Table des matières

Liste des Figures

Liste des tableaux

INTRODUCTION GENERALE

En suivant l'actualité technologique et électronique dans le domaine des Réseaux de capteurs sans fil, il s'agit de l'une des dix nouvelles technologies qui bouleverseront le monde et notre manière de vivre et de travailler. En effet, ces nouveaux types de réseaux viennent au secours de l'environnement et de l'industrie. Depuis quelques décennies, le besoin d'observer et de contrôler des phénomènes physiques tels que la température, la pression ou encore la luminosité est essentiel pour de nombreuses applications industrielles et scientifiques.

Bien que ce type de réseaux partage des similarités avec les concepts généraux des réseaux ad-hoc, l'ensemble de ses propres caractéristiques le rend différent des réseaux conventionnels. A la différence des réseaux mobiles ad hoc, les nœuds de détection sont plus susceptibles d'être à l'arrêt pour toute la période de leur vie. Même si les nœuds de capteurs sont fixes, la topologie du réseau peut changer. Pendant les périodes de faible activité, les nœuds peuvent passer à l'état inactif pour économiser l'énergie. Lorsque certains nœuds sont à court de puissance de la batterie et meurent, de nouveaux nœuds peuvent être ajoutés au réseau. Bien que tous les nœuds soient initialement équipés d'énergie égale, certains nœuds peuvent éprouver une plus grande activité à la suite de la région où ils se trouvent.

Une propriété importante des réseaux de capteurs est la nécessité que les capteurs diffusent les données du nœud collecteur ou de la station de base de manière fiable, dans un intervalle de temps qui permet à l'utilisateur ou une application de répondre à l'information en temps opportun, car si les informations mis à jour sont inutile cela peut conduire à des résultats désastreux.

Une autre caractéristique importante est l'extensibilité et la variation de la taille du réseau, la densité des nœuds et la topologie. Les réseaux de capteurs sont très denses par rapport aux réseaux ad hoc mobiles et filaires. Ceci provient du fait que la plage de détection est inférieure à la portée de communication et donc plusieurs nœuds sont nécessaires pour atteindre une couverture de détection suffisante. Les nœuds de capteurs doivent être résistants à des pannes et des attaques.

Le problème qui se pose dans le contexte des RCSF est l'adaptation de l'approche de routage utilisée avec le grand nombre de nœuds existants dans un environnement caractérisé par de

modestes capacités de calcul, des réserves d'énergie et de capacité mémoire limitées. Il semble donc important que toute conception de protocole de routage doit étudier les problèmes importants tels que la tolérance aux fautes , l'utilisation optimale des ressources des nœuds, le passage à l'échelle et l'assurance d'une bonne qualité de service [21].

L'objectif du travail en cours est d'évaluer la performance du protocole de routage Directed Diffusion en termes de diffusion de données à travers des études de simulation. Ensuite, ce travail permettra de comparer le protocole de diffusion dirigée avec un autre protocole traditionnel de diffusion de données, nommé : Flooding ou inondations en termes d'énergie. Ce travail a été utilisé pour comparer ces deux protocoles de diffusion des données sur les paramètres suivants.

- Overhead de communication
- Consommation d'énergie

Notre étude s'étale sur trois chapitres, un premier chapitre qui présente d'une manière brève les réseaux de capteurs et leurs différents domaines d'application ainsi que les protocoles de routages.

Un second chapitre qui détaille le fonctionnement du protocole de routage Directed Diffusion ainsi que le Protocole Flooding.

Un dernier chapitre qui montre notre étude et implémentation du protocole Directed diffusion et l'évaluation et comparaison de ce dernier avec le Floofing. Ce dernier chapitre explique les deux métriques d'évaluation et les outils utilisés pour l'environnement d'exécution et de compilation des programmes.

Enfin on terminera avec une Conclusion générale et une Bibliographie.

CHAPITRE I

ROUTAGE DANS LES RESEAUX DE CAPTEURS SANS FIL (RCSF)

Sommaire

1. INTRODUCTION

La technologie des réseaux de capteurs sans fil est un domaine en plein essor, de plus en plus d'applications utilisent cette technologie, les avancées électroniques et informatiques d'aujourd'hui sont capables de développer de minuscules capteurs capables de capter des données, calculer des informations à l'aide de ces données collectés et de communiquer à travers un réseau.

Nous aborderons dans ce chapitre une brève introduction aux Réseaux de Capteurs sans fil, nous citerons les différentes topologies ainsi que la communication entre les nœuds, dit principalement routage de données, une description des protocoles de routages utilisés dans ce domaine nous éclaira sur la communication dans les RCSF.

2. ARCHITECTURE ET MODELE D'UN NŒUD CAPTEUR

Un capteur est un dispositif qui perçoit une propriété physique et qui mappe la valeur à une mesure quantitative [1]. Un nœud capteur est composé principalement d'un processeur, une mémoire, un émetteur/récepteur radio, un ensemble de capteurs (capturer les grandeurs physiques tel que : température), et une pile (batterie), il est parfaitement autonome et il représente la notion de base dans un réseau de capteurs sans fil. Cependant ses ressources sont relativement faibles. La figure suivante illustre son architecture.

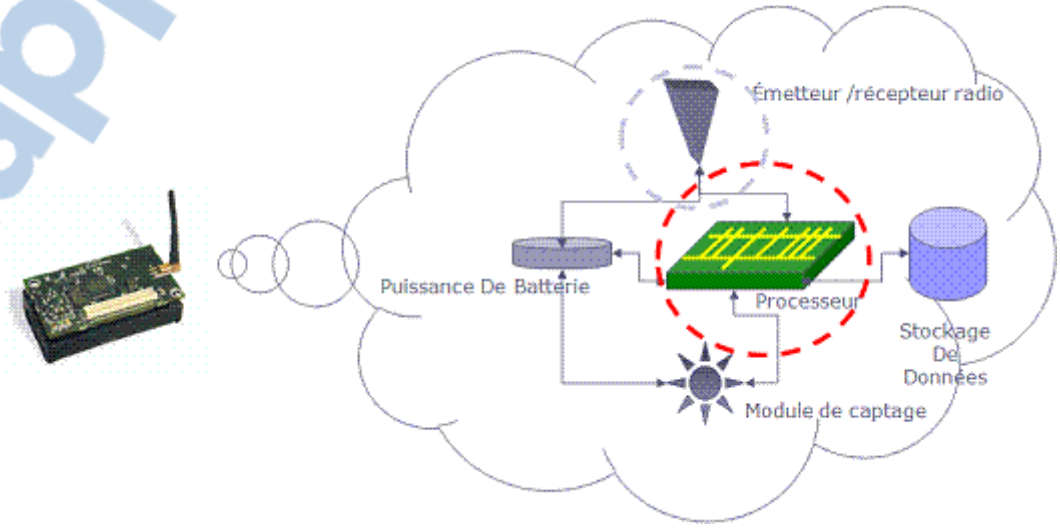


Figure 1. 1 : Anatomie d'un Nœud Capteur

Il est important de citer les différents types de nœuds capteurs qui existent et qu'on peut trouver dans les RCSF.

- **Un nœud régulier** est un nœud doté d'une unité de transmission et d'une unité de traitement de données. L'unité de transmission de données est responsable de toutes les émissions et réceptions de données via un support de communication sans fil. [2]

- Un **nœud capteur** ou nœud source est un nœud régulier équipé d'une unité d'acquisition ou de détection. L'unité d'acquisition est généralement dotée d'un capteur ou plusieurs capteurs qui obtiennent des mesures analogiques (physiques et physiologiques) et d'un convertisseur Analogique/Numérique qui convertit l'information relevée en un signal numérique compréhensible par l'unité de traitement. [2]
- Un **nœud actionneur ou robot** est un nœud régulier doté d'une unité lui permettant d'exécuter certaines tâches spécifiques comme des tâches mécaniques (se déplacer, combattre un incendie, piloter un automate, etc.) [2]
- Un **nœud puits** est un nœud régulier doté d'un convertisseur série connecté à une seconde unité de communication (GPRS, Wi-Fi, WiMax, etc.). La seconde unité de communication fournit une retransmission transparente des données provenant de nœuds capteurs à un utilisateur final ou d'autres réseaux comme internet. [2]
- Un **nœud passerelle (ou gateway)** est un nœud régulier permettant de relayer le trafic dans le réseau sur le même canal de communication. [2]

3. RESEAUX DE CAPTEURS SANS FIL

Un RCSF est composé d'un ensemble de nœuds capteurs, limités en capacité mémoire et de calcul, devant être économes en énergie, ce qui les contraint à exploiter une faible puissance de transmission et des portées et des débits modestes. Ces nœuds capteurs sont organisés en champs. Chacun de ces nœuds est autonome et a la capacité de collecter des données et de les transférer au nœud passerelle (dit "sink" en anglais ou puits) par l'intermédiaire d'une architecture multi-sauts. Le puits transmet ensuite ces données par Internet ou par satellite à un ordinateur central «Gestionnaire de taches» pour analyser et évaluer ces données et prendre des décisions.

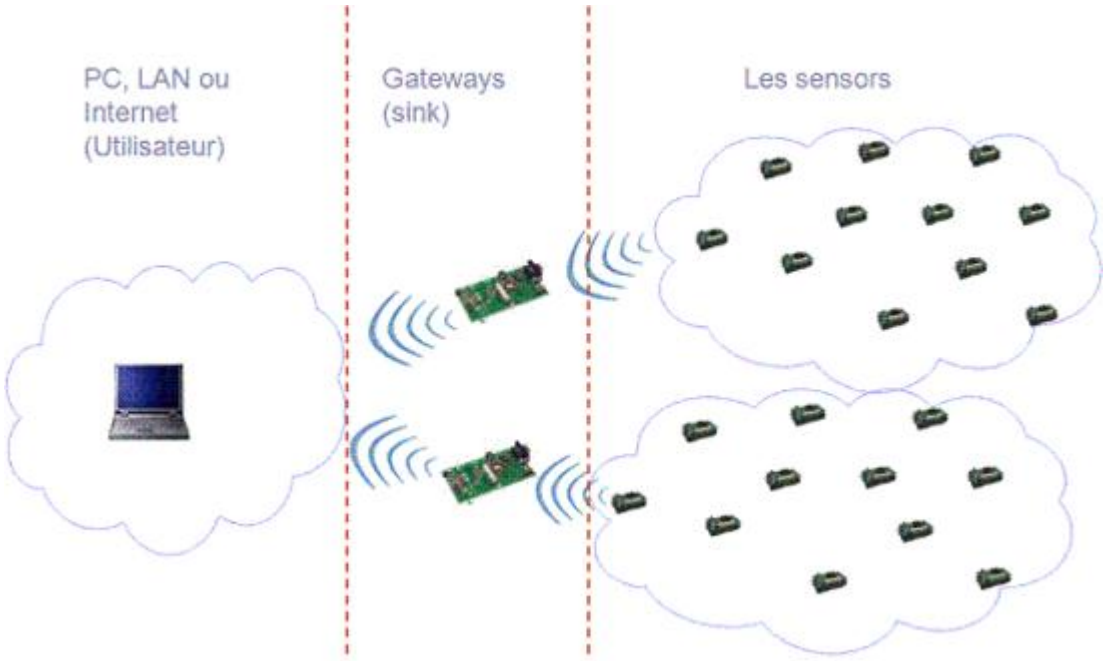


Figure 1. 2 : Architecture d'un réseau de capteurs sans fil

Un ensemble de métriques permet de déterminer le design d'un réseau de capteurs. Ces facteurs influencent sur l'architecture des réseaux de capteurs et le choix des protocoles à implémenter [4]:

La figure 1.3 illustre les facteurs influençant les RCSF.

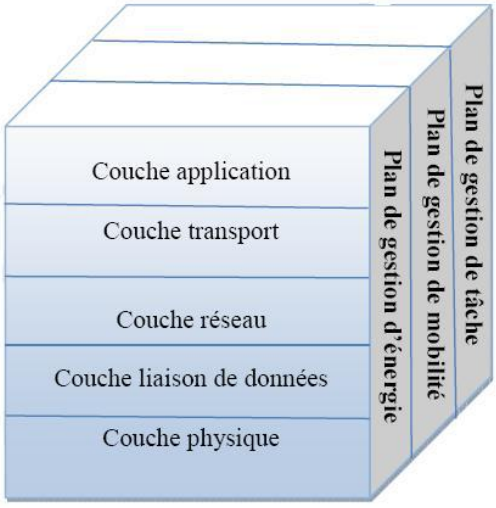


Figure 1. 3 : Pile protocolaire dans les réseaux de capteurs sans fil [4]

4. DOMAINES D'APPLICATION DES RESEAUX DE CAPTEURS SANS FIL

Le champ d'applications des réseaux de capteurs est de plus en plus élargi grâce aux évolutions techniques que connaissent les domaines de l'électronique et des télécommunications. Parmi ces évolutions, on peut citer la diminution de taille et du coût des capteurs, ainsi que l'élargissement des gammes de capteurs disponibles (mouvement, températures, ...) et l'évolution des supports de communication sans fil. En effet, les applications des réseaux de capteurs peuvent être militaires, médicales, environnementales, commerciales, etc.

- **Applications militaires**

Un réseau de capteurs déployé dans un secteur stratégique ou difficile d'accès, permet par exemple d'y surveiller tous les mouvements (alliés ou ennemis), ou d'analyser le champ de bataille avant d'y envoyer du renfort.

- **Applications médicales**

Il existe déjà dans le monde médical, des gélules multi-capteurs pouvant être avalées qui permettent, sans avoir recours à la chirurgie, de transmettre des images de l'intérieur du corps humain.

- **Applications environnementales**

Des capteurs de températures peuvent être dispersés à partir d'avions dans le but de détecter d'éventuels problèmes environnementaux dans le domaine couvert par les capteurs dans une optique d'intervenir à temps afin d'empêcher que d'éventuels incendie, inondation, volcan ou tsunami ne se produisent.

- **Applications commerciales**

Des nœuds capteurs peuvent être utilisés pour améliorer les processus de stockage et de livraison. Le réseau peut ainsi être utilisé pour connaitre la position, l'état et la direction d'une marchandise. Un client attendant une marchandise peut alors avoir un avis de livraison en temps réel et connaitre la position des marchandises qu'il a commandées.

- **Applications de traçabilité et de localisation**

  Suite à une avalanche il est nécessaire de localiser les victimes enterrées sous la neige en équipant les personnes susceptibles de se trouver dans des zones à risque par des capteurs.

Ainsi, les équipes de sauvetage peuvent localiser plus facilement les victimes. Contrairement aux solutions de traçabilité et de localisation basées sur le système de GPS (Global Positionning System), les réseaux de capteurs peuvent être très utiles dans des endroits clos comme les mines par exemple. [5]

5. TOPOLOGIES DE ROUTAGE DES RESEAUX DE CAPTEURS SANS FIL

La topologie détermine l'organisation des nœuds capteurs dans le réseau. Il existe deux principales topologies dans les protocoles de routage pour les RCSF.

- **Topologie plate** : dans une topologie plate, tous les nœuds possèdent le même rôle. Les nœuds sont semblables en termes de ressources.
- **Topologie hiérarchique** : afin d'augmenter la scalabilité du système, les topologies hiérarchiques ont été introduites en divisant les nœuds en plusieurs niveaux de responsabilité. L'une des méthodes les plus employées est le clustering, où le réseau est partitionné en groupes appelés "clusters". Un cluster est constitué d'un chef (cluster-head) et de ses membres.

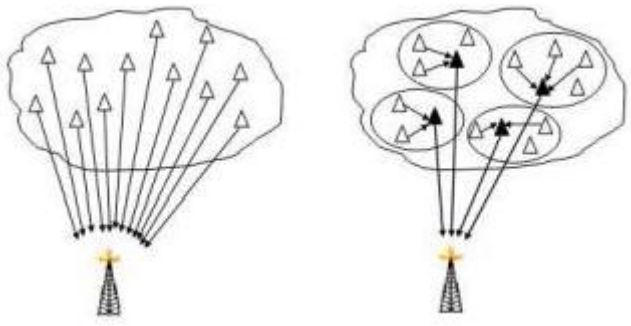


Figure 1. 4 : **Topologie plate et clustérisé à gauche d'un RCSF [22]**

Après le déploiement des nœuds capteurs sur une certaine zone de captage, ceux-ci commencent par la découverte de leurs voisins afin de construire la

topologie de communication. Ainsi, ils deviennent capables d'accomplir les tâches que leur sont affectées.

Selon une communication multi-sauts, les capteurs sont chargés de collecter des données, les router vers un nœud particulier appelé nœud puits. Ce dernier analyse ces données et transmet à son tour l'information collectée à l'utilisateur via internet ou bien satellite. [5]

6. PROTOCOLES DE ROUTAGE DES RESEAUX DE CAPTEURS SANS FIL

Dans ce qui suit, nous citons un ensemble de protocoles de routage répandus dans les applications des réseaux de capteurs sans fil. Nous donnons un bref aperçu sur le principe de chacun :

- **SMECN :** Small Minimum Energy Communication Network est un protocole qui crée un sous graphe du réseau contenant un chemin avec le minimum d'énergie consommée [14] ;

- **Protocole LEACH :** Low-Energy Adaptive Clustering Hierarchy : forme des clusters pour minimiser la dissipation d'énergie [15] ;

- **Protocole SAR :** Sequential Assignment Routing : crée plusieurs arbres dont la racine de chacun est un nœud voisin du collecteur puis sélectionne un arbre pour le routage de données selon les ressources d'énergie et d'autres métriques de QoS [16] ;

- **Protocole Flooding** : dans ce protocole, chaque nœud diffuse la donnée à tous ses voisins sans tenir en compte s'ils l'ont déjà reçue ou non. Il est évident que ce protocole présente un ensemble d'inconvénient :

    - *Implosion* : le même message est dupliqué plusieurs fois ; chaque nœud reçoit autant de fois la même donnée que le nombre de ses voisins ;
    - *Chevauchement* : si deux nœuds observent le phénomène dans la même région, la même information sera envoyée deux fois ;

- Il utilise aveuglement les ressources disponibles sans tenir en compte de leur quantité.

- **Protocole Gossiping** : dans ce protocole, chaque nœud choisit aléatoirement un sous ensemble de ses voisins et leur envoie les données. L'implosion n'est plus un inconvénient pour ce protocole. Cependant, l'envoi d'un message à tous les nœuds demande plus de temps [17].

- **Protocole SPIN** : Sensors Protocols for Information via Negotiation utilise trois types de message ADV, REQ et DATA pour communiquer entre les nœuds. Puis, il n'envoie la donnée que s'il y a des nœuds intéressés [18].

- **Protocole Directed Diffusion** : diffuse l'intérêt puis détermine un chemin gradient pour la dissémination des données vers les nœuds intéressés.

En raison du thème de notre master, nous avons choisi d'étudier le protocole de diffusion dirigée plus en détail, car le but est d'évaluer ce dernier en le comparant au protocole Flooding.

7. CONCLUSION

Nous avons parcouru les différentes notions de base dans les réseaux de capteurs sans fil ainsi que les protocoles de routages utilisés dans ces derniers, le prochain chapitre détaillera le fonctionnement du protocole « Directed Diffusion » qui est l'un des protocoles de communications les plus important dans ce domaine et le protocole « Flooding », il nous sera plus évident de comparer et d'évaluer les deux protocoles par la suite.

CHAPITRE II

PRINCIPE DES PROTOCOLE DE ROUTAGE DIRECTED DIFFUSION ET FLOODING DANS LES RESEAUX DE CAPTEURS SANS FIL(RCSF)

Sommaire

1. INTRODUCTION

Après avoir défini un réseau de capteurs, on abordera dans ce chapitre la communication entre les nœuds de ce réseau, en effet il existe plusieurs protocoles de routages avec chacun sa spécificité.

Le principe de « Flooding » dit inondation est très présent dans le routage, il consiste à diffuser des messages à tous les voisins d'un nœud et cela est très monopolisant point de vue consommation d'énergie dans le réseau. C'est pour cela que l'amélioration des topologies réseaux et des protocoles de routages afin de minimiser la notion de Flooding est primordial dans ce domaine.

Le protocole de routage nommé « Directed Diffusion » est un protocole dont le principe est de trouver le meilleur chemin pour diriger les données d'un nœud source vers le nœud « sink » ou station de base. C'est ce que nous verrons par la suite dans un exemple concret.

2. FONCTIONNEMENT DU PROTOCOLE « DIRECTED DIFFUSION » [4]

Directed Diffusion (DD) fut l'un des premiers protocoles centrés-données. Il a été proposé par C.Intanagonwiwat, R.Govindan et D.Estrin [6] et est devenu l'un des protocoles les plus répandus dans les réseaux de capteurs. Sa création représente une importante avancée dans le domaine du routage. Il a été une base pour la conception de plusieurs protocoles de routage pour les réseaux de capteurs. [4]

Il n'y a pas meilleur manière de comprendre le fonctionnement de la communication entre les nœuds capteurs que par un exemple, ce dernier définit les différentes étapes d'établissement de chemin de routage de ce protocole.

- **Scénario** : Dans ce qui suit, nous prenons comme exemple, un réseau de capteurs connu pour détecter des cibles dans une région donnée. L'utilisateur de ce réseau le charge d'effectuer la tâche suivante : "Fournir, durant les T prochaines secondes et toutes les 10 ms, la position de n'importe quel véhicule roulant se trouvant dans la sous-règion R du champ de captage".

Nous détaillerons, ci-après, les étapes du fonctionnement du protocole Directed Diffusion pour la dissémination des données.

a. Dissémination des intérêts et établissement des gradients

Le collecteur (nœud sink) commence par envoyer périodiquement un message d'intérêt.

Cette tâche est définie par un ensemble de paires "attributs-valeurs". Une description simplifiée de l'exemple précédent pourrait être de la forme suivante :

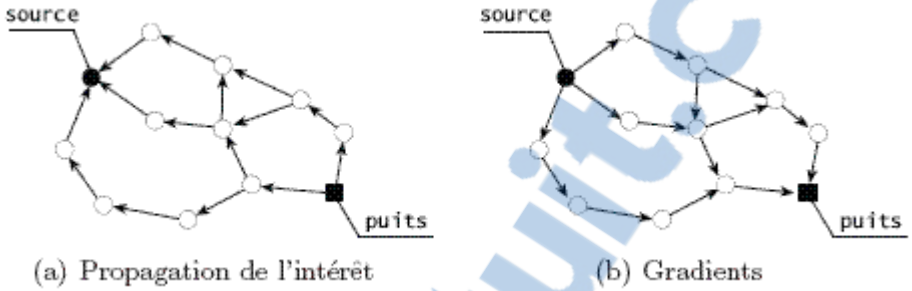
```
type = véhicule roulant        // le type de cible à détecter
intervalle = 10 ms             // envoyer des événements toutes les 10 ms
durée = 10 minutes             // durant les prochaines 10 minutes
rect = [0,100,200,400]         // par les noeuds de cette région
```

Le message ainsi décrit permet de spécifier les données par lesquelles l'utilisateur est intéressé. Pour cette raison, il est appelé intérêt. Il peut contenir différents attributs. L'exemple précédent spécifie : le type de cible détectée, le débit (l'intervalle) avec lequel les réponses doivent être envoyées, la durée de validité de l'intérêt, et la sous-région de captage concernée par l'intérêt.

Au départ, l'intérêt est diffusé par le nœud collecteur à tous ses voisins. Le débit demandé initialement par l'intérêt envoyé est plus faible que celui demandé par l'utilisateur *(Par exemple : intervalle = 1s, au lieu de : intervalle = 10ms).*

Chaque nœud du réseau maintient à son niveau un cache qui a pour but de garder trace des intérêts reçus. Chaque entrée du cache correspond à un intérêt distinct et contient plusieurs champs de gradients qui identifient les voisins depuis lesquels l'intérêt a été reçu. Un gradient spécifie une valeur et une direction dans laquelle les données répondants à l'intérêt seront envoyées. Le choix de la sémantique de la valeur associée au gradient dépend de l'application du réseau de capteurs. Dans l'exemple cité précédemment, cette valeur représente le débit avec lequel les données seront envoyées. Dans d'autres applications, cette valeur peut, par exemple, être une probabilité p avec laquelle une donnée est envoyée vers le voisin.

Chaque gradient possède une certaine durée de vie qui représente la durée de vie de l'intérêt associé. Quand un gradient expire, il est retiré de l'entrée d'intérêt. Quand tous les gradients d'une entrée d'intérêt expirent, l'entrée elle-même est supprimée du cache.



(a) Propagation de l'intérêt (b) Gradients

Figure 2. 1 : Propagation des intérêts et établissement des gradients

Le nœud source (NS) détecte un évènement.

Quand un nœud reçoit un intérêt, il vérifie son cache d'intérêts. Si aucune entrée correspondante à l'intérêt reçu n'existe (il est différent des intérêts du cache), le nœud crée une nouvelle entrée dans son cache, dans laquelle il va définir un gradient vers le voisin à partir duquel l'intérêt a été reçu. S'il existe une entrée correspondante, mais aucun gradient pour l'émetteur de l'intérêt, le nœud ajoute un gradient vers le voisin émetteur. Finalement, s'il existe déjà une entrée et un gradient pour l'émetteur, le nœud les met simplement à jour.

Par la suite, chaque nœud ayant reçu un intérêt diffuse ce dernier à tous ses voisins. Bien que l'intérêt soit initialement généré par le nœud collecteur, chaque nœud émetteur semble être l'origine de cet intérêt auprès de ses voisins récepteurs.

De cette manière, l'intérêt est diffusé dans tout le réseau par la technique d'inondation. Cette technique peut causer une forte dépense en énergie. Cependant, l'inondation de tout le réseau est inévitable en l'absence d'informations sur les nœuds capteurs susceptibles de satisfaire l'intérêt.

Ainsi, un intérêt est disséminé à partir du nœud collecteur sur tout le réseau. Cette dissémination installe des gradients dans chaque nœud du réseau pour orienter les

données de réponses à l'intérêt inondé, vers le nœud collecteur ; telle que montre la figure 2.3.

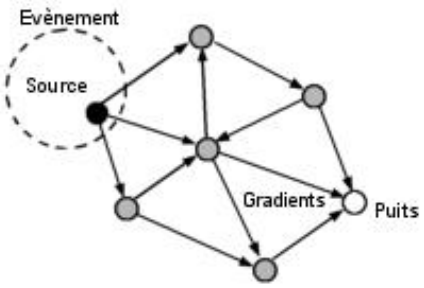


Figure 2. 2 : **Etablissement des gradients [4]**

Notons que le collecteur rediffuse périodiquement chaque intérêt. Ceci est nécessaire pour rétablir les gradients en cas de changements topologiques dans le réseau (défaillances de liens, mobilité, pannes de nœuds, etc.). Le taux de rafraîchissement d'intérêt est un paramètre de conception du protocole soumis à un compromis entre la charge du réseau et la fiabilité de transmission.

b. Propagation des données

Quand un nœud capte un évènement dans son champ de déploiement, il consulte son cache d'intérêts pour rechercher un enregistrement correspondant aux données captées. Si l'intérêt est trouvé, le nœud (qui devient désormais source de données pour cet intérêt) commence à envoyer l'évènement capté à tous ses voisins (constitués par les gradients établis dans la phase d'inondation d'intérêt) avec un faible débit appelé débit exploratoire. Suivant l'intérêt donné dans l'exemple adopté, l'évènement suivant forme une réponse pour le nœud collecteur :

```
type = véhicule roulant        // type du véhicule détecté.
instance = camion               // instance de ce type.
position = [125, 220]          // position du noeud.
intensité = 0.6                 // amplitude de signal capté.
confiance = 0.85                // degrés de confiance de la correspondance.
date = 01 :20 :40               // heure de génération de la donnée.
```

Au départ, les données sont considérées comme exploratoires, et sont envoyées à tous les voisins, pour lesquels un gradient a été établi, avec un faible débit (dans l'exemple adopté, le débit est égal à 1 donnée par seconde). Ces données exploratoires voyagent vers le nœud collecteur dans le sens inverse de celui de la propagation des intérêts. Elles sont prévues pour la construction et la réparation des chemins qui seront adoptés par la suite.

Un nœud intermédiaire qui vient de recevoir un message de donnée exploratoire d'un nœud voisin, cherche dans son cache d'intérêts une entrée correspondante à l'évènement capté. Si aucune entrée n'est trouvée, le message de données est écarté. Sinon, le nœud vérifie alors son cache de données (associé à l'entrée de l'intérêt). Ce dernier garde trace des données récemment envoyées et permet, entre autres, l'empêchement des boucles. Si le message reçu correspond à une entrée du cache de données, il est abandonné car il a déjà été envoyé. Autrement, le message est inséré dans le cache puis diffusé aux voisins pour lesquels un gradient a été établi.

Ainsi, chaque nœud intermédiaire exécute le même mécanisme en diffusant les données exploratoires vers tous ses voisins afin d'atteindre le nœud collecteur.

Cette étape est caractérisée par la propagation des données exploratoires (appelée aussi phase d'exploration).

c. Renforcement Positif

La phase de propagation des données exploratoires a pour but d'explorer les chemins existants entre la source et le collecteur. Le renforcement positif sert à choisir une route (selon un certain critère) parmi celles découvertes, afin d'obtenir les données à haut débit appelées données renforcées. Pour cela, quand les données exploratoires atteignent le nœud collecteur, ce dernier choisit l'un de ses voisins appropriés et lui envoie un message de renforcement positif. Dans l'exemple adopté, un message de renforcement positif est identique à l'intérêt initial, mais le débit des données demandées est plus élevé (la valeur de l'attribut intervalle est plus petite. Exemple : intervalle = 10ms). Un nœud intermédiaire qui reçoit ce message de renforcement, renforce son gradient vers l'émetteur. Dans l'exemple, un gradient est considéré comme étant renforcé si sa valeur (c'est-à-dire le débit de données demandé)

est supérieure à 1 donnée / sec. Par la suite, le nœud doit, à son tour, envoyer le message de renforcement à l'un de ses voisins (en suivant le même critère de sélection). De cette manière, l'un des chemins explorés est récursivement renforcé (figure 2.3).

Pour le choix du chemin à renforcer, le nœud collecteur, et par la suite les nœuds intermédiaires, appliquent localement la règle de renforcement positif. Grâce au cache de données, un nœud choisit le premier voisin à partir duquel il a reçu une donnée exploratoire correspondant à l'intérêt. Par conséquent, un chemin ayant la plus faible latence est établi entre la source et le collecteur. D'autres règles peuvent être appliquées. Par exemple, un nœud peut choisir le voisin à partir duquel il a reçu le plus grand nombre de messages de données. Ainsi, le chemin le plus fiable est élu.

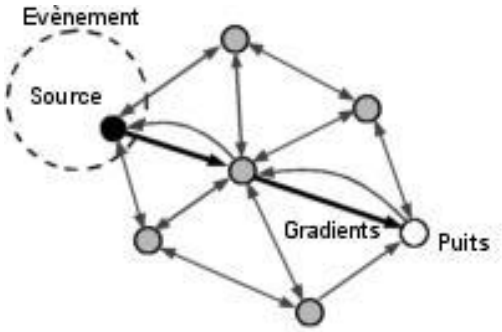


Figure 2. 3 : Renforcement d'un chemin. [4]

Une fois qu'un chemin reliant le nœud source au nœud collecteur soit renforcé, les données générées par la source sont envoyées avec un débit plus élevé à travers ce chemin. Un nœud qui vient de recevoir une donnée renforcée enverra celle-ci uniquement aux voisins pour lesquelles il possède un gradient renforcé.

Cependant, même en présence d'un chemin renforcé, le nœud source rediffuse périodiquement des données exploratoires dans le but de renforcer ultérieurement un meilleur chemin vers le nœud collecteur. Dans notre exemple, les données exploratoires sont envoyées chaque seconde.

Remarque : Il existe une autre étape qui est le Renforcement négatif, cette étape n'est pas définit car elle parle des chemins défaillants et de pannes, et cela n'est pas utile à détailler pour ce thème.

3. FONCTIONNEMENT DU PROTOCOLE « FLOODING »[3]

Dans le protocole Flooding [3] (appelé aussi l'inondation), chaque nœud reçoit les messages (sous forme d'un paquet de donnée), ensuite il le diffuse dans le réseau. Ainsi ce protocole consiste à transmettre tous les nouveaux paquets reçus et qui ne lui sont pas destinés.

Ce protocole n'a nul besoin ni de maintenir une table de routage, ni de découvrir son voisinage et maintenir une topologie bien précise. Par contre, ce protocole présente deux inconvénients majeurs qui sont le problème de duplication des paquets (le problème d'implosion) et le problème d'overlap. En effet, le problème d'implosion est illustré par la figure 2.4, les deux nœuds B et C reçoivent le même paquet du nœud A, ensuite ces mêmes nœuds vont diffuser le même paquet au nœud D, d'où ce derniers reçoit deux copies de même paquet. Ainsi on ne peut plus distinguer entre les paquets récents et les anciens paquets.

D'autre part, le problème d'overlap est illustré par la figure 2.5, il se produit lorsque deux nœuds observent la même région puis ils diffusent la même information vers d'autres nœuds. La faiblesse de ce protocole est qu'il est « aveugle » en termes de consommation d'énergie. En effet ce protocole autorise une forte circulation de données et une grande consommation en terme énergie, ce qui engendre une diminution importante de sa durée de vie.

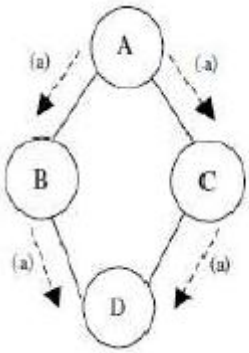


Figure 2. 4 : **le problème d'implosion**

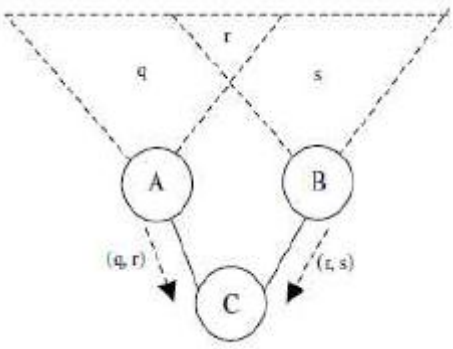


Figure 2. 5 : **le problème d'overlap**

4. CONCLUSION

Dans le protocole « DD » au départ, le nœud collecteur définit, en utilisant une liste de paires attribut-valeurs, un message qui exprime les données par lesquelles il est intéressé. Ce message, appelé intérêt, est diffusé à travers le réseau pour trouver les potentielles sources de données. Cette dissémination installe des gradients dans le réseau pour orienter les données provenant des sources vers le collecteur. Un gradient est un lien de réponse à un voisin à partir duquel l'intérêt a été reçu, il indique la direction que devront suivre les données envoyées par les capteurs. Il est caractérisé par une valeur et une durée de vie dérivées des champs de l'intérêt reçu. Quand un capteur capte des données qui répondent à un intérêt reçu, il envoie à tous les voisins pour lesquels un gradient existe un message de donnée considérée comme exploratoire. Lorsqu'un nœud intermédiaire reçoit une nouvelle donnée exploratoire, il l'insère dans le cache de données et la renvoie à tous les voisins pour lesquels un gradient existe. Quand le nœud collecteur reçoit une nouvelle donnée exploratoire, il envoie un message de renforcement positif à son émetteur. Ce dernier utilise le cache de données pour renvoyer le message de renforcement au premier voisin ayant fourni la donnée exploratoire. Chaque nœud intermédiaire réitère la même opération après réception d'un message de renforcement et ceci jusqu'à atteindre la source. Ainsi le chemin le plus rapide pour fournir les données est renforcé entre la source et le collecteur. Les données captées seront, par la suite, envoyées à travers ce chemin renforcé.

Le protocole « Directed Diffusion » permet de diminuer le nombre de messages circulant dans le réseau comparé au protocole flooding. Ce type de protocole se diffère du flooding vu qu'il utilise la notion de routage par saut (multihop path routing).

CHAPITRE III

EVALUATION ET COMPARAISON DES PROTOCOLES DIRECTED DIFFUSION ET FLOODING DANS LES RCSF

Sommaire

1. INTRODUCTION

Vu l'intérêt mondial que les différents domaines d'applications portent à la technologie des réseaux de capteurs, il est impératif d'évaluer les protocoles de ce type d'architecture d'un point de vu consommation d'énergie, délais de propagation des données, ou résistance au pannes,…etc. En effet le déploiement des réseaux de capteurs dans des zones non surveillées et dont l'intervention humaine est minime ou quasiment impossible, exige des capteurs un effort important pour survivre, et pour cause, les ressources des nœuds de capteurs sont assez limitées comme nous l'avons vu précédemment, la fiabilité des communications et le fonctionnement autonome des nœuds sont aussi des facteurs de risques, le réseau peut être mis hors service.

Dans ce dernier chapitre nous allons évaluer d'une manière précise les différentes variantes du protocole Directed Diffusion, et cela grâce à notre implémentation simple et démonstrative de ce dernier, en effet notre application développée avec tinyos-2.x et la simulation avec TOSSIM, nous ont permis d'évaluer ce protocole de manière clair, et ainsi pouvoir le comparer avec l'autre protocole de routage qui est le Flooding.

Nous présentons en premier lieu l'environnement de simulation utilisé avec les métriques de performances mesurées, les scénarios de simulations adoptées puis, nous donnons l'interprétation des résultats obtenus à l'issue de ces simulations.

2. L'ENVIRONNEMENT DE SIMULATION TOSSIM ET TINYOS

Parmi les simulateurs les plus usités dans la communauté des réseaux de capteurs sans fil, nous citons TOSSIM. Celui-ci fonctionne avec le système d'exploitation embarqué TinyOS. Pour l'évaluation de notre travail, nous avons choisi d'utiliser ce simulateur.

Notre choix se justifie par les particularités qu'offre son système d'exploitation TinyOS ; que nous décrivons ci-après :

TinyOS [12] est un système d'exploitation open source conçu pour des réseaux de capteurs sans fil développé et maintenu par l'université de Berkeley et de nombreux

contributeurs. Il est actuellement utilisé par plus de 500 universités et centre de recherche dans le monde.

La particularité principale de ce système d'exploitation est sa taille extrêmement réduite en termes de mémoire, grâce à une architecture basée sur une association de composants, réduisant la taille du code nécessaire à sa mise en place, ceci permettant le respect des contraintes de mémoire qu'observent les réseaux de capteurs. La bibliothèque de composants de TinyOS est particulièrement complète puisqu'on y retrouve des protocoles de réseaux, des pilotes de capteurs et des outils d'acquisition de données. L'ensemble de ces composants peut être utilisé tel quel ou être adapté à une application précise (mesure de température, taux d'humidité, etc.). De plus, en s'appuyant sur un fonctionnement événementiel, TinyOS propose à l'utilisateur une gestion très précise de la consommation du capteur et permet de mieux s'adapter à la nature aléatoire de la communication sans fil entre interfaces physiques.

TOSSIM [13] est un simulateur conçu pour les réseaux de capteurs fonctionnant dans un environnement TinyOS. L'utilisateur peut s'en servir pour la compilation d'une application sur un ordinateur au lieu de la compiler sur de vrais capteurs. Ceci est possible grâce à la commande :

« $ make micaz sim », utilisée dans la version TinyOs-2.x. (celle que nous utilisons), ou « $ make pc » utilisée dans TinyOs-1.x.

TOSSIM supporte deux interfaces de programmation : Python et C++. Python permet d'intéragir dynamiquement avec une simulation en cours d'exécution. Cependant, puisque l'interpréteur peut causer un goulot d'étranglement lors de l'obtention des résultats, TOSSIM a également une interface C++. Habituellement, la transformation de code d'un langage à un autre est très simple.

3. IMPLEMENTATION ET SIMULATION DU PROTOCOLE DIRECTED DIFFUSION

Nous avons utilisés le système d'exploitation TinyOs-2.x avec le simulateur TOSSIM configuré, cela dans une machine Virtuelle qui tourne sous le système d'exploitation UBUNTU, cette version étant ancienne n'était pas flexible pour la mise à

jour de certains logiciels, cependant en manque d'infrastructures réels, cet environnement de simulation est un avantage pour nos expérimentations.

a. Scénario de Simulation

Durant notre phase de simulation, nous avons exécuté plusieurs scénarios, en variant à chaque fois le nombre de nœuds, plus précisément la topologie de notre réseau, en effet nous avons utilisé une topologie de 10 nœuds afin de valider notre application, après cela nous avons ajouté d'autres topologies. Ceci permet d'analyser chaque paramètre d'évaluation et de comparer les deux protocoles.

Au début de chaque exécution, les nœuds sont déployés dans un terrain carré de taille « 200 x 200 » qui est modifiable suivant la topologie et le nombre de nœuds déployés. Pour un réseau de N nœuds, le nœud dont l'identificateur ID=1 représente le collecteur (sink). Le Ns qui est le nœud source capteur d'évènement, est choisi dans notre application, cela facilite notre évaluation, vu le manque d'infrastructures réels, le résultat étant le même dans ce cas.

b. Phases du protocole Directed Diffusion dans une topologie de 10 nœuds

Nous avons validé notre implémentation de ce protocole en testons notre application sur une topologie de 10 nœuds généré grâce à un fichier « matlab » comme suit :

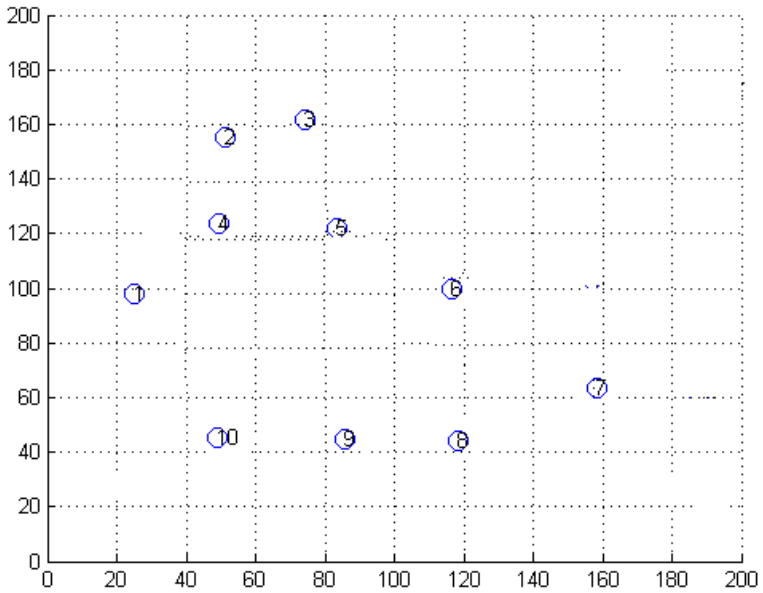


Figure 3. 1 : topologie de simulation de 10 nœuds

Dans ce cas nous avons choisi le nœud dont l'ID=7 comme nœud source (capteur d'évènement) et le Sink dont l'ID=1 et qui restera fixe. Un débit fixe pour toutes les phases est programmé, ainsi qu'un intervalle (Timer) qui synchronise l'envoi des intérêts.

La structure des messages « interest » et « donnée_exploratoires » définit dans le fichier : « DD.h » sont montrés dans la figure 3.2 :

```
#ifndef _DD_INC_
#define _DD_INC_
// ============== Message Interet ==================
typedef struct {
  uint16_t seqNum ;        // numero de seq (32- bits)
  uint16_t sink ;          // noeud sink qui est la source qui genere le packet
  uint16_t prevHop;        // dernier saut
  int8_t  ttl;             // time to live ttl
  uint16_t expiration ;    // temps d'expiration de l'interet(ou renforcement)
  uint8_t  numAttrs ;      // nombre d'attributs contenus dans un packet
  Attribute  attributes[MAX_ATT] ;  // attributes tuples
} __attribute__((packed)) InterestMessage ;

// ============= Message de donnees  exp======================

typedef struct {
  uint16_t seqNum ;        // numero de seq (32- bits)
  uint16_t source ;        // noeud source qui genere le packet
  uint16_t prevHop ;       // dernier saut
  int8_t hopsToSrc ;       // nombre de sauts traversés depuis la source
  uint8_t  numAttrs ;      // nombre d'attributs contenus dans un packet
  Attribute  attributes[MAX_ATT] ;  // attributes tuples

} __attribute__((packed)) DataMessage ;
#endif
```

Figure 3.2 : Structure des Messages « Interest » et « Donnes exploratoire »

Nous pouvons voir la représentation graphique de notre implémentation de ce protocole grâce à TinyOs, à l'aide de la commande « make micaz docs » qui nous donne le schéma suivant à travers une page web.

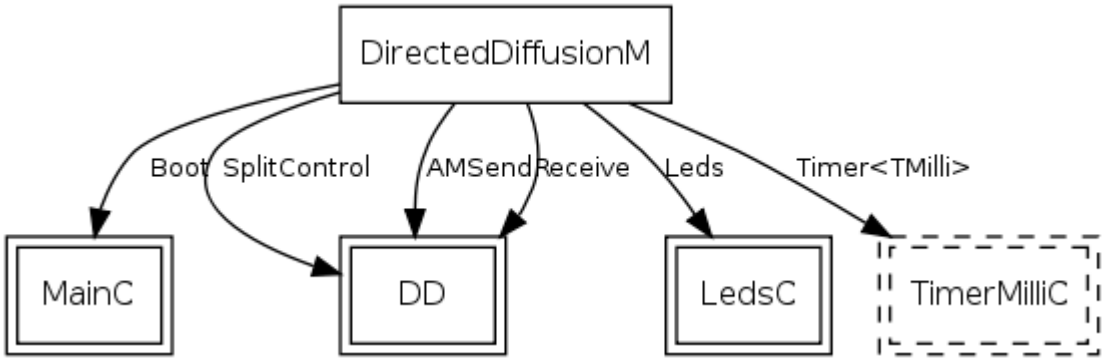


Figure 3. 3. a: Représentation graphique du programme « DD »

Ce programme utilise les interfaces :

- Boot : permet d'initialiser tous les composants au démarrage, elle est fournie par la configuration MainC qui est le coeur de l'application
- Leds : utilisée pour la manipulation des leds, fournie par LedsC
- Timer<TMilli> : c'est une interface de synchronisation qui permet de gérer le timer d'émissiondes interests et d'allumage des leds
- SplitControl : utilisée pour l'émission radio fournie par la configuration
- ActivemessageC : permet l'accès à la liaison sans fil et l'encapsulation de messages qui pourront être ensuite envoyés via la liaison sans fil.
- AMSend : pour l'envoi du paquet
- Packet : pour accéder aux données du message
- AMPacket : fournie l'adresse locale et la fonctionnalité d'accès au paquet
- Receive : pour la réception des messages

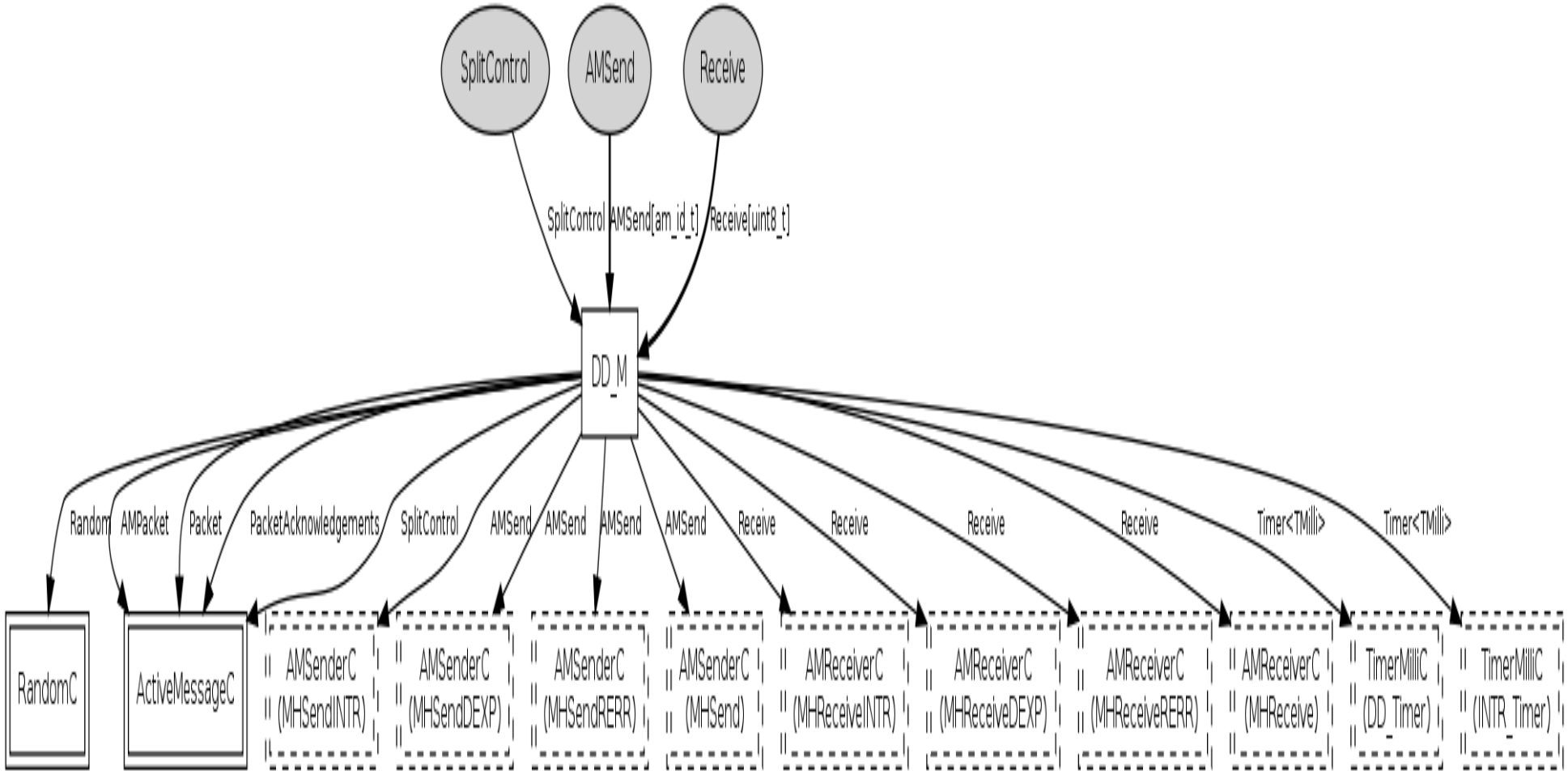


Figure 3. 2 .b : Représentation graphique du programme « DD suite »

Les différentes phases du scénario sont présentées comme suit :

- ***Phase de Dissémination des intérêts et établissement des gradiants :***

Au lancement de l'application le nœud « Sink » diffuse le message « interest » à tous les nœuds voisins, cette diffusion est propagée progressivement par les nœuds au fur et à mesure qu'ils reçoivent les « interets », chaque nœud recevant ce message met à jour le « cache d'interests » qui est une structure de type tableau définit. A chaque passage d'un « interest » par un nœud un « Gradiant » est établit, ce dernier est un message qui contient les informations des voisins dont on a reçu l'interest. Ce nœud rediffuse à son tour le même « interest ».

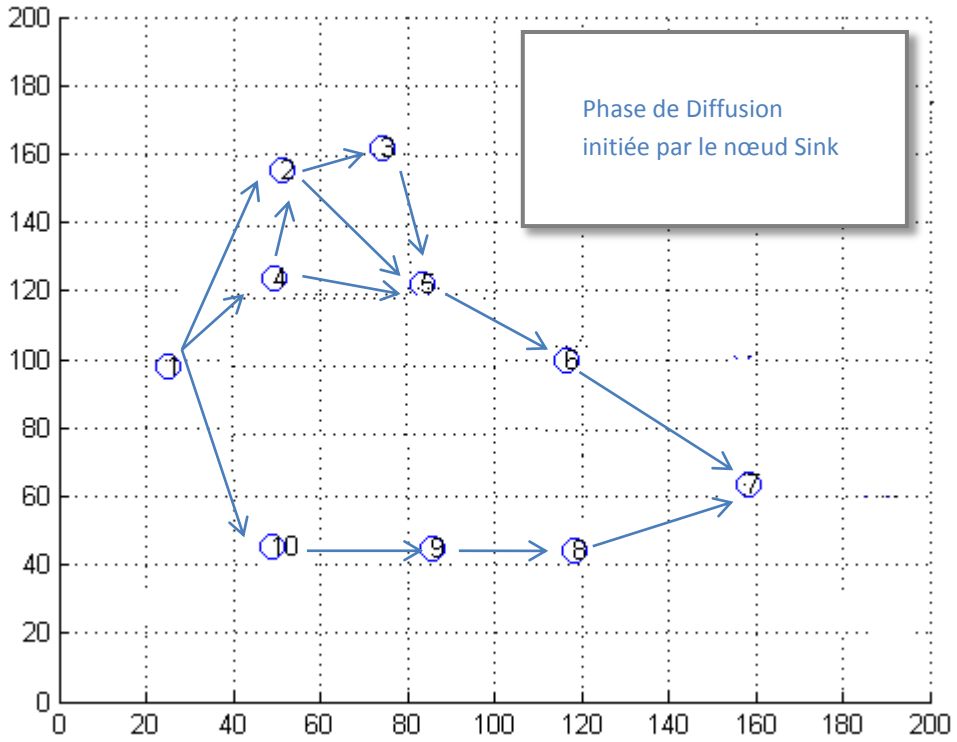


Figure 3. 3: Phase de Dissémination des intérêts

```
// structure de donnée du gradiant

typedef struct DataGradientStruct
{
  uint16_t source ;     // source du packet
```

```
  uint16_t prevHop ;    // dernier saut
} DataGradient ;
```

Figure 3. 5 : Structure du Gradiant associé à chaque voisin

- ***Phase de propagation des données exploratoires :***

Après cette étape d'inondation des « intérêts », le nœuds NS dont l'ID=7 envoie une donnée exploratoire après réception du message d'intérêts par ces voisins (Nœud = 8 et Nœud = 6), ce message (données_exploratoires) est alors à son tour propagé dans le chemin inverse jusqu'au nœud Sink, cette étape permet d'établir les meilleurs chemins, dans notre cas grâce au nombre de sauts, mais le Débit peut être un facteur pour le meilleur chemin.

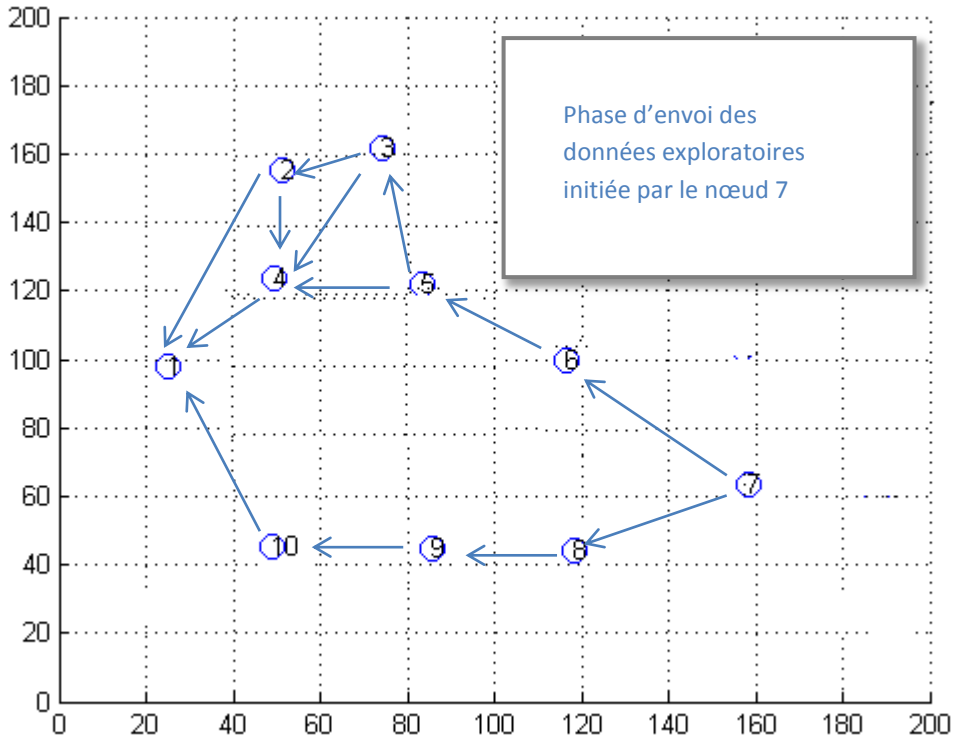


Figure 3. 6: **Phase de Dissémination des données exploratoires**

- ***Phase de Renforcement du Chemin (établissement de la route) :***

Le nœud Sink renvoi alors un message de renforcement positif au Voisin concerné par le Chemin choisi, dans notre cas le meilleur Chemin jusqu'au Ns=7 est :

Sink->Nœud 10 -> Nœud 9 -> Nœud 8 -> Ns=7

Chaque nœud de cette chaine reçoit un message de renforcement afin d'établir le Chemin.

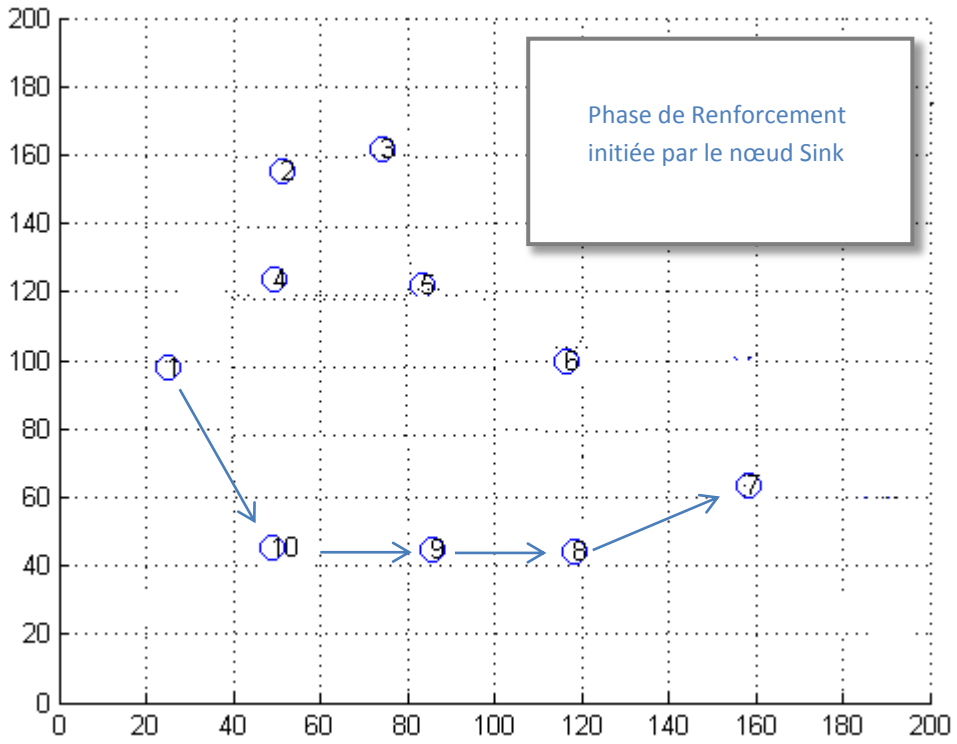


Figure 3. 7: Phase de Renforcement Positif

Le chemin ainsi établi sera utilisé pour l'acheminement des données.

Après avoir validé nos résultats de simulation et compiler l'application en utilisant cette topologie, nous avons poursuivi notre étude en variant le nombre de nœuds, allant à une topologie de 25 nœuds.

Grace à un programme python nous téléchargeons les différents fichiers nommées : « topologie.txt », ces derniers contiennent les dispositions des nœuds et leurs nombre.

Un fichier « noise.txt » est utilisé pour la simulation du bruit entre les nœuds, cela nous rapproche plus de l'environnement de capteurs réel.

Nous pouvons analyser les résultats de cette simulation dans les sections suivantes.

4. SIMULATION DU PROTOCOLE FLOODING

Lors de la simulation du protocole de Flooding, nous avons utilisé les mêmes topologies que celles de Directed Diffusion.

Le protocole de Flooding est un programme de diffusion multi sauts. Dans lequel chaque capteur rediffuse le même message reçu à ses voisins directs jusqu'à ce que le message soit traité par chaque capteur dans le réseau.

Le message sera détruit automatiquement par un capteur quand le nombre de rediffusions de ce paquet atteigne un maximum, pour cela la structure du message doit contenir un champ indiquant le nombre de rediffusion qui équivalent au champ TTL «time to live» dans les datagrammes IP, ce champ est modifié par chaque capteur traitant ce paquet avant la rediffusion.

Le même paquet est rediffusé deux fois par le même capteur ce qui entraine une perte de ressource réseaux « bande passante, batterie, temps ».

Nous avons utilisé la même topologie de 10 nœuds et initié la diffusion par le nœud Sink (ID=1).

Il n'existe pas de phases similaires à Directed Diffusion dans le flooding, le paquet est rediffusé dans tous les sens jusqu'à expiration du TTL, mais dans notre cas le test du flooding utilisé exerce sur la couche de diffusion, en provoquant le noeud avec l'ID 1 à injecter 2 nouvelles valeurs dans le réseau toutes les 4 secondes. Pour l'objet de 32 bits avec une clé 0x1234, le noeud 1 bascule LED 0 quand il envoie, et tous les autres nœuds basculent LED 0 quand il reçoit la valeur correcte. Pour l'objet de 16 bits avec une clé 0x2345, le noeud 1 bascule LED 1 quand il envoie, et tous les autres nœuds basculent LED 1 quand il reçoit la valeur correcte.

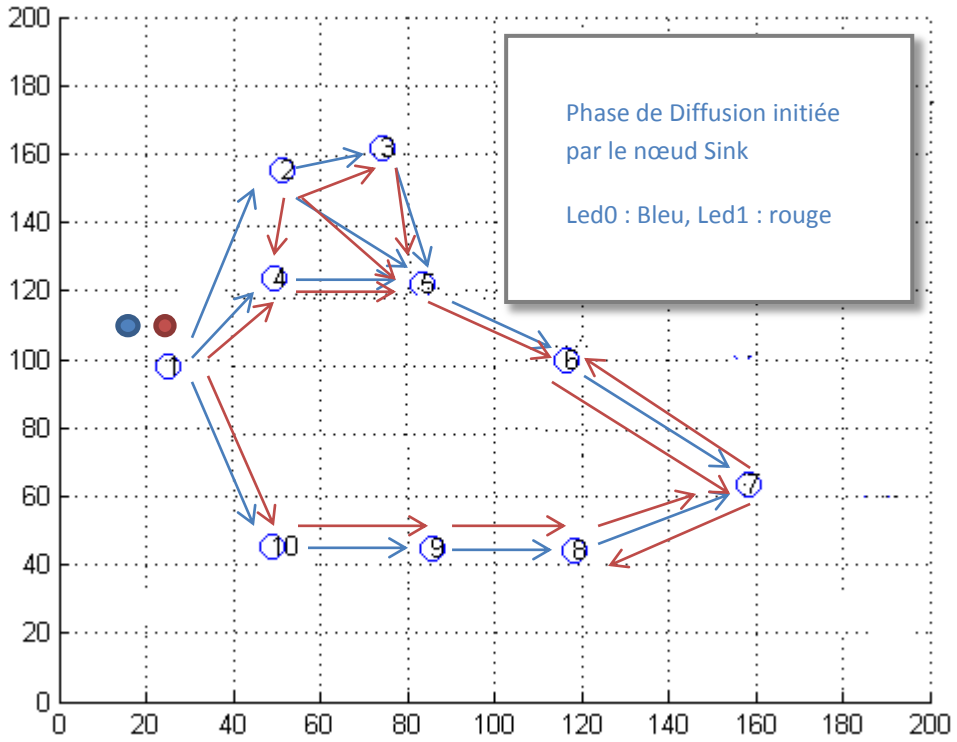


Figure 3.8: Phase de Dissémination des messages avec le jeu de Leds

Vous imaginez alors que le nombre de messages est plus important que le protocole Directed Diffusion vu que les nœuds ne traitent pas les doublons (message répété) et ne mettent pas à jour un cache ou une table de routage.

L'application se trouvant dans le répertoire « apps » de tinyOS-2.x, nommée « Dissemination » permet de tester ce protocole sur les différentes topologies.

5. RESULTATS DE SIMULATIONS

Grace au simulateur TOSSIM nous avons pu simuler les deux protocoles, voici un aperçu des imprimés écran pour une topologie de 10 Nœuds comme nous l'avons vu précédemment.

- **Aperçus des résultats de simulation du protocole Directed Diffusion :**Nous pouvons voir au niveau des carrées rouges, la diffusion des messages « interests »

initié par le nœud Sink (Nœud 1) (DEBUG (ID nœud)), cette diffusion continue avec les autres nœuds après réception de ce même message.

Les carrées bleu montre la réception de ce message par les différents nœuds.

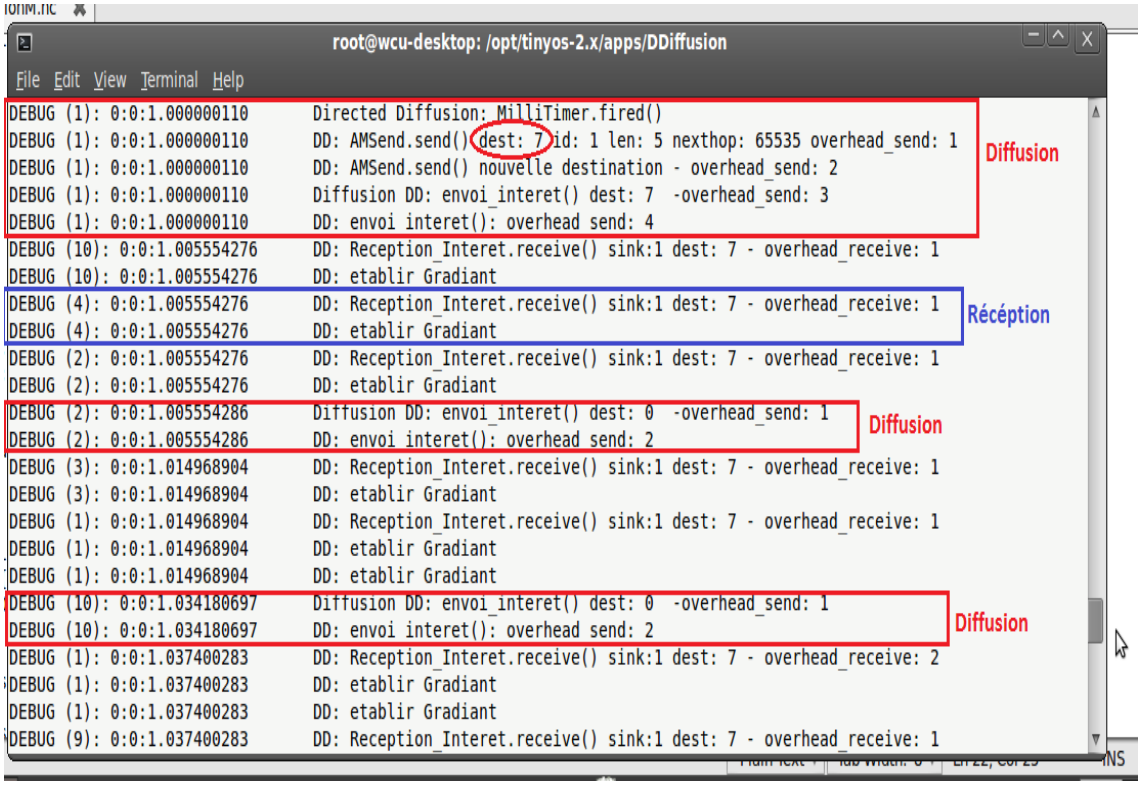


Figure 3.9: Aperçus des résultats de simulation 1

La figure suivante montre la suite de la simulation, car on ne peut afficher tout dans un seul écran. On montre ici par le carrée bleu et vert qu'après réception de l'interest par le nœud 7 qui est le nœud source (dans le programme on la appelé dest=7), concerné, ce dernier répond par l'envoi d'un message de donnée exploratoire au nœud 8 dont il a reçu l'intérêt en premier.

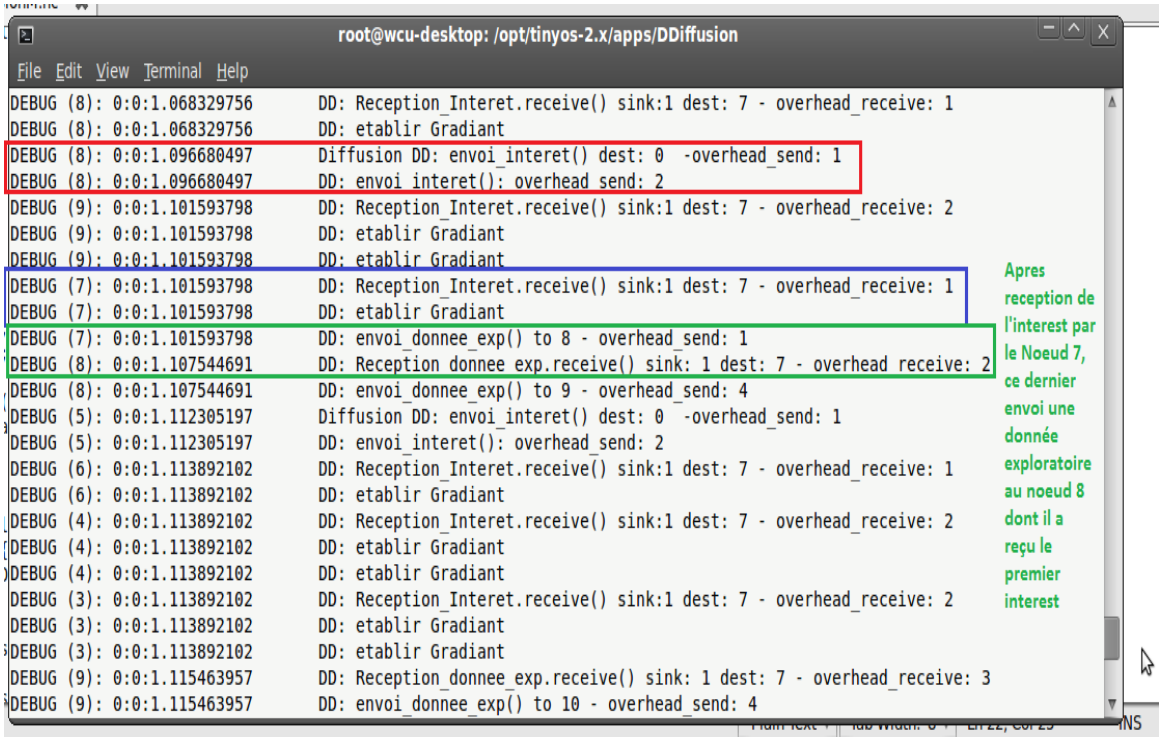


Figure 3.10 : Aperçus des résultats de simulation 2

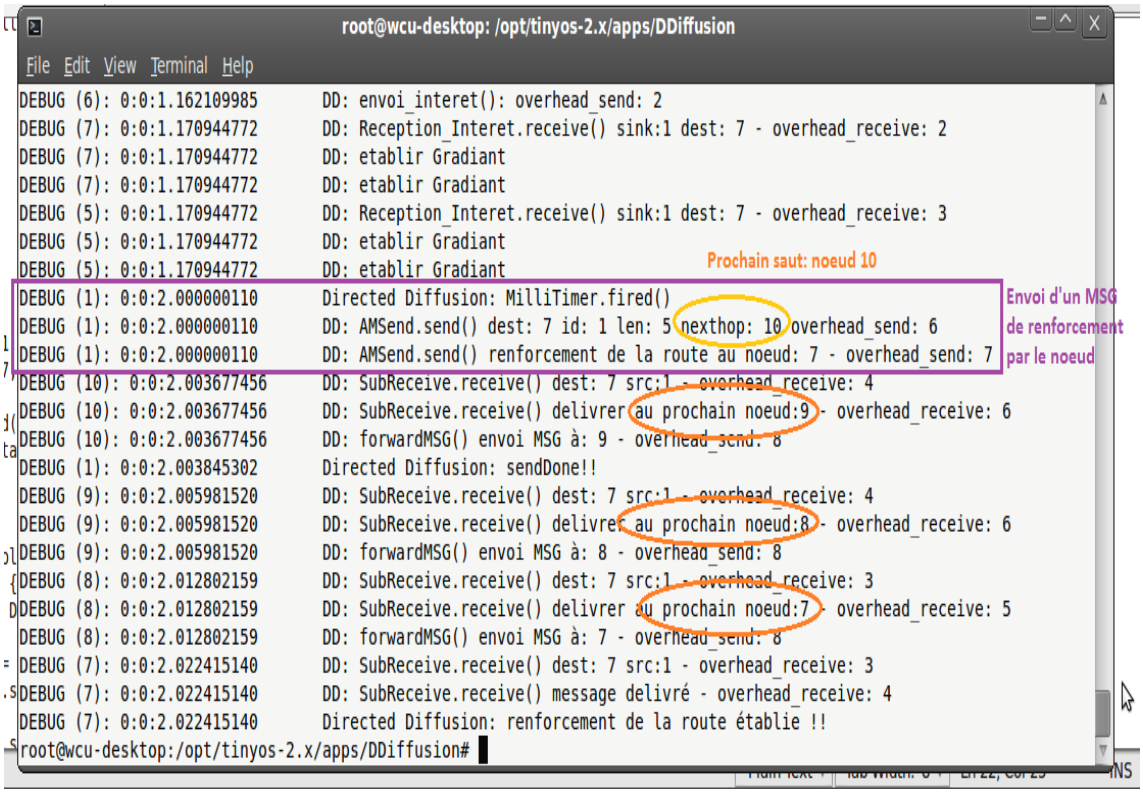


Figure 3.11 : Aperçus des résultats de simulation 3

Après réception du message de donnée exploratoire par le nœud collecteur, ce dernier transmet un message de renforcement positif au voisin inclus dans le chemin qui sera établit, comme le montre l'aperçu de la simulation 3.

- **Aperçu des résultats de simulation du Flooding**

Voici un aperçu des résultats de simulation de la Dissémination des messages, on ne peut bien sur afficher les résultats sur un seul écran, alors on a pris les imprimés écran des informations les plus importantes.

Vous pouvez voir la réception des messages 32 bits et 16 bits ce qui entraine un basculement des Leds par les nœuds, ce traitement de Leds ne nous intéresse pas réellement, ce qui nous intéresse c'est la diffusion des messages et la réception, déjà que les messages de 32 bits et 16 bits représentent 2 messages envoyés par un seul nœud, ce qui fait la répétition de la diffusion d'une certaine manière, ceci est fait pour donner un exemple du fonctionnement du flooding qui répète les messages.

Directed Diffusion aussi envoi un intérêt périodique mais dès l'établissement du chemin vers un nœud pour un intérêt précis, ce chemin est enregistré par les nœuds.

Les carrés bleus montrent la réception des messages par les différents nœuds.

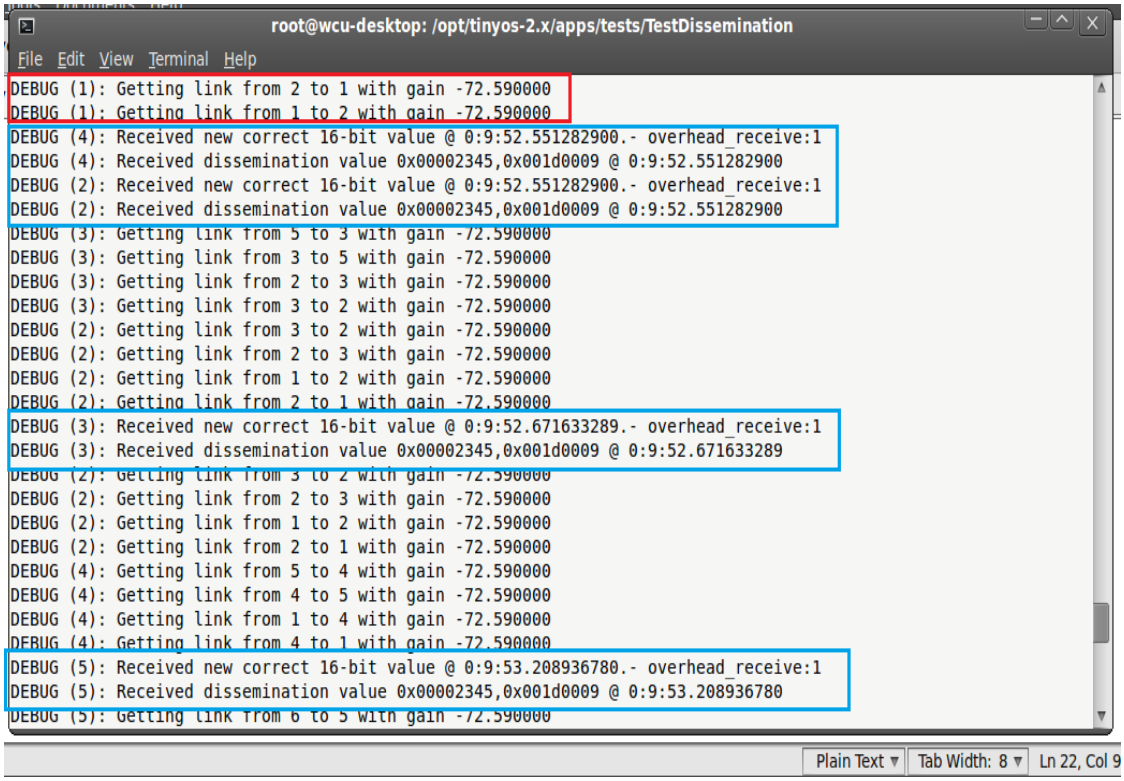


Figure 3. 12: Aperçus des résultats de simulation Flooding 1

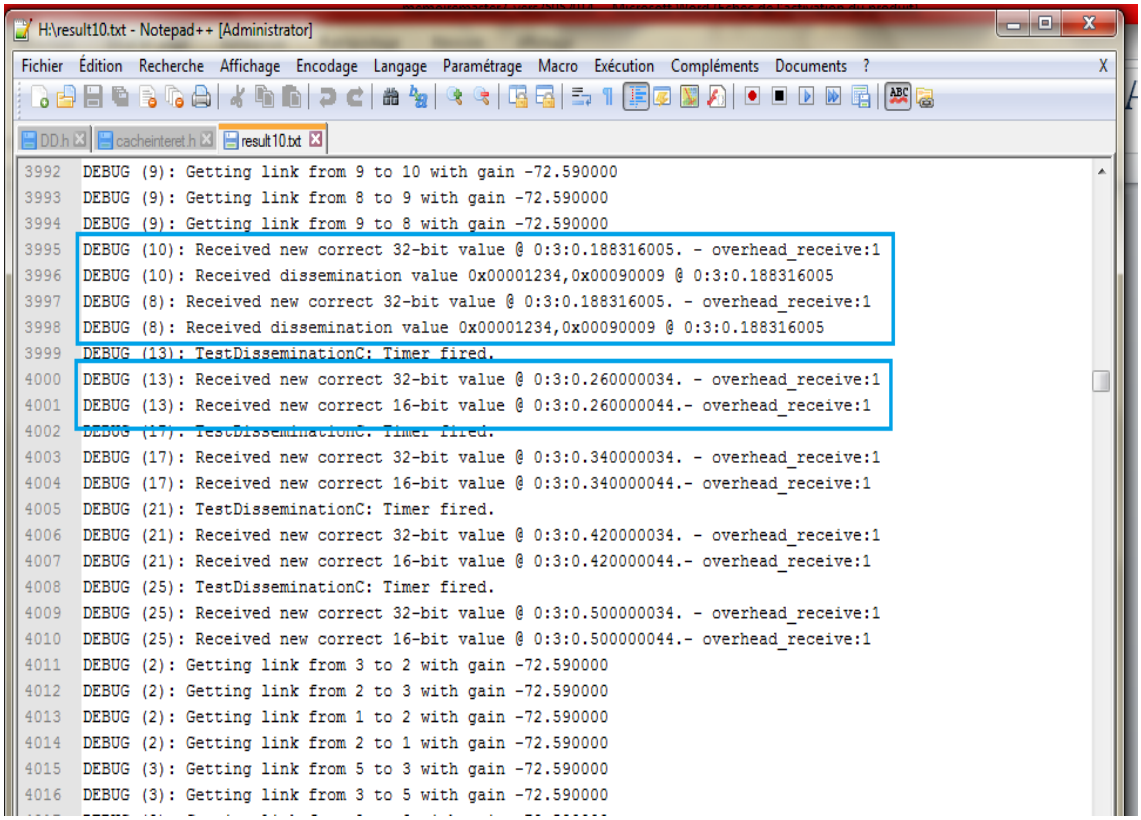


Figure 3. 13 : Aperçus des résultats de simulation Flooding 2

6. EVALUATION DES PROTOCOLES DIRECTED DIFFUSION ET FLOODING

- Métriques de simulation :

Il existe plusieurs métriques de simulations, mais dans les protocoles de routage deux paramètres sont très importants, la métriques d'overhead et la consommation d'énergie, que nous allons étudier par la suite.

a. Overhead de Communication :

- Directed Diffusion :

La métrique de l'overhead (appelée aussi surcoût de communication) représente le nombre total de bits qui circule dans le réseau, produit par tous les nœuds.

Dans notre travail, nous avons évalué cette propriété sur les deux protocoles Directed Diffusion et Flooding.

Pour le protocole Directed diffusion, nous avons ajouté 2 variable dans notre programme « overhead_send) pour les messages envoyés, et « overhead_receive » pour les messages reçus par les nœuds. A chaque envoi ou réception d'un message par les nœuds ces deux variables s'incrémentes, à la fin on assommera le totale des messages échangés suivant la formule suivante.

Remarque : vous pouvez apercevoir dans les résultats de simulation les variables overhead pour les échanges de chaque nœud.

$$\mathbf{TOTAL_{Overhead_{send}}} = \left(\mathbf{Total_{interest_{send}}} \mathbf{x\ Taille_{msg_{interest}}}\right) + \left(\mathbf{Tota_{donnees_{exp_{send}}}}\ \mathbf{x\ Taille_{donnees_{exp}}}\right) + \left(\mathbf{Total_{msg\_renforcement_{send}}} \mathbf{x\ Taille\_msg\_renforcement}\right)$$

$$\mathbf{TOTAL_{Overhead_{receive}}} = \left(\mathbf{Total_{interest_{receive}}} \mathbf{x\ Taille_{msg_{interest}}}\right) + \left(\mathbf{Tota_{donnees_{exp_{receive}}}} \mathbf{x\ Taille_{donnees_{exp}}}\right) + \left(\mathbf{Total_{msg\_renforcement_{receive}}} \mathbf{x\ Taille\_msg\_renforcement}\right)$$

$$\mathbf{TOTAL_{OVERHEAD_{COMMUNICATION}}} = \mathbf{TOTAL_{Overhead_{send}}} + \mathbf{TOTAL\_Overhead_{receive}}$$

NOTE:

Taille_msg_interest = 10 octets (80 bits)
Taille_donnees_exp = 8 octets (64 bits)
Taille_msg_renforcement = 8 octets (64 bits)

- Flooding ou Dissémination :

Pour le Flooding la procédure de calcul de l'Overhead est la même, à chaque envoi et réception les deux variables que nous avons introduit s'incrémentent, et afin d'avoir une parfaite comparaison nous pouvons changés la tailles des messages utilisés dans le flooding afin qu'ils correspondent aux tailles des messages dans Directed Diffusion.

Mais nous avons gardé les tailles par défauts car les résultats sont de toutes les façons plus importants que le protocole de routage Directed Diffusion.

Taille_msg_Led 0 = 4 octets (32 bits)
Taille_msg_Led 1 = 2 octets (16 bits)

b. Consommation d'Energie :

Sachant que les réseaux de capteurs sont basés sur la communication multi-sauts, chaque nœud joue à la fois un rôle d'initiateur de données et de routeur également, le mal fonctionnement d'un certain nombre de nœud entraîne un

changement significatif sur la topologie globale du réseau, et peut nécessiter un routage de paquets différent et une réorganisation totale du réseau. C'est pour cela que le facteur de consommation d'énergie est d'une importance primordiale dans les réseaux de capteurs. La majorité des travaux de recherche menés actuellement se concentrent sur ce problème afin de concevoir des algorithmes et protocoles spécifiques à ce genre de réseau qui consomment le minimum d'énergie.

- **PowerTOSSIM :**

PowerTOSSIM est l'extension de TOSSIM qui contient un modèle de consommation d'énergie. Pour les valeurs de consommation, les auteurs se sont basés sur le Mica2. Ils ont supposé connaitre les consommations des différents composants de ce nœud suivant leurs états. Grâce au modèle de simulation basé sur TinyOS, on connaît immédiatement l'état des composants puisque les changements d'états correspondent à des événements dans TinyOS et donc dans TOSSIM. Par conséquent, PowerTOSSIM permet d'avoir des résultats très proches de la réalité sur le plan de la consommation d'énergie. Enfin, les simulateurs TOSSIM et PowerTOSSIM ne conviennent que pour des applications écrites en TinyOS.

Afin de calculer l'énergie consommée par notre programme suivant la topologie nous avons installé PowertTossim sous tinyOs-2.x, ce dernier contient un fichier Python nommé : « postprocessZ.py » ce fichier permet d'interpréter le résultat d'un fichier trace, dans notre cas le fichier trace contient les paramètres et séquences de CPU et RADIO, LEDs ,…etc. [19]. Ces paramètres doivent être interprétés afin de calculer l'énergie consommé de chaque Nœud et cela pour les 4 topologies que nous avons. Il est important avant de savoir qu'il faut ajouter le Channel Energie : **« t.addChannel("ENERGY_HANDLER", sys stdout »** dans le Fichier python « test.py » et aussi ajouter dans le « makefile » du programme ce qui suit [20]:

« CFLAGS += -DPOWERTOSSIMZ »

Voici un aperçu du Fichier trace de notre Topologie de 10 Nœuds pour le protocole Directed Diffusion :

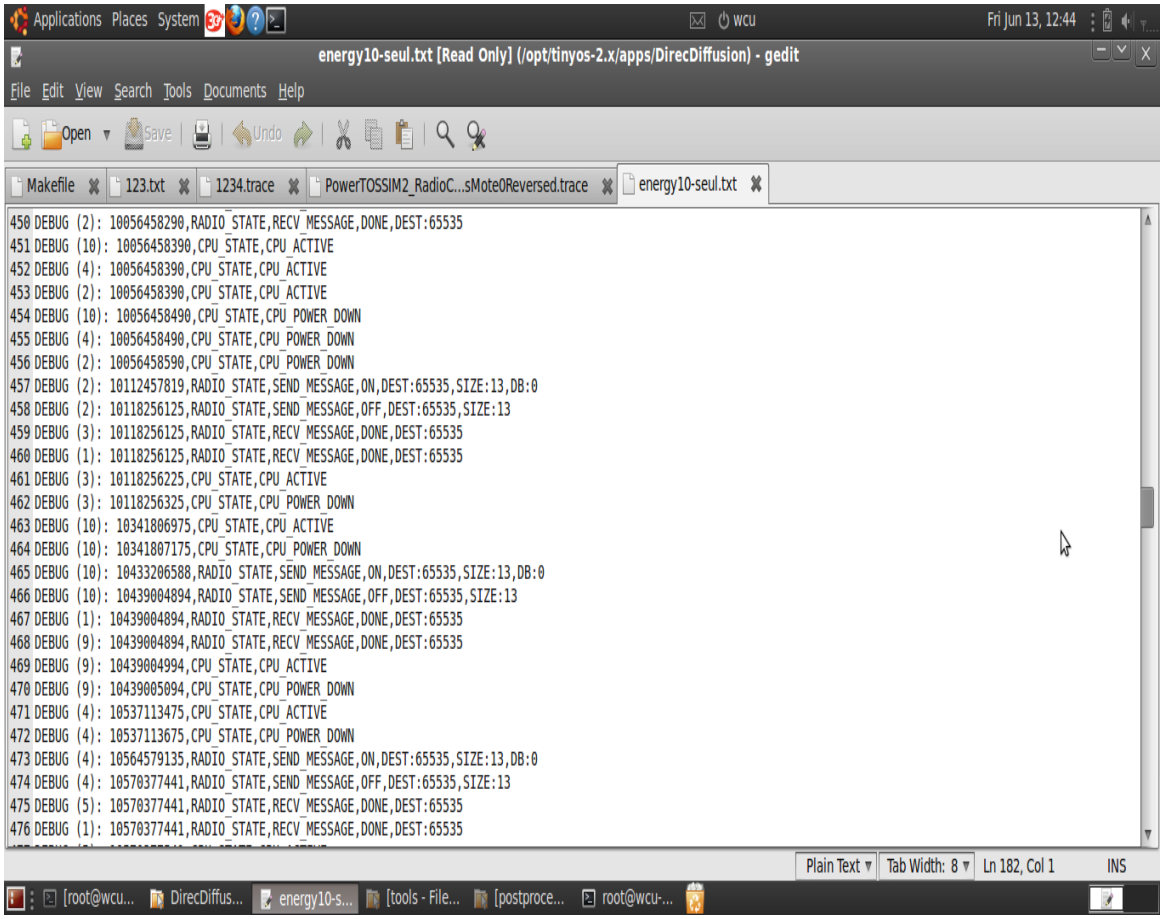


Figure 3.14: Aperçu du fichier trace d'Energie pour 10 nœuds

Après avoir exécuté la commande qui permet de générer l'Energie consommé, nous obtenons le résultat de l'imprimé Ecran suivant :

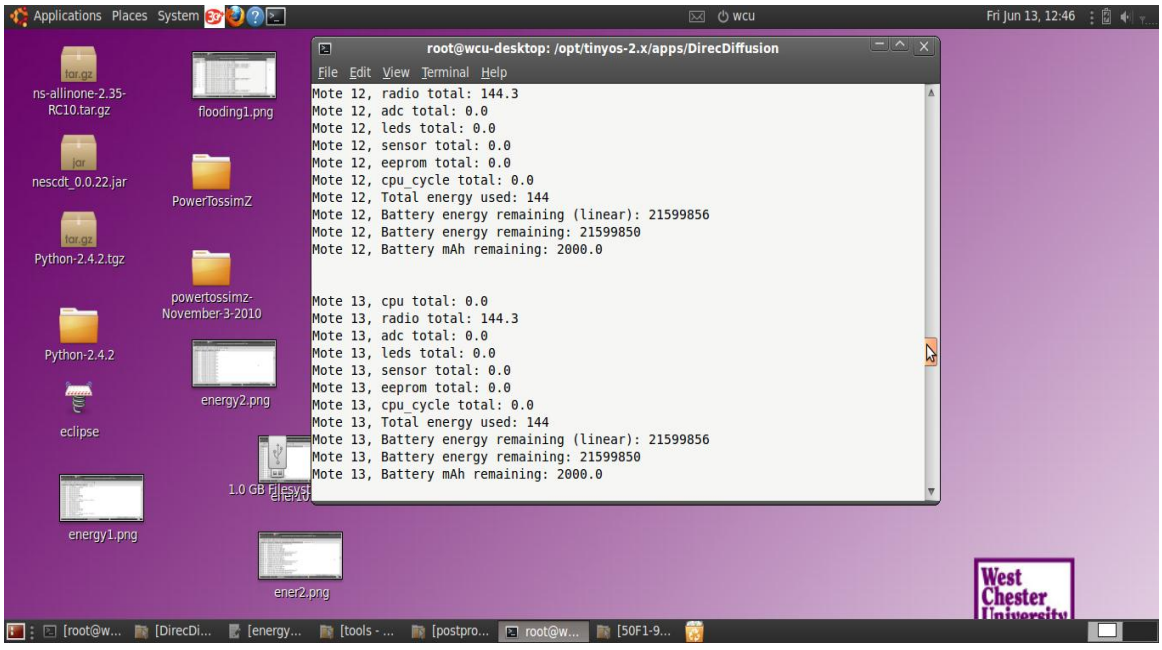


Figure 3.15: Aperçu de l'Energie consommée pour chaque nœuds

La commande utilisée et **: « python2.4 postprocessZ.py energy10-seul.txt > energy10.txt ».**

On obtient ainsi le Total de l'énergie consommé pour chaque nœuds, nous calculons la somme (addition) de l'énergie de tous les nœuds afin d'obtenir le Total de consommation d'énergie pour une topologie. Le même travail est effectué pour le Flooding.

Le résultat sera interprété dans les prochains chapitres.

- Observations et discussion

- **Overhead de Communication :**

Le résultat de calcul de l'overhead par les programmes est récapitulé dans le tableau suivant :

Topologie (Nombre de nœuds)	Total_Overhead_Directed Diffusion (octets)	Total_Overhead_Flooding (octets)
10	516	948
15	752	1376
20	998	1814
25	1248	2302

Tableau 3. 1: Données overhead Directed Diffusion et Flooding

La figure 3.17 illustre la courbe de l'overhead pour les différentes topologies, et ce, en variant la taille du réseau.

L'axe des abscisses représente le nombre de nœuds (topologies), et l'axe des ordonnées représente les octets.

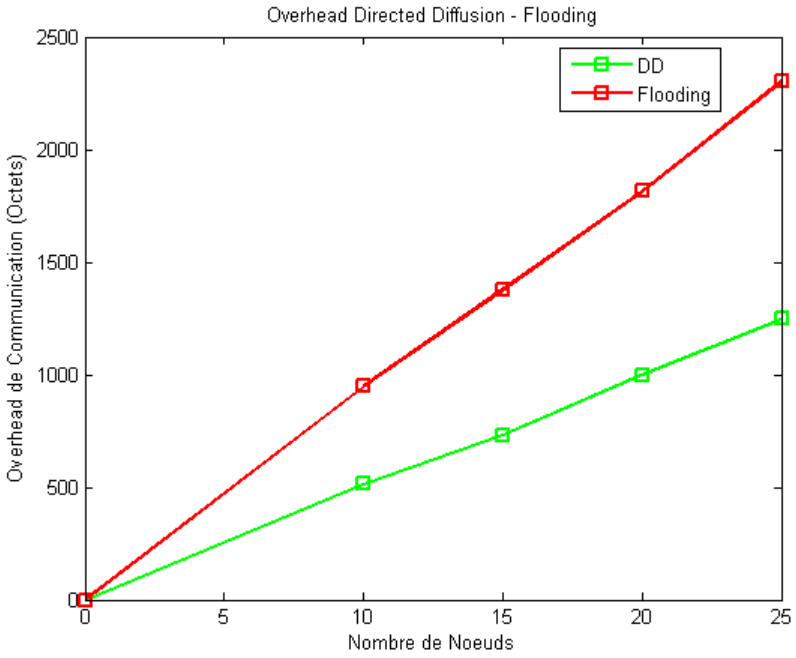


Figure 3.16: Overhead en fonction du nombre de nœuds

Nous remarquons clairement que plus la taille du réseau augmente plus l'overhead augment. Néanmoins l'overhead généré par le flooding est considérablement plus élevé et croit de manière exponentielle par rapport à l'overhead généré par directed diffusion. Ce résultat confirme que Directed diffusion est un protocole plus optimal par rapport au flooding.

Le graphe de comparaison des Overhead des deux protocoles, révèle que le protocole Directed Diffusion présente un Overhead plus bas que celui du Flooding et ceci pour les différentes topologies testés.

On remarque aussi que la progression de l'Overhead est plus contrôlée que celle du Flooding qui augmente d'une manière exponentielle.

On ne doit pas oublier que la taille des paquets dans le protocole Directed Diffusion est plus importante que celle du Flooding ce qui confirme que l'Overhead du Flooding est nettement plus important que celui de Directed Diffusion.

Nous avons testé les deux protocoles dans une première phase de découverte, mais on ne doit pas oublier que le protocole Directed Diffusion après avoir établi un chemin vers une source, ce chemin est alors enregistré et lui sert après donc pour

le même nœud il n'y aura pas d'étapes de Dissémination comme la première fois lors de la mise en place du réseau. Ce qui n'est pas le cas du Flooding.

- **Consommation d'Energie :**

Le résultat de calcul de l'énergie consommée (mJ) par les programmes est récapitulé dans le tableau suivant :

Topologie (Nombre de nœuds)	Total_Energie_Consommée_Directed Diffusion (mJ)	Total_ Energie_Consommée _Flooding (mJ)
10	1449,7	3444,29326
15	2171,2	5166,43989
20	2892,7	6888,58682
25	3614,2	10835,73315

Tableau 3. 2: Données Energie Consommée Directed Diffusion et Flooding

La figure 3.18 illustre la courbe de l'énergie pour les différentes topologies. L'axe des abscisses représente le nombre de nœuds (topologies), et l'axe des ordonnées représente l'Energie en mJoules.

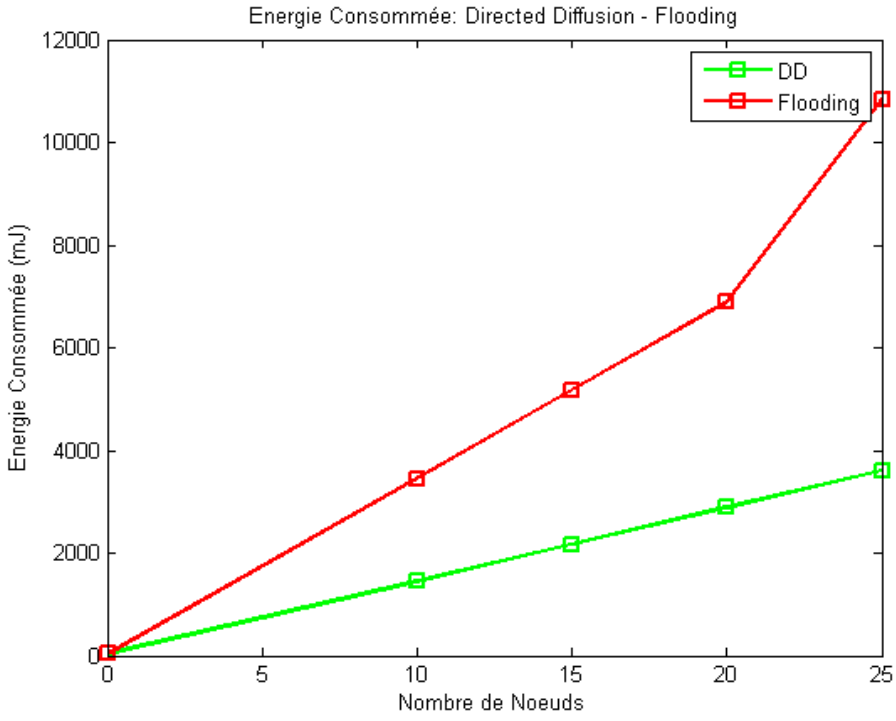


Figure 3.17: Energie Consommée en fonction du nombre de nœuds

Le graphe précèdent montre en effet que le protocole Directed Diffusion a un taux d'énergie consommée moins important que le Flooding, cela est normal vu que Directed diffusion est un protocole de routage qui diminue le principe de l'inondation pour l'utiliser le moins possible pour trouver les chemins vers des nœuds sources.
PowerTossim donne l'énergie de chaque composant LED, RADIO, …etc, ce qui fait que l'estimation de l'énergie avec cet outil est très proche de l'estimation d'énergie réelle.

La dissémination des intérêts repose essentiellement sur le principe de l'inondation, et les données exploratoires sont propagées à travers tout le réseau. Cette approche est assez inefficace et peut gaspiller beaucoup d'énergie, particulièrement dans un réseau à forte densité. Dans un tel réseau, où le nombre de voisins pour un nœud est élevé, le nombre de réceptions d'un même intérêt au niveau de chaque nœud pendant la phase de dissémination devient importante. Par conséquent, le nombre de gradients maintenus ainsi que celui des données exploratoires envoyées sont plus élevés.

Concernant le Flooding, l'accroissement de la quantité d'énergie consommée, à cause de l'augmentation du nombre de messages transmis a un impact négatif sur ce protocole, en effet en plus du principe de l'inondation le gaspillage d'énergie affecte le réseau considérablement.

7. CONCLUSION

Nous avons vu dans ce chapitre que le protocole Directed Diffusion est meilleur que le Flooding, cependant Directed Diffusion a beaucoup d'inconvénients, notamment dans la consommation d'énergie surtout dans les réseaux très denses, l'établissement des meilleurs chemins qui n'est pas toujours fiable, car nous avons évaluer ce protocole de la manière la plus simple possible, mais dans la réalité des capteurs l'évaluation des métriques étudiés est plus importante.

L'inconvénient majeur du protocole Directed Diffusion du point de vue gestion de la source énergétique est l'absence d'une métrique qui lui permet le choix des routes les plus efficaces en consommation d'énergie parmi celles explorées. Cette utilisation aveugle peut gaspiller de l'énergie inutilement ce qui provoque un arrêt du réseau.

CONCLUSION GENERALE

Conclusion générale

Les Réseaux de capteurs sans fil (WSN) sont constitués de nombreux minuscules capteurs déployés à haute densité dans les régions nécessitant une surveillance et un suivi. Ces capteurs peuvent être déployés à un coût beaucoup plus faible que le système traditionnel câblé.

Un capteur typique est constitué d'un ou plusieurs éléments de détection (mouvement, température, pression, etc…), une batterie, récepteurs radio faible puissance, le microprocesseur et une mémoire limitée.

Un aspect important de ces réseaux est que les nœuds sont sans surveillance, ont une énergie limitée et la topologie de réseau est inconnue. Beaucoup de défis de conception qui se posent dans les réseaux de capteurs sont en raison des ressources limitées dont elles disposent et de leur déploiement dans des environnements hostiles.

Dans ce mémoire, nous avons présenté une brève description des réseaux de capteurs ainsi que les différents protocoles de routage, nous avons présenté aussi une description détaillée du fonctionnement du protocole Directed Diffusion. Sa simplicité de conception et ses avantages font de lui l'un des protocoles les plus répandus et les plus faciles à implémenter dans diverses applications des réseaux de capteurs sans fil.

Directed Diffusion est caractérisé par sa technique d'exploration périodique afin d'élire les chemins de routage les plus fiables. Il utilise par ailleurs, des règles de renforcements positif et négatif pour la gestion de ces routes.

Suite à notre analyse de ce protocole, nous avons noté un ensemble de caractéristiques. En effet, Directed Diffusion possède des avantages qu'offrent ses phases d'exploration et de renforcements positif et négatif. Nous résumons les privilèges de Directed Diffusion dans les points suivants :

– Protocole réactif ;
– Interaction locale ;
– Routage multi-chemin ;
– Recouvrement de route.

Néanmoins, Directed Diffusion marque certaines limitations lui empêchant d'avoir une meilleure gestion de pannes. Ces manques sont les suivants :

– Détection et recouvrement retardés des pannes ;
– Grande perte de données ;
– Mauvaise gestion de l'énergie ;
– Mécanisme d'élimination de panne inefficace.

Bien que Directed Diffusion soit un protocole réactif et utilise des explorations périodiques pour la sélection des routes les plus fiables, nous avons vu qu'il choisit de manière empirique les chemins fournissant les meilleurs délais de transmission et ne permet pas encore de prendre en compte le facteur d'énergie lors du choix de ses routes.

Pour ce qui est du protocole Flooding, notre modeste travail a montré que pour un réseau peu dense la quantité de bande passante, énergie et paquets est importante dans ce protocole, c'est pour cette raison que les équipes de recherches ont développé et développent encore des protocoles de routages multicast qui favorisent un chemin suivant des métriques précises, même si cela n'est pas chose évidente dans le domaine des Réseau de Capteurs sans fil, vu leurs caractéristiques et leurs ressources.

En comparant ces deux protocoles nous avons vu et observé des résultats pour un réseau de capteurs dont le nombre est peu important, une topologie de 25 nœuds ce n'est pas ce qu'on trouve généralement dans les réseaux déployés dans la réalité, cela veut dire que l'overhead et la consommation d'énergie dans les réseaux de 100 nœuds et plus sont plus importants, d'autres mécanismes de réduction de ces métriques doivent être déployés pour diminuer ces variantes, ce qui est déjà en train de se faire dans ce domaine afin d'atteindre les performances à moindre coût.

BIBLIOGRAPHIE

[1] N.LABRAOUI, La sécurité dans les réseaux sans fil ad hoc: Agrégation de données et localisation, thèse de doctorat, 2012.

[2] C.T.KONE, Conception de l'architecture d'un réseau de capteurs sans fil de grande dimension , 2011.

[3] S. Hedetniemi and A. Liestman, A survey of gossiping and broadcasting in communication networks, Networks, Vol. 18, No. 4, pp. 319-349, 1988.

[4] F.Z. BENHAMIDA, Tolérance Aux Pannes Dans Les Réseaux De Capteurs Sans Fil , thèse de magister, 2009.

[5] S.ATHMANI, Protocole de sécurité Pour les Réseaux de capteurs Sans Fil , thèse de magister, 2010.

[6] C. Intanagonwiwat, R. Govindan, and D.Estrin. Directed Diffusion : a scalable and robust communication paradigm for sensor networks. In ACM Mobicom, pages 56–67. ACM, Aout 2000.

[7] P. Li, Y. Lin, and W. Zeng, *Search on security in sensor networks*. Journal of Software, Vol. 17, pp. 2577- 2588, 2006.

[8] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, *A survey on sensor networks*. IEEE Communications Magazine, Vol. 40, Issue: 8, pp. 102–114, Août 2002.

[9] Chris Karlof and David Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. Ad Hoc Networks, 1(2-3):293–315, 2003.

[10] A.Bachir, A. Ouadjaout, L. Khelladi, M. Bagaa, N. Lasla, Y.Challal, Information Security in Wireless Sensor Networks, Handbook/Encyclopedia on Ad Hoc and Ubiquitous Computing, edited by: Agrawal Dharma P., and Xie Bin., World Scientific, 2009.

[11] N. El-Bendary, O. S. Soliman , N.I. Ghali , A. Hassanien, V. P. H. Liu , A Secure Directed Diffusion Routing Protocol for Wireless Sensor Networks ,2011

[12] TinyOS Community. an open-source Os for the networked sensor regime. Site Web : http ://www.tinyos.net, 2004.

[13] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim : accurate and scalable simulation of entire tinyos applications. In Proceedings of the 1st international conference on Embedded networked sensor systems, November 2003.

[14] L. Li and J.Y. Halpern. minimum-energy mobile wireless networks revisited.In IEEE Int. Conf.Commun. ICC, Juin 2001.

[15] W.R. Heinzelman, A. Chandrakasan, and H Balakrishnan. "energy-efficient communication protocol for wireless microsensor networks". In IEEE Proc, pages 1–10, Janvier 2002.

[16] K. Sohrabi. protocols for self-organization of a wireless sensor network. In IEEE Personal Commun, 2000.

[17] S. Hedetniemi, S. Hedetniemi, and A. Liestman. a survey of gossiping and broadcasting in communication networks. Proc.ACM MobiCom. Networks, 18 :319–349, 1988.

[18] W.R. Heinzelman, J. Kulik, and H. Balakrishnan. adaptive protocols for information dissemination in wireless sensor networks. In ACM Mobicom, Aout 1999.

[19] O.R MERAD BOUDIA. Agrégation des Données et Sécurité dans les Réseaux de Capteurs Sans Fil. Thése de doctorat 2014.

[20] S.A H. SEDJELMACI. Mise en Œuvre de Mécanismes de Securité Basées sur les IDS dans les Réseaux de Capteurs Sans Fil. Thèse de doctorat, 2012.

[21] M. AISSANI. Amélioration du Routage dans les Réseaux de Capteurs Sans Fil. Thèse de doctorat, 2011.

[22] M.LEHSAINI, Diffusion et couverture basées sur le clustering dans les réseaux de capteurs : application à la domotique, Thèse de Doctorat, 2009.

ANNEXE 1

1. Fichier “test.py”

```
#! /usr/bin/python
from TOSSIM import *
import sys

t = Tossim([])
r = t.radio()
f = open("topology10.txt", "r")       # ouvrir le fichier topology10.txt

num_node = 10                         #nombre de noeuds
end_time = 2.5 #second

lines = f.readlines()
for line in lines:
  s = line.split()
  if (len(s) > 0):
    print " ", s[0], " ", s[1], " ", s[2];
    r.add(int(s[0]), int(s[1]), float(s[2]))

t.addChannel("Boot", sys.stdout)     #channel pour le debugage de simulation
t.addChannel("APPS", sys.stdout)
t.addChannel("DD", sys.stdout)
t.addChannel("ENERGY_HANDLER", sys.stdout) //energie consommee

noise = open("noise.txt", "r")      #ouvrir le fichier noise.txt
lines = noise.readlines()
for line in lines:
  str = line.strip()
  if (str != ""):
    val = int(str)
    for i in range(1, num_node+1):
      t.getNode(i).addNoiseTraceReading(val)

for i in range(1, num_node+1):
  print "Creer model de bruit pour ",i;
  t.getNode(i).createNoiseModel()
  t.getNode(i).bootAtTime(1000 * i);

while True:
  t.runNextEvent()
  if t.time() > end_time * 10000000000:
break
```

ANNEXE 2

1. Procédure d'installation de TinyOs-2.x sous UBUNTU 12.04:
   - Installation de UBUNTU 12.04 avec mise à jour des packages.
   - Installation de TinyOs :

1. Choisir une version de UBUNTU recente
2. Ouvrir un terminal sous UBUNTU sous l'utilisateur « root »
3. Taper la commande suivante:
   $ sudo gedit /etc/apt/sources.list
4. Dans le fichier sources.list ajouter la source de depot de package tinyos suivant :

#tinyos
deb http://hinrg.cs.jhu.edu/tinyos oneiric main
5. Retourner au terminal et taper les commandes suivanteslines:

$ sudo apt-get update
$ sudo apt-get install tinyos-2.1.1
6. Aprés que l'installation soit terminée taper les commandes suivantes:

$ gedit ~/.bashrc
7. Dans le fichier .bashrc ajouter les lignes suivantes et sauvegarder :

export TOSROOT=/opt/tinyos-2.1.1
export TOSDIR=$TOSROOT/tos
export CLASSPATH=$TOSROOT/support/sdk/java/tinyos.jar:.:$CLASSPATH
export MAKERULES=$TOSROOT/support/make/Makerules
export PATH=/opt/msp430/bin:$PATH
#Sourcing the tinyos environment variable setup script
source /opt/tinyos-2.1.1/tinyos.sh
8. Taper exit dans le terminal
9. Ouvrir un nouveau terminal, si tout va bien vous verrez ce message "setting up for Tinyos 2.1.1"
10. Chercher l'icone de Software Center
11. chercher "Synaptic Package Manager" et l'installer
12. Chercher "Synaptic Package Manager"

13. quand c'est ouvert
a. Chercher avr-libc-tinyos
b. effacer ce package
c.Chercher avr-gcc-tinyos
d. effacer ce package
e. Chercher avr-binutils-tinyos
f. effacer ce package
g. appuyer sur « Apply »
h. chercher avr-libc
i. cliquer pour Installation
j. chercher gcc-avr
k. cliquer pour Installation
l. chercher binutils-avr
m. cliquer pour Installation
n. Cliquer sur Apply

14. Félicitation vous avez TinyOS installé.

Installer la cersion java correcte

1- sudo add-apt-repository ppa:webupd8team/java

2- sudo apt-get update

3- sudo apt-get install oracle-java7-installer

Note: il faut être sur que la version du NesC est bien 1.3.3 et est installée.

2. Procédure d'installation de PowerTossimZ:

PowerTOSSIM-Z: Realistic Energy Modeling of MicaZ in Tossim

Releases in: TinyOS 2.0.2
TinyOS 2.1.1 TinyOS 2.x

Fichiers dans PowerTOSSIMZ à copier afin d'avoir le fichier de trace d'énergie :

- **Dans tos/lib/tossim**
  - ActiveMessageC.nc (changes)
    - SimSchedulerBasicP.nc (changes)
    - TinySchedulerC.nc (changes)
    - TossimPacketModelC.nc (changes) (fix bug - "duration =")
    - PacketEnergyEstimator.nc (new)
    - PacketEnergyEstimatorC.nc (new)
    - PacketEnergyEstimatorP.nc (new)

- **Dans tos/chips/atm128/pins/sim**
  - HplAtm128GeneralIOPortP.nc (new)
    - HplAtm128GeneralIOPinP.nc (changes)

- **Dans tos/chips/atm128/sim**
  - Atm128EnergyHandler.nc (new)
    - Atm128EnergyHandlerC.nc (new)
    - Atm128EnergyHandlerP.nc (new)
    - McuSleepC.nc (changes)

- **Dans support/make/sim.extra**
  - Ajouter "build_storage" à la ligne sim-exe pour construire le fichier flash .h

Ajouter la directive "POWERTOSSIMZ" dans le makefile comme suit:
CFLAGS += -DPOWERTOSSIMZ, ajouter le channel "**ENERGY_HANDLER**" dans votre fichier python.

Interpréter fichier trace ou texte résultat.

Dans le répertoire postprocessZ/ copier tout dans le répertoire de votre application et exécuter :

python postprocessZ.py Energy.txt > votre fichier résultat.txt

Note : en cas d'erreur « NO MODULE TOSSIM FIND » exécuter la commande :

Chmod 755 dans les fichier *.so