergence de la meilleure solution.

2.4 Algorithmes de colonies de fourmis et problème de routage dans les réseaux de communication

2.4.1 Principe général

Comme nous venons de présenter, les colonies de fourmis sont des colonies intelligentes et sociales avec des membres qui coopèrent les unes avec les autres pour apporter de la nourriture dans leur nid. Sur la base du comportement de ces fourmis, les algorithmes de colonie de fourmis ont été inspirés pour former une classe de méta-heuristiques pour plusieurs problèmes d'optimisation [18, 1]. En fait, le problème de routage dans les réseaux informatiques est parmi les problèmes les plus favorables auxquels ces algorithmes ont été toujours appliqués.

L'application communément adoptée d'un algorithme de colonie de fourmis au problème de routage consiste à utiliser des fourmis artificielles modélisées sous forme de paquets de contrôle. Ces derniers sont libérés de manière concurrente et indépendante à partir des nœuds du réseau afin de l'explorer et établir de bons chemins, sur la base de certaines métriques, entre leurs nœuds sources et ceux de destination. Chaque fourmi, $F_{s\to d}$ (*Forward Ant*), qui est libérée d'une source s, se déplace d'un nœud à un autre suivant une certaine règle de transition basée sur l'information de phéromone et potentiellement, sur une autre information locale (e.g. dans un WSAN, cette information peut correspondre à la puissance de transmission nécessaire à la communication entre deux nœuds voisins i et j). Cette fourmi fait usage d'une certaine table de phéromones qui garde trace des valeurs de phéromone est qui est sur chaque noeud i du réseau. Cette table comporte pour chaque destination d, un vecteur d'entrées de valeurs réelles avec une entrée $\tau^d_{i,j}$ pour chaque voisin j du nœud i. Ces entrées, qui coïncident aux variables de phéromones, représentent les désirabilités à choisir chaque nœud voisin j de i comme prochain saut pour aller à d. Lorsqu'une fourmi arrive à son nœud de destination, elle évalue la qualité du trajet parcouru et se convertit en une autre, appelée $F_{d\to s}$ *(Backward Ant).* Cette dernière prend le chemin inverse de sa fourmi $F_{s\to d}$ correspondante pour se retourner à son nœud source tout en mettant à jour les tables de phéromones des nœuds visités [19].

Dans ce qui suit, nous allons présenter un cadre plus détaillé qui illustre cette façon d'appliquer ces algorithmes de colonies de fourmis au problème de routage dans les réseaux informatiques.

2.4.2 Cadre général d'application

Généralement, toute application d'un algorithme de colonies de fourmis au problème de routage obéit au même schéma de fonctionnement décrit dans la section précédente avec certaines différences qui sont propres à chaque application. Par exemple, elles diffèrent selon la façon dont les paquets de contrôle qui correspondent aux fourmis sont générées, suivant la manière de mise à jour de la phéromone, etc. Afin de donner une vue générale pouvant résumer l'essentiel de ces applications, nous reprenons le cadre général proposé dans [19] qui présente un tel résumé. La figure (2.2) illustre ce cadre qui comporte, principalement, quatre modules représentant les fonctionnalités et les structures de données devant être implémentés sur chaque nœud routeur donné :

- Unité de génération et de management des agents mobiles [1],

1. Un agent mobile correspondant à un paquet de contrôle intelligent modélisant une fourmi.

- Structures de données de routage,
- Médiateur des communications d'un agent (RID [2]),
- Unité d'acheminement de données.

Dans ce qui suit, nous présentons brièvement la fonction de chaque unité.

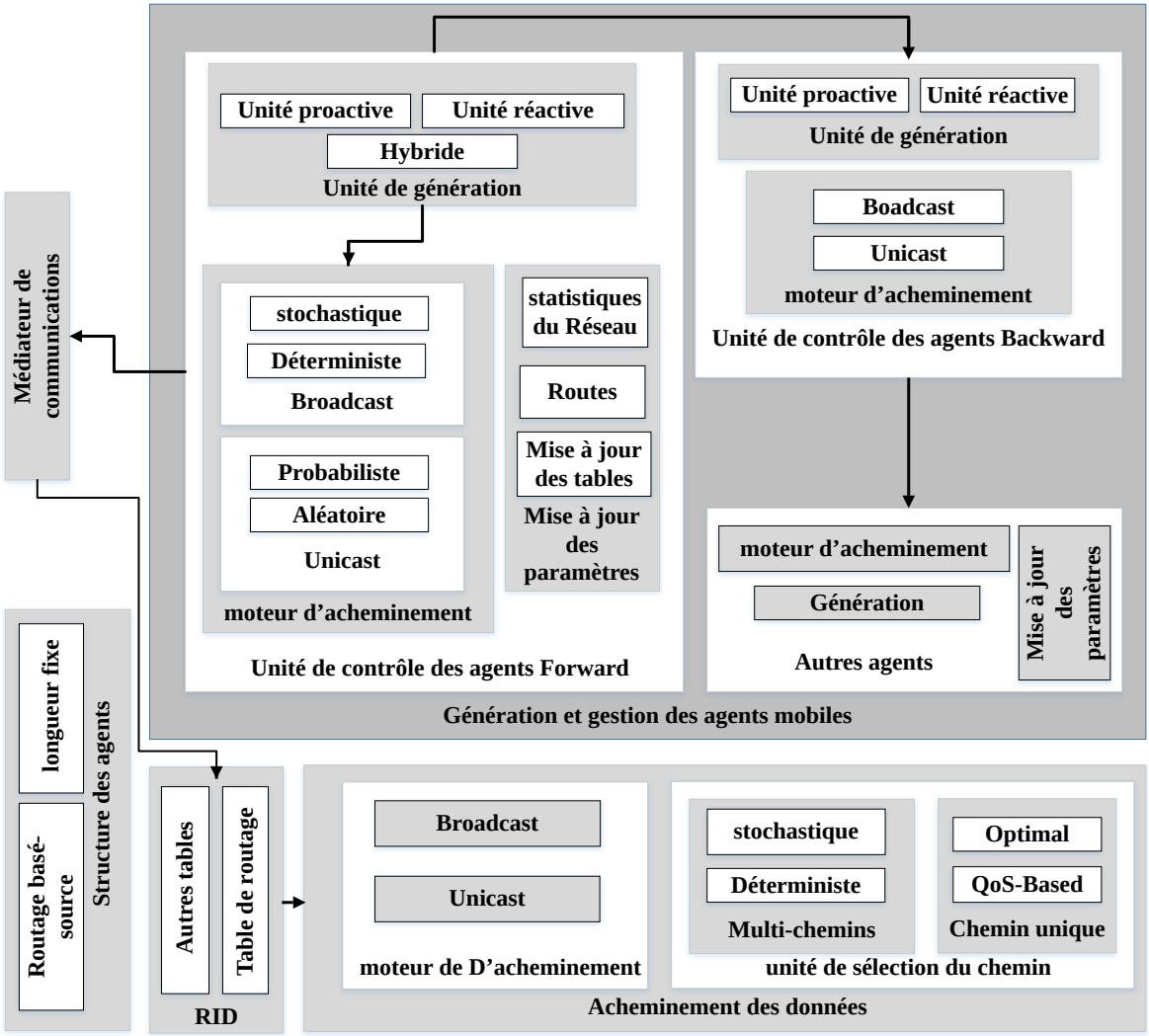


FIGURE 2.2: Cadre général résumant la manière d'application d'un algorithme de colonies de fourmis au problème de routage.

Unité de génération et de management des agents mobiles

Comme nous avons déjà cité, une application d'un algorithme de colonies de fourmis au problème de routage met en œuvre deux types de fourmis : Forward et Backward. La tâche principale d'une fourmi Forward consiste à trouver une route vers une destination particulière et de recueillir potentiellement des informations sur cette route. Une unité de contrôle spécialisée est consacrée à une telle fourmi Forward comme l'illustre la figure (2.2). Elle comprend trois composants :

- Unité de génération d'agents qui détermine la façon avec laquelle les agents Forward sont générées. Cette génération peut se faire soit de manière proactive et donc,

2. Routing Information Database.

périodiquement, soit de manière réactive. Une solution hybride peut être adoptée.

- Un moteur d'acheminement qui définit la manière d'acheminement de cet agent vers le prochain saut. En particulier, un (unicast) ou plusieurs (broadcast) nœuds voisins peuvent être sélectionnés. Généralement, le choix du prochain saut se fait sur la base de certaines probabilités qui sont calculées sur la base d'une règle de transition ou de déplacement qui dépend des valeurs de phéromone et aussi d'autres valeurs heuristiques.
- Module dédié au mise à jour des paramètres qui permet la collecte de certaines informations comme le temps de bout en bout, l'énergie des nœuds, etc. et la mise à jour de certaines structures de données.

À la différence d'une fourmi Forward, une fourmi Backward emprunte le chemin inverse de la première fourmi tout en mettant à jour les structures de données nécessaires. L'unité de contrôle propre à de tels agents comprend :

- Un bloc de génération qui décide de la génération ou pas d'une fourmi Forward comme réponse à l'arrivée d'une fourmi Backward.
- Et d'un moteur d'acheminement qui précise la manière d'acheminement de cet agent. Durant cet acheminement, les structures de données nécessaires sont mises à jour.

Outres ces deux types de fourmis Forward et Backward, d'autres fourmis peuvent également être utilisées.

Structures de données de routage

Ces informations sont des structures de données qui sont maintenues localement au niveau de chaque nœud. Elles regroupent toutes les informations nécessaires à l'acheminement des fourmis et des données. Ces structures comprennent principalement les tables de routage des agents fourmis, des données et d'autres tables qui maintiennent certaines statistiques connexes à l'état des nœuds et celui du réseau.

Médiateur des communications d'un agent

Ce module assure toutes les fonctions nécessaires à l'accès et à la mise à jour des différentes tables de routage à l'intérieur d'un noeud donné par les agents.

Unité d'acheminement de données

Ce module décide de la façon dont les paquets de données doivent être acheminées. Il fait usage des informations collectées par les agents fourmis qui sont disponibles au niveau des tables de routage. Ce module comporte deux composants :

- Un moteur d'acheminement qui envoie le paquet donnée, principalement, vers le prochain saut.
- Moteur de sélection de chemin (s) à emprunter. Deux manières sont généralement adoptées. Soit en choisissant une multitude de chemins qui présente l'avantage d'assurer un équilibre en ce qui concerne la consommation énergétique des nœuds, soit, en choisissant le meilleur chemin déjà défini par les fourmis.

2.5 Classification des protocoles de routage basés-ACO dans les WSNs

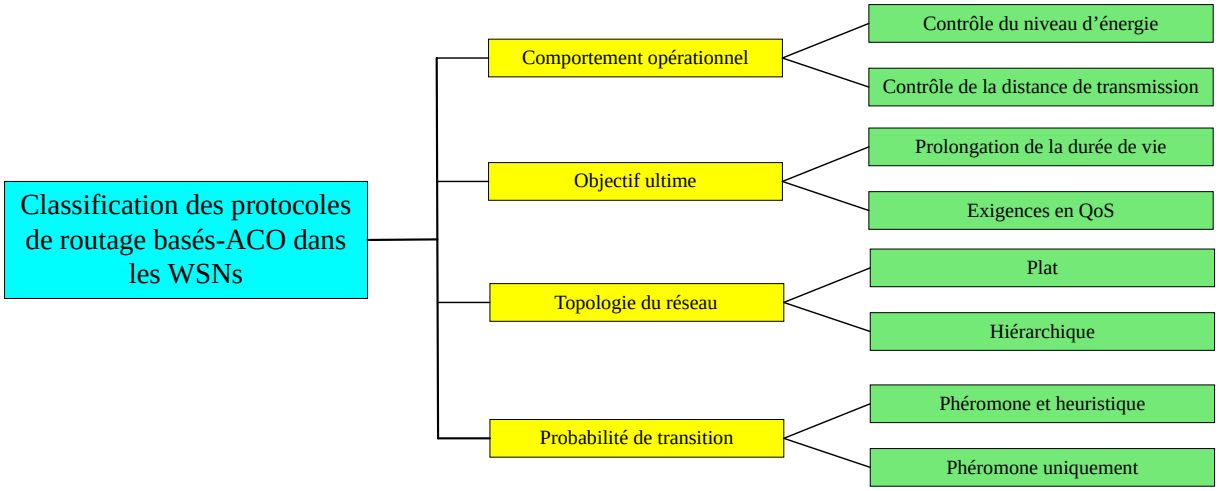


FIGURE 2.3: Classification des protocoles de routage basés-ACO dans les WSNs.

Les protocoles de routage qui sont basés sur les algorithmes de colonies de fourmis peuvent être classés en plusieurs classes. Une classification possible est celle proposée dans [20] que nous présentons brièvement dans ce qui suit. La figure (2.3) illustre cette classification. Comme le montre cette figure, ces protocoles de routage peuvent être classés selon les quatre points de vue suivants :

- **Comportement opérationnel :** un protocole de routage basé-ACO qui est dédié aux WSNs peut visé soit le contrôle de l'énergie des nœuds ou la minimisation de la distance à la quelle le noeud capteur émet. Dans le premier cas, le choix des nœuds voisins qui ont une énergie résiduelle importante est favorisé. Dans le deuxième, l'objectif est de déterminer la distance de transmission appropriée de chaque noeud.
- **Objectif ultime :** de tels protocoles de routage peuvent également être classés selon si le protocole vise la maximisation de la durée de vie du réseau ou l'assurance de certaine qualité de service.
- **Topologie du réseau :** selon la topologie réseau, le routage peut être plat ou hiérarchique. Dans le premier type, chaque nœud effectue la même tâche de recherche de chemin alors qu'avec le deuxième, plusieurs groupes de nœuds sont formés et la sélection des chemins se fait entre groupes.
- **Probabilité de transition :** un routage basé ACO peut également être catégorisé selon la règle de transition des fourmis. Particulièrement, selon si cette règle considère conjointement la phéromone et la valeur heuristique ou uniquement de la phéromone.

2.6 Quelques travaux connexes

Dans ce qui suit, quatre protocoles de routage basés-ACO qui sont appliqués aux réseaux ad hoc et aux WSNs sont brièvement présentés. Le tableau (2.1) illustre la com-

paraison entre les quatre protocoles présentés vis-à-vis de la classification présentée dans la section précédente.

Description de protocole « CEDAR »

CEDAR (Cost Efficient Delay-bounded Ad hoc Routing) est un protocole de routage réactif dédié spécialement aux réseaux ad hoc. Il vise à trouver un chemin entre un noeud source et un autre de destination en procédant à une assignation de puissances de transmission qui minimise la somme des puissances assignées et qui donne une route avec un nombre de sauts total qui ne dépasse pas une certaine limite donnée.

CEDAR procède selon deux étapes, une première étape d'initialisation où une route initiale qui connecte le noeud source du noeud de destination est établie, et une deuxième phase d'évolution qui vise à faire évoluer, à travers la recherche des fourmis, la première route construite vers une deuxième répondant aux deux objectifs cités auparavant.

La première étape débute par le noeud source lorsqu'il y a des données à envoyer. Cela se fait via la diffusion d'un message RREQ (Route Request) à son voisinage. Chaque noeud recevant ce message le rediffuse jusqu'à ce qu'il arrive au noeud de destination. Une fois au niveau de ce dernier, le message RREQ est converti en un message RREP (Route Replay). Ce dernier retourne vers le noeud source sur la base de la route initialement empruntée par RREQ. Un message RREP collecte, durant son passage, les informations qui concernent le coût et le nombre de sauts du chemin emprunté, et met à jour, à chaque fois les structures de données nécessaires sur ces nœuds.

Lorsque le message RREP arrive au noeud source, la deuxième étape débute en libérant périodiquement une fourmi Forward avec comme rôle la détermination et le maintien du meilleur chemin étant donné que les nœuds sont mobiles.

Chaque fourmi Forward peut prendre l'un des deux types suivants : déterministe ou non-déterministe. Une fourmi non-déterministe a comme rôle l'exploration et la découverte d'un meilleur chemin alors qu'une fourmi déterministe vise à maintenir à chaque fois le meilleur chemin trouvé et aussi à renforcer l'intensité de phéromone sur ce même chemin afin de faire orienter les fourmis aux alentours.

Une fourmi Forward se déplace d'un noeud à un autre selon une certaine règle de transition qui dépend de son type. Lorsqu'elle arrive au noeud de destination, elle se convertit en fourmi Backward et emprunte le chemin inverse de la fourmi Forward correspondante. Une fourmi Backward procède similairement à un message RREP en collectant les informations connexes à l'état du chemin et aussi en mettant à jour les structures de données appropriées.

En fait, dans notre travail, nous avons appliqué une application centralisée de cet algorithme CEDAR. Les règles de transition des fourmis et les règles de mise à jour des structures données sont les mêmes à part d'autres modifications qui concernent l'application centralisée au lieu de celle distribuée du protocole. Tous ces détails y compris nos propres modifications seront présentés dans le chapitre suivant.

Description du protocole « EEABR »

le protocole EEABR (Energy-Efficient Ant-Based Routing) [21] est une amélioration d'un premier protocole BABR [21]. L'algorithme vise à réduire d'un côté, la charge de communication due à la communication excessive des fourmis et d'un autre côté, l'épuisement énergétique qui peut être causé par la communication de données. Deux considérations sont prises en compte lors de la mise à jour de la phéromone au cours du processus de

création de chemin, le niveau d'énergie de chaque noeud et les longueurs des chemins qui seront construits.

Dans EEABR, chaque fourmi Forward stocke dans sa mémoire M_k uniquement, l'identifiant du noeud précédent et du noeud courant afin d'éviter un stockage excessif d'informations. Une fourmi k qui est sur le noeud i sélectionne son prochain saut en calculant pour chaque voisin j la probabilité de transition comme suit :

$$P_k(i,j) = \begin{cases} \frac{[T(i,j)]^{\alpha}[E(j)]^{\beta}}{\sum_{u \notin M_k}[T(i,u)]^{\alpha}[E(u)]^{\beta}} & si\ j \notin M_k \\ 0 & sinon \end{cases} \tag{2.5}$$

Dans cette formule (2.5), $T\,(i,j)$ correspond à l'intensité de phéromone sur le lien (i,j) et $E\,(j)$ correspond à la fonction de visibilité qui est calculée comme suit : $1/(C-e_j)$ où C représente l'énergie initiale des nœuds et e_j correspond à l'énergie résiduelle du noeud j.

Une fourmi Forward qui arrive à sa destination se convertit en fourmi Backward et procède à la mise à jour des pistes de phéromone du chemin visité. La quantité de phéromone déposée est donnée par la formule suivante :

$$\Delta T_k = \frac{1}{C - \frac{E_k^{min} - Fd_k}{E_k^{avg} - Fd_k}} \tag{2.6}$$

Dans cette formule (2.6), Fd_k est le nombre de nœuds visités par la fourmi Forward k, E_k^{min} et E_k^{avg} sont respectivement la valeur minimale et moyenne du vecteur E_k qui enregistre les énergies résiduelles des nœuds traversés par K. La table de routage est ensuite mise à jour comme suit :

$$T_k(i,j) = (1-\rho)T_k(i,j) + \left[\frac{\Delta T_k}{\varphi B d_k}\right] \tag{2.7}$$

Dans cette formule (2.7), φ est un coefficient, Bd_k est la distance parcourue par k, représentée par le nombre de nœuds visités par la fourmi Backward k.

L'idée dernière cette mise à jour est de construire la meilleure distribution de phéromones en faisant en sorte que les nœuds proches du puits aient plus de niveaux de phéromones et de forcer les nœuds qui sont loin du puits à trouver de meilleurs chemins.

Description du protocole « ACO-QoSR »

ACO-QoSR [22] s'agit d'un protocole de routage destiné à répondre à des exigences de délai strictes et à des ressources de puissance et de calcul limitées dans les nœuds de capteurs. Le problème abordé est de trouver des chemins entre nœuds capteurs et noeud source qui répondent aux deux restrictions suivantes : le temps de bout en bout doit être au-dessous d'une certaine valeur limite donnée D, et le rapport entre l'énergie résiduelle et l'énergie initiale ($ERR = E_{residual}/E_{intial}$) dépasse un certain seuil donné.

Dans $ACO-QoSR$, lorsqu'un nœud capteur source veut délivrer des données, il vérifie sa table de routage pour trouver un chemin approprié. S'il n'y a pas de tel chemin dans

cette table, un processus de recherche de chemin est lancé pour trouver une nouvelle route. Une fourmi donnée qui est au niveau du noeud i sélectionne son prochain saut j, parmi tous les voisins N_i de i, selon la probabilité de sélection suivante :

$$P_{i,j} = \frac{[\tau_{i,j}]^{\alpha} [\eta_{i,j}]^{\beta}}{\sum_{k \in N_i} [\tau_{i,k}]^{\alpha} [\eta_{i,k}]^{\beta}} \tag{2.8}$$

Dans cette formule (2.8), l'information heuristique $\eta_{i,j}$ est définie comme le rapport entre l'énergie résiduelle du nœud j et l'énergie résiduelle totale de tous les nœuds voisins de i. Cette valeur est calculée comme suit :

$$\eta_{ij} = \frac{E_{residual}(j)}{\sum_{k \in N_i} E_{residual}(k)} \tag{2.9}$$

Cette dernière formule (2.9) est définie pour favoriser les nœuds ayant une énergie résiduelle plus importante afin d'équilibrer la consommation d'énergie. L'intensité de phéromone de chaque lien (i, j) est mise à jour par la fourmi Backward comme suit :

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t)$$

Où l'incrément de phéromone de la fourmi k est calculée comme suit :

$$\Delta\tau^{k} = \begin{cases} f\left(ERR_k^*, \Delta\tau^{k-1}\right) & Delay(k) \leq D \\ 0 & Delay(k) \succ D \end{cases} \tag{2.10}$$

Dans la formule (2.10), $Delay(k)$ est le délai du chemin parcouru par la fourmi k et ERR_k^* correspond au rapport de l'énergie résiduelle normalisée du chemin, c'est-à-dire :

$$ERR_k^* = \frac{ERR_k}{Hop_k} \tag{2.11}$$

Dans cette formule (2.11), ERR_k est le rapport de l'énergie résiduelle totale du trajet et Hop_k est le nombre total de nœuds visités par la fourmi k . Dans la formule (2.10), l'incrément de phéromone est décrit davantage par la fonction suivante :

$$f(ERR_k^*, \Delta\tau^{k-1}) = \begin{cases} \Delta\tau^{k-1} + \lambda(ERR_k^* - ERR_{k-1}^*), & ERR_k^* \succ ERR_{k-1}^* \\ \Delta\tau^{k-1}, & ERR_k^* \leq ERR_{k-1}^* \end{cases}$$

Donc, à partir de la formule (2.10), si le délai est supérieur à la limite en temps donnée D, la phéromone sur le trajet parcouru ne sera pas augmentée. Sinon, une certaine quantité de phéromone tend à être ajoutée.

TABLE 2.1: Comparaison des protocoles de routage présentés.

Référence	**Protocole**	**Comportement opérationnel**	**Objectif ultime**	**Topologie du réseau**	**Probabilité de transition**
[1]	CEDAR	Contrôle des distances de transmission	Exigences en QoS	Plat	Phéromone et heuristique
[21]	EEABR	Contrôle du niveau d'énergie	Prolongation de la durée de vie	Plat	Phéromone et heuristique
[22]	ACO-QoSR	Contrôle du niveau d'énergie	Exigences en QoS	Plat	Phéromone et heuristique
[23]	E&D ANTS	Contrôle du niveau d'énergie	Exigences en QoS	Plat	Phéromone et heuristique

Description du protocole « E&D ANTS »

E&D ANTS est un protocole qui vise à étendre la durée de vie du réseau tout en assurant des communications en temps-réel. C'est un protocole réactif dans lequel plusieurs fourmis Forward sont utilisées pour découvrir de meilleurs chemins avec un minimum d'énergie et délai. Donc, l'objectif est de minimiser le modèle suivant :

$$g(t) = min\left\{Energy * Delay\right\} \tag{2.12}$$

Toute fourmi qui est au niveau d'un noeud i sélectionne son prochain saut en calculant pour chaque noeud voisin j sa probabilité de sélection somme suit :

$$p_{i,j} = \frac{\varpi\tau_{ij} + (1-\varpi)\,\eta_{i,j}}{\varpi + (1-\varpi)\,(\mid L_i \mid -1)} \tag{2.13}$$

Dans cette formule (2.13), ϖ, avec $(0 \leq \varpi \leq 1)$, est un facteur de pondération et L_i est l'ensemble voisins du nœud i. L'information heuristique $\eta_{i,j}$ est définie comme suit :

$$\eta_{i,j} = \frac{e_j}{\sum_{k \in L_i} e_k} \tag{2.14}$$

Dans cette formule (2.14), e_j correspond à l'énergie résiduelle du nœud j. D'après la formule (2.13), le choix du prochain saut est déterminé selon la valeur de phéromone qui

encode l'information de délai et d'énergie. L'intensité de phéromone du chemin est mise à jour comme suit :

$$\tau_{ij} = \rho\tau_{ij} + \triangle\tau_{ij}$$

Dans cette dernière formule, l'incrément $\triangle\tau_{ij}$ de phéromone est calculé en considérant la qualité relative du chemin parcouru dans l'itération courante $k+1$ vis-à-vis de certaines statistiques sur la qualité des chemins trouvés dans le passé proche. Cette valeur est calculée comme suit :

$$\triangle\tau_{ij} = \tau_0 \left\{ 1 - \frac{z_{new} - g\,(k)}{z - g\,(k)} \right\} \tag{2.15}$$

Dans cette dernière formule (2.15), τ_0 est la valeur initiale de phéromone, z_{new} est le coût qui dépend du produit $Energy * Delay$ de la nouvelle solution trouvée à l'itération $(k+1)$, z correspond au coût moyen des derniers k solutions. Donc, si le coût de la meilleure solution est inférieur à la moyenne, l'intensité de phéromone du nouveau chemin augmentera, sinon elle diminuera.

2.7 Conclusion

Dans ce chapitre, nous avons donné une description brève des algorithmes de colonies de fourmis et de la façon de leur application au problème de routage dans les réseaux informatiques comme les WSNs et les WSANs. La source d'inspiration qui a mené à leur imitation, la première application de ces algorithmes et aussi leur application au problème de routage dans les réseaux informatiques ont toutes été présentées. En outres, quelques travaux connexes à l'application de ces algorithmes au problème de routage dans les WSNs et WSANs ont été également résumées.

Chapitre 3

Contribution

3.1 Introduction

Dans ce chapitre, nous allons décrire en détail notre protocole proposé que nous baptisons CADER comme acronyme de *Centralized Ant-based Delay-bounded and Energy-efficient Routing protocol*. Nous allons présenter dans un premier temps, le modèle réseau adopté dans ce présent travail avec les différentes hypothèses considérées. Nous allons donner ensuite une description brève de notre protocole pour passer ensuite à une description détaillée dans laquelle nous présentons les principaux messages échangés, les principales structures de données utilisées au niveau de chaque nœud et aussi les différentes étapes de fonctionnement du protocole.

3.2 Modèle réseau

Nous assumons notre modèle réseau comme suit :

- Notre réseau est constitué d'un ensemble de nœuds capteurs avec un nombre limité de nœuds actionneurs. Tous ces nœuds sont statiques.
- Chaque nœud du réseau est conscient de ses propres coordonnées. En fait, de telle information est généralement indispensable dans un WSAN car, les nœuds actionneurs doivent connaître la localisation des événements qui arrivent afin qu'ils puissent intervenir et entreprendre les actions les plus appropriées.
- Les nœuds capteurs peuvent ajuster leurs portées de communication en ajustant leurs niveaux de puissance de transmission sans dépasser une puissance maximale P_{max} qui correspond à une portée maximale R_{max}.
- Soit $G = \{V, E\}$ le graphe qui modélise la topologie de notre WSN où :
  - $V = \{v_1, v_2, ..., v_n\}$ est l'ensemble de sommets représentant les différents nœuds du réseau tel que $|V| = n$. Soit A, avec $A \subset V$, l'ensemble de nœuds actionneurs.
  - Et E l'ensemble d'arcs (liens) qui relient ces nœuds et qui est défini comme suit :

    $$E = \{(u, v) | u \in V, v \in V, \delta_{u,v} \leq R_{max}\}$$

    sachant que :

    $$\delta_{u,v} = \sqrt{(x_u - x_v)^2 + (y_x - y_v)^2}$$

représente la distance euclidienne entre les deux nœuds u et v, avec $u = (x_u; y_u)$ et $v = (x_v; y_v)$.

- Soit $S = \{s_1, s_2, ..., s_m\}$ l'ensemble de nœuds sources qui captent et collectent les données de l'environnement, tel que $|S| = m$, $S \subseteq V$ et $S \cap A = \emptyset$.
- Similairement à [24], la puissance de transmission nécessaire pour que le nœud i communique avec j est calculée en assumant la formule :

$$power_{i,j} = \delta_{i,j}^{\alpha}$$

où α représente l'exposant d'affaiblissement de propagation "path loss".

3.3 Définition du problème

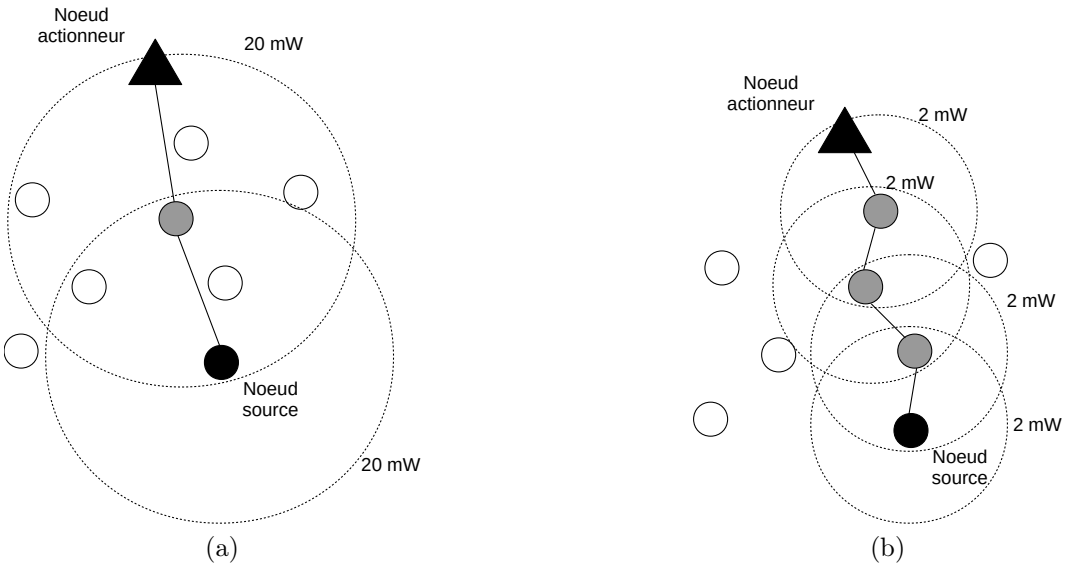


FIGURE 3.1: Deux chemins potentiels qui connectent le nœud capteur source avec le nœud actionneur : (a) un chemin de 2 sauts avec une puissance de transmission totale de 40 mW, (b) une route de 4 sauts avec une puissance totale de 8 mW [1].

Dans ce présent travail, nous nous intéressons au routage des données prélevées par les nœuds capteurs et leur délivrance au noeud actionneur approprié. Ce routage doit être économe en énergie à cause des ressources énergétiques limitées des nœuds capteurs. Il doit également assurer une latence qui ne dépasse pas une certaine échéance donnée car, les nœuds actionneurs doivent agir rapidement. En fait, pour économiser plus d'énergie, il faut prendre des routes avec des nœuds qui émettent sur de courtes distances et donc des routes avec un nombre de nœuds plus important. Cela mène à son tour, à des routes qui génèrent une grande latence. Par conséquent, étant donnée que la conservation de cette énergie se fait au détriment d'un temps de latence plus important, un tel routage doit assurer un compromis entre la conservation de l'énergie des nœuds et le respect de l'échéance imposée. La figure (3.1) illustre clairement le problème. Cette figure qui est adaptée de [1], présente un exemple de deux chemins possibles qui connectent un noeud capteur avec son nœud actionneur correspondant. Le premier chemin, illustré par la figure (3.1a), a une longueur de 2 sauts avec une puissance de transmission totale de 40 mW,

alors que le deuxième chemin qui est illustré par la figure (3.1b) a une longueur de 4 sauts avec une puissance totale de 8 mW. Donc, le deuxième chemin permet d'économiser plus d'énergie par rapport au premier chemin au détriment d'une longueur de route plus importante et donc, une latence plus grande.

3.4 Vue d'ensemble de notre protocole CADER

CADER est un protocole de routage qui est conçu spécialement pour les WSANs. Son rôle principale est d'établir, lorsque des évènements arrivent, des routes qui connectent les nœuds capteurs sources avec leurs nœuds actionneurs correspondants. Ces routes doivent assurer la délivrance en temps opportun (i.e. avant la borne temporelle imposée) des données prélevées tout en minimisant l'énergie qui est consommée durant cet acheminement.

Notre protocole procède suivant trois étapes :

1. Comme première étape, une clusterisation du réseau en clusters est réalisée avec l'établissement d'un arbre de routage du plus court chemin qui connecte chaque cluster-head avec ses membres. Avec cette clusterisation, chaque nœud actionneur est considéré comme cluster-head et les nœuds capteurs les plus proches de lui en termes de nombre de sauts représentent ses membres. L'arbre de routage créé nous servira par la suite pour la collection et la délivrance des informations des nœuds membres à leurs cluster-heads et aussi pour la notification de ces derniers de l'occurrence des événements.
2. Dans la deuxième étape, les positions des nœuds membres avec leurs identifiants sont collectés et acheminés vers leurs cluster-heads. Chacun de ces derniers et sur la base de ces informations crée la matrice d'adjacence de son sous-réseau qui lui servira lors du calcul des routes demandées.
3. Durant la troisième étape et lorsqu'un nœud capteur donné détecte un événement, il informe son nœud actionneur correspondant en utilisant l'arbre de routage établi dans la première étape. Ce nœud actionneur lui calcule une route appropriée en appliquant l'algorithme de colonie de fourmis et lui renvoie cette route lorsqu'il termine. À ce moment, le nœud capteur peut commencer à envoyer ses données sur la base de cette nouvelle route construite.

3.5 Description détaillée de notre protocole CADER

Dans cette section, nous allons procéder à une description détaillée de notre protocole CADER en présentant les principales structures de données utilisées, les principaux messages échangés et aussi les différentes étapes de fonctionnement.

3.5.1 Structures de données utilisées

Le fonctionnement de notre protocole se base sur certaines structures de données qui sont utilisées au niveau de chaque nœud. Ces structures diffèrent selon le type du nœud (nœud capteur ou nœud actionneur). Dans ce qui suit, nous décrivons brièvement les principales structures de données utilisées.

Structures de données utilisées au niveau du nœud capteur

Un nœud capteur a comme rôle, le prélèvement et la communication des données propres à son environnement immédiat. Il participe également à leur acheminement vers le nœud actionneur. Un nœud capteur i fait usage principalement, des structures de données suivantes :

- **Parent_i :** signifie l'identifiant du nœud parent du nœud i dans l'arbre du plus court chemin qui sera établi dans la première étape du fonctionnement de notre protocole.
- **Id_i :** correspond à l'identifiant du nœud i.
- **ActorId_i :** correspond à l'identifiant du nœud actionneur avec lequel le nœud capteur i sera associé.
- **$\text{HopToActor}_\text{i}$:** signifie le nombre de sauts qui sépare le nœud i de son nœud actionneur correspondant dans l'arbre du plus court chemin.
- **$\text{childNodes}_\text{i}$:** signifie l'ensemble de nœuds fils du nœud i dans l'arbre du plus court chemin.

Structures de données utilisées au niveau du nœud actionneur

Un nœud actionneur s'intéresse aux différentes données prélevées par les différents nœuds capteurs pour réagir ensuite et prendre les actions les plus appropriées. Un nœud actionneur a fait usage, comme un nœud capteur, des deux structures de données suivantes : Id_a et $childNodes_a$ en plus des structures suivantes qui sont propres à l'application de l'algorithme de colonie de fourmis :

- **$\text{AdjacencyMat}_\text{a}$:** la matrice d'adjacence connexe à la partie du réseau qui correspond au cluster de l'actionneur a.
- **delayTab_a :** c'est une matrice qui sauvegarde pour chaque couple *(i,j),* le délai $delayTab_i(j)$ du chemin qui sépare le nœud i de son nœud actionneur en empruntant le voisin j.
- **$\text{bestCostTab}_\text{a}$:** c'est une matrice qui sauvegarde pour chaque nœud i le coût du meilleur chemin déjà connu $bestCost_i^d$ qui mène vers le nœud actionneur destinataire d.
- **$\text{pheromoneTab}_\text{a}$:** c'est une matrice qui sauvegarde pour chaque couple *(i,j),* l'intensité de phéromone $\tau_{i,j}^d$ sur le lien qui connecte i à j.

3.5.2 Liste et formats des paquets

Différents messages sont échangés entre les différents nœuds du réseau au cours du fonctionnement de notre protocole CADER. La liste de ces messages et les détails de chacun d'eux sont présentés dans ce qui suit.

HCM (Hop Configuration Message)

Ce message permet la clusterisation du réseau et l'établissement de l'arbre du plus court chemin qui servira à la collection des informations des nœuds membres et aussi à la notification de l'occurrence des événements et la demande d'établissement de routes. La distance entre le nœud actionneur et chaque nœud capteur est calculée en nombre de sauts. Cela se fait à partir du nœud actionneur qui est responsable de la diffusion initiale du message HCM. Un tel message se compose, comme l'illustre la figure (3.2), des quatre champs suivants :

- **ActorId :** stocke l'identifiant du nœud actionneur qui a commencé la diffusion du message.
- **Sender :** correspond à l'identifiant du nœud qui commence la diffusion/retransmet le message.
- **HopToActor :** stocke la distance en nombre de sauts dont ce message a connu depuis sa diffusion par le nœud actionneur.
- **Parent :** stocke l'identifiant du nœud parent d'un nœud donné. Il servira à la détermination des nœuds fils de chaque nœud dans l'arbre de routage.

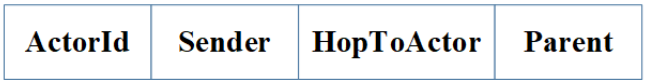


FIGURE 3.2: La structure d'un message HCM (Hop Configuration Message).

CIM (Collecting Information Message)

Le message CIM permet la collection des informations propres aux nœuds capteurs membres de chaque cluster, en particulier leurs coordonnées et leurs identifiants, et leur délivrance au nœud actionneur correspondant. Chaque message CIM contient l'identifiant du prochain saut (*NextHop*) et aussi la liste des informations des nœuds parcourus lors de l'acheminement du message CIM. Les coordonnées et l'identifiant de chaque nœud parcouru sont sauvegarder dans le message.

FIGURE 3.3: La structure d'un message CIM (Collecting Information Message).

RFR (Request For Route Message)

Ce message est créé par chaque nœud capteur voulant notifier son nœud actionneur de l'occurrence d'un événement tout en lui demandant l'établissement d'une route appropriée. Le message RFR se compose, comme l'illustre la figure (3.4), des trois champs :

- **Source :** ce champ détermine l'identifiant du nœud capteur source qui a détecté l'évènement.
- **Destination :** ce champ stocke l'identifiant du nœud actionneur destinataire.
- **NextHop :** ce champ enregistre l'identifiant du prochain saut.

FIGURE 3.4: La structure d'un message RFR (Request For Route Message).

RM (Route Message)

Ce message permet de notifier le nœud source de sa route calculée par son nœud actionneur correspondant via l'application de l'algorithme de colonie de fourmis. Un paquet RM qui est généré par le nœud actionneur comporte, comme l'illustre la figure (3.5), les deux champs suivants :

- **Destination :** correspond à l'identifiant du nœud destinataire du message, c.à.d. le nœud capteur dont le nœud actionneur a calculé la route.
- **Route :** la route calculée par le nœud actionneur c.à.d. les identifiants des nœuds capteurs qui sont inclus dans cette route.

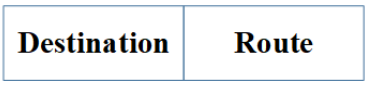


FIGURE 3.5: La structure d'un message RM (Route Message).

3.5.3 Étapes de fonctionnement

Notre protocole fonctionne suivant les trois étapes que nous décrivons dans la suite de cette section, à savoir : (1) formation des clusters, (2) Collection des informations des nœuds membres et, (3) Notification de l'occurrence de l'événement par le nœud source et la formation de sa route.

Formation des clusters

Dans cette première étape, l'organisation du réseau en clusters est réalisée et l'établissement des arbres qui connectent les nœuds membres avec leurs cluster-heads est fait. Chacun de ces clusters est formé autour d'un actionneur donné qui assure le rôle du cluster-head. Chaque nœud capteur est associé avec le nœud actionneur le plus proche de lui en termes de nombre de sauts.

Cette clusterisation est amorcée par les nœuds actionneurs via la diffusion du message HCM. Ce processus de clusterisation est illustré par l'algorithme (1). Il est lancé par chaque nœud actionneur a qui diffuse à son voisinage un message HCM avec comme valeurs des champs *Sender, HopToActor, ActorId* et *Parent* respectivement, les valeurs id_a, 0, id_a et -1 [1]. Chaque nœud voisin recevant ce message vérifie, comme l'illustre la ligne 8 de l'algorithme (1), l'existence d'une meilleure route vers l'un des nœuds actionneurs en comparant la distance de son meilleur chemin (*HopToActor* [2]*)* avec une incrémentation de celle qui lui arrive avec le message HCM (HCM.HopToActor+1). Si une telle route existe alors ce nœud met à jour ses informations locales à savoir son parent, sa distance du nœud actionneur (HopToActor) et aussi l'identifiant du nœud actionneur source du message HCM (*ActorId*) comme l'illustre les lignes 9-11 de l'algorithme (1). Une fois mis à jours, il met à jour les champs du message HCM comme l'illustre les lignes 12-14 de l'algorithme (1) et rediffuse le message à son voisinage. Il est noté que chaque nœud recevant un message HCM vérifie au début s'il est le parent du nœud qui a émis le message HCM en comparant son identifiant avec celui du champ *Parent* du message HCM reçu. Si c'est le cas alors il ajoute l'identifiant du nœud qui a émis ce message (*HCM.Sender*) à l'ensemble de ses fils, sinon il supprime cet identifiant *HCM.Sender* de ce dernier ensemble si *HCM.Sender* appartient à ce même ensemble car, il n'est plus le parent de *HCM.Sender.*

À la fin de cette première étape, chaque nœud capteur connaît le nœud actionneur avec lequel il est associé, son parent qui l'emmène vers ce même actionneur et aussi l'ensemble de ses fils. Ces informations seront exploitées dans les deux prochaines étapes afin d'un

1. Indéfini.
2. Initialisée à ∞.

Algorithme 1 : Formation des clusters

```
1  Chaque noeud actionneur a ∈ A diffuse un message HCM avec HopToActor = 0,
     Sender = Id_a, ActorId = Id_a et Parent = −1;
   // R_HCM est l'ensemble de noeuds recevant un message HCM
2  pour chaque noeud i ∈ R_HCM faire
3      si HCM.Parent = Id_i alors
4          Ajouter HCM.Sender aux fils de i;
5      sinon
6          Mettre à jour l'ensemble des fils de i si nécessaire;
7      fin
8      si HopToActor_i > HCM.HopToActor + 1 alors
           // Mettre a jour mes informations locales
9          Parent_i ← HCM.Sender;
10         HopToActor_i ← HCM.HopToActor + 1;
11         ActorId_i ← HCM.ActorId;
           // Mettre à jour les champs du message HCM
12         HCM.Sender ← Id_i;
13         HCM.HopToActor ← HopToActor_i;
14         HCM.Parent ← Parent_i;
15         Le noeud i diffuse le message HCM met à jour;
16     fin
17 fin
```

côté, pouvoir collecter et délivrer les informations des nœuds membres à leurs nœuds actionneurs correspondants, et d'un autre côté, pouvoir notifier le nœud actionneur de l'occurrence d'un évènement et donc la demande d'établissement d'une nouvelle route.

Collection des informations des nœuds membres

Sur la base de l'arbre de routage établi préalablement, chaque nœud capteur peut notifier son nœud actionneur de toute donnée prélevée. Cependant, l'acheminement de ces données le long de cet arbre cause une consommation importante de l'énergie des nœuds contribuant à cet acheminement car, ces nœuds vont émettre sur de longues distances, c.à.d. vers les voisins les plus éloignés d'eux. Afin de minimiser cette énergie, il est nécessaire de chercher d'autres chemins qui minimisent cette énergie totale consommée tout en respectant la borne temporelle donnée. Pour cette raison, on va appliquer les algorithmes de colonie de fourmis localement au niveau de chaque nœud actionneur dans le but de trouver de meilleurs chemins. Mais pour que cela soit possible, chaque nœud actionneur doit posséder les informations, en particulier les coordonnées et les identifiants, propres à ses membres.

Cette collection est initiée par les nœuds bordures de chaque cluster. Un nœud bordure est un nœud qui n'a aucun nœud fils. Un tel nœud et après la fin de la première étape, crée et envoie à son parent un message CIM comportant ses propres informations comme l'illustre les lignes 2-4 de l'algorithme 2. Chaque nœud capteur recevant un tel message vérifie dans un premier temps, s'il est le parent de l'émetteur du message en comparant son identifiant avec celui du champ *Parent* du message CIM comme l'illustre la ligne 7 de l'algorithme 2. Si cela est le cas alors, il reste dans l'attente des messages CIMs des autres nœuds fils restants. Une fois tous ces messages reçus, il crée un nouveau message CIM et

met les valeurs appropriées des deux champs *Parent* et *NextHop*. Il ajoute également les informations des nœuds contenues dans les messages CIMs reçus précédemment ainsi que ses propres informations et diffuse le message à son voisinage. Cette procédure se répète d'un nœud à un autre jusqu'à ce que chaque nœud actionneur reçoive des messages CIMs de tous ses fils. Dans ce dernier cas, ce nœud actionneur récupère toutes les informations de ses membres contenues dans ces messages et crée la matrice d'adjacence connexe à la partie du réseau qui correspond à son cluster. À ce moment, il reste dans l'attente des messages de notification et de demande d'établissement de routes pouvant lui arriver lorsque des évènements sont détectés dans son cluster.

Algorithme 2 : Collection des informations des nœuds membres

```
   // Un noeud bordure est un noeud qui ne possède aucun noeud fils
 1 pour chaque noeud bordure i faire
 2 |  CIM.NextHop ← Parent_i;
 3 |  CIM.Sender ← Id_i;
 4 |  Le noeud i ajoute ses propres informations au message CIM et diffuse le
   |   message;
 5 fin
   // R_CIM est l'ensemble de noeuds recevant un message CIM
 6 pour chaque noeud j ∈ R_CIM faire
 7 |  si CIM.NextHop = Id_j alors
 8 |  |  répéter
 9 |  |  |  Enregistrer les informations des autres noeuds stockées dans le message
   |  |  |   CIM;
10 |  |  jusqu'à j reçoit des messages CIMs de tous ses fils;
11 |  |  CIM.NextHop ← Parent_j;
12 |  |  CIM.Sender ← Id_j;
13 |  |  Le noeud j ajoute les informations reçues des autres noeuds et ses propres
   |  |   informations au message CIM et diffuse le message;
14 |  fin
15 fin
```

Notification de l'occurrence de l'événement par le nœud source et la formation de sa route

Après la fin de cette deuxième étape, chaque nœud actionneur possède les informations nécessaires lui permettant de répondre aux demandes de ses membres en leur calculant les routes appropriées. Chaque nœud capteur détectant un événement notifie son nœud actionneur et lui demande de lui calculer une route en lui envoyant une message RFR (Request For Route Message). Ce dernier est acheminé le long de l'arbre crée dans la première étape. Lorsque le nœud actionneur reçoive ce message, il procède à l'application de l'algorithme de colonie de fourmis afin de lui trouver un chemin qui est énergétiquement efficace avec une latence qui ne dépasse pas la borne temporelle imposée Γ. Dans ce présent travail, *la contrainte du temps qu'on essaye de respecter correspond à un certain nombre de sauts donné qu'il ne faut pas dépasser*. En fait, la latence d'une route donnée est proportionnelle à son nombre de sauts. C'est pour cette raison qu'on s'est limité à cette considération.

Algorithme 3 : Notre application centralisée de l'algorithme CEDAR par chaque nœud actionneur.

```
1  Initilisation des tables de routages (bestCostTable, DelayTable et
     pheromoneTab);
2  pour t ← 1 à nbIteration faire
3      convertToBant ← false;
4      stop ← false;
       // La Libération d'une fourmi Backward et son acheminement
5      tant que stop ≠ true faire
6          Libérer une fourmi ForwardAnt (Fant) à partir du noeud source;
7          si (Fant.nextHop ≠ actorId) And ((Fant.det = true And
             Fant.accDelay < Γ) Ou (Fant.det = false And
             Fant.accCost < Fant.costLimit)) alors
8              selection du prochain saut de Fant;
9          sinon
10             si Fant.nextHop = actorId alors
11                 convertToBant ← true;
12             fin
13             stop ← true;
14         fin
15     fin
       // La converion de la fourmi Fant en fourmi Backward, son
          acheminement et la mise à jour des structures de données
          appropriées
16     si convertToBant = true alors
17         Convertir Fant en fourmi Backward (Bant);
18         tant que Bant.nextHop ≠ source faire
19             Mettre à jour les tables de routage nécessaires de tous les noeuds
                 voisins du noeud courant du Bant;
20             Faire déplacer Bant vers son prochain saut;
21         fin
22     fin
23     Evaporer la phéromone au niveau de tous les noeuds ;
24 fin
```

Pour l'algorithme de colonie de fourmis, nous appliquons une adaptation centralisée de l'algorithme CEDAR [1]. Ce dernier est un algorithme de routage distribué qui est dédié aux réseaux Ad hoc. Il applique cet algorithme de colonie de fourmis afin de trouver une route qui est énergétiquement efficace et à délai borné entre un nœud source et un autre nœud destinataire. Cette route doit avoir une puissance de transmission totale minimale et ne dépasse pas un nombre de sauts donné. Comme nous avons mentionné dans l'introduction, l'application répartie de cet algorithme dans les WSANs peut ne pas être bénéfique à cause du nombre de messages de contrôle important nécessaires à l'émergence d'une bonne route et qui peut causer l'épuisement des batteries des nœuds capteurs.

Pour notre application centralisée et en connaissant l'identifiant du nœud source, chaque nœud actionneur applique l'algorithme de colonie de fourmis sur la base de la matrice d'adjacence créée précédemment et aussi sur la base de certaines structures de

données, en particulier, sur les trois tables de routage suivantes : *delayTab*, *bestCostTab* et *pheromoneTab*. Le principe de cette application est illustré par l'algorithme (3). Vous avez à chaque fois une fourmi *Forward ant (Fant)* qui est libérée à partir du nœud source. Cette fourmi se déplace d'un nœud à un autre sur la base de certaines règles jusqu'à ce qu'elle arrive au nœud destinataire qui est le nœud actionneur. Une fois au niveau de ce dernier nœud, elle se convertit en fourmi *Backward Ant (Bant)* et prend le chemin inverse de sa fourmi *Fant* correspondante sur la base de la liste des nœuds visités lors du chemin aller. Cette fourmi *Bant* se déplace d'un nœud à un autre jusqu'au nœud source tout en mettant à jour les tables de routage nécessaires. Lorsque la fourmi *Bant* termine ses déplacements, donc, à la fin de chaque itération, toutes les entrées de la table de phéromone sont diminuées par un certain facteur *DECAY_FACTOR* afin d'oublier tout mauvais chemin. Cette même itération se répète un certain nombre de fois qui est égal à *nbIteration* durant lesquelles un bon chemin émerge.

Deux types de fourmis *Fant* sont distingués : une fourmi *Fant* probabiliste et une deuxième déterministe. Une *Fant* probabiliste a comme rôle l'exploration et la découverte d'un meilleur chemin. Par contre, une *Fant* déterministe permet de renforcer l'intensité de phéromone sur le meilleur chemin connu jusqu'à l'itération courante de la recherche. Ce renforcement permet d'orienter à chaque fois la recherche des fourmis probabilistes autour de ce meilleur chemin. Il est à noter que l'intensité de phéromone est toujours comprise entre 0 et 1 afin d'éviter toute situation de stagnation [3]. Une fourmi *Fant* est une structure de données qui possède principalement les champs suivants : *det, exLimit, accCost, costLimit, accDelay, delayLimit* et *VisitedNodes*. *det* détermine si *Fant* est déterministe ou pas ; *exLimit* est utilisé pour suivre le nombre d'explorations faites par *Fant* depuis sa libération *; accCost* représente le coût accumulé depuis la libération de *Fant*, il est initialisé à 0 ; *costLimit* est le coût limite au-delà duquel *Fant* n'a plus la possibilité de se déplacer, il est initialisé à $1.5 * bestCost_i^d$, ce champ avec *accCost* sont utilisés pour ne pas laisser *Fant* chercher indéfiniment ; *accDelay* est le délai accumulé par *Fant* depuis son nœud source, il est initialisé à 0 ; *delayLimit* correspond à la borne temporelle qu'il ne faut pas dépasser ; et *visitedNodes* correspond à la liste des nœuds déjà visités par *Fant.* C'est uniquement une *Fant* probabiliste qui fait usage des trois champs *exLimit*, *accCost* et *costLimit*. De manière similaire, une fourmi *Bant* fait usage des champs suivants : det, accCost, accDelay, visitedNodes qui sont similaires à ceux d'une fourmi *Fant* en plus d'un champ additionnel *localCost* qui lui servira lors du calcul du coût du chemin de retour.

Les règles de déplacement d'une fourmi *Fant* sont similaires à celles du protocole CEDAR. Si *Fant* est probabiliste alors elle a une chance de 50% d'explorer et donc, choisir son prochain saut de manière probabiliste sinon elle le choisit de manière déterministe. Cette même fourmi est également autorisée à explorer tant que son nombre d'exploration n'a pas dépassé une certaine valeur *EXPLORE_LIMIT.*

Toute fourmi *Fant* qui a choisi d'explorer sélectionne, lorsqu'elle est au niveau du nœud i, son prochain saut n parmi ses voisins N_i en calculant pour chaque voisin $j \in N_i$ sa probabilité de sélection pour aller à la destination d comme suit :

$$p_{i,j}^d = \frac{d_j^d / power_{i,j}}{\sum_{k \in N_i} d_k^d / power_{i,k}}$$

où

3. i.e. Une situation de stagnation correspond à la situation où les fourmis tendent à emprunter toujours le même chemin.

$$d_j^d = \begin{cases} 0 & si\ j \in Fant.visitedNodes \\ 1 + \frac{\tau_{i,j}^d}{bestCost_i^d + \varepsilon} & si\ \tau_{i,j}^d\ existe \\ 1 & Sinon \end{cases}$$

Dans cette deuxième formule, d_j^d représente la désirabilité de prendre le voisin j pour aller à d. Cette désirabilité est calculée en se basant sur les deux valeurs $\tau_{i,j}^d$ et $bestCost_i^d$.

Par contre, une fourmi *Fant* déterministe ou bien une *Fant* probabiliste mais qui n'est pas censée explorer sélectionne son prochain saut comme suit :

$$n = \arg\max_{j \in N_i} d_j^d$$

et

$$Fant.accDelay + delayTab_i(n) < \Gamma$$

Une fourmi *Fant* probabiliste cesse de se déplacer lorsque la valeur de son champ *accCost* dépasse celle du champ *costLimit*. En contrepartie, une fourmi *Fant* déterministe cesse de se déplacer lorsque la valeur de son champ *accDelay* dépasse Γ. Si ces deux situations n'occurrent pas et *Fant* arrive au nœud actionneur de destination, alors elle se convertit en fourmi *Bant*. À ce moment, *Bant* retourne au nœud source sur la base de liste des nœuds visités de *Fant* tout en mettant à jour les tables de routage appropriées. Elle démarre à partir de ce nœud destinataire avec une valeur initiale de 0 pour les trois champs *accCost*, *localCost* et *accDelay*. Lorsque *Fant* arrive au niveau du nœud i après avoir été au niveau du nœud j, le nœud i met à jour les deux champs *accCost* et *localCost* comme suit :

$$accCost' = accCost + linkCost + extraCost$$

$$localCost' = linkCost$$

où $linkCost$ et $extraCost$ sont définis comme suit :

$$linkCost = power_{i,j}/P_{max}$$

$$extraCost = max(linkCost - localCost, 0)$$

Après ce calcul, le nœud i incrémente la valeur du champ *accDelay* et procède à la mise à jour des tables de routage comme l'illustre l'algorithme (4). Comme l'illustre cet algorithme, lorsque *Bant* est déterministe, chaque entrée dans chaque table *delayTab* et *bestCostTab* du nœud i qui correspond à son voisin j sont mises à jour respectivement, avec les deux valeurs des deux champs *accDelay*, *accCost* de *Bant*. Une certaine valeur de phéromone qui est inversement proportionnelle à la valeur du champ *accCost* est également ajoutée à $\tau_{i,j}^d$. En contrepartie, si *Bant* est non-déterministe alors l'entrée $delayTab_i(j)$ est mise à jour si la valeur du champ *accDelay* est plus petite. De la même façon, l'entrée $bestCost_i^d$ est également mise à jour si la valeur du champ *accCost* est aussi plus petite. Dans ce dernier cas, la valeur de l'entrée $\tau_{i,j}^d$ est mise à 1 (la valeur maximale) car, un meilleur chemin est trouvée. Cela permet d'orienter les futures *Fant* autour de ce nouveau chemin. Dernièrement et dans le cas où la valeur du champ *accCost* est supérieure à celle de l'entrée $bestCost_i^d$, une certaine valeur de phéromone est ajoutée à cette dernière entrée.

Algorithme 4 : Mise à jour des structures de données appropriées.

si *Bant.det=true* **alors**
  $delayTab_i(j) \leftarrow Bant.accDelay$;
  $bestCost_i^d \leftarrow Bant.accCost$;
  $\tau_{i,j}^d \leftarrow \tau_{i,j}^d + \frac{1}{Bant.accCost+\varepsilon}$;
sinon
  si $Bant.accDelay < delayTab_i(j)$ **alors**
    $delayTab_i(j) \leftarrow Bant.accDelay$;
  fin
  si $Bant.accCost < bestCost_i^d$ **alors**
    $bestCost_i^d \leftarrow Bant.accCost$;
    $\tau_{i,j}^d \leftarrow 1$;
  sinon
    $\tau_{i,j}^d \leftarrow \tau_{i,j}^d + \frac{bestCost_i^d+\varepsilon}{\beta(Bant.accCost+\varepsilon)}$;
  fin
fin
$\tau_{i,j}^d \leftarrow min(\tau_{i,j}^d, 1)$;

Comme nous avons déjà mentionné, toutes les entrées dans la table de phéromone sont diminuées à la fin de chaque itération afin d'oublier toute mauvaise solution. Cette mise à jour se fait comme suit :

$$\tau_{i,j}^d = (1 - DECAY\_FACTOR) \times \tau_{i,j}^d$$

Cette même traversée aller-retour qui correspond à une itération de recherche se répète *nbIteration* de fois. À la fin de ce nombre d'itérations, la meilleure route est prise et le nœud actionneur informe le nœud source par un message RM (Route Message). À ce moment, le nœud capteur source commence à utiliser ce nouveau chemin pour notifier le nœud actionneur de ses données.

3.6 Conclusion

Dans ce chapitre, nous avons présenté notre protocole CADER avec une explication détaillée de son fonctionnement. Cela a été fait en présentant tous les détails lui concernant à savoir, les principales structures de données utilisées, les messages échangés durant son fonctionnement et les étapes de son déroulement. Dans le chapitre suivant, nous allons procéder à la validation de notre proposition via simulation en présentant la méthodologie de simulation adoptée et aussi les résultats obtenus.

Chapitre 4

Validation

4.1 Introduction

Dans ce chapitre, nous présentons le processus de validation de notre protocole CADER avec les résultats obtenus. Cette validation est menée via simulation en utilisant le simulateur réseau J-Sim. Ce chapitre présente l'essentiel de cette validation en présentant dans premier temps, une brève description du simulateur J-Sim. Ensuite, nous passons vers une présentation des paramètres de simulation considérés et la méthodologie adoptée pour terminer à la fin avec une démonstration des résultats obtenus et aussi avec les discussions nécessaires.

4.2 Simulateur J-Sim

J-Sim est un outil logiciel de simulation des réseaux informatiques. Il présente une plateforme complète permettant le développement de nouveaux protocoles et l'évaluation de leurs performances. Il est construit sur la base d'une architecture logicielle à base de composants appelée ACA (Automatic Component Architecture). Chaque composant est doté de ports qui lui permet de communiquer avec d'autres composants. Les différents composants sont faiblement couplés. Cela favorise leur réutilisation dans d'autres applications.

Au-dessus de cette architecture ACA, J-Sim intègre un cadre applicatif (framework INET) qui implémente les principales entités pouvant constituer un réseau informatique et qui permettent l'implémentation de nouveaux protocoles.

J-Sim est doté également d'un autre framework qui est dédié spécialement à la simulation des réseaux de capteurs sans fil. Ce cadre définit trois types de nœuds :

1. Un nœud capteur,
2. Un nœud puits,
3. Et un nœud cible qui génère des événements. Par exemple, un véhicule mobile qui génère des vibration au sol, et donc, qui génère des évènement.

Mener des simulation avec J-Sim nécessite l'utilisation de deux langages informatiques distincts. Le premier est le langage Java qui permet le développement des composants qui implémentent nos propres conceptions de protocoles, nos propres modèles, etc. Le deuxième est le langage Tcl/Java qui permet la définition des scénarios de simulation souhaités et la connexion des différents composants déjà existants ou nouvellement développés. En fait, Tcl/Java propose de nouvelles commandes qui permettent la manipulation

direct, à partir d'un environnement Tcl, des objets Java comme l'instanciation d'un classe, l'invocation d'une méthode d'un objet donné, etc.

L'intégration d'un nouveau protocole dans J-Sim et son expérimentation se fait selon le processus suivant :

1. L'implémentation des détails du protocole proposé sous la forme d'un composant à l'aide du langage Java.
2. La définition des différents scénarios de simulation visés à l'aide du langage Tcl/Java. Lors de cette définition, il faut savoir connecter le composant qui implémente le protocole proposé avec les autres composants qui implémentent les autres protocoles dans les autres couches réseaux.
3. Le lancement des différentes simulations et la récupération des résultats.

En fait, c'est sur la base de ce processus que nous avons intégré notre protocole CADER et mener nos différentes expérimentations.

4.3 Méthodologie et paramètres de simulation

Table 4.1: Paramètres de simulation

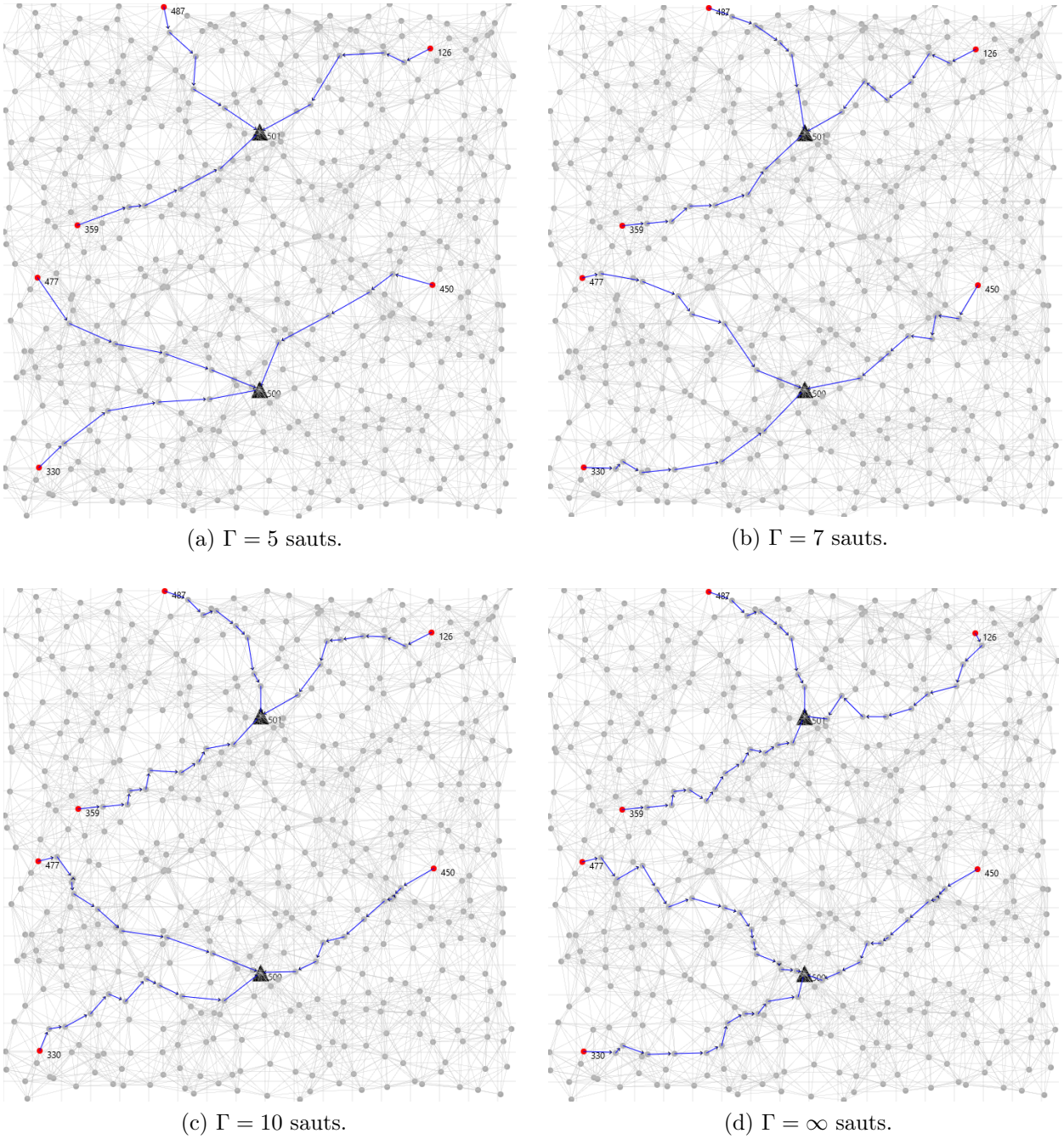
Paramètre	**Valeur**
Nombre de nœuds capteur	500
Nombre de nœuds actionnaires	2
Nombre de nœuds source	6
Densité	20
Portée maximale (m)	80
α	2
Nombre d'itérations de recherche	600
EXPLORE_LIMIT	3
DECAY_FACTOR	0.05

En utilisant le simulateur J-sim, nous avons effectué une série de simulations afin de présenter les performances de notre protocole CADER. Le tableau (4.1) illustre les principaux paramètres qui ont été utilisés pour mener nos différentes expérimentations. Les six premiers paramètres sont liés au réseau simulé alors que les trois restants sont propres à l'algorithme de colonie de fourmis. Comme l'illustre le tableau 4.1, nous avons considéré un réseau de 500 nœuds capteurs avec deux nœuds actionneurs et une densité [1] de 20. Chaque nœud peut communiquer sur une distance de 80 m. 6 nœuds capteurs sources répartis à différents endroits du réseau ont été considérés. La topologie réseau sur laquelle nous nous sommes basés a été générée avec une aire qui a été définie selon la relation suivante : $\sqrt{n\pi r_c^2/d}$ où n représente le nombre de nœuds capteurs, r_c correspond au rayon de communication et d correspond à la densité réseau. Le nombre d'itérations de recherche lors de l'application de l'algorithme de colonie de fourmis a été fixé à 600 itérations. La valeur de EXPLORE_LIMIT qui correspond au nombre d'explorations autorisées par

1. Le nombre moyen de voisins d'un nœud capteur.

chaque fourmi non-déterministe a été mise à 3. À la fin, l'intensité des valeurs de phéromone dans la table de phéromone de chaque nœud actionneur sont diminuées par un facteur de (1-DECAY_FACTOR) qui correspond à (1-0.05).

Plusieurs expérimentations ont été menées afin d'un côté, monter la capacité de notre solution à assurer un compromis entre l'assurance de latence donnée (Γ) et la minimisation de l'énergie totale qui est consommée par chaque route construite, et d'un autre côté, valider l'opération de notre application de l'algorithme de colonie de fourmis. Divers résultats ont été obtenus qui seront présentés, en particulier, sous forme d'instantanés et de courbes.



(a) $\Gamma = 5$ sauts.

(b) $\Gamma = 7$ sauts.

(c) $\Gamma = 10$ sauts.

(d) $\Gamma = \infty$ sauts.

FIGURE 4.1: Instantanés montrant les différentes routes finales construites avec différentes bornes temporelles (valeurs de Γ).

4.4 Résultats et discussion

Dans cette section, nous présentons les différents résultats obtenus par nos différentes expérimentations. En particulier, nous présentons :

- L'effet de la variation de la borne temporelle sur le comportement de notre protocole en présentant des instantanés des routes construites selon différentes valeurs de Γ et aussi, en présentant les différents coûts de ces routes,
- des résultats propres à l'application de l'algorithme de colonie de fourmis en présentant des instantanés qui montrent l'évolution du processus d'établissement d'une route donnée avec les différentes itérations de recherche et aussi, en présentant l'évolution du coût de cette meilleure route recherchée.

4.4.1 Effet de la borne temporelle

Dans un premier temps, nous validons visuellement l'effet de la variation de la borne temporelle Γ sur le comportement de notre protocole. Pour cela, nous avons mené quatre expérimentations différentes selon différentes valeurs de Γ : ∞, 10, 7 et 5 sauts. Pour chaque expérimentation qui correspond à une valeur donnée de Γ, nous avons généré un instantané qui illustre les routes construites des six nœuds sources considérés. Les figures (4.1a), (4.1b), (4.1c) et (4.1d) illustrent les instantanés obtenus à partir des expérimentations menées respectivement avec les valeurs 5, 7, 10 et ∞ de Γ.

À partir de ces quatre figures, deux principales remarques peuvent être tirées. La première est pertinente au processus de clusterisation. En fait, il est clair que chaque nœud capteur source rejoint le nœud actionneur le plus proche de lui. Par exemple, les trois nœuds sources 487, 126 et 359 sont associés avec le nœud actionneur 501 alors que les trois sources 450, 477 et 330 sont associés avec le nœud actionneur 500. Ceci est tout a fait logique car, chaque nœud capteur sélectionne durant la première étape du fonctionnement de notre protocole le nœud actionneur le plus proche de lui en termes de nombre de sauts comme cluster-head.

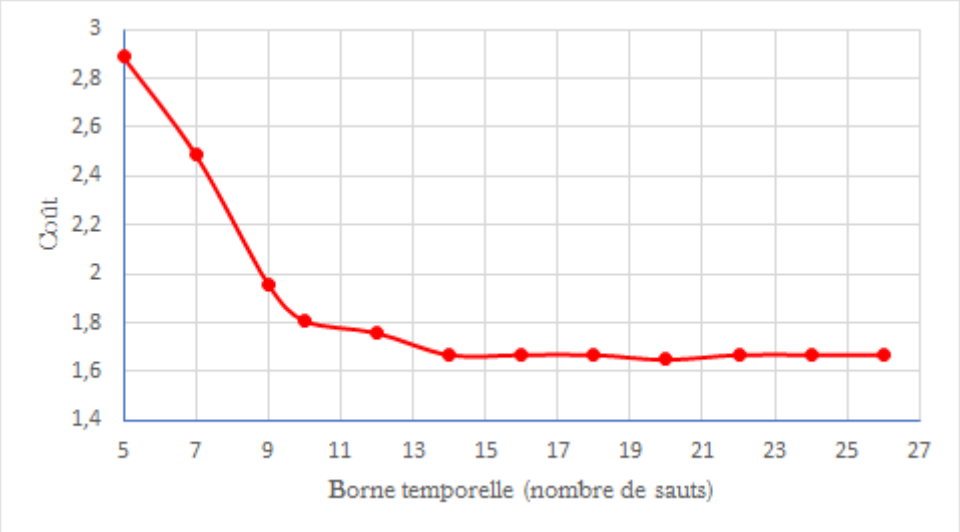
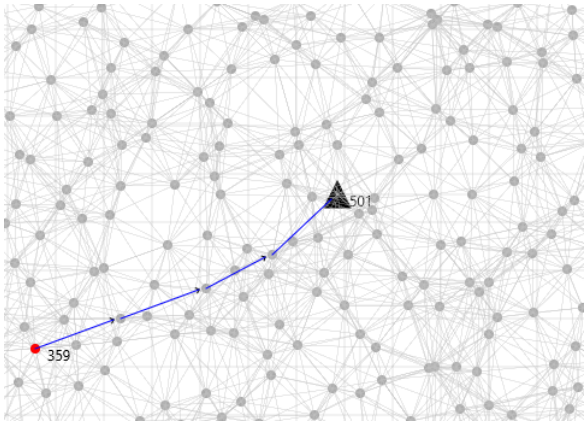
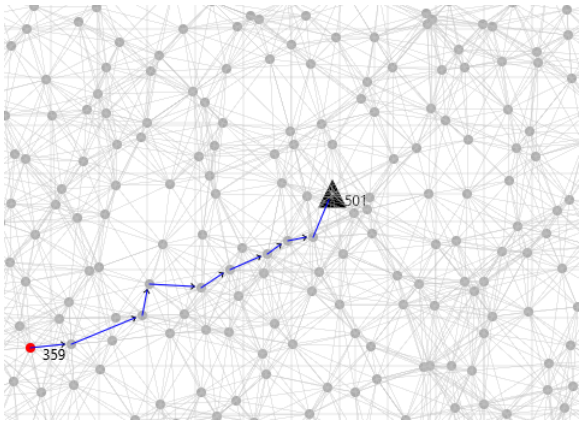


FIGURE 4.2: Coût total de la route finale construite pour le nœud source 359 en fonction de la borne temporelle Γ.

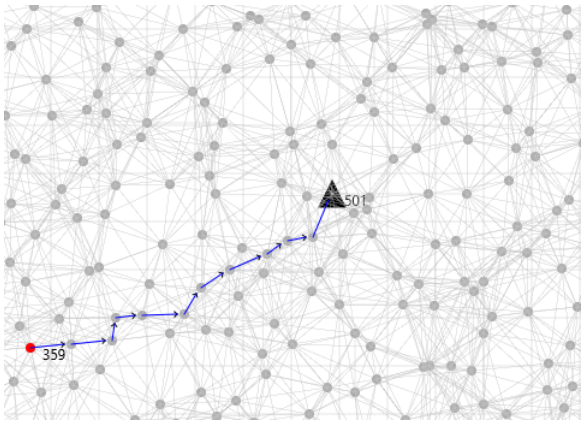
La deuxième remarque est liée à la capacité de notre protocole à assurer le compromis désiré entre la minimisation de la consommation énergétique et l'assurance de borne temporelle donnée. Comme l'illustre les quatre figures, le nombre de sauts des routes construites ne dépasse jamais la borne temporelle donnée. En addition, nous pouvons remarquer que lorsque la valeur de Γ augmente, le nombre moyen de sauts des routes établies augmente également. Par exemple, si nous prenons la figure (4.1a), aucune route des six nœuds capteurs sources a un nombre de sauts qui dépasse la valeur $\Gamma = 5$ alors qu'avec une valeur de $\Gamma = 7$ et comme l'illustre la figure (4.1b), ce nombre de sauts dépasse la valeur de 5 mais ne dépasse jamais la borne 7. Cette augmentation dans le nombre de sauts à chaque augmentation dans la valeur de Γ et comme l'illustre la figure (4.2) permet d'avoir à chaque fois des routes plus économes en énergie. Comme l'illustre toujours la figure (4.2), le coût des routes construites augmente à chaque augmentation dans la valeur de Γ. Ceci avec ces instantanés générés montrent clairement la capacité de notre protocole à assurer le compromis désiré.



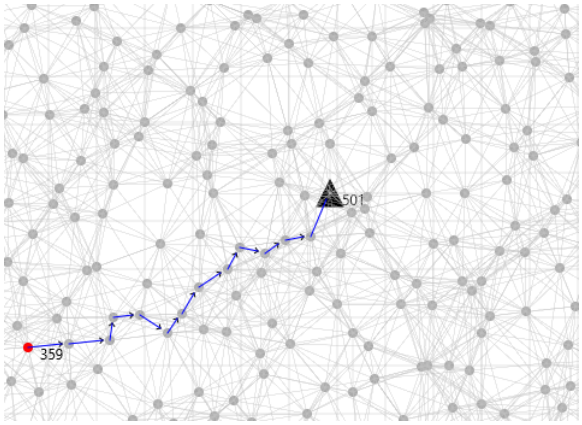
(a) L'itération 0.



(b) L'itération 20.



(c) L'itération 60.



(d) L'itération 200.

FIGURE 4.3: Instantanés montrant l'évolution de la route du nœud source 359 suivant les différentes itérations de la recherche des fourmis en considérant une valeur de Γ qui est égale à ∞.

4.4.2 Résultats additionnels propres à l'application de l'algorithme de colonie de fourmis

Dans cette section, nous présentons des résultats additionnels propres au processus de recherche qui est mené par les fourmis afin de trouver la route désirée. Pour cela, nous présentons dans un premier temps des instantanés qui illustrent la meilleure route trouvée à différentes itérations de recherche, à savoir : 0, 20, 60 et 200. Ces instantanés sont illustrés respectivement par les figures (4.3a), (4.3b), (4.3c) et (4.3d). Nous considérons uniquement les résultats du nœud source 359 et nous supposons une valeur de Γ qui est égale à ∞. Comme nous pouvons constater à partir de ces figures, la route établie initialement (4.3a) a quatre sauts. Ceci est tout a fait logique car, c'est le plus court chemin en termes de nombre de sauts qui est considéré initialement lorsque les tables de routage sont initialisées. À partir de cette itération, nous pouvons constater des routes avec un nombre de sauts qui augmente à chaque avancement dans le processus de recherche. Cette augmentation dans le nombre de sauts se traduit par une amélioration dans le coût du meilleur chemin trouvée. Ceci est illustré par la figure (4.4) qui montre le coût du meilleur chemin trouvé en fonction du nombre d'itérations de recherche. Au début, le coût du meilleur chemin est très important. Ce coût connaît une diminution agressive dans les 50 premières itérations. Cette diminution se continue à partir de l'itération 50 mais avec une cadence moins importante jusqu'à l'itération 200 à partir de laquelle ce coût se stabilise. Ce dernier coût correspond au coût du meilleur chemin émergeant qui connecte le nœud source 359 avec son nœud actionneur correspondant. Cette dernière courbe (4.4) avec les différents instantanés présentés dans cette section et aussi les résultats de la section précédente illustrent clairement la bonne application de l'algorithme de colonie de fourmis.

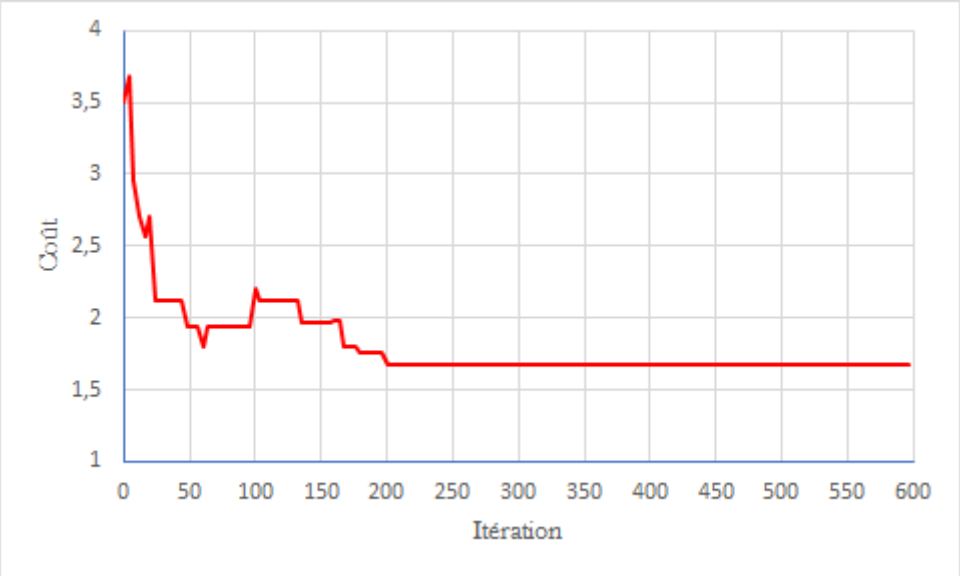


FIGURE 4.4: Coût de la route construite du nœud source 359 à différentes itérations de recherche avec une valeur de Γ qui est égale à ∞.

4.5 Conclusion

Dans ce chapitre, nous avons présenté l'essentiel de notre processus de simulation qui a été mené avec le simulateur J-Sim. Les paramètres de simulation sur lesquels nous nous sommes basés, la méthodologie de simulation adoptée et les résultats obtenus ont été tous présentés. Ces résultats obtenus ont montré la capacité de notre protocole à assurer le compromis désiré entre la minimisation de l'énergie consommée et l'arrivée avant l'échéance imposée. Ces résultats ont montré également la bonne application de l'algorithme de colonie de fourmis.

Conclusion générale

Dans ce mémoire, nous nous sommes intéressés aux réseaux de capteurs et d'actionneurs sans fil (WSANs). Ces réseaux sont constitués principalement de deux types de nœuds. Les premiers sont des nœuds capteurs qui s'occupent du prélèvement des mesures de notre environnement physique. Ils sont caractérisés par des ressources énergétiques, de calcul et de stockage limitées et sont déployés avec un grand nombre. Les deuxièmes sont des nœuds actionneurs qui agissent sur l'environnement sur la base de ce que les nœuds capteurs leur envoient. À la différence des nœuds capteurs, les nœuds actionneurs possèdent des ressources plus importantes.

Deux types de communication peuvent être distingués dans les WSANs : communication entre capteur et actionneur et communication entre actionneur et actionneur. Dans ce présent travail, nous nous sommes intéressés au premier type. L'objectif d'une telle communication entre capteur et actionneur consiste à pouvoir établir des routes économes en énergie à cause des capacités énergétiques limitées des nœuds capteurs, avec une latence qui ne dépasse pas l'échéance donnée. En fait, les données prélevées par les nœuds capteurs doivent arriver à temps pour que les nœuds actionneurs puissent réagir rapidement. Le problème posé avec ce routage est le fait que nous sommes en face de deux objectifs contradictoires. La conservation de l'énergie se fait au détriment d'un temps de latence plus important. Par conséquent, toute solution réseau qui est dédiée à ces réseaux doit tenir en compte de cela et assurer un compromis entre la conservation de l'énergie et le respect de la latence donnée. En fait, c'est à cette problématique que notre travail a répondu.

Pour pouvoir répondre à cette question, nous nous sommes basés sur les algorithmes de colonie de fourmis pour proposer notre propre protocole que nous avons baptisé CADER comme acronyme de *« Centralized Ant-based Delay-bounded and Energy-efficient Routing in wireless sensor and actor networks »*. À la différence de d'autres solutions qui existent dans la littérature, nous avons choisi d'appliquer ces algorithmes de manière centralisée. Deux motivations nous ont motivé pour aller dans cette direction. Premièrement, l'application répartie de ces algorithmes de colonie de fourmis et comme il est communément adoptée dans les travaux connexes, nécessite un grand nombre de messages de contrôle qui cause une consommation énergétique importante qui peut être supérieure à celle qu'on gagnera après l'émergence de la route désirée et son utilisation. Notre deuxième motivation est la présence de multiples nœuds actionneurs avec des ressources plus importantes auxquels on peut déléguer cette tâche de calcul de routes. Il faut uniquement assurer le moyen de mettre à la disposition de ces nœuds actionneurs, les informations des nœuds capteurs qui sont associés avec eux. Cette tâche qui est couverte par notre protocole et qui est bien décrite dans le chapitre contribution.

Pour valider notre protocole, nous nous sommes basés sur le simulateur J-sim en intégrant compétemment notre protocole dedans. Différentes expérimentations ont été menées qui nous ont donné des résultats positifs. Ces résultats ont été présentés sous forme de courbes et aussi sous forme d'instantanés.

En fait, notre protocole trouve uniquement des chemins avec des puissances de transmission totales minimales sans tenir en compte de l'énergie résiduelle des nœuds capteurs. Par conséquent, aucun équilibrage de la consommation de cette énergie a été considéré. Cela limite la durée de vie du réseau, chose qui n'est pas souhaitable. Pour cette raison et comme future travail, nous visons l'amélioration de notre protocole pour considérer cet équilibrage de cette consommation d'énergie.

Bibliographie

[1] Ke Li and Chien-Chung Shen. Balancing transmission power and hop count in ad hoc unicast routing with swarm intelligence. In *2008 IEEE Swarm Intelligence Symposium*, pages 1–8. IEEE, 2008.

[2] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Bradford Company, USA, 2004.

[3] Ahmed Sobeih, Jennifer C Hou, Lu-Chuan Kung, Ning Li, Honghai Zhang, Wei-Peng Chen, Hung-Ying Tyan, and Hyuk Lim. J-sim : a simulation and emulation environment for wireless sensor networks. *IEEE Wireless Communications*, 13(4) :104–119, 2006.

[4] Nouha Sghaier. *Techniques de conservation de l'énergie dans les réseaux de capteurs mobiles : découverte de voisinage et routage*. Theses, Université Paris-Est, 2013.

[5] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy conservation in wireless sensor networks : A survey. *Ad hoc networks*, 7(3) :537–568, 2009.

[6] Ian F. Akyildiz and Ismail H. Kasimoglu. Wireless sensor and actor networks : research challenges. *Ad Hoc Networks*, 2(4) :351–367, 2004.

[7] Falko Dressler. *Self-Organization in Sensor and Actor Networks*. Wiley, 2007.

[8] Djamila Bendouda. *Contrôle des réseaux de capteurs et actionneurs sans fil pour la supervision*. PhD thesis, 2018.

[9] Bilel Romdhani. *Exploitation de l'hétérogénéité des réseaux de capteurs et d'actionneurs dans la conception des protocoles d'auto-organisation et de routage*. Theses, INSA de Lyon, 2012.

[10] Hamidreza Salarian, Kwan-Wu Chin, and Fazel Naghdy. Coordination in wireless sensor–actuator networks : A survey. *Journal of Parallel and Distributed Computing*, 72(7) :856–867, 2012.

[11] Damien Roth. *Gestion de la mobilité dans les réseaux de capteurs sans fil*. PhD thesis, Université de Strasbourg, 2012.

[12] Carlos De Morais Cordeiro and Dharma Prakash Agrawal. *Ad hoc and sensor networks : theory and applications*. World Scientific Publishing Company, 2011.

[13] Alexandre Mouradian. *Proposition et vérification formelle de protocoles de communications temps-réel pour les réseaux de capteurs sans fil*. Theses, INSA de Lyon, 2013.

[14] Hermann Kopetz. *Real-time systems : design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[15] A Costanzo, T Luong, and V Marill. Optimisation par colonies de fourmis. *Thé Van MARILL Guillaume*, 19, 2006.

[16] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system : optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1) :29–41, 1996.

[17] Inès Alaya. *Optimisation multi-objectif par colonies de fourmis : cas des problèmes de sac à dos*. PhD thesis, Université Claude Bernard-Lyon I ; Université de la Manouba (Tunisie), 2009.

[18] Johann Dréo, Alain Pétrowski, Patrick Siarry, and Eric Taillard. *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.

[19] Muhammad Saleem, Gianni A Di Caro, and Muddassar Farooq. Swarm intelligence based routing protocol for wireless sensor networks : Survey and future directions. *Information Sciences*, 181(20) :4597–4624, 2011.

[20] Xuxun Liu. Routing protocols based on ant colony optimization in wireless sensor networks : a survey. *IEEE Access*, 5 :26303–26317, 2017.

[21] Tiago Camilo, Carlos Carreto, Jorge Sá Silva, and Fernando Boavida. An energy-efficient ant-based routing algorithm for wireless sensor networks. In *International workshop on ant colony optimization and swarm intelligence*, pages 49–59. Springer, 2006.

[22] Wenyu Cai, Xinyu Jin, Yu Zhang, Kangsheng Chen, and Rui Wang. Aco based qos routing algorithm for wireless sensor networks. In *International conference on ubiquitous intelligence and computing*, pages 419–428. Springer, 2006.

[23] Yao-feng Wen, Yu-quan Chen, and Min Pan. Adaptive ant-based routing in wireless sensor networks using energy* delay metrics. *Journal of Zhejiang University-SCIENCE A*, 9(4) :531–538, 2008.

[24] Hakki Bagci, Ibrahim Korpeoglu, and Adnan Yazıcı. A distributed fault-tolerant topology control algorithm for heterogeneous wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 26(4) :914–923, 2014.