REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

MEMOIRE

Pour l'obtention de Diplôme de Master en Informatique

Option : Systèmes Informatiques

Intitulé :

Amélioration de l'algorithme de routage hiérarchique LEACH

Présenté par : MOKDES Lydia et LOUNI Meriem

Dirigé par : M[r] M.DEMRI

Promotion 2013/2014

Remerciements

Nous tenons en premier lieu à remercier le bon Dieu de nous avoir donné la chance d'y exister, le courage et la patience pour réaliser ce projet.

Nous remercions nos chers parents pour leur grand soutien.

Nous remercions notre promoteur Monsieur DEMRI.M *pour nous avoir encadré et donné la chance d'acquérir des connaissances dans le domaine des Réseaux de Capteurs Sans Fil, pour sa présence, sa motivation et son sens d'organisation.*

Nous remercions également les membres du jury pour avoir accepté de juger ce travail.

Nos remerciements à ceux et celles qui de loin ou de prés ont contribué à la réalisation de ce travail.

Résumé

Les réseaux de capteurs sans fil (RCSF) est une technologie émergente qui vise à offrir des capacités innovantes. Leur utilisation ne devrait cesser d'augmenter et ceci dans de nombreux domaines qu'ils soient scientifiques, logistiques, militaires ou encore sanitaires. Cependant, la taille des capteurs constitue une limitation importante, principalement en terme d'autonomie d'énergie et donc de durée de vie car la batterie doit être très petite, c'est pourquoi de nombreux travaux portent aujourd'hui sur la gestion de l'énergie consommée par les capteurs dans un réseau en prenant en considération, en premier lieu, les communications et les algorithmes de routage des données. C'est dans ce but que nous avons proposé un algorithme de routage hiérarchique LEACH++ qui est une amélioration du protocole LEACH.

Afin de confirmer les améliorations apportées par notre algorithme, nous avons conduit une simulation à l'aide du simulateur TOSSIM, dans laquelle les performances de notre algorithme sont évaluées et comparées avec le protocole de clustering existant LEACH utilisé dans les réseaux de capteurs.

Mots clés: RCSF, économie d'énergie, LEACH, LEACH++, approche basé clustering, protocole de routage hiérarchique.

Sommaire

Liste des figures

Introduction générale

Aujourd'hui les réseaux de capteurs deviennent de plus en plus répandus. Ils sont utilisés dans divers domaines. Leurs applications sont de plus en plus nombreuses et diversifiée, citant par exemple: le domaine scientifique, militaire, environnementale ou encore la santé. Ces réseaux sont différents des autres réseaux sans fil car ils ont en général les spécificités suivantes : une grande densité, faible débit, faible capacité d'énergie et un environnement inaccessible. Ces deux dernières spécificités ont fait de l'énergie une contrainte très importante puisque les batteries des capteurs ne sont pas généralement rechargeables. La consommation d'énergie au niveau des capteurs a une grande influence sur la durée de vie du réseau en entier. Il est donc impératif de mettre en place des protocoles de routage efficaces en énergie, et qui prennent en compte les contraintes imposées par ces capteurs.

Dans les réseaux sans fil, on donne plus de l'importance à l'acheminement de l'information qui est assuré par des algorithmes de routage. Ces algorithmes de routage doivent prendre en considération les changements de la topologie du réseau, ainsi que d'autres caractéristiques comme la bande passante, le nombre de liens, la limitation d'énergie, etc. En particulier pour les réseaux de capteurs qui se caractérisent par des liens volatiles et des dispositifs fragiles, les protocoles de routage perdent leurs performances quand un lien est perdu ou un dispositif cesse de fonctionner. Dans ce contexte, plusieurs recherches ont été menées notamment pour garantir le routage de l'information de n'importe quel noeud vers la station de base.

L'objectif de ce mémoire est de développer un protocole existant LEACH qui est l'un des algorithmes de routage hiérarchique les plus populaires pour les réseaux de capteurs sans fil, il est basé sur l'approche de clustering; Son principal avantage est de minimiser la consommation énergétique des éléments du réseau. Dans ce protocole le réseau est divisé en clusters et chaque cluster possède un nœud maitre appelé Cluster Head.

Le souci principal est d'assurer la livraison de données à la station de base tout en prolongeant la vie du système. Pour cela, nous avons tout d'abord étudié les performances de LEACH, puis nous avons proposé une version améliorée de LEACH appelée LEACH++.

Dans cette version, nous avons partitionné le réseau en zones tel qu'au niveau de chaque zone il y a un Cluster Head. Par ailleurs, pour faire face aux pannes au niveau de chaque zone, un autre CH adjoint est élu. Si le Cluster Head cesse de fonctionner, son adjoint le remplace dans sa mission.

Ce mémoire s'articule autour de quatre chapitres :

Le premier chapitre présente des généralités sur les réseaux de capteurs sans fil, leurs caractéristiques, leurs architectures et leurs domaines d'applications.

Le deuxième chapitre est consacré au routage dans les réseaux de capteurs sans fil, ainsi que leur classification, et nous allons terminer par la citation de quelques exemples de protocoles existants pour chaque classe.

Le troisième chapitre est consacré à la description détaillée du protocole de routage LEACH et ses variantes.

Et dans le quatrième chapitre, nous allons détailler notre plateforme de simulation et les étapes d'implémentation de nos protocoles, pour passer à la comparaison des résultats de simulation.

Enfin, nous terminons par une conclusion générale qui résume nos conclusions tirées de l'étude de l'état de l'art et de la discussion des résultats obtenus par notre étude comparative.

Chapitre I: Généralités sur les réseaux de capteurs sans fil

I.1 Introduction :

Au cours des dernières décennies, nous avons assisté à une miniaturisation du matériel informatique. Cette tendance à la miniaturisation a apporté une nouvelle génération de réseaux informatiques et télécoms présentant des défis importants. Les réseaux de capteurs sans fil (RCSF) sont l'une des technologies visant à résoudre les problèmes de cette nouvelle ère de l'informatique embarquée et omniprésente. De très nombreux états de l'art ont été proposés, nous allons donc dans le présent chapitre présenter les réseaux de capteurs sans fil, leurs architectures de communication et leurs applications.

Nous allons discuter également les principaux facteurs et contraintes qui influencent la conception des réseaux de capteurs sans fil.

I.2 Un nœud capteur :

I.2.1 Définition:

Un nœud capteur est un dispositif équipé de fonctionnalités de sensation avancées. Il mesure ou détecte un événement réel, comme le mouvement, la chaleur ou la lumière et convertit la valeur mesurée dans une représentation analogique ou numérique. Il prélève des informations et élabore à partir d'une grandeur physique (information d'entrée), une autre grandeur physique de nature électrique.

I.2.2 Architecture d'un nœud capteur : [A01]

Un nœud capteur est composé de plusieurs éléments ou modules correspondant chacun à une tâche particulière d'acquisition, de traitement, ou de transmission de données. Il comprend également une source d'énergie.

- ✓ **Unité de captage (*Sensing unit*)** : elle est composée de deux sous unités, un dispositif de capture physique qui prélève l'information de l'environnement local et un convertisseur analogique/numérique appelé CAN. [A02]
- ✓ **l'unité de traitement** : composée d'un processeur et d'une mémoire intégrant un système d'exploitation spécifique (TinyOS, par exemple). Cette unité possède deux interfaces, une interface pour l'unité d'acquisition et une interface pour l'unité de communication. Elle acquiert les informations en provenance de l'unité d'acquisition et les envoie à l'unité de communication. Cette unité est chargée aussi d'exécuter les protocoles de communications qui permettent de faire collaborer le capteur avec d'autres capteurs. Elle peut aussi analyser les données captées. [A03] [A04] [A02]
- ✓ **Unité de communication (*Transceiver unit*)** : elle est composée d'un émetteur/récepteur (module radio) permettant la communication entre les différents nœuds du réseau. [A02]

- ✓ **Unité d'énergie (*Power unit*)** : c'est la batterie qui, n'est généralement ni rechargeable ni remplaçable. La capacité d'énergie limitée au niveau des capteurs représente la contrainte principale lors de la conception de protocoles pour les réseaux de capteurs. [A05]

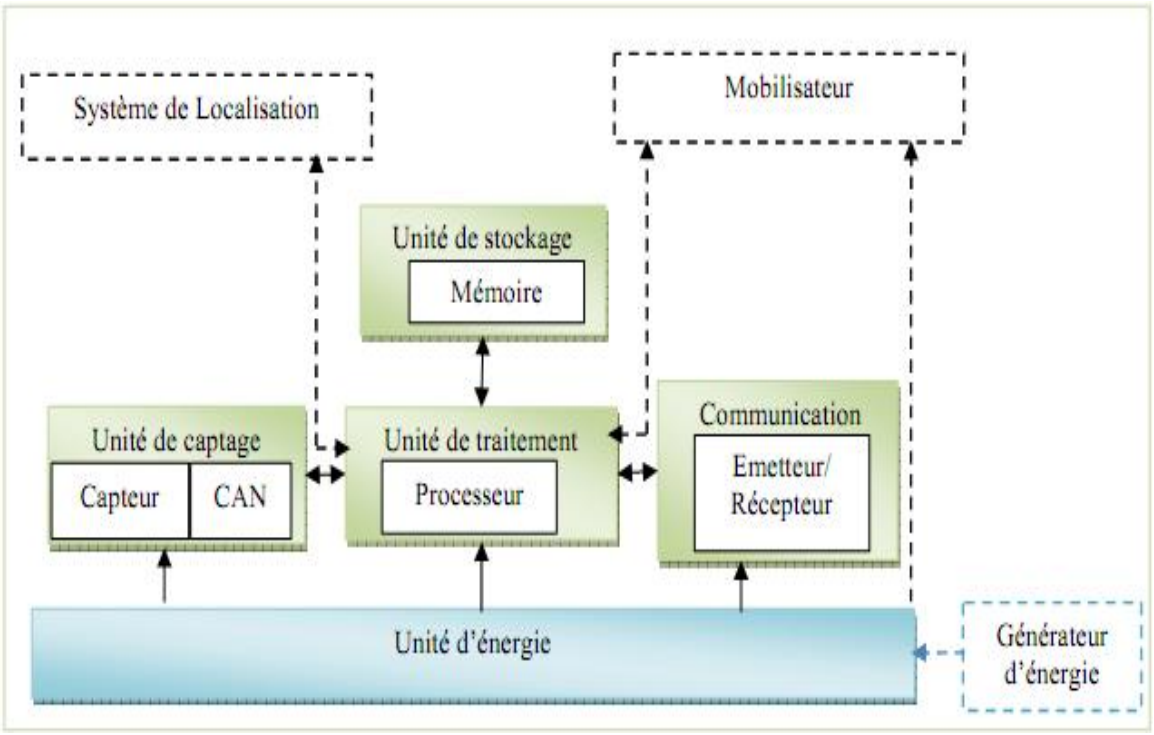

Figure I.1 : Architecture physique d'un nœud capteur

I.3 Les réseaux de capteurs sans fil

I.3.1 Définition :

Les réseaux de capteurs sans fil (RCSF) sont un type particulier des réseaux Ad-hoc, dans lesquels les nœuds sont des « capteurs intelligents ». Ils se composent généralement d'un grand nombre de capteurs communicants entre eux via des liens radio pour le partage d'information et le traitement coopératif. Dans ce type de réseau, les capteurs échangent des informations par exemple sur l'environnement pour construire une vue globale de la région contrôlée, qui est rendue accessible à l'utilisateur externe par un ou plusieurs nœud(s). Les données collectées par ces capteurs sont acheminées directement ou via les autres capteurs de proche en proche à un « point de collecte », appelé station de base (ou SINK s'il s'agit d'un nœud). Cette dernière peut être connectée à une machine puissante via internet ou par satellite.

En outre, l'utilisateur peut adresser ses requêtes aux capteurs en précisant l'information d'intérêt.

I.3.2 Architecture d'un réseau de capteurs sans fil :

Un réseau de capteurs sans fil est composé d'un grand nombre de nœuds. Chaque capteur est doté d'un module d'acquisition qui lui permet de mesurer des informations environnementales: température, humidité, pression, accélération, sons, image, vidéo etc.

Les données collectées par ces nœuds capteurs sont routées vers une ou plusieurs stations de base ou nœud puits (**Sink** en anglais*)*. Ce dernier est un point de collecte de données capturées. Il peut communiquer les données collectées à l'utilisateur final à travers un réseau de communication, éventuellement l'Internet ou un satellite. L'utilisateur peut à son tour utiliser la station de base comme passerelle, afin de transmettre ses requêtes au réseau.

En général, un RCSF est composé de quatre éléments montrés par la figure suivante :

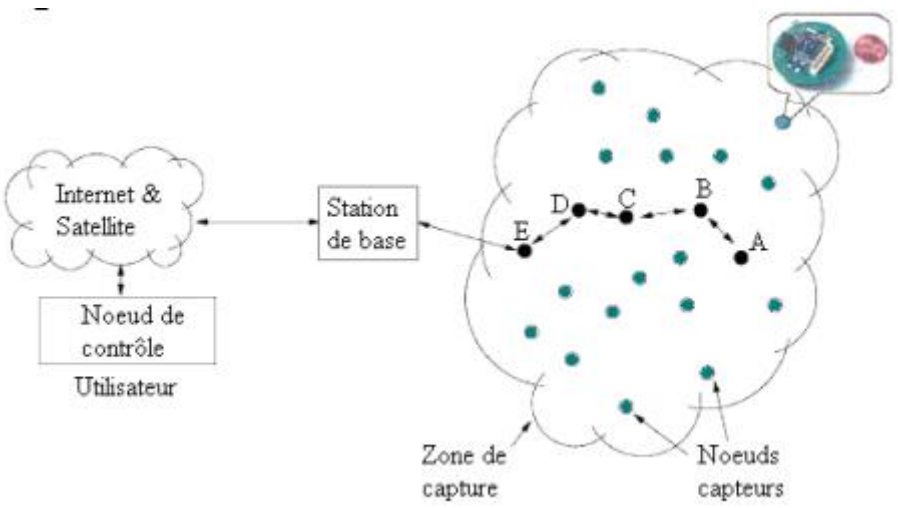

Figure I.2 : Architecture d'un réseau de capteurs sans fil

I.4 La pile protocolaire dans les réseaux de capteurs sans fil :

Dans les réseaux de capteurs sans fil, les nœuds capteurs utilisent une pile protocolaire spécifique en trois dimensions composée de cinq couches et de trois niveaux (niveau de gestion d'énergie, niveau de gestion de la mobilité, niveau de gestion des tâches). Les trois niveaux sont accessibles par toutes les couches de la pile. Il existe une interaction entre les différentes couches et les différents niveaux.

I.4.1 Les couches de la pile protocolaire : [A06] [A07]

Les cinq couches de la pile protocolaire sont les suivantes:

- ✓ **La couche physique**

  Cette couche décrit les procédures et les fonctions mécaniques et électriques nécessaires pour établir, maintenir et libérer une connexion physique entre deux ou plusieurs capteurs. Elle est responsable de la sélection des fréquences (utilisation des bandes ISM), de la détection d'un signal, et le traitement du signal (la modulation).

- ✓ **La couche liaison de données**

  Cette couche a pour rôle le multiplexage des flux de données, le partage de l'accès au medium et le contrôle d'erreur. Elle assure une connexion point à point ou point à multipoint fiable dans une communication réseau.

  La couche liaison de données est composée de deux couches : la couche LLC (Contrôle des Liens Logiques) et la couche MAC (Contrôle d'accès au medium).

  Les protocoles de la sous couche MAC sont appelés à effectuer d'importantes opérations : établir des liens de communication entre les nœuds capteurs voisins, fournir une fiabilité entre ces nœuds voisins et partager équitablement les canaux de communication entre les nœuds du réseau.

- ✓ **La couche réseau**

  Cette couche est responsable des fonctions de routage et de la gestion de la topologie.

- ✓ **La couche transport**

  Cette couche est responsable du maintien des flux de données dans les applications utilisées et de la sauvegarde des données dans le cache des capteurs. Elle est particulièrement nécessaire pour accéder au réseau de capteurs par le biais d'un réseau externe comme l'internet.

- ✓ **La couche application**

  Cette couche est conçue suivant les fonctionnalités des capteurs. Elle doit fournir des mécanismes :

  - pour l'interprétation des données perçues.
  - permettant à l'utilisateur d'interagir avec le réseau de capteurs.
  - qui rendent transparents à l'utilisateur les logiciels utilisés dans les couches inférieures.
  - d'agrégation de données.
  - de synchronisation des nœuds.
  - d'authentification, de distribution de clés dans le but d'assurer la sécurité des données.

I.4.2 Les niveaux de gestion dans les réseaux de capteurs sans fil :

Les niveaux de gestion propres aux réseaux de capteurs sans fil sont les suivants :

- Le niveau de gestion d'énergie : gère l'énergie consommée par les capteurs.
- Le niveau de gestion de la mobilité : ce niveau détecte et enregistre le mouvement des nœuds capteurs et permet de maintenir l'itinéraire d'un capteur vers un utilisateur et garder la trace de l'emplacement de ses voisins.
- Le niveau de gestion des tâches : s'occupe de la distribution des tâches pour une région donnée.

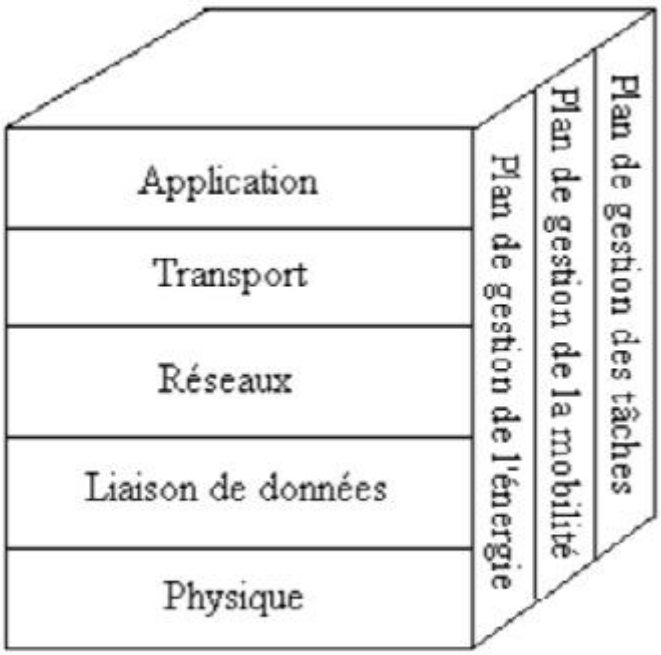

Figure I.3 : La pile protocolaire dans les RCSF

I.5 Caractéristiques des réseaux de capteurs sans fil : [A08]

Un réseau de capteurs présente les caractéristiques suivantes :

- **absence d'infrastructure :** les réseaux Ad-hoc en général et les réseaux de capteurs en particulier se distinguent des autres réseaux par la propriété d'absence d'infrastructure préexistante et de tout genre d'administration centralisée.
- **taille importante :** un réseau de capteurs peut contenir des milliers de nœuds.
- **interférences :** les liens radio ne sont pas isolés, deux transmissions simultanées sur une même fréquence, ou utilisant des fréquences proches, peuvent interférer.

- **topologie dynamique :** les capteurs peuvent être attachés à des objets mobiles qui se déplacent d'une façon libre et arbitraire rendant ainsi la topologie du réseau fréquemment changeante.
- **sécurité physique limitée :** les réseaux de capteurs sans fil sont plus touchés par le paramètre de sécurité que les réseaux filaires classiques. Cela se justifie par les contraintes et limitations physiques qui font que le contrôle des données transférées doit être minimisé.
- **bande passante limitée :** une des caractéristiques primordiales des réseaux basés sur la communication sans fil est l'utilisation d'un médium de communication partagé. Ce partage fait que la bande passante réservée à un nœud est limitée.
- **contrainte d'énergie, de stockage et de calcul :** la caractéristique la plus critique dans les réseaux de capteurs est la modestie de ses ressources énergétiques car chaque capteur du réseau possède de faibles ressources en termes d'énergie (batterie). Afin de prolonger la durée de vie du réseau, une minimisation des dépenses énergétiques est exigée chez chaque nœud. Ainsi, la capacité de stockage et la puissance de calcul sont limitées dans un capteur.

I.6 Les contraintes des réseaux de capteurs sans fil :

Les principales contraintes des réseaux de capteurs sans fil sont : la bande passante de transmission limitée et la puissance de transmission limitée.

I.6.1 Les contraintes liées aux medias de transmission :

La consommation d'énergie est un point crucial dans les réseaux de capteurs sans fil. Les circuits de communication et les antennes sont les éléments qui consomment le plus d'énergie.

I.6.2 Les contraintes liées aux MAC :

La perte d'énergie au niveau de la couche MAC est causée par plusieurs facteurs, notamment:

- Les collisions qui sont inévitables lorsque le canal de transmission est partagé de façon distribuée. Les paquets de données victimes de collisions doivent être retransmis ce qui implique une consommation d'énergie supplémentaire.
- La plupart des protocoles MAC distribués nécessitent un contrôle de message pour une transmission fiable des données (RTS/CTS dans IEEE 802.11). Ce type de contrôle de message consomme de l'énergie.
- L'overhearing et l'idel listennig consomment aussi de l'énergie.

- Overhearing : un nœud reçoit un paquet qui ne lui est pas destiné.
- Idle listening : un nœud écoute le canal pour connaitre son état (occupé ou libre).

I.7 Les différents facteurs de conception : [A09] [A10] [A11]

La conception des réseaux de capteurs est influencée par de nombreux facteurs comme la tolérance aux pannes, les coûts de production, la consommation d'énergie, l'environnement ou la topologie du réseau. Ces facteurs représentent la base de la conception des protocoles ou d'algorithmes pour les réseaux de capteurs.

Nous allons introduire les facteurs les plus importants à prendre en compte dans la conception d'un réseau de capteurs sans fil. Ces facteurs sont un guide dans la conception et l'implémentation des protocoles des différentes couches protocolaires propres aux réseaux de capteurs sans fil.

- ✓ **Tolérance aux pannes** : Les nœuds peuvent être sujets à des pannes dues à leur fabrication (ce sont des produits de série bon marché, il peut donc y avoir des capteurs défectueux) ou plus fréquemment à un manque d'énergie. Les interactions externes (chocs, interférences) peuvent aussi être la cause des dysfonctionnements. Afin que les pannes n'affectent pas la tâche première du réseau, il faut évaluer la capacité du réseau à fonctionner sans interruption.
- ✓ **Coût de fabrication** : Les nœuds sont des produits fabriqués en série du fait de leur grand nombre. Il faut que le coût de fabrication de ces nœuds soit tel que le coût global du réseau ne soit pas supérieur à celui d'un réseau classique afin de pouvoir justifier son intérêt.
- ✓ **Topologie du réseau** : En raison de leur forte densité dans la zone à observer, il faut que les nœuds capteurs soient capables d'adapter leur fonctionnement afin de maintenir la topologie souhaitée.

  On distingue généralement trois phases dans la mise en place et l'évolution d'un réseau :

  - Déploiement : Les nœuds sont soit répartis de manière prédéfinie soit de manière aléatoire (lancés en masse depuis un avion). Il faut alors que ceux-ci s'organisent de manière autonome.
  - Post-Déploiement -Exploitation : Durant la phase d'exploitation, la topologie du réseau peut être soumise à des changements dus à des modifications de la position des nœuds ou bien à des pannes.

- Redéploiement : L'ajout de nouveaux capteurs dans un réseau existant implique aussi une remise à jour de la topologie.

✓ **Consommation d'énergie** : L'économie d'énergie est une des problématiques majeures dans les réseaux de capteurs. En effet, la recharge des sources d'énergie est souvent trop coûteuse et parfois impossible. Il faut donc que les capteurs économisent au maximum l'énergie afin de pouvoir fonctionner.

✓ **Durée de vie du réseau**

C'est l'intervalle de temps qui sépare l'instant de déploiement du réseau de l'instant où l'énergie du premier nœud s'épuise. Selon l'application, la durée de vie exigée pour un réseau peut varier entre quelques heures et plusieurs années.

✓ **Ressources limitées**

En plus de l'énergie, les nœuds capteurs ont aussi une capacité de traitement et de mémoire limitée. En effet, les industriels veulent mettre en œuvre des capteurs simples, petits et peu coûteux qui peuvent être achetés en masse.

✓ **Bande passante limitée**

Afin de minimiser l'énergie consommée lors de transfert de données entre les nœuds, les capteurs opèrent à bas débit. Typiquement, le débit utilisé est de quelques dizaines de Kb/s.

Un débit de transmission réduit n'est pas handicapant pour un réseau de capteurs où les fréquences de transmission ne sont pas importantes.

✓ **Facteur d'échelle**

Le nombre de nœuds déployés pour une application peut atteindre des milliers. Dans ce cas, le réseau doit fonctionner avec des densités de capteurs très grandes. Un nombre aussi important de nœuds engendre beaucoup de transmissions inter nodales et nécessite que la station de base soit équipée de mémoire suffisante pour stocker les informations reçues.

✓ **Topologie dynamique**

La topologie des réseaux de capteurs peut changer au cours du temps pour les raisons suivantes **:**

- Les nœuds capteurs peuvent être déployés dans des environnements hostiles (champ de bataille par exemple), la défaillance d'un nœud capteur est, donc très probable.
- Un nœud capteur peut devenir non opérationnel à cause de l'expiration de son énergie.

- Dans certaines applications, les nœuds capteurs et les stations de base sont mobiles.

✓ **Agrégation de données**

Dans les réseaux de capteurs, les données produites par les nœuds capteurs voisins sont très corrélées spatialement et temporellement. Ceci peut engendrer la réception par la station de base d'informations redondantes. Réduire la quantité d'informations redondantes transmises par les capteurs permet de réduire la consommation d'énergie dans le réseau et ainsi d'améliorer sa durée de vie. L'une des techniques utilisée pour réduire la transmission d'informations redondantes est l'agrégation des données. Avec cette technique, les nœuds intermédiaires agrègent l'information reçue de plusieurs sources. Cette technique est connue aussi sous le nom de fusion de données.

✓ **Les contraintes matérielles**

Un capteur est composé de quatre unités de base et de composants optionnels qui dépendent de l'application. Certaines techniques de routage et tâches de perception nécessitent des informations sur la localisation avec une grande exactitude. Par conséquent un capteur doit être doté d'un système de localisation. Un mobilisateur peut être nécessaire pour déplacer des nœuds afin d'effectuer certaines tâches. Toutes ces sous unités occupent de la place dans un dispositif dont la taille peut être inférieure à 1 cm3. Ajouter à cela le fait que l'autonomie de la batterie des capteurs est très limitée. Compte tenu de tous ces facteurs, les capteurs doivent être disponibles, autonomes et doivent s'adapter à l'environnement.

✓ **L'environnement**

Généralement les nœuds sont déployés dans des zones géographiques lointaines pour mesurer et contrôler un phénomène. Le champ de captage peut être un océan, un champ contaminé par des produits chimiques ou biologiques, un champ de bataille, un domicile ou encore un building. Généralement un champ de captage est un environnement inaccessible.

✓ **Le media de transmission**

Dans un réseau de capteurs, les nœuds sont interconnectés à travers une interface de communication sans fil : une liaison radio, infrarouge ou optique. Pour réaliser les différentes opérations d'un réseau, le media de transmission choisi doit être fiable. La plupart des réseaux de capteurs utilisent les radios fréquences pour communiquer.

I.8 Domaines d'applications des réseaux de capteurs

La miniaturisation, l'adaptabilité, le faible coût et l'avancement dans les communications sans fil permettent aux réseaux de capteurs d'envahir plusieurs domaines d'applications. Ils permettent aussi d'étendre le domaine des applications existantes. Parmi ces domaines où ces réseaux se révèlent très utiles et peuvent offrir de meilleures contributions, on peut noter le militaire, la santé, l'environnemental, et les maisons intelligentes ...

- ✓ **Surveillance militaire et traque de cibles** [A12]

Comme beaucoup d'autres technologies de l'information, les réseaux de capteurs sans fil proviennent principalement de la recherche militaire. Des réseaux de capteurs autonomes sont envisagés comme l'ingrédient essentiel dans cette lancée vers des systèmes de guerre centrés sur les réseaux. Ils peuvent être rapidement déployés et utilisés pour la surveillance des champs de bataille afin de fournir des renseignements concernant l'emplacement, le nombre, le mouvement, et l'identité des soldats et des véhicules, ou bien encore pour la détection des agents chimiques, biologiques et nucléaires.

- ✓ **Applications à la sécurité :** [A13]

L'application des réseaux de capteurs dans le domaine de la sécurité peut diminuer considérablement les dépenses financières consacrées à la sécurisation des lieux et des êtres humains. Ainsi, l'intégration des capteurs dans de grandes structures telles que les ponts ou les bâtiments aidera à détecter les fissures et les altérations dans la structure suite à un séisme ou au vieillissement de la structure. Le déploiement d'un réseau de capteurs de mouvement peut constituer un système d'alarme qui servira à détecter les intrusions dans une zone de surveillance.

- ✓ **Applications environnementales :** [A08]

Le contrôle des paramètres environnementaux par les réseaux de capteurs peut donner naissance à plusieurs applications. Par exemple, le déploiement des thermo-capteurs dans une forêt peut aider à détecter un éventuel début de feu et par suite faciliter la lutte contre les feux de forêt avant leur propagation. Le déploiement des capteurs chimiques dans les milieux urbains peut aider à détecter la pollution et analyser la qualité d'air. De même leur déploiement dans les sites industriels empêche les risques industriels tels que la fuite de produits toxiques (gaz, produits chimiques, éléments radioactifs, pétrole, etc.).

Dans le domaine de l'agriculture, les capteurs peuvent être utilisés pour réagir aux changements climatiques par exemple le processus d'irrigation lors de la détection de zones sèches dans un champ agricole.

- ✓ **Applications médicales :** [A14]

Dans le domaine de la médecine, les réseaux de capteurs peuvent être utilisés pour assurer une surveillance permanente des organes vitaux de l'être humain grâce à des micro capteurs qui pourront être avalés ou implantés sous la peau (surveillance de la glycémie, détection de cancers, ...). Ils peuvent aussi faciliter le diagnostic de quelques maladies en effectuant des mesures physiologiques telles que : la tension artérielle, battements du cœur, ... à l'aide des capteurs ayant chacun une tâche bien particulière. Les données physiologiques collectées par les capteurs peuvent être stockées pendant une longue durée pour le suivi d'un patient. D'autre part, ces réseaux peuvent détecter des comportements anormaux (chute d'un lit, choc, cri, ...) chez les personnes dépendantes (handicapées ou âgées).

- ✓ **La domestique :** [A15]

Avec le développement technologique, les capteurs peuvent être embarqués dans des appareils, tels que les aspirateurs, les fours à micro-ondes, les réfrigérateurs,... . Ces capteurs embarqués peuvent interagir entre eux et avec un réseau externe via internet pour permettre à un utilisateur de contrôler les appareils domestiques localement ou à distance.

Le déploiement des capteurs de mouvement et de température dans les futures maisons dites intelligentes permet d'automatiser plusieurs opérations domestiques telles que : la lumière s'éteint et la musique se met en état d'arrêt quand la chambre est vide, la climatisation et le chauffage s'ajustent selon les points multiples de mesure, le déclenchement d'une alarme par le capteur anti-intrusion quand un intrus veut accéder à la maison.

- ✓ **Applications commerciales :**

Des nœuds capteurs pourraient améliorer le processus de stockage et de livraison. Le réseau ainsi formé, pourra être utilisé pour connaître la position, l'état et la direction d'un paquet ou d'une cargaison. Un client attendant un paquet peut alors avoir un avis de livraison en temps réel et connaître la position du paquet. Des entreprises manufacturières, via des réseaux de capteurs pourraient suivre le procédé de production à partir des matières premières jusqu'au produit final livré. Grâce aux réseaux de capteurs, les entreprises pourraient offrir une meilleure qualité de service tout en réduisant leurs coûts. Les produits en fin de vie pourraient être mieux démontés et recyclés ou réutilisés si les micro-capteurs en garantissent le bon état. Dans les immeubles, le système de climatisation peut être conçu en intégrant plusieurs micro-capteurs dans les tuiles du plancher et les meubles. Ainsi, La climatisation pourra être déclenchée seulement aux endroits où il y a des personnes présentes et seulement si c'est nécessaire.

I.9 Enjeux particuliers dans les réseaux de capteurs : [A08]

Les deux enjeux fondamentaux dans les réseaux de capteurs : le routage et la structuration des réseaux.

Le routage permet l'acheminement des informations vers une destination donnée à travers un réseau de connexion. Le problème de routage consiste à déterminer un acheminement optimal des paquets à travers le réseau au sens d'un certain critère de performance comme la consommation énergétique. Le problème consiste à trouver l'investissement de moindre coût qui assure le routage du trafic nominal et garantit la qualité de service.

Le problème qui se pose dans le contexte des réseaux de capteurs est l'adaptation de la méthode d'acheminement utilisée avec le grand nombre de nœuds existant dans un environnement caractérisé par de changements de topologies, de modestes capacités de calcul, de sauvegarde, et d'énergie.

Toute conception de protocole de routage implique l'étude des problèmes suivants :

- Minimiser la charge du réseau en optimisant le nombre d'envois et de réceptions des paquets. Cette minimisation aboutit à une consommation énergétique minimale et une longue durée de vie du réseau.
- Offrir un support pour pouvoir effectuer des communications multi-sauts fiables.
- Assurer un routage optimal si possible.
- Offrir une bonne qualité concernant les temps de latence.
- Auto organiser le réseau, ceci peut être nécessaire dans plusieurs cas.

Un réseau comportant un grand nombre de nœuds placés dans des endroits hostiles où la configuration manuelle n'est pas faisable doit être capable de s'auto-organiser. Un autre cas est celui où un nœud est inséré ou retiré (à cause d'un manque d'énergie ou destruction physique). Ainsi le réseau doit être capable de se reconfigurer pour continuer son fonctionnement.

I.10 Le système d'exploitation TinyOS : [A16]

Le système d'exploitation TinyOS[C02] s'appuie sur le langage NesC. Celui-ci propose une architecture basée sur des composants, permettant de réduire considérablement la taille mémoire du système et de ses applications.

Figure I.4: Sigle de TinyOS

Chaque composant correspond à un élément matériel (LEDs, timer,...) et peut être réutilisé dans différentes applications. Ces applications sont des ensembles de composants réutilisables et portables associés dans un but précis. Les composants peuvent être des concepts abstraits ou bien des interfaces logicielles aux entrées-sorties matérielles de la cible étudiée (carte ou dispositif électronique)

Chaque composant est tout d'abord constitué d'un frame. Elle contient l'état interne du composant. Il s'agit d'un espace mémoire réserve de taille fixe permettant au composant de stocker les variables globales et les données qu'il utilise pour réaliser ses fonctionnalités. Il n'en existe qu'une seule par composant et celle-ci est allouée statiquement à la compilation.

L'implémentation de composants s'effectue en déclarant des taches, des commandes ou des événements qui permettent de faire appel aux fonctionnalités d'autres composants auxquels ils sont liées.

Les taches sont utilisées pour effectuer la plupart des blocs d'instruction d'une application.

Une commande permet l'exécution d'une fonctionnalité dans un autre composant.

Les événements permettent de faire le lien entre les interruptions matérielles (pression d'un bouton, changement d'état d'une entrée,...) et les couches logicielles que constituent les taches.

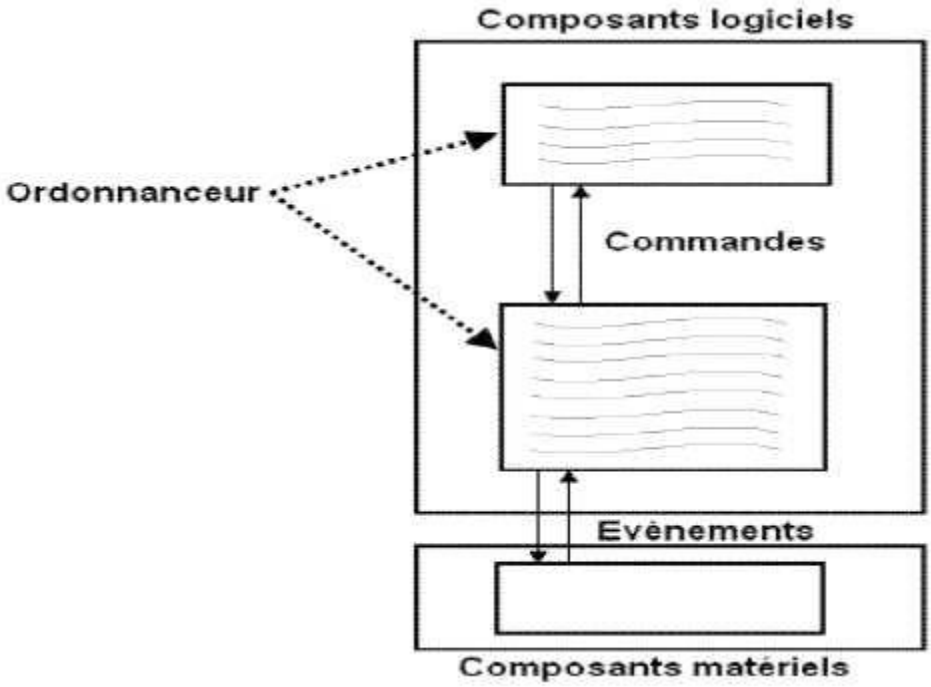

Figure I.5: Schéma des interactions internes au système

I.11Le langage de programmation NesC :

NesC est un langage de programmation orienté composants syntaxiquement proche du langage C. Il est conçu pour la réalisation des systèmes embarqués distribués, en particulier, les RCSF.

Il existe trois types de fichiers sources des applications NesC: les fichiers interfaces, les fichiers configurations et modules qui constituent les composants [CO3].

Les composants NesC présentent des similarités avec des objets. Les états sont encapsulés et on peut y accéder par des interfaces. En NesC, l'ensemble des composants et leurs interactions sont fixés à la compilation pour plus d'efficacité. Ce type de compilation permet d'optimiser l'application pour une exécution plus performante. En langage objet, cette phase est réalisée lors de l'exécution ce qui rend celle ci plus lente.

I.11.1 Développement :

Les interfaces spécifient un ensemble de fonctions, appelées commandes, qui doivent être implémentées par le fournisseur de l'interface et un ensemble de fonctions, appelées évènements, qui doivent être implémentées par l'utilisateur de l'interface. Afin de distinguer les fonctions concernant un évènement de celles concernant une commande, les en-têtes des fonctions sont précédés des mots-clés respectifs *event* ou *command*. Voici un exemple simple d'interface :

```
Interface Timer {
        command result_t start(char type, uint32_t interval);
        command result_t stop();
        event result_t fired();
}
```

De plus, comme il a été mentionné précédemment, le modèle mémoire fixé par TinyOS n'autorise pas les pointeurs de fonctions. Afin de néanmoins proposer un mécanisme alternatif, NesC utilise des interfaces paramétrées. Celles-ci permettent à l'utilisateur de créer un ensemble d'interfaces identiques et d'en sélectionner une seule à appeler grâce à un identifiant.

```
interface SendMsg [uint8 t id]
```

Dans la pratique, NesC permet de déclarer deux types de fichiers: les modules et les configurations.

Les configurations, permettent de décrire les composants composites c'est-à-dire des composants composés d'autres composants c'est à dire de granularité supérieure. Une configuration permet donc de décrire l'architecture. Une configuration est donc constituée de modules et/ou d'interfaces ainsi que de la description des liaisons entre ces composants.

```
configuration NomModule {
}
implementation {
        //liste des modules et configurations utilises, ex :
        components Main,Module1,…,ModuleN,Config1,…,ConfigM
        ;
        //descriptifs des liaisons
        //Interface fournie <- Interface requise ou
        //Interface requise -> interface fournie, ex :
        Main.StdControl -> Module1.StdControl ;
}
```

Détaillons ici, quelques caractéristiques concernant les configurations. La première d'entre elles concerne une simplification. Lors du descriptif des liaisons, il est en effet possible de ne pas préciser l'interface fournie par un module. Dans ce cas, elle possédera le même nom que celle requise. L'autre caractéristique concerne le composant Main. En effet, il est à noter que ce composant est obligatoirement présent dans la configuration décrivant l'ensemble de l'application car son rôle est de démarrer l'exécution de l'application.

Les modules sont eux les éléments de base de la programmation. Ils permettent en effet d'implémenter les composants et sont stockés dans un fichier possédant la structure suivante :

```
module NomModuleM {
provides {
//liste des interfaces fournies, ex :
interface NomInterfaceFournie1 ;
}
uses {
//liste des interfaces requises, ex :
interface NomInterfaceRequise1 ;
}
}
implementation {
//déclarations des variables, ex :
int state ;
//implementations des fonctions decrites par les
//interfaces fournies ;
}
```

I.11.2 Compilation :

La première étape de ce processus consiste à compiler les fichiers nécessaires à l'application et au système d'exploitation. Celle-ci est réalisée via le compilateur NesC fourni par TinyOs. Son rôle est premièrement de transformer les fichiers NesC en fichier C et deuxièmement d'y intégrer les fichiers du noyau de TinyOs. Ce qui permet d'obtenir un fichier source C unique. Une fois cette étape accomplie, il ne reste alors qu'à utiliser un compilateur C traditionnel qui va utiliser le fichier précédemment créé afin de générer une application exécutable. Celle-ci sera donc constituée par la « fusion » du système d'exploitation et du code applicatif. Ces différentes phases peuvent être synthétisées comme l'illustre la figure suivante :

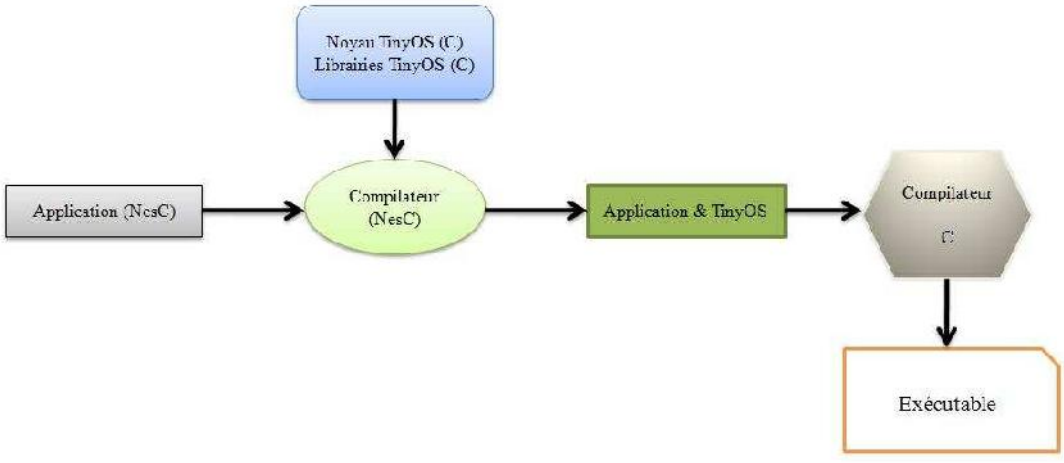

Figure I.6: Etapes de compilation d'une application sous TinyOS

I.11.3 Les architectures cibles de TinyOS :

Il existe de nombreuses cibles possibles pour ce système d'exploitation embarqué. Malgré leurs différences, elles respectent toutes globalement la même architecture basée sur un noyau central autour duquel s'articulent les différentes interfaces d'entrée/sortie, de communication et d'alimentation.

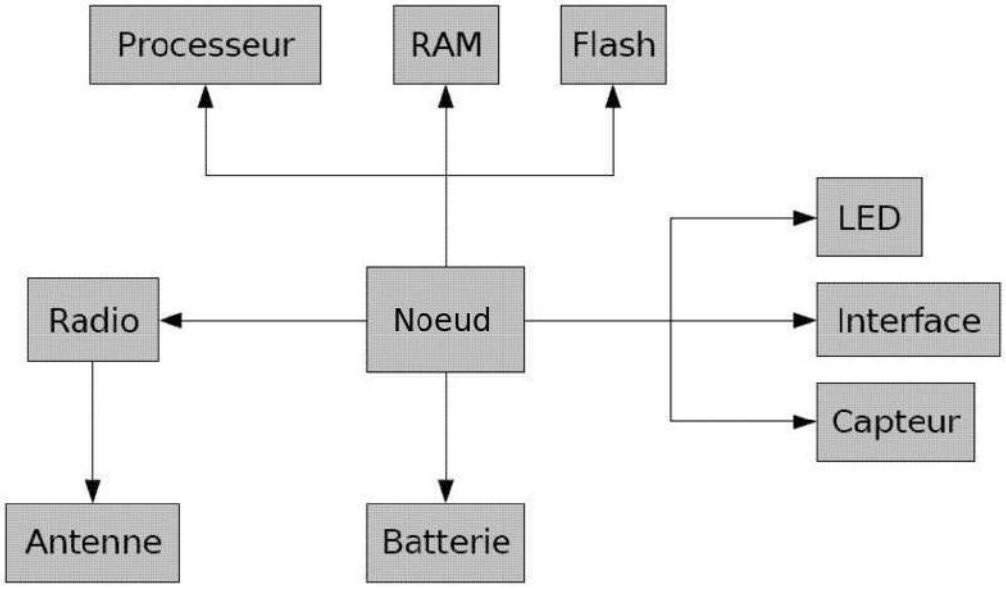

Figure I.7 : Schéma général d'une architecture cible pour TinyOS

On peut voir sur la figure 1.7 les différents composants qui constituent une architecture ciblée par TinyOS. Pour être plus précis, chaque groupe de composants possède son propre rôle :

- **Mote, processeur, RAM et Flash :** On appelle généralement Mote, la carte physique utilisant TinyOS pour fonctionner. Celle-ci a pour cœur le bloc constitué du processeur et des mémoires RAM et Flash. Cet ensemble est à la base du calcul binaire et du stockage, à la fois temporaire pour les données et définitif pour le système TinyOS.
- **Radio et antenne :** TinyOS est prévu pour mettre en place des réseaux sans fils, les équipements étudiés sont donc généralement équipés d'une radio ainsi que d'une antenne afin de se connecter à la couche physique que constitue les émissions hertziennes.

- **LED, interface, capteur :** TinyOS est prévu pour mettre en place des réseaux de capteurs, on retrouve donc des équipements bardés de différents types de détecteurs et autres entrées.
- **Batterie :** Comme tout dispositif embarqué, ils disposent d'une alimentation autonome telle qu'une batterie, et parfois d'un panneau solaire pour permettre de recharger cette batterie, ce qui lui permet d'être disposé dans un endroit parfois inaccessible.

Cependant quelques différences existent suivant les fabricants. Chacun d'eux développe son type de dispositif. Avec des contraintes hardware aussi strictes dues aux contraintes des systèmes embarqués, la partie software doit être la plus adaptée possible, d' où un lien très fort entre ces deux parties.

I.12 Conclusion

Les réseaux de capteurs sans fil présentent un intérêt considérable et une nouvelle étape dans l'évolution des technologies de l'information et de la communication. Cette nouvelle technologie suscite un intérêt croissant vu la diversité de ces applications : santé, environnement, industrie et même dans le domaine sportif.

Dans ce premier chapitre, nous avons présenté les réseaux de capteurs sans fil, leurs architectures de communication, la pile protocolaire des capteurs et leurs diverses applications. Cependant, nous avons remarqué que plusieurs facteurs et contraintes compliquent la gestion de ce type de réseaux. En effet, les réseaux de capteurs se caractérisent par une capacité énergétique limitée rendant l'optimisation de la consommation d'énergie dans des réseaux pareils une tâche critique pour prolonger la durée de vie du réseau.

Chapitre II: Le routage dans les RCSF

II.1 Introduction :

Un réseau de capteurs sans fil consiste en un ensemble de nœuds capteurs à grande échelle dans un champ de captage pour détecter, recueillir et transmettre les données concernant un phénomène observé, vers le nœud puits via les liens sans fil. Cependant, suivant le nombre de nœuds du réseau et l'étendu du champ de captage, certains nœuds ne pourront pas transmettre directement leurs messages au nœud collecteur. Ainsi, la collaboration entre les nœuds pour garantir cette transmission est une exigence. De cette manière, les messages sont propagés par les nœuds intermédiaires en établissant les chemins multi-sauts entre la source lointaine et le puits. Ce processus d'acheminement des messages d'un nœud source du réseau vers un nœud destinataire s'appelle le routage.

Le routage est considéré comme étant un service important; il permet l'acheminement des paquets de données captées à partir de nœuds capteurs vers la station de base. Le choix du chemin à prendre se fait selon certains critères de performance tels que la consommation en énergie, la fiabilité, le délai de transmission...etc.

Plusieurs travaux de recherche dans le domaine des RCSFs ont été effectués récemment et ont abouti à une multitude protocoles de routage destinés à ces réseaux.

Dans ce chapitre nous présentons un état de l'art sur le routage dans les réseaux de capteurs sans fil. Nous verrons plusieurs classifications des protocoles de routage dans les réseaux de capteurs sans fil, vu leur multiplicité, il nous semble important de commencer par leur classification qui les groupe suivant un certain nombre de critères.

II.2 Classification des protocoles de routage :

Pendant la conception d'un protocole de routage pour un réseau de capteurs sans fil, il est nécessaire de prendre en compte l'énergie et les ressources limitées des nœuds capteurs, la qualité du canal sans fil, la possibilité de perdre des paquets et la latence (temps d'attente).

L'efficacité en consommation d'énergie représente une métrique de performance significative qui influence directement la durée de vie du réseau en entier.

Plusieurs stratégies de routage ont été proposées pour les réseaux de capteurs sans fil. Leurs principes de fonctionnement diffèrent suivant la philosophie de la classe à laquelle ils appartiennent. On donne une classification qui se base sur quatre critères :

- la structure du réseau,
- le fonctionnement du protocole,

- le mode d'établissement des chemins,
- l'initiateur de communication.

II.2.1 Protocoles basés sur la structure du réseau :

II.2.1.1 Protocoles de routage hiérarchique :

Lorsque la taille du réseau devient de plus en plus importante, sa gestion devient plus difficile. Les protocoles de routage à plat fonctionnent bien quand le réseau ne comprend pas un grand nombre de nœuds. Une des structures les plus connues est la hiérarchie. La technique de hiérarchisation sert à partitionner le réseau en sous ensembles afin de faciliter la gestion du réseau surtout le routage. Dans ce type de protocoles, des nœuds spéciaux peuvent avoir des rôles supplémentaires. Nous distinguons deux types de groupes de nœuds : la zone et le cluster.

Un cluster est défini par un ensemble de nœuds et possède un nœud nommé nœud-chef ou Cluster Head (CH). Le rôle du CH est de faire le relais entre les nœuds du cluster et la station de base directement ou via d'autres CHs. Le CH possède généralement des ressources énergétiques supérieures aux autres nœuds du réseau. Cette technique est appelée clusterisation. Une zone est définie par un ensemble de nœuds mais ne possède pas un nœud-chef (ou CH). Ainsi, un cluster est une sous-classe d'une zone. Dans la suite nous citons quelques uns.

❖ **LEACH (Low Energy Adaptive Clustering Hierarchy) [B01]**

Le protocole LEACH est le plus populaire des protocoles de routage hiérarchique. Son principal avantage est de minimiser la consommation énergétique des éléments du réseau.

LEACH est un protocole conçu pour le routage dans les réseaux de capteurs homogènes où les capteurs ont les mêmes caractéristiques et les mêmes capacités.

Dans ce protocole le réseau est divisé en clusters et chaque cluster possède un nœud maitre appelé cluster-Head. Ce dernier prend en charge la gestion de son cluster, il communique directement avec la station de base ce qui permet de minimiser la consommation et réduire la quantité d'informations envoyées à la station de base. Le cluster-Head est élu périodiquement parmi les nœuds formant le cluster, en fonction de l'état de sa batterie.

Le protocole se déroule en rounds. Chaque round se compose de deux phases qui se

succèdent plusieurs fois: la phase initialisation et la phase transmission qui seront détaillées dans le chapitre suivant.

❖ ***PEGASIS (Power-Efficient Gathering in Sensor Information Systems)*[B02]**

PEGASIS est une version améliorée de LEACH. L'idée principale de PEGASIS est de former une chaîne entre les nœuds de sorte que chaque nœud communique à un voisin proche. Les données collectées sont transmises d'un nœud à un autre qui les agrège jusqu'à ce qu'elles arrivent à un nœud particulier qui les transmet à la station de base. Les nœuds qui transmettent les données à la station de base, sont choisis tour à tour selon une politique round-robin dans le but de réduire l'énergie moyenne dépensée par un nœud durant une période (round). Contrairement à LEACH, PEGASIS évite la formation des clusters et procure à un seul nœud dans la chaîne l'envoi de données à la station de base. D'ailleurs, PEGASIS suppose que les nœuds sont capables de modifier leur puissance de transmission.

Les résultats de simulation ont montré que PEGASIS peut prolonger de deux à trois fois la durée de vie d'un réseau de capteurs relativement à LEACH en fonction du critère choisi pour évaluer la durée de vie d'un réseau. Un tel gain de performance est réalisé par l'élimination du surcoût causé par le processus de formation de clusters dans LEACH, et par la réduction du nombre de transmissions et de réceptions en agrégeant les données.

Bien que le surcoût du clustering soit évité, PEGASIS exige toujours un ajustement dynamique de la topologie puisqu'un nœud devrait connaître le niveau d'énergie de ses voisins avant de relayer ses données. Cependant, un tel ajustement de la topologie pourrait causer un surcoût important. En outre, PEGASIS suppose que tout nœud communique directement avec la station de base qui gère la topologie d'une manière centralisée. Or, cette supposition est loin de la réalité car les capteurs communiquent généralement en mode multi-sauts pour atteindre la station de base. D'autre part, PEGASIS suppose que tous les nœuds maintiennent une table contenant les localisations de tous les autres nœuds dans le réseau. En résumé, PEGASIS est adapté seulement aux capteurs sans fil dont les nœuds sont immobiles. Son évaluation dans des environnements mobiles pourrait dégrader considérablement ses performances.

❖ **TEEN (Threshold sensitive Energy Efficient Network protocol) [B03]**

Le protocole TEEN a été développé à partir de LEACH mais au contraire de LEACH qui est orienté temps, TEEN est quant à lui orienté événement.

La différence majeure entre ces deux protocoles résulte dans le fait que TEEN n'utilise pas TDMA qui est inadapté pour les applications orientées événements. Les chefs de zones élus dans TEEN ne transmettent donc pas de *schedule TDMA* à leurs membres, mais un message qui contient la tâche demandée au capteur, la valeur critique après laquelle les membres doivent envoyer leurs rapports de données (Hard Threshold) et le changement minimal obligeant le nœud à envoyer un nouveau rapport (Soft Threshold).

Donc lorsqu'un nœud détecte que sa valeur captée a dépassé le HT, il doit émettre un rapport au chef de zone, mais uniquement si sa valeur a changé suffisamment (par rapport au ST).

II.2.1.2 *Protocoles de routage plat (Flat Routing)*

Dans les réseaux plats, chaque nœud joue généralement le même rôle. En raison du grand nombre de ces nœuds, il n'est pas possible d'attribuer un identifiant global à chaque nœud. Cette considération a conduit à un routage centré, où la station de base envoie des requêtes à certaines régions et attend les réponses par les capteurs situés dans les régions sélectionnées. Puisque les données sont demandées par le biais des requêtes, les attributs basés sur les noms sont nécessaires pour préciser les propriétés des données. Des travaux dans le routage centré, par exemple, SPIN et Directed Diffusion ont montré l'économie d'énergie à travers les données de négociation et l'élimination des donnés redondantes. Ces deux protocoles ont motivé la conception de nombreux autres protocoles qui suivent un concept similaire.

- **Le protocole Flooding : [B04]**

Dans le protocole Flooding (appelé aussi l'inondation), chaque nœud reçoit les messages (sous forme d'un paquet de données), ensuite il le diffuse dans le réseau. Ainsi ce protocole consiste à transmettre tous les nouveaux paquets reçus et qui ne lui sont pas destiné. 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 II.1, 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 dernier reçoit deux copies de même paquet. Ainsi on ne peut plus distinguer entre les paquets récents et les paquets vieux.

D'autre part, le problème d'overlap est illustré par la figure II.2, 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 terme 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.

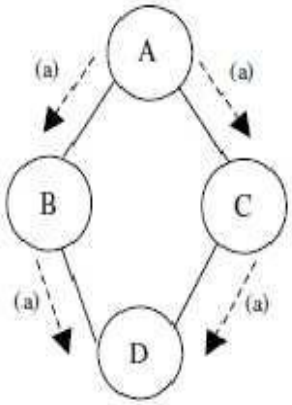

Figure II.1: le problème d'implosion

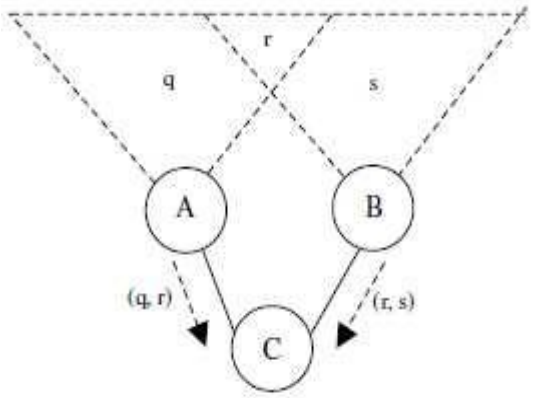

Figure II.2 : le problème d'overlap

❖ **Le protocole « Directed Diffusion » : [B05]**

Directed Diffusion est un protocole de routage pour les réseaux de capteurs.il est composé principalement de deux phases.

La première consiste à la phase de diffusion des messages d'intérêts. En effet, le Sink demande un service en envoyant les intérêts à tout le réseau (voir figure II.3 (a)). L'intérêt représente une tâche à accomplir par le réseau et il peut être destiné pour un ou plusieurs nœuds.

La deuxième phase présente la réaction d'un nœud suite à la réception d'un intérêt. D'abord, le nœud vérifie s'il est concerné par ce message, ensuite si oui il enregistre l'identité du nœud émetteur de l'intérêt en question (figure II.3(b)), afin de construire le gradient des routes menant vers le Sink. Si le nœud n'est pas destiné par l'intérêt, il continue à le propager à tous ces voisins. Une fois le message arrivé au destinataire, la route vers la station de base

est alors établie et ainsi le nœud cible choisi cette route pour envoyer les informations demandées (figure II.3 (c)). La tâche demandée par le message d'intérêt peut être générée périodique, par exemple du genre «envoi moi la température chaque heure ».

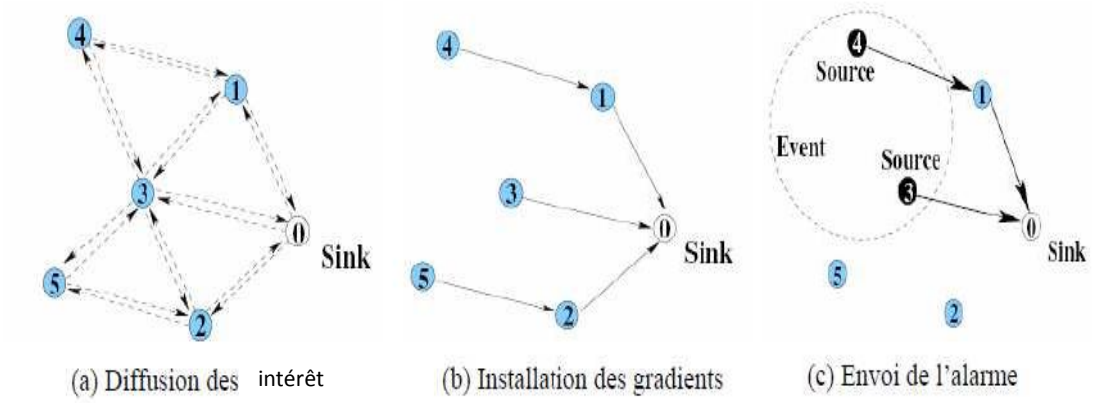

Figure II.3 : Les phases du protocole Directed Diffusion

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).

❖ SPIN (Sensor Protocols for Information via Negotiation): [B06][B21]

L'idée derrière le SPIN est de nommer les données en utilisant des descripteurs de haut niveau ou des métadonnées. Avant transmission, les métadonnées sont échangées entre les capteurs par un mécanisme de publicité de données. Chaque nœud recevant de nouvelles données, l'annonce à ses voisins et les voisins intéressés récupèrent les données en envoyant une requête. Cela permet à un utilisateur d'interroger n'importe quel nœud et d'obtenir immédiatement l'information demandée. Le fonctionnement du protocole SPIN permet de réduire la charge du réseau par rapport aux méthodes de diffusion traditionnelles telles que l'inondation ou l'algorithme de Gossiping.

Le protocole SPIN utilise essentiellement trois types de paquets ADV/REQ/DATA. Un nœud voulant émettre une donnée commence par envoyer un paquet ADV. Ce paquet ADV consiste d'une métadonnée sur les données à émettre. Les métadonnées peuvent décrire plusieurs aspects comme le type des données et la localisation de son origine. Les

nœuds qui reçoivent ce paquet vérifient si les données les intéressent. Si oui, ils répondent par un paquet REQ. Le nœud qui a initié la communication envoie alors un paquet DATA pour chaque réponse REQ reçue (voir la Figure II.4). Un nœud peut parfaitement ne pas répondre aux messages ADV, par exemple dans le but d'économiser son énergie. Ensuite chaque nœud qui fait office de relais peut très bien agréger ses propres donnés aux données qui sont déjà contenues dans le paquet.

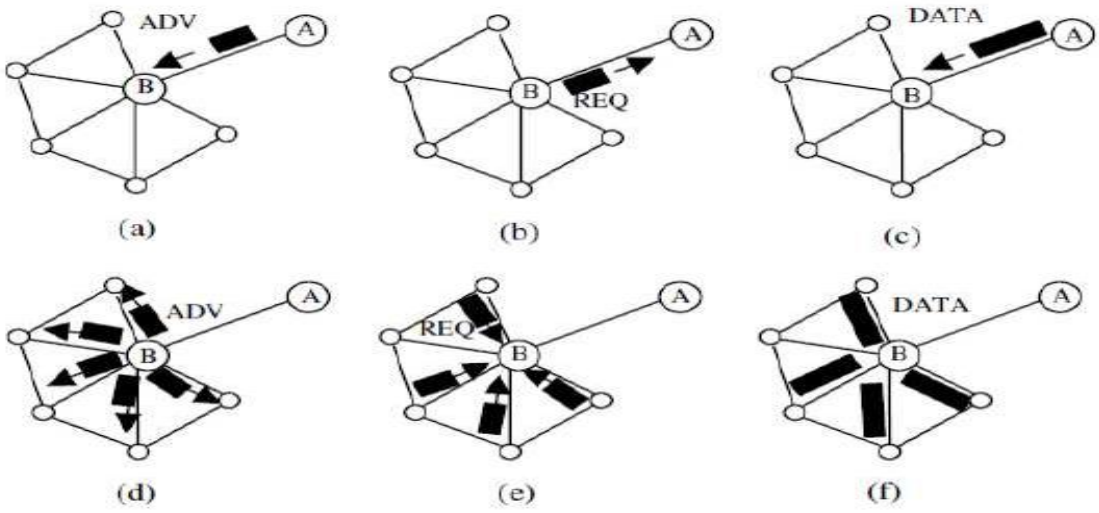

Figure II.4 : Fonctionnement du protocole SPIN

❖ **Energy Aware Routing (EAR) [BO7]**

Le problème avec les protocoles actuels est qu'ils trouvent le chemin optimal en énergie et l'utilisent pour toutes les communications. Cependant ce n'est pas la meilleure des méthodes pour prolonger la durée de fonctionnement d'un réseau. Utiliser fréquemment le même chemin optimal induit l'épuisement de la batterie des nœuds qui constituent ce chemin et par conséquent le réseau sera partitionné. EAR a été introduit pour contrer ce problème. Le principe de base est que, pour augmenter la durée de fonctionnement du réseau, il est nécessaire d'utiliser un sous ensemble de chemins optimaux occasionnellement. C'est une sorte d'équilibrage de charge dans le réseau. Pour assurer ce principe, plusieurs chemins sont découverts, des sources vers la destination (la station de base). Un chemin est choisi aléatoirement selon une probabilité.

C'est-à-dire qu'il n'y a pas de chemin qui soit utilisé fréquemment. EAR est un protocole réactif, la communication est initiée par la destination. La station de base initie le chemin et le maintient. Il est semblable à la diffusion dans certains cas. Plusieurs chemins sont maintenus entre la source et la station de base. Contrairement à Directed Diffusion qui envoie les données à travers tous les chemins à intervalle régulier, EAR utilise un seul chemin à chaque fois. Grace au choix probabiliste des chemins, il peut continuellement évaluer différents chemins et choisir les probabilités.

- **Gossiping**

C'est un mécanisme de diffusion plus souple que le flooding. Un nœud envoie un paquet de données à un nombre aléatoire de voisins. Les voisins font suivre le paquet de la même manière jusqu'à l'arrivée de l'information à la destination. Le Gossiping réduit les redondances de données mais l'arrivée des données à la destination est retardée.

II.2.1.3 Les protocoles basés sur la localisation

Ces protocoles utilisent la localisation pour adresser les nœuds capteurs. Le routage basé sur la localisation est utile dans les applications où la position des nœuds dans la couverture géographique du réseau est nécessaire pour qu'un nœud source puisse répondre à une requête. Une telle requête peut indiquer une zone géographique où se produit un phénomène intéressant (du point de vue de l'utilisateur) .

C'est aux nœuds de déterminer les endroits approximatifs des autres nœuds. Ce type de topologie est mieux adapté aux réseaux avec une forte mobilité.

Avant d'envoyer ses données à un nœud destination, le nœud source utilise un mécanisme pour déterminer la localisation de la destination puis inclus l'identifiant de la zone de localisation et du nœud destination dans l'entête du paquet à envoyer.

À titre d'exemple des protocoles utilisant une topologie basée sur la localisation, nous pouvons citer GEAR (Geographic and Energy Aware Routing) et LAR (Location-Aided Routing protocol).

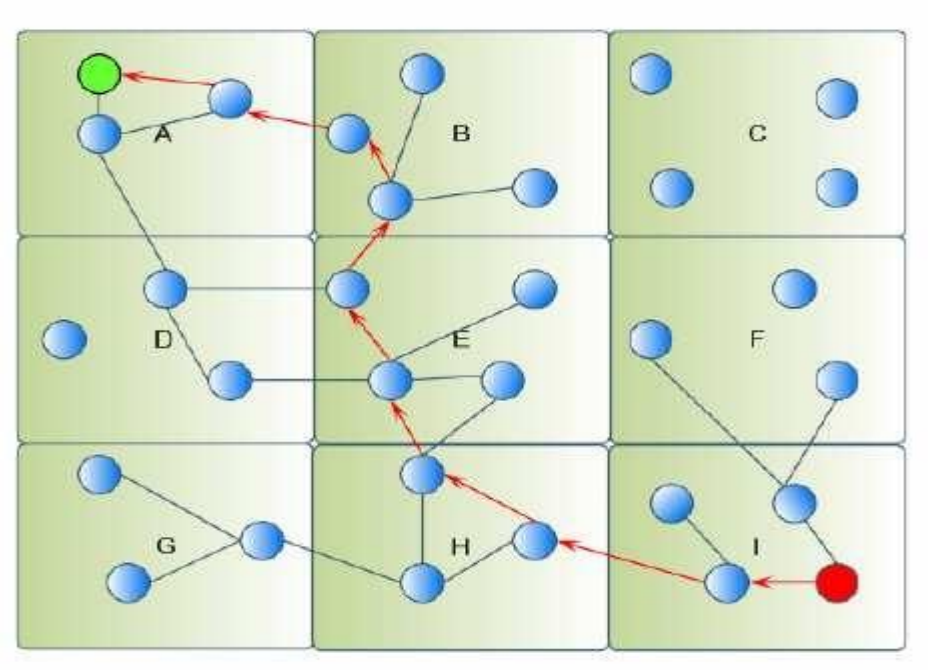

Figure II.5 : Topologie basée sur la localisation

❖ ***GEAR* (Geographic and Energy Aware Routing):[B09][B10]**

GEAR suggère l'utilisation d'information géographique tout en disséminant les demandes des données vers les régions appropriées plutôt que d'envoyer les intérêts à tout le réseau puisque ces demandes incluent souvent des attributs géographiques concernant une certaine région. GEAR se comporte comme un protocole de routage sur demande. Dans le cas de ce type de protocoles, la découverte de la route est basée sur la demande. Ainsi il n'y a aucun maintien de table de routage dans le nœud. La découverte de la route est lancée seulement quand un nœud veut transmettre des données, aussi aucune mise à jour périodique ou le maintien des tables de routage n'est effectué.

Chaque nœud garde un coût estimé et un coût instruit pour atteindre une région cible à travers ses voisins. Le coût estimé est une combinaison entre l'énergie et la distance résiduelle jusqu'à la destination. Le coût instruit est une amélioration du coût estimé qui prend compte d'autres paramètres de routage comme les trous dans le réseau. Un trou se produit quand un nœud n'a aucun voisin plus proche de la région cible que lui-même, alors le trou doit être contourné. S'il n'y a aucun trou, le coût estimé est égal au coût instruit. Il y a

deux phases dans l'algorithme de livraison des paquets.

La première phase c'est l'expédition des intérêts vers la région de cible: après avoir reçu un paquet, le nœud en question vérifie si ses voisins appartiennent à la région cible, s'il existe un, il est choisi comme le prochain saut. Si non, il vérifie ses voisins si l'un d'eux est plus proche de la région cible que lui même. Si il y en a plus, le voisin le plus proche de la région cible est choisi comme prochain saut. S'ils sont tous plus loin que le nœud lui-même, ceci signifie qu'il y a un trou. Dans ce cas, un des voisins est sélectionné aléatoirement pour expédier le paquet en se basant sur son coût instruit dans le but de contourner le trou. Ce choix peut être mis à jour selon la convergence du coût instruit pendant la livraison des paquets. La deuxième phase c'est l'expédition des paquets dans la région : Si le paquet a atteint la région, il peut être diffusé dans cette région par le Flooding géographique récursif ou le Flooding restreint. Le Flooding restreint est bon lorsqu'on n'a pas une grande densité de capteurs dans la région. Dans les réseaux à haute densité, le Flooding géographique récursif est plus efficace en termes d'énergie que le Flooding restreint.

Chaque nœud connaît sa propre localisation, son niveau d'énergie, la localisation de chacun de ses voisins et leurs niveaux d'énergie par l'échange des messages d'information (échange des messages Hello). Un chemin existe entre deux nœuds si chacun d'eux a un niveau d'énergie suffisant pour échanger les paquets entre eux (chaque nœud est dans la marge de transmission de l'autre), dans le cas contraire c'est un trou.

II.2.2 Selon le mode de fonctionnement du protocole : [B03]

Le mode de fonctionnement définit la manière avec laquelle les données sont propagées dans le réseau. Selon ce critère, les protocoles de routage peuvent être classifiés en quatre catégories : routage basé sur la qualité de services « OdS » (Quality of Service « QoS » based routing), routage basé sur les requêtes (query-based routing), routage multi-chemins (Multi-path roting), et routage basé sur la négociation (Negociation based routing) **[B22]**.

II.2.2.1 Protocoles basés sur la QoS

Dans ce type de protocoles, les performances du réseau sont prises en compte pour garantir un délai bout-en-bout raisonnable qui répond aux besoins de l'application. C'est surtout le cas des applications industrielles et militaires.

❖ **SAR (Sequential Assignment Routing)**

Il est basé sur la construction d'arbre à partir des voisins du puits. Les liens de chaque arbre sont choisis en fonction du délai observé et de l'énergie résiduelle des nœuds. Les données sont associées à un niveau de priorité. La création des arbres est assez lourde.

❖ **SPEED**

Une métrique supplémentaire par rapport à GEAR : le délai.

Se base sur une table de positions. Il estime le délai sur chaque saut en calculant le délai d'aller-retour (en retranchant le temps de traitement coté récepteur). Le prochain saut est choisi parmi les voisins qui sont plus proches de la destination que le nœud.

II.2.2.2 Routage basé sur les multi-chemins

Dans cette catégorie, les protocoles de routage utilisent des chemins multiples plutôt qu'un chemin simple afin d'augmenter la performance du réseau. La fiabilité d'un protocole peut être mesurée par sa capacité à trouver des chemins alternatifs entre la source et la destination en cas de défaillance du chemin primaire. Pour cette raison, certains protocoles construisent plusieurs chemins indépendants, c.à.d: ils ne partagent qu'un nombre réduit (voir nul) de nœuds. Malgré leur grande tolérance aux pannes, ces protocoles requièrent plus de ressources énergétiques et plus de message de contrôle.

II.2.2.3 Routage basé sur les requêtes :

Dans ce type de routage, le puits génère des requêtes afin d'interroger les capteurs.

Ces requêtes sont exprimées soit par un schéma valeur-attribut ou bien en utilisant un langage spécifique (par exemple SQL : Structured Query Language). Les nœuds qui détiennent les données requises doivent les envoyer au nœud demandeur à travers le chemin inverse de la requête. Les requêtes émises par le puits peuvent aussi être ciblées sur des régions spécifiques du réseau.

II.2.2.4 Routage basé sur la négociation :

En détectant le même phénomène, les nœuds capteurs inondent le réseau par les mêmes paquets de données. Ce problème de redondance peut être résolu en employant des protocoles de routage basés sur la négociation. En effet, avant de transmettre, les nœuds capteurs négocient entre eux leurs données en échangeant des paquets de signalisation spéciales, appelés métadonnées. Ces paquets permettent de vérifier si les nœuds voisins disposent déjà de la donné à transmettre. Cette procédure garantit que seules les informations utiles seront transmises et élimine la redondance des données.

II.2.3 Selon le mode d'établissement des chemins :

rfcSuivant la manière de création et de maintien des chemins pendant le routage, nous distinguons trois catégories de protocoles de routage : les protocoles proactifs, les protocoles réactifs et les protocoles hybrides **[B23]**.

II.2.3.1 Les protocoles proactifs

Utilisent l'échange régulier de messages de contrôle pour maintenir au niveau de chaque nœud des tables de routage (qui associent à chaque destination ou groupe de destinations un voisin direct par lequel les paquets doivent être relayés) vers toute destination atteignable depuis celui-ci. Ces tables sont maintenues mêmes quand les routes ne sont pas utilisées. Cette approche permet de disposer d'une route vers chaque destination immédiatement au moment ou un paquet doit être envoyé. Les protocoles proactifs sont adaptés aux applications qui nécessitent un prélèvement périodique de données. Et par conséquent, les capteurs peuvent se mettre en veille pendant les périodes d'inactivité, et n'enclencher leur dispositif de capture qu'à des instants particuliers.

Dans la suite de cette partie, nous présentons les principaux algorithmes de routage proactifs de la littérature.

- **Destination Sequenced Distance Vector (DSDV) : [B12]**

Le protocole DSDV est basé sur l'algorithme distribué de Bellman-Ford. Chaque nœud du réseau maintient dans sa table de routage un ensemble d'informations pour chaque destination contenant :

– l'adresse du destinataire.

– le nombre de sauts pour l'atteindre, c'est à dire le nombre de liens de communication existants pour atteindre ces destinations.

– un numéro de séquence associé au destinataire. Il est utilisé pour faire la distinction entre les anciennes routes et les nouvelles routes découvertes vers cette destination pour éviter la formation des boucles de routage.

- **Global State Routing (GSR) : [B13]**

Le protocole GSR est similaire au protocole DSDV décrit précédemment. Il est également basé sur les états de liens entre nœuds, c'est-à-dire sur la connaissance que chaque nœud possède de son voisinage. Il utilise ainsi une vue globale de la topologie du réseau. Chaque nœud *i* maintient une table de voisinage *Neighi*, une table de topologie *Topi*, construite par les états de liens de chaque nœud du réseau, une table des nœuds suivant *Nexti* et une table de distance *Disti*. La table de topologie *Topi* contient pour chaque destination *j* l'information de l'état de lien telle qu'elle a été envoyée par *j* et une estampille pour chaque information. Pour chaque nœud de destination *j*, la table *NEXTi* contient le nœud vers lequel les paquets destinés à *j* seront envoyés. Finalement, la table de distance *Di* contient la plus courte distance pour chaque nœud destination.

- **Wireless Routing Protocole (WRP) : [B14]**

Le protocole WRP est basé sur l'utilisation des algorithmes de recherche de chemins appelé *Path-Finding Algorithm* (PFA). Il en existe de nombreux dans la littérature et ils utilisent tous le principe suivant : chaque nœud a la connaissance du nombre minimum de liens qui le sépare de tous les autres nœuds. Ces algorithmes impliquent donc de connaître les distances entre chaque nœud du réseau pour pouvoir calculer les plus courts chemins.

- **Cluster head Gateway Switch Routing (CGSR): [B15]**

Le protocole CGSR est issu du protocole DSDV et est basé sur une architecture de réseau basée sur des groupes.

Chaque nœud du réseau est dans un groupe et est d'un des types suivant :

– Clusterhead : c'est le représentant du groupe, qui a pour voisin, tous les autres nœuds du groupe.

– Liaison : ce sont des nœuds communs à plusieurs groupes.

– Sans état : ces nœuds n'ont aucun statut particulier.

Le routage s'effectue de la manière suivante : le nœud source transmet ses paquets de données à son représentant de groupe. Le représentant envoie les paquets aux nœuds de liaison, qui relient ce représentant au représentant suivant dans le chemin qui existe vers la destination. Le processus se répète, jusqu'à ce que le représentant du groupe dans lequel appartient la destination soit atteint. Ce représentant transmet alors les paquets reçus vers le nœud destination.

Chaque nœud maintient deux tables : une table de membre de groupe qui associe à chaque nœud destination son clusterhead et une table de routage qui indique le prochain saut pour atteindre le groupe de destination. Chaque nœud diffuse cette table périodiquement et met la sienne à jour en fonction de celles qu'elle reçoit de la même manière que le fait DSDV. Ce routage est déterministe mais ne donne pas le chemin optimal.

II.2.3.2 Les protocoles réactifs :

Dits aussi les protocoles de routage à la demande, créent et maintiennent les routes selon les besoins. Lorsqu'un nœud a besoin d'une route, une procédure de découverte globale est lancée. Cette procédure s'achève par la découverte de la route ou lorsque toutes les permutations de routes possibles ont été examinées. La route trouvée est maintenue par une procédure de maintenance de routes jusqu'à ce que la destination soit inaccessible à partir du nœud source ou que le nœud source n'aura plus besoin de cette route.

- **Dynamic Source Routing (DSR) : [B16] [B17]**

Le protocole DSR est basé sur l'utilisation de la technique du routage par la source. Dans cette technique la source détermine la séquence complète des nœuds à travers lesquels les paquets de données seront envoyés. Avant d'envoyer un paquet de données vers un autre nœud l'émetteur diffuse un paquet « route request ». Si l'opération de découverte de routes est réussie, l'émetteur reçoit un paquet « route response » qui contient une séquence de nœuds à travers laquelle la destination peut être atteinte. Le paquet « route request » contient un champ d'enregistrement de routes, dans lequel sera accumulée la séquence de nœuds visités durant la propagation de la requête dans le réseau. L'utilisation de la technique de routage par la source fait que les nœuds de transit n'ont pas besoin de maintenir les informations de mise à jour pour envoyer les paquets de données, puisque ces derniers contiennent toutes les décisions de routage.

- **AODV (Ad-hoc On Demand Distance Vector) : [B18]**

AODV est un protocole à vecteur de distance, comme DSDV, mais il est réactif plutôt que proactif comme DSDV. En effet, AODV ne demande une route que lorsqu' il en a besoin.

AODV utilise les numéros de séquence d'une façon similaire à DSDV pour éviter les boucles de routage et pour indiquer la « nouveauté » des routes. Une entrée de la table de routage contient essentiellement l'adresse de la destination, l'adresse du nœud suivant, la distance en nombre de sauts (i.e. le nombre de nœuds nécessaires pour atteindre la destination), le numéro de séquence destination, le temps d'expiration de chaque entrée dans la table.

Lorsqu'un nœud a besoin de trouver une route vers une destination dont l'entrée dans la table de routage n'existe pas ou est expirée, il diffuse un message de demande de route (Route Request message, RREQ) à tous ses voisins. Le message RREQ est diffusé à travers le réseau jusqu'à atteindre la destination. Durant son parcours à travers le réseau, le message RREQ réalise la création des entrées temporaires des tables de routage pour la route inverse des nœuds à travers lesquels il passe. Si la destination, ou une route vers elle, est trouvée, une route est rendue disponible en envoyant un message réponse de route (Route Reply, RREP) au nœud source. Cette réponse de route traverse le long du chemin temporaire inversé du message RREQ. Dans son chemin de retour vers la source, RREP introduit la création des entrées pour la destination dans les tables de routage des nœuds intermédiaires. Les entrées de routage expirent après une certaine période (time-out). Précisons que le protocole AODV ne supporte que les liens symétriques dans la construction des chemins inverses.

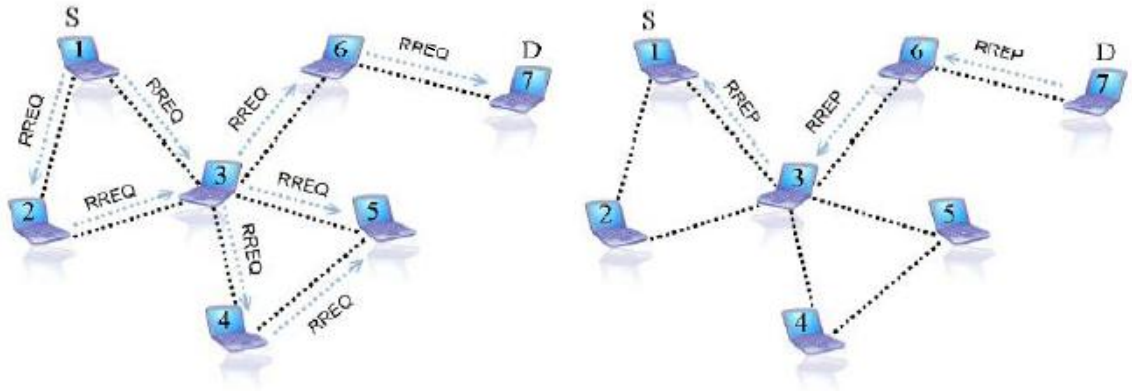

Figure II.6 : Fonctionnement de la procédure de demande de route dans AODV.

II.2.3.3 Protocoles hybrides

Ces protocoles combinent les deux idées des protocoles proactifs et réactifs. Ils utilisent un protocole proactif pour apprendre le proche voisinage (par exemple le voisinage à deux ou à trois sauts). Ainsi, ils disposent de routes immédiatement dans le voisinage. Au-delà de la

zone du voisinage, le protocole hybride fait appel à un protocole réactif pour chercher des routes.

Dans le reste de cette section, nous présentons une vue d'ensemble détaillée des principaux protocoles de routage dans les RCSFs appartenant à ces classes.

- **Zone Routing Protocol (ZRP) : [B19]**

Le protocole ZRP met en place simultanément, un routage proactif et un routage réactif, afin de combiner les avantages des deux approches. Pour ce faire, il passe par un concept de découpage du réseau en différentes zones, appelées « zones de routage ». Une zone de routage pour un nœud, est définie par son « rayon de zone ». Ce rayon correspond au nombre de sauts maximum qu'il peut y avoir entre deux nœuds.

Le routage au sein d'une zone se fait de manière proactive, via le protocole IARP (Intrazone Routing Protocol) et le routage vers les nœuds extérieurs de la zone se fait de façon réactive, grâce au protocole IERP (Interzone Routing Protocol). En plus de ces deux protocoles, ZRP utilise le protocole BRP (Bordercast Routing Protocol). Ce dernier a pour but de construire la liste des nœuds périphériques d'une zone ainsi que les routes permettant de les atteindre, en utilisant les données de la topologie fournies par le protocole IARP. Il est utilisé pour propager des requêtes de recherche de routes de l'IERP dans le réseau.

La recherche des chemins s'effectue comme suit : on vérifie tout d'abord si le nœud destinataire se trouve dans la zone du nœud source, auquel cas le chemin est déjà connu. Autrement, une demande d'établissement de route RREQ est initiée vers tous les nœuds périphériques. Ces derniers vérifient si la destination existe dans leurs zones. Dans le cas d'une réponse affirmative, la source recevra alors un paquet RREP contenant le chemin menant à la destination. Dans le cas contraire, les nœuds périphériques diffusent la requête à leurs propres nœuds périphériques qui, à leur tour, effectuent le même trait. Nous remarquons que pour un rayon de zone égal à deux, la zone de routage du nœud *S* est constituée par tous les nœuds qui sont autour du nœud *S* avec un maximum de deux sauts les séparant. Sont donc inclus dans la zone de routage, tous les voisins du nœud *S* ainsi que tous les voisins de ces voisins.

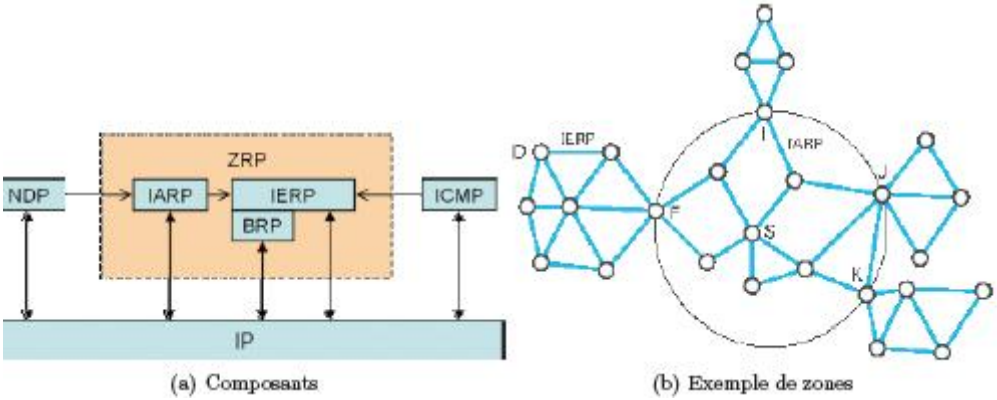

Figure II.7 : Le protocole hybride ZRP.

ZHLS (Zone-based Hierarchical Link State Protocol):[B20]

Le protocole ZHLS est basé sur la décomposition d'un réseau en zones. Contrairement à la plupart des protocoles dit hiérarchiques, il n'y a pas ici de représentant pour chaque zone. La topologie d'un réseau est ainsi partagée en deux niveaux :

– Un niveau nœud indique la façon dont les nœuds d'une zone sont connectés entre eux physiquement. Un lien virtuel peut exister entre deux zones s'il existe au moins un nœud d'une autre zone.

– Un niveau zone qui renseigne sur le schéma de connexion des différentes zones.

Ces niveaux différents entraînent donc deux différents types de liens : les liens inter-nœuds et les liens inter-zones.

Le réseau est donc décomposé. Il résulte de cette décomposition un routage inter-zone et un routage intra-zone qui est permise par l'adressage mis en place et qui consiste en un identifiant de zone, un identifiant de nœuds et l'utilisation de LSP (Link State Packet) qui renseignent sur l'état des liens. Il est alors également possible de distinguer deux classes de LSP : la classe des LSP orientés nœuds pour lesquels un nœud donné contient des informations sur son voisin et celle des LSP orientés zones qui sont, quant à elles, échangées de manière globale. Ainsi chaque nœud du réseau possède une connaissance complète concernant les nœuds de sa propre zone et seulement une connaissance partielle du reste des nœuds. Les nœuds déterminent leur position physique en utilisant le GPS. La carte de zone est établie pendant la phase de composition du réseau.

II.2.4 Classification selon l'initiateur de communication :

La communication dans un réseau de capteurs peut être lancée par les nœuds sources ou par les nœuds destinataires.

II.2.4.1 Communication lancée par la source :

Dans les protocoles de communication lancée par la source, les nœuds envoient des données à la destination quand ils les ont capturées. Ces protocoles utilisent les données rapportées avec time-driven. Ceci signifie que les données sont envoyées à certains intervalles ou quand les nœuds capturent certains évènements.

II.2.4.2 Communication lancée par la destination :

Les protocoles de communication lancée par la destination utilisent les données rapportées avec query-driven, et dans ce cas, les nœuds répondent aux requêtes envoyées par la destination ou un autre nœud différent. C'est-à-dire propager les requêtes à tous les nœuds d'une région topologique et attendre la réception des données du nœud capteur concerné dans cette région.

II.3 Conclusion :

Nous avons essayé à travers ce chapitre de mettre les points sur les différents protocoles de routage dédiés aux RCSF, et les classer en quatre classes différentes selon plusieurs critères. Cette mise au point nous a permis de déduire que le protocole LEACH « protocole hiérarchique » est le plus important des protocoles de routage pour la consommation d'énergie. Cela nous a mené à faire une étude complète de ce protocole proposé dans le chapitre qui suit.

Chapitre III: Consommation d'énergie

III.1 Introduction

Dans **[B01],** Heinzelman et al. Ont proposé un algorithme de clustering distribué appelé LEACH pour le routage dans les réseaux de capteurs homogènes.

Les réseaux de capteurs sans fil se distinguent par leur densité importante des nœuds, leur autonomie énergétique limitée, leur topologie mobile, etc. Ces contraintes ont posé plusieurs défis pour la conception et la gestion de réseaux de capteurs sans fil. L'un de ces défis est la mise en revue et le développement des nouveaux protocoles de routage adéquats à la nature particulière des réseaux de capteurs sans fil. Il arrive que plusieurs nœuds dans un réseau de capteurs sans fil combinent les données capturées et les communique à la station de base. Il apparait donc qu'il existe des différences considérables entre les réseaux de capteurs sans fil et les réseaux traditionnels. C'est pourquoi il n'est pas toujours possible d'utiliser les mêmes protocoles. Dans notre étude, nous nous sommes basés sur le routage basé clustering, il consiste à partitionner le réseau en groupes (cluster), ou dans chacun d'entre eux un seul nœud est sélectionné comme leader pour jouer le rôle spécial de point de transfert.

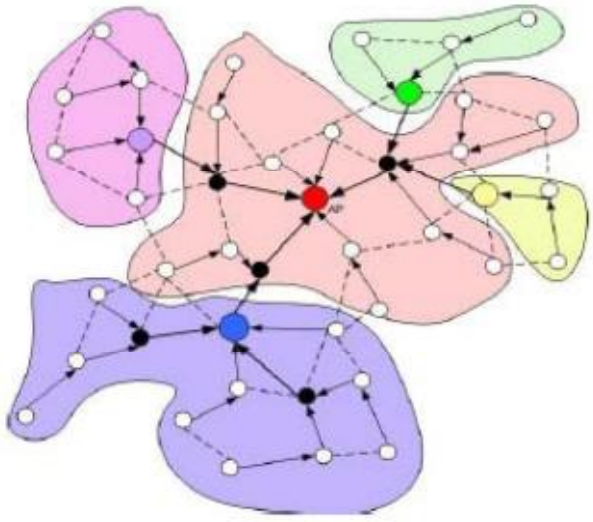

Figure III.1 : routage basé clustering

LEACH est un protocole de routage destiné aux réseaux de capteurs. Son principal avantage est de minimiser la consommation énergétique des éléments du réseau. Dans ce

chapitre nous présentons une étude sur le protocole de base LEACH, ensuite une étude sur ses variantes.

III.2 Le protocole de base LEACH (Low Energy Adaptive Clustering Hierarchy) [B01, C01]

Le protocole de base est bien le protocole LEACH, son principe est la division du réseau en un ensemble de clusters selon la force reçue du signal et d'utiliser les CHs (Cluster Heads) comme des routeurs pour passer les données à la station de base. Ce dernier prend en charge la gestion de son cluster. Il est élu périodiquement parmi les nœuds formant le cluster, en fonction de l'état de sa batterie.

Ce protocole permet ainsi la structuration du réseau de manière hiérarchique dans le but d'économiser l'énergie des capteurs.

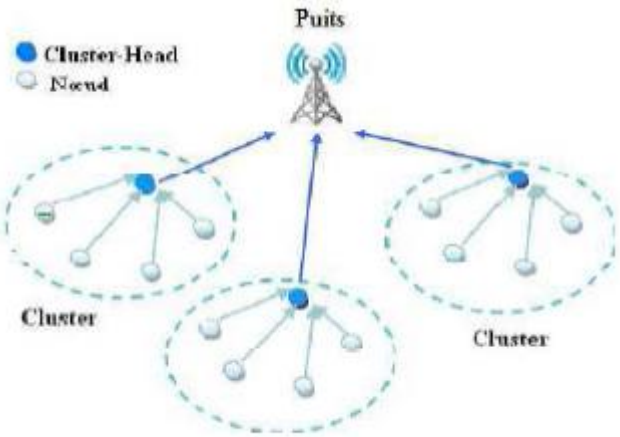

Figure III.2 : Routage hiérarchique basé sur le clustering

Les dispositifs principaux de LEACH sont :

- Coordination et contrôle localisés entre les nœuds : pour l'initialisation et le traitement de grappe.

- Rotation randomisée de cluster : effectuée par ‘’la station de base’’ ou ‘’les têtes de cluster’’.
- Compression locale (agrégation) : Les nœuds CH compressent les données arrivant des nœuds appartenant à leur grappe respective, et envoi un paquet d'agrégation à la station de base afin de réduire la quantité d'information qui doit être transmise à la station de base.

Dans LEACH, le traitement est séparé dans des cycles de longueur constante, ou chaque cycle commence par une phase d'initialisation suivie d'une phase de transmission. La durée d'un cycle est déterminée.

III.2.1 Phase d'initialisation

Cette phase d'initialisation repose sur trois sous phases comme l'indique la figure III.3, la phase d'annonce, la phase d'organisation des groupes et la phase d'ordonnancement, qui permettent la formation des clusters et l'élection des CHs qui seront détaillées ci-dessous.

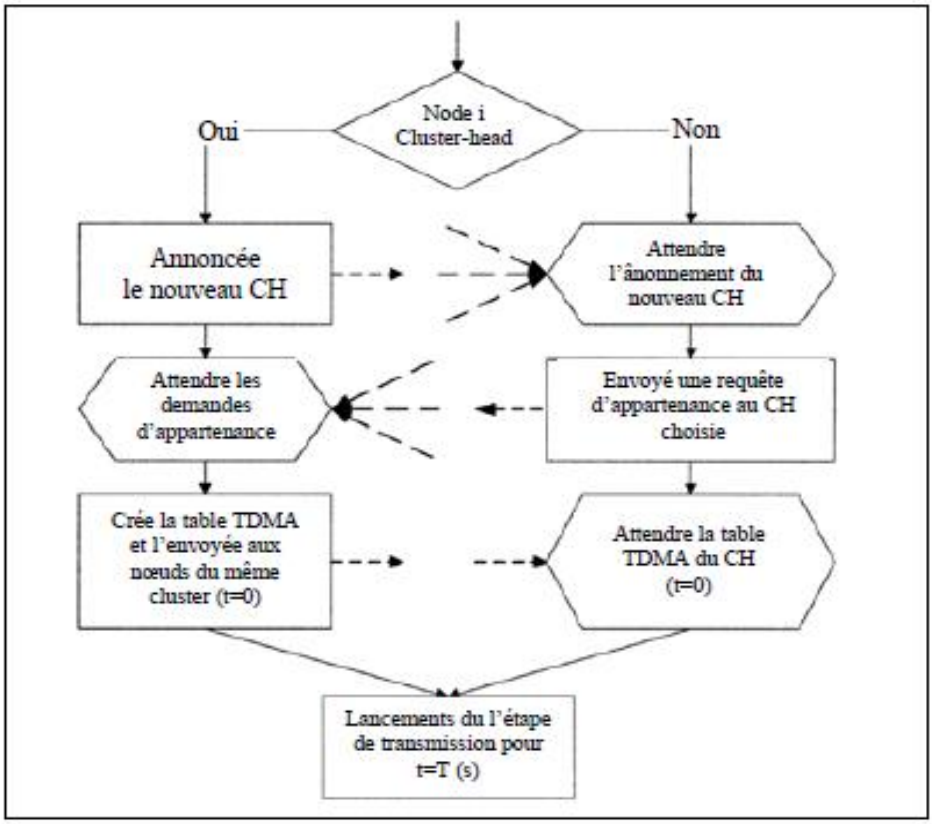

Figure III.3 :Opération de l'étape d'initialisation de LEACH

III.2.1.1 La -phase d'annonce

Avant de lancer cette phase, on désire avoir un certain nombre de CH. Ce nombre que l'on note K, est fixe et il est inchangé durant tous les rounds. On estime que le pourcentage optimal de nombre de CH désirés devrait être de 5% à 15% du nombre totale de nœuds. Si ce pourcentage n'est pas respecté, cela mènera à une grande dissipation d'énergie dans le réseau. En effet, si le nombre de CH est très élevé, on aura un nombre important de nœuds(CH) qui se consacrent aux tâches très couteuses en ressources énergétique. Ainsi, on aura une dissipation d'énergie considérable dans le réseau. De plus, si le nombre de CH est très petit, ces derniers vont gérer des groupes de grandes tailles. Ainsi, ces CH s'épuiseront rapidement à cause du travail important qui leur est demandé.

Cette phase commence par l'annonce de nouveau round par le nœud puits, et par la prise de décision locale d'un nœud pour devenir CH avec une certaine probabilité $\mathbf{P_i\ (t)}$ au début du round r+1 qui commence à l'instant t. Chaque nœud i génère un nombre aléatoire entre 0 et 1. Si ce nombre est inférieur à $\mathbf{P_i\ (t)}$, le membre deviendra CH durant le round r+1. $P_i(t)$ est calculé en fonction de K et de round r :

$$\text{Nombre (CH)} = \sum_{i=1}^{N} \gamma_i(t) = K$$

Où :

N est le nombre total de nœuds dans le réseau.

Si on a N nœud et K CH, alors il faudra N/K rounds durant lesquels un nœud doit être élu seulement une seule fois autant que CH avant que le round soit réinitialisé à 0. Donc la probabilité de devenir CH pour chaque nœud i est :

$$P_i(t) = \frac{le\ nombre\ de\ CH\ désirés}{le\ nombre\ de\ noeuds\ qui\ ont\ pas\ encore\ été\ élus\ CH\ durant\ les\ r\ rounds\ précédents}$$

$$P_i(t) = \begin{cases} \frac{K}{N - K(r \bmod N/K)} & \text{si } C_i(t) = 1 \\ 1 & \text{si } C_i(t) = 0 \end{cases} \quad \ldots\ldots\ldots (1)$$

Où $C_i(t)$ égale à 0 si le nœud i a déjà été CH durant l'un des (r mod N/K) rounds précédents, et il est égal à 1 dans le cas contraire. Donc, seuls les nœuds qui n'ont pas encore été CH. ont vraisemblablement une énergie résiduelle suffisante que les autres et ils pourront être choisis.la somme des $C_i(t)$ (i=0…N) représente le nombre totale des nœuds éligibles d'être CH à l'instant t.

Il est égal à :

$$\sum_{i=1}^{N} \gamma_i(t) = N - k^*(r \bmod N/K) \qquad \ldots..(2)$$

Utilisant l'équation (1) et (2), le nombre de CH par round est:

$$\text{Nombre (CH)} = \sum_{i=1}^{N} \gamma_{i(t)} \cdot c_{i(t)} = (K^*(r \bmod N/K)) * \left(\frac{K}{N - K(r \bmod N/K)}\right) = K$$

La probabilité $P_i(t)$ est basée sur la supposition que tous les nœuds sont initialement homogènes et commencent avec la même quantité résiduelle d'énergie et meurent approximativement en même temps.

III.2.1.2 La phase d'organisation de groupes

Après qu'un nœud soit élu CH, il doit informer les autres nœuds non-CH de son nouveau rang dans le round courant. Pour cela, un message d'avertissement contenant l'identificateur du CH est diffusé à tous les nœuds non-CH en utilisant le protocole MAC CSMA pour éviter les collisions entre les CH. La diffusion permet de s'assurer que tous les nœuds non-CH ont reçu le message. La décision de jointure est basée donc sur l'amplitude du signale reçu, le CH ayant le signal le plus fort (i.e. le plus proche) sera choisi. En cas d'égalité des signaux, les nœuds non-CH choisissent aléatoirement leurs CH.

Chaque membre informe son CH de sa décision. Une fois que le CH ait reçu la demande, il lui envoie un message d'acquittement.

III.2.1.3 La phase d'ordonnancement

Après la formation des groupes, chaque CH agit comme un centre de commande locale pour coordonner les transmissions des données au sein de son groupe. Il crée un ordonnanceur (Schedule) TDMA et assigne à chaque nœud membre un slot de temps durant lequel il peut transmettre ses données. L'ensemble des slots assignés aux nœuds d'un groupe est appelé frame.

La durée de chaque frame diffère selon le nombre de membres du groupe. Par ailleurs, afin de minimiser les interférences entre les transmissions dans des groupes adjacents, chaque CH choisit aléatoirement un code dans une liste de codes de propagation CDMA. Il le transmit par la suite à ses membres afin de l'utiliser pour leurs transmissions.

III.2.2 La phase de transmission

Cette phase est plus longue que la phase précédente, et permet la collecte des données captées. En utilisant l'ordonnanceur TDMA, les membres émettent leurs données captées pendant leurs propres slots. Cela permet d'éteindre leurs interface de communication en dehors de leurs slots afin d'économiser leurs énergie. Ces données sont ensuite agrégées par les CH qui les fusionnent et les compressent, et envoient le résultat final au nœud puits. Après un certain temps prédéterminé, le réseau va passer à un nouveau round. Ce processus est répété jusqu'à ce que tous les nœuds de réseau seront élu CH. Une seule fois, tout au long des rounds précédents. Dans ce cas, le round est réinitialisé à 0.

III.3 La durée de vie du réseau

Au niveau du protocole LEACH, la durée de vie du réseau est faible, parce que dans LEACH, les nœuds s'épuisent plus rapidement vue la distance entre les CHs et leurs membres d'un coté et la distance entre les CHs et la station de base comme dans la figure III.4 En effet, les phases d'initialisation c'est-à-dire les phases de formation de clusters qui induisent un nombre important de messages de contrôle vont se faire à chaque nouveau round impliquant une consommation d'énergie supplémentaire.

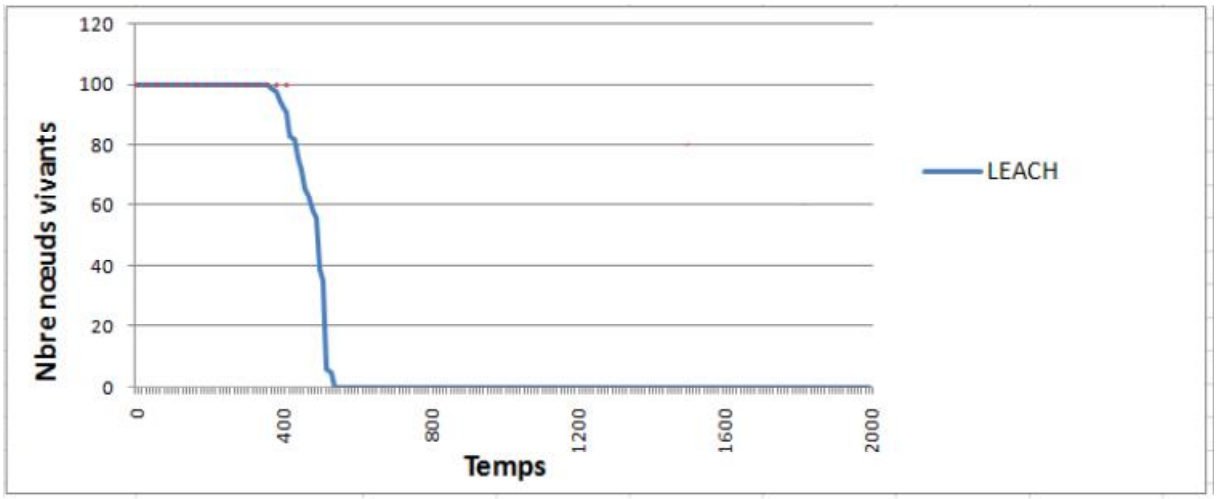

Figure III.4: La durée de vie du réseau au niveau de LEACH

III.4 Avantages et limites

Le protocole LEACH engendre beaucoup d'avantages en ce qu'il offre comme bonne manipulation de ressources de réseau en respectant plusieurs contraintes telles que la consommation d'énergie.

Bien que LEACH puisse augmenter la durée de vie du réseau, il présente certaines limitations. Il choisit aléatoirement les clusters heads pour jouer le rôle spécial de point de transfert à la station de base et il ne pose aucune contrainte sur leurs états et leurs fonctionnement ainsi que sur leurs niveaux d'énergie. Donc, les Clusters Heads peuvent tomber en panne et par conséquent, il pourrait exister des nœuds morts (destruction physique accidentelle de CH par exemple) ce qui ne garantit pas une livraison fiable des données dans les RCSF.

III.5 Les variantes de LEACH

Les auteurs ont comparé les réseaux homogènes et hétérogènes en terme de dissipation d'énergie dans tout le réseau et ils ont analysé les performances des réseaux à un saut et ceux à sauts multiples. Ils ont choisi pour cela LEACH comme représentant des réseaux homogènes et ils l'ont comparé avec un réseau hétérogène à un saut. Les auteurs ont constaté que l'utilisation des communications à un saut entre les membres d'un cluster et leur Cluster Head correspondant pourrait ne pas être le bon choix quand l'index k de perte de propagation ($k > 2$) pour les communications intra-clusters est plus grand.

D'autre part, LEACH pourrait produire des clusters possédant une taille importante dans les réseaux denses et des clusters dont la taille est limitée dans les réseaux de petites tailles. Dans ces deux cas, les Cluster Heads pourraient rapidement épuiser leur puissance de batterie. Dans les réseaux denses, les Cluster Heads coordonnent entre plusieurs membres des clusters alors que dans les réseaux de petites tailles, les Cluster Heads sont placés loin de la station de base ce qui nécessite des transmissions de forte puissance. Dans le même article, les auteurs ont proposé une version améliorée de LEACH appelée M-LEACH (Multi-hop LEACH), dans laquelle les membres d'un cluster peuvent être à plus d'un saut de leur Cluster Head correspondant et communiquent avec lui en mode multi-saut.

Ainsi, ils ont illustré les cas dans lesquels M-LEACH surpasse LEACH. Cependant, cette version proposée exige que chaque capteur doit être capable d'agréger les données, ce qui augmente l'overhead pour tous les capteurs. Pour améliorer cette stratégie, les auteurs se sont focalisés sur les réseaux de capteurs hétérogènes, dans lesquels deux types de capteurs sont déployés : capteurs de grandes capacités (Super Sensor) et capteurs simples. Les capteurs de grandes capacités ont des capacités de traitement et de communication si élevées et agissent comme Cluster Heads, alors que les autres sont des capteurs simples avec une puissance limitée, affiliés au Cluster Head le plus proche dans leur voisinage et communiquent avec lui directement ou en mode multi-saut.

En outre, une autre variante de LEACH appelée LEACH-C a été conçue pour améliorer les performances de LEACH. Cette variante utilise une architecture centralisée pour choisir les Cluster Heads tout en impliquant la station de base et l'information de localisation des capteurs.

Cependant, elle augmente considérablement l'overhead du réseau puisque tous les capteurs devront envoyer leurs informations de localisation à la station de base en même temps pendant chaque phase d'élection de Cluster Heads. Plusieurs travaux présentés dans la littérature ont prouvé qu'une telle architecture centralisée ne supporte pas le passage à l'échelle et est plus particulièrement appropriée à des réseaux de petite taille.

D'une manière similaire à LEACH-C, BCDCP (Base-Station Controlled Dynamic Clustering Protocol) implique le niveau d'énergie des capteurs envoyé à la station de base pour construire des clusters homogènes durant la phase d'installation (1ère phase). La station de base choisit aléatoirement les Cluster Heads tout en garantissant une distribution uniforme de leurs emplacements dans la zone d'intérêt dans laquelle ils sont déployés, et exécute un algorithme itératif de fusion pour trouver le nombre optimal de clusters. Puis, elle établit les routes inter-clusters (CH-to-CH) pour l'acheminement des données d'un Cluster Head à un autre, et crée un schedule pour chaque cluster qui le diffuse dans le réseau. Durant la deuxième phase, les Cluster Heads transmettent les données collectées à la station de base par des chemins CH-to CH. Néanmoins, BCDCP présente les mêmes limitations que LEACH-C puisqu'il utilise une architecture centralisée pour élire les Cluster Heads.

Les techniques de clustering que nous avons présentés dans cette section, quoiqu'elles préconisent une solution garantissant l'équilibre des charges dans l'élection des Cluster Heads, ont un impact négatif sur les Cluster Heads, puisque leur choix se fait aléatoirement. Or, ces derniers consomment leur énergie plus rapidement qu'un nœud ordinaire puisqu'ils supportent des fonctions additionnelles comme l'agrégation des données et le routage. Le choix d'un Cluster Head qui a un niveau d'énergie plus faible, pourrait vite devenir un goulet d'étranglement de son cluster.

D'autre part, dans la phase de reconstruction des clusters, un overhead de communications et de calculs est généré puisque tous les capteurs envoient simultanément leurs niveaux d'énergie à la station de base et la connaissance appropriée de la topologie du réseau est exigée.

III.6 Conclusion

Dans ce chapitre, nous avons présenté et étudié l'algorithme de routage LEACH conçu aux RCSF, ensuite ses variantes.

Notre constat nous a permis d'illustrer leurs limites. D'où, nous avons pensé à améliorer ce dernier pour assurer la fiabilité de livraison de données tout en consommation d'énergie. On va entamer dans le chapitre qui suit l'étape de test qui constitue un point important dans le processus d'analyse des performances de ces algorithmes (LEACH et LEACH++).

Chapitre IV:Implémentation et tests

IV.1 Introduction:

Dans le domaine des réseaux de capteurs sans fil, la simulation est une étape incontournable lorsqu'on veut tester et évaluer des modèles d'application ou des protocoles de communication. L'expérimentation réelle s'avère quelques fois très couteuse. De plus, la simulation offre un gain considérable en temps, une flexibilité en permettant la variation des paramètres et une meilleure visualisation des résultats sous forme de graphes faciles à analyser et interpréter. Cependant, la simulation ne peut pas remplacer l'implémentation réelle. La validité du modèle simulé ne garantit pas le bon déroulement de son implémentation réelle. Des erreurs de programmation peuvent toujours survenir au moment de l'implémentation du simulateur. L'étape de la simulation doit être suivie par une implémentation physique pour vérifier le comportement réel des modèles. Dans ce chapitre, l'analyse des performances des algorithmes (LEACH, LEACH++) est évaluée à l'aide du simulateur TOSSIM.

IV.2 Objectif :

L'objectif principal de notre travail est d'améliorer le protocole de routage hiérarchique LEACH.

L'un des problèmes de LEACH est que c'est uniquement le CH qui envoie les données à la station de base, donc, il n'y aura pas de données envoyées de CH à la station de base si ce dernier cesse de fonctionner.

LEACH++ est un algorithme amélioré de LEACH, qui assure la livraison de données à la station de base tout en prolongeant la durée de vie du système durant les différents rounds de clustering du réseau et obtient des améliorations pour les prochains rounds s'il y a un évènement imprévu dans le réseau (comme l'échec d'un nœud ou sa mort), LEACH++ est basé sur LEACH et similaire à lui.

Dans LEACH, l'occurrence d'une panne permet la perte des données car c'est le Cluster Head qui envoie les données qu'il a reçu de la part de ces membres à la station de base. Mais dans LEACH++, un autre Cluster Head adjoint sera élu dès que le Cluster Head principal tombe en panne ce qui permet d'assurer la livraison des données à la station de base.

LEACH++ tente de diminuer la consommation d'énergie et garantir un routage efficace.

IV.3 Environnement de développement :

IV.3.1 TinyOS et NesC :

TinyOS est un système d'exploitation intégré, modulaire, destiné aux réseaux de capteurs miniatures.

NesC est un langage de programmation orienté composants. Il est conçu pour la réalisation des systèmes embarqués distribués, en particulier, les RCSF (Ayant été cités précédemment dans le premier chapitre).

IV.3.2 TOSSIM :

Avant sa mise en place, le déploiement d'un RCSF nécessite une phase de simulation afin d'assurer un bon fonctionnement de tous les protocoles de communication qu'il utilise.

En effet, pour de grands réseaux, le nombre de capteurs peut atteindre plusieurs milliers et entraine donc un coût financier relativement important. Ainsi, il faut réduire au maximum les erreurs de conception. Malgré cela, il reste des facteurs réels qui ne peuvent être pris en compte par la simulation, tels que les contraintes physiques (perturbations électromagnétiques, inondations, …, etc.) ou les aléas (détériorations dues à un animal… etc.). Pour arriver à simuler le comportement des capteurs au sein d'un RCSF, un outil très puissant a été développé et proposé pour TinyOS sous le nom de TOSSIM. Le principal but de TOSSIM est de créer une simulation très proche de ce qui se passe dans les RCSF dans le monde réel. Une économie d'effort et une préservation du matériel sont possibles grâce à cet outil**.**

Pour une compréhension moins complexe de l'activité du réseau, TOSSIM peut être utilisé avec une interface graphique TinyViz. Cette dernière est équipée de plusieurs API plugins qui permettent d'ajouter plusieurs fonctions à notre simulateur comme par exemple suivre la dépense d'énergie en utilisant un autre simulateur qui s'appelle Power TOSSIM.

IV.3.2.1 TinyViz :

TinyViz est une interface graphique Java. Elle permet de donner un aperçu des capteurs à tout instant ainsi que les divers messages qu'ils émettent. Elle détermine un délai entre chaque itération des capteurs afin de permettre une analyse pas à pas du bon déroulement des actions en activant différents modes comme Radio, CPU, etc.

Nous allons détailler un peu ce que fait chaque bouton présent dans l'interface (la partie du haut en noir) :

- **ON/OFF** : il met en marche ou éteint un capteur.
- **Delay** : il permet de sélectionner la durée au bout de laquelle se déclenche le timer.
- **Play** : il permet de lancer la simulation ou de la mettre en pause.
- **Grilles** : il permet d'avoir une grille pour situer les capteurs en espace
- **Clear** : il efface tous les messages qui transitent entre les capteurs.
- **Arrêt** : il met fin à la simulation.

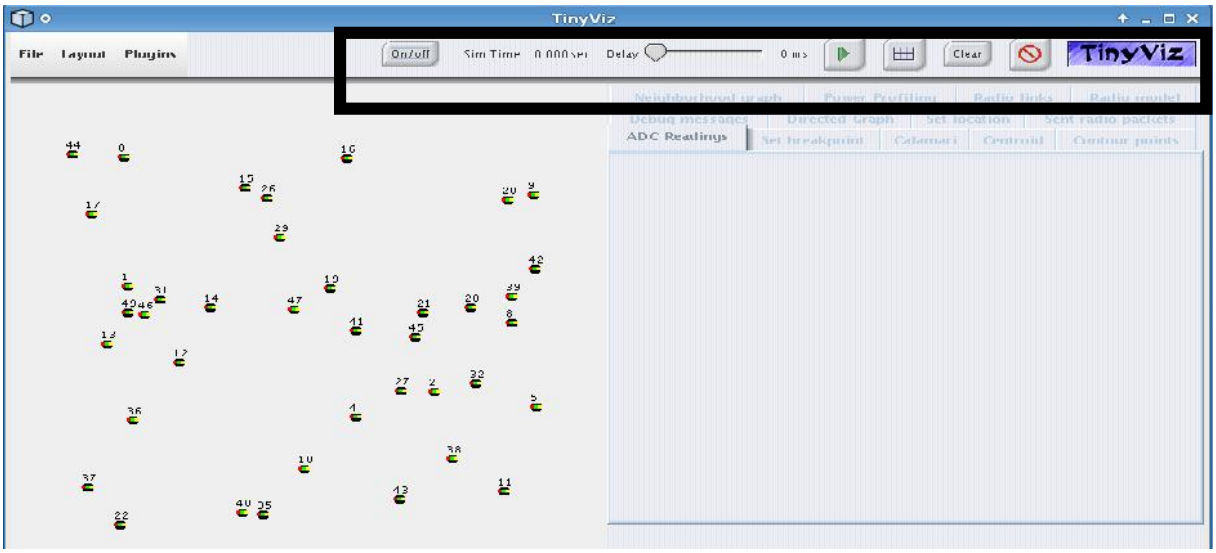

Figure IV.1 : La fenêtre du logiciel TinyViz.

Plusieurs plugins (en vert qui sont présenté dans la figure ci-dessous) sont disponibles pour visualiser la simulation sous différentes formes. Les plugins dont on s'est beaucoup servi sont « Debug messages » pour afficher tous les messages de type « dbg » afin de vérifier la nature des messages et « Radio links » qui nous permet de voir, avec des flèches ou des cercles, si un capteur est en train d'émettre ou non, les échanges entre les capteurs peuvent être de deux manières :

- Unicast : échange effectué entre deux capteurs seulement.
- Broadcast : message émis par un capteur à l'ensemble du réseau.

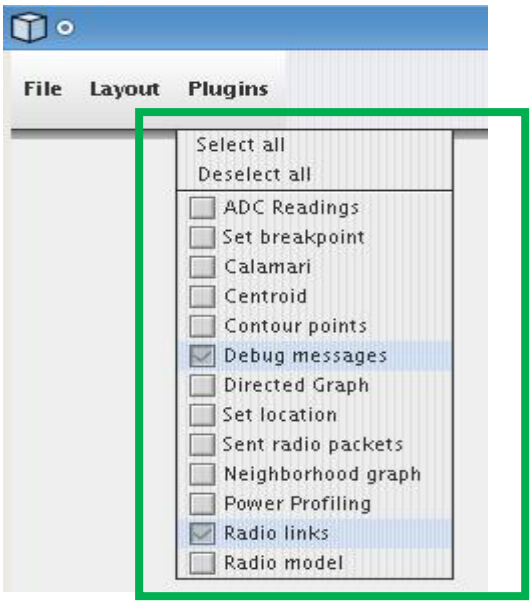

Figure IV.2 : Les différents plugins

IV.3.3 Power TOSSIM :

Power TOSSIM 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 (nœud développé à l'université de Berkeley). Les auteurs connaissent les consommations des différents composants de ce nœud suivant leurs états. Il faut donc connaitre l'état de chaque composant d'un nœud pendant la simulation. Grâce au modèle de simulation basé sur TinyOS, on connait immédiatement l'état des composants autres que le microcontrôleur puisque les changements d'états correspondent à des événements dans TinyOS et donc dans TOSSIM. Plusieurs composants du nœud sont parfois abstraits dans TOSSIM par un seul composant. L'estimation de la consommation du microcontrôleur est plus délicate : il faut instrumenter le code pour être capable de compter le nombre d'exécution de chaque bloc d'instruction, et il faut faire correspondre chaque bloc d'instruction avec son code en assembleur. Lors de la simulation, on note le nombre de passage, d'exécution de chaque bloc d'instructions. Sachant combien d'instruction élémentaire contient chaque bloc de base, on en déduit le nombre d'instructions effectuées par le microcontrôleur et donc sa consommation. Cette approche est intéressante mais elle ne permet pas de varier la précision du modèle de consommation. Enfin, les simulateurs TOSSIM et Power TOSSIM ne conviennent que pour des applications écrites en TinyOS.

IV.4 Le choix de notre simulateur :

Notre choix de TOSSIM se justifie par le fait que :

- TOSSIM simule fidèlement le comportement d'un réseau en s'appuyant sur la plateforme TinyOS. En effet le code de simulation peut être exécuté directement dans un capteur utilisant le système TinyOS ;
- il simule un réseau d'une manière simple et efficace ;
- il supporte un grand nombre de nœuds qui peut aller jusqu'à un millier ;
- Grâce à l'interface graphique TinyViz, on peut visualiser les échanges radios pour avoir une vue globale du réseau.

IV.5 Implémentation et déroulement:

Dans cette section, nous décrivons les structures de données, de fichiers ainsi que les principaux commandes et événements nécessaires pour l'implémentation du protocole LEACH++.

IV.5.1 Les fichiers de l'application:

Notre application est formée des composants suivants :

- ❖ un module, appelé «LeachM.nc ».
- ❖ une configuration, appelée «Leach.nc ».
- ❖ un fichier d'entête, appelé «Head.h ».
- ❖ un fichier Makefile, appelé « Makefile »

IV.5.2 Structure de données :

Le paquet dans TinyOS est envoyé dans une structure appelée TOS_Msg, qui est contenue dans un champ « int8_t data [TOSH_DATA_LENGTH] ». Les structures de données du paquet diffèrent selon le rang du nœud (station de base, CH ou membre).

A) Le nœud puits

```
typedef struct PUITS
{
    uint16_t  ID;              //l'identificateur de chaque noeud qui correspond à TOS_LOCAL_ADRESS
    uint8_t   round;           //le round courant
    uint8_t   Depth;           //la profondeur du neoud dans le réseau
    float     probability;     //la probabilité que chaque noeud devienne CH
}PUITS;
```

B) Le nœud Cluster Head

```
typedef struct CLUSTER_HEAD
{
   uint16_t  ID_MEMBRE;          //l'identificateur de chaque noeud qui correspond à TOS_LOCAL_ADRESS
   uint16_t  ID_CH;              //l'identificateur du CH du cluster contenant le noeud membre
   uint8_t   agregation;         //la donnée agrégée à envoyer au noeud PUITS
   uint16_t  FREQ;               //La fréquence avec laquelle les membres d'un Cluster envoient
   uint16_t  SLOT_ATTRIBUER;     //le slot attribué à chaque membre
   uint8_t   NBR_MBR;            //le nombre de membres utilisé pour la connectivité
}CLUSTER_HEAD;
```

C) Le nœud membre

```
typedef struct MEMBRE
{
   uint16_t ID_MEMBRE;          //l'identificateur de chaque noeud qui correspond à TOS_LOCAL_ADRESS
   uint16_t ID_CH;              //l'identificateur du CH du cluster contenant le noeud membre
   uint8_t  temp;               //variable qui contient la température captée
   uint8_t  req;                //si req=1 alors le neoud membre préviend le CH qu'il fait partie de son groupe
                                //si il est égal à 2 ça veut dire qu'il a envoyé la valeur captée
}MEMBRE;
```

IV.5.3 Structure de fichiers

A)- La configuration Leach.nc

```
includes MH;

configuration Leach
{

}
implementation
{
   components Main, LeachM, GenericComm as Comm, TimerC, RandomLFSR, LedsC;
   // components c'est l'ensemble des composants référencés par  la configuration
   // Main est exécuté en premier lieu, sa présence est obligatoire


       // le coté gauche relie une interface à son implémentation fournie par le coté droit
       // le composant qui utilise l'interface -> le compsasnt qui fournie une implémentation de l'interface
   Main.StdControl -> LeachM.StdControl;
   // relier StdControl utilisée par Main à StdControl implémenté par LeachM

   Main.StdControl -> Comm.Control;
   //relier StdControl utilisée par Main à Control implémenté par COm

   Main.StdControl -> TimerC.StdControl;
   //relier StdControl utilisée par Main à StdControl implémenté par TimerC

   LeachM.LEACH_ReceiveMsg -> Comm.ReceiveMsg[AM_MHMESSAGE];
   // relier LEACH_ReceiveMsg utilisée par LeachM à ReceiveMsg[AM_MHMESSAGE]implémenté par Comm

   LeachM.LEACH_SendMsg -> Comm.SendMsg[AM_MHMESSAGE];
   // relier LEACH_SendMsg utilisée par LeachM à SendMsg[AM_MHMESSAGE]implémenté par Comm


 LeachM.ANNONCE_ReceiveMsg -> Comm.ReceiveMsg[AM_MEMBRE];
 // relier ANNONCE_ReceiveMsg utilisée par LeachM à ReceiveMsg[AM_MEMBRE] implémenté par Comm

 LeachM.ANNONCE_SendMsg -> Comm.SendMsg[AM_MEMBRE];
 // relier  ANNONCE_SendMsg utilisée par LeachM à SendMsg[AM_MEMBRE] implémenté par Comm

 LeachM.ORGANISATION_ReceiveMsg -> Comm.ReceiveMsg[AM_ORGANISATION];
 // relier  ORGANISATION_ReceiveMsg utilisée par LeachM à ReceiveMsg[AM_ORGANISATION] implémenté par Comm

 LeachM.ORGANISATION_SendMsg -> Comm.SendMsg[AM_ORGANISATION];
 // relier  ORGANISATION_SendMsg utilisée par LeachM à SendMsg[AM_ORGANISATION] implémenté par Comm

 LeachM.SLOT_ReceiveMsg -> Comm.ReceiveMsg[AM_SLOT];
 // relier  SLOT_ReceiveMsg utilisée par LeachM à ReceiveMsg[AM_SLOT] implémenté par Comm

 LeachM.SLOT_SendMsg -> Comm.SendMsg[AM_SLOT];
 // relier  SLOT_SendMsg utilisée par LeachM à SendMsg[AM_SLOT] implémenté par Comm

 LeachM.AGGREGATION_ReceiveMsg -> Comm.ReceiveMsg[AM_AGGREGATION];
 // relier  AGGREGATION_ReceiveMsg utilisée par LeachM à ReceiveMsg[AM_AGGREGATION] implémenté par Comm

 LeachM.AGGREGATION_SendMsg -> Comm.SendMsg[AM_AGGREGATION];
 // relier  AGGREGATION_SendMsg utilisée par LeachM à SendMsg[AM_AGGREGATION] implémenté par Comm

 LeachM.Random -> RandomLFSR;
 // relier  Random utilisée par LeachM à Random implémenté par RandomLFSR

 LeachM.Leds -> LedsC.Leds;
 // relier Leds  utilisée par LeachM à Leds implémenté par LedsC
```

```
    LeachM.ReqRelayTimer -> TimerC.Timer[(uint8_t)unique("Timer")];
    // relier  ReqRelayTimer utilisée par LeachM à Timer[(uint8_t)unique("Timer")] implémenté par TimerC

    LeachM.RoundTimer -> TimerC.Timer[(uint8_t)unique("Timer")];
    // relier  RoundTimer utilisée par LeachM à Timer[(uint8_t)unique("Timer")] implémenté par TimerC

    LeachM.AnnonceTimer -> TimerC.Timer[(uint8_t)unique("Timer")];
    // relier  AnnonceTimer utilisée par LeachM à  Timer[(uint8_t)unique("Timer")]implémenté par TimerC

    LeachM.OrganisationTimer -> TimerC.Timer[(uint8_t)unique("Timer")];
    // relier  OrganisationTimer utilisée par LeachM à Timer[(uint8_t)unique("Timer")] implémenté par TimerC

    LeachM.OrdonnecementTimer -> TimerC.Timer[(uint8_t)unique("Timer")];
    // relier  OrdonnecementTimer utilisée par LeachM à Timer[(uint8_t)unique("Timer")] implémenté par TimerC

    LeachM.AggregerTimer -> TimerC.Timer[(uint8_t)unique("Timer")];
    // relier  AggregerTimer utilisée par LeachM à Timer[(uint8_t)unique("Timer")] implémenté par TimerC

    LeachM.NBRECPNNECTimer -> TimerC.Timer[(uint8_t)unique("Timer")];
    // relier  NBRECPNNECTimer utilisée par LeachM à Timer[(uint8_t)unique("Timer")] implémenté par TimerC

    LeachM.Timer -> TimerC.Timer[(uint8_t)unique("Timer")];
    // relier  Timer utilisée par LeachM à Timer[(uint8_t)unique("Timer")] implémenté par TimerC
}
```

B)- le module LeachM.nc

```
includes AM;
includes MH;

module LeachM      // LeachM est le nom du module
{
   provides
   {   // les inetrfaces fournies
   }
   uses
   {   // les interfaces utilisées
   }
}

implementation     // code à implémenter
{
   // Déclaration des variables locales à chaque noeuds

/*****************************************************************************/
/*****************************************************************************/
   static void init()
   {
       // Procédure d'initialisation du round et du nombre de CH désiré pour chauqe round
       // par la station de base
   }
/*****************************************************************************/
/*****************************************************************************/
```

```
static void Envoi()
{

       proba=(float)K/(N-K*(r%(N/K)));      //calculer la probabilité pour qu'un noeud devienne CH
                                            // par la station de base

       call LEACH_SendMsg.send(TOS_BCAST_ADDR,sizeof(struct PUITS),&buffer);
}
/*****************************************************************************/
/*****************************************************************************/
static void Envoi_MEMBRE_CH()
{
       // le noeud annonce au CH qu'il est membre de son cluster

       call ORGANISATION_SendMsg.send(TOS_BCAST_ADDR,sizeof(struct MEMBRE),&buffer);
       //envoi des noeuds aux CH auxquels ils comptent appatenir

}
/*****************************************************************************/
/*****************************************************************************/
command result_t StdControl.init()
{   // initilaisation
}
/*****************************************************************************/
/*****************************************************************************/
```

```
command result_t StdControl.start()
{       // démarrage
        Envoi();
        call RoundTimer.start(TIMER_REPEAT, ROUND_LENGTH);
        //Timer apres lequel le noeud PUITS annonce le nouveau round
}
/*****************************************************************************/
/*****************************************************************************/
command result_t StdControl.stop()
{// arret
}
/*****************************************************************************/
/*****************************************************************************/
```

```
event TOS_MsgPtr LEACH_ReceiveMsg.receive(TOS_MsgPtr pmsg)
{
          //Réception du noeud actuel du déclenchement d'un nouveau round du noeud puits
          // l'energie du noeud actuel se décrémente aprés réception du round
          // initialiser le nombre de membres pour chaque CH à zéro
          // initialiser le nombre de membres réels qui font partie d'un CH sans compter
          // ceux qui n'arrivent pas à joindre le CH à zéro

          // initialiser la température moyenne aggrégée par le CH à zéro

          call ReqRelayTimer.start(TIMER_ONE_SHOT,(call Random.rand())%800+200);
          //Timer apres lequel les noeuds aprés réception du nouveau round du PUITS
          //le renvoie à leurs voisins proches, ce dernier fera de meme

          // l'energie du noeud se décrémente aprés renvoi du round

          if (isClusterHead)            // si le noeud actuel est un CH
          {
              call Timer.start(TIMER_REPEAT, 1000);
              //Timer utilisé pour les LED lorsque les noeuds sont élu CH

              call AnnonceTimer.start(TIMER_ONE_SHOT,ANNONCE_LENGTH+((call Random.rand())%800+200));
              //Timer apres lequel les noeuds annoncent qu'ils sont CH
          }
}
```

```
/*****************************************************************************/
/*****************************************************************************/
event TOS_MsgPtr ANNONCE_ReceiveMsg.receive(TOS_MsgPtr pmsg)
{       //utilisé par les CH pour annoncer  aux noeuds qu'ils sont les chefs

        // Réception de l'annonce du CH de son statut par le noeud actuel
        // décrémenter l'energie du membre aprés réception de l'annonce
        // le CH dans lequel le noeud membre décide d'appartenir est choisi

         call OrganisationTimer.start(TIMER_ONE_SHOT,ANNONCE_LENGTH+((call Random.rand())%800+200));
         //Timer apres lequel un noeud membre prévienne
         // son Chef qu'il fera partie de son Cluster

}

/*****************************************************************************/
/*****************************************************************************/
```

```
event TOS_MsgPtr ORGANISATION_ReceiveMsg.receive(TOS_MsgPtr pmsg)
{    //réception des CH les msg de ses membres

          //si l'energie du CH > au seuil
          {
             // si le neoud membre préviend le CH qu'il fera partie de son groupe
             {
                  //le CH reçoit la demande d'admission du noeud
                  //décrémenter l'energie du CH aprés réception du msg
                  // décrémenter l'energie du membre aprés envoi de la demande d'admission
                  // incrémenter le nombre de membres pour chaque CH
                  //attribuer un slot à chaque membre

                   // envoyer les données avec un code CDMA

                   call SLOT_SendMsg.send(ptrCH->ID_MEMBRE,sizeof(struct CLUSTER_HEAD),&buffer);
                   //ENVOI CH DES SLOTS
             }
             else
             {
                 // si le noeud a envoyé la valeur captée
                 {
                    // le CH a reçu la température
                    //décrémenter l'energie du CH aprés réception de la température
                    //décrémenter l'energie du noeud membre aprés envoi de la température

                    //ajouter cette température à la température moyenne aggrégée par le CH
                    //incrémenter le nombre de membres réels qui font partie d'un CH
```

```
                    call AggregerTimer.start(TIMER_ONE_SHOT,((Table_Entree_Membre)*(SLOT_LENGTH)));
                   //Timer  apres lequel les CH envoient la donnée au noeud PUITS aprés agrégation
                }
            }
        }

        else //(energy <= seuilenergy)         // l'energie du CH est épuisée
        {
               // afficher batterie faible
               // élire un CH Adjoint
               // décrémenter le nombre de membres de chaque CH car l'ancien CH n'est plus actif

               call ORGANISATION_SendMsg.send(TOS_BCAST_ADDR,sizeof(struct MEMBRE),&buffer);
               // envoi des noeuds aux CH auxquels ils comptent appatenir
        }
}
/*************************************************************************/
/*************************************************************************/
event TOS_MsgPtr SLOT_ReceiveMsg.receive(TOS_MsgPtr pmsg)     //réception des membres  des slots
{
        // le noeud actuel a reçu le slot du CH choisi
        // décrémenter l'energie du membre aprés réception du slot
        // décrémenter l'energie du CH aprés envoi du slot

        call OrdonnecementTimer.start(TIMER_ONE_SHOT,((ptrCH->SLOT_ATTRIBUER)*(SLOT_LENGTH))+ptrCH->FREQ);
        //Timer apres lequel les noeuds membres envoient leurs données pendant leur SLOT attribué par le CH
}
/*************************************************************************/
/*************************************************************************/
event TOS_MsgPtr AGGREGATION_ReceiveMsg.receive(TOS_MsgPtr pmsg)
{   //réception du noeud puits des données agrégées par le CH

        // le noeud PUITS reçoit l'agrégation du CH
        //l'energie du CH se décrémente aprés l'envoie de l'aggrégation de la température

        call NBRECPNNECTimer.start(TIMER_ONE_SHOT,30000);
        //Timer apres lequel le noeud PUITS calcule la somme de tous les noeuds connectés
}
/*************************************************************************/
/*************************************************************************/
event result_t ReqRelayTimer.fired()       //Timer apres lequel les noeuds apres réception du nouveau round du PUITS
{                                          //le renvoie à leur voisins proche, ce dernier fera de meme

      call LEACH_SendMsg.send(TOS_BCAST_ADDR,sizeof(struct PUITS),&buffer);
}
/*************************************************************************/
/*************************************************************************/

event result_t Timer.fired()      //Timer utilisé pour les LED lorsque les noeuds sont élu CH
{  // activer le LED rouge
}
/*************************************************************************/
/*************************************************************************/
```

```
event result_t AnnonceTimer.fired()      //Timer apres lequel les noeuds annoncent qu'ils sont CH
{
       // activer les LEDs
       // le CH actuel annonce
       // décrémenter l'energie du CH aprés l'annonce

       call ANNONCE_SendMsg.send(TOS_BCAST_ADDR,sizeof(struct PUITS),&buffer);
       //reception des noeuds membres des nouveau CH
}
/*****************************************************************************/
/*****************************************************************************/
event result_t OrganisationTimer.fired()  //Timer apres lequel un noeud membre prévienne
{                                         // son Chef qu'il fera partie de son Cluster
       Envoie_MEMBRE_CH();                // annoncer à un CH d'etre choisi
}
/*****************************************************************************/
/*****************************************************************************/
event result_t OrdonnecementTimer.fired()       //Timer pour  apres lequel les noeuds membres envoient leurs données
{                                               //pendant leurs SLOT attribué par le CH
      // le noeud a envoyé la valeur captée

      call ORGANISATION_SendMsg.send(ptrM->ID_CH,sizeof(struct MEMBRE),&buffer);
      //envoi des noeuds aux CH auxquels ils comptent appatenir
}
/*****************************************************************************/
/*****************************************************************************/

event result_t AggregerTimer.fired()    //Timer apres lequel les CH envoient la donnée au noeud PUITS aprés agrégation
{
        //la donnée aggregée à envoyer au noeud PUITS = temp_moyenne/Table_Membre_Reelle

        call  AGGREGATION_SendMsg.send(BASE_STATION_ADDRESS,sizeof(struct CLUSTER_HEAD),&buffer);
        //envoi  du CH les données agregées au noeud puits
}
/*****************************************************************************/
/*****************************************************************************/
event result_t NBRECPNNECTimer.fired()
{//Timer pour apres lequel le noeud PUITS calcule la somme de tous les noeuds connectés
}
/*****************************************************************************/
/*****************************************************************************/
event result_t RoundTimer.fired()
{//Timer apres lequel le noeud PUITS annonce le nouveau round
}
/*****************************************************************************/
/*****************************************************************************/
event result_t LEACH_SendMsg.sendDone(TOS_MsgPtr msg, result_t success)
{
}
/*****************************************************************************/
/*****************************************************************************/
```

```
event result_t ANNONCE_SendMsg.sendDone(TOS_MsgPtr msg, result_t success)
{//reception des noeuds membres des nouveaux CHs
}
/****************************************************************************/
/****************************************************************************/
event result_t ORGANISATION_SendMsg.sendDone(TOS_MsgPtr msg, result_t success)
{//envoi des noeuds aux CH auxquels ils comptent appatenir
}
/****************************************************************************/
/****************************************************************************/
event result_t SLOT_SendMsg.sendDone(TOS_MsgPtr msg, result_t success)
{// envoi du CH des slots
}
/****************************************************************************/
/****************************************************************************/
event result_t AGGREGATION_SendMsg.sendDone(TOS_MsgPtr msg, result_t success)
{ //envoi du CH les données agrégées au noeud puits
}
/****************************************************************************/
/****************************************************************************/
}// fin de l'implémentation
```

IV.4.3 Environnement d'exécution du simulateur

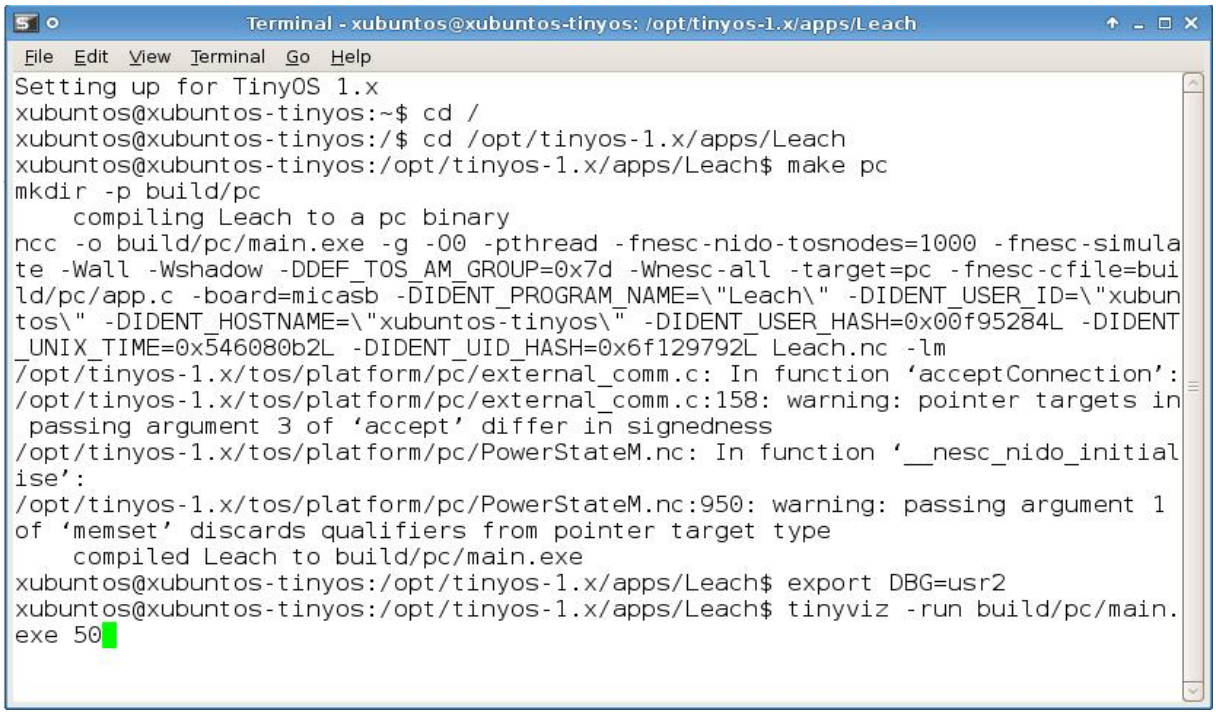

Figure IV.3: Environnement d'exécution du simulateur

- Tout d'abord, on accède au répertoire HOME par la commande suivante : **cd /**
- Après, on met le chemin de notre application : **cd opt/tinyos-1.x/apps/Leach** pour accéder à l'application « LEACH++».
- Ensuite, on la compile par la commande : **make pc**.
- Et enfin on l'exécute par la commande : **export DBG=usr2**, et la commande **Tinyviz –run build/pc/main.exe nbre_capteurs.**

IV.4.4 Déroulement :

Dans cette partie, nous expliquons et nous déroulons les phases de l'algorithme LEACH++ en faisant appel à TinyViz. Un fichier de configuration est créé et permet à TinyViz de se lancer avec les paramètres spécifiés. Ces derniers représentent : le nombre et l'emplacement des capteurs, la durée de la simulation et les plugins que nous souhaitons activer dès le début de la simulation comme Debug Messages. A propos des captures d'écran de TinyViz, nous nous limitons, dans cette étape, à la partie où l'on peut visualiser les capteurs.

1. Déclenchement du nouveau round et annonce des CHs: La figure VI-4 représente les transmissions broadcast qui se passent durant les différentes étapes de l'algorithme LEACH++. Une transmission broadcast est repérée par un cercle bleu. Le nœud puits envoie un broadcast aux nœuds voisins pour l'annonce du round. Ses voisins prennent le relai en envoyant à leur tour selon une transmission broadcast. De plus, nous pouvons voir que le nœud 37 est élu CH. Cet évènement est marqué par l'activation du LED rouge des CH. Ensuite, le CH 37 diffuse une annonce pour signaler son statut comme dans la figure IV.4.

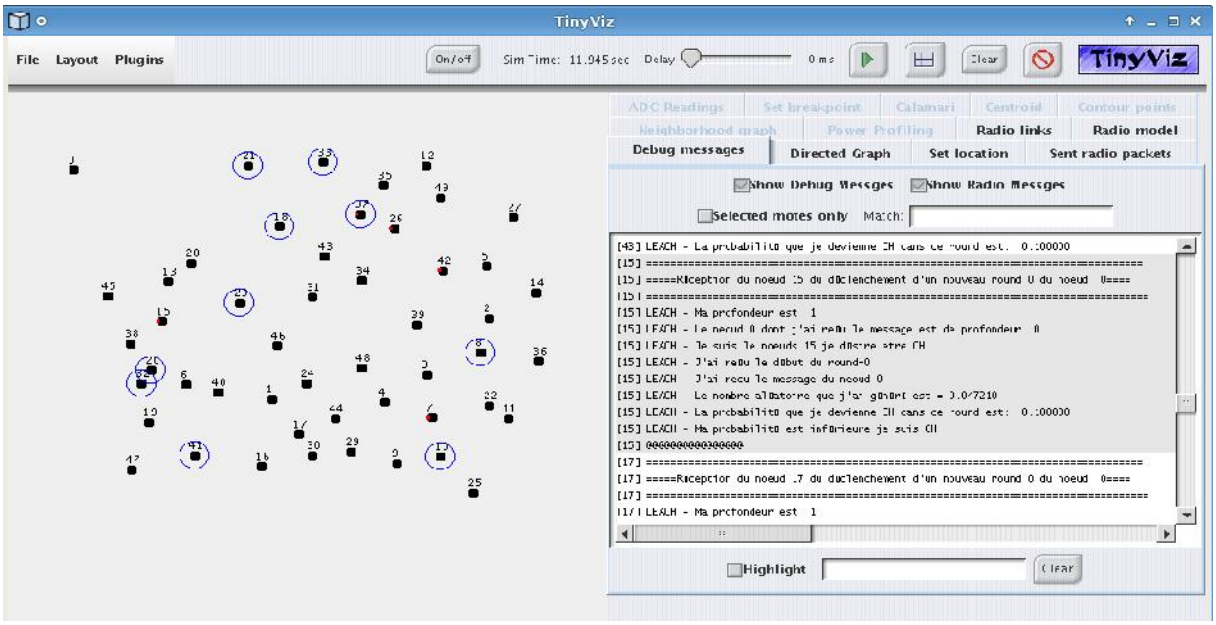

Figure IV.4 : Déclenchement et relai du nouveau round, annonce du CH 15.

2. Formation de clusters et envoi des données:

La figure VI.5 représente quelques transmissions unicast qui se passent durant les différentes étapes de l'algorithme LEACH. Une transmission unicast est repérée par une flèche.

Durant la première étape, les nœuds non-CH répondent à l'annonce du CH le plus proche. La figure suivante illustre la formation du cluster du CH 37**.** Quant à la seconde étape, chaque membre capte une donnée ; dans notre cas il s'agit de la température et attend le début de son slot pour qu'il puisse l'envoyer à son CH.

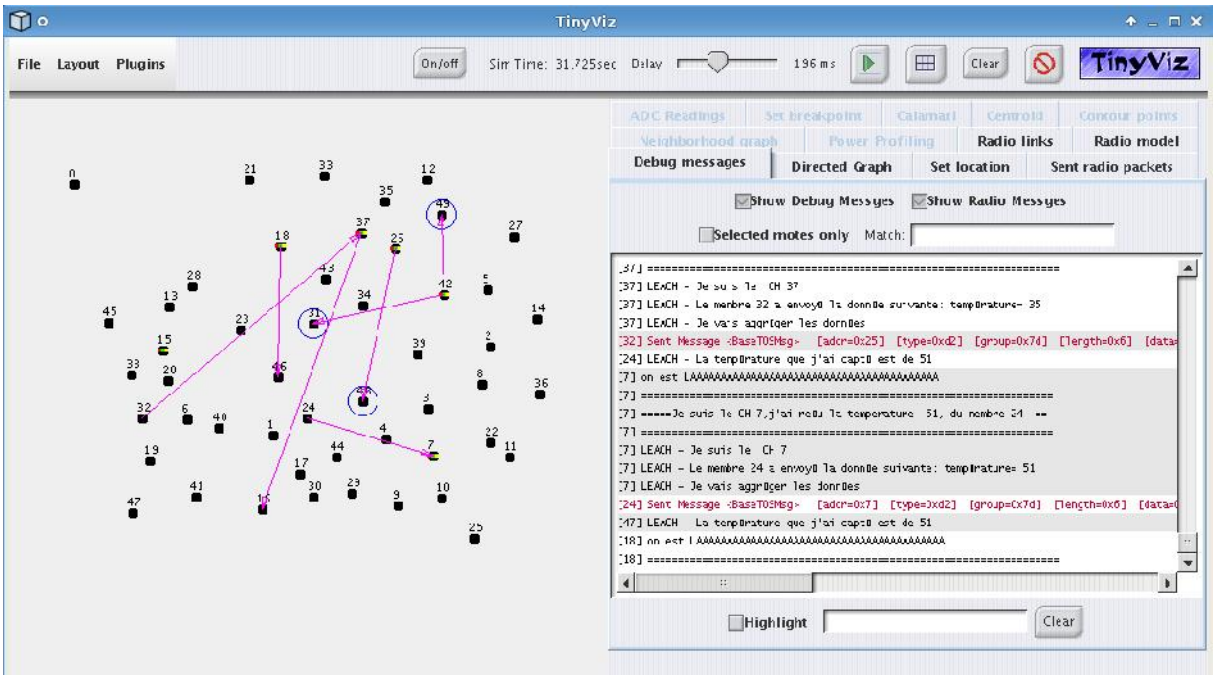

Figure IV.5 : Formation de groupes et envoi des données.

3. Envoi des résultats d'agrégation des températures au nœud puits : dans la figure IV.6, le CH 37 agrège les températures reçues et envoie son résultat d'agrégation au nœud puits comme dans la figure suivante.

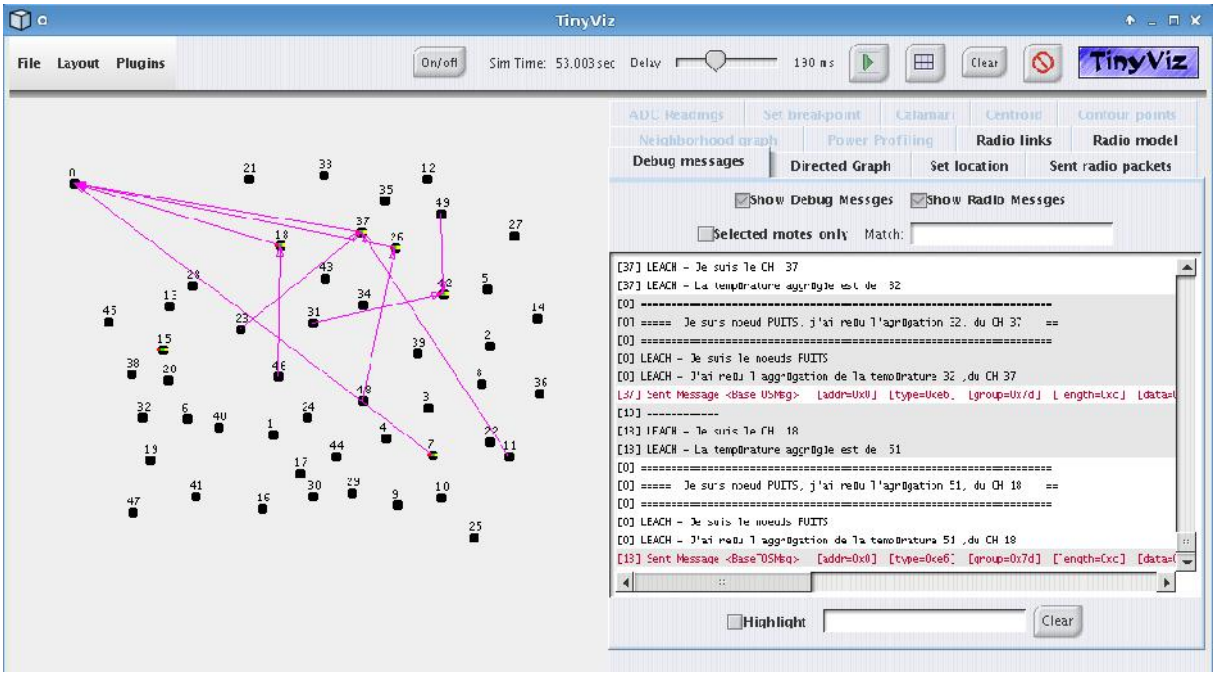

Figure IV.6 : Envoi du résultat d'agrégation du CH au nœud puits.

IV.5 Résultats:

Dans cette section, nous allons présenter les résultats que nous avons obtenus jusqu'à maintenant selon ces différents critères de performances :

IV.5.1. Critères de performances:

Pour la comparaison de notre protocole avec le protocole LEACH, nous prenons en compte les métriques suivantes:

- **Moyenne d'énergie consommée:**

  La faible autonomie en énergie est le point le plus crucial dans les RCSF. Ce qui fait que l'un des principaux critères de performance pour un capteur est la durée d'utilisation efficace de son énergie embarquée avant de l'épuiser. Parce qu'une fois qu'elle sera épuisée, il n'ya pas d'autre moyen de recharger à nouveau sa batterie, s'il a été déployé à des endroits inaccessible et n'utilisant pas de cellules solaires.

  Ainsi l'Energie Consommée (MEC) mesure en moyenne la quantité d'énergie consommée par les capteurs, elle est obtenue par la formule mathématique suivante :

$$\mathbf{M.E.C} \quad \frac{\sum \mathbf{Ei}\ (i=1,\ n)}{\mathbf{n}}$$

Ou :

Ei: l'énergie consommée pour un capteur i, elle représente la différence entre l'énergie initiale et son énergie résiduelle.

n: le nombre de capteurs dans le réseau.

- **Ecart type de la consommation:**

  En mathématique, l'écart type est une quantité réelle positive, éventuellement infinie, utilisée dans le domaine des probabilités pour caractériser la répartition d'une variable aléatoire autour de sa moyenne.

Dans notre cas, on l'a calculé de la manière suivante :

$$\textbf{Ecartype} = \sqrt{\frac{\sum_{(i=1,\,n)} (\textbf{M.E.C} - E_i)^2}{n}}$$

ou :

M.E.C : la moyenne de l'énergie consommée.

E_i : l'énergie consommée par un nœud i du réseau.

n : nombre de nœuds du réseau.

IV.5.2. Discussion des résultats:

Dans ce qui suit, nous allons présenter et analyser les résultats de simulation obtenus suivant les critères de performance discutées précédemment. Les simulations sont effectuées sur des capteurs déployés de manière aléatoire. Dans tous les scénarios de simulations générées, les deux protocoles LEACH et LEACH++ sont comparés, pour différencier leurs performances. Afin d'examiner l'influence de la densité du réseau sur les performances des deux protocoles, nous avons mené des simulations en variant le nombre de nœuds déployés dans le réseau (de 50 à 200 nœuds par un pas de 50 nœuds) augmentant ainsi la densité du réseau.

- **Moyenne d'énergie consommée (MEC)**

Protocoles / Nombre de Nœuds	LEACH	LEACH++
50	1645	1600,47
100	1877,24	1497,55
150	1727,18	1444,10
200	1827,62	1810,16

Tab. IV.1.Moyenne d'énergie consommée.

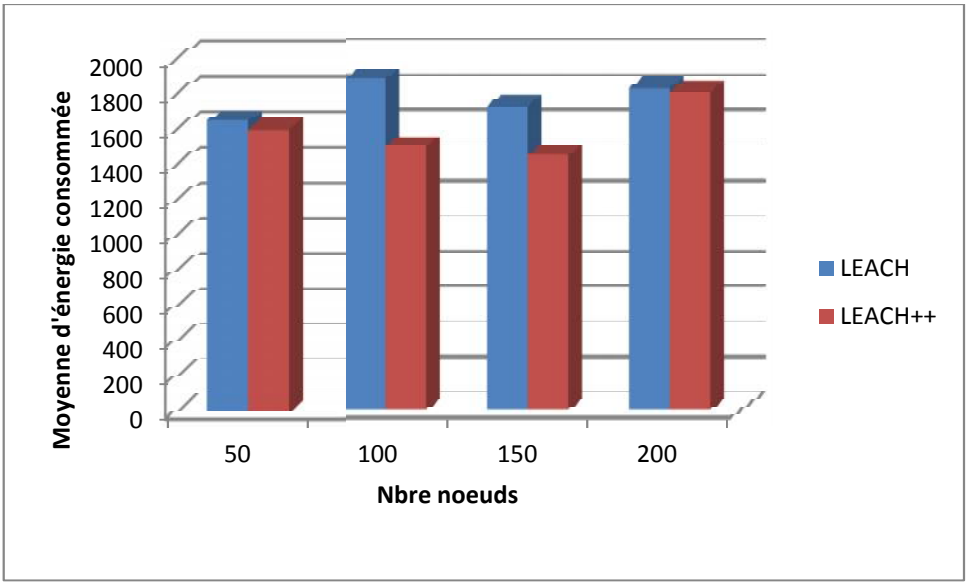

Figure IV.7 : Moyenne d'énergie consommée par les nœuds.

Le tableau **(Tab.IV.1.)** représente les résultats de la moyenne d'énergie consommée **(M.E.C)** obtenus par l'énergie consommée pour n capteur i (E_i)/ le nombre de capteurs n**.**

Le graphe de la figure **(Fig.IV.7)** représente les résultats de la moyenne d'énergie consommée **(M.E.C)** en fonction du nombre de nœuds du réseau. Nous pouvons remarquer sur le graphe que le protocole LEACH++ consomme moins d'énergie en le comparant avec le protocole LEACH de base.

On peut donc dire que le protocole LEACH++ est plus performant que le protocole LEACH, en marquant moins de moyenne d'énergie consommée.

Ce gain en énergie en LEACH++ revient au fait qu'un CH adjoint est élu dès que le Cluster Head cesse de fonctionner.

Ecart type de la consommation :

Dans notre cas nous avons calculé l'écart type de la moyenne de l'énergie consommée pour évaluer l'équilibrage de la consommation de l'énergie dans le réseau.

Protocoles / Nombre de nœuds	LEACH	LEACH++
50	219,3	209,66
100	273,46	218,02
150	260,42	197,57
200	294,96	265,19

Tab. IV.8 : l'écart type de l'énergie consommée.

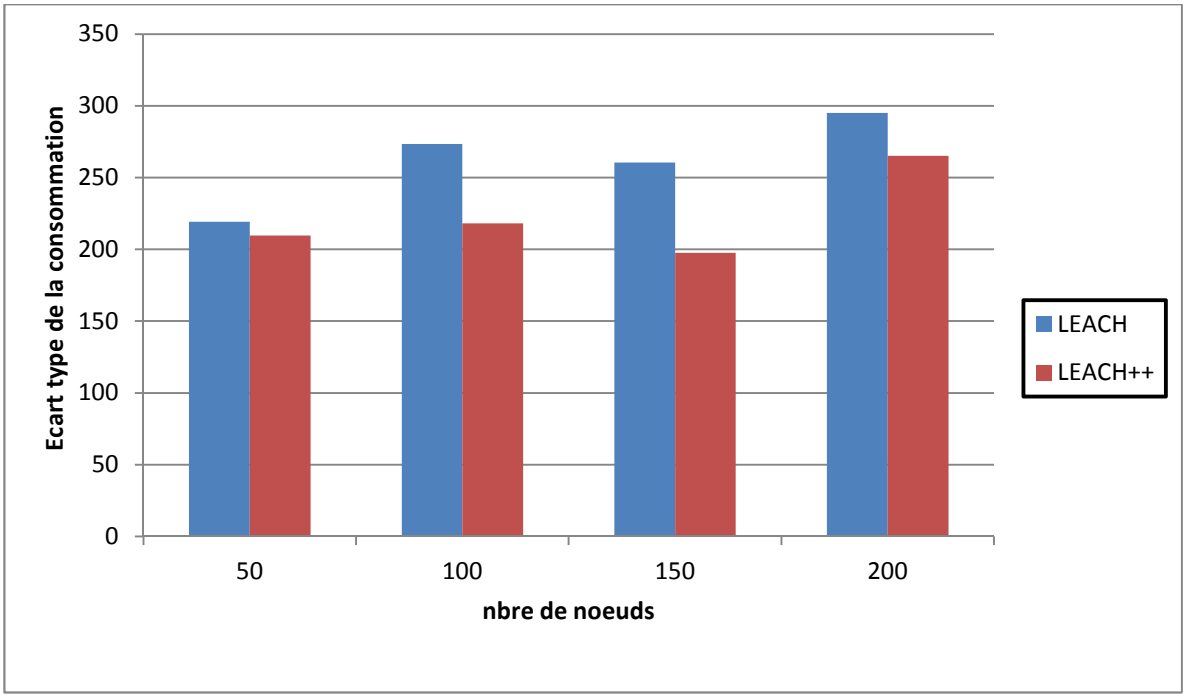

Figure IV.6.Ecart type de l'énergie consommée.

Le graphe de la figure **(Fig. IV.8)** représente les résultats de l'écart type en fonction du nombre de nœuds du réseau. Nous constatons que l'écart type de l'énergie consommée est plus important dans le protocole LEACH comparé au protocole LEACH++.

Ces résultats sont évidents du moment que l'énergie consommée varie en fonction du nombre de nœuds dans le réseau et la moyenne d'énergie consommée par LEACH++est la moins grande.

IV.6 Conclusion

Dans ce chapitre, nous avons présenté l'environnement pour implémenter et évaluer le protocole LEACH++. Dans cet environnement, on trouve TinyOS qui est un système d'exploitation léger dédié pour les réseaux de capteurs, le langage NesC qui est un langage orienté composant et TOSSIM qui est un simulateur pour les RCSF.

L'évaluation du protocole LEACH, nous a permis de déduire que ce protocole n'est pas efficace pour l'économie d'énergie. Dans cette optique, pour pallier cette limite, nous avons proposé une version améliorée de ce protocole appelée LEACH ++ de telle sorte qu'il soit efficace et fiable.

Nous avons effectué une étude comparative entre les deux protocoles selon différents métriques, les résultats des tests qu'on a obtenus jusqu'à maintenant ont montré que notre solution prolonge la durée de vie du réseau.

Conclusion générale

La conception des réseaux de capteurs est fortement influencée par la limitation de la ressource énergétique disponible au niveau des nœuds capteurs. Actuellement, la plupart des travaux de recherche sur ce type de réseaux, sont consacrés à la conception des protocoles de routage visant à minimiser l'énergie inhérente aux communications qui sont la source principale de consommation d'énergie afin d'optimiser la durée de fonctionnement du réseau. Dans cette optique, le routage hiérarchique s'est présenté comme étant une solution prometteuse pour conserver l'énergie des nœuds, et faciliter la transmission des données capturées dans le réseau vers la station de base. Selon ce type de routage, les nœuds du réseau sont organisés en clusters gérés par un seul nœud Cluster Head.

Quand le CH cesse de fonctionner, le routage hiérarchique devient incapable d'assurer l'arrivée des données des nœuds CH à la station de base. Dans ce mémoire, le CH Adjoint remplace le CH, afin de résoudre ce problème.

Dans ce mémoire, nous avons réalisé une étude pour atteindre un routage efficace et fiable dans les réseaux de capteurs sans fil. Cet aspect est fondamental pour ce genre de réseau où le routage se réalise en collaboration avec les différents nœuds du réseau. De ce fait, un protocole de routage doit prendre en compte les contraintes matérielles d'un capteur : une batterie faible, une capacité de stockage modeste, une bande passante faible, etc.

L'approche du clustering qui permet de partitionner le réseau en zones, est une approche prometteuse. Pour atteindre cet objectif, nous avons proposé une amélioration du protocole de routage hiérarchique nommé LEACH basé sur une topologie structurée en zones.

Afin de valider les améliorations apportées par notre protocole en termes de prolongement de la durée de vie du réseau, nous avons simulé le fonctionnement de notre algorithme dans le système d'exploitation TinyOS en utilisant le simulateur TOSSIM et l'interface graphique TinyViz puis on a comparé les résultats fournis avec le protocole LEACH.

Les résultats fournis par notre implémentation du protocole LEACH++ permet d'assurer une livraison fiable des données dans les RCFS.

Enfin, comme perspectives nous envisageons d'améliorer les performances de notre protocole de routage et l'implémenter sur des capteurs réels.

1. **[A01]** Akyildiz, I.F; W. Su, Sankarasubramaniam, E. Cayirci, « Aservey on Sensor Network », IEEE Communication Magazine, Auguste, 102_114(2002).
2. **[A02]** Qinghua Wang , Ilangko Balasingham, « Wireless Sensor Networks _An Introduction » Wireless Sensor Networks: Application-Centric Design, Geoff V Merrett and Yen Kheng Tan (Ed.), ISBN: 978-953-307-321-7, InTech, 2010.
3. **[A03]** Ben L. Titzer, J. Palsberg, « Nonintrusive Precision Instrementation of Microcontroller Software », ACM, New York, NY, ETATS-UNIS, Volume 40 Issue 7, Page 59 _ 68, July2005.
4. **[A04]** H .Karl, A. Willig, « Protocols and Architectures of Communication », (ISCC'03), IEEE COMPUTER SOCIETY, 2003.
5. **[A05]** A. Montoya, D. C. Restrepo, D.A. Ovalle, «Artificial Intelligence for Wireless Sensor Networks Enhancement » InTech, 2010.
6. **[A06]** I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Livre: «Wireless sensor networks: a survey », Broadband and Wireless Networking Laboratory, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta USA. December 2001.
7. **[A07]** Sayad Maya, « Energy Efficient Protocol (EEP) : un protocole de routage efficace en énergie pour réseaux de capteurs sans fil », mémoire de fin d'études, Ecole Nationale Supérieure d'Informatique (ESI), Algérie, 2009.
8. **[A08]** Kamal BEYDOUN « conception d'un protocole de routage hierarchique pour les réseaux de capteurs », thèse, L'U.F.R DES SCIENCES ET TECHNIQUES DE L'UNIVERSITE DE FRANCHE-COMTE, 2010.
9. **[A09]** I. Akyildiz, W. Su, E. Cayirci, Y. Sankarasubramaniam. « A survey on sensor networks», *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102-114, Georgia Institute of Technology, Atlanta, USA. Août 2002.
10. **[A10]** H. Cam, S. Ozdemir, P. Nair, and D. Muthuavinashippan, «ESPDA: Energy-Efficient and Secure Pattern Based Data Aggregation for Wireless Sensor Networks », in press, IEEE Sensor, Toronto, Canada, 2003.
11. **[A11]** Sofiane MOAD « Optimisation de la consommation d'énergie dans les sréseaux de capteurs sans fil », Rapport de stage, Université de IFSIC-Rennes, Laboratoire de recherche : DYONISOS-IRISA, 2007.
12. **[A12]** DARPA IPTO, SensIT : Sensor Information Technology program, 1999.
13. **[A13]** Adel CHOUHA « Traitement et transfert d'images par réseau de capteurs sans fil », Mémoire de Magistère, Université Hadj Lakhder-Batna, 2010.
14. **[A14]** **Andrews, P. Johnson and D.C**. Remote continuous monitoring in the home. Telemedicine and Telecare. June 1996, Vol. 2, 2, pp. 107-113.
15. **[A15]** E. M. Petriu, N.D. Georganas, D.C. Petriu, D. Makrakis, and V.Z. Groza. Sensor_based information appliances. IEEE Instrumentation Measurement Magazine. December 2000, Vol. 3, 4, pp. 31-35.
16. **[A16]** MEDJOUM Khaled « Les Système Embarqué TinyOS », mémoire de fin d'études, Université de Brest, Faculté de Sciences et Technologie, Département Informatique.

17. **[BO1]** W.R. Heinzelman, A. Chandrakasan, H. Balakrishnan. « Energy-efficient Communication Protocol for Wireless Microsensor Networks ». Proceedings of the IEEE Hawaii International Conference on System Sciences. 2000, Vol. 2, p. 10.
18. **[BO2]** Lindsey, S. Raghavendra, C.S. PEGASIS: Power-efficient gathering in sensor information systems. IEEE Aerospace Conference Proceedings. 2002, Vol. 3, pp. 3-1130.
19. **[BO3]** XUE Yong AGUILAR Andres, GONZALEZ Andres, BRROUX Mickaël, «Agrégation de données dans les réseaux de capteurs », projet (rapport final),univérsité technologie Compiène, 10/2010.
20. **[BO4]** S. Hedetniemi and A. Liestman, « A survey of gossiping and broadcasting in communication networks », Networks, Vol. 18, No. 4, pp. 319-349, 1988.
21. **[BO5]** C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In MOBICOM, pages 56–67, 2000.
22. **[BO6]** W. Heinzelman, J. Kulik, and H. Balakrishnan, « Adaptive protocols for information dissemination in wireless sensor networks », in the Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99), Seattle, WA,August 1999.
23. **[BO7]** R. Shah, J.Rabaey, Article: « Aware routing for low energy ad hoc sensor networks », in Proceedings of the IEEE Wireless Communications and Networking Conference, Orlando, FL, March 2002
24. **[BO8]** Boubiche Djallel Eddine, « Protocole de routage pour les réseaux de capteurs sans fil », mémoire de fin d'étude, Université de l'Hadj Lakhdar, Batna, 2007.
25. **[BO9]** Jamal N. Al-Karaki Ahmed E. Kamal « Routing Techniques in Wireless Sensor Networks: A Survey », Dept. of Electrical and Computer Engineering Iowa State University, Ames, Iowa 50011.
26. **[B10]** Y. Yu, D. Estrin, and R. Govindan, « Geographical and Energy-Aware Routing: A Recursive Data Dissemination Protocol for Wireless Sensor Networks », UCLA Computer Science Department Technical Report, UCLA-CSD TR-01-0023, May 2001.
27. **[B12]** C. E. Perkins & P. Bhagwat « Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers », dans *Proc. ACM SIGCOMM' 94 Conference on Communications Architectures, Protocols and Applications*, 1994, p. 234–244.
28. **[B13]** T.-W. Chen & M. Gerla « Global State Routing: A new routing scheme for ad-hoc wireless networks », dans *Proc. IEEE Int'l Conf. on Communications, (ICC 1998)* (Atlanta, GA, USA), 1998, p. 171–175.
29. **[B14]** S. Murthy & J. J. Garcia-Luna-Aceves – « An efficient routing protocol for wireless networks », *Mob. Netw. Appl.* **1** (1996), no. 2, p. 183–197.
30. **[B15]** C.-C. Chiang, H.-K. Wu, W. Liu & M. Gerla – « Routing in clustered multihop,mobile wireless networks with fading channel », dans *Proc. IEEE Singapore Int'l Conf. on Networks (SICON'97)* (Singapore), 1997, p. 197–211.
31. **[B16]** D. B. Johnson & D. A. Maltz « Dynamic source routing in ad hoc wireless networks », in *Mobile Computing* (T. Imielinski & H. Korth, éds.), vol. 353, Kluwer Academic Publishers, 1996, p. 153–181.

32. **[B17]** D. A. Johnson, D. B. Maltz & J. Broch – « DSR: The dynamic source routing protocol for multihop wireless networks », ch. 5, p. 139–172, Addison-Wesley, 2001.
33. **[B18]** C. E. Perkins, E. M. Royer, S. R. Das. Ad hoc on demand distance vector (aodv) routing. In IETF, Internet Draft, draft-ietf-manet-aodv-05.txt. [En ligne] 2000.
34. **[B19]** Z. J. Haas – « A new routing protocol for the reconfigurable wireless networks», dans *Proc. 6th IEEE Int'l Conf. on Universal Personal Communications*
35. **[B20]** Hamma, T. Katoh, T. Bista, B.B. Takata, T. An Efficient ZHLS Routing Protocol for Mobile Ad Hoc Networks. 17th International Conference on Database and Expert Systems Applications. 2006, pp. 66-70.
36. **[B21]** J. Kulik, W. R. Heinzelman, and H. Balakrishnan, « Negotiation-based protocols for disseminating information in wireless sensor networks », Wireless Networks, Volume: 8, pp. 169-185, 2002.
37. **[B22]** J.N Al-Karaki, A. E. Kamal, « Routing Techniques in Wireless Sensor Networks: A Survey », NEC Laboratories America, IEEE Communications Magazine, Mars 2005.
38. **[B23]** R. Jurdak, « Wireless Ad Hoc and Sensor Networks: A Cross_Layer Design perspective University College Dublin, 2007.
39. **[C01]** W.R.Heinzelman, A. Chandrakasan, H. Balakrishnan, « An application-Specific Protocol Architecture for Wireless Microsensor Networks » in IEEE Tmnsactions on Wireless Communication (October 2002).
40. **[C02]** Mathieu Badnet, Nicolas Belloir «Réseaux de capteurs : Mise en place d'une plateforme de test et d'expérimentation », Master Technologie de l'Internet 1ère année, France, 2005/2006.
41. **[C03]** David Gay, Philip Levis, « TinyOS Programming », Livre, ISBN: 0521896061, Nombre de Pages: 264, Presse de l'université de Cambridge, 28 Juin 2006.