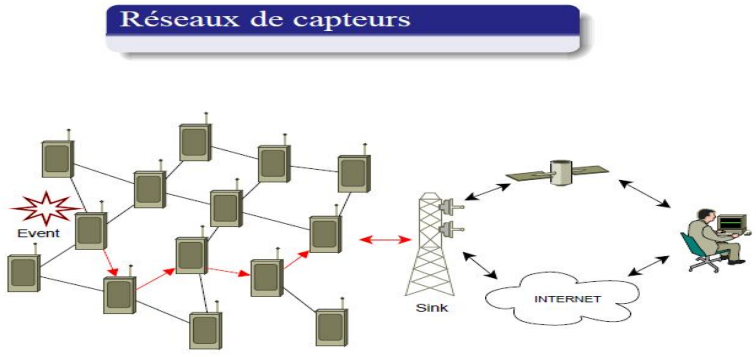
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET
DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI DE TIZI OUZOU
FACULTE DE GENIE ELECTRIQUE ET DE L'INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

MEMOIRE POUR L'OPTENTION DU DIPLOME MASTER II EN INFORMATIQUE

THEME

LE ROUTAGE DANS LES RESEAUX DE CAPTEURS SANS FILS



Proposé et dirigé par
Mr DAOUI M

Réalisée par

Mlle FERRAT HANANE

Mlle CHERDIOUI SABRINA

Remerciements

***C**'est grâce à la volonté que Dieu nous procure, celui qui nous a permis d'être à ce niveau d'études. Une détermination d'aller chaque fois plus loin dans le but d'acquérir plus de savoir et de compréhension, la seule chose qu'on espère est d'être toujours à la hauteur.*

***C**'est avec un grand plaisir qu'on réserve ces lignes en signe de gratitude et de reconnaissance à tous ceux qui ont contribués de près ou de loin à l'élaboration de ce travail.*

***O**n tient à remercier notre encadreur Mr DAOUI M pour son encadrement et pour l'encouragement et l'intérêt qu'il nous a apporté pour l'accomplissement de ce projet de fin d'étude.*

***O**n remercie vivement les membres du jury pour l'honneur qu'ils nous ont fait en acceptant de juger notre travail.*

***O**n tient également à remercier l'ensemble des enseignants pour tout ce qu'ils nous ont transmit tout au long de notre formation Licence Master et inchallah doctorat.*

***E**nfin on tient à exprimer nos remerciements pour toutes personnes qui ont de loin ou de prêt contribué à la réalisation de ce travail :*

A nos chers parents pour l'amour et le soutient dont ils nous ont entouré
A toute notre famille

***Q**u'ils sachent à travers ces quelques mots combien nous leurs sont reconnaissantes et combien on sait tout ce qu'on leurs doit.*

Merci à vous tous

Dédicaces

A mon père,
A ma mère,

Qui m'ont aimé et aimé me voir réussir.
Qui m'ont soutenu le long de mes études.
A vous deux, en témoignage de ma reconnaissance infinie, de ma gratitude et de mon amour éternel. Pour la patience et les sacrifices que vous avez endurés pour moi et dont je vous suis à jamais redevable.
Que Dieu vous garde afin que votre regard puisse suivre ma destinée.

A mes chers frères

Farid et Aziz pour leurs soutient durant toute ces années d'études

A ma très chère sœur Sihem

Qui ma aider pour surmonté toute difficulté, pour ces conseils précieux et sa présence et son encouragement à avancer et a réussir.

A ma belle sœur Warda et à mon adorable petite nièce Anaîs

A mon beau frère khaled

A toutes mes amies

A mon amie « Kamélia » qui a étais toujours présente pour moi

A mon amie «sabrina » que je remercie pour son soutient

A toute la promotion informatique de Master II 2010/2011

HANANE FERRAT

Dédicaces

Je me souviens de la parole de M. Ait –SIMMAR Belkacem chef librairie des offices des publications universitaire OPU à Tizi Ouzou, auquel je partais avec mon père acheter mes livres informatiques, il me disait : « tant que ta la volonté ma fille tu peux aller très loin ».

Mes sincères remerciements à M.Ait-Simmar Belkacem

A mes très chers parents

A mes trois chers frères Said, Mounir et Nassim,

A toute ma famille Cherdioui, Mallek, Houcini et Mokrani

A mes tantes Samia, Fatma, Naima, Malika et leurs maris Mezian, Aziz et Ali.

A Soraya, Hanane, Yanisse et Abedlkrim

Résumé

Dans la vie courante, l'utilisation des réseaux capteurs sans fil est de plus en plus demandée pour la supervision et la sécurité.

Les capteurs fonctionnent à basse tension et sont gérés par un système d'exploitation spécialisé. TinyOS est un système d'exploitation qui permet d'exécuter des applications sur les capteurs. Le NesC est le langage utilisé pour ce développement de ces applications.

La communication entre les différents noeuds capteurs se fait par ondes radios et l'acheminement des données à travers ces différents nœuds (capteurs) obéit à un protocole de routage. La qualité de ce protocole a une incidence directe sur les performances globale du réseau.

Le routage dans les réseaux de capteurs est une opération essentielle. Il permet aux nœuds éloignés de la station de base de récolter, de pouvoir acheminer leurs informations à cette dernière en les faisant passer d'un nœud à autre. Toutefois, à cause des caractéristiques des nœuds capteurs, pauvres en énergie et en puissance de calcul, les protocoles classiques ne sont pas facilement adaptables.

Dans cette optique, notre projet présentera la synthèse des protocoles de routages portés sur un réseau de capteurs sans fils, on s'intéressera aux caractéristiques essentielles de chaque protocole de routage. Nous y décrivons également l'évaluation sous TinyOS de deux protocoles AODV-adapté et LEACH pris comme exemples de simulation sur le simulateur TOSSIM.

Mots-clés: Réseaux de capteurs sans fil, réseaux Ad Hoc, routage, protocole de routage, TinyOS, Tossim, LEACH, AODV, AODV-adapté

Abstract

In everyday life, the use of wireless sensors Network is increasingly required for supervision and safety.

The sensors operate at low voltage and were managed by a specialized operating system. TinyOS is an operating system that allows running applications on the sensors. The Nesc is the language used for this development.

Communication between the various sensors is via radio waves and the flow of data through these different nodes (sensors) follows a routing protocol. The quality of this protocol has a direct impact on the global performance for network.

The Routing in sensor networks is an essential operation. It allows nodes distant from the base station to collect, can send their information to it by passing from one node to another. However, because of the characteristics of sensor nodes, low in energy and computing power, standard protocols are not easily adaptable.

In this context, our project will present a summary of routing protocols on a network of sensors without son will address the essential characteristics of each routing protocol. It also profiles the evaluation of both protocols in TinyOS AODV-adapte and LEACH as an example of simulation on the simulator Tossim.

Key words: Wireless Sensor Networks, Ad Hoc Networoks, routing, routing protocol, TinyOS, Tossim, LEACH, AODV, AODV-adapte.

SOMMAIRE

SOMMAIRE

CHAPITRE II : LE ROUTAGE DANS LES RESEAUX DE CAPTEURS SANS FIL

SOMMAIRE

SOMMAIRE

SOMMAIRE

CHAPITRE IV : TinyOS ET TOSSIM

SOMMAIRE

CHAPITRE V : IMPLEMENTATION DE LEACH ET AODV SUR TOSSIM

SOMMAIRE

SOMMAIRE

LISTE DES FIGURES

Chapitre I :

Chapitre II :

Chapitre III :

LISTE DES FIGURES

Chapitre IV :

Chapitre V :

LISTE DES FIGURES

LISTE DES TABLEAUX

Chapitre I :

Chapitre II :

Chapitre V :

LISTE DES GRAPHES

Chapitre V :

Liste des abréviations

ACK	*ACKnowledge*
ACQUIRE	*ACtive QUery forwarding In sensoR nEtworks*
ADV	*ADVertisement message*
APTEEN	*Adaptive Threshold sensitive Energy Efficient sensor Network protocol*
APS	*Application Support Sublayer*
AODV	*Ad-hoc On-Demand Distance Vector*
CDMA	*Code Division Multiple Access*
CH	*Cluster-head*
CBRP	*Cluster Based Routing Protocol*
CSMA	*Carrier Sense Multiple Access*
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance*
CTS	*Clear To Send*
DC	*Data-Centric*
DD	*Directed Diffusion*
DIFS	*Distributed Inter Frame Space*
DSDV	*Destination Sequence Distance Vector*
DSR	*Dynamic Source Routing*
DV	*Demand Distance*
EAR	*Energy Aware Routing*
ED	*Energie Disponible*
EED	*End-to-End Delay*
EN	*Energie Nécessaire*
ERP	*Energy Routing Parameter*
FDMA	*Frequency Division Multiple Access*
FSR	*Fisheye State Routing*
GAF	*Geographic Adaptive Fidelity*
GBR	*Gradient-Based Routing*
GEAR	*Geographic and Energy Aware Routing*
GPRS	*General Packet Radio Service*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communication*
GSR	*Global State Routing*
HPARP	*Hierarchical Power-Aware Routing Protocol*
IARP	*Intrazone Routing Protocol*
IBM	*International Business Machines*
ID	*IDentificateur*
IEEE	*Institute of Electrical and Electronics Engineers*
IERP	*Interzone Routing Protocol*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*

LEACH	*Low Energy Adaptive Clustering Hierarchical*
LDA	*Location Dependent Address*
LS	*Link State*
MAC	*Medium Access Control*
MANET	*Mobile Ad hoc NETwork*
MCFA	*Minimum Cost Forwarding Algorithm*
MECN	*Minimum Energy Communication Network*
MSG	*MeSsaGe*
NesC	*Network embedded system C*
NWK	*NetWorK*
OLSR	*Optimized Link State Routing Protocol*
OSPF	*Open Shortest Path First*
QoS	*Quality of Service*
PEGASIS	*Power-Efficient GAthering in Sensor Information Systems*
PHY	*PHYsique*
PT	*PuiTs*
RCSF	*Réseau de Capteurs Sans Fil*
RF	*Radio Frequency*
RIP	*Routing Information Protocol*
RREP	*Route Reply*
RREQ	*Route Request*
RTS	*Ready To Send ou Request To Send*
SAP	*Services Acces Point*
SAR	*Sequential Assignment Routing*
SB	*Station de Base*
SMACS	*Self-organizing Medium Access Control for Sensor networks*
SMP	*Sensor Management Protocol*
SPIN	*Sensor Protocols for Information via Negotiation*
SOP	*Self Organizing Protocol*
TCP	*Transmission Control Protocol*
TDMA	*Time Division Multiple Access*
TEEN	Threshold sensitive Energy Efficient sensor Network protocol
TTDD	*Two- Tier Data Dissemination*
TOSSIM	*TinyOS SIMulator*
TinyOS	*Tiny Operating System*
TinyViz	*Tiny VisualiZation*
UDP	*User Datagram Protocol*
UDP-Like	*User Datagram Protocol Like*
VGAR	*Virtual Grid Architecture Routing*
WSNs	*Wireless Sensor Networks*
ZC	*ZigBee Coordinator*
ZE	*ZigBee End Device*
ZR	*ZigBee Router*
ZDO	*Zigbee Device Object*
ZRP	*Zone Routing Protocol*

INTRODUCTION GENERALE

La recherche dans le domaine des capteurs subit actuellement une révolution importante, ouvrant des perspectives d'impacts significatifs dans de nombreux domaines d'applications (sécurité, santé, environnement, fabrication, télécommunication, robotique,…). Les nouvelles technologies permettent de réduire le coût et la consommation d'énergie et d'augmenter la précision et les performances des capteurs, des processeurs et des circuits spécifiques. Un nombre très important de capteurs peut donc être envisagé, intégré et organisé en réseau.

Par conséquent, il est important de développer des recherches permettant d'imaginer des réseaux denses, sans fils entre des nœuds hétérogènes et ayant pour rôles de collecter des données d'un environnement donné et de les diffuser au sein du réseau. Ce type de réseaux de capteurs pourrait avoir de très diverses applications. Pour que tels réseaux soient intéressants, il faut qu'ils respectent un certain nombre de contraintes. Tout d'abord, ils doivent être sans fils, ceci pour pouvoir être installés sans difficulté. Ensuite, les nœuds du réseau doivent être autonomes, pour les mêmes raisons de faisabilité et rentabilité. Les traitements des données internes vont également engendrer des consommations en énergie non négligeables dont il faut tenir compte. Le problème de la source d'énergie doit également être évoqué. Ces réseaux de capteurs doivent être Ad Hoc, tout d'abord dans un souci de simplicité d'installation, mais aussi et surtout dans le souci de permettre au réseau de rester opérationnel même après les défaillances de quelques nœuds causées fort probablement par le problème d'autonomie. Ils doivent pouvoir s'autogérer, en utilisant des protocoles permettant d'apprendre des éléments tels que : la topologie du réseau, le positionnement relatif des nœuds au sein du réseau, les routes possibles pour communiquer avec un tel nœud.

Les différents problèmes sont interdépendants, puisque les protocoles de ce type de réseaux Ad Hoc doivent aussi consommer le moins d'énergie possible. Et puisque la topologie est aléatoire, le nombre de messages échangés pour le contrôle peut devenir très important si on ne prend pas de précautions.

L'objectif de ce travail est d'étudier le mécanisme de routage spécifique au RCSF. Voir la possibilité de tirer profil des protocoles de routage disponibles dans les réseaux Ad Hoc et d'évaluer leurs adaptabilités au RCSF.

A titre d'exemple, nous implémenterons le protocole LEACH, très utilisé actuellement et le comparerons à une implémentation simplifié d'AODV.

INTRODUCTION GENERALE

La comparaison se fera sur la base de la consommation énergétique. Ce paramètre est très important dans l'étude de performance des RCSF.

Organisation du rapport :

Pour mener à bien notre travail et pour une démarche plus compréhensible nous avons élaboré le plan suivant qui est organisé en 5 chapitres :

Le ***premier chapitre*** serra consacré à l'étude des deux types des réseaux : AH HOC et les réseaux de capteurs sans fil et leurs architectures, leurs caractéristiques et un ensemble de domaines d'applications de ces deux réseaux.

Nous présenterons également l'alliance ZigBee qui est un élément très important pour assurer le routage dans les réseaux de capteurs sans fils .

Le ***deuxième chapitre*** portera sur le routage au niveau des RCSF. Ce chapitre nous permettra de mieux connaitre la manière de communiquer dans les RCSF ainsi l'ensemble des éléments indispensable pour réaliser cette communication.

Nous avons détaillé sur protocoles de routage, leurs hiérarchies et leurs topologies. Nous avons aussi décrit un ensemble de protocoles pouvant être utilisés dans les réseaux de capteurs sans fils en présentant vers la fin du chapitre le protocole LEACH.

Le ***troisième chapitre*** portera sur l'étude du protocole de routage AODV, en faisant appuie sur sa table de routage, son numéro de séquence, le format de ses messages ainsi que le principe de son fonctionnement.

Dans le ***quatrième chapitre*** nous allons aborder trois points essentiels, nous commencerons par le système d'exploitation TinyOS où nous allons préciser quelques descriptions de ce protocole ainsi que son architecture et l'ensemble d'équipement qui peuvent être supporté par ce système. Par la suite, nous allons passer au second point qui concerne le langage de programmation NesC en présentant ses fonctionnalités et une petite description de ses fichiers ainsi que la manière de les compiler. Nous terminons ce chapitre par le dernier point qui représente une description du simulateur TOSSIM de TinyOS.

Le ***dernier chapitre*** donnera les détails d'implémentation du protocole LEACH et AODV sur le simulateur TOSSIM, nous allons les évaluer en terme de consommation d'énergie.

Enfin, nous clôturerons par une conclusion générale et des perspectives pour l'avenir.

Des annexes sont ajoutées à la fin du rapport dans le but de plus de clarté :

INTRODUCTION GENERALE

De ce fait : l'annexe A serra consacré a l'installation de la plateforme TinyOS.

Annexe B : portera sur le fonctionnement du simulateur TOSSIM

Annexe C : sur le format des messages échangés entre les nœuds capteurs.

Annexe d : portera sur la définition des différentes fonctionnalités de la pile ZigBee.

CHAPITRE I :

LES RESEAUX AH HOC ET LES RESEAUX DE CAPTEURS SANS FIL.

Introduction

Les environnements mobiles ont connu un grand essor grâce à leur flexibilité d'utilisation. Les réseaux mobiles sans fil peuvent être classifiés en deux catégories : les réseaux avec infrastructure et ceux sans infrastructure (les réseaux ad-hoc).

Les réseaux ad-hoc MANET (Mobile ad-hoc Network) sont des réseaux mobiles sans infrastructure. Ce type de réseau possède une classe particulière qui est la classe des réseaux de capteurs sans fil.

Les réseaux de capteurs sont des systèmes auto configurables. Ils doivent pouvoir s'adapter à des situations imprévisibles. Pendant la conception d'un réseau de capteurs sans fil une précision acceptable des informations et une utilisation optimale des ressources sont recherchées.

Dans ce qui suit, nous allons étudier les réseaux ad-hoc et les réseaux de capteurs sans fil apparentés aux réseaux ad-hoc, leurs descriptions, leurs principales caractéristiques, leur architecture ainsi que leurs éventuelles applications. En outre, l'architecture de communication dans les réseaux de capteurs sera détaillée ainsi que les différences qui les distinguent des réseaux Ad Hoc traditionnels.

I Les réseaux Ad Hoc

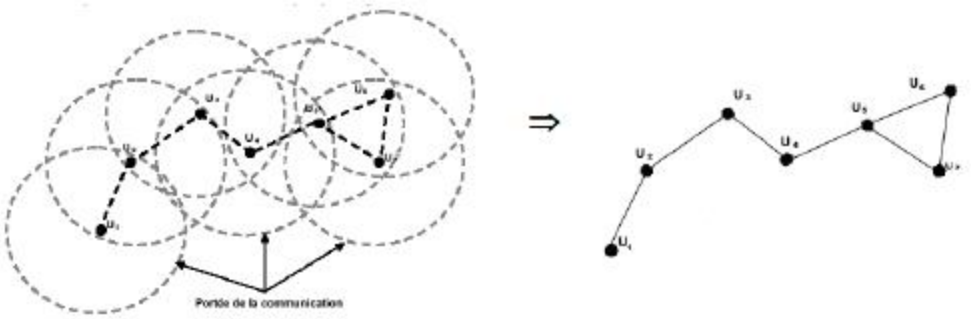
Le concept des réseaux Ad Hoc essaye d'étendre la notion de la mobilité à toutes les composantes de l'environnement mobile. Ici, contrairement aux réseaux basés sur la communication cellulaire, aucune administration centralisée n'est disponible. Ce sont les hôtes mobiles, eux même, qui forment, d'une manière ad hoc, une infrastructure du réseau. Aucune supposition n'est faite sur la taille du réseau ad hoc, théoriquement, le réseau peut contenir plusieurs milliers d'unités mobiles.

Les réseaux ad hoc sont idéals pour les applications caractérisées par une absence d'une infrastructure préexistante, tel que les applications militaires ou les autres applications de tactique comme les opérations de secours (incendies, tremblements de terre,…) et les missions d'exploration.

Un réseau ad hoc, appelé généralement MANET (Mobile Ad hoc Network), est une collection d'unités mobiles munies d'interfaces de communication sans fil, formant un réseau temporaire sans recourir à aucune infrastructure fixe ou à une administration centralisée. Dans de tels environnements, les unités se comportent comme des hôtes et/ou des routeurs.
Les nœuds des MANETs sont équipés d'émetteurs et de récepteurs sans fil utilisant des antennes qui peuvent être omnidirectionnelles (broadcast), directionnelles (point à point), ou une combinaison de ces deux types. Ils maintiennent d'une manière coopérative la connectivité du réseau, en fonction de leurs positions, la puissance de transmission et les interférences entre les canaux de communication.

I.1 Modèle d'un système de réseau Ad Hoc

A un instant t, un réseau Ad Hoc peut être modélisé par un graphe non orienté (Bidirectionnelle) Gt= (Vt , Et), où **Vt** représente l'ensemble des nœuds (c-a-d les unités mobiles), et **Et** représente l'ensemble des liens existants entre ces nœuds (figure I.1).
Si e=(u,v)€ Et , cela veut dire que les nœuds u et v sont en mesure de se communiquer directement à l'instant t.



a- système du réseau Ad Hoc **b- graphe modèle du réseau**

Figure I.1: modèle du réseau Ad Hoc [1].

La mobilité des nœuds appartenant à un réseau ad hoc fait que sa topologie peut changer à n'importe quel moment, ce qui entraîne les déconnexions fréquentes (figureI.2).

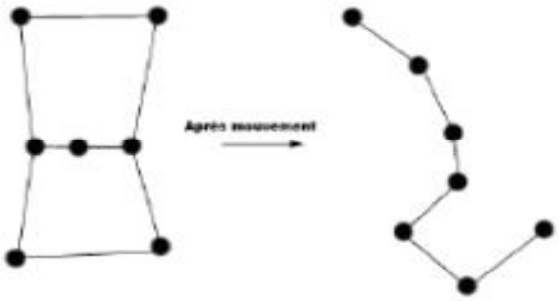


Figure I.2 : Changement de topologie dans les réseaux Ad Hoc [1].

I.2 Caractéristiques des réseaux Ad Hoc

Les réseaux Ad hoc sont principalement caractérisés par :

- *Des contraintes d'énergie* : Les hôtes mobiles sont alimentés par des sources d'énergie autonomes comme les batteries.
- *Une bande passante limitée* : Une des caractéristiques primordiales des réseaux basés sur la communication sans fil est l'utilisation d'un média de communication partagé.
- *Une topologie dynamique* : La topologie des réseaux Ad hoc change d'une manière fréquente et rapide à cause du déplacement arbitraire permanent des unités mobiles.

- *Une sécurité physique limitée* : Etant basés sur les communications sans fil, les réseaux Ad hoc sont plus sensibles aux attaques qui menacent les données transmises.

- *Communication multi sauts* : lorsqu'un nœud veut joindre un autre qui est hors de portée, les messages sont transmis par des nœuds intermédiaires.

I.3 Applications des réseaux Ad hoc

Les applications ayant recours aux réseaux ad hoc couvrent un très large spectre, incluant les applications militaires et de tactique, et plus simplement les applications de calcul distribué.

D'une façon générale, les réseaux ad hoc sont utilisés dans toute application où le déploiement d'une infrastructure réseau filaire est trop contraignant, soit parce qu'il est difficile à mettre en place, soit parce que la durée d'installation du réseau ne justifie pas de câblage à demeure.

II Les réseaux de capteurs sans fil

Un réseau de capteurs sans fil (RCSF), ou "Wireless Sensor Network" (WSN), est composé d'un ensemble d'unités de traitements embarquées, appelées "motes", communicant via des liens sans fil organisé en champs appelé **sensor fields**. Le but général d'un WSN est la collecte d'un ensemble de paramètres de l'environnement entourant les motes, telles que la température ou la pression de l'atmosphère, afin de les acheminer vers des points de traitement.

Un réseau de capteur sans fil est composé d'un ensemble de nœuds disposés d'une manière aléatoire dans l'environnement et pour cela l'utilisation des algorithmes d'auto organisation est très importante.

Pour assurer le bon fonctionnement du réseau un ensemble de contraintes doivent être prise en compte tel que la consommation d'énergie et l'utilisation des protocoles de routage pour s'assurer d'un cheminement meilleur des données entre les différents nœuds.

II.1Architecture des réseaux de capteurs

Puisque les RCSF se caractérisent par l'absence d'une infrastructure déterminée au préalable, les nœuds capteurs la construisent tout en permettant l'interaction avec l'environnement où ils appartiennent et en répondant aux différentes requêtes venant des utilisateurs ou des réseaux externes.

II.1.1 Organisation des nœuds

Un RCSF est composé d'un ensemble de nœuds capteurs. Ces nœuds capteurs sont organisés en champs « sensor fields » (voir figure suivante). Chacun de ces nœuds a la capacité de collecter des données et de les transférer au nœud passerelle (dit "sink" en anglais ou puits) par l'intermédiaire d'une architecture multi-sauts. Le puits transmet ensuite ces

données par Internet ou par satellite à l'ordinateur central «Gestionnaire de taches» pour analyser ces données et prendre des décisions.

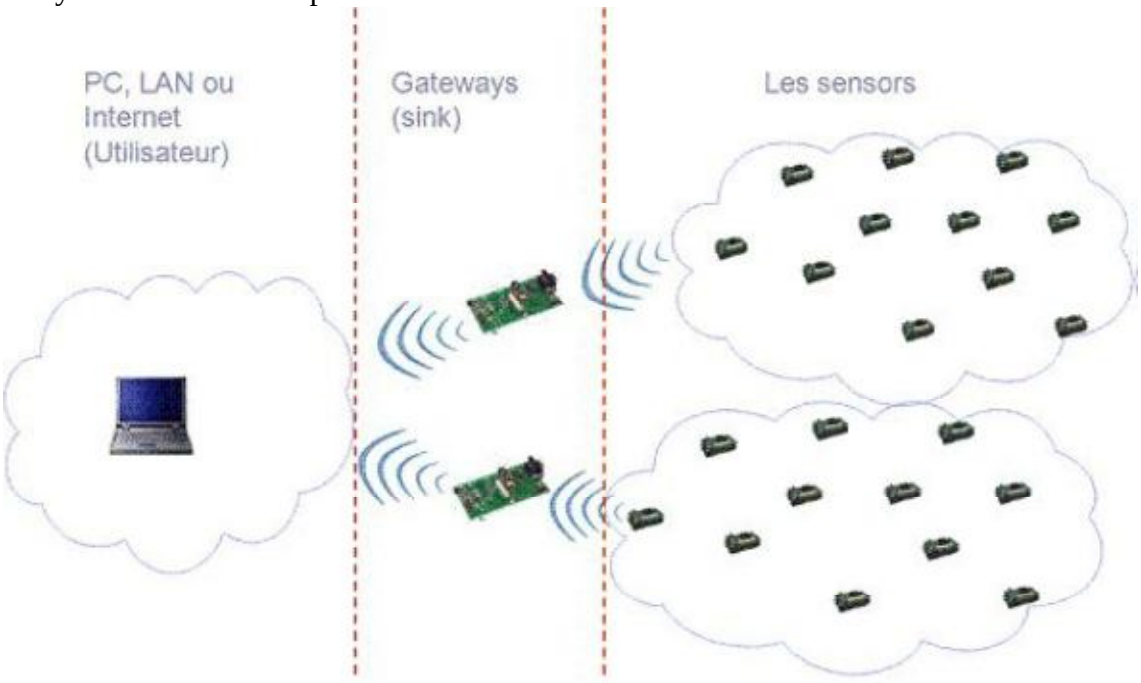


Figure I.3 : Architecture d'un RCSF [2].

II.1.2 Anatomie d'un nœud capteur

Un nœud capteur (dit aussi "mote" en anglais) est composé principalement d'un processeur, une mémoire, un émetteur/récepteur radio, et une pile (voir figure suivante). Il existe plusieurs modèles commercialisés dans le marché. Parmi les plus célèbres, les "mote" MICAx et TelosBe de Crossbow**[3].**

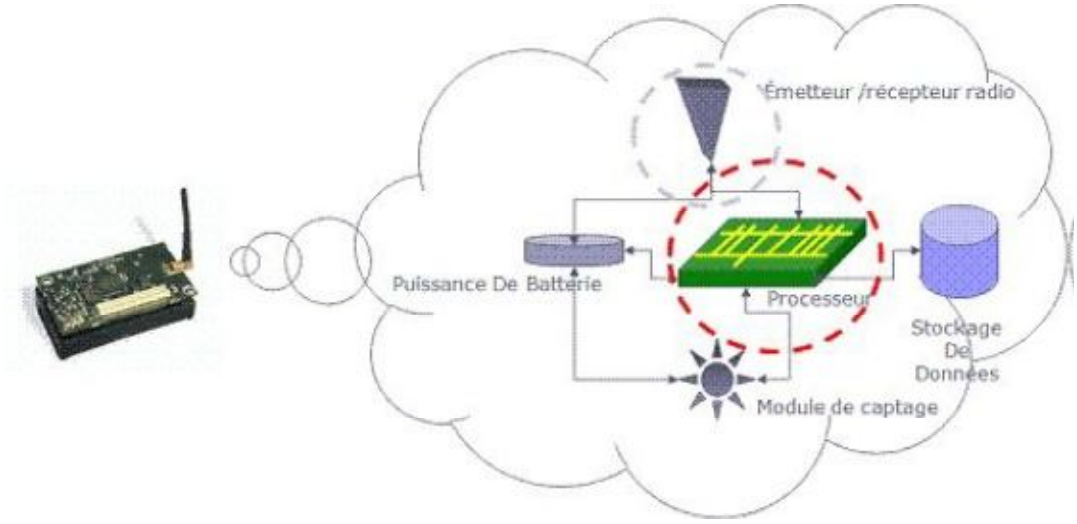


Figure I.4: Anatomie d'un nœud capteur.

II.1.3 Architecture protocolaire

Dans le but d'un établissement efficace d'un RCSF, une architecture en couches est adoptée afin d'améliorer la robustesse du réseau. Une pile protocolaire de cinq couches est donc utilisée par les nœuds du réseau. Citons la couche application, la couche transport, la couche réseau, la couche liaison de données et la couche physique.

De plus, cette pile possède trois plans (niveaux) de gestion : le *plan de gestion des tâches* qui permet de bien affecter les tâches aux nœuds capteurs, *le plan de gestion de mobilité* qui permet de garder une image sur la localisation des nœuds pendant la phase de routage, et, *le plan de gestion de l'énergie* qui permet de conserver le maximum d'énergie.

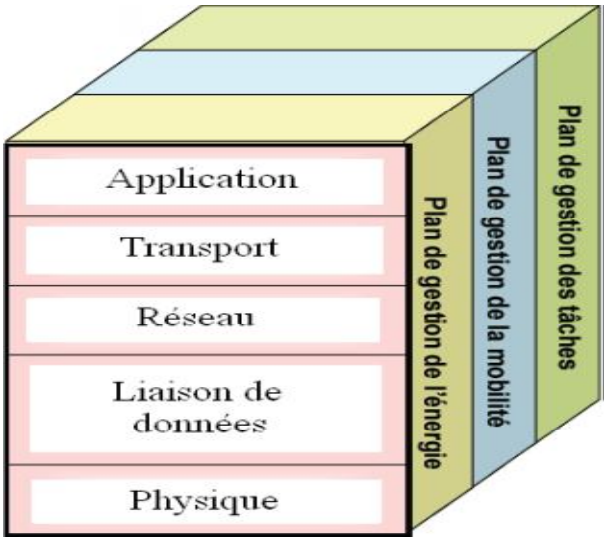


Figure I.5 :La pile protocolaire dans les réseaux de capteur [4].

II.1.3.1 Couches de la pile protocolaire [25]

- **Couche application :** Elle assure l'interface avec les applications. Il s'agit donc de la couche la plus proche des utilisateurs, gérée directement par les logiciels. Parmi les protocoles d'application, nous citons : SMP *(Sensor Management Protocol)* et TADAP *(Task Assignement and Data Advertisement Protocol)*.

- **Couche transport :** Elle vérifie le bon acheminement des données et la qualité de la transmission. Dans les RCSF, la fiabilité de transmission n'est pas majeure. Ainsi, les erreurs et les pertes sont tolérées. Par conséquent, un protocole de transport proche du protocole UDP et appelé UDP-Like *(User Datagram Protocol Like)* est utilisé. Cependant, comme le protocole de transport universel est TCP *(Transmission Control Protocol*), les RCSF doivent donc posséder, lors d'une communication avec un réseau externe, une interface TCP-spliting pour vérifier la compatibilité entre ces deux réseaux communicants.

- **Couche réseau :** Elle s'occupe du routage de données fournies par la couche transport. Elle établit les routes entre les nœuds capteurs et le nœud puits et sélectionne le

meilleur chemin en termes d'énergie, délai de transmission, débit, etc. Les protocoles de routage conçus pour les RCSF sont différents de ceux conçus pour les réseaux Ad Hoc puisque les RCSF sont différents selon plusieurs critères comme :

- ❖ l'absence d'adressage fixe des nœuds tout en utilisant un adressage basé-attribut.
- ❖ l'établissement des communications multi-sauts.
- ❖ l'établissement des routes liant plusieurs sources en une seule destination pour agréger (fusionner) des données similaires, etc.

Parmi ces protocoles, nous citons : LEACH *(Low-Energy Adaptive Clustering Hierarchy)*et SAR (*Sequential Assignement Routing*).

- **Couche liaison de données :** Elle est responsable de l'accès au media physique et la détection et la correction d'erreurs intervenues sur la couche physique. De plus, elle établit une communication saut-par-saut entre les nœuds. C'est-à-dire, elle détermine les liens de communication entre eux dans une distance d'un seul saut.

Parmi les protocoles de liaison de données, nous citons: SMACS *(Self-organizing Medium Access Control for Sensor networks)* et EAR *(Eavesdrop And Register).*

- **Couche physique :** Elle permet de moduler les données et les acheminer dans le media physique tout en choisissant les bonnes fréquences.

II.1.3.2 Plans de gestion

Les plans de gestion d'énergie, de mobilité et de tâche contrôlent l'énergie, le mouvement et la distribution de tâches au sein d'un nœud capteur. Ces plans aident les nœuds capteurs à coordonner la tâche de captage et minimiser la consommation d'énergie. Ils sont donc nécessaires pour que les nœuds capteurs puissent collaborer ensemble, acheminer les données dans un réseau mobile et partager les ressources entre eux en utilisant efficacement l'énergie disponible. Ainsi, le réseau peut prolonger sa durée de vie.

- ***Plan de gestion d'énergie*** : Contrôle l'utilisation de la batterie. Par exemple, après la réception d'un message, le capteur éteint son récepteur afin d'éviter la duplication des messages déjà reçus. En outre, si le niveau d'énergie devient bas, le nœud diffuse à ses voisins une alerte les informant qu'il ne peut pas participer au routage. L'énergie restante est réservée au captage ;

- ***Plan de gestion de mobilité*** : Détecte et enregistre le mouvement du nœud capteur. Ainsi, le nœud peut garder trace de ses nœuds voisins. En déterminant leurs voisins, les nœuds capteurs peuvent balancer l'utilisation de leur énergie et la réalisation de tâche ;

- ***Plan de gestion de tâche*** : Ordonnance les différentes tâches de captage de données dans une région spécifique. Il n'est pas nécessaire que tous les nœuds de cette région effectuent la tâche de captage au même temps ; certains nœuds exécutent cette tâche plus que d'autres selon leur niveau de batterie.

II-2 ZigBee/IEEE 802.15.4 [24]

ZigBee est une technologie de réseau sans fil personnel (WPAN) destinée à l'´électronique embarqué à très faible consommation énergétique. Elle propose une pile protocolaire

propriétaire et légère, déclinable dans plusieurs versions plus ou moins complètes, pour des applications de transferts de données à faibles débits et de faibles taux d'utilisation du médium.

ZigBee s'appuie sur la norme IEEE 802.15.4 **[7] [8]** (que nous désignerons par la suite par « 802.15.4 ») pour les couches physique et liaison, qui sont les couches 1 et 2 du modèle OSI. *ZigBee* propose ensuite ses propres couches supérieures (réseau, etc.) qui doivent faire l'objet d'une demande auprès de la *ZigBee Alliance* pour être utilisées. A ce titre, des droits sont perçus par la *ZigBee Alliance* pour tout emploi d'une pile *ZigBee* dans le cadre d'une application industrielle.

II.2.1 Objectifs et domaine d'application

ZigBee est un réseau sans fil à courte portée qui utilise les ondes hertziennes pour transporter des messages entre deux ou plusieurs entités réseaux.

Il est caractérisé par une portée comprise entre quelques mètres et quelques centaines de mètres et un débit faible (max. 250kbits/s). La différence entre *ZigBee*et et la plupart des autres WPAN existants se situe au niveau de l'utilisation du médium hertzien est que ZigBee est optimisé pour une faible utilisation du médium partagé par tous.

Typiquement, un module *ZigBee* occupera le médium pendant quelques millisecondes en émission, attendra éventuellement une réponse ou un acquittement, puis se mettra en veille pendant une longue période avant l'´émission suivante, qui aura lieu à un instant prédéterminé.

ZigBee est conçu pour interconnecter des unités embarquées autonomes comme des capteurs, à des unités de contrôle ou de commande. De telles entités embarquées pouvant dès lors être alimentées pendant plusieurs mois par des piles standards.

ZigBee est utilisé dans des applications types :

- *Home automation* : Chauffage, ventilation, air conditionné, sécurité, éclairage ;
- *Industrielles* : Contrôle de machines, etc. ;
- *Auto motive* : Contrôle de la pression des pneus, etc.;
- *Agriculture*: Mesure de l'humidité du sol, mesure de la salinité du sol, etc. ;
- *Autres* : Contrôle d'équipements électroniques, communication entre PC et périphériques, etc.

II.2.2 Caractéristiques de ZigBee

Les objectifs visés par ZigBee peuvent être résumés dans les points suivants :

- Usage sans restrictions géographiques
- Pénétration à travers les murs et plafonds
- Coût avantageux
- Débit : 10kbps-115.2kbps
- Portée radio: 10-75m
- Jusqu'à 2 ans de durée de vie de batterie standards ;

Positionnement :

La figure suivante illustre le positionnement de ZigBee par rapport à d'autres standards :

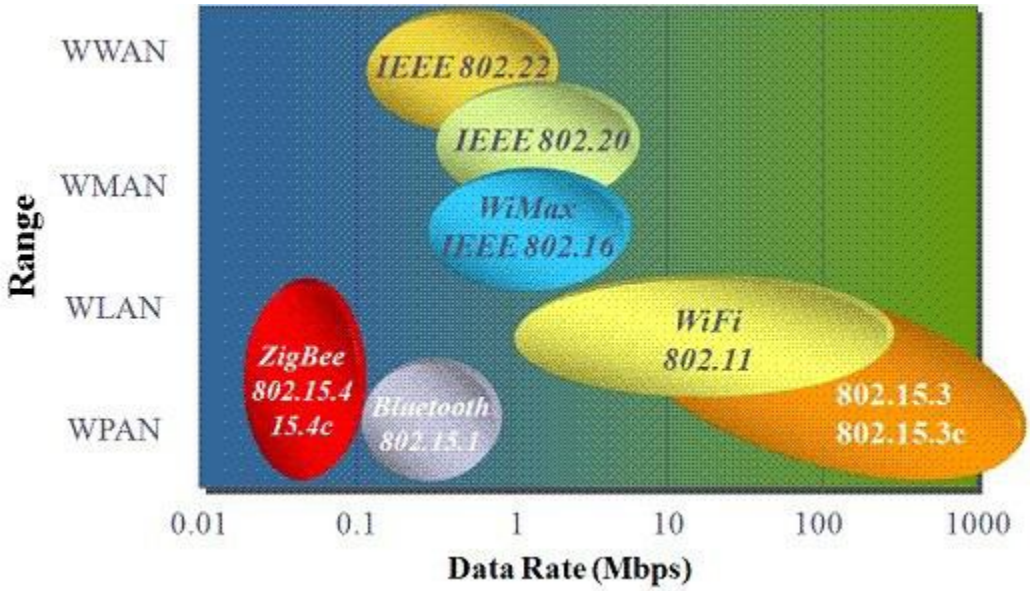


Figure I.6: Positionnement du ZigBee .

II.2.3 L'architecture ZigBee /IEEE 802.15.4 [24]

L'architecture de la pile *Zigbee* est basée sur le modèle OSI à sept 7 couches. Chacune de ses couches rendant un ensemble de services spécifiques à la couche supérieure au travers d'interfaces appelées "Services Acces Point" (SAP).

Elle se base sur la norme IEEE 802.15.4 pour définir les deux couches inférieures (PHY et MAC) en tant que fondations et vient rajouter la couche "network" (NWK) et le cadre pour les couches applicatives qui comprend la sous couche de support applicatif (Application Support Sub-layer), les objets systèmes *Zigbee* (Zigbee Device Objects) et les systèmes définis par les fabricants comme l' illustre la figure suivante I.7(plus de détails se porter vers l'annexe D).

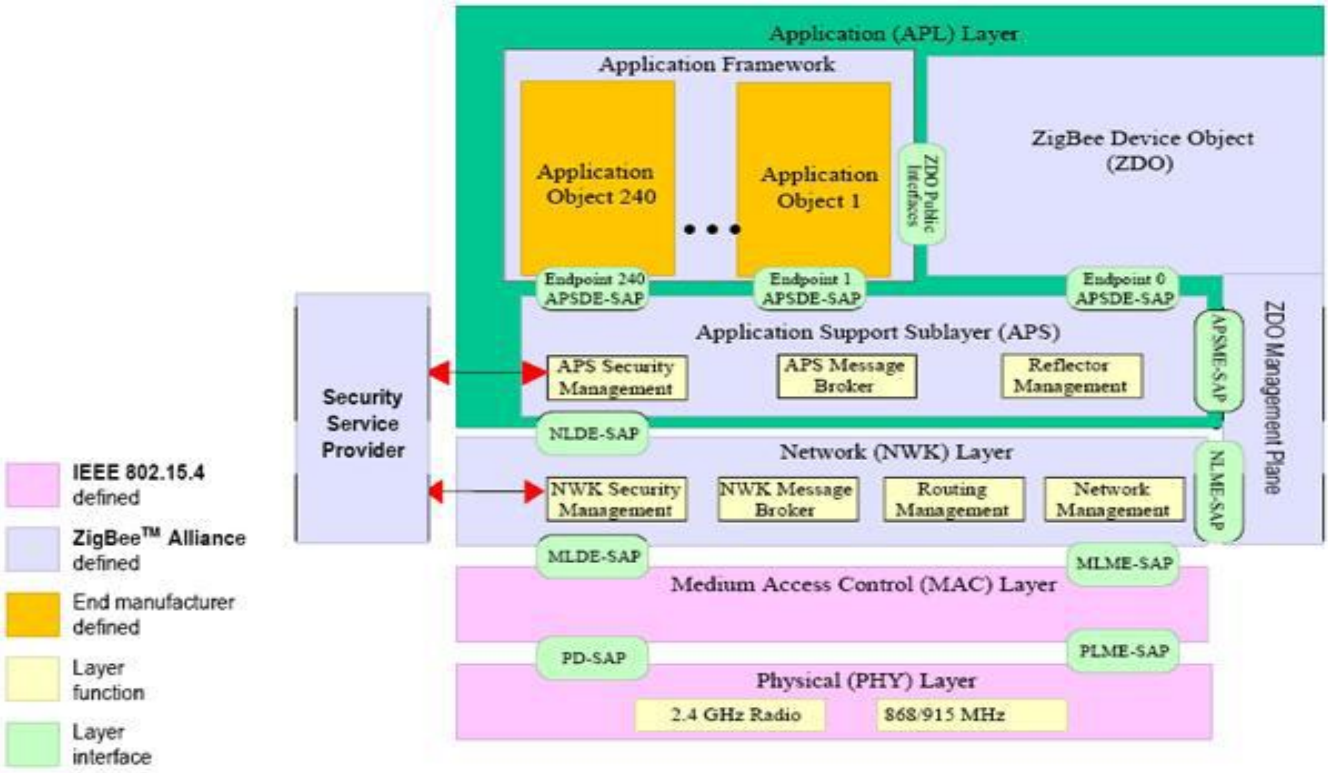


Figure I.7 :L'architecture ZigBee /IEEE 802.15.4[20].

II.2.3.1 Rôle des différentes couches :

- **Les sous couches MAC IEEE 802.15.4 et PHY :**

La couche PHY a été conçue pour de hauts besoins d'intégration à faibles coûts. La couche MAC (Media Access control) a été conçue pour intégrer de multiples topologies sans complexité. Son rôle est de :

- ❖ Contrôler les accès au canal radio via un mécanisme CSMA-CA.
- ❖ Transmettre les "beacons frames",
- ❖ Synchronisation réseau,
- ❖ Fournir un système de transmission fiable.

- **La couche Network NWK** :

La couche NWK a été conçue pour permettre au réseau de s'étendre avec des émetteurs à basse consommation d'énergie et pour gérer un grand nombre de nœuds avec un temps de latence très faible.

- ❖ Mécanismes pour joindre, quitter et former un nouveau réseau.
- ❖ Configuration et adressage des nouveaux nœuds.
- ❖ Découverte et maintenance des routes entre les dispositifs.
- ❖ gestion des types de services applicatifs.

- **La couche Application (Application Support Sublayer)**

  La couche applicative Zigbee est constituée des couches : *Application Support Sublayer* (APS), Zigbee Device Object (ZDO) et de l'Application Framework définie par les fabricants.

  - ❖ La couche APS a pour responsabilité de maintenir les tables de routage entre les systèmes se rendant les mêmes services et de faire suivre les messages entre les nœuds mais aussi de déterminer quels sont les autres objets qui fonctionnent dans le même espace.
  - ❖ La couche ZDO détermine le rôle du système dans le réseau (ex : ZC, ZR ou ZED) mais aussi d'initier ou de répondre aux demandes de "bind" (insertion d'un autre module zigbee dans la table de routage).

II.2.3.2 Aperçus de MAC IEEE 802.15.4 [2]

La couche MAC IEEE 802.15.4 :

- Utilise deux modes d'adressage IEEE 64-bit & 16-bit ;
- Accès canal CSMA-CA ;
- Utilise une structure de trame simple ;
- Permet d'utiliser le mécanisme de beaconing; réveil périodique, vérification de l'arrivé d'un beacon ;
- Economise l'énergie à travers la mise en veille entre deux beacons, et les nœuds ne devant pas router ou recevoir les données aléatoirement peuvent se mettre en veille ;
- Assure une transmission fiable de données.

Network Beacon

- Identifie le réseau ;
- Décrit la structure de la super-trame ;
- Indique la présence de données ;
- Présent uniquement lorsque le réseau est actif ;
- Il est optionnel.

II.2.4 Mécanisme d'accès au canal

IEEE 802.15.4 utilise CSMA/CA décliné en deux versions selon la configuration du réseau:

- Si le "beaconing" n'est pas utilisé, IEEE 802.15.4 utilise CSMA/CA sans slots
- Si le "beaconing" est utilisé, IEEE 802.15.4 utilise CSMA/CA avec slots et la structure super-trame

- Dans un réseau local Ethernet classique, la méthode d'accès utilisée par les machines est le **CSMA/CD** (*Carrier Sense Multiple Access with Collision Detection*), pour lequel chaque machine est libre de communiquer à n'importe quel moment. Chaque machine envoyant un message vérifie qu'aucun autre message n'a été envoyé en même temps par une autre machine. Si c'est le cas, les deux machines patientent pendant un temps aléatoire avant de recommencer à émettre.

- Dans un environnement sans fil ce procédé n'est pas possible dans la mesure où deux stations communiquant avec un récepteur ne s'entendent pas forcément mutuellement

en raison de leur rayon de portée. Ainsi la norme 802.11 propose un protocole similaire appelé **CSMA/CA** (*Carrier Sense Multiple Access with Collision Avoidance*).

- Le protocole CSMA/CA utilise un mécanisme d'esquive de collision basé sur un principe d'accusé de réception réciproque entre l'émetteur et le récepteur :

  La station voulant émettre écoute le réseau. Si le réseau est encombré, la transmission est différée. Dans le cas contraire, si le média est libre pendant un temps donné (appelé **DIFS** pour *Distributed Inter Frame Space*), alors la station peut émettre. La station transmet un message appelé Ready To Send (ou *Request To Send*, noté RTS signifiant prêt à émettre) contenant des informations sur le volume des données qu'elle souhaite émettre et sa vitesse de transmission. Le récepteur (généralement un point d'accès) répond un *Clear To Send* (**CTS**, signifiant *Le champ est libre pour émettre*), puis la station commence l'émission des données.

- À la réception de toutes les données émises par la station, le récepteur envoie un accusé de réception (**ACK**). Toutes les stations avoisinantes patientent alors pendant un temps qu'elles considèrent être celui nécessaire à la transmission du volume d'information à émettre à la vitesse annoncée.

II.3 Caractéristiques des réseaux de capteurs

- **Energie limitée**

  Les RCSF visent la consommation d'énergie puisque l'alimentation de chaque nœud est assurée par une source d'énergie limitée et généralement irremplaçable à cause de l'environnement hostile où il est déployé.
  De ce fait, la durée de vie d'un RCSF dépend fortement de la conservation d'énergie au niveau de chaque nœud.

- **Modèle de communication**

  Les nœuds dans les RCSF communiquent selon un paradigme plusieurs-à-un (*many to one*). En effets, les nœuds capteurs collectent des informations à partir de leur environnement et les envoient toutes vers un seul nœud qui représente le centre de traitement.

- **Densité de déploiement**

  La densité est plus élevée dans les RCSF que dans les réseaux AdHoc. Le nombre de nœuds capteurs peut atteindre des millions de nœuds pour permettre une meilleure granularité de surveillance. De plus, si plusieurs nœuds capteurs se retrouvent dans une région, un nœud défaillant pourra être remplacé par un autre.

Cependant, la densité de déploiement donne naissance à des challenges pour la communication entre les nœuds. En effet, elle peut provoquer des collisions des paquets transmis.

- **Absence d'adressage fixe des nœuds [21]**

Les nœuds dans les réseaux sans fil classiques sont identifiés par des adresses IP. Cependant, cette notion n'existe pas dans les RCSF.
Ces derniers utilisent un adressage basé sur l'attribut du phénomène capté, on parle donc de l'adressage basé-attribut.
En effet, les requêtes des utilisateurs ne sont pas généralement destinées à un seul nœud, mais plutôt, à un ensemble de nœuds identifiés par un attribut. Par exemple, identifier un ensemble de nœuds par « les nœuds qui captent le volume du CO2 dépassant 0,0375 % dans l'atmosphère».

- **Limitations de ressources physiques**

Habituellement les nœuds capteurs ont une taille très petite, ce facteur de forme limite la quantité de ressources qui peuvent être mises dans ces nœuds.

Par conséquent, la capacité de traitement et de mémoire devient très limitée. Les nœuds capteurs collaborent en traitant partiellement les mesures captées et envoient seulement les résultats à l'utilisateur.

Une autre conséquence, est que ces limitations imposent des portées de transmission réduites contraignant les informations à être relayées de nœud en nœud avant d'atteindre le destinataire. C'est la raison pour laquelle les RCSF adoptent des communications multi-sauts

- **Sécurité**

Les contraintes et limitations physiques font que l'absence d'une sécurité physique dans l'environnement hostile où ils sont déployés expose les nœuds à un danger qui tend vers la falsification de l'information. En effet, les nœuds capteurs eux-mêmes sont des points de vulnérabilité du réseau car ils peuvent être modifiés, remplacés ou supprimés.

De plus, les réseaux de capteurs sans fils sont plus sensibles aux attaques qui menacent les données transmises en raison de l'absence d'infrastructure.

II.4 Contraintes de conception des RCSF

Les principaux facteurs et contraintes influençant l'architecture des réseaux decapteurs peuvent être résumés comme suit:

- *La tolérance de fautes* : Certain nœuds capteurs peuvent générer des erreurs ou ne plus fonctionner à cause d'un manque d'énergie, un problème physique ou une interférence. Ces problèmes n'affectent pas le reste du réseau, c'est le principe de la tolérance de fautes. La tolérance de fautes est la capacité de maintenir les fonctionnalités du réseau sans interruptions dues à une erreur intervenue sur un ou plusieurs capteurs.

- *L'échelle* : Le nombre de nœuds déployés pour un projet peut atteindre le million. Un nombre aussi important de nœuds engendre beaucoup de transmissions inter nodales et nécessite que le puits "sink " soit équipé de beaucoup de mémoire pour stocker les informations reçues.

- *Les coûts de production* : Souvent, les réseaux de capteurs sont composés d'un très grand nombre de nœuds. Le prix d'un nœud est critique afin de pouvoir concurrencer un réseau de surveillance traditionnel. Actuellement un nœud capteur ne coûte souvent pas beaucoup.

- *L'environnement* : Les capteurs sont souvent déployés en masse dans des endroits tels que des champs de bataille au-delà des lignes ennemies, à l'intérieur de grandes machines, au fond d'un océan, dans des champs biologiquement ou chimiquement souillés,... Par conséquent, ils doivent pouvoir fonctionner sans surveillance dans des régions géographiques éloignées.

- *La topologie de réseau* : Le déploiement d'un grand nombre de nœuds nécessite une maintenance de la topologie. Cette maintenance consiste en trois phases : Déploiement, Post-déploiement (les capteurs peuvent bouger, ne plus fonctionner,...), Redéploiement de nœuds additionnels **[11].**

- *Les contraintes matérielles* : La principale contrainte matérielle est la taille du capteur. Les autres contraintes sont que la consommation d'énergie doit être moindre pour que le réseau survive le plus longtemps possible, qu'il s'adapte aux différents environnements (fortes chaleurs, eau,..), qu'il soit autonome et très résistant vu qu'il est souvent déployé dans des environnements hostiles.

- *Les médias de transmission* : Dans un réseau de capteurs, les nœuds sont reliés par une architecture sans-fil. Pour permettre des opérations sur ces réseaux dans le monde entier, le média de transmission doit être normé. On utilise le plus souvent l'infrarouge (qui est license-free, robuste aux interférences, et peu onéreux), le bluetooth et les communications radio ZigBee.

- *La consommation d'énergie* : Un capteur, de par sa taille, est limité en énergie (< 1.2V). Dans la plupart des cas le remplacement de la batterie est impossible. Ce qui veut dire que la durée de vie d'un capteur dépend grandement de la durée de vie de la batterie. Dans un réseau de capteurs (multi-sauts) chaque nœud collecte des données et envoie/transmet des valeurs. Le dysfonctionnement de quelques nœuds nécessite un

changement de la topologie du réseau et un re-routage des paquets. Toutes ces opérations sont gourmandes en énergie, c'est pour cette raison que les recherches actuelles se concentrent principalement sur les moyens de réduire cette consommation.

II.5 Applications des RCSF

Les RCSF peuvent avoir beaucoup d'applications. Parmi elles, nous citons :

- *Découvertes de catastrophes naturelles* : On peut créer un réseau autonome en dispersant les nœuds dans la nature. Des capteurs peuvent ainsi signaler des événements tels que feux de forêts, tempêtes ou inondations. Ceci permet une intervention beaucoup plus rapide et efficace des secours.

- *Détection d'intrusions* : En plaçant, à différents points stratégiques, des capteurs, on peut ainsi prévenir des cambriolages ou des passages de gibier sur une voie de chemin de fer (par exemple) sans avoir à recourir à de coûteux dispositifs de surveillance vidéo.

- *Applications métier* : On pourrait imaginer devoir stocker des données nécessitant un certain taux d'humidité et une certaine température (min ou max). Dans ces applications, le réseau doit pouvoir collecter ces différentes informations et alerter en temps réel si les seuils critiques sont dépassés.

- *Contrôle de la pollution* : On pourrait disperser des capteurs au-dessus d'un emplacement industriel pour détecter et contrôler des fuites de gaz ou de produits chimiques. Ces applications permettraient de donner l'alerte en un temps record et de pouvoir suivre l'évolution de la catastrophe.

- *Agriculture* : Des nœuds peuvent être incorporés dans la terre. On peut ensuite questionner le réseau de capteurs sur l'état du champ (déterminer par exemple les secteurs les plus secs afin de les arroser en priorité). On peut aussi imaginer équiper des troupeaux de bétail de capteurs pour connaître en tout temps, leur position ce qui éviterait aux éleveurs d'avoir recours à des chiens de berger.

- *Surveillance médicale* : En implantant sous la peau de mini capteurs vidéo, on peut recevoir des images en temps réel d'une partie du corps sans aucune chirurgie pendant environ 24h. On peut ainsi surveiller la progression d'une maladie ou la reconstruction d'un muscle.

- *Contrôle d'édifices* : On peut inclure sur les parois des barrages des capteurs qui permettent de calculer en temps réel la pression exercée. Il est donc possible de réguler le niveau d'eau si les limites sont atteintes. On peut aussi imaginer inclure des capteurs entre les sacs de sables formant une digue de fortune. La détection rapide d'infiltration d'eau peut servir à renforcer le barrage en conséquence. Cette technique peut aussi être utilisée pour d'autres constructions tels que ponts, voies de chemins de fer, routes de montagnes, bâtiments et autres ouvrages d'art.

II.6 Projets d'applications en cours [21]

Après avoir exposé un certain nombre d'applications potentielles envisagées pour les RCSF, nous décrivons dans ce qui suit des projets d'applications qui sont en cours de réalisation.

II.6.1Newtrax pour les applications militaires

Afin de répondre aux exigences des opérations militaires, les forces canadiennes choisissent la compagnie privée Newtrax (qui a comme vision de fournir des solutions de mesure, contrôle, messagerie et localisation) comme partenaire industriel pour un contrat évalué à 1,5 milliards de dollars pour fournir sa technologie des RCSF maillés. Cette architecture offre une résilience et extensibilité supérieure. De plus, chaque nœud peut être ajusté pour fonctionner selon différentes fréquences entre 100 MHz et 1 GHz.

Figure I.8 : Un service militaire utilisant un réseau RCSF.

II.6.2 Glacsweb pour les applications environnementales :

Glacsweb est un projet réalisé par l'université de Southampton en Grande-Bretagne. Il vise à comprendre les changements climatiques et leurs effets sur le niveau de la mer, étude des glaciers. Ces différentes applications permettent d'ores et déjà d'étudier et d'apporter des solutions aux différents problèmes liés à la gestion de réseaux de capteurs de grande envergure.

Figure I.9: Exemple d'utilisation de Glacsweb.

II.6.3 Projet en cours dans le domaine médicale

Actuellement, des micro-caméras qui peuvent être avalées existent. Elles sont capables, sans avoir recours à la chirurgie, de transmettre des images de l'intérieur d'un corps humain avec une autonomie de 24 heures. Le projet actuel est de créer une rétine artificielle composée de 100 micro-capteurs pour corriger la vue. Un émetteur-récepteur transmet les données à une station de base branchée à un ordinateur personnel, à un poste des soins infirmiers ou à un assistant numérique.

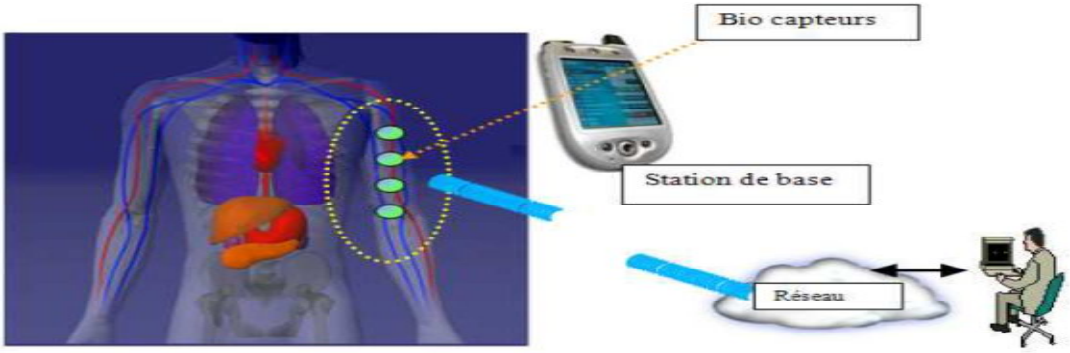


Figure I.10 : Application des réseaux de capteurs en médecine.

III Différences entre les réseaux de capteurs et les réseaux Ad hoc classiques

Dans les réseaux Ad hoc traditionnels, les tâches qui traitent le routage, et la gestionde mobilité visent l'optimisation des différents paramètres de qualité de service (QoS) tel que l'efficacité dans le débit et les délais de transmission sous la contrainte de mobilité. La consommation d'énergie est d'une importance secondaire, puisque les batteries des unités mobiles utilisées peuvent être facilement remplacées.

Cependant, les réseaux de capteurs englobent un grand nombre de nœuds possédant des sources d'énergie irremplaçable à cause de leur utilisation distante non-assistée dans les environnements hostiles. Ces capteurs communiquent entre eux avec un taux de transmission très faible de l'ordre de 1 à 100kbps. Pour cela, et contrairement aux réseaux ad hoc classiques, le but principal des techniques utilisées est de prolonger la durée de vie des batteries afin de prévenir les dégradations de connectivité dans le réseau.

Les communications dans les réseaux de capteurs sont, dans la plus part des temps, unidirectionnels à partir des capteurs vers le nœud puits.
Donc, malgré les réseaux de capteurs sans fil sont apparentées aux réseaux ad hoc, les spécificités, les objectifs et les besoins de ces types de réseaux diffèrent.

Le tableau comparatif I.1 suivant résume les différences entre les réseaux de capteurs et les réseaux ad hoc ordinaires **[23][24]**.

	Capteurs	Ad hoc
Flot de communication	"many to one"	"any to any"
Contrainte clé	Energie	Débit
Communication	Broadcast	Point à point
Relation entre les nœuds	Collaboration	Chaque noeud à son objectif
Identificateur	Très grand nombre de nœuds sans ID	Présence de la notion d'ID.
Routage	*Data-centric:* souvent pas d'adresses uniques, les requêtes sont envoyées à tous les nœuds.	*Adress-centric:* une adresse unique pour chaque nœud utilisé pour réaliser la communication entre les nœuds.
Nombre de nœuds	Grand nombre de nœuds (de l'ordre de mille).	Nombre de nœud moyen (de l'ordre de cents).
Agrégation	Les nœuds agrègent les données avant de les transmettre	Généralement pas d'agrégation de données.
Capacité	Des petits nœuds plus susceptibles aux pannes, avec moins capacité de traitements et de stockage.	Des nœuds ayant plus de capacité de traitements et de stockage.

Protocole de routage	Les protocoles de routage doivent prendre en considération certaines contraintes: le grand nombre de nœuds, moins de ressources mémoire et de calcule, la consommation d'énergie.	Plusieurs types de protocoles de routage peuvent être utilisés.
Mobilité	La mobilité des nœuds est restreinte.	Mobilité des nœuds.

TABLEAU I.1 Comparaison entre les WSN et les réseaux Ad Hoc

Conclusion

Dans ce chapitre nous avons décrit les RCSF qui sont apparentés aux réseaux Ad Hoc. Ces réseaux connaissent un grand essor grâce à la multitude d'applications qu'ils offrent ainsi que leurs caractéristiques inhérentes telles que leur déploiement aléatoire et leur faible coût, leur grande mobilité et ceci grâce aux récents développements concernant la miniaturisation des composants électroniques (construction des capteurs de quelques millimètres cubes de volume).

Dans le domaine de la recherche, les RCSF posent un certain nombre de challenges au niveau de la taille des capteurs. Autrement dit, on recherche une miniaturisation maximale et des performances optimales quant à la transmission, le débit et la consommation d'énergie qui est un facteur à prendre en considération dans la conception des RCSF. En effet, la plus grande partie de cette ressource est utilisée pendant la communication.
Toutes ces recherches vont dans le même sens: améliorer au maximum les performances d'un RCSF. En revanche chaque tentative d'amélioration d'un critère pose des problèmes majeurs sur un autre. Par exemple la miniaturisation de la batterie pose le problème d'une durée de vie plus courte.

Ces différentes caractéristiques des capteurs sont à prendre en considération dans la communication des RCSF. En effet, la propagation et l'acheminement de données dans un RCSF représentent une fonctionnalité très importante. Ils doivent prendre en considération toutes les caractéristiques du réseau afin d'assurer les meilleures performances du système.

C'est dans cet objectif que le chapitre suivant serra consacré au routage dans les réseaux de capteurs sans fils, en s'appuyant particulièrement sur le l'étude du protocole de routage LEACH.

CHAPITRE II :

LE ROUTAGE DANS LES RESEAUX DE CAPTEURS SANS FILS (WSN)

Chapitre II : le routage dans les réseaux de capteurs sans fils

INTRODUCTION

Le routage dans les réseaux de capteurs sans fils (RCSF),consiste à déterminer une route, une transmission fiable des données captées des nœuds capteurs vers le puits "sink", cette méthode d'acheminement doit être optimale minimisant la consommation d'énergie des capteurs et augmentant ainsi la durée de vie du réseau.

Les différents algorithmes ou protocoles, conçus pour les réseaux Ad Hoc peuvent être exploité dans les réseaux de capteurs sans fils. Mais leur principal inconvénient c'est qu'ils ne prennent pas en compte la consommation d'énergie. Alors que pour les réseaux de capteur sans fils(WSN), l'énergie est très critique puisqu'un nœud capteur est alimenté par une batterie à énergie limitée.

Mais aussi, les applications des RCSF nécessitent en général un déploiement dense des nœuds qui exigent une gestion soigneuse des ressources et exigent également que l'écoulement des données mesurées de sources multiples soit à un puits particulier,d'où lanécessité de les améliorer ou de développer de nouveaux protocoles de routage spécifiques aux *RCS*F qui assurent une consommation minimale d'énergie tout en maintenant le bon fonctionnement du réseau et sans dégrader ses performances.

Dans le cadre de ce chapitre, nous allons citer en premier lieu,les métriques permettant de tester l'efficacité d'un protocole de routage.

En second lieu, nous allons décrire les critères servant à la classification des protocoles de routage .Pour par la suite, traiter des exemples de protocole de routage que ce soit dans les réseaux Ad-hoc et dans les réseaux de capteurs sans fil RCSF, nous nous intéresserons plus particulièrement en fin de chapitre à l'étude du fonctionnement du protocole LEACH .

I Métriques de routage[4]

Dans cette section, nous étudions les métriques communes utilisées pour mesurer l'efficacité des protocoles de routage.

Les protocoles de routage permettent aux nœuds de comparer les métriques calculées (à travers un algorithme) afin de déterminer les routes optimales à emprunter. Plus la métrique est optimale, plus le protocole de routage considère que la probabilité d'atteindre le nœud puits à travers ce nœud intermédiaire est grande.

Plusieurs métriques peuvent affecter le routage en termes d'énergie, délai, longueur du chemin, etc. De plus, elles peuvent être considérées seules ou combinées (hybrides)**.**

Chapitre II : le routage dans les réseaux de capteurs sans fils

I.1 Métriques pour la consommation énergétique

Les protocoles de routage utilisent cet ensemble de métriques pour minimiser la consommation d'énergie pendant le routage**.**

L'idée est de calculer l'énergie disponible (ED) pour chaque nœud du réseau et l'énergie nécessaire (EN) pour les transmissions des paquets entre une paire de nœuds. Les routes entre les nœuds et le puits sont établies et chacune d'elles est caractérisée par la somme des ED des nœuds qui la constituent et par la somme des EN des liaisons qui la construisent.

La consommation d'énergie suit plusieurs approches dont on peut citer :

I.1.1Par considération de puissance

La route choisie est celle caractérisée par la somme des ED la plus élevée.

I.1.2 Par considération du coût

La route choisie est celle caractérisée par la plus petite somme des EN.

I.1.3 Par considération de puissance et du coût

Cette métrique est la combinaison des deux métriques précédentes. La route choisie est celle caractérisée par la plus petite somme des EN et la plus grande somme des ED.

I.2 Nombre de sauts

Les protocoles de routage utilisent cette métrique pour minimiser le nombre de sauts pendant le routage. L'idée est de calculer le nombre de nœuds intermédiaires pouvant être traversés lors d'une transmission d'un paquet du nœud source vers le nœud puits. La route choisie est celle qui contient un nombre minimum de nœuds (minimum de sauts)**.**

I.3 Perte de paquets

Les protocoles de routage utilisent cette métrique dans le but de minimiser le nombre de paquets de données perdus lors du transfert depuis une source vers une destination pendant le routage. L'idée est de calculer *le ratio* des paquets perdus et des paquets émis transitant dans le réseau. *Autrement dit, on calcule le nombre de paquets perdus sur le nombre de paquets transmis lors d'une transmission.*

I.4. Délai de bout-en-bout EED

L'EED (*End-to-End Delay*) est le temps moyen nécessaire pour qu'un paquet de données soit acheminé à partir de la source vers la destination.

Cette technique est parmi les métriques les plus connues dans les réseaux sans fil. Les protocoles de routage l'utilisent pour minimiser le temps de propagation des paquets de données échangés pendant le routage.

IIClassification des protocoles de routage [2]

Les méthodes employées pour la classification des protocoles de routages peuvent être distinguées suivant plusieurs critères comme illustré sur la figure suivante :

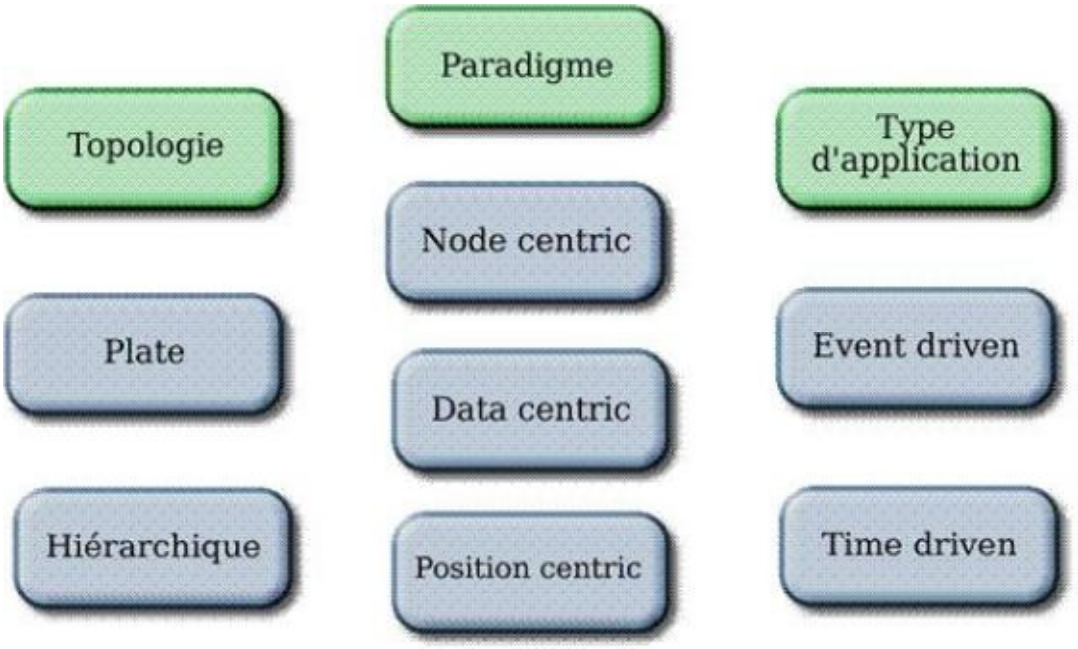


Figure II.1:Classification des protocoles de routages

II.1 Selon latopologie des réseaux decapteurs sans fils

Les topologies des réseaux de capteur sont déterminées à partir des protocoles de routage utilisés pour l'acheminement des données entre les nœuds et le Sink.
Ces protocoles peuvent être hiérarchiques, plat (Flat) ou basé localisation.

II.1.1 Topologie Hiérarchique

Les protocoles à topologie hiérarchique forment des réseaux dans lesquels un nœud central *Sink* (le niveau supérieur de la hiérarchie) est relié à un ou plusieurs autres nœuds qui appartiennent à un niveau plus bas dans la hiérarchie (deuxième niveau) avec une liaison point à point.

Chacun des nœuds du deuxième niveau aura également un ou plusieurs autres nœuds de niveau plus bas dans la hiérarchie (troisième niveau) reliées à lui avec une liaison point à point (comme l'illustre la figure II.)

Chaque ensemble de nœuds forme une sorte de motif (Cluster). Le nœud central n'a aucun autre nœud au-dessus de lui dans la hiérarchie sauf le centre de traitement des données ou la passerelle si elle existe. Les nœuds du deuxième niveau jouent le rôle des passerelles entre ceux du troisième niveau et le Sink. Dans ce cas, le routage devient plus simple, puisqu'il s'agit de passer par les passerelles pour atteindre le nœud destination.

Un réseau basé sur une topologie hiérarchique doit avoir au moins trois niveaux dans sa hiérarchie, puisqu'un réseau avec un nœud central *Sink* et seulement un niveau hiérarchique au-dessous, forme une topologie en étoile.

À titre d'exemple des protocoles utilisant une topologie hiérarchique on peut citer le protocole LEACH *(Low-energy Adaptive Clustering Hierarchy)*, CBRP (*Cluster Based Routing Protocol*),etc.

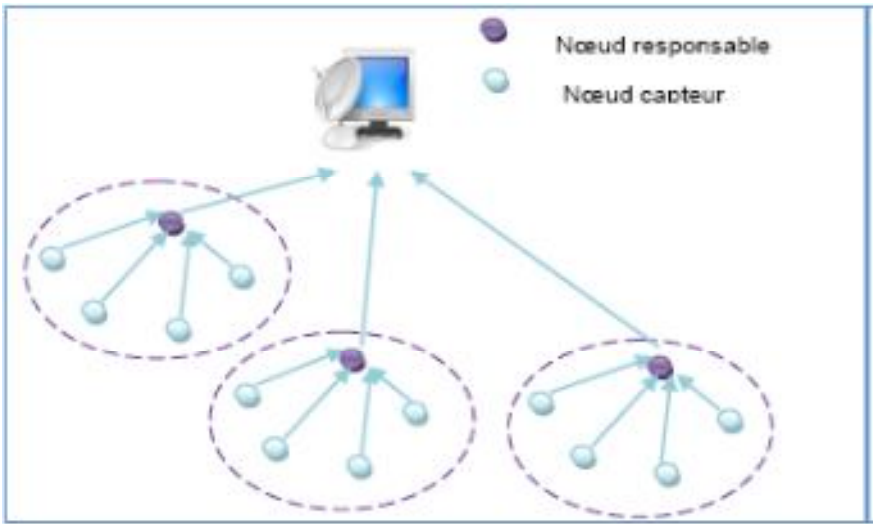


Figure II.2 : Topologie hiérarchique

II.1.2 Topologie plate (Flat)

Les protocoles à topologie plate (*flat*) considèrent que tous les nœuds sont égaux, ont les mêmes fonctions, et peuvent communiquer entre eux sans devoir passer par un nœud particulier ou une passerelle. Seul un nœud particulier, le *Sink*, est chargé de la collecte des données issues des différents nœuds capteurs afin de les transmettre vers les centres de traitement.

En cas où la destination ne fait pas partie du voisinage de la source, les données seront transmises en utilisant les sauts multiples à travers les nœuds intermédiaires comme c'est illustré dans la figure II.2.
Ce type de réseau représente l'avantage de l'existence de différents chemins d'une source vers une destination et c'est pour remédier au problème de changement brusque de topologie ou la défaillance d'un nœud intermédiaire.

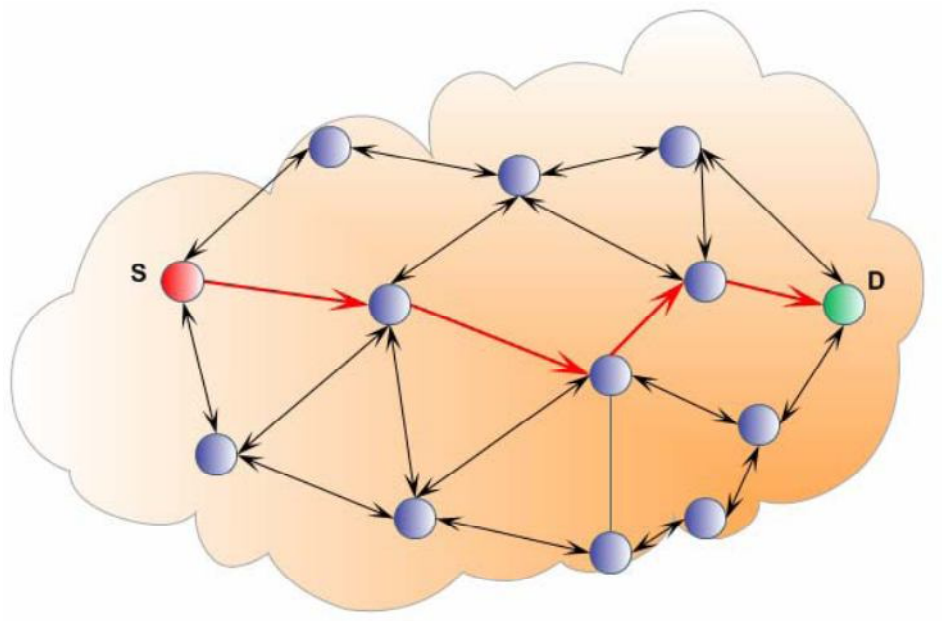


Figure II. 3 : Topologie plate (Flat)

À titre d'exemple des protocoles utilisant une topologie plate on peut citer le protocole Direct Diffusion,SAR (*Sequential Assignment Routing*), etc.

II.1.3 Topologie basée Localisation

Ce type de topologie est mieux adapté aux réseaux avec une forte mobilité. Elle est utilisée dans les applications où il est plus intéressant d'interroger le système en se basant sur la localisation des nœuds et où on peut tirer profit des positions des nœuds pour prendre des décisions qui minimisent le nombre de messages transmis pendant le routage.

Avant d'envoyer ses données à un nœud destination, le nœud source utilise un mécanisme pour déterminer sa localisation. L'information temporaire de localisation appelée LDA (*Location Dependent Address*) qui est un triplé de coordonnées géographiques (longitude, latitude, altitude) obtenues, par exemple, au moyen d'un GPS(*Global Positioning System*), système de localisation global mais ce genre de système reste trop coûteux pour un RCSF.

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

La Figure II.4 représente la topologie basée sur la localisation.

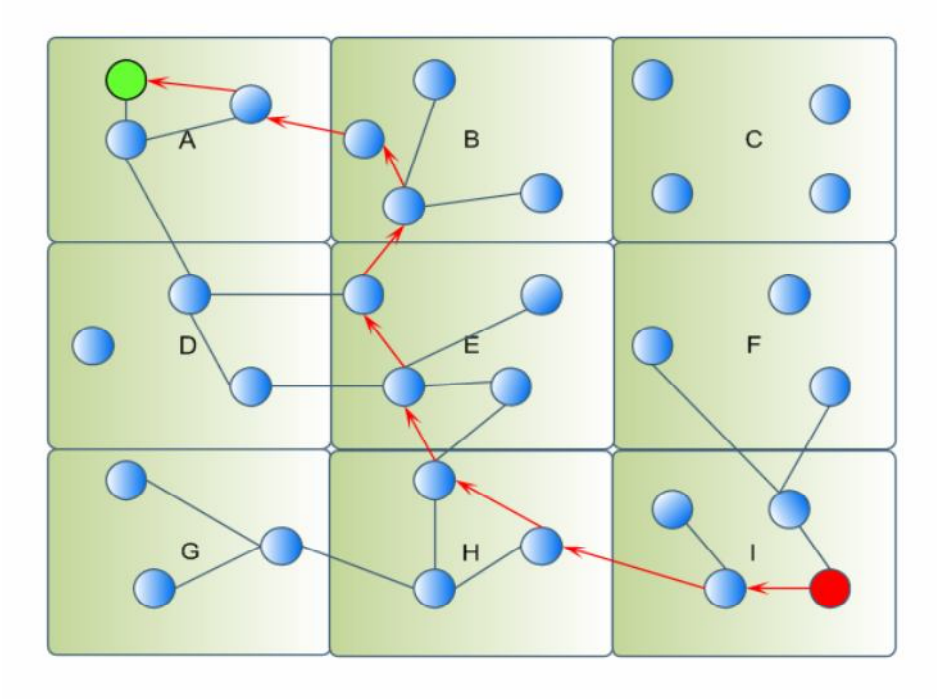


Figure II.4 : Topologie Basée Localisation

II.2 Selon le Paradigme de communication

Dans les RCSF, il existe trois paradigmes de communication :

- ***Node centric*** : ce paradigme est celui employé dans les réseaux conventionnels, où les communications se basent sur *l'identification des nœuds participants*, qui se fait à l'aide d'adresses IP.

- ***Data centric :*** dans un RCSF, *la donnée est plus importante* que le nœud lui-même, ce qui rend son identification inutile. De ce fait dans le paradigme data centric, les communicants sont identifiés par leurs données. Ainsi, le système peut être vu comme une base de données distribuée, où les nœuds forment des tables virtuelles, alimentées par les données captées.

- ***Position centric :*** dans cette approche, *les positions des nœuds* représentent le moyen principal d'adressage et de routage. Dans certaines applications, il est plus intéressant d'interroger le système en utilisant les positions des nœuds, que leurs adresses IP. Dans ce cas, le routage s'effectue grâce à des techniques géométriques afin d'acheminer l'information d'une zone géographique vers une autre.

II.3 Type d'application

La méthode de captage des données dans un RCSF dépend de l'application et de l'importance de la donnée. De ce fait, les RCSF peuvent être catégorisés comme time-driven ou event-driven.

- **Application time-driven** : un réseau time-driven est approprié pour des applications qui nécessitent un prélèvement *périodique des données.* Par exemple, cela est utile dans des applications de monitoring (feu, météo) afin d'établir des rapports périodiques.

- **Application event-driven :** dans des applications temps réel, les capteurs doivent *réagir immédiatement à des changements soudains des valeurs captées.* Un prélèvement périodique des données est inadapté pour ce type de scénarios. Pour cela, le protocole doit être réactif et doit donner des réponses rapides à l'occurrence d'un certain nombre d'évènements.

III. Types de protocoles de routages

Le mode d'établissement des chemins permet de classer les protocoles de routage en trois catégories : proactif, réactif ou hybride.

III.1 Les protocoles proactifs

Les protocoles de routage proactifs établissent au préalable les meilleures routes pour chaque nœud vers toutes les destinations possibles. Ils maintiennent en *temps réel* une vision de l'état du réseau qui soit suffisante pour que tous les sous-ensembles (nœuds) soient connectés à chaque instant (éventuellement en passant par plusieurs liens consécutifs).

Pour cela, chaque nœud envoie périodiquement à tous ses nœuds voisins, sa table de routage contenant l'état de tous ses liensconnus et permet ainsi, de garder une vision globale à jour de l'état du réseau car les sous ensemble du réseau possèdent alors assez d'information pour trouver le meilleur chemin jusqu'à tout autres éléments dans le réseau. Le trafic de contrôle ne contient que l'information sur les liens appartenant à ce nœud, au lieu de toute l'information sur tous les liens, ce qui constitue une économie certaine.

Dans le cas d'un réseau dense, les tables de routages deviennent vite volumineuses ce qui constitue un réel inconvénient. De plus, des routes peuvent être sauvegardées sans qu'elles ne soient utilisées.

Les principaux protocoles proactifs sont :

- *OLSR (Optimized Link State Routing)* : Il utilise un routage de type état des liens pour déterminer dynamiquement les tables de routage.

- *FSR (Fisheye State Routing)* : Chaque nœud diffuse sa table de routage avec une fréquence qui dépend du nombre de sauts qu'un paquet doit effectuer.

- *TBRPF (Topology Broadcast Based on Reverse-Path Forwarding)* : Les nœuds échangent régulièrement des informations sur la topologie du réseau afin de créer leurs tables de routage.

III.2 Les protocoles réactifs

Contrairement aux protocoles proactifs, les tables de routages des nœuds ne sont plus envoyées périodiquement, mais seulement *à la demande*, lorsque le trafic utilisateur doit être acheminé vers une destination vers laquelle un chemin n'est pas connu à ce moment-là.
Dans ce cas, une requête pour établir un tel chemin est diffusé dans tout le réseau. La destination (ou un nœud connaissant un chemin vers la destination) recevra alors cette requête diffusée, et pourra y répondre pour établir un chemin, envoyant une réponse prenant le chemin pris depuis la source de la requête.

La connaissance de ce chemin sera maintenue dans le réseau tant que le trafic utilisateur l'empruntera, puis disparaîtra une fois les informations transmises.

Ce type de méthode réduit la quantité d'information de routage à transmettre. Mais les méthodes réactives sont susceptibles de ne pas gérer correctement un trafic utilisateur empruntant des routes trop différentes et trop changeantes. En effet, le fait que l'existence de certains liens ne soit pas connue peut créer des situations au cours desquelles les chemins les plus courts ne sont pas connus, et donc pas utilisés.

Voici quelques exemples de protocoles réactifs :

- *AODV (Ad hoc On demand Distance Vector Routing)* : qui est un protocole de type vecteur de distance. L'étude de ce protocole sera détaillée dans le 3ème chapitre.
- *DSR (Dynamic Source Routing)*.

III.3 Les protocoles hybrides

Les protocoles hybrides sont utilisés dans un réseau découpé en zone. Ils emploient un protocole *proactif dans la zone*et un protocole réactif pour les communications inter-zones
Ils fonctionnent en mode proactif pour garder la connaissance locale de la topologie : l'Intrazone Routing Protocol (IARP). Ils utilisent le mode réactif pour les nœuds lointains : l'Interzone Routing Protocol (IERP).On a donc un routage à deux niveaux avec l'utilisation de zones qui permet d'optimiser la diffusion des requêtes de demande de « route ».

Le principal protocole hybride est ZRP (Zone Routing Protocol). C'est un protocole de type vecteur de distance. Il est plus résistant à la mobilité sur des réseaux de grandes tailles. Le protocole CBRP (Cluster Based Routing Protocol) fait également partie de cette famille.

III.4 Les protocoles dédiés aux réseaux de capteurs

Les protocoles dédiés pour les réseaux de capteurs, sont peu nombreux et les implémentations également. Les protocoles dédiés sont répartis en quatre catégories :

- protocoles hiérarchiques : énergie, agrégation et fusion des messages dans un cluster
- protocoles basés sur la localisation : les capteurs ont un système GPS basse puissance qui leur permet de déduire la distance et le coût
- protocoles centrés données : requêtes à certaines régions

- considération du flux réseau : adopte des approches d'établissement de chemins en considérant la QoS

On peut citer en exemple le protocole Spin (Sensor Protocols for Information via Negotiation).

III.5 Les protocoles à vecteur de distance

Les échanges se limitent au voisinage direct. Chaque nœud maintient une table de vecteur de distance (distance = coût, vecteur = direction) qui lui indique les nœuds atteints via ses voisins avec le coût de chaque chemin. Un nœud diffuse son DV périodiquement ou quand il le met à jour.

➢ ***Etapes d'un protocole vecteur de distance*** *:*

1. calculer le coût de chaque lien avec ses voisins directs,
2. échanger ceci avec les voisins uniquement,
3. mettre à jour sa table de routage en fonction des tables des voisins.

➢ ***Les informations échangées :***

1. chaque nœud envoie périodiquement à ses voisins :
2. le nombre de sauts qui le sépare d'une destination donnée,
3. le prochain saut vers cette destination.
4. rajoute les routes directement dans la table de routage.

III.6 Les protocoles à état de liens

Echanges *périodiques* avec tout le réseau. Chaque nœud informe tout le réseau de sa liste de voisins, ainsi tous les nœuds sont capables de construire la carte du réseau en considérant uniquement les liens bidirectionnels. La construction de carte du réseau permet à un nœud de construire une table de routage en appliquant un algorithme de plus court chemin sur la carte comme Dijkstra**[30].**

➢ ***Etapes d'un protocole état de liens :***

1. connaitre les voisins directs à l'aide des HELLO,
2. donner un coût à chaque lien,
3. diffuser cette information à tout le réseau,
4. appliquer un algorithme pour calculer les chemins vers toutes les destinations du réseau.

➢ ***Les informations échangées :***

– chaque nœud envoie des informations concernant :
  - ❖ ses liens avec ses voisins,
  - ❖ l'état de chaque lien.

– ces informations sont diffusées à tous les nœuds du réseau,
– chaque nœud calcule sa table de routage en se basant sur ces informations.

IV Exemples des protocoles de routages [26]

Dans ce qui suit, nous nous intéressons au cas de quelques protocoles de routage adaptés aux différentes topologies.

IV.1 protocole de routage non hiérarchique AD HOC

IV.1.1 le protocole de routage « DSDV »

DSDV Destination Sequenced Distance Vector est un protocole proactif de routage à vecteur de distance. Chaque nœud du réseau maintient une *table de routage* contenant le *saut suivant* et le *nombre de sauts* pour toutes les destinations possibles.
Des diffusions de mises à jour périodiques tendent à maintenir la table de routage complètement actualisée à tout moment.

Par ailleurs, la topologie des réseaux Ad-hoc est dynamique et des boucles de routes peuvent survenir lorsque des informations incorrectes de routage sont présentées dans le réseau après un changement dans la topologie du réseau (lien brisé par exemple).

Dans ce contexte et afin d'éviter le bouclage (loop), DSDV utilise les *numéros de séquence* (Sequence Number) pour indiquer la « nouveauté » d'une route. Une route R est considérée plus favorable qu'une autre R', si R a un numéro de séquence plus grand ; si ces deux routes ont le même numéro de séquence, alors R est plus favorable s'il possède un nombre inférieur de sauts.

Le numéro de séquence pour une route est initialisé par le nœud émetteur et incrémenté pour chaque nouvel avertissement d'une nouvelle route. Quand un nœud détecte un lien brisé vers une destination D, il met à jour le *nombre de sauts* pour l'entrée de la destination D dans sa table avec la *valeur infini* et incrémente son *numéro de séquence*.

IV.1.2 le protocole de routage « GSR »

Le protocole GSR *Global State Routing* est un protocole similaire au protocole DSDV décrit précédemment. Ce protocole utilise les idées du routage basé sur l'état des liens (Link State, LS). De la même manière que les protocoles LS, les messages de routage sont générés suivant les changements d'états des liens. Lors de la réception d'un message de routage, le nœud met à jour sa table de topologie (représentée par un graphe dans LS) et cela dans le cas où le numéro de séquence du message reçu est supérieur à la valeur du numéro de séquence sauvegardé dans la table (exactement comme le fait le protocole DSDV).

Par la suite, le nœud reconstruit sa table de routage et diffuse les mises à jour à ses voisins.
La principale modification de GSR sur l'algorithme LS traditionnel, est la façon de diffusion des informations de routage qui circulent dans le réseau. Dans LS, si on détecte des changements de topologie, les paquets d'états de liens sont générés et diffusés par inondation dans tout le réseau. Par contre, GSR maintient la table - la plus récente - d'état des liens reçus à travers les voisins, et l'échange uniquement avec ses voisins locaux, d'une façon périodique.

La table de distance contient la plus courte distance pour chaque nœud destination. Le calcul des chemins peut se faire avec n'importe quel algorithme de recherche des plus courts

chemins. Par exemple, l'algorithme du GSR utilise l'algorithme de *Dijkstra* modifié de telle façon qu'il puisse construire la table des nœuds suivants (NEXT HOP) et la table de distance Table_D, en parallèle avec la construction de l'arbre des plus courts chemins (l'arbre dont la racine est le nœud source).

IV.1.3 Le protocole de routage « FSR »

Le protocole *FSR Fisheye State Routing* peut être vu comme une amélioration du protocole GSR présenté précédemment. Le nombre élevé de messages de mise à jour échangés implique une grande consommation de la bande passante, ce qui a un effet négatif dans les réseaux Ad-hoc caractérisés par une bande passante limitée.

Le protocole FSR est similaire à LS, dans sa sauvegarde de la topologie au niveau de chaque nœud. L'image complète de la topologie du réseau est gardée au niveau de chaque nœud, et les meilleurs chemins sont échangés en utilisant cette image. La modification principale réside dans la manière avec laquelle les informations de routage circulent. Dans FSR, la diffusion par inondation de messages n'existe pas. L'échange se fait uniquement avec les voisins directs.

IV.1.4 Le protocole de routage «AODV»

AODVAd-hoc On Demand Distance Vector est un protocole à vecteur de distance, comme DSDV, mais il est réactif contrairement à 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 noeud suivant, la distance en nombre de sauts* (i.e. le nombre de nœuds nécessaires pour atteindre la destination), *le numéro de séquence destination* ainsi que *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 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 noeud source. Cette réponse de route traverse le long du chemin temporaire inversé du message RREQ. Précisons que le protocole AODV ne supporte que les liens symétriques dans la construction des chemins inverses.
La Figure II-5 illustre le mécanisme de création des routes d'AODV.

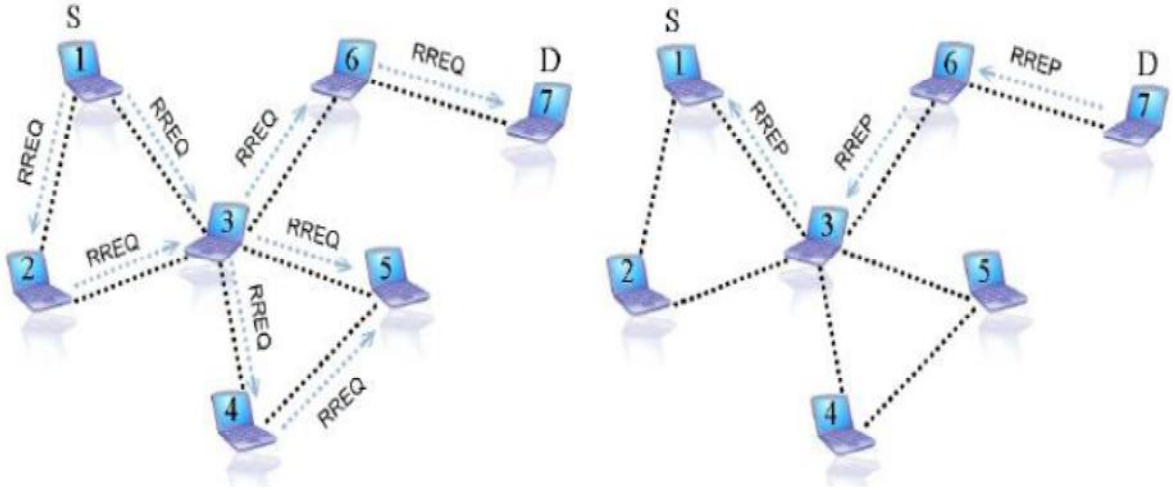


Figure II-5: Fonctionnement de la procédure de demande de route dans AODV

Il y a d'abord diffusion de la demande de route. Ensuite, la destination envoie une réponse, qui, grâce aux informations recueillies par les nœuds lors de la diffusion de la demande de route, crée une route de la destination vers la source. A noter que le protocole de routage AODV n'assure pas la détection du meilleur chemin existant entre la source et la destination.

Les nœuds voisins sont détectés par des messages périodiques HELLO (un message particulier de RREP). Si un nœud **x** ne reçoit pas un message HELLO d'un voisin **y** par lequel il envoie des données, ce lien est considéré brisé et une indication de défaillance de lien est envoyée à ses voisins actifs. Ces derniers propagent l'indication à leurs voisins qui utilisaient le lien entre x et y. Lorsque le message du lien brisé atteint finalement les sources affectées, celles-ci peuvent choisir d'arrêter l'envoi des données ou de demander une nouvelle route en envoyant un nouveau message RREQ.

Le chapitre suivant serra consacré à une étude plus approfondie d'AODV.

IV.2protocoles de routage non hiérarchiques dans les RCSF

IV.2.1le protocole de routage *« SPIN»*

Le fonctionnement du protocole SPIN permet de réduire la charge du réseau par rapport aux méthodes de diffusion traditionnelles telles que l'inondation, etc.

Le protocole SPIN utilise essentiellement trois types de paquets pour communiquer ADV/REQ/DATA respectivement pour advertises new data, requests data et le message actuel. Un nœud voulant émettre une donnée commence par envoyer un paquet ADV. Ce paquet ADV contenant des méta-données sur les données à émettre. Les méta-données contiennent la description des données en question. Les nœuds qui reçoivent le paquet ADV vérifient si les données les intéressent. Si c'est le cas, ils répondent par un paquet REQ pour demander les données. Le nœud qui a initié la communication envoie alors un paquet DATA pour chaque réponse REQ reçue (voir la Figure II.6). Lorsque le nœud s'aperçoit que son

énergie est descendue sous un certain seuil, il change son mode de fonctionnement, et ne répond à aucun message ADV.

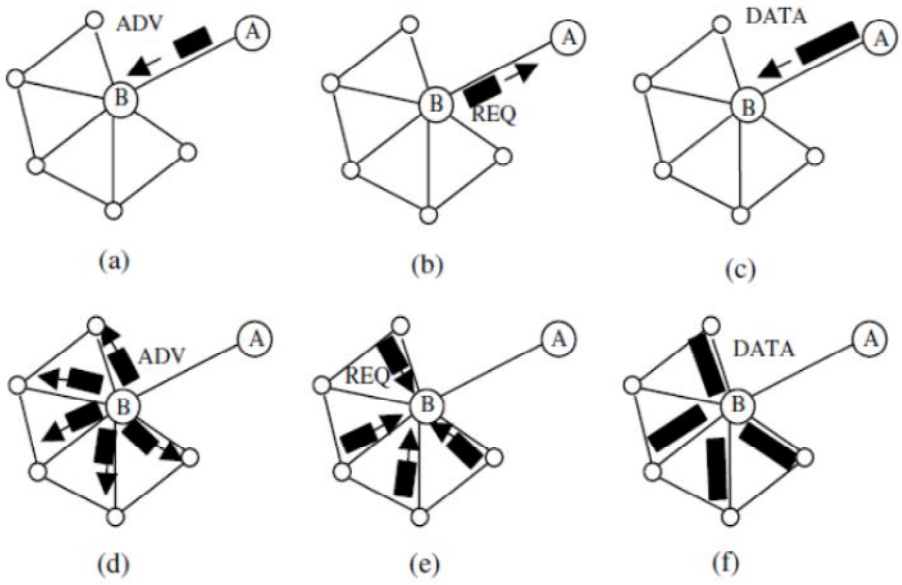


Figure II.6 : Fonctionnement du protocole SPIN

IV.2.2 Le protocole de routage « Direct Diffusion »

Le protocole *DD Direct Diffusion* est l'inverse de SPIN : les nœuds intéressés par une donnée diffusent une requête. Les nœuds voisins prennent en compte cette requête, répondent en fonction et rediffusent à leur tour la requête.

La diffusion dirigée est un protocole important dans le routage *data-centric* des réseaux de capteurs. L'idée vise à diffuser des données aux nœuds en utilisant un schéma de nommage pour les données. La diffusion dirigée suggère donc l'utilisation de paires attribut-valeur pour les données et les requêtes des capteurs.

Afin de créer une requête, un nœud est défini à l'aide d'une liste de paires attribut-valeur comme le nom des objets, l'intervalle, la durée, la zone géographique, etc. La requête contient aussi plusieurs champs de gradient.
Un gradient est un lien réponse avec un voisin dont le paquet a été reçu et qui est caractérisé par le débit, la durée et la date d'expiration de données.

Ainsi on utilisant les intérêts et les gradients, les routes sont établies entre la destination et les sources. *Plusieurs routes* sont établies de telle sorte que l'une d'elle est choisie par renforcement.

IV.3protocoles de routage hiérarchiques

IV.3.1 Le protocole de routage « LEACH »

Le protocole LEACH issu des travaux de Heinzelman et al**. [6][7]** présente aujourd'hui d'excellents résultats en termes d'économie d'énergie. S'appuyant sur le clustering qui consiste à partitionner le réseau en groupes (clusters). Les nœuds transmettent leurs données vers des représentants de groupes dits cluster-heads (CHs), qui à leur tour envoient ces données vers la destination désirée ou la station de base

Dans certaines applications les cluster-heads font des traitements simples (agrégations par exemple) sur les données reçues avant de les retransmettre à la station de base. Elle offre aussi une meilleure allocation de ressources et aide à améliorer le contrôle de l'énergie dans le réseau **[8] [9] [31].** Le fonctionnement de ce protocole serra plus détailler vers la fin de ce chapitre.

IV.3.2Les protocoles de routage «TEEN & APTEEN »

Les protocoles *Threshold sensitive Energy Efficient sensor Network* protocol (TEEN) et *Adaptive Threshold sensitive Energy Efficient sensor Network protocol* (APTEEN) conviennent pour les applications critiques en termede temps.

Le protocole TEEN est conçu pour être réactif aux changements soudains dans les attributs de contrôle tel que la température. La réactivité est très importante pour les applications temps réel où le réseau opère en mode réactif.

TEEN propose une *approche hiérarchique* avec l'utilisation d'un mécanisme *centré-données*.Dans *TEEN*, les nœuds capturent de manière continue, mais la transmission de données n'est pas faite fréquemment.

Après que les clusters soient formés, le cluster-Head (CH), qui est élu périodiquement en fonction de son niveau d'énergie, envoie à ses membres un *seuil hard* déterminant les attributs de capture, et un autre *soft* donnant les petits changements dans les valeurs des attributs de captures dans la période de transmission du cluster.

Les nœuds surveillent leur environnement continuellement. Si un paramètre des attributs atteint son seuil hard, et il y a un changement des valeurs capturées supérieur ou égal au seuil soft, le nœud envoie les données capturées à leurs destinations. A chaque changement de temps du cluster, les paramètres sont diffusés à nouveau.

Adaptative-TEEN (APTEEN) est une extension de TEEN qui fait à la fois la collection des captures périodiques de données et qui réagit aux événements critiques (réactif).

APTEEN a la même architecture que TEEN. Le cluster-head diffuse les attributs, les valeurs des seuils et l'ordonnancement de la transmission de tous les nœuds. Les cluster-heads effectuent l'agrégation de données pour économiser de l'énergie.

APTEEN supporte trois types différents de requêtes :

- *Historique* : pour analyser les anciennes valeurs des données
- *One-time* : pour avoir une vue instantanée du réseau
- *Contrôler un évènement pendant une certaine période.*

D'autres exemples de protocoles de routages dans les réseaux de capteurs sans fil sont résumés dans la figure suivante :

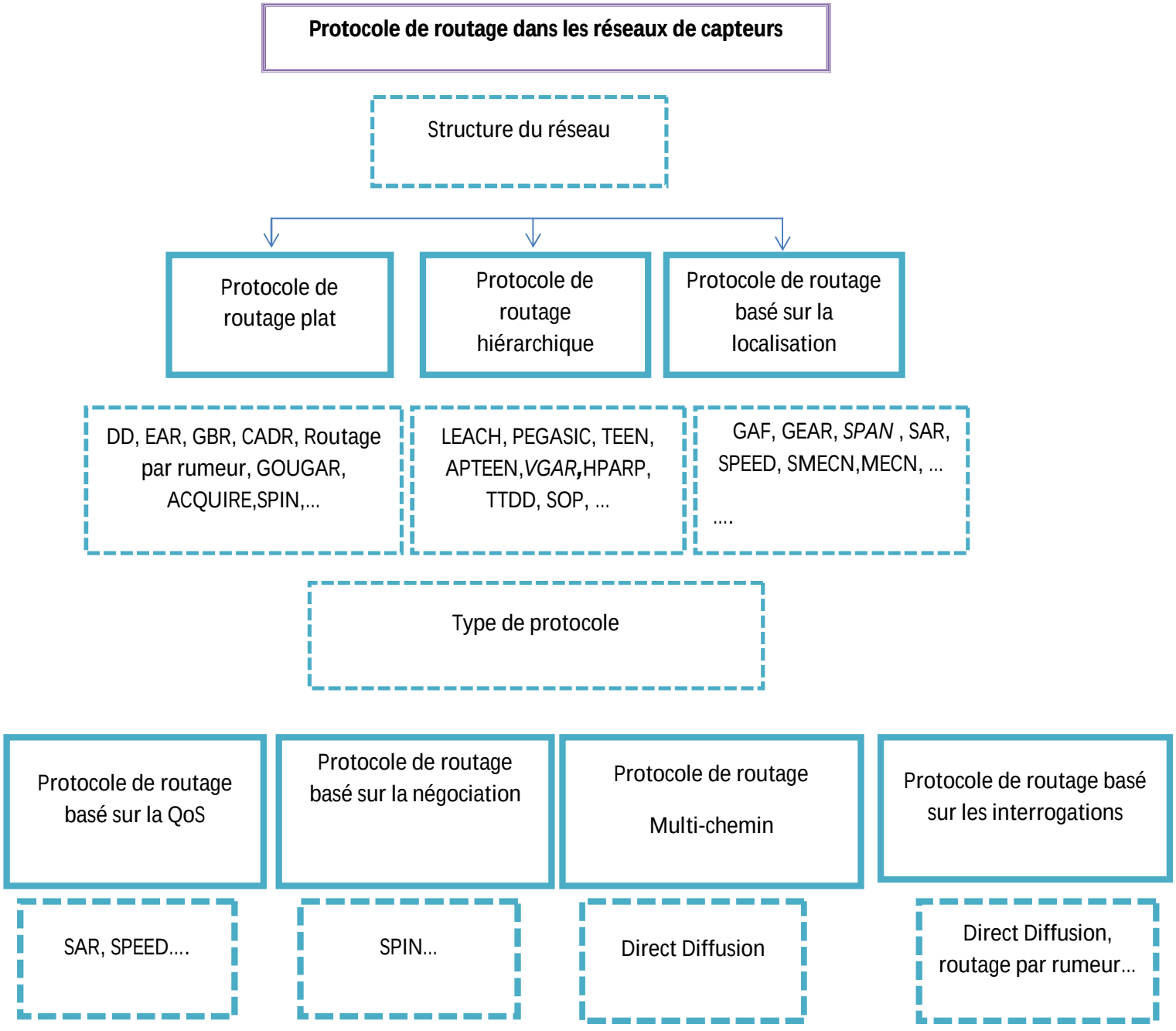


Figure II.7 Principaux protocoles de routages dans les RCSF[5]

V Le fonctionnement du protocole hiérarchique LEACH (*Low-energy Adaptive Clustering Hierarchy*)

LEACH est comme décrit dans la section **IV.2.2.1** un protocole de routage hiérarchique, qui se base sur le partitionnement des nœuds en *Clusters*. Pour chaque *Cluster* un nœud particulier est élu *Clusterhead*, ce dernier sera responsable de l'échange des données entre les nœuds membres et le Sink (Figure II.8).

Le *Clusterhead* est un nœud ordinaire qui est choisis selon un algorithme spécifique d'élection qui prend en compte des différents critères comme l'énergie disponible dans les nœuds ou si le nœud a servi comme Clusterhead pendant la dernière période.

Le rôle du *Clusterhead* est consommant en termes d'énergie puisqu'il est actif et prend en charge la transmission de toutes les données collectées des membres vers le *Sink* pendant toute la durée d'un cycle. Si ce rôle es fixé pour seul un nœud, il épuisera son énergie rapidement, et après sa défaillance, tous ses membres seront sans *Clusterhead* (*headless*) et donc inutile. C'est pourquoi, ce fardeau est échangé entre les différents nœuds du réseau.
Pour un membre, il est généralement beaucoup plus facile d'atteindre le *Clusterhead* que de transmettre directement vers le *Sink*.

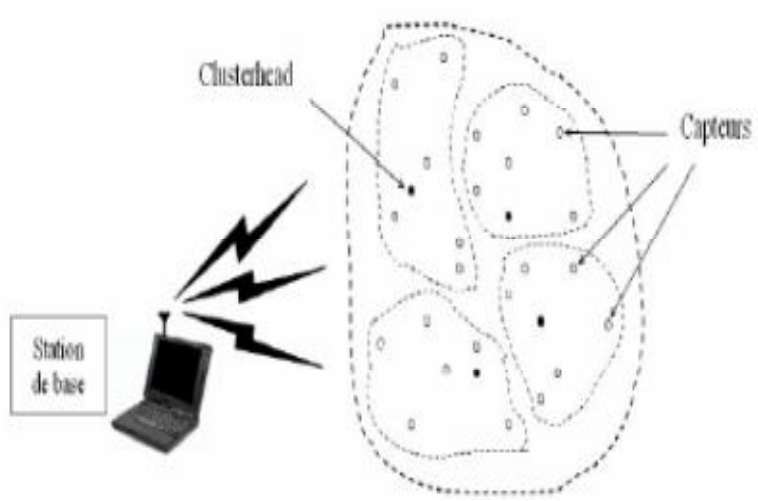


Figure II.8 Le Clustering dans un réseau de capteurs [32]

V.1. Protocoles MAC utilisés par LEACH

Pendant son fonctionnement, le protocole LEACH appelle certains schémas des protocoles MAC qui seront détaillés dans cette section pour mieux comprendre le déroulement de ce dernier.

Comme les RCSF ont des caractéristiques distinctes de tout autre type de réseaux sans fil, les protocoles MAC conçus pour ces derniers ne sont pas toujours applicables dans les RCSF.

Deux versions des protocoles MAC pour l'accès au media sont alors proposées pour les RCSF

- l'accès aléatoire ;
- et l'allocation fixe **[11]**.

V.1.1. *Accès aléatoire*

Dans le schéma à accès aléatoire, les nœuds qui possèdent des données à transmettre doivent essayer d'obtenir l'autorisation pour l'accès au media tout en réduisant les collisions avec les transmissions des données des autres nœuds.

Lorsqu'un nœud veut transmettre un message, il examine le média pour vérifier s'il est libre ou occupé par un autre nœud. Dans le cas où le media est libre, ce nœud pourra émettre son message afin d'éviter les collisions.

Cela dit, des nœuds peuvent émettre des données en même temps, ce qui mène à des collisions. Il est nécessaire donc que celles-ci soient détectées et que ces données soient retransmises. Si les retransmissions se passent encore en même temps, d'autres collisions vont se produire.

Une solution à ce problème consiste à introduire que chaque nœud attende un délai aléatoire avant de retransmettre ses données, ce qui réduit la probabilité d'une autre collision. **[12]**

V.1.2. *Allocation fixe*

Les schémas à allocation fixe permettent d'allouer pour chaque nœud le media de transmission suivant des intervalles de temps (schéma TDMA) ou un schéma de codage particulier (schéma CDMA).

Étant donné que chaque nœud est attribué en exclusivité à un intervalle, il n'y a presque pas de collisions entre les données. Toutefois, les schémas à allocation fixe s'avèrent inefficaces lorsque tous les nœuds n'ont pas de données à transmettre. En effet, ces intervalles sont affectés à des nœuds qui n'ont pas besoin de les utiliser. **[13]**

a. TDMA

Le schéma d'accès multiple à répartition de temps ou TDMA (*Time Division Multiple Access*) permet de diviser le temps en intervalles (*time-slot*) attribués à chaque nœud (voir figure II.9). Ainsi, un seul nœud a le droit d'accès au canal (il utilise toute la plage de la bande passante du canal), mais doit émettre ses données pendant les intervalles de temps qui lui sont accordés. **[12]**

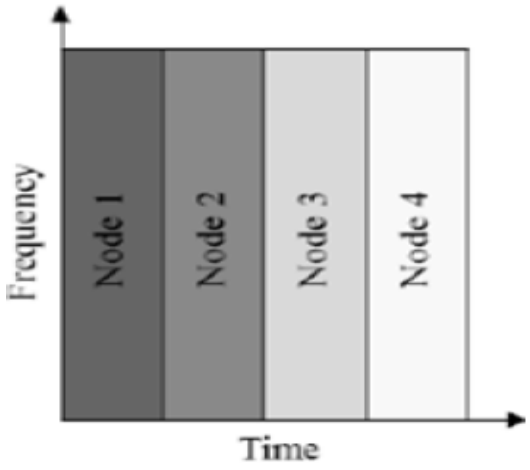


Figure II.9 Diagrammes représentant le protocole MAC TDMA. [15]

b. CDMA

Le schéma d'accès multiple par répartition en code ou CDMA (C*ode Division Multiple Access*) permet de côtoyer plusieurs nœuds simultanément (voir figure II.10).

En effet, il ne divise ni la plage de fréquences ni l'intervalle de temps. Ainsi, des nœuds peuvent émettre leurs données continuellement et selon une large plage de fréquence.

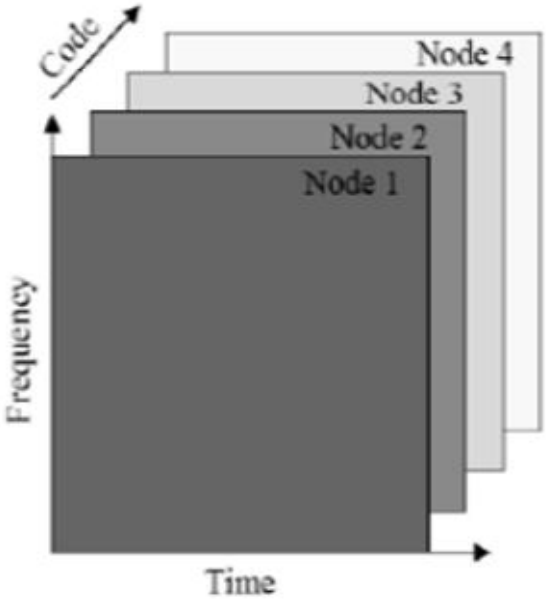


Figure. II.10 : Diagrammes représentant le protocole MAC CDMA[15].

V.2 Implémentation du protocole LEACH

V.2.1 Algorithme détaillé de LEACH

L'algorithme se déroule en cycles ou « *rounds* ». *Chaque round à* approximativement le même intervalle de temps déterminé au préalable. Un round est constitué de deux phases :

- Une phase d'initialisation
- une phase de transmission.

V.2.1.1. Phase d'initialisation

La phase d'initialisation est composée de 3 sous phases:

- Phase d'annonce,
- Phase d'organisation des groupes
- Phase d'ordonnancement,

Phase d'annonce :

C'est la phase où les clusters sont formés et les cluster-heads CHs sont élus pour une période déterminée « round ». Cette élection se fait d'une manière cyclique dans le but d'équilibrer la dissipation d'énergie.

On estime que le pourcentage optimal du nombre de CHs désirés devrait être de 5% à 15% du nombre total de nœuds **[16]**.

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 énergétiques. Ainsi, on aura une dissipation d'énergie considérable dans le réseau.

Et 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 de travail important qui leur est demandés.

On commence la phase d'annonce par l'annonce du *nouveau round* qui sera faite par le nœud puits, mais également par la prise de décision locale d'un *nœud pour devenir CH* au début du round qui commencera à l'instant t.

Durant cette étape, chaque nœud N choisit aléatoirement un nombre dans l'intervalle [0,1], si ce nombre est inférieur à un seuil Pi(t) alors le nœud est élu cluster-head *CH durant le round r+1*.

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

Où : K est le nombre de CHs désirés, ce nombre reste fixe et inchangé durant tous les rounds.

La probabilité Pi(t) de devenir CH pour chaque nœud i est calculée ainsi **[57]**:

$$P_i(t) = \frac{\text{le nombre de CH désirés}}{\text{Le nombre de nœuds qui n'ont pas encore été élus CH durant les r rounds précédents}}$$

$$P_i(t) = \begin{cases} \dfrac{K}{N - k*(r \bmod N/k)} & : Ci(t)=1 \\ 1 & : Ci(t)=0 \end{cases} \qquad \dots (1)$$

Où

- **N** : est le nombre total de nœuds dans le réseau. Si on a N nœuds et K CHs, alors, il faudra N/K rounds durant lesquels un nœud doit être élu seulement une seule fois autant que CHs avant que le round soit réinitialisé à 0.
- **r** : numéro du round courant ;

- **Le terme Ci(t)** : représente si un nœud i est élu ou pas, il est égal 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.

$\sum_{i=1}^{N} Ci(t)$ Représente le nombre total des nœuds éligibles d'être CH à l'instant t. Il est égal à :

$$\sum_{i=1}^{N} Ci(t) = N - K*(r \bmod N/K) \qquad \dots (2)$$

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

$$Nombre(CH) = \sum_{i=1}^{N} Pi(t) * Ci(t) = (K * (r \bmod N/K)) * \left(\frac{K}{N - k*(r \bmod N/k)}\right) = K$$

- **Phase d'organisation de groupes [10] :**

Dès que les cluster-heads sont élus, chacun d'eux envoie un message de notification aux autres nœuds du réseau, i.e : chaque nœud élu CH, doit informer les autres nœuds non-CH de son nouveau rang dans le round courant et cela par diffusion d'un un *message d'avertissement ADV* contenant *l'identificateur du CH* à tous les nœuds non –CH en utilisant le protocole MAC CSMA pour éviter les collisions entre les CH.

Les nœuds non-CH décident alors de leur appartenance à un cluster selon l'amplitude du signal reçu en choisissant le signal le plus fort.

- **Phase d'ordonnancement [18]:**

Après la formation des groupes, chaque CH agit comme un centre de commande local 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 ou créneau 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.

La figure II.11 nous résume les trois sous phases de la phase d'initialisation :

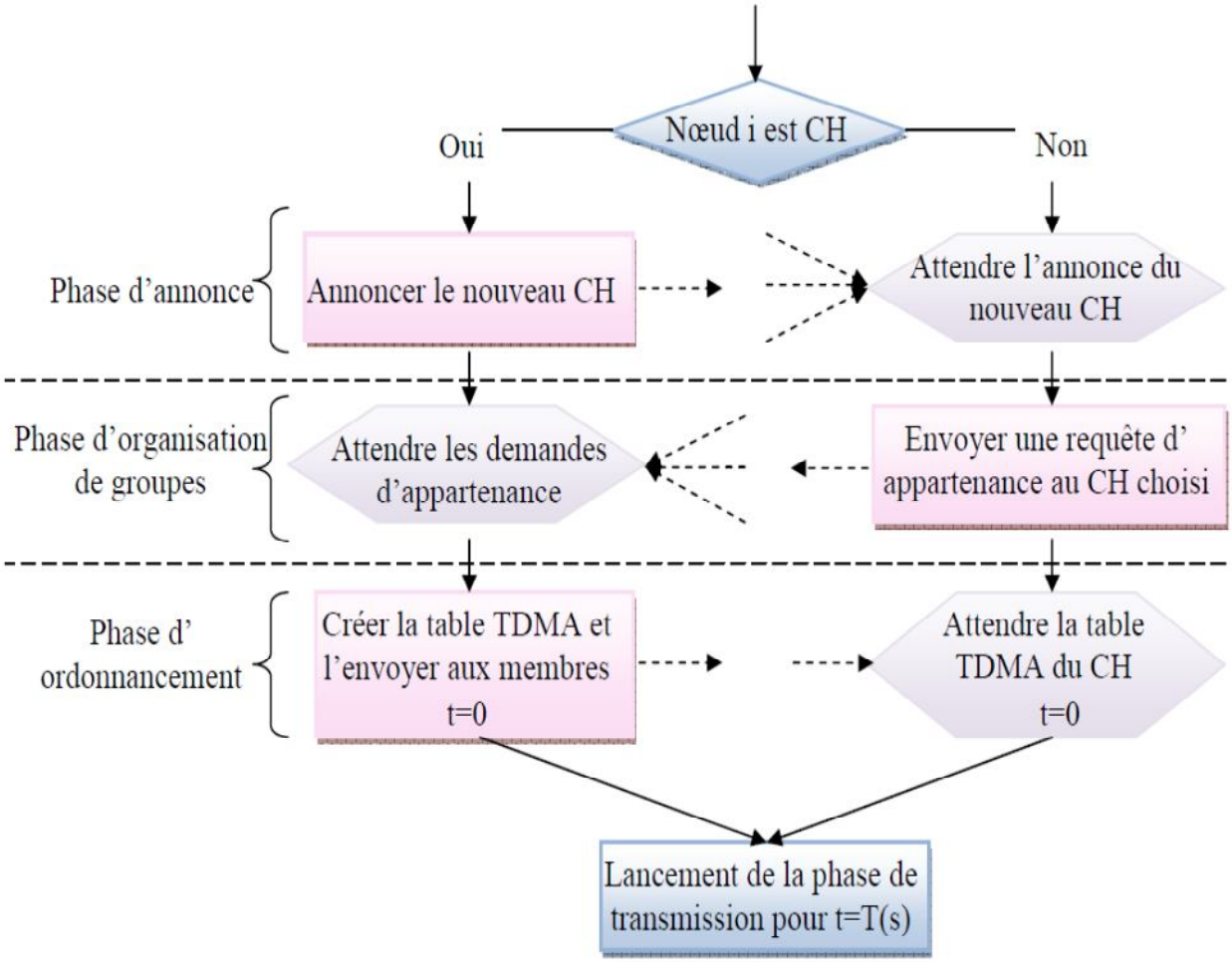


Figure. II.11 : Opérations de l'étape d'initialisation de LEACH [16]

V.2.1.2. Phase de transmission

Cette phase est plus longue que la phase précédente, et permet la collecte de données captées. En utilisant l'ordonnanceur TDMA, les membres émettent leurs données captées pendant leurs propres slots (Figure II-12)

Cela leur permet d'éteindre leurs interfaces de communication en dehors de leurs slots afin d'économiser leur énergie. Ces données sont ensuite agrégées par les CHs 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 du réseau soient élus CH, une seule fois, tout au long des rounds précédents. Dans ce cas, le round est réinitialisé à 0.

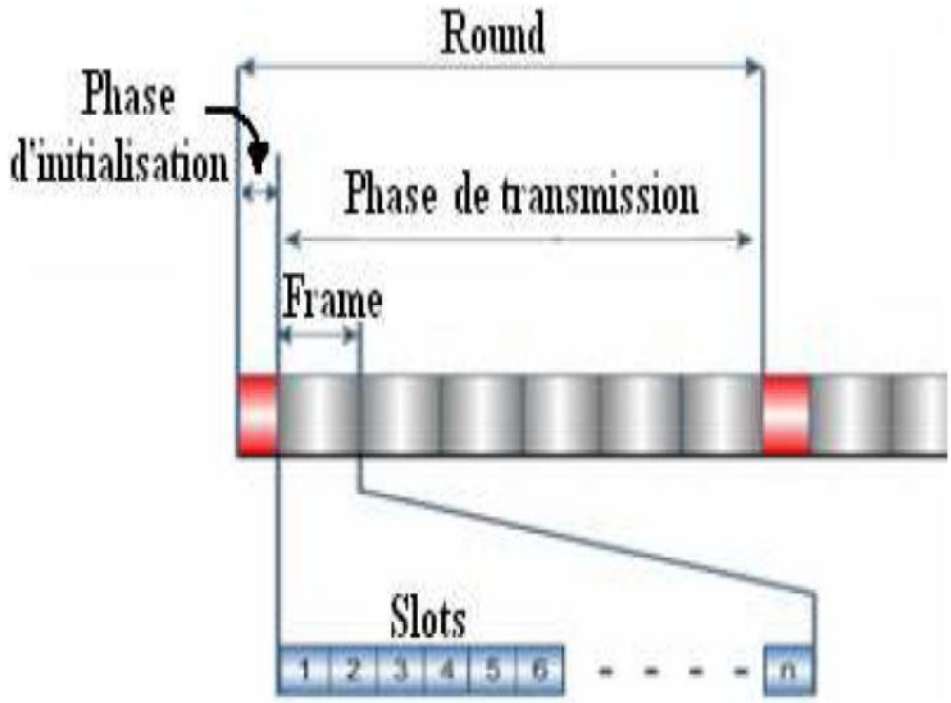


Figure. II.12 : Répartition du temps et différentes phases pour chaque round. **[10]**

VI Comparaison entre les protocoles de routage :

Le tableau suivant résume les caractéristiques des principaux protocoles de routage dans les réseaux de capteur **[22], [6]**

	Classification	Mobilité	Basé sur la négociation	Agrégation	Mise à l'échelle	Multi-chemin	Conservation d'énergie
Diffusion dirigée	Plat	Possible	Oui	Oui	Limitée	Oui	Limitée
EAR	Plat	Limitée	Non	Non	Limitée	Non	Min
GBR	Plat	Limitée	Non	Oui	Limitée	Non	Min
Rumeur Routing	Plat	Très limitée	Non	Oui	Bonne	Non	Min
LEACH	Hiérarchique	Station de base fixe	Non	Oui	Bonne	Non	Max
PEGASIS	Hiérarchique	Station de base fixe	Non	Non	Bonne	Non	Max

Hiérarchique-PEGASIS	Hiérarchique	Station de base fixe	Non	Non	Bonne	Non	Max
TEEN and APTEEN	Hiérarchique	Station de base fixe	Non	Oui	Bonne	Non	Max
MECN	Hiérarchique	Non	Non	Non	Faible	Non	Min
GAF	Géographie	Limité	Non	Non	Limitée	Non	Limitée
SPIN	Plat	Possible	Oui	Oui	Limitée	Oui	Limitée

TABLEAU.II .2: **Classification et comparaison des protocoles de routages dans les réseaux de capteurs**

Conclusion

La communication est très importante dans un réseau de capteurs, puisqu'elle permet aux nœuds de coopérer entre eux et d'acheminer les données vers leur destination.

Dans ce chapitre nous avons pu aborder les techniques de routage au niveau des réseaux de capteurs sans fil, nous avons également fait une étude détaillée des critères de classification des protocoles de routage. Ces protocoles de routage ont été classifiés selon plusieurs critères, parmi ces critères: la structure du réseau, le paradigme de communication, les types de protocoles, etc. Chaque critère regroupe plusieurs catégories et certains de ces protocoles peuvent appartenir à une ou plusieurs catégories de routage.

L'objectif de la plupart des protocoles est d'optimiser la consommation d'énergie afin d'assurer une durée de vie plus longue au réseau ainsi que d'éviter la redondante des paquets, c'est particulièrement dans cet objectif qu'on a mis l'accent sur l'étude du fonctionnement de l'un des principaux protocoles de routage : le protocole LEACH.

Le chapitre suivant serra consacré à l'étude du protocole de routage AODV, ce qui va nous aider pour la suite de notre étude et nous permettre de voir son adaptabilité au sein des réseaux de capteurs.

CHAPITRE III :

LE PROTOCOLE DE ROUTAGE AODV

Chapitre III : Le protocole de routage AODV

Introduction

AODV signifie Ad hoc On demand Distance Vector. C'est un protocole réactif, c'est-à-dire qu'il ne crée les routes que lorsque c'est utile. En s'affranchissant de maintenir les tables de routages constamment, il permet une réduction sensible du nombre de trames échangées, et donc une amélioration de la bande passante. Lorsqu'il est utilisé dans un réseau important, AODV génère un gros volume de diffusion, même pour des destinations qui sont proches de l'émetteur.

AODV est une combinaison de DSDV et de DSR. Il emprunte, à DSR ses mécanismes de découverte et de maintenance des routes « Route Discovery » et « Route Maintenance »;et à DSDV, son routage « saut par saut », les numéros de séquences ainsi que la diffusion des mises à jour des tables de routage. Ce protocole fait l'objet d'un grand nombre de recherches dû à toutes ses caractéristiques; il est en effet l'un des principaux protocoles au sein du groupe de recherche MANET.

AODV communique avec le principe d'envoie et de réception de messages entre les différents nœuds qui sont RREQ, RREP, RERR qui seront détaillés dans la suite de ce chapitre.

I Table de routage

Chaque nœud maintient une table de routage. Cette table contient une entrée pour chaque destination accessible. Une entrée contient principalement les informations suivantes :

- L'adresse de la destination,
- Le numéro de séquence associé à la destination,
- La distance qui est le nombre de sauts nécessaire pour atteindre la destination (hop count),
- Le nœud suivant, à qui il faut envoyer les paquets (next hop),
- Le temps de vie pour lequel la route est considérée correcte. (Le temps d'expiration de l'entrée de la table),
- La liste des voisins qui utilisent cette route,

Si une nouvelle route est nécessaire, ou qu'une route disparaît, la mise à jour de ces tables s'effectue par l'échange de trois types de messages entre les nœuds **[14]** :

- ***RREQ*** Route Request, un message de demande de route,
- ***RREP*** Route Reply, un message de réponse à un RREQ,
- ***RERR*** Route Error, un message qui signale la perte d'une route.

II Principe des numéros de séquence

L'AODV utilise les principes des numéros de séquence à fin de maintenir la consistance des informations de routage. Les numéros de séquence permettent d'utiliser les routes les plus nouvelles ou autrement dit les plus fraîches (fresh routes). Chaque nœud possède un numéro de séquence. Il est le seul habilité à l'incrémenter. Ce numéro personnel ne peut être incrémenté que dans deux situations :

- La première situation: Avant d'entreprendre un processus de recherche de route par l'envoi d'un paquet RREQ, le nœud incrémente son numéro.

- La deuxième situation : Avant de répondre à un message RREQ par un message RREP, le numéro de séquence doit être remplacé par la valeur maximale entre son numéro de séquence actuel et celui contenu dans le message RREQ.

Une mise à jour de la table de routage ne s'effectue que si l'une des conditions suivantes est observée :

- Le numéro de séquence du paquet de contrôle est strictement supérieur au numéro de séquence présent dans la table.
- Les numéros de séquence (de la table et du paquet) sont égaux mais, la distance en hops du paquet plus 1 est inférieure à la distance actuelle dans la table de routage.
- Le numéro de séquence pour cette destination est inconnu.

Cette façon de procéder garantit la création de route sans boucles.

III Formats des messages [27][28]

III.1 Route Request (RREQ) Message Format : requête de demande de route

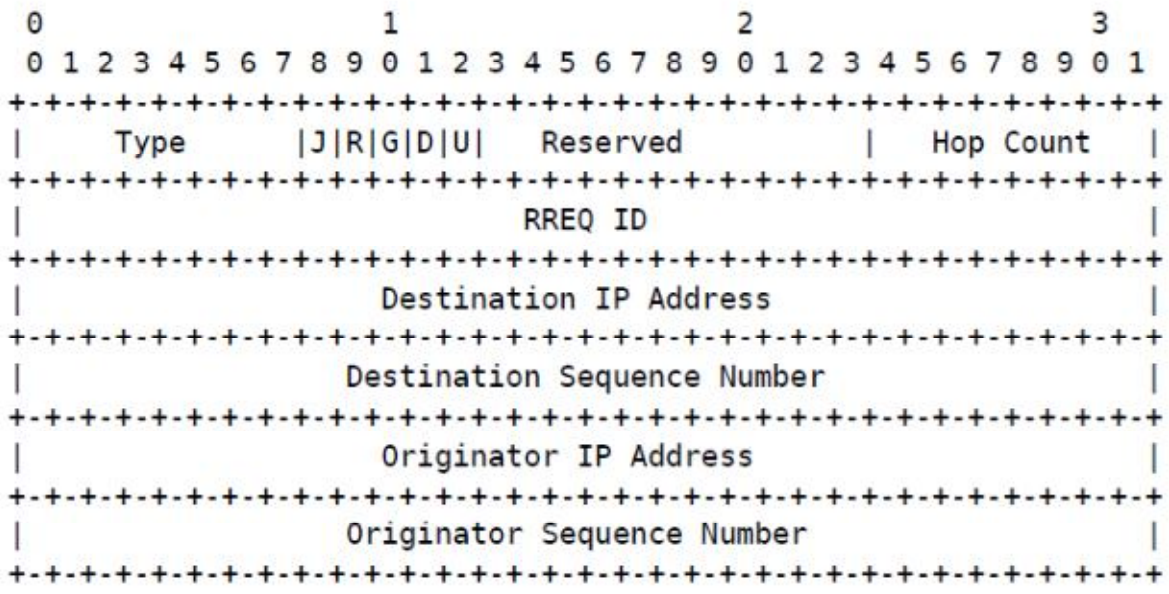


Figure III-1 : Format d'un message RREQ[27]

Le format du message RREQ est illustré ci-dessus, il contient les champs suivants:

- **Type :** 1
- **Le champ J** (Join flag) : Il est réservé pour le multicast
- **Le champ R** (Repair flag) ou indicateur de réparation, réservé pour le multicast.
- **Le champ G** (Gratuitous RREP flag) : Drapeau indiquant que le paquet RREQ est un paquet gratuit (Gratui-tous RREP). Ce type de paquets est envoyé de la destination vers la source pour vérifier que le chemin du retour est opérationnel (i.e. que la route source- destination est bidirectionnelle)
- **Le champ D** (Destination Only Flag) : Drapeau indiquant que seule la destination peut répondre à ce paquet.
- **Le champ U**(Unknown Destination Flag) : Drapeau indiquant que le numéro de séquence de la destination est inconnu.

- **Le champ Reserved** : Fixé à 0 par la RFC, ignoré à la réception.
- **Le champ Hop Count** : Indique le nombre de sauts depuis le nœud ayant généré le paquet jusqu'au nœud actuel.
- **Le champ RREQ ID** : Ce champ identifie de manière unique un paquet RREQ au cours d'unprocessus de découverte de route.
- **Le champ Destination IP Address** : Renferme l'adresse IP de la destination du message.
- **Le champ Destination SequenceNumber** : Renferme le numéro de séquence de la destination du message.
- **Le champ Originator IP Address** : Renferme l'adresse IP de la source du message.
- **Le champ OriginatorSequenceNumber** : Renferme le numéro de séquence de la source du message.

III.2 Route Reply (RREP) Message Format

Un message RREP se distingue d'un message RREQ par les champs suivants, comme indiqué dans la figure

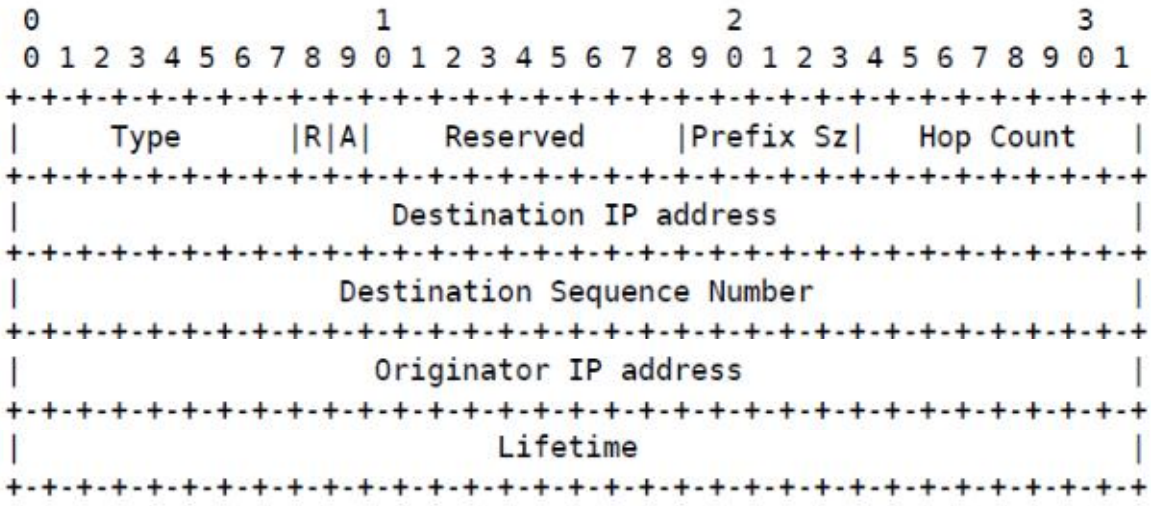


Figure III-2 : Format d'un message RREP[27]

- **Le drapeau R** (Repair Flag) : Ce drapeau est utilisé pour le multicast.
- **Le drapeau A** (Acknowledgement required): Ce drapeau indique qu'un acquittement est requis.
- **Le champ Prefix Size :** Si ce champ codé sur 5 bits a une valeur non nulle, alors son contenu spécifie que le saut suivant indiqué peut être utilisé par tout autre nœud ayant le même préfixe que la destination. Ceci a pour objectif de permettre le routage par sous-réseaux (subnets) en définissant un seul préfixe de routage commun à tous les nœuds d'un sous-réseau.
- **Le champ Lifetime** : indique la durée en millisecondes pendant laquelle un nœud recevant le message RREP peut considérer la route annoncée valide.

III.3 Route ERRor (RERR) Message Format

Le format des messages Route Error (RERR) est illustré dans la figure 3

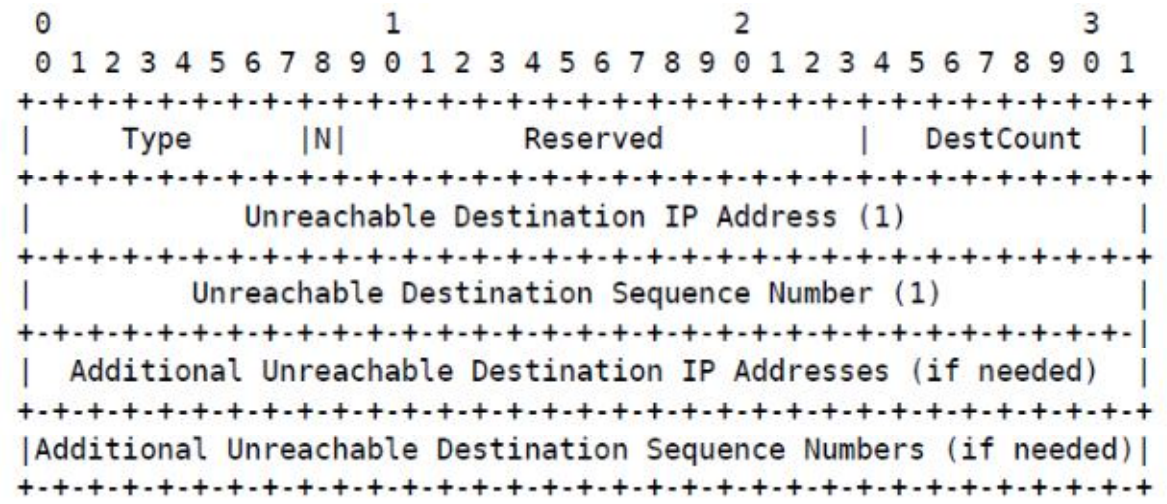


Figure III- 3 : format d'un message RERR[27]

Il se distingue des autres messages par les informations suivantes :

- **Le drapeau N (No Delete Flag) :** Ce drapeau est utilisé si un nœud vient de réparer un lien local. Il indique aux autres nœuds de ne pas supprimer la route de leurs tables de routage.
- **Le champ DestCount :** Ce champ indique le nombre de destinations non joignables annoncées dans le message. Du fait de la nature des messages RERR, ce champ doit valoir au moins 1.
- **Le champ Unreachable Destination IP Address :**Dans ces champs, il est indiqué la liste des adresses IP des différentes destinations non joignables concernées par ce message.
- **Le champ Unreachable Destination SequenceNumber :** Dans ces champs, il est indiqué la liste des numéros de séquences des différentes destinations non joignables concernées par ce message.

IV Le mécanisme de découverte de route (Route Discovery)

Le mécanisme de découverte de route est le processus de découvertes des routes à la demande des nœuds sources. Nous verrons dans cette partie le détail de cette procédure ainsi que le rôle joué par chacun des nœuds du réseau (source, nœuds intermédiaires, destination).

Remplissage de la table de routage et création des listes des précurseurs

Quand un nœud reçoit un paquet de contrôle AODV d'un voisin ou quand il crée ou met à jour une route vers une destination particulière ou un sous-réseau, il consulte sa table pour chercher une éventuelle entrée correspondante. Si sa recherche s'avère infructueuse, le nœud

y crée une nouvelle entrée. Sinon, la mise à jour n'est effectuée qu'après comparaison des numéros de séquences de la destination de la nouvelle entrée avec l'ancienne.

Enfin, pour chaque route enregistrée dans la table de routage, le nœud maintient une liste de précurseurs pouvant acheminer le trafic sur celle-ci. La liste des précurseurs contient la liste des nœuds voisins ayant acheminé ou généré un message Route Reply (RREP).

IV.1 Génération des messages Route Request (RREQ)

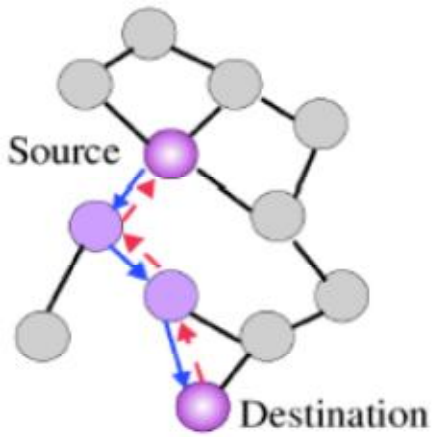
Un nœud génère un message Route Request (RREQ) quand il veut transmettre des données à une destination qui ne figure pas dans sa table de routage. Le nœud commence par enregistrer l'ID du message RREQ qu'il va transmettre ainsi que l'instant de génération correspondant (PATH_DISCOVERY_TIME) pour ne pas avoir à traiter le message quand ses voisins vont le lui propager.

Ensuite, la source émet le message RREQ et déclenche un compteur en attendant une réponse. Si à l'expiration aucun message RREP n'est reçu, la source peut retenter de retransmettre un message RREQ après un temps de Back off et ainsi de suite, sans pour autant dépasser un nombre maximal de tentatives RREQ_RETRIES.

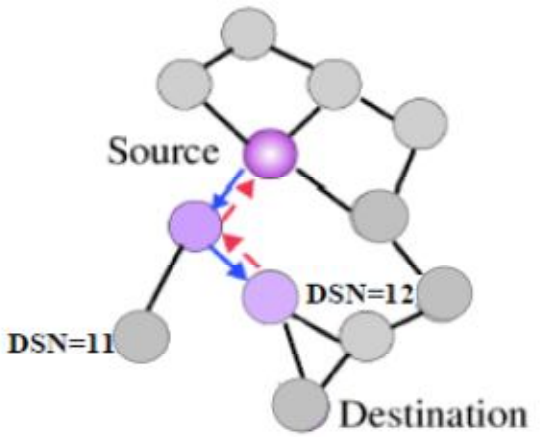
IV.3 Traitement et acheminement des messages Route Request (RREQ)

Quand un nœud reçoit un message RREQ, il commence par mettre à jour la route vers le saut précédent (par où est venu le message). Ensuite, il vérifie s'il n'a pas reçu la même requête auparavant, si c'est le cas, il détruit le paquet reçu. Si la requête n'a pas été reçue auparavant, le nœud incrémente le nombre de sauts du paquet puis cherche une route vers la source dans sa table. Selon le résultat de cette recherche, le nœud peut mettre à jour ou insérer une nouvelle route vers la source. Le nœud vérifie l'adresse de destination et dans ce cas deux cas se présentes :

- Il est la destination (Figure III-4.c),
- Il possède une route vers la destination avec un numéro de séquence supérieur ou égal à celui repris dans le RREQ (Figure III-4.d) il envoie (en *unicast*) un paquet RREP vers la source.



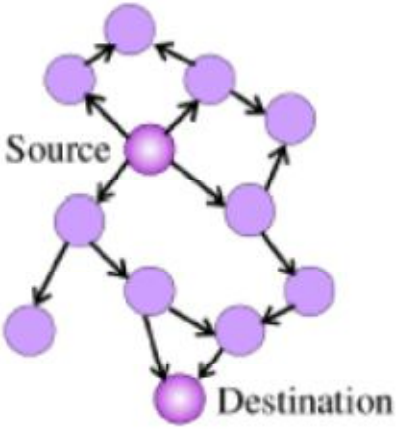
III-4-c. *Formation du chemin bidirectionnel vers la source*

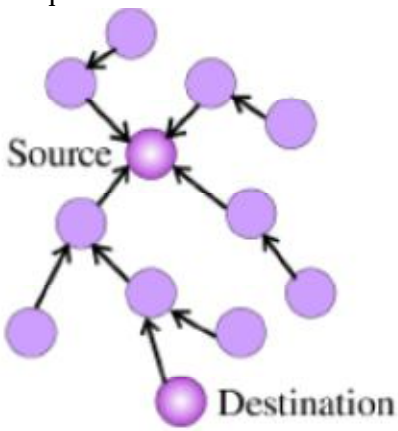


III-4-d. *Formation du chemin bidirectionnel vers la source[29]*

Chaque nœud traversé incrémentera le nombre de hops (sauts) et ajoutera une entrée à sa table pour la destination. Un chemin bidirectionnel est donc désormais disponible entre la source et la destination. Dans le cas contraire il rediffuse le RREQ (Figure III-4.a) et met à jour l'information relative à la source et établit des pointeurs de retour vers la source du RREQ dans les tables de routage (figure III-4.b).

Chaque nœud conserve une trace des IP sources et des ID de broadcast des RREQ afin d'éliminer les paquets redondants grâce à leur ID identiques.





III-4-a.*Propagation du paquet RREQ* **III-4-b.** *Formation de chemin inverse[29]*

Remarque : les figures a,b,c,d dans cet ordre forme les étapes de découverte de route

IV.4 Réception et acheminement des messages Route Reply (RREP)

Une fois que la source a reçu les RREP, elle peut commencer à émettre des paquets de données vers la destination. Si, ultérieurement, la source reçoit un RREP contenant un numéro de séquence supérieur ou le même mais avec un nombre de saut plus petit, elle mettra à jour son information de routage vers cette destination et commencera à utiliser la meilleure route.

Afin de limiter le coût dans le réseau, AODV propose d'étendre la recherche progressivement. Initialement, la requête est diffusée à un nombre de sauts limité. Si la source ne reçoit aucune réponse après un délai d'attente déterminé, elle retransmet un autre message de recherche en augmentant le nombre de sauts maximum. En cas de non réponse, cette procédure est répétée un nombre maximum de fois avant de déclarer que cette destination est injoignable.

IV.5 Exemple de découverte de routes

Dans l'exemple qui suit, nous montrons comment AODV effectue la découverte de routes. Nous nous plaçons dans le cas où le nœud A souhaiterait communiquer avec le nœud I, en supposant que le nœud A ne possède pas de route vers I dans sa table de routage et que I n'est pas à l'intérieur de sa zone de portée.

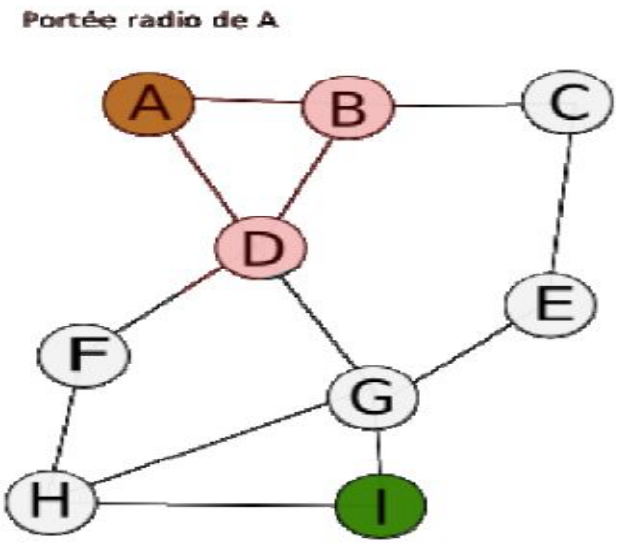


Figure III-5 : mise en place du protocole[29]

1. A construit un paquet spécial de demande de route "RREQ". Ce paquet est diffusé en broadcast aux nœuds voisins (B, D sur le schéma).

2. Le paquet arrive sur le nœud B et D. Chacun d'eux effectue ce traitement :
   Si le couple unique Adresse Source et Identifiant de requête présent dans le paquet est déjà dans la table d'historique local du nœud, alors ce paquet est rejeté, car le nœud en déduit qu'il s'agit d'un doublon. Sinon, il stocke le couple dans sa table locale et le traitement se poursuit à l'étape 3.

3. Le récepteur examine la destination dans sa table de routage, si elle existe alors, il émet un paquet RREP à destination de l'émetteur pour le prévenir qu'il peut passer par lui. L'envoi du paquet RREP n'est seulement effectué que lorsqu'une route vers la destination est plus récente que celle déjà calculée. Afin de déterminer si une route est plus récente qu'une autre, les nœuds du réseau se basent sur le numéro de séquence de destination. L'algorithme est le suivant : si le numéro de séquence stocké dans la table de routage du nœud est supérieur ou égal à celui du paquet alors la demande est rejetée. Dans le cas où la route est plus ancienne, alors nous passons à l'étape suivante.

4. Les nœuds B et D ne connaissant pas la destination, incrémentent le Hop Count du paquet et extraient, de ce dernier, les informations nécessaires à la construction de la route inverse. Le Hop Count passe à 1, et le paquet est à nouveau diffusé. La diffusion de B atteint C et D. C crée alors une entrée de route inverse et rediffuse le paquet. D, quant à lui, a déjà reçu ce paquet par A et rejette l'envoi de B car il s'agit, en effet, d'un doublon.
   La diffusion de D atteint les nœuds F et G et le paquet est enregistré. Après que les nœuds E, H, I ont réceptionné le paquet, celui-ci atteint I qui évidemment sait où I se trouve.

5. En réponse à la demande, I forme un paquet de réponse RREP qu'il envoie en mode unicast. Ce paquet emprunte la route inverse créée lors de la découverte de routes. Le numéro de séquence, destination est incrémenté par I à partir de la valeur qu'il avait dans sa table locale, pour ainsi dire, le numéro de séquence passe à 1. Lorsque le

paquet effectue le chemin inverse jusqu'à l'émetteur, le Hop Count est incrémenté afin que l'émetteur connaisse la distance qui le sépare.

Le chemin inverse obtenu est désigné par les segments en rouges, ces segments sont orientés vers le haut.

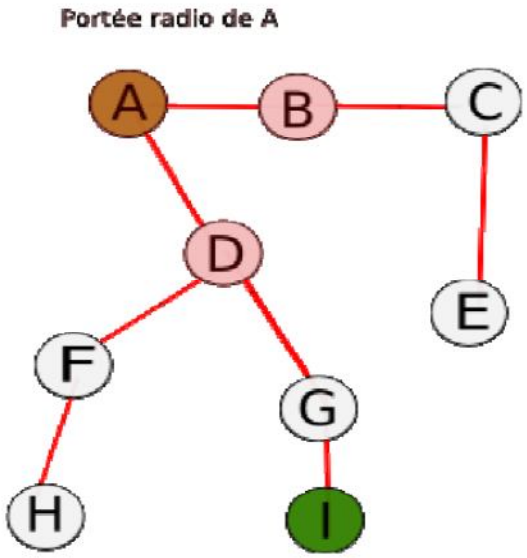


Figure III-6: découverte de route

V Le mécanisme de maintenance de route (Route Maintenance)

La maintenance des routes correspond au processus qui permet à un protocole de s'adapter à la topologie changeante du réseau. Le principe consiste à vérifier par intervalle de temps régulier si un nœud présent dans la table de routage demeure toujours accessible. Les changements détectés par ce processus correspondent à des cas variés : déplacement d'un nœud dans une zone plus éloignée, atténuation de la batterie, déconnection.

V.1 Maintien de la connectivité locale

La RFC 3651 exige que chaque nœud garde trace de sa connectivité aux nœuds voisins ainsi que la connectivité des autres voisins à lui. Pour cela, et Afin de maintenir des routes consistantes, une transmission périodique du message « HELLO » (qui est un RREP avec un TTL de 1) est effectuée. Si trois messages « HELLO » ne sont pas reçus consécutivement à partir d'un nœud voisin, le lien en question est considéré défaillant. Les défaillances des liens sont, généralement, dû à la mobilité du réseau. Les mouvements des nœuds qui ne participent pas dans le chemin actif, n'affectent pas la consistance des données de routage.

Dans le cas de défaillance d'un lien un des traitements suivant sont appliqués.

V.2 Traitements liés aux messages Route Error (RERR)

Un nœud génère un message Route Error (RERR) dans l'un des cas suivants :

- Ou bien il détecte une coupure de lien avec le saut suivant d'une route active dans sa table de routage au moment de la transmission de données
- Ou bien il reçoit un paquet de données destiné à un nœud vers lequel il n'a pas de route active dans sa table de routage

- Ou bien il reçoit un message RERR d'un de ses voisins

D'une manière générale, la création et l'envoi de messages RERR requiert les étapes suivantes :

- Lister les destinations concernées
- Déterminer les voisins (figurant dans la liste des précurseurs de la route) touchés par cette invalidation
- Délivrer des messages RERR appropriés à ces voisins

Selon le nombre des précurseurs, les messages RERR sont envoyés en mode broadcast (s'il y a plusieurs précurseurs), unicast (s.il n'y a qu'un seul).

V.3 Réparation locale (Local Repair) [28]

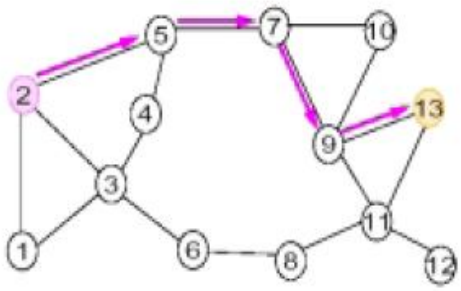
Quand un lien d'une route active est coupé, le nœud peut choisir de le réparer si la distance qui le sépare de l'autre extrémité du lien ne dépasse pas un nombre maximal MAX_REPAIR_TTL de sauts. Pour réparer le lien, le nœud incrémente le numéro de séquence de la destination puis diffuse un message RREQ vers celle-ci.

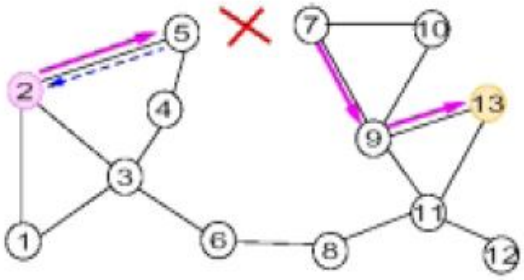
Si au bout de cette phase de redécouverte du lien une réponse RREP de la destination en question n'est pas reçue, la phase de réparation est vouée à l'échec et le nœud commencera à générer et diffuser un message RERR comme il a été détaillé dans le paragraphe précédent.

Par contre, si le nœud reçoit un ou plusieurs messages RREP pour la requête diffusée, il génère un message RERR pour indiquer aux autres nœuds que la route contenant le lien défectueux ne doit pas être supprimée, puis mets à jour sa table de routage et remplace l'ancienne route par la nouvelle.

V.4 Exemples de maintenance de route

Soit l'exemple illustratif de la figure III.7 ; Le nœud 2 voulant transmettre de l'information au nœud 13. La phase de découverte de route a été déjà faite et le nœud source commence à envoyer des paquets de données à travers le chemin indiqué dans la Figure III.7.a. Quand un lien, reliant le nœud 5 avec le nœud qui le suit dans le chemin de routage, c'est-à-dire le nœud7, devient défaillant, le nœud 5 diffuse un paquet RERR « *Route ERRor*», avec une valeur de numéro de séquence égale à l'ancienne valeur du paquet RREP incrémentée de un, et une valeur infinie de la distance (Figure III.7.b). Le paquet d'erreur est diffusé aux voisins actifs, jusqu'à ce qu'il arrive à la source, le nœud 2. Une fois le paquet est reçu, la source peut initier le processus de la découverte de routes(Figure III.7.c).



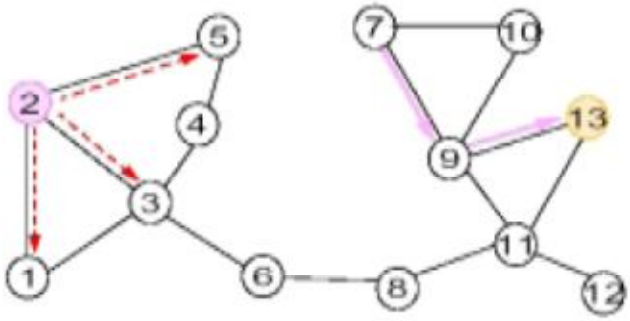


Route ERRor(RERR)

Dest.	N° séquence Dest.	Distance
13	1	4,3,2,1

Source	Dest.	N° séquence Dest.	Distance
2	13	2	INF

a. *Le chemin d'acheminement de paquets* **b.** *La diffusion du RERR par le nœud 5*



Route_Request(RREQ) :

RREQ_id	Source	N° séquence Source	Dest.	N° séquence Dest.	Distance
1234	2	50	13	2	0

c.*Initialisation du processus de découverte de route*

Figure III-7:processus de maintenance de route dans AODV[29]

Conclusion

Dans ce chapitre nous avons mené une étude détaillée du protocole AODV ce qui nous a permis de synthétiser ces avantages et inconvénients.
AODV ne met à jour les tables de routage que lorsque c'est nécessaire, et économise ainsi la bande passante. Néanmoins, il s'avère moins rapide puisqu'il faut mettre à jour la table de routage avant de communiquer.

Dans le but de réduire la taille des paquets et des tables de routage et ainsi d'économiser la bande passante et avoir un gain au niveau d'énergie, nous avons modifié le contenu de la table de routage en éliminant le champ destination de cette dernière.

Dans les réseaux de capteurs sans fil, tous les nœuds envoient l'ensemble des informations collectées vers un seul nœud destination appelé le nœud puits ou Sink.
L'adresse de ce nœud est connue au préalable. Donc, il n'est pas nécessaire de la préciser à chaque transmission.

En effet, la réduction du volume d'informations échangées permet de réduire l'utilisation du canal radio très gourmand en consommation énergétique.

Dans le chapitre suivant, nous allons présenter le système d'exploitation minime TinyOS destiné aux réseaux de capteurs sans fil et également du simulateur de TinyOS TOSSIM .

CHAPITRE IV:

TINYOS ET TOSSIM

Introduction

TinyOS est un système d'exploitation intégré, modulaire, destiné aux réseaux de capteurs. Cette plate-forme logicielle open-source est composée d'une série d'outils développés par l'Université de Berkeley. En effet, TinyOS est le plus répandu des OS (Operating System) pour les réseaux de capteurs sans fil. Un grand nombre de ces groupes de recherche ou entreprises participent activement au développement de cet OS en fournissant de nouveaux modules, de nouvelles applications, etc. La librairie TinyOS comprend les protocoles réseaux, les drivers pour capteurs et les outils d'acquisition de données.

Dans le système TinyOS, les librairies et les applications sont écrits en nesC, un nouveau langage pour le développement d'applications orientées composants. Le langage nesC est principalement dédié aux systèmes embarqués comme les réseaux de capteurs. NesC a une syntaxe proche du langage C mais supporte le modèle concurrent de TinyOS ainsi que des mécanismes pour la structuration, le nommage et l'assemblage de composants logiciels en des systèmes réseaux embarqués fiables. L'objectif principal est de permettre aux concepteurs d'applications de construire des composants qui peuvent être composés rapidement en des systèmes complets, concurrents, tout en permettant une vérification profonde à la compilation.

TinyOS définit un nombre important de concepts qui sont exprimés dans nesC. Premièrement, les applications nesC sont construites à partir de composants ayant des interfaces bidirectionnelles bien définies. Deuxièmement, nesC définit un modèle de concurrence basé sur les notions de tâches, les "handlers" d'événements matériels.

Dans ce qui suit nous allons voir plus en détails le système d'exploitation TinyOS, le langage de programmation NesC ainsi que la description du simulateur TOSSIM.

Chapitre IV: TinyOS et TOSSIM

I. Le système d'exploitation TinyOS

Figure IV-1 : Cigle du système d'exploitation TinyOS[21].

TinyOS est un système d'exploitation open source conçu pour les réseaux de capteurs par l'université américaine de BERKELEY**[39][40][41].** Le caractère open source permet à ce système d'être régulièrement enrichi par une multitude d'utilisateurs. Sa conception a été entièrement réalisée en NesC, qui est un langage orienté composant syntaxiquement proche du C.

Pour autant, la bibliothèque de composants de TinyOS est particulièrement complète puisqu'on y retrouve des protocoles réseaux, des pilotes de capteurs et des outils d'acquisition de données. Un programme s'exécutant sur TinyOS est constitué d'une sélection de composants systèmes et de composants développés spécifiquement pour l'application à laquelle il sera destiné (mesure de température, taux d'humidité…).

I.1 L'utilité d'un nouvel OS pour les "motes"[2]

Les OS classiques sont généralement conçus pour un usage générique. Ils sont ainsi conçus en supposant une disponibilité sans limite des ressources. Leur objectif est la facilité d'usage, la rapidité et efficacité. Parmi leurs caractéristiques, on peut citer:

- Architecture Multi-thread, ce qui nécessite une capacité mémoire importante
- Modèle E/S
- Séparation entre espace noyau et utilisateur
- Pas de contraintes d'énergie
- Ressources disponibles

De ce fait, les OS classiques ne sont pas appropriés aux "motes" (nœuds capteurs), vus que ces derniers sont caractérisés par :

- Ressources énergétiques basses
- Mémoire limitée
- CPU lente
- Petite taille

- Communication radio :
  - Bande-passante faible
  - Portée radio courte

Propriétés de l'OS souhaité pour les "motes"

- Image mémoire petite
- Efficacité en calcul et consommation d'énergie
- La communication est fondamentale
- Construction efficace d'applications

I.2 La solution TinyOS

I.2.1 Caractéristiques de TinyOS[21][2][33]

TinyOS a été créé pour répondre aux caractéristiques et aux nécessités des RCSF telles que :

- **Taille réduite** : TinyOS a une empreinte mémoire très faible puisqu'il ne prend que 4 Ko de mémoire libre et 300 à 400 octets dans le cadre d'une distribution minimale.

- **Applications orientées composants**: Un programme s'exécutant sur TinyOS est constitué d'une sélection de composants qui peut être utilisée telle quelle ou bien adaptée à une application précise (mesure de température, du taux d'humidité, etc.). A cette fin, TinyOS fournit une réserve de composants systèmes utilisables au besoin. Parmi les plus fréquents, on cite ceux concernant les entrée/sorties, les timers, etc. TinyOS définit les composants impliqués dans la création d'une application ainsi que la manière dont ils sont reliés. Cette liaison entre composants repose sur la notion d'interface.

- **Programmation orienté événement**: Le plus gros avantage de TinyOS est qu'il est basé sur un fonctionnement événementiel, c'est à dire qu'il ne devient actif qu'à l'apparition de certains événements. Le reste du temps, le capteur se trouve en état de veille afin de garantir une durée de vie maximale aux faibles ressources énergétiques du capteur.
  Ce fonctionnement événementiel (event-driven) s'oppose au fonctionnement dit temporel (time-driven) où les actions du système sont gérées par une horloge donnée.

- **Non Préemptif:** Le caractère préemptif d'un système d'exploitation précise si celui-ci permet l'interruption d'une tâche en cours. TinyOS ne gère pas ce mécanisme de préemption entre les tâches. Autrement dit, une tâche ne peut pas interrompre une autre tâche. Ce mode de fonctionnement permet de bannir les opérations pouvant bloquer le système et donne la priorité aux interruptions matérielles (i.e. les événements peuvent interrompre les tâches).

  TinyOS est donc basé sur une structure à deux niveaux de planification :

  - *Les événements* : ils sont utilisés pour réaliser des processus urgents et courts.
  - *Les tâches* : les tâches sont pensées pour réaliser une plus grande quantité de traitements et elles ne sont pas critiques dans le temps. Les tâches sont exécutées complètement.

- **Pas de temps réel** : Lorsqu'un système est dit « temps réel » celui-ci gère des niveaux de priorité dans ses tâches permettant de respecter des échéances données par son environnement. TinyOS n'est pas prévu pour avoir un fonctionnement temps réel.

I.2.2 Aperçus général de TinyOS

- Système d'exploitation pour réseaux de capteurs embarqués
- Ensemble de composants logiciels qui peuvent être reliés ensemble en un seul exécutable sur un mote

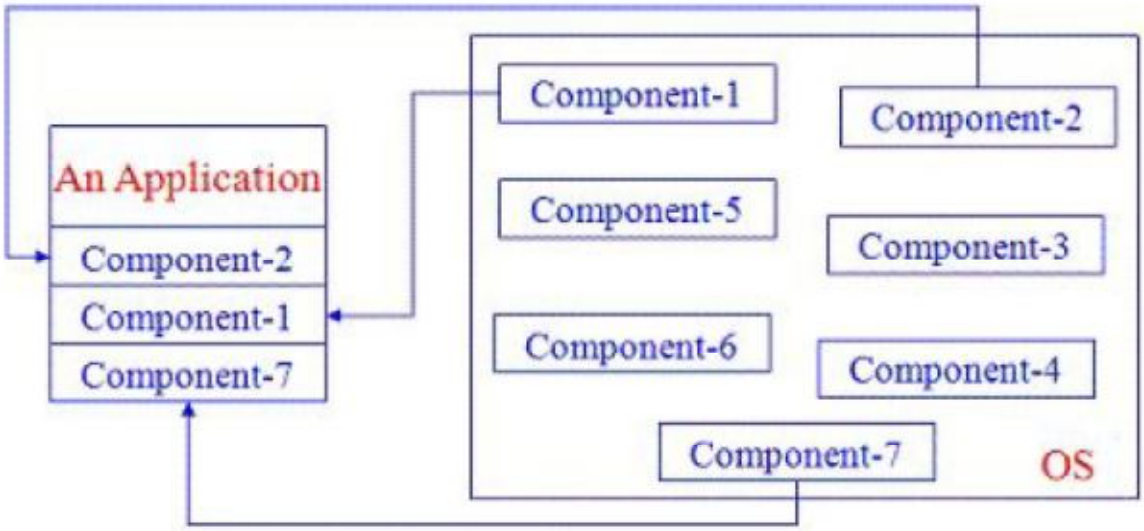


Figure IV.2 : TinyOS : un ensemble de composants logiciels [2]

Fonctions minimales

- Deux threads: tâches et handlers d'événements matériels
- Pas de gestion de la mémoire...

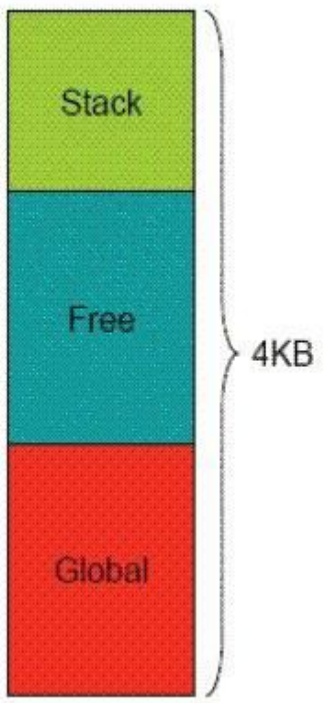
I.3 Equipements supportés par TinyOS

TinyOS peut être implémenté sur un PC capteur (ATMega8, AVRMote, Mica, Rene2, MSP430, Telos). Au-delà de cette liste, il est possible d'implémenter tout type de plateforme embarquée physique en redéveloppant les bibliothèques nécessaires à la prise en compte des entrées sorties nécessaires.

I.4. Allocation de la mémoire

Il est important de préciser de quelle façon un système d'exploitation aborde la gestion de la mémoire. C'est encore plus significatif lorsque ce système travaille dans un espace restreint.

TinyOS occupe un espace mémoire faible répartie en :



- *pile* : sert de mémoire temporaire au fonctionnement du système notamment pour l'empilement et le dépilement des variables locales ;
- *La mémoire libre* : pour le reste du stockage temporaire.
- *Les variables globales* : réservent un espace mémoire pour le stockage de valeurs pouvant être accessible depuis des applications différentes ;

La gestion de la mémoire possède de plus quelques propriétés. Ainsi, il n'y a pas d'allocation dynamique de mémoire et pas de pointeurs de fonctions. Bien sûr cela simplifie grandement l'implémentation. Par ailleurs, il n'existe pas de mécanisme de protection de la mémoire sous TinyOS ce qui rend le système particulièrement vulnérable aux corruptions de la mémoire.

Figure IV.3 Organisation de la mémoire dans TinyOS

I.5. Allocation de ressources

Le choix d'un ordonnanceur détermine le fonctionnement global du système et le dotera de propriétés précises telles que la capacité à fonctionner en événementiel.

L'ordonnanceur TinyOS se compose de :

- 2 niveaux de priorités (bas pour les tâches, haut pour les événements).
- 1 file d'attente FIFO (disposant une capacité de 7Ko).

A l'appel d'une tâche, celle-ci va prendre place dans la FIFO en fonction de sa priorité (plus elle est grande, plus le placement est proche de la sortie). Dans le cas où la file d'attente est pleine, la tâche dont la priorité est la plus faible est enlevée de la file FIFO. Lorsque la file est vide, le système met en veille le dispositif jusqu'au lancement de la prochaine interruption.

I.6 Modèle d'exécution de TinyOS

Pour maintenir une grande efficacité requise par les réseaux de capteurs, TinyOS utilise une programmation par événement. L'implémentation des composants de TinyOS s'effectue en déclarant des tâches, des commandes ou des événements.

TinyOS dispose de 2 types d'entités ordonnançables : interruptions (événements) et tâches **[34]**.

I.6.1 Programmation par événement

Dans un système basé sur la programmation par événement, chaque exécution est partagée entre les différentes tâches de traitement. Dans TinyOS, chaque module est désigné pour fonctionner en attendant continuellement de répondre aux événements inattendus. Quand un événement est signalé, le traitement correspondant est exécuté. Une fois totalement terminé, la main est redonnée au système pour continuer sa tâche antérieure.

En plus de l'efficacité de l'allocation du CPU, la programmation par événement permet d'obtenir des opérations économiques en énergie. Il est très important aussi pour la consommation d'énergie que les applications signalent la fin de leurs événements et l'utilisation du CPU. Dans TinyOS, les taches associées avec un événement sont exécutées très rapidement après leurs signalisations. Une fois terminé, le CPU entre en veille en attendant une nouvelle réception d'événement.

I.6.2 Tâches

Un des facteurs limitant la programmation par événement est la longue exécution des tâches qui peut interrompre d'autres programmes importants. Si l'exécution d'un événement ne finit jamais, toutes les autres fonctions vont être interrompues. Pour éviter ce problème, TinyOS fourni un mécanisme d'exécution appelé *tâche*. Une tâche est un bout de programme qui s'exécute jusqu'à la fin sans interférer avec les autres événements. Les tâches sont utilisées pour effectuer la plupart des blocs d'instructions d'une application.

A l'appel d'une tâche, celle-ci va prendre place dans une file d'attente de type FIFO mais elle ne sera exécutée que lorsque il n'y a plus d'événements. En plus les tâches peuvent être interrompues à tout moment par des événements.

D'autre part, il n'y a pas de mécanisme de préemption entre les tâches et une tache activée s'exécute en entier. Ce mode de fonctionnement permet de bannir les opérations pouvant bloquer le système (inter blocage, famine, …). Par ailleurs, lorsque la file d'attente des taches est vide, le système d'exploitation met en veille le dispositif jusqu'au lancement de la prochaine interruption (on retrouve le fonctionnement Event-driven).

II. Le langage de programmation NesC

II.1 Présentation

Le système d'exploitation TinyOS 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. Chaque composant correspond à un élément matériel (LEDs, timer, ADC …) et peut être réutilisé dans différentes applications.

Un exemple d'architecture est donné en figure suivante :

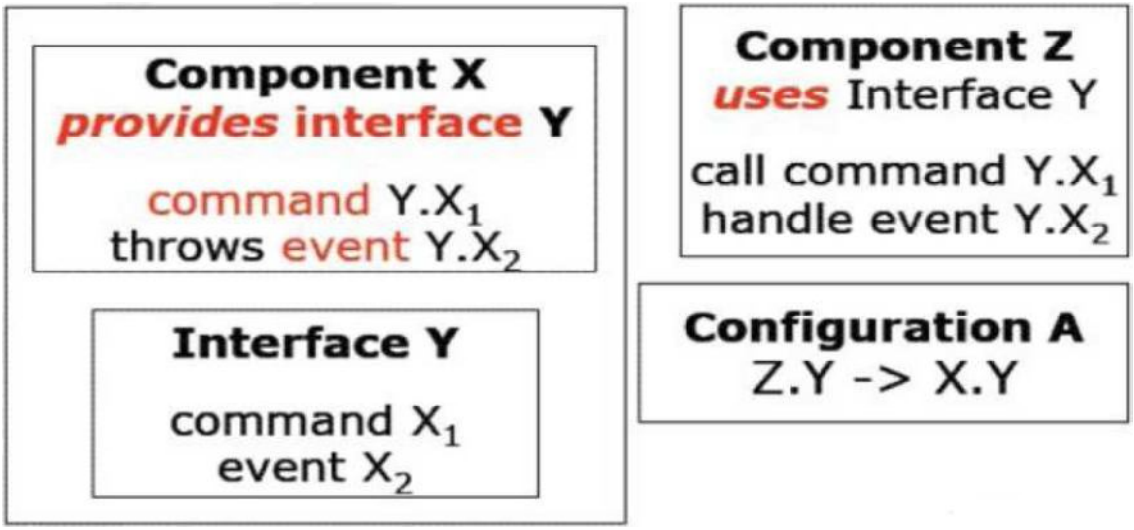


Figure IV.4 :Architecture générale d'une application NesC[21].

Les composants NesC présentent des similarités avec les objets. Les états sont encapsulés et on peut y accéder par des interfaces. En NesC, l'ensemble des composants et leurs interactions est fixé 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.

II.2 Concepts du langage NesC

L'unité de code de base de NesC est le composant "component", qui a comme rôle :

- Exécution des Commandes
- Lancement des Events

Un composant implémente des interfaces utilisées par d'autres composants pour communiquer avec ce composant.

Deux types de composants existent

- Les modules, qui mettent en application des spécifications d'un composant ;
- Les configurations, qui se chargent d'unir différents composants en fonction des interfaces (commandes ou évènements).

II.3 Les fichiers dans NesC [28]

Les fichiers de NesC sont classés en trois types : Interfaces, modules et configurations.

- *Interface* : Ce type de fichiers déclare les services fournis et ceux qui seront utilisés. Ces fichiers se trouvent dans le répertoire /tos/interface. (Exemple : StdControl.nc).
- *Module :* Le type Module contient le code de l'application, en mettant en œuvre une ou plusieurs interfaces (Exemple : BlinkM.nc).

- *Configuration :* Dans ces fichiers, on déclare la manière d'unir les différents composants (Exemple : Blink.nc).

II.3.1 Architecture des fichiers Nesc

Dans cette section nous allons voir en détail l'architecture des fichiers NesC définis précédemment **[19].**

- **Interface** : Les interfaces définissent les interactions entre deux composants. Elles spécifient un ensemble de fonctions à implémenter par les composants fournisseurs de l'interface (commands), et un ensemble à implémenter par les composants utilisateurs de l'interface (events).
  - Les « commands » font typiquement des appels du haut vers le bas (des composants applicatifs vers les composants plus proches du matériel).
  - Alors que les « events » remontent les signaux du bas vers le haut.

Exemple d'interface

```
Interface SendMsg{

command result_t send (uint16_t address, uint8_t length, TOS_MsgPtr msg) ;

event result_t sendDone(TOS_MsgPtr msg, result_t success);

}
```

Exemple : L'Appel d'une commande et le signaled'un événement.

- Pour appeler une commande**,** il faut utiliser le mot-clé call. Par exemple**:**

```
call Send.send (1, sizeof(Msg), &msg1) ;
```

- Pour signaler un événement, il faut utiliser le mot-clé signal. Par exemple:

```
signal Send.sendDone(&msg1, SUCCESS);
```

- **Module**

  Cette partie du code est généralement plus étendue et c'est dans celle-ci que l'on programme réellement le comportement qu'on souhaite voir réalisé par l'application. Cette partie-là est à son tour divisée en trois sous-sections : Uses, Provides, Implementation.

Exemple

```
1 module MonAppliM {
2 provides {... }
3 uses {... }
4 }
5 implementation {... }
```

La première sous-section, « provides », indique au compilateur les interfaces que va fournir notre composant. Par exemple, si notre composant est une application, on doit fournir au moins l'interface StdControl.

```
1 provides {
2 interface interfacequenousfournissons;
3 }
```

La sous-section « uses » informe le compilateur que nous allons faire usage d'une interface (on pourra donc effectuer des appels aux méthodes de cette interface). Finalement, utiliser une interface oblige implicitement à gérer les évènements pouvant se produire du fait d'avoir utilisé cette interface précise.

```
1 uses {
2 interface interfacequenousutilisons;
3 }
```

La sous-section « implementation », est celle qui contiendra toutes les méthodes nécessaires pour définir le comportement souhaité à notre composant ou à notre application. Cette sous-section doit contenir au moins:

- Les variables stockées dans la mémoire des capteurs que vont utiliser notre application ;
- Les fonctions mises en œuvre pour les interfaces fournis ;
- Les événements mis en œuvre venant des interfaces utilisées.

- **Configurations**

  C'est à ce niveau que l'on déclare les composants qui vont constituer l'application. Cette possibilité offerte par le langage permet de faire de la programmation modulaire et de réutiliser des composants préalablement définis. Deux composants peuvent être connectés (Wiring) forcement par la même interface.

Il existe trois possibilités de connexion (wiring statements) dans NesC:

composant1.interface -> composant2.interface (composant1 utilise l'interface fournit par composant2)

composant1.interface <- composant2.interface (composant1 fournit l'interface utilisée par composant2)
composant1.interface = composant2.interface (lien bidirectionnel)

Exemple :

```
configuration app { }
implementation {
uses c1, c2, c3;
c1 -> c2;
c2.out -> c3.triangle;
c3 <- c2.side;
}
```

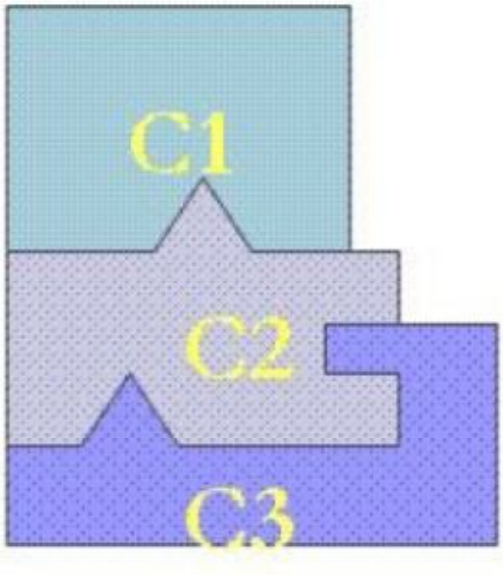


Figure IV.5 Illustration d'une configuration basée sur la connexion de 3 modules[2]

II.3.2 Types de fonctions en NesC

Dans NesC, les fonctions peuvent être de types très variés. Il y a d'abord les fonctions classiques qui ont la même sémantique et sont invoqué de la même manière qu'en C. Il y a aussi des types supplémentaires de fonctions : command, task et event **[35].** Les fonctions ((command)) sont principalement des fonctions qui sont exécutées de manière synchrone, c'est-à-dire que lorsqu'elles sont appelées, elles sont exécutées immédiatement. L'appel à ces fonctions est défini comme suit :

call interface . nomFonction

Les fonctions ((***task***)) sont exécutées dans l'application, en utilisant la même philosophie que les fils ou ((threads)).

L'instruction ***post interface.nomTache*** permet d'invoquer une tâche. Le programme appelant et la tâche s'exécute en parallèle.

Les fonctions ((***event***)) sont des fonctions qui sont appelées quand on détecte un signal dans le système. Ce type de fonctions est invoqué de la manière suivante :

signal interface . nomEvenement

II.3.3 Commands vs. Events vs. Tasks

Dans ce qui suit, nous présentons une comparaison entre les commandes, les évènements et les tâches.

Command

- ✓ Non bloquante i.e. prend les paramètres, commence le traitement et retourne dans l'application;
- ✓ Reporte le travail qui consomme du temps en postant une tâche ;
- ✓ Appelle des commandes sur d'autres composants.

Event

- ✓ Appeler des commandes, signaler d'autres événements, poster des tâches mais ne peut pas être signalé par des commandes ;
- ✓ Peut interrompre une tâche mais pas l'inverse.

Task

- ✓ Ordonnancement FIFO ;
- ✓ Non préemptive par une autre tâche, préemptive par un événement ;
- ✓ Utilisée pour réaliser un travail qui nécessite beaucoup de calculs.

- ✓ Peut-être postée par une "command" ou un "event".

Après avoir écrit l'application en NesC, on peut procéder à la compilation et la simulation de l'application **[19].**

II.4 Compilation d'une application NesC

Les fichiers de NesC portent l'extension .nc. Par ailleurs, le compilateur de NesC est appelé ncc. Pour effectuer la compilation, les fichiers vus dans la partie précédente, doivent se situer dans le même répertoire contenant aussi un makefile de la forme :

```
COMPONENT= nom de l'application
include ../Makerules
```

Ce Makefile permet de compiler le composant en spécifiant en paramètre la plateforme sur laquelle doit fonctionner l'application. Par exemple, pour un capteur de type mica2, la commande qui permet de compiler l'application est : « make mica2 ».

Le compilateur ncc offre aussi la possibilité de pouvoir compiler l'application pour l'utiliser sur TOSSIM, le simulateur de TinyOS. Dans ce cas-là, la commande sera : « make pc **»[25][38].** Cette commande génère un exécutable «main.exe» dans l'arborescence « /repertoire_courant/build/pc ».

La figure suivante montre le processus de compilation pour une application TinyOS écrite en NesC :

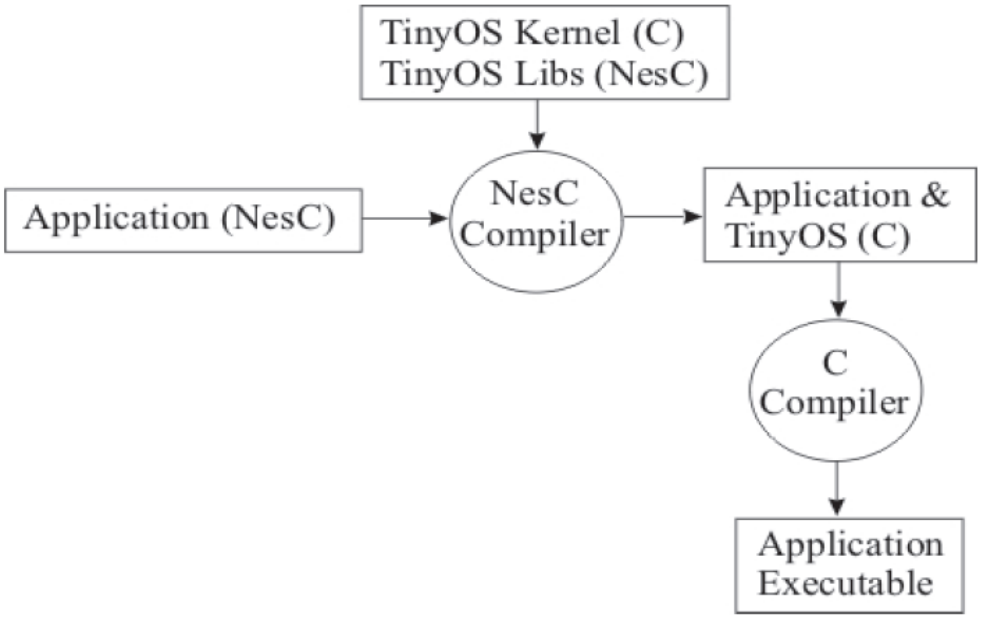


Figure IV.6 : Processus de compilation[21].

II.5 Exemple d'application

```
configuration Temperature {

}

implementation {

components Main, TemperatureM, LedsC, Temp,TimerC,GenericComm as Comm;

Main.StdControl ->TemperatureM.StdControl;

Main.StdControl ->TimerC;

Main.StdControl ->Comm;

TemperatureM.Timer ->TimerC.Timer[unique("Timer")];

TemperatureM.Leds ->LedsC;

TemperatureM.TempControl ->Temp.StdControl;

TemperatureM.ADC ->Temp;

TemperatureM.SendMsg ->Comm.SendMsg;

TemperatureM.ReceiveMsg ->Comm.ReceiveMsg;

}
```

Il est à noter que la configuration Main est obligatoirement présente dans la configuration décrivant l'ensemble de l'application car son rôle est de démarrer l'exécution de l'application.

L'application ci-dessus utilise le composant LedsC, ce qui permet de manipuler les leds du capteur.

```
module TemperatureM

{

//l'interface fournie

provides {

interface StdControl;

  }


        uses {

        interface Timer;

        interface Leds;

        interface StdControl as TempControl;

        interface SendMsg;

        interface ADC;

        interface ReceiveMsg;

        }

}

implementation {

...//code des redéfinitions des interfaces

}
```

III TOSSIM (TinyOS SIMulator)

TOSSIM est un simulateur d'évènements discrets pour les réseaux de capteurs **[36][37]** utilisant le système TinyOS (l'OS le plus utiliser pour les WSNs). Au lieu de compiler l'application directement dans le capteur, l'utilisateur peut le faire dans la framework

TOSSIM qui s'exécute sur un PC. Le principale but de TOSSIM est de créer une simulation très proche de ce qui ce passe dans ces réseaux dans le monde réel.

TOSSIM utilise une interface graphique écrite en Java, TinyViz, pour visualiser de manière intuitive le comportement de chaque capteur au sein du réseau.

Cette application est équipée par plusieurs API plugins qui permet d'ajouter plusieurs fonctions au simulateur comme par exemple contrôler les entrées radio ou bien suivre la dépense d'énergie en utilisant un autre simulateur qui s'appelle Power TOSSIM.

III.1 Topologie d'un réseau avec TinyOS

Le simulateur TOSSIM de TinyOS permet de modéliser deux différents types de topologies pour un réseau de capteurs, se sont deux modèles de radios pour la communication

TOSSIM fourni:

- La première topologie, c'est le modèle par défaut « *simple* », dans laquelle chaque nœud peut communiquer avec n'importe quel autre nœud dans le réseau. Les paquets sont transmis dans le réseau sans aucune erreur et ils sont reçus par chaque nœud. Cette topologie est trop théorique et ne reflète pas vraiment ce qui se passe en monde réel tel que l'influence de l'environnement sur la stabilité de la communication.

- Le deuxième type est le modèle « lossy ». Dans ce modèle les nœuds sont placés dans un graphe direct formé d'un couple (a, b) ce qui signifie qu'un paquet envoyé par le nœud a peut être reçu par le nœud b.

La topologie « lossy » est défini de la façon suivante :
<noeud id> : <noeud id> : <taux de bit d'erreur>

Le premier *noeud id* indique que le second *nœud id* peut recevoir un paquet transmis par le premier *nœud id*. Ceci représente un lien direct dans un graphe.
Le troisième paramètre est un nombre entre 0 et 1 et il exprime le taux d'erreur bit d'une liaison.

Chaque bit est considéré de manière indépendante. Par exemple, une valeur de 0,01 signifie que chaque bit transmis a une chance de 1% d'être perdue, tandis que 1,0 signifie que chaque bit sera perdue, et de 0,0 signifie que le bit sera transmis sans erreur.

Le graph du modèle lossy peut être spécifié au démarrage de TOSSIM, il suffit de préciser le modèle lossy avec l'option –r =lossy .Si on veut spécifier un fichier de configuration de la topologie lossy en entrée, il nous suffit de mettre –rf= fichier.nss. Le fichier « file.nss » a le

format suivant:

<mote ID>:<mote ID>: taux d'erreur binaire

<mote ID>:<mote ID>: taux d'erreur binaire

<mote ID>:<mote ID>: taux d'erreur binaire

- Un exemple de topologiesde réseau est fourni dans le repertoire « /opt/tinyos-1.x/contrib/bvr/topologies ».

III.2 Avantages

- TOSSIM permet de simuler fidèlement le comportement d'un réseau de capteurs puisque le code est écrit en respectant l'implémentation de TinyOS. Donc une fois la simulation est terminée, le code peut être exécuté directement dans un capteur utilisant le systèmeTinyOS avec un petit changement.

- Il prend en charge la couche MAC, l'encodage de données, le timing, et les acquittements de synchronisation.

- Il permet également l'affichage des évènements et des messages de débuggage pour chaque capteur mais aussi simultanément pour l'ensemble de capteurs.

- A cela se rajoute l'interface graphique TinyViz qui permet la visualisation des échanges radios, d'avoir une vue globale de l'activité du réseau étudié.

II.3 Inconvénients

- TOSSIM fournie une abstraction de certains phénomènes du monde réel.
  - *La Radio:* il ne modélise pas la propagation dans la radio, à la place, il fournit une abstraction de l'indépendance directe de taux d'erreurs entre deux nœuds.

  - *Puissance/energie:* il ajoute des annotations aux composants qui consomment de l'énergie pour fournir des informations concernant le changement de l'état énergétique.

- Il exige l'exécution du même code pour tous les nœuds. Donc il ne distingue pas les différents types de capteurs.

- Manque de souplesse et d'extensibilité puisqu'il est lié à une interface de visualisation conçue séparément.

CONCLUSION

Ce chapitre a été consacré à l'étude de notre environnement de travail. A travers les différents points essentiels que nous avons cités, nous avons abordé l'architecture du système d'exploitation TinyOS, son mode d'ordonnancement, la compilation des différentes applications écrites en NesC et également la description du simulateur TOSSIM sur lequel se déroulent ces dernières.

Nous allons passer à l'implémentation de toutes les étapes de notre étude et donner des résultats démonstratifs qui les justifient. Le prochain chapitre sera dédié à l'implémentation des protocoles de routage AODV et LEACH afin d'obtenir une comparaison entre ces deux protocoles.

CHAPITRE V :

IMPLEMENTATION DE LEACH ET AODV SUR TOSSIM

Introduction

Tel qu'on l'a montré au cours du chapitre précédent, l'un des objectifs principaux de notretravail est l'adaptation du protocole de routage AODV aux réseaux de capteurs sans fil.

De ce fait, nous avons implémenté dans TinyOS un nouveau protocole AODV-adapté quiest une amélioration du protocole AODV afin de mieux prendre en charge la bande passante et la consommation énergétique.

L'autre objectif de ce chapitre estd'évaluer les performances des deux protocoles LEACH et AODV- adapté. Le choix de LEACH comme référence est dû au fait que ce dernier est très utilisé sur les réseaux de capteurs actuellement.

Pour cela, nous allons commencer par déterminer l'ensemble des structures de données qui seront utilisées pendant la phase d'implémentation des tables de routagesainsi qu'un ensemble de commandes utiliséespendant la phase de simulation. Par la suite nous allons passer à une simulation des deux protocoles afin de préciser l'amélioration d'AODV- adapté.

Nous terminerons ce chapitre par une petite comparaison entre les deux protocoles, AODV-adapté et LEACH en appuyant sur les résultats de simulation.

I. Implémentation du protocole AODV

I.1 Structure des données

Afin d'implémenter le protocole AODV, les informations suivantes sont requises :

I.1.1 Structure des messages RREQ,RREP,RERR :

La figure V.1 montre la structure du message RREQ envoyé par le nœud source.
Le nœud source maintient une seule valeur de RreqID. Pour chaque message RREQ envoyé en broadcast,la valeur de RreqID est augmenté de un. Le champ *metric* représente le nombre de sauts nécessaire pour atteindre la destination.

Le champ 'destSeq' (destination sequencenumber) est initialisé à zéro, sa valeur serra incrémenté à chaque envoie du message RREQ en broadcast par le nœud source.

Pour le message RREQ, nous avons modifié la structure du message original et proposé la structure de la figure V.1 dans laquelle nous avons supprimé l'adresse de destination.

```
AODV_Rreq_Msg {
uint8_tsrc; // Adresse du noeud source générant la requête Rreq
uint16_trreqID; // route request ID
uint16_tsrcSeq; // Le numéro de sequence du noeud source
uint16_tdestSeq; //Dernier numéro de séquence reçu par le nœud source vers la destination
uint8_t metric[1]; // la sélection du nombre de sauts
};
```

Figure V.1 : Structure du message RREQ

La figure V.2 montre la structure du message RREP. La destination initialise le message RREP et transmis ce paquet vers le nœud suivant (next hop) afin d'atteindre le nœud source.

Pour le message RREP, nous avons modifié la structure du message original et proposé la structure de la figure V.2 dans laquelle nous avons supprimé également l'adresse de destination.

```
AODV_Rreply_Msg {
uint8_tsrc;                 // l'adresse source de  RREQ
uint16_tdestSeq; // le numéro de séquence de la destination
                    uint8_t metric[1]; // la sélection du nombre de sauts
                   };
```

Figure V.2: Structure de Route Reply message

La figure V.3 montre la structure de message RERR, ce message est envoyé uniquement lors de détection d'erreurs lors de transmission des messages:

```
AODV_Rerr_Msg {
uint8_tdest;                // l'adresse de destination
uint16_tdestSeq;            //le dernier numéro de séquence reçu par la destination
};
```

Figure V.3: Structure de Route Error message

I.1.2 Structure d'un paquet AODV

La structure d'un paquet AODV est représentée dans la figure suivante:

```
Tos_Msg {
uint16_taddr;          // adresse de next hop (Broadcast / unicast address)
uint8_t type;           // message ID du paquet
uint8_t group;          // les groupes id pour les motes. Le groupe par defautest 0x7d
uint8_t length;         // length of the packet payload
SHop_MsgSHopmsg; // AODV single hop message
MESSAGESdatamsg;
        };

SHop_Msg {
uint8_tsrc;           // l'adresse source du mote
uint8_tseq;           // le numéro de séquence du paquet
};
```

Figure V.4: Packet structure of AODV messages

Avec le champ message représente les différentes requêtes envoyées dans le réseau :

MESSAGES ={AODV_Rreq_Msg , AODV_Rreply_Msg , AODV_Rerr_Msg}

La structure d'un paquet de données AODV est présenté dans la figure V.5 suivante:

```
Tos_Msg {
uint16_taddr;               // address of the next hop (Broadcast / unicast address)
uint8_t type;                // the message id of the packet
uint8_t group;               // the group id of the motes. Default group id is 0x7d
uint8_t length;              // length of the packet payload
SHop_MsgSHopmsg;
AODV_Msgaodv;          // AODV header
 MESSAGES Appdatamsg; // data payload of the application
 };


SHop_Msg {
uint8_tsrc;          // Source address of the mote
uint8_tseq;          // Sequence number of the packet
  };


AODV_Msg {
MHop_Headermhop;
uint8_tseq;         // AODV sequence number
uint8_tttl;          // time to live
  };
```

```
MHop_Header {
uint8_tsrc;          // Source address of the packet originator
uint_tdest;          // Destination address for the packet
uint8_t app;         // message ID of the packet
uint8_t length;      // length of AODV application data payload
};
```

Figure V.5: Structure d'un paquet de donnée AODV

I.1.3Structure de Route Cache table et Route table

Lors de notre développement du protocole AODV la structure de Route_Cache_Tablea était largement utilisée, nous nous somme basées sur elle pour effectuer nos modifications.

La table Route_Cache_Table est mise à jour par chaque nœud source recevant un paquet de type RouteRequest.
La table de routage Route_Table est mise à jour par chaque nœud source recevant le paquet Route_Reply.

La structure des deux tables est décrite dans les deux figures V.6 et V.7

```
AODV_Route_Cache {
uint8_tsrc; // l'adresse source de message RREQ
uint8_tnextHop; // address of the next hop in reverse path to the originator
uint16_trreqID; // Request ID of RREQ message
uint16_tdestSeq; // Destination sequence number of RREQ message
uint8_tnumHops; // Hop count of RREQ message from the originator - metric
};
```

Figure V.6: Structure de Route Cache Table

```
AODV_Route_Table{uint8_t src; // the source address in the RREQ message
uint8_tnextHop; // address of the next hop to the destination
uint16_tdestSeq; // Destination sequence number of RREP message
uint8_tnumHops; // Hop count of RREP from the base station
};
```

FigureV.7: Structure de Route Table

I.2 Liste des fichiers sources

L'implémentation du protocole de routage AODV nécessite l'utilisation des fichiers sources suivants situant dans le répertoire « contrib/hsn/tos/lib »

AODV.nc	- Fichier de configuration AODV
AODV_Core.nc	- Implémentation de l'algorithme AODV
AODV_PacketForwarder.nc	- composant qui envoie et traite les paquets des données envoyé par l'aodv
SingleHopManager.nc	- Configuration des fichiers pour pour l'envoie/reception des packets
SingleHopManagerM.nc	- Implémentation des paquets send/receive
AODV.h	-Définit les structures de données pour la table de routage cache et la table de routage Route-Table
WSN_Messages.h	- Définit les formats des paquets AODV

Remarque:

La modification du nœud Sink peut s'effectuer au niveau du fichier AODV.h comme suit :

#define AODV_ROOT_NODE 0 // 0 représente le nœud puits.

I.3Variable d'environnement dbg

Chaque message est assigné à un ou plusieurs debug flag. Quand la simulation démarre, le simulateur lit les variables d'environnement pour déterminer le debug flag correspondant au message désiré.

Par défaut tous les debug message sont des sorties.

Le debug flag est définit dans le fichier d'entête situant dans le répertoire ***tos\types\dbg_modes.h (explicité dans annexe B)***

Les différents debug flag existant sont :

ALL : permettre tous les messages disponibles
BOOT : Simulation boot et StdControl
CLOCK :l'horloge materiel
LED : Mote leds
ROUTE : routage systèmes
AM : Active les messages transmission/reception
PACKET : La transmission/reception au niveau paquet.
POWER : Niveau d'énergie au niveau hardware
RADIO : Low-level radio operations: bits and bytes
ADC : ADC
USR1 : mode sortie utilisateur 1
USR2 : mode sortie utilisateur 2
USR3 : mode sortie utilisateur 3
TEMP :pour l'utilisationtemporaire

Pour obtenir la liste des dbg disponibles ainsi les différentes options nous utilisons la commande suivante

Build/pc/main.exe –help

Les résultats obtenus sont décrit dans la figure V.8 suivante :

```
root@ferrat:/opt/tinyos-1.x/contrib/hsn/apps/TraceRouteTestAODV# build/pc/main.e
xe --help
Usage: build/pc/main.exe [options] num_nodes
  [options] are:
  -h, --help        Display this message.
  -gui              pauses simulation waiting for GUI to connect
  -nodbgout      only send dbg messages to GUI, not to stdout
  -a=<model>        specifies ADC model (generic is default)
                    options: generic random
  -b=<sec>          motes boot over first <sec> seconds (default 10)
  -ef=<file>        use <file> for eeprom; otherwise anonymous file is used
  -l=<scale>        run sim at <scale> times real time (fp constant)
  -r=<model>        specifies a radio model (simple is default)
                    options: simple lossy
  -rf=<file>        specifies file for lossy mode (lossy.nss is default)
                    implicitly selects lossy model
  -s=<num>          only boot <num> of nodes
  -t=<sec>          run simulation for <sec> virtual seconds
  -seed=<seed>      use random seed <seed>
  -p                do power profiling
  -cpuprof          do cpu profiling (only if compiled with cilly!)
  num_nodes         number of nodes to simulate

Known dbg modes: all, boot, clock, task, sched, sensor, led, crypto, route, am,
crc, packet, encode, radio, logger, adc, i2c, uart, prog, sounder, time, power,
sim, queue, simradio, hardware, simmem, usr1, usr2, usr3, temp, error, none
```

Figures V.8 : différentes options et modes dbg

II Simulation du protocole AODV-adapté

II.1 La table de routage d'AODV :

Pour pouvoir afficher le contenu de la table de routage dans TOSSIM, nous utilisons la variable d'environnement « route ».

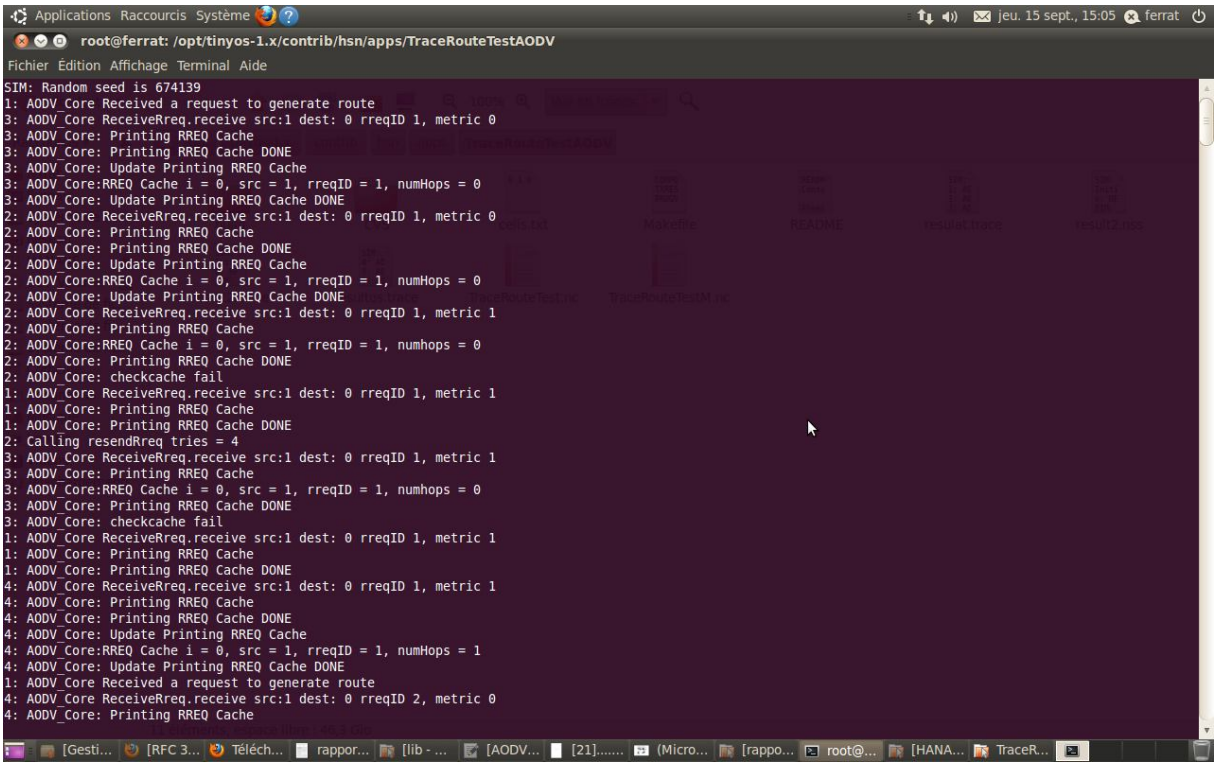
Compilation et exécution :

$**make pc** /* compiler l'application sur pc pour la tester avec le simulateur TOSSIM celagénèrera un dossier build/pc/ dans lequel se trouve notre main.exe */

$**export DBG=route** //appel à la variable d'environnement route

$**./build/pc/main.exe 6** //lancer la simulation en exécutant le main.exe se trouvant dans le dossier build/pc

Les résultats obtenus sont représentés dans la figure V.9 suivante :



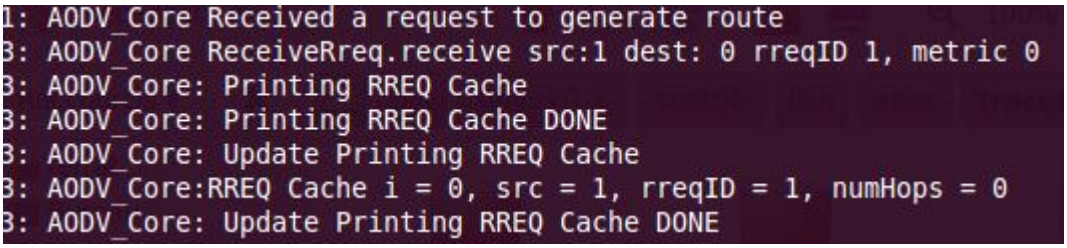


Figure V.9: le contenu de la table de routage AODV

La précision à chaque fois de l'adresse destination augmente la consommation d'énergie en bande passante. Les messages échangés entre les nœuds sont envoyés sous forme de bits d'une manière séquentielle par un canal radio délimité par la fréquence de la bande passante.

L'augmentation de nombre de bits envoyés nécessite une augmentation au niveau de la consommation et puisque le champ adresse destination est sur 32 bits, cela veut dire 32 bits en plus, c'est de l'énergie perdue.

II.2 La solution envisagée :le protocole AODV-adapté

Cette solution nous permet de réduire la taille de la table de routage :

Scénario élaboré :

Le scénario est le suivant :

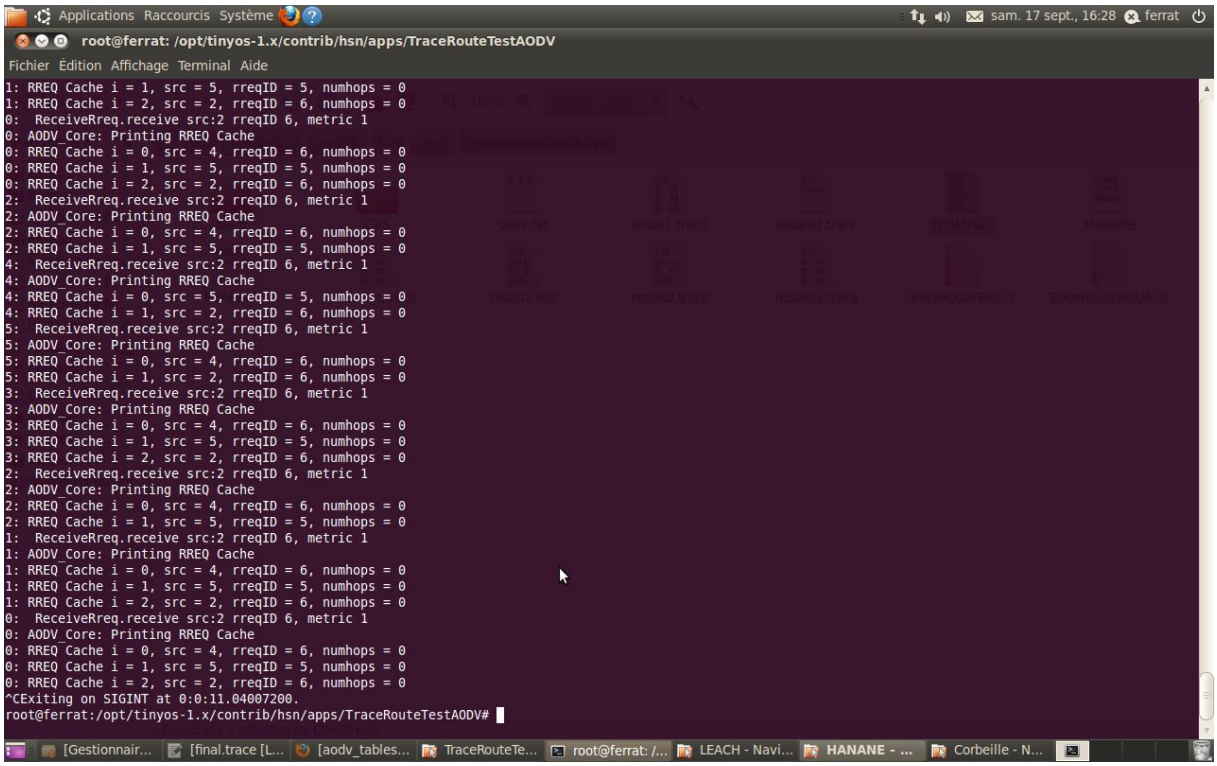
Nous disposons de 6 nœuds capteur à simuler dans le simulateur Tossim.

En activantla simulation du protocole AODV, Chacun des nœuds va envoyer l'ensemble des informations collectées au nœud puits.

Etapes de simulation :

1- On se positionne sur le répertoire contrib/hsn/apps/TraceRouteTestAODV comme suit : $cd /opt/tinyos-1.x/ contrib/hsn/apps/TraceRouteTestAODV
2- $ make pc
3- $ export dbg=route
4- ./build/pc/main.exe **6**

Nous aurons les résultats décrits dans la figure suivante :



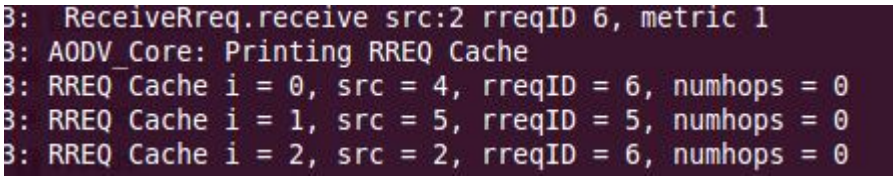
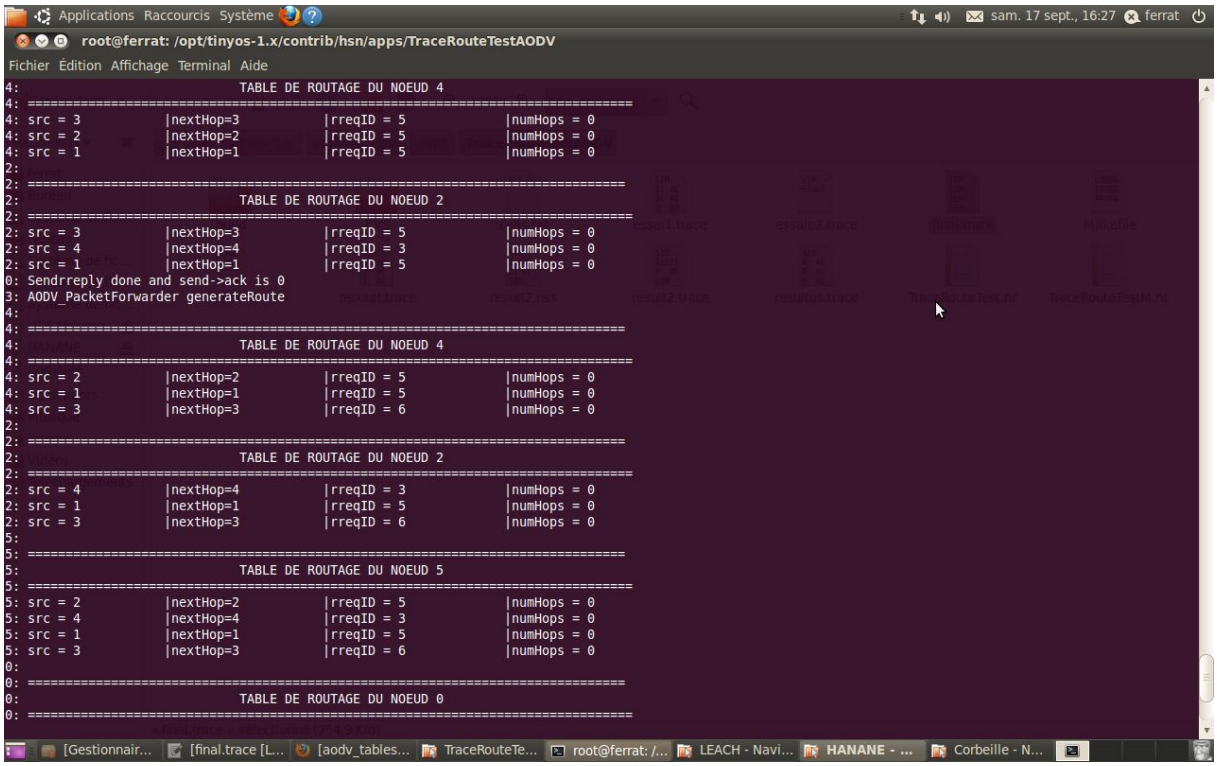


Figure V.10 : le contenue de la table de routage AODV-adapté

Pour la bonne visibilité du contenu de la table de routage, nous avons mis au point une structure qui nous permet de mieux observer le contenu.

Les étapes de simulation sont les même mais uniquement nous utilisons la variable d'environnement usr3(qui un output). Les résultats son décrit dans la figure qui suit :

$ export DBG=usr3



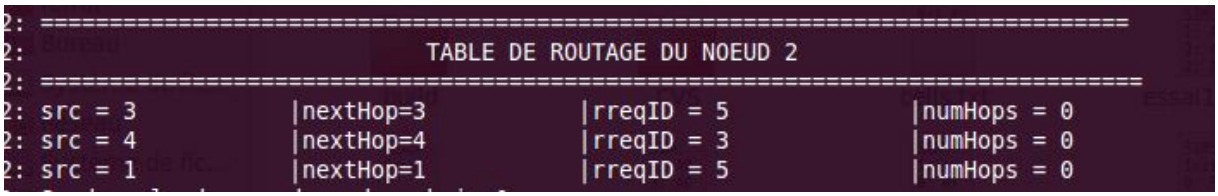


Figure V.11 : la table de routage AODV-adapté

Remarque : les résultats obtenus précédemment peuvent être récupérer dans un fichier trace en tapant la commande :

./build/pc/main.exe –t=60 10 >resultat.trace avec –t est le temps de simulation en seconde.

III. Implémentation du protocole LEACH :

Dans cette section, nous donnons la description détaillée du schéma d'implémentation du protocole LEACH. Nous décrivons les structures de données ainsi que les principaux commandes et événements nécessaires pour l'implémentation de ce dernier.

Nous citons les principales notations qui seront utilisées tout au long de la description de ce schéma.

Notation	Description
PT	Noeud puits
CHi	Noeud cluster Head i.
MBRi	Noeud membre i.
Ni	Noeud i quelconque du réseau.
CHs	Ensemble de tous les noeuds CH.
MBR	Ensemble de tous les noeuds membres dans le réseau.
IDi	Identificateur d'un noeud i (i=PT, CH, MBR, N).
||	Opérateur de concaténation
R	Round courant
ADV	Diffusion d'un *message d'avertissement* contenant *l'identificateur du CH* à tous les nœuds non –CH
Info	Information transmise par le paquet (slots, donnée captée, etc.)

TABLEAU V-1 : Notations utilisées par le protocole LEACH.

1) **Puits** : Il annonce l'information contenant le nouveau round en envoyant un broadcast.

```
//le puits déclenche le nouveau round en broadcast

PT ⟶ broadcast ; IDpt || R || info
```

Algorithme du déclenchement du round par le nœud puits :

```
//Broadcast du paquet
DATA ⟶ ID =IDpt
DATA ⟶ Round= R
Send (Broadcast,DATA)
```

2) **Relai des annonces du déclenchement du round**: Les nœuds du réseau relayent le paquet reçu à travers tous le réseau.

Ni → broadcast; IDNi || R || Info

3) **CHi**: Après l'étape de leur élection, les CHs envoient aux autres nœuds l'information qui contient leurs nouveaux statuts.

CHi → broadcast : ADV

Algorithme de formation d'un cluster-head:

```
//Décision d'un nœud Ni de devenir CH à l'instant t
If ( n < Pi(t)){
ADV →ID =IDCHi
ADV →Round= R
Send (Broadcast, ADV)
}
```

4) **MEMBREi**: Les nœuds reçoivent les annonces des CHs. Ensuite, ils envoient l'information qui contient leur décision d'appartenance au CH choisi. Pendant son slot, un membre envoie à son CH la valeur captée dans le champ information.

MBRi → CH : IDMBR || R || Info

Algorithme de demande d'admission au groupe pour un CH :

```
//demanded'admission du groupe
DATA->ID=IDMBR ,
DATA->Round=R,
Send(IDCH, DATA)
```

5) **CHi**: Il envoie au nœud puits l'information contenant les données agrégées.

CH→ PT : IDCH || R || Info

6) **Puits** : Il vérifie l'intégrité de données. Dans le cas favorable, il accepte les données agrégées sinon il les refuse.

III. 1. Structures de données :

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

A) Le noeud puits :

```
typedefstruct PUITS

{uint16_t ID; //l'identificateur du puits qui correspond à tos_local_address=0

uint8_t round; //le round courant

floatprobability; //la probabilité que chaque nœud devienne  un CH

uint8_tDepth; //la puissance du signal (profondeur du noeud) dans le réseau

}PUITS;
```

B) Le nœud CH :

```
typedefstruct CLUSTER_HEAD{   uint16_t ID_CH; //l'identificateur de chaque CH qui correspond à tos_local_address uint16_t ID_MEMBRE; //l'identificateur du membre qui appartiendra à ce CH   uint8_t data_agre; //la donnée agrégée à envoyer au nœud puits   uint16_t SLOT_ATT; //le slot attribué à chaque membre uint16_t FREQ; //la fréquence avec laquelle un membre envoie sa donnée

}CLUSTER_HEAD;
```

C) **Le nœudMEMBRE :**

```
typedefstructMEMBRE{

uint16_t ID_MEMBRE;//l'identificateur de chaque membre qui correspond à tos_local_adress

uint16_t ID_CH; //l'identificateur du CH auquel appartiendra le noeud membre

uint8_tdonnee; //la donnée capturée

}MEMBRE;
```

III.2. Evénements et commandes : Nous citons ici les principaux événements utilisés pour l'implémentation de LEACH.

Evénement	Sortie	Fonction
LEACH_ReceiveMsg.receive(TOS_MsgPtrpmsg)	TOS_MsgPtr	Réception du round
ANNONCE_ReceiveMsg.receive(TOS_MsgPtrpmsg)	TOS_MsgPtr	Annonce du CH
ORGANISATION_ReceiveMsg.receive(TOS_MsgPtrpmsg)	TOS_MsgPtr	Formation de groupes
SLOT_ReceiveMsg.receive(TOS_MsgPtrpmsg)	TOS_MsgPtr	Réception des slots
Donnees_ReceiveMsg.receive(TOS_MsgPtrpmsg)	TOS_MsgPtr	Réception du CH desdonnées captées
AGGREGATION_ReceiveMsg.receive(TOS_MsgPtrp	TOS_MsgPtr	Réception du puits des résultats

msg)		d'agrégation
ReqRelayTimer.fired()	result_t	Relai des annonces du round
RoundTimer.fired()	result_t	Envoi du nouveau round par le nœud puits

TABLEAU.V-2 Evènements et commandes utilisés pour l'implémentation de LEACH

III.3. Déroulement

Dans ce qui suit, nous allons dérouler le fonctionnement du protocole LEACH en faisant appel à l'interface graphique TinyViz.

En ligne de commande :

```
#Allez au répertoire de l'application
root@ubuntu:/home/sabrina# cd /opt/tinyos-1.x/apps/LEACH

#On compile l'application, plateform=pc
root@ubuntu:/opt/tinyos-1.x/apps/LEACH# make pc

#activer le mode debugging mode de sortie utilisateur 2
root@ubuntu:/opt/tinyos-1.x/apps/LEACH# export DBG=usr2

#récupérer la trace d'exécution l'application dans le fichier result Durée de
l'excution60sec, nombre de nœuds 30
root@ubuntu:/opt/tinyos-1.x/apps/LEACH# ./build/pc/main.exe –t=60 30 >result.trace

#simulation sous tinyvizroot@ubuntu:/opt/tinyos-1.x/apps/LEACH# tinyviz –run
./build/pc/main.exe 30

#ou bien, on ouvre deux fenêtre de commandes une pour récupérer les traces de
l'exécution du protocole, l'autre pour visualiser son fonctionnement

root@ubuntu:/opt/tinyos-1.x/apps/LEACH# ./build/pc/main.exe –gui 30

root@ubuntu:/opt/tinyos-1.x/tools/java/net/tinyos/sim# java -jar tinyviz.jar
```

III.3.1 Déclenchement et relai du nouveau round, et annonce des CHs

Les figures V-12, V-13, V-14 représentent 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 (comme illustré dans la figure V-12)

Figure V.12 La diffusion du nouveau round en broadcast par le nœud puits

Ses voisins prennent le relai en envoyant à leur tour selon une transmission broadcast (Figure V-13)

Figure V-13 Diffusion du round par les voisins en broadcast

De plus, nous pouvons voir que les nœuds 15, 7, 18, 26 sont élus CHs. Ces évènements sont marqués par l'activation des LED rouges des CHs (illustré dans la figure V-14). Ensuite, le CHs 15,7, 18, 26 diffusent des ADV pour signaler leurs statuts.

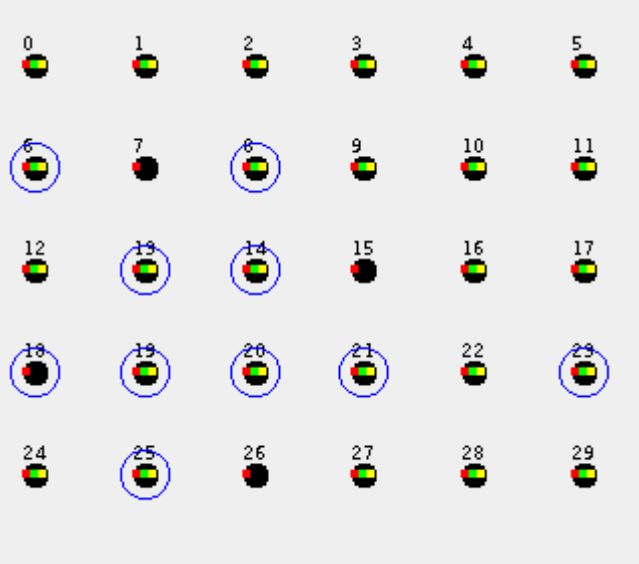


Figure. V-14 : Déclenchement et relai du nouveau round, annonce des CHs 15,7,18,26.

III.3.2 Formation de groupes et envoi des données

La figure V-15 représente quelques transmissions unicast qui se passent durant 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 IV-13 illustre la formation des clusters des CHs 7, 18, 26

Quant à la seconde étape, chaque membre capte la donnée perçue et attend le début de son slot pour qu'il puisse l'envoyer à son CH.

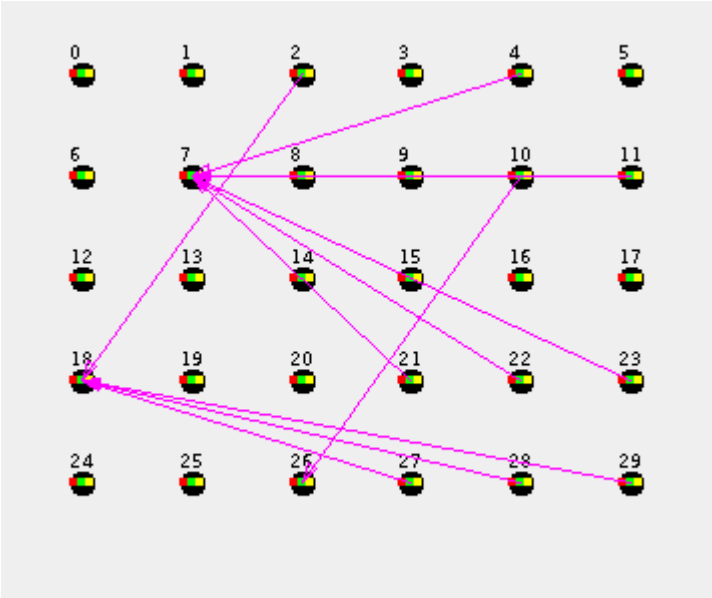


Figure. V-15 : Formation de groupes et envoi des données

III.3.3 Envoi des résultats d'agrégation des données captées au nœud puits :

Dans la figure V-16, les CHs 7, 26,18 agrègent les données reçues et envoie les résultats d'agrégation au nœud puits.

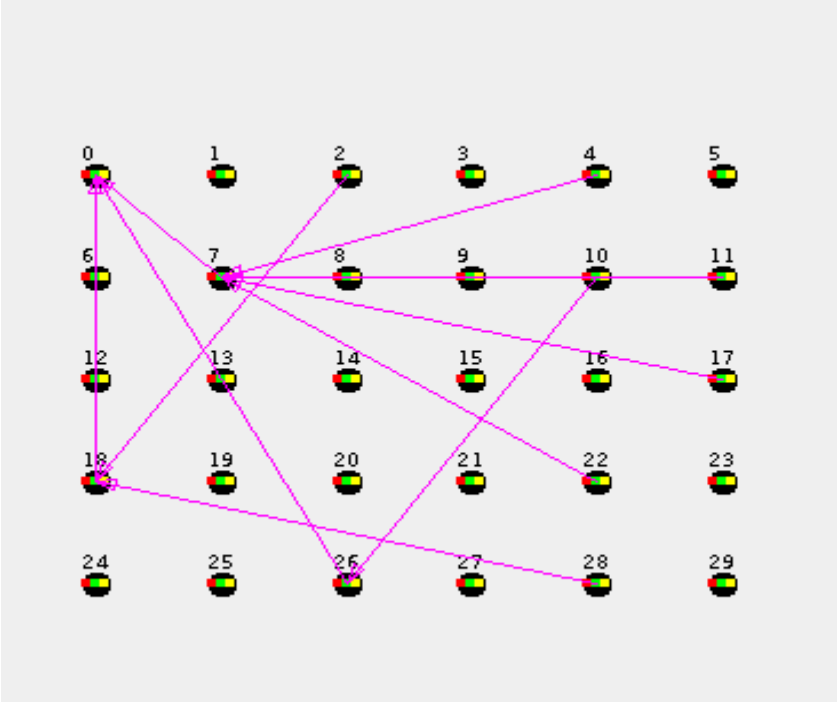


Figure. V-16 : Envoi du résultat d'agrégation du CHs 7,18,26 au nœud puits.

IV. Simulation et évaluation de performances

Pour évaluer les performances des deux protocoles routage ADOV-adapté et LEACH, nous avons effectué des simulations avec les mêmes paramètres et métriques pour les deux protocoles.

IV.1 Métrique considérée

Pour la comparaison entre les deux protocoles, nous prenons en compte la métrique suivante :

La consommation d'énergie

Pour la comparaison des deux protocoles LEACH et AODV-adapté, nous nous intéressons à l'énergie consommée au niveau des nœuds des deux protocoles.

IV.2. Paramétrage de la simulation

Avant de lancer les simulations, nous devons ajuster certains paramètres qui sont présentés par le tableau V-3 :

Nombre de nœuds du réseau	30,40,50,60
Délai de la simulation	60 secondes
Protocoles de routage	LEACH, AODV, AODV-adapté
Taille de paquet de données	29 octets : c'est le paquet de transmission de TinyOS
Système d'exploitation	TinyOS
Simulateur	TOSSIM et POWER-TOSSIM
Langage de programmation	Nesc
Modèle de propagation	simple

TABLEAU.V-3 Paramètres du contexte de la simulation.

V-Résultats et interprétations

Nous avons dans un premier temps mesuré la consommation énergétique des nœuds capteurs pour lesdeux protocoles AODV et AODV-adapté pour des échantillons de 30, 40, 50 et 60 nœuds.

Les résultats de ce test sont récupérés dans des fichiers tracessous la forme suivante :

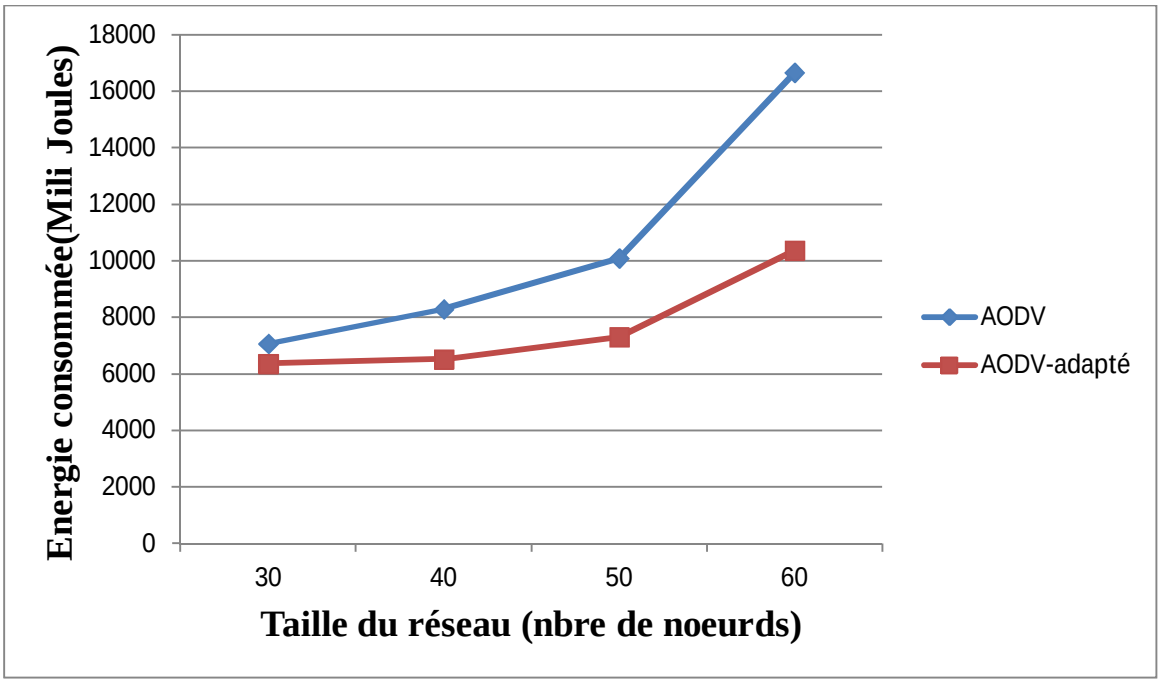
```
Mote 0, cpu total: 202.292555                    Mote 0, radio total: 222.140548
Mote 0, adc total: 0.000000          Mote 0, leds total: 0.000000
Mote 0, sensor total: 0.000000                   Mote 0, eeprom total: 0.000000
Mote 0, cpu_cycle total: 0.000000    Mote 0, Total energy: 424.433104

……..

Mote 29, cpu total: 90.075650        Mote 29, radio total: 153.342176
Mote 29, adc total: 0.000000         Mote 29, leds total: 0.000000          Mote
29, sensor total: 0.000000                  Mote 29, eeprom total: 0.000000
Mote 29, cpu_cycle total: 0.000000       Mote 29, Total energy: 243.417826
```

La consommation énergétique mesurée pour les deux protocoles est la somme de l'énergie consommée par tous les nœuds capteurs.

V-1 Consommation d'énergie par les nœuds pour les protocoles AODV-adapté et AODV

Nœuds **Consommation énergétique(MJ)**	30	40	50	60
AODV	7069,51	8290,85	10096,83	16663,72
AODV-adapté	6362,48	6518,87	7305,61	10371,18

TABLEAU V-4 Consommation énergétiques des nœuds capteurs pour les protocoles AODV et AODV-adapté



Graphe . V-1 : Consommation d'énergie des nœuds capteurs pour les protocoles AODV et AODV_adapté.

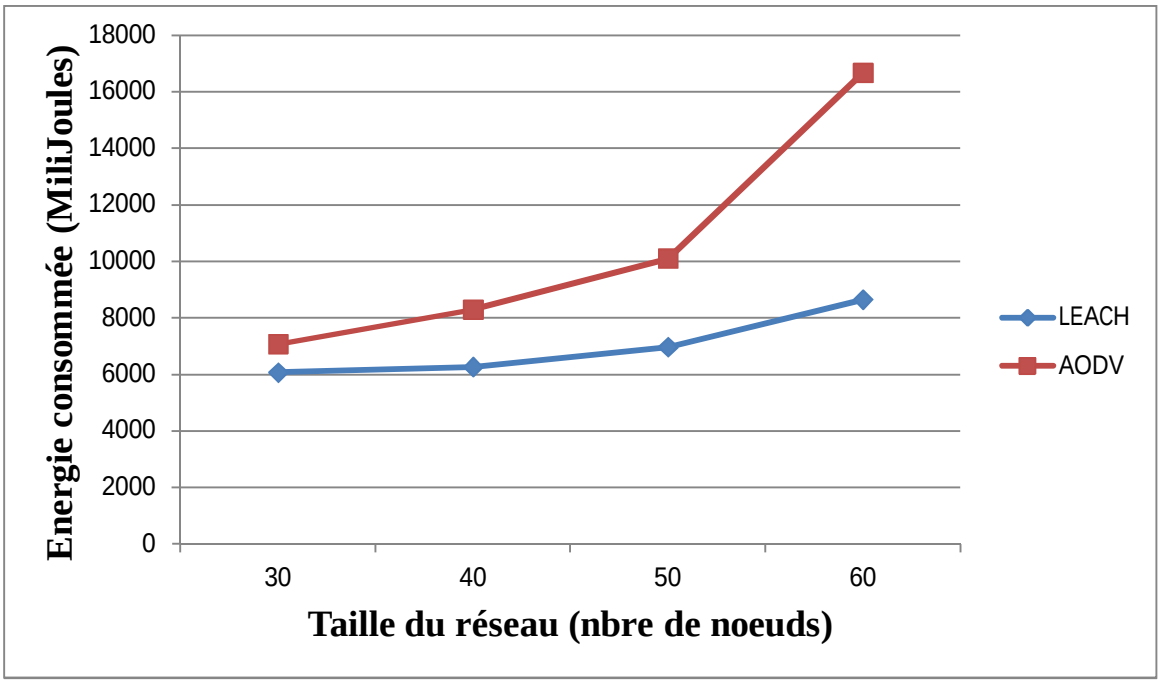
Comme l'illustre les résultats du graphe **V.1**, l'énergie consommée pour le protocole AODV est plus élevée que celle d'AODV-adapté. Cette hausse enregistrée dans la consommation d'énergie est induite par les tâches de recherches et d'établissementsde routes vers le destinataireeffectuées par les nœuds du réseau. Par contre, dans le protocole AODV-adapté, le nœud destinataire est le puits, il est connu par tous les nœuds capteursdu réseau.
Ce qui nous a permis d'améliorer le format de la table de routage (comme il est décrit dans la section II.2) et ainsi le format et la taille des messages échangés entre les nœuds.

V-2 Consommation d'énergie des nœuds pour les protocoles LEACH et AODV :

Il est également intéressant d'analyser la consommation d'énergie par nombre de nœuds du réseau pour les protocoles LEACH et AODV :

Nœuds **Consommation énergétique**	30	40	50	60
LEACH	6062,48	6257	6965,48	8646,25
AODV	7069,51	8290,85	10096,83	16663,72

TABLEAU V-5 Consommation énergétiques des nœuds capteurs pour les protocoles LEACH et AODV



Graphe . V-2 : Consommation d'énergie des nœuds capteurs pour les protocoles LEACH et AODV

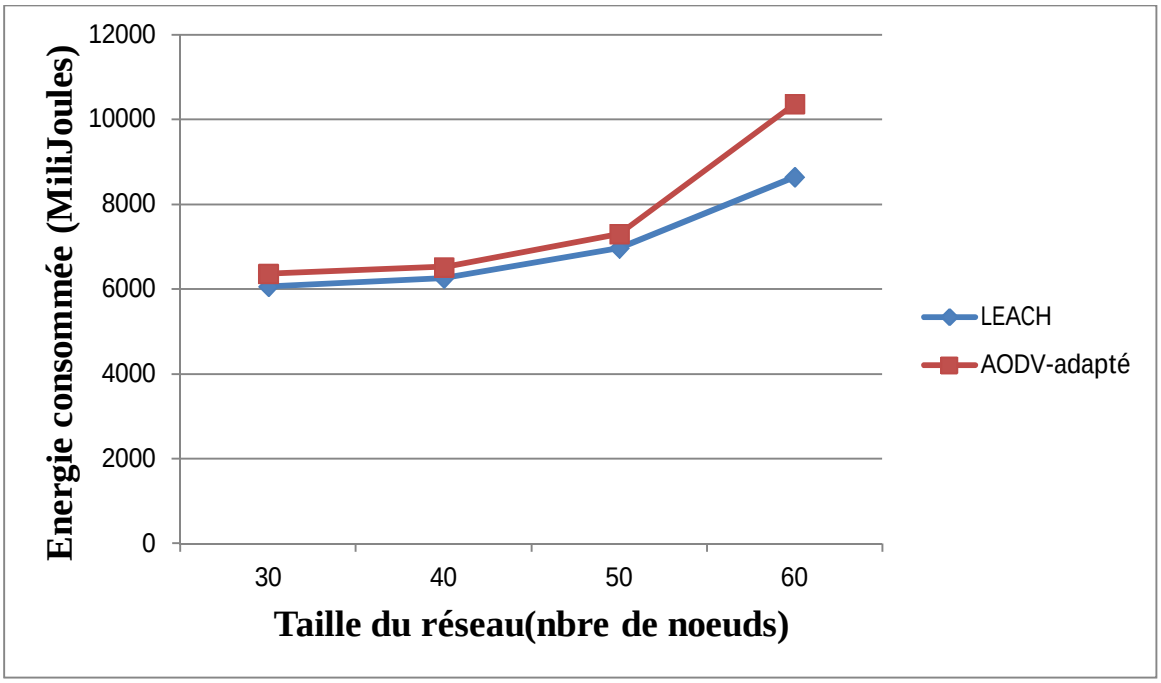
Comme l'illustre le résultat du graphe V-2, nous remarquons que la consommation d'énergie des nœuds capteurs pour le protocole AODV est nettement plus élevée par rapport au protocole LEACH du fait que le protocole LEACH est un protocole destiné uniquement au réseaux de capteurs et qui se base sur l'agrégation des données qui est faite par les CHs ce qui réduit considérablement le flux des données et minimise ainsi la consommation d'énergie des capteurs dans le réseau. D'autre part, le protocole AODV(qui est un protocole des réseaux Ad Hoc) a été adapté aux réseaux de capteurs sans fil ce qu'ille laisse moins fiable que LEACH coté consommation d'énergie.

V-3 Consommation d'énergie des nœuds pour les protocoles LEACH , AODV et AODV-adapté :

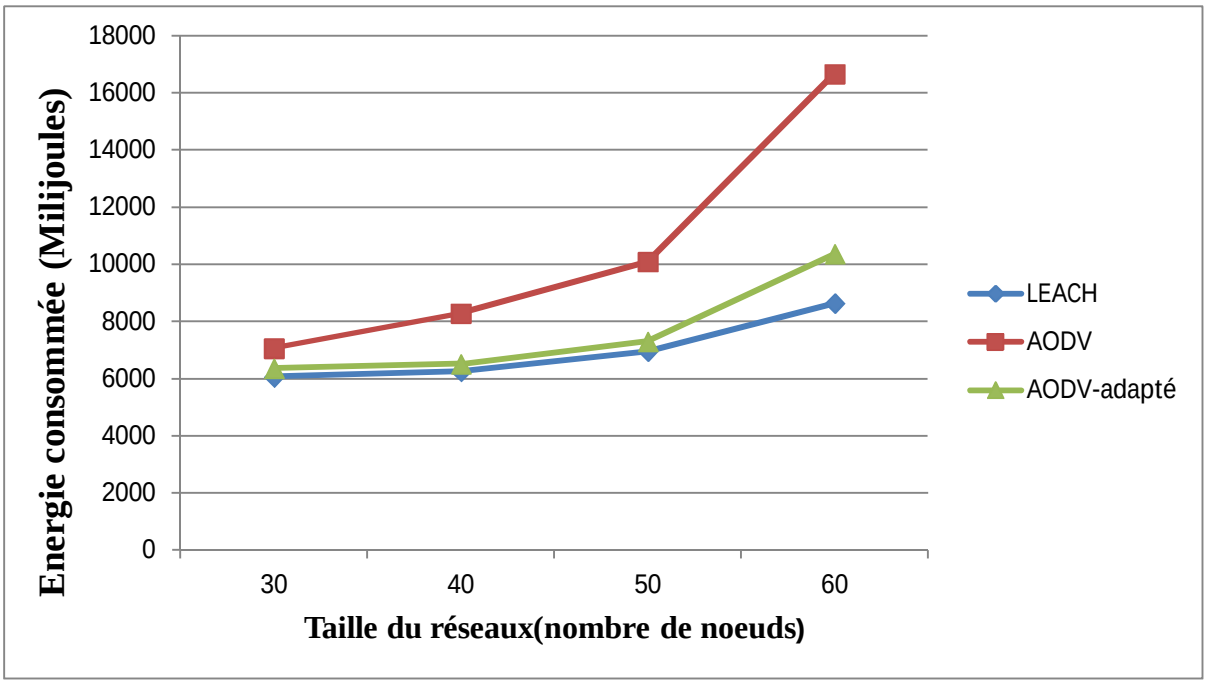
noeuds consommation E	30	40	50	60
LEACH	6062,48	6257	6965,48	8646,25
AODV	7069,51	8290,85	10096,83	16663,72
AODV-adapté	6362,48	6518,87	7305,61	10371,18

Tableau V-6 Consommation énergétiques des nœuds capteurs pour les protocoles LEACH , AODV et AODV-adapté

Les résultats de cette simulation sont représentés dans le graphe suivant :



Graphe . V-3 : Consommation d'énergie des nœuds capteurs pour les protocoles LEACH et AODV-adapté



Graphe V-4 : Consommation d'énergie des nœuds capteurs pour les protocoles LEACH, AODV et AODV-adapté

A travers les résultats des deux graphes V-3, V-4 qui illustre la consommation énergétiques des nœuds capteurs dans le réseau pour les protocoles LEACH, AODV et

AODV-adapté, on constate que le protocole AODV consomme bien plus que LEACH car la recherche des routes à la demandepour le protocole AODV peut consommer beaucoup d'énergie aux capteurs ce qui constitue une limitation à leurs autonomie d'énergie. Les nœuds capteurs qui utilisent le protocole AODV-adapté consomme moins d'énergie que ceux qui utilisent le protocole AODV. Une amélioration au niveau de la bande passante engendrant un gain d'énergie, ce qui montre l'adaptabilité du protocole AODV-adapté aux réseaux de capteurs.

Néanmoins, le protocole LEACH reste le mieux adapté pour la conservation d'énergie dans les réseaux par rapport à notre protocole AODV-adapté.

VI-Comparaison entre le protocole AODV et le protocole LEACH

VI .1 LEACH

- **Avantages**

Le protocole LEACH présente les avantages suivants :

- Auto-configuration des nœuds avec la formation de clusters
- Les données sont fusionnées pour réduire la quantité d'informations transmise vers la station de base.
- La consommation d'énergie est partagée sur l'ensemble des nœuds prolongeant ainsi la durée de vie du réseau.

- **Inconvénients**

En revanche, LEACH a les inconvénients suivants :

- Sans justifier leurs choix, les auteurs fixent le pourcentage optimal de CHs pour le réseau à 5% du nombre total des nœuds. Néanmoins, la topologie, la densité et le nombre de nœuds peuvent être différents dans d'autres réseaux.
- Aucune suggestion n'est faite à propos du temps de réélection des CHs (temps des itérations).
- Les CHs les plus éloignés de la station de base meurent rapidement par rapport à ceux qui sont proches de la station.

VI.2 AODV

L'un des avantages d'AODV est l'utilisation de numéro de séquence dans les messages. Ces numéros de séquences permettent l'éviter les problèmes de boucles infinies et sont essentiels au processus de mise à jour de la table de routage.

Un autre avantage est le rappel de l'adresse IP du nœud origine dans chaque message. Ceci permet de ne pas perdre la trace du nœud à l'origine de l'envoi du message lors des différents relais.

Un inconvénient d'AODV est qu'il n'existe pas de format générique des messages. Chaque message a son propre format : RREQ, RREP, RERR.

VI.3 Comparaison

Le protocole de routage AODV et LEACH

- AODV est un protocole destiné aux réseaux ad hoc tandis que Leach c'est un protocole des réseaux de capteurs.

- LEACH utilise le mécanisme de clustering et il dispose d'un chef de cluster.
- Pour AODV la consommation d'énergie est non partagé entre nœuds contrairement à LEACH la consommation d'énergie est partagé entre les membres d'un cluster.

Conclusion

De nombreux travaux portent aujourd'hui sur l'acheminement d'informations dans les réseaux de capteurs sans fil par les capteurs dans un réseau en prenant en considération, en premier lieu, les communications.

C'est dans ce but que nous avons présenté dans ce chapitre, deux protocoles de routage de type de réseaux différents, le protocole de routage hiérarchique LEACH qui répond aux besoins d'un réseau étendu, notamment un réseau de capteurs sans fil, suit une approche basée sur les groupes. Cette approche a montré son efficacité, comparée à la topologie plate, en termes de consommation et de dissipation uniforme d'énergie prolongeant ainsi la durée de vie du réseau.

Toutefois le protocole LEACH est soumis à certaines contraintes, des inconvénients. Par exemple, la communication unicast, établie entre le nœud puits et les CHs et entre ces derniers et leurs membres, n'est pas toujours efficace par rapport à la communication multi-sauts.

Mais également les chefs de clusters sont soumis à de forte concentration d'énergie dû aux données envoyées par leurs membres.

Dans ce chapitre nous avons également étudié l'implémentation et l'adaptabilité du protocole AODV aux réseaux de capteurs. Ce protocole reste toujours ouvert pour des nouvelles études pour plus d'amélioration du coté énergie et routage car l'optimisation d'un protocole reste toujours un point critique que la majorité des chercheurs essayent d'atteindre.

CONCLUSION GENERALE

Les réseaux de capteurs suscitent un grand intérêt dans le domaine de la recherche. En effet plusieurs travaux ont été proposés afin d'améliorer leurs performances. Parmi ces travaux, les protocoles de routage, qui constituent la plus grande part desrecherches réalisées sur ces derniers.

L'étude réalisée durant ce projet nous a permis de découvrir le monde des réseaux de capteurs sans fils, ainsi que leurs caractéristiques et leurs intérêts dans notre vie.

Dans les réseaux de capteurs sans fil, les nœuds communiquent à travers l'envoie des messages, assurant de telle façon la maintenance et la performance du réseau.

Une des principales problématiques lors du développement de réseaux de capteurs sans fils est le routage, l'absence d'infrastructure et les faibles capacités de calculs et de batteries rendent la fonction du routage plus critique.

L'objectif de notre travail consistait à interpréter les traces résultantes de la simulation des protocoles de routage, plus précisément des deux protocoles de routage AODV-adapté et LEACH suivant la métrique de la consommation d'énergie.

En premier lieu, nous avons présenté un état de l'art sur les réseaux Ad Hoc et les réseaux de capteurs sans fil où nous avons étudié plusieurs aspects liés à ses réseaux, allant des caractéristiques des capteurs, leurs architecture protocolaire, l'alliance ZigBee, en passant par les différentes contraintes imposées par les réseaux de capteurs sans fil et leurs différents domaines d'applications.

Nous avons également étudié les protocoles de routage dans les réseaux de capteurs du point de vue: caractéristiques et techniques de communications utilisées. Nous nous sommes intéressées plus particulièrement au protocole LEACH qui est une référence dans les réseaux de capteurs. Cet état de l'art s'achève par une étude comparative de ces protocoles. Ceci nous a permis de relever les principaux avantages et inconvénients des différentes approches utilisées dans le routage.

Nous avons étudié dans le chapitre III, le fonctionnement du protocole AODV ainsi que son adaptabilité aux réseaux de capteurs, la minimisation la taille de la table de routage nous a permis d'avoir un gain d'espace et d'énergie lors de l'envoie et la réception des messages entre les nœuds capteurs.

Par la suite, on s'est intéressée à l'environnement de simulation TOSSIM, et plus particulièrement à la présentation du système d'exploitation TinyOS minime destiné pour les réseaux de capteurs.

CONCLUSION GENERALE

Les tests et comparaisons effectuées entre AODV et LEACH ont montré que :
Le protocole LEACH prolonge la durée de vie du réseau et minimise la consommation d'énergie comparé au protocole AODV et ceci grâce à la mise en veille des capteurs ne participant pas au routage ainsi que l'agrégation des données qui permet de réduire la quantité de paquets circulant dans le réseau.

Par ailleurs, AODV qui reste une référence dans les réseaux Ad Hoc, son adaptabilité dans les réseaux de capteurs sans fil est toujours ouverte à des futures recherches et améliorations coté routage et gain d'énergie.

BIBLIOGRAPHIE

[1] LEMLOUM, T. *Le Routage dans les Réseaux Mobiles Ad Hoc*. Source : disponible sur http://opera.inrialpes.fr/people/Tayeb.Lemlouma/Papers/AdHoc_Presentation.pdf

[2] CHALLAL, Y. *Réseaux de capteurs sans fils*. 2008, 103 p, support-SIT60.PDF. Source : disponible sur http://moodle.utc.fr/course/view.php?name=SIT60

[3] Source : disponible sur http:// www.xbow.com

[4] BOUNEGTA, N. *Approche distribuée pour la sécurité d'un réseau de capteurs sans fils (RCSF)*. 2010. Source: disponible sur http://www.memoireonline.com/08/10/3831/Approche-distribuee-pour-la-securite-dun-reseau-de-capteurs-sans-fils-RCSF.html

[5] YASER, Y. *Routage pour la gestion de l'énergie dans les réseaux de capteurs sans fil* . 2010. 147 p. Source : disponible http://www.scd.uha.fr/flora/servlet/DocumentFileManager?source=ged&document=ged:IDOCS:3342&resolution=MEDIUM&recordId=defaultfortmelectro%3ATM_ELECTRO%3A231&file=

[6] W. Heinzelman, A.P. Chandrakasan and H. Balakrishnan, *Energy-Efficient Communication Protocol for Wireless Microsensor Networks*. pp. 258 –269. (2002).

[7] W. Heinzelman, A.P. Chandrakasan and H. Balakrishnan, *"An Application-Specific Protocol Architecture for Wireless Microsensor Networks"*, In IEEE Transactions on Wireless Communications, Vol. 1, No. 4, pp. 660-670. (2002)

[8] T. Kwon and M. Gerla, *"Clutering with Power Control"*, In Proceedings MILCOM, volume 2. (1999)

BIBLIOGRAPHIE

[9] E.-S. Jung and N. H. Vaidya, *"A power control MAC protocol for ad-hoc networks"*, In ACM MOBICOM. (2002)

[10] OULARBI, Mounira , KASSAB , Soraya. *Élaboration d'un protocole de routage efficace en énergie pour réseaux de capteurs sans fil.* diplôme d'Ingénieur d'Etat en Informatique, LABORATOIRE LMCS 2010, Source : disponible http://share.esi.dz/index.php?option=com_docman&task=doc_details&gid=411&Itemid=1

[11] KHELLADI, Lyes, BADACHE, Nadjib**.** *Les réseaux de capteurs: état de l'art,* Rapport de recherche, Alger, Février 2004. Source : disponible http://www.lsi-usthb.dz/Rapports_pdf/2004/LSI-TR0304.pdf

[12] Sébastien Tixeuil, Ted Herman, **«** *Un algorithme TDMA réparti pour les réseaux de capteurs* **»**, INRIA Projet Grand Large, Universités Iowa et Paris-Sud XI, 2004.

[13] Preetha Radhakrishnan, **"***Enhanced routing protocol for graceful degradation in wireless sensor networks during attacks***"**, Thèse d'ingéniorat, Université de Madras, Chennai, Décembre 2005.

[14] BEYDOUN, Kamal**,** *Conception d'un protocole de routage hierarchique pour les reseaux de capteurs*. 2009, 141 p. Source : disponible http://lifc.univ-fcomte.fr/home/~hguyennet/These_beydoun.pdf

[15] Eric Lawrey**, «**The suitability of OFDM as a modulation technique for wireless telecommunications, with a CDMA comparison**»,** Projet d'ingéniorat, Université James Cook, Australie, 2001.

[16] Djallel Eddine Boubiche, **«***Protocole de routage pour les réseaux de capteurs sans fil*», Mémoire de magistère, Université de l'Hadj Lakhdar, Batna, Algérie, 2008.

[17] Wendi Beth Heinzelman, **«***Application-Specific Protocol Architectures for Wireless Network* », IEEE Transactions on Wireless Communications, Massachusetts Institute of Technology, June 2000.

BIBLIOGRAPHIE

[18] Yasser Romdhane, **«** Evaluation des performances des protocoles S-MAC et Directed Diffusion dans les réseaux de capteurs », Projet de fin d'études, Ecole Supérieure des Communications de Tunis (Sup'Com), 2006 / 2007.

[19] L.Daumas, O.Uberti, *Implémentation d'un protocole de routage dans un réseau de capteurs sans-fils.* master 1 informatique, Université d'Avignon, 2008

[20] *ZIGBEE Protocole pour WPAN.* Source : disponible site : http://wapiti.telecom-lille1.eu/commun/ens/peda/options/ST/RIO/pub/exposes/exposesrio2006/Descharles-Waia/standards.htm

[21] Berrachedi, A, Diarbakirli, A, *Sécurisation du protocole de routage hiérarchique LEACH dans les réseaux de capteurs sans fil***.** 2008/2009 disponible sur le site : http://takeitesi.blogspot.com/

[22] *TinyOS Getting Started Guide***,** Document 7430-0022-0, October 2003, source: disponible http://www.es.ele.tue.nl/education/oo2/files/doc/Getting_Started_Guide.pdf

[23] HAMZI,A, *Plateforme basée agents pour l'aide à la conception et la simulation des réseaux de capteurs sans fil*, Juillet 2007diponible sur le site : http://share.esi.dz/index.php?option=com_docman&task=doc_details&gid=84&Itemid=1

[24] Gérard Chalhoub, *Routage et MAC dans les réseaux de capteurs sans fil.* 2010 disponible sur le site :http://sancy.univ-bpclermont.fr/~chalhoub/wsn.pdf

[25] *Getting started guide* , August 2004.Document 7430-0022-05**. Source: disponible:** http://sfdoccentral.symantec.com/sf/4.0/aix/pdf/getting_started.pdf

[26] Kamal BEYDOUN **,** « *conception d'un protocole de routage hierarchique pour les reseaux de capteurs* **»,** GRADE DE DOCTEUR DE L'UNIVERSITE DE FRANCHE-COMTE Spécialité Informatique, 2009.

BIBLIOGRAPHIE

[27] *Ad hoc On-Demand Distance Vector (AODV) Routing*, Network Working Group C. Perkins Nokia Research Center, E. Belding-Royer, University of California, Santa Barbara, S. Das, University of Cincinnati, July 2003. RFC 3561. Category: Experimental
Source : disponible http://www.ietf.org/rfc/rfc3561.txt

[28] ESSOUIHLI**,** O, « *OPTIMISATION DES PERFORMANCES DES RESEAUX MANET PAR LE LOAD BALANCING* **»,** Projet de fin d'études, Ecole Supérieure des Communications de Tunis (Sup'Com), 2006.

[29] BACCOUR,N , *« Etude comparative de deux simulateurs pour les réseaux sans fil Ad-hoc »***,** Projet de fin d'études, l'Ecole Nationale d'Ingénieurs de Sfax, juillet 2005

[30] FARES, Abdelfatah. *Développement d'une bibliothèque de capteurs.* 2008. 54 p

[31] V. Kawadia and P. R. Kumar, *"Power Control and Clustering in Ad Hoc Networks"*, IEEE INFOCOM. (2003)

[32] DEHNI, Lahcene , BENNANI, Younès, KRIEF, Francine, *LEA2C : Une nouvelle approche de routage dans les réseaux de capteurs pour l'optimisation de la consommation d'énergie. Source : disponible sur* http://www-lipn.univ paris13.fr/~bennani/Publis/LEA2C_Gres_2005.pdf

[33] HENI, J, CAUDRON, A, BRISSET , P, Marc GUITARD,*reseau de capteurs sans fils* , Option : Systèmes Embarqués, 3ème année électroniques

[34] ZNAIDI,W**,** *Modélisation formelle de réseaux de capteurs à partir de tinyos* ,Juin 2006 ,Organisme d'accueil : **VERIMAG**

[35] Source : disponible http://www.docstoc.com/docs/2162186/TinyOS-Tutorial-Part-I

[36] Source : disponible http://webs.cs.berkeley.edu/tos/nest/doc/tutorial/tossim-lesson.html

[37] Source : disponible http://webs.cs.berkeley.edu/tos/tinyos-1.x/tools/java/net/tinyos/sim/README

BIBLIOGRAPHIE

[38] Source : disponible http://cs.acadiau.ca/~shussain/wsn/apps/tos1/mica2/allfiles_p.html

[39] Source : disponible http://www.tinyos.net/tinyos-1.x/doc/index.html

[40] Source : disponible http://www.5secondfuse.com/tinyos/install.html

[41] Source : disponible http://www.tinyos.net/tinyos-1.x/doc/tutorial/

ANNEXE A : INSTALLATION DE LA PLATEFORME TINYOS

Les réseaux de capteurs utilisent un très grand nombre de capteurs qui ont la fonction d'analyser leurs environnement et qui reliés ensemble forment un réseau sans fil en se basant sur des protocoles de routage pour communiquer et propager les données récoltées aux capteurs appartenant à leurs zones de couverture. Chaque capteur relayant l'information sur sa propre zone de couverture, le réseau se trouve entièrement couvert.

Afin de répondre aux caractéristiques et aux nécessités des réseaux de capteurs, telles qu'une taille de mémoire réduite ainsi qu'une basse consommation d'énergie, un système d'exploitation spécialisé minime destiné pour ces réseaux a été créé***TinyOS***.Il est le système actuellement le plus utilisé dans les applications nécessitant des capteurs.

Enfin, pour le développement des applications légères, il n'existe actuellement qu'un langage de programmation capable d'interagir avec le système d'exploitation **TinyOs : NesC**. Ce langage dédié est proche du C traditionnel mais il est **orienté composants**.

Dans cette annexe, nous allons introduire et développer les étapes d'installation de la plateforme TinyOS, ainsi que le fonctionnement et l'utilisation du simulateur Tossim et l'interface graphique TinyViz.

I- ENVIRONNEMENT DE TRAVAIL

I.1. Le système d'exploitation TinyOS

TinyOS est le système d'exploitation open-source, conçu par des chercheurs de Berkeley, pour des réseaux de capteurs sans fils. Son architecture est basée sur une association de composants s'appuyant sur le langage NesC qui est un langage orienté composant, syntaxiquement proche du C.

TinyOS s'appuie sur un fonctionnement évènementiel, ceci, combiné avec une programmation orientée composant, permet de réduire la taille du code nécessaire à sa mise en place et respecte ainsi les contraintes émises par les capteurs en termes d'économie de mémoire et d'énergie.

Un composant correspond à un élément matériel (LEDs, timer, ...) et peut être réutilisé dans différentes applications.

Un autre grand intérêt à utiliser TinyOS est sa bibliothèque de composants très complète implémentant des pilotes de capteurs, des outils de capture et des protocoles réseaux plus particulièrement intéressants pour des recherches comme la simulation des taux de perte de paquets sous le simulateur Tossim, etc

a- Plates-formes sous TinyOS

TinyOS est prévu pour fonctionner sous plusieurs plates-formes comme Windows (2000 et XP) ou bien GNU/Linux.

Les versions de TinyOS sont multiples, parmi elles la version stable (V. 1.x) et La version en développement (V. 2.x)

On visitant le site officiel de TinyOS, on peut trouver toutes les versions existantes de TinyOS sous forme de zip comme l'illustre la figure A-1 :

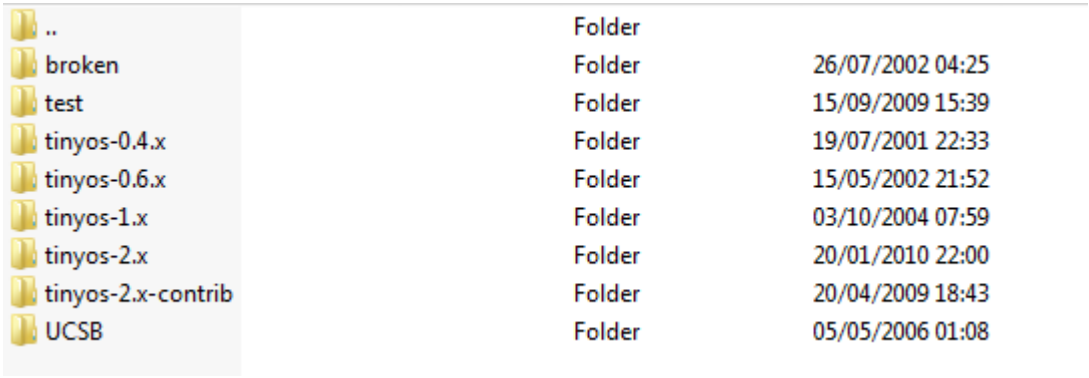


Figure .A-1. Les différentes versions de TinyOS

I.2. Simulation : TOSSIM et TinyViz

I.2.1.Le simulateur TOSSIM

Afin de simuler le comportement des capteurs, un outil très puissant a été développé et proposé sous le nom de TOSSIM. Ce dernier est souvent utilisé avec une interface graphique (TinyViz) pour une meilleure compréhension et visualisation de l'état du réseau. L'utilisation de ces deux logiciels est immédiate dès lors que TinyOS est opérationnel.

I.2.2. TinyViz

L'outil TinyViz est une application graphique qui nous permet d'avoir un aperçu de notre réseau sans avoir à déployer les capteurs dans la nature. Une économie d'effort et une préservation du matériel sont possibles grâce à cet outil. L'application permet une analyse étape par étape en activant les différents modes disponibles.

Cet annexe sera exploité pour l'installation des différentes versions de TinyOS plus précisément TinyOS-2.1.1, TinyOS-1.x, TinyOS-2.x et TinyOS-2.x-contrib

II-Etapes d'installation de TinyOS sous Linux

Nous avons procédés à l'installation de TinyOS sur la distribution Ubuntu 10.10 et Ubuntu11.04 (debian)

III.1. Arborescence exploitée :

- Les systèmes d'exploitation tinyos-1.x, 2.x, 2.x-contrib et 2.1.1 seront mis dans le répertoire /opt.
- La figure IV-2 nous illustre quelques exemples de répertoires existants dans l'arborescence des différentes versions de tinyos.

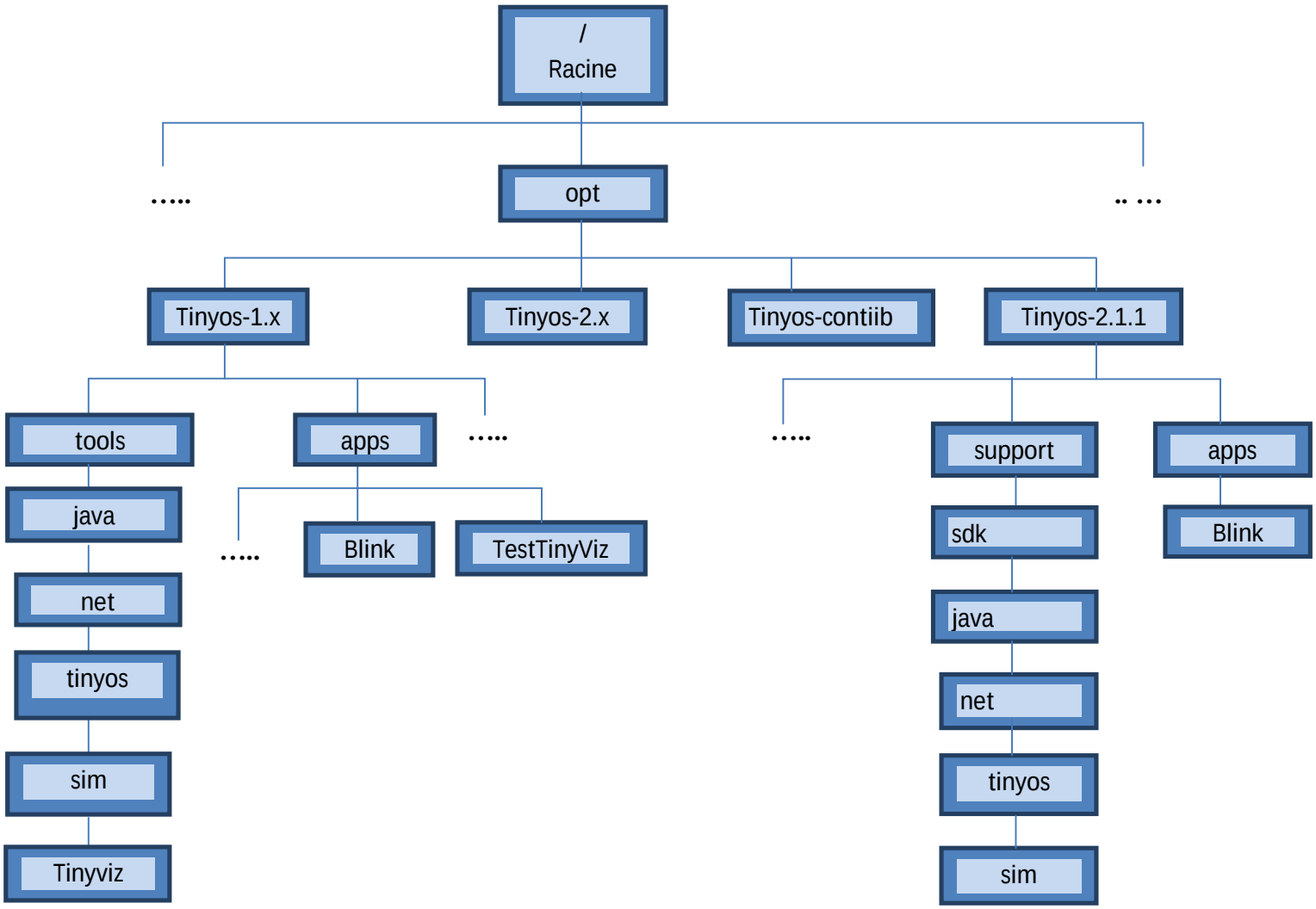


Figure A-2. **Arborescence exploitée**

II.2. Installation et test d'installation de TinyOS-1.x, TinyOS-2.x et TinyOS-2.x-contrib

Après plusieurs recherches effectuées sur le système d'exploitation des capteurs sans fil TinyOS, la version TinyOS-1.x nous paressent la plus adéquate à notre recherche, malgré la difficulté trouvée dans son installation due aux rares tutoriaux existants dessus.

Le point majeur qui caractérise le TinyOS-1.x, c'est qu'on plus de son simulateur TOSSIM la disponibilité de son interface graphique Tinyviz qui est absente dans les versions tinyos-2.x et tinyos-2.x-contrib.

Cette interface graphique programmée en java nous permet de mieux assimiler le fonctionnement des plugins mais également de représenter le réseau de capteurs sans fil émulé grâce au simulateur TOSSIM.

En plus de la simulation des capteurs, l'option de la réception des résultats de la simulation est disponible dans la même fenêtre.

III-2-1-Installation de tinyos-1.x sous Linux :

On se référant sur un article porté spécialement sur l'installation TinyOs-1.x sous Debian, nous avons suivi les étapes suivantes :

Sous terminal :

1- Sous mode root : on ouvre le fichier /etc/apt/sources.list

```
sabrina@sabrina-eMachines-E510:~$ su
Mot de passe :
root@sabrina-eMachines-E510:/home/sabrina# cd

root@sabrina-eMachines-E510:~# sudo /etc/apt/sources.list
```

ANNEXE A : INSTALLATION DE LA PLATEFORME TINYOS

On ajoute les lignes (dépôts universels et standard) suivantes :

```
# installation de tinyos -1.x

deb http://tinyos.stanford.edu/tinyos/dists/ubuntu edgy main
deb http://us.archive.ubuntu.com/ubuntu/ feisty main restricted
deb-src http://us.archive.ubuntu.com/ubuntu/ feisty main restricted

deb http://us.archive.ubuntu.com/ubuntu/ feisty-updates main restricted
deb-src http://us.archive.ubuntu.com/ubuntu/ feisty-updates main restricted

deb http://us.archive.ubuntu.com/ubuntu/ feisty universe
deb-src http://us.archive.ubuntu.com/ubuntu/ feisty universe

deb http://us.archive.ubuntu.com/ubuntu/ feisty multiverse
deb-src http://us.archive.ubuntu.com/ubuntu/ feisty multiverse

deb http://security.ubuntu.com/ubuntu feisty-security main restricted
deb-src http://security.ubuntu.com/ubuntu feisty-security main restricted
deb http://security.ubuntu.com/ubuntu feisty-security universe
deb-src http://security.ubuntu.com/ubuntu feisty-security universe
deb http://security.ubuntu.com/ubuntu feisty-security multiverse
deb-src http://security.ubuntu.com/ubuntu feisty-security multiverse

deb http://tinyos.stanford.edu/tinyos/dists/ubuntu feisty main
```

- Il s'agit des versions supportées sous Debian par les dépôts de TinyOS mise sous la forme suivante :

deb http://tinyos.stanford.edu/tinyos/dists/ubuntu <distribution> main

- La version la plus récente supportée par ces dépôts TinyOS est lucid, ce qui donne:

deb http://tinyos.stanford.edu/tinyos/dists/ubuntu lucid main

2- l'environnement de développement de TinyOS requiert l'utilisation du compilateur avrgcc, perl, flex … ainsi que la JDK 1.5 ou une version plus récente.

De ce fait, pour installer TinyOScertains paquets doivent être installé préalable :

- Cvs
- Subversion
- Autoconf
- automake1.9
- python-dev
- g++
- g++-3.4
- gperf
- swig

- sun-java5-jdk
- graphviz
- alien
- fakeroot

```
root@sabrina-eMachines-E510:~# sudo apt-get install cvs subversion autoconf
automake1.9 python-dev

root@sabrina-eMachines-E510:~# sudo apt-get install g++ g++-3.4 gperf swig sun-
java5-jdk graphviz alien fakeroot

root@sabrina-eMachines-E510:~# sudo apt-get install tinyos-msp430 tinyos-avr
```

Quelques remarques :

- Si les messages apparaissent :
  Aucune version du paquet sun-java5-jdk n'est disponible, mais il existe dans la base de données.
  E: Impossible de trouver le paquet g++-3.4
  E: Impossible de trouver de paquet correspondant à l'expression rationnelle «g++-3.4»
  E: Le paquet « sun-java5-jdk » n'a pas de version susceptible d'être installée

- Cela signifie que le paquet est manquant, qu'il est devenu obsolète ou qu'il n'est disponible que sur une autre source.
- Si Ubuntu 10.10 est installé comme système d'exploitation cohabitant avec Windows , java-open6-jdk est installé par défaut : le chemin pour y accéder est: \usr\include\lib mais si Ubuntu 10.10 est installé comme application sous Windows le jdk est inexistante et il faudra l'installer.

- Pour le paquet g++_3.4 : cette version est obsolète sous Ubuntu 10.10 qui accepte à la place le paquet g++_4.4 installé automatiquement en installant le g++.

- Sous Ubuntu 10.10 tinyos- avr et tinyos -msp430 sont sous d'autres appellations :avr-tinyos et msp430-tinyos.

3- Nous ajoutons certaines variables d'environnements qui permettent d'utiliser le TinyOS-1.x, TinyOS-2x en spécifiant le chemin d'accès vers ces derniers.

```
root@sabrina-eMachines-E510:~#sudo gedit ~/.bashrc
```

- Dans notre fichier ~/.bashrc nous ajoutons les lignes suivantes.

Ce sont les nouvelles variables utiles pour TinyOS.

ANNEXE A : INSTALLATION DE LA PLATEFORME TINYOS

a- Le chemin d'accès à java :

```
# Java
export JDKROOT=/usr/lib/jvm/java-6-openjdk
export JAVAXROOT=$JDKROOT

if [ "$LD_LIBRARY_PATH" == "" ]; then
 # So Java can find libtoscomm.so and libgetenv.so
export LD_LIBRARY_PATH=$JDKROOT/jre/lib/i386
fi
```

b- Le choix de la version par défaut de TinyOS, nous utilisons la version tinyos-1.x :

```
        # Set your default version of TinyOS here.
        if [ "$TINYOS_VER" == "" ]; then
          export TINYOS_VER=1
        fi
```

c-On ajoute également ces lignes suivantes :

```
# Preserve any non Tinyos related class paths that have been set prior
# to this file being sourced. We check the for an empty OLD_CLASSPATH
# to ensure that OLD_CLASSPATH only gets set once.
if [ "$OLD_CLASSPATH" == "" ]; then
if [ "$CLASSPATH" == "" ]; then
            export OLD_CLASSPATH="empty"
else
            export OLD_CLASSPATH=$CLASSPATH
            fi
        fi
```

d- Configuration de l'environnement pour les versions de TinyOS :

```
# Conditional environmental setup for 3 versions of TinyOS
if [ "$TINYOS_VER" == "1" ]; then

echo "Setting up for TinyOS 1.x"

export TOSROOT=/opt/tinyos-1.x
export TOSDIR=$TOSROOT/tos
export MAKERULES=$TOSROOT/tools/make/Makerules

unset CLASSPATH

  # Restore the old non tinyosclasspath if its not empty
  if [ "$OLD_CLASSPATH" != "empty" ]; then
        export CLASSPATH=$OLD_CLASSPATH
  fi

  export CLASSPATH=$CLASSPATH\:`$TOSROOT/tools/java/javapath`


  # These aliases only apply to a 1.x environment
aliastinyviz="$TOSROOT/tools/java/net/tinyos/sim/tinyviz"
  alias Deluge="java net.tinyos.tools.Deluge"

else

echo "Setting up for TinyOS 2.x"
export TOSROOT=/opt/tinyos-2.x
export TOSDIR=$TOSROOT/tos

  unset CLASSPATH

  # Restore the old non tinyosclasspath if its not empty
  if [ "$OLD_CLASSPATH" != "empty" ]; then
        export CLASSPATH=$OLD_CLASSPATH
  fi

export
CLASSPATH=$CLASSPATH\:$TOSROOT/support/sdk/java/tinyos.jar\:$TOSR
OOT/support/sdk/java/

export PYTHONPATH=:$TOSROOT/support/sdk/python/
export MAKERULES=$TOSROOT/support/make/Makerules
unset TOSMAKE_PATH
```

e-Définir le chemin pour la variable d'environnement msp430 (compilateur pour les plateformes telos) et l'utilitaire alias :

```
# Finally set the path for the new MSPGCCROOT
export MSPGCCROOT=/opt/msp430
export PATH="$MSPGCCROOT/bin:$PATH"

# Helpful aliases
aliassf="java net.tinyos.sf.SerialForwarder"
alias listen="java net.tinyos.tools.Listen"

alias apps="cd $TOSROOT/apps"
aliastos="cd $TOSROOT/tos"
```

f- Quelques fonctions de changements d'environnements :

```
# Environment changing functions

# Change this shell's environment to support Tinyos 2.x development
function tos2 () {
  export TINYOS_VER=2
  . ~/.bash_tinyos
}

# Change this shell's environment to support Tinyos 1.x development
function tos1 () {
  export TINYOS_VER=1
  . ~/.bash_tinyos
}
```

g- Définir la variable d'environnement MOTECOM :

```
# Set the MOTECOM variable to the first mote in motelist
functionsetMoteCom () {
   MOTE=`motelist -c| cut -d, -f2`

if [ "$BOOMERANG" == "1" ]; then
      TYPE="tmote"
else
      TYPE="telosb"
fi
```

h- A la fin de notre ~/.bashrc , on ajoute ces quelques lignes :

```
# Add this to your .bashrc
if [ -f ~/.bash_tinyos ]; then
. ~/.bash_tinyos
fi
```

- Cela nous permettra de basculer entre les environnements à la volée, par défaut nous avons configuré TinyOS-1.x.
- **tos** (Raccourci vers le dossier « tos » de l'environnement configuré)

```
Setting up for TinyOS 1.x
root@sabrina-eMachines-E510:~# tos
root@sabrina-eMachines-E510:/opt/tinyos-1.x/tos#...

Setting up for TinyOS 2.x
root@sabrina-eMachines-E510:~# tos
root@sabrina-eMachines-E510:/opt/tinyos-2.x/tos#
```

4- *L'installation de TinyOS 1.x:*

Tapez les commandes qui suivront :

- La première commande nous demande un mot de passe c'est la touche entrer, si cette commande n'est pas exécuter correctement le dossier build dont on aura besoin pour lancer tinyviz ne contiendra pas l'exécutable main.exe important dans la récupération des résultats d'exécutions de commandes de tinyviz.

```
root@sabrina-eMachines-E510:~# cvs
-d:pserver:anonymous@tinyos.cvs.sourceforge.net:/cvsroot/tinyos login
Logging in to :pserver:anonymous@tinyos.cvs.sourceforge.net:2401/cvsroot/tinyos
CVS password:
        cvs login: CVS password file /root/.cvspass does not exist - creating a new file

root@sabrina-eMachines-E510:~# cvs -z3 -
d:pserver:anonymous@tinyos.cvs.sourceforge.net:/cvsroot/tinyos co tinyos-1.x
root@sabrina-eMachines-E510:~# sudo mv tinyos-1.x /opt
```

5- *Facultatif: installation TinyOS-2.x et tinyos-2.x-contrib*

- Les mêmes commandes que celles utilisées pour le tinyos-1.x en modifiant la version à celle souhaitée pour l'installation :

```
root@sabrina-eMachines-E510:~# cvs -
d:pserver:anonymous@tinyos.cvs.sourceforge.net:/cvsroot/tinyos login

root@sabrina-eMachines-E510:~# cvs -z3 -
d:pserver:anonymous@tinyos.cvs.sourceforge.net:/cvsroot/tinyos co tinyos-2.x

root@sabrina-eMachines-E510:~# cvs -z3 -
d:pserver:anonymous@tinyos.cvs.sourceforge.net:/cvsroot/tinyos co tinyos-2.x-contrib
```

6- Test de l'installation :

- *Depuis le terminal* :

```
root@sabrina-eMachines-E510:~# su
Mot de passe :
Setting up for TinyOS 1.x
```

- **Script toscheck :**

Pour vérifier que les outils tels que nesc, avr, msp430 et autres ont été installés correctement et que les variables d'environnement sont définies.Le toscheck est un script qui va exécuter ces fonctions.

On ouvrons le terminal, on se positionne au répertoire suivant :*opt/tinyos-1.x/tools/scripts* et on lance ./ *toscheck*.
La dernière ligne doit être :

```
.........
toscheckcompletedwithouterror
```

Si des erreurs sont signalées, on doit assurez de régler le problème.

II-3- SIMULATIONS AVEC TinyViz :

1- make<platform>:

Dans le répertoire apps de TinyOS-1.x plusieurs applications dans celle de Blink existe, pour compiler, on utilise la commande *make* : *make*<*platform*>qui crée un dossier *build* comportant des fichiers ainsi qu'un exécutable main.exe

Comme la montre les figures suivantes :

Figure A-3 L'application Blink

- *On lance ces commandes :*

```
root@sabrina-eMachines-E510:/home/sabrina# cd /opt/tinyos-1.x
root@sabrina-eMachines-E510:/opt/tinyos-1.x# cd apps
root@sabrina-eMachines-E510:/opt/tinyos-1.x/apps# cd Blink
root@sabrina-eMachines-E510:/opt/tinyos-1.x/apps/Blink# make pc
mkdir -p build/pc
compiling Blink to a pc binary
ncc -o build/pc/main.exe -g -O0 -pthread -fnesc-nido-tosnodes=1000 -fnesc-
simulate
……
compiled Blink to build/pc/main.exe
```

- *On obtient notre nouveau dossier build:*

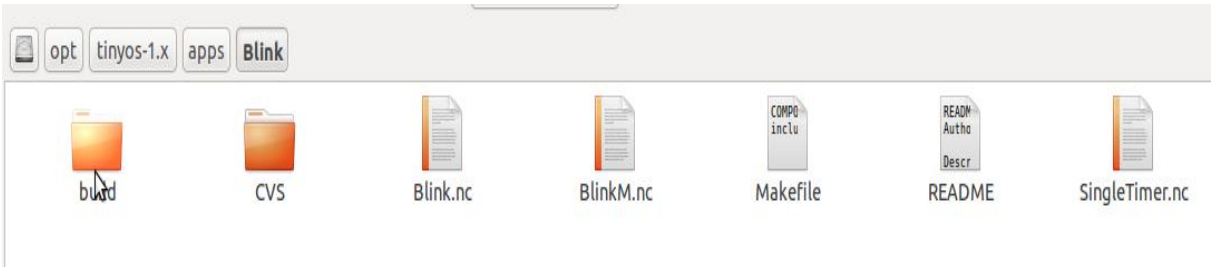


Fig.A-4 make pc surl'application Blink

- On compilant la plate-forme pc on obtient l'exécutable main.exe

Fig.A-5 L'exécutable main.exe

Maintenant, on peut lancer la simulation des nœuds capteurs :

```
root@sabrina-eMachines-E510:/opt/tinyos-1.x/apps/Blink# export DBG=led
root@sabrina-eMachines-E510:/opt/tinyos-1.x/apps/Blink# build/pc/main.exe
Usage: build/pc/main.exe [-h|--help] [options] num_nodes_total
root@sabrina-eMachines-E510:/opt/tinyos-1.x/apps/Blink# build/pc/main.exe 1
```

- *La commande très importante :* build/pc/main.exe – help nous permet de visualiser les différentes options offertes pour la simulation des RCSF

2- interface TinyViz du simulateur TOSSIM :

Pour compiler TinyViz, on doit se placer au repertoireopt/tinyos-1.x/tools/java et taper la commande : make pour compiler lesfichiers.class en fichiers.java

```
root@sabrina-eMachines-E510:/opt/tinyos-1.x/apps/Blink# cd /opt/tinyos-
1.x/tools/java/net/tinyos/simot@sabrina-eMachines-E510:/opt/tinyos-
1.x/tools/java/net/tinyos/sim# make clean
cleaning /opt/tinyos-1.x/tools/java/net/tinyos/sim

root@sabrina-eMachines-E510:/opt/tinyos-1.x/tools/java# make

root@sabrina-eMachines-E510:/opt/tinyos-1.x/tools/java/net/tinyos/sim# make
... /opt/tinyos-1.x/tools/java/net/tinyos/sim
(cd msg; make)
……..
root@sabrina-eMachines-E510:/opt/tinyos-1.x/tools/java/net/tinyos/sim# cd lossy/
root@sabrina-eMachines-E510:/opt/tinyos-1.x/tools/java/net/tinyos/sim/lossy# make
... /opt/tinyos-1.x/tools/java/net/tinyos/sim/lossy
javac Mote.java
javac SpatialReader.java
Note: SpatialReader.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
javac TopologyGenerator.java
root@sabrina-eMachines-E510:/opt/tinyos-1.x/tools/java/net/tinyos/sim/lossy# cd
/opt/tinyos-1.x/tools/java/net/tinyos/sim
root@sabrina-eMachines-E510:/opt/tinyos-1.x/tools/java/net/tinyos/sim# make jarfile
```

3- La commande *makejarfile* compile le *makefile* du dossier simen créant des classes java dans les différents dossiers spécifiés dans le makefile comme le dossier event, paquet, msg,… et autres, mais également crée un jar de tinyviz nommé simdriver.jar ou simdriver.temp.jar qui permettra de lancer l'interface tinyviz par la suite.

❖ Résultat du makejarfile

```
SUBDIRS = event plugins packet lossy script
ROOT = ../../..
PLUGINS_SRC = $(wildcard plugins/*.java)
PLUGINS = $(PLUGINS_SRC:.java=.class)
INITIAL_TARGETS = msgsjython ../sf/old/nido/NidoSerialDataSource.class
OTHER_CLEAN = msgs-clean plugins-list-clean jarclean
# Uncomment this line to make jarfile mandatory
FINAL_TARGETS = jarfile
include $(ROOT)/Makefile.include
../sf/nido/NidoSerialDataSource.class: ../sf/old/nido/NidoSerialDataSource.java
(cd ../sf/nido; $(MAKE))
msgs:
(cdmsg; $(MAKE))
msgs-clean:
(cdmsg; $(MAKE) clean)
# Make sure that jython gets built
jython: $(ROOT)/org/python/core/parser.class
$(ROOT)/org/python/core/parser.class:
(cd $(ROOT)/org/python && $(MAKE))
(cd $(ROOT)/org/apache && $(MAKE))
# Create a list of default plugins
plugins/plugins.list: $(PLUGINS)
echo $(PLUGINS) > plugins/plugins.list
plugins-list-clean:
rm -f plugins/plugins.list
# This is ugly. The only way to embed a jar file inside another is to
# unpack it and repack them together into a single flat file.
jarfile: plugins/plugins.list
@echo "Creating simdriver.jar..."
(cd $(ROOT); \
jarcmf net/tinyos/sim/simdriver.manifest \
net/tinyos/sim/simdriver-tmp.jar \
net/tinyos/sim/*.class \
net/tinyos/sim/event/*.class \
net/tinyos/sim/lossy/*.class \
net/tinyos/sim/msg/*.class \
net/tinyos/sim/packet/*.class \
net/tinyos/sim/plugins/*.class \
net/tinyos/sim/script/*.class \net/tinyos/sim/script/reflect/*.class \
net/tinyos/sim/ui \
```

```
net/tinyos/sim/plugins/plugins.list \
net/tinyos/sf/*.class \
net/tinyos/util/*.class \
net/tinyos/packet/*.class \
net/tinyos/message/*.class \
net/tinyos/message/avrmote/*.class \
org/apache/oro/text/regex/*.class \
org/python/compiler/*.class \
org/python/core/*.class \
org/python/modules/*.class \
org/python/parser/*.class \
org/python/parser/ast/*.class \
org/python/rmi/*.class \
org/python/util/*.class)
rm -rfjarbuild-tmp
mkdirjarbuild-tmp
(cdjarbuild-tmp; jar xf ../simdriver-tmp.jar; jar xf ../$(ROOT)/jars/oalnf.jar; rm -rf
METAINF;
jarcmf ../simdriver.manifest ../simdriver.jar .)
rm -rf simdriver-tmp.jar jarbuild-tmp
jarclean:
rm -f simdriver.jar
```

- on lance la commande : **./tinyviz**

La fenêtre de tinyviz apparait comme suit :

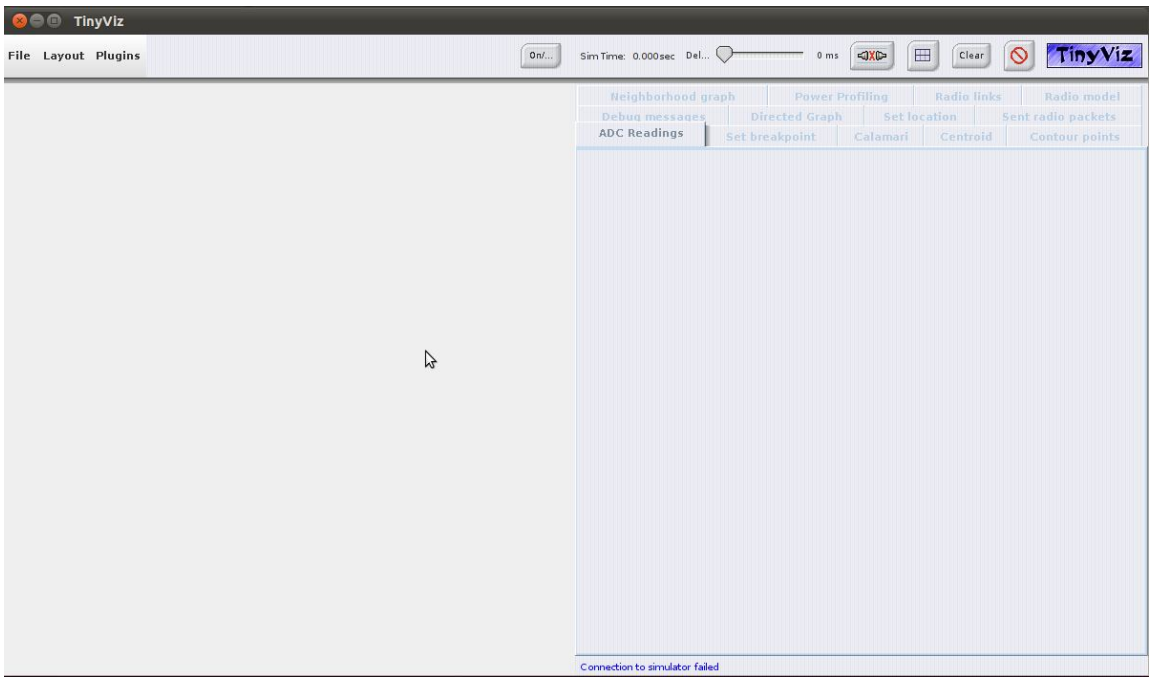


Figure A-6 Fenêtre TinyViz

Problèmes qui peuvent surgirent:

4- Normalement, la commande *makejarfile*crée toutes les classes spécifiées dans le makefile du répertoire sim, s'il y a des classes manquantes dans certains dossiers ou inexistantes dans d'autres, vous pouvez recopier les fichiers.java de chaque dossier où se trouve ce problème, modifier leurs extensions en .class et les mettre dans le dossier avec les fichiers.java tel que chaque dossier doit avoir ces fichiers.java et ces même fichiers.class. (la commande make sur /opt/tinyos-1.x/tools/java)
5- Les dossiers à corriger se trouvent tous cité dans le makefile du répertoire sim.
6- La commande *make clean* doit être exécutée qu'une seule fois, ça répétition efface les fichiers.class des dossiers du le répertoire sim.

3- L'interface TinyViz avec ses options :

Si on veut lancer la simulation de 30 nœuds capteurs avec l'interface graphique tinyviz, on peut exécuter les commandes suivantes :

```
cd /opt/tinyos-1.x/apps/TestTinyViz
make pc
export DBG=led
build/pc/main.exe 30
tinyviz -run build/pc/main.exe 30
```

Une fois TinyViz est lancé, on peut visualiser une fenêtre suivante :

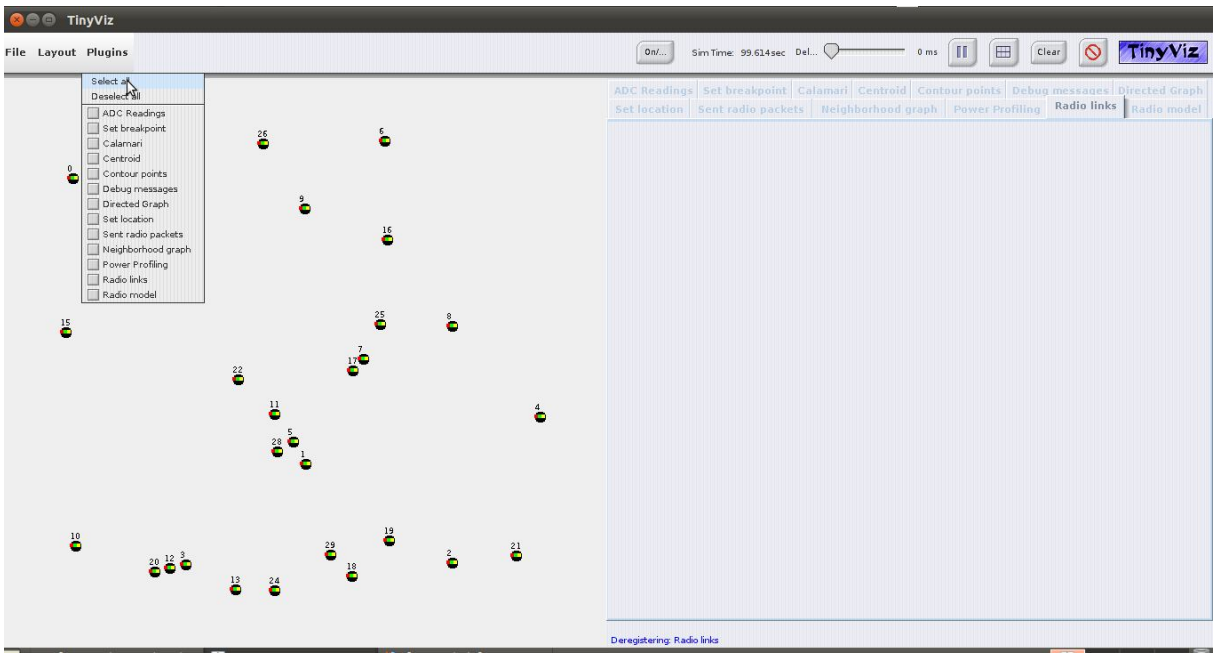


Figure A-7 l'exécution de TinyViz

Dans la partie gauche de cette figure, on distingue les capteurs qui sont déplaçables dans l'espace. Quant à la partie droite, on distingue les commandes permettant d'intervenir sur la simulation:

- *On/Off:* met en marche ou éteint un capteur.
- *Delay:* permet de sélectionner la durée au bout de laquelle se déclenche le timer.
- Play: permet de lancer la simulation où de la mettre en pause
- *Bouton de grilles:* affiche un quadrillage sur la zone des capteurs afin de pouvoir les situer dans l'espace.
- *Clear:* efface tous les messages qui avaient été affichés lors de la simulation.
- *Stop:* arrête la simulation et ferme la fenêtre.

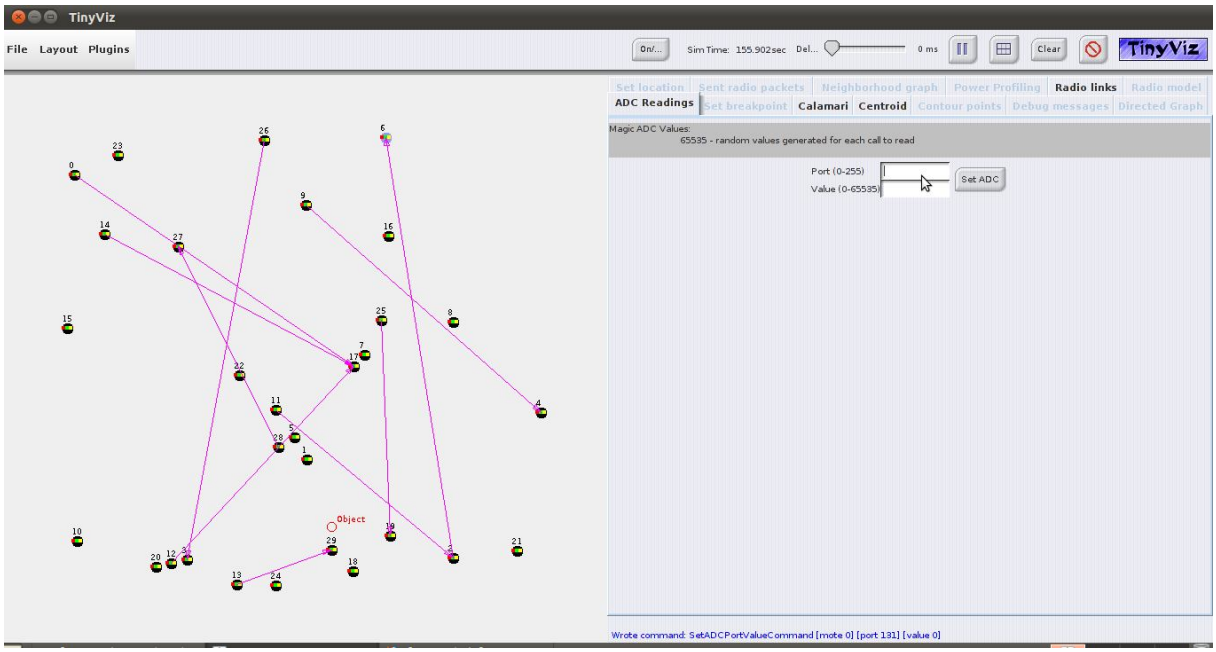
Pour lancer une application, il faut régler le **Delay** souhaité entre chaque application, choisir les plugins de visualisation que l'on souhaite, et, appuyer sur **Play**. La simulation démarre.

Chaque onglet contient un plugin qui permet de visualiser la simulation de façon plus ou moins détaillée.

Par exemple, en activant le plugin **Debug Messages**, tous les messages de type **Debug**apparaîtront dans l'onglet correspondant.

Le plugin **Radio Links** permet de visualiser graphiquement par des flèches, les échanges effectués entre les capteurs. Plus précisément, si un capteur envoie un broadcast, il sera repéré par un cercle. Par contre, s'il envoie un message direct (unicast) alors le lien de communication sera repéré par une flèche.

- Si on sélectionne le plugin **Radio Links** par exemple**:**



FigIV.8 Activation du plugin Radio model

- Pour pouvoir faire fonctionner le plugin debug message :

On lance les commandes suivantes :

```
cd .. /CntToLedsAndRfm
make pc
export DBG=led,am
tinyviz -run build/pc/main.exe 4
```

III- L'installation de TinyOS-2.1.1

Sur le site officiel de tinyOS, un article sur l'installation de la dernière version de tinyOS est disponible la V.2.1.1, nous nous sommes referez pour installer tinyos-2.1.1 :

1-sous terminal

- sudogedit /etc/apt/sources.list

1- *Depuis le fichier* /etc/apt/sources.list, *on rajoute les lignes suivantes* :

```
#tinyOS
deb http://tinyos.stanford.edu/tinyos/dists/ubuntu lucid main

#et sous ubuntu 10.10 en plus de la ligne précédente, on rajoute les deux lignes suivantes:
deb http://tinyos.stanford.edu/tinyos/dists/ubuntu edgy main
deb http://tinyos.stanford.edu/tinyos/dists/ubuntu feisty main
```

2- *on revient au terminal pour installer correctement le tinyos-2.1.1*:

```
sudo apt-get update
sudo apt-get install tinyos
sudo apt-get install tinyos-2.1.1
```

3- On complète le fichier ~/.bashrc ou ~/.profile en ajoutant les variables d'environnement suivantes :

- sudogedit ~/.bashrc

On ajoute les lignes suivantes :

```
# Java dépend de la version de java que vous disposer
export JDKROOT=/usr/lib/jvm/java-6-open
export JAVAXROOT=$JDKROOT

#Sourcing the tinyos environment variable setup script
source /opt/tinyos-2.1.1/tinyos.sh
```

2-Résultat de l'installation de tinyos-2.1.1 sous terminal

```
sabrina@sabrina-eMachines-E510:~$ su
Mot de passe :
Setting up for TinyOS 2.1.1
```

3-L'utilisation du simulateur TOSSIM

Afin de lancer le simulateur TOSSIM, certaines commandes doivent être tapez au préalable :

```
root@sabrina-eMachines-E510:~# cd /opt/tinyos-2.1.1/support/sdk/java/net/tinyos/sim

root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/support/sdk/java/net/tinyos/sim# make
clean
cleaning /opt/tinyos-2.1.1/support/sdk/java/net/tinyos/sim

root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/support/sdk/java/net/tinyos/sim# make
... /opt/tinyos-2.1.1/support/sdk/java/net/tinyos/sim
javac LinkLayerModel.java

root@sabrina-eMachines-E510:/opt/tinyos-2.1.1# echo $MAKERULES
/opt/tinyos-2.x/support/make/Makerules

root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/apps/Blink# make micaz
mkdir -p build/micaz
compiling BlinkAppC to a micaz binary
ncc -o build/micaz/main.exe -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -
target=micaz
-fnesc-cfile=build/micaz/app.c -board=micasb -DDEFINED_TOS_AM_GROUP=0x22 -
-param
max-inline-insns-single=100000 -DIDENT_APPNAME=\"BlinkAppC\"
-DIDENT_USERNAME=\"root\" -DIDENT_HOSTNAME=\"sabrina-eMachin\"
-DIDENT_USERHASH=0xefa7fbb4L -DIDENT_TIMESTAMP=0x4dcb213fL
-DIDENT_UIDHASH=0xbe14ec34L -fnesc-dump=wiring -fnesc-
dump='interfaces(!abstract())'
-fnesc-dump='referenced(interfacedefs, components)' -fnesc-
dumpfile=build/micaz/wiringcheck.
xml BlinkAppC.nc -lm
compiled BlinkAppC to build/micaz/main.exe
2052 bytes in ROM
51 bytes in RAM
avr-objcopy --output-target=srec build/micaz/main.exe build/micaz/main.srec
avr-objcopy --output-target=ihex build/micaz/main.exe build/micaz/main.ihex
writing TOS image
```

4-Problèmes rencontrés avec TOSSIM

En exécutant la commande

- root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/apps/Blink# make micazsim

On obtenait toujours une compilation qui se terminer avec des erreurs :

```
mkdir -p simbuild/micaz
make: python2.5-config : commandeintrouvable
make: python2.5-config : commandeintrouvable
make: python2.5-config : commandeintrouvable
placing object files in simbuild/micaz
writing XML schema to app.xml
compilingBlinkAppC to object file sim.o
ncc -c -shared -fPIC -o simbuild/micaz/sim.o -g -O0 -tossim -fnesc-nido-tosnodes=1000 -
fnescsimulate
-fnesc-nido-motenumber=sim_node\(\) -fnesc-gcc=gcc -Wall -Wshadow -Wnesc-all
-target=micaz -fnesc-cfile=simbuild/micaz/app.c -board=micasb
-DDEFINED_TOS_AM_GROUP=0x22 --param max-inline-insns-single=100000
-DIDENT_APPNAME=\"BlinkAppC\" -DIDENT_USERNAME=\"root\"
-DIDENT_HOSTNAME=\"sabrina-eMachin\" -DIDENT_USERHASH=0xefa7fbb4L
-DIDENT_TIMESTAMP=0x4dcb216bL -DIDENT_UIDHASH=0x4b4b683fL -Wno-nesc-
datarace
BlinkAppC.nc -fnesc-dump=components -fnesc-dump=variables -fnesc-dump=constants
-fnesc-dump=typedefs -fnesc-dump=interfacedefs -fnesc-dump=tags -fnesc-dumpfile=app.xml
compiling Python support and C libraries into pytossim.o, tossim.o, and c-support.o
g++ -c -shared -fPIC -o simbuild/micaz/pytossim.o -g -O0 -
DIDENT_APPNAME=\"BlinkAppC\"
-DIDENT_USERNAME=\"root\" -DIDENT_HOSTNAME=\"sabrina-eMachin\"
-DIDENT_USERHASH=0xefa7fbb4L -DIDENT_TIMESTAMP=0x4dcb216bL
-DIDENT_UIDHASH=0x4b4b683fL /opt/tinyos-2.1.1/tos/lib/tossim/tossim_wrap.cxx
-I/include/python2.5 -I/opt/tinyos-2.1.1/tos/lib/tossim -DHAVE_CONFIG_H
make: g++ : commandeintrouvable
make: *** [sim-exe] Erreur 127
```

- Ce qui indique quelques problèmes avec les paquets python et g++ qui ne trouve pas.

La version du python supportée par tossim est 2-6-6

- root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/apps/Blink# python --version

Python 2.6.6

Correction des erreurssurvenus:

- root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/apps/Blink# sudo apt-get install g++

Pour le problème des fichiers manquant du python 2.6 :

- sudo apt-get install python-devou python-devel

Et allez à :

- /usr / include / lib / python2.6 et copier tous les fichiers de ce dernier et les mettre dans :
- /opt/tinyos-2.1.1/tos/lib/tossim

Résultats:

```
root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/apps/Blink# makemicazsim
mkdir -p simbuild/micaz
placing object files in simbuild/micaz
writing XML schema to app.xml
compilingBlinkAppC to object file sim.o
……
compiling Python support and C libraries into pytossim.o, tossim.o, and c-
support.o
g++ -c -shared -fPIC -o simbuild/micaz/pytossim.o -g -O0 –
……..
copying Python script interface TOSSIM.py from lib/tossim to local directory
*** Successfully built micaz TOSSIM library.
```

5-Quelques exemples de commandes utilisées sous TOSSIM

- root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/tos/lib/tossim# cd/opt/tinyos-2.1.1/support/make
- root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/support/make# tos-check-env

Résultats:

```
Path:
/usr/local/sbin
/usr/local/bin
/usr/sbin
/usr/bin
/sbin
/bin
/usr/games

Classpath:
/opt/tinyos-2.1.1/support/sdk/java
--> WARNING: CLASSPATH may not include /opt/tinyos-
2.1.1/support/sdk/java/tinyos.jar. Please
ensure that /opt/tinyos-2.1.1/support/sdk/java/tinyos.jar is in your CLASSPATH or you
may
experience configuration problems
--> WARNING: CLASSPATH may not include '.' (that is, the symbol for the current
working
directory). Please add '.' to your CLASSPATH or you may experience configuration
problems.
```

rpms:
nesc:
/usr/bin/nescc
Version: nescc: 1.3.1

perl:
/usr/bin/perl
Version: v5.10.1 (*) built for i686-linux-gnu-thread-multi

flex:
bison:
java:
/usr/bin/java
--> WARNING: The JAVA version found first by tos-check-env may not be version 1.4 or version1.5one of which is required by TOS. Please ensure that the located Java version is 1.4 or 1.5

graphviz:
/usr/bin/dot
dot - graphviz version 2.26.3 (20100126.1600)
--> WARNING: The graphviz (dot) version found by tos-check-env is not 1.10. Please update your
graphviz version if you'd like to use the nescdoc documentation generator.

- tos-check-env completed with errors:

--> WARNING: CLASSPATH may not include /opt/tinyos-2.1.1/support/sdk/java/tinyos.jar. Please ensure that /opt/tinyos-2.1.1/support/sdk/java/tinyos.jar is in your CLASSPATH or you may experience configuration problems
--> WARNING: The JAVA version found first by tos-check-env may not be version 1.4 or version 1.5one of which is required by TOS. Please ensure that the located Java version is 1.4 or 1.5
--> WARNING: The graphviz (dot) version found by tos-check-env is not 1.10. Please update your graphviz version if you'd like to use the nescdoc documentation generator.

Versions de nesc, ncc et gcc :

- root@sabrina-eMachines-E510:/opt/tinyos-2.1.1/support/make# ncc --version

ncc: 1.2.4
nescc: 1.3.1
gcc: gcc (Ubuntu/Linaro 4.4.4-14ubuntu5) 4.4.5
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

6-Exemples d'applications

Chaque application possède un répertoire composé de 3 fichiers nécessaires :

1. Makefile
2. Un fichier de configuration de l'application
3. Un fichier contenant le module de l'application

- Makefile est le fichier qui est compilé lors de la commande **make**, il fait appel à d'autres fichiers

Premier programme sous TinyOs-2.1.1

Suivant un article sur l'installation de tinyos-2.1.1, un exemple de programmation a été proposé, nous l'avons testé:

1- Dans un nouveau fichier, on recopie le code suivant :

```
/* */
configurationSkelAppC { }
implementation
{ componentsMainC, SkelC;
SkelC ->MainC.Boot;
}
```

On le sauvegarde sous le nom suivant SkelAppC.nc

2- Dans un autre nouveau fichier, on recopie le code qui suit:

```
/* The SKEL application */
moduleSkelC
{
uses interface Boot;
} implementation
{ event void Boot.booted()
{
1;
}}
```

De même, on le sauvegarde sous le nom suivant SkelC.nc

3- En dernier lieu, on recopie le code suivant dans un nouveau fichier

```
COMPONENT=SkelAppC
include $(MAKERULES)
```

- Sauvegardé sous le nom Makefile

Vérification:

- Les fichiers "SkelC.nc, SkelAppC.nc et Makefile" doivent être dans /opt/tinyos-2.1.1/apps
- Dans un terminal, on compile les programmes

```
cd /opt/tinyos-2.1.1/apps
maketelosb
```

Résultats et remarques

- *Un exemple de programmation sous tinyos-2.1.1*

Suivant un tutorial de programmation en nesc, nous avons essayé un autre exemple d'application :

sous terminal:

- cd /opt/tinyos-2.1.1
- mkdir Simple
- cd Simple

1. Nous avons besoin de créer une un fichier configuration sous le nom de SimpleAppC.nc
   - sudogedit

```
configuration SimpleAppC{
} implementation{
componentsSimpleC, MainC;
SimpleC.Boot ->MainC.Boot;
}
```

- Il y a deux composants dans ce programme, le premier nommé :SimpleC et un composant Main nommé : MainC.
- Le composant MainC fournit le signal Boot.booted qui est essentiellement un point d'entrée dans l'application.

2. Nous avons besoin de crée un autre composant dans un fichier nommé SimpleC.nc. Il a la définition d'une implémentation du composant SimpleC.

- sudogedit

```
moduleSimpleC{
uses interface Boot;
}
implementation{
event void Boot.booted()
{
//The entry point of the program
}
}
```

3. Maintenant nous avons besoin de crée un fichier nommé Makefile ainsi le compilateur peut le compiler :

- sudogedit

```
COMPONENT=SimpleAppC
include $(MAKERULES)
```

Maintenant on peut le compiler avec la commande suivante:

- makemicaz

La grande partie de la réalisation de notre projet était la compréhension de l'environnement TinyOS. A travers beaucoup de pratiques résumées sur cette annexe nous nous somme consacré aux différentes étapes intéressantes et appropriées à l'installation de ce système d'exploitation TinyOS.

L'architecture du système TinyOS ; son mode d'exécution et son mode d'ordonnancement des différents types d'action qui sont évènement, commande et tâche seront explicités dans les annexes en fin du rapport.

Annexe B :

L'utilisation du simulateur TOSSIM

Compilez votre application TinyOS sous la plateforme PC prévu pour le simulateur TOSSIM en tapant :

$ make pc : Cela va construire un fichier exécutable / pc / main.exe

Après la compilation de l'application, vous pouvez exécuter l'application avec:

/ Build / pc / main.exe <options> <number_motes_to_sim>

Tous les messages de débogage seront affichés dans la fenêtre du terminal.

- **Utilisation des variables DBG**
  - Avant d'exécuter la simulation, vous devez définir la variable d'environnement DBG. Cette dernière vous permet de spécifier exactement les messages de débogage qui sont montrés. Cela vous permet de limiter les messages de débogage à des événements particuliers comme la radio, compteur.

Exemple:

```
$ export DBG=led
 $ ./build/pc/main.exe -t=20 1
SIM: Random seed is 546875
0: LEDS: Red on.
0: LEDS: Red off.
0: LEDS: Red on.
0: LEDS: Red off.
0: LEDS: Red on.
0: LEDS: Red off.
0: LEDS: Red on.
0: LEDS: Red off.
0: LEDS: Red on.
0: LEDS: Red off.
0: LEDS: Red on.
Simulation of 1 motes completed.
```

- **Ajoutant des instructions DEBUG pour CODE**

Les déclarations de débogage peuvent être facilement ajoutées à des applications

• Exemple:

dbg (DBG_TEMP, «Compteur: La valeur est% i \ n", (int) state); Ce qui affichera la valeur du compteur dans la fenêtre de commande.

Liste complète des modes de DBG à: TOS / types / dbg_modes.h

```
// $Id: dbg_modes.h,v 1.1.1.1 2007/11/05 19:10:43 jpolastre Exp $

/**
 * @author Philip Levis (derived from work by Mike Castelle)
 * @author Phil Levis (pal)
 */
#ifndef DBG_MODES_H
#define DBG_MODES_H

typedef long long TOS_dbg_mode;

#define DBG_MODE(x)  (1ULL << (x))

enum {
  DBG_ALL =          (~0ULL),   /* umm, "verbose"          */

/*====== Core mote modes =============*/
  DBG_BOOT =         DBG_MODE(0),    /* the boot sequence       */
  DBG_CLOCK =        DBG_MODE(1),    /* clock              */
  DBG_TASK =         DBG_MODE(2),    /* task stuff              */
  DBG_SCHED =        DBG_MODE(3),    /* switch, scheduling      */
  DBG_SENSOR =       DBG_MODE(4),    /* sensor readings              */
  DBG_LED =          DBG_MODE(5),    /* LEDs               */
  DBG_CRYPTO =          DBG_MODE(6), /* Cryptography/security        */
/*====== Networking modes ============*/
  DBG_ROUTE =        DBG_MODE(7),    /* network routing         */
  DBG_AM =           DBG_MODE(8),    /* Active Messages         */
  DBG_CRC =          DBG_MODE(9),    /* packet CRC stuff        */
  DBG_PACKET =       DBG_MODE(10),   /* Packet level stuff           */
  DBG_ENCODE =       DBG_MODE(11),   /* Radio encoding/decoding      */
  DBG_RADIO =        DBG_MODE(12),   /* radio bits                   */

/*====== Misc. hardware & system =====*/
  DBG_LOG =          DBG_MODE(13),   /* Logger component        */
  DBG_ADC =          DBG_MODE(14),   /* Analog Digital Converter     */
  DBG_I2C =          DBG_MODE(15),   /* I2C bus            */
  DBG_UART =         DBG_MODE(16),   /* UART                    */
  DBG_PROG =         DBG_MODE(17),   /* Remote programming      */
  DBG_SOUNDER =      DBG_MODE(18),   /* SOUNDER component            */
  DBG_TIME =            DBG_MODE(19),   /* Time and Timer components    */
  DBG_POWER =           DBG_MODE(20),   /* Power profiling      */

/*====== Simulator modes =============*/
  DBG_SIM =             DBG_MODE(21),   /* Simulator                    */
  DBG_QUEUE =           DBG_MODE(22),   /* Simulator event queue        */
  DBG_SIMRADIO =DBG_MODE(23),   /* Simulator radio model        */
  DBG_HARD =            DBG_MODE(24),   /* Hardware emulation           */
  DBG_MEM =             DBG_MODE(25),   /* malloc/free                  */
//DBG_RESERVED =DBG_MODE(26),   /* reserved for future use      */
```

```
/*====== For application use =========*/
  DBG_USR1 =          DBG_MODE(27),   /* User component 1        */
  DBG_USR2 =          DBG_MODE(28),   /* User component 2        */
  DBG_USR3 =          DBG_MODE(29),   /* User component 3        */
  DBG_TEMP =          DBG_MODE(30),   /* Temorpary testing use   */

  DBG_ERROR =         DBG_MODE(31),   /* Error condition         */
  DBG_NONE =          0,         /* Nothing                      */

  DBG_DEFAULT =      DBG_ALL          /* default modes, 0 for none    */
};
#define DBG_NAMETAB \
      {"all", DBG_ALL}, \
      {"boot", DBG_BOOT|DBG_ERROR}, \
      {"clock", DBG_CLOCK|DBG_ERROR}, \
         {"task", DBG_TASK|DBG_ERROR}, \
      {"sched", DBG_SCHED|DBG_ERROR}, \
      {"sensor", DBG_SENSOR|DBG_ERROR}, \
      {"led", DBG_LED|DBG_ERROR}, \
      {"crypto", DBG_CRYPTO|DBG_ERROR}, \
\
         {"route", DBG_ROUTE|DBG_ERROR}, \
         {"am", DBG_AM|DBG_ERROR}, \
         {"crc", DBG_CRC|DBG_ERROR}, \
         {"packet", DBG_PACKET|DBG_ERROR}, \
         {"encode", DBG_ENCODE|DBG_ERROR}, \
         {"radio", DBG_RADIO|DBG_ERROR}, \
\
      {"logger", DBG_LOG|DBG_ERROR}, \
         {"adc", DBG_ADC|DBG_ERROR}, \
         {"i2c", DBG_I2C|DBG_ERROR}, \
         {"uart", DBG_UART|DBG_ERROR}, \
         {"prog", DBG_PROG|DBG_ERROR}, \
         {"sounder", DBG_SOUNDER|DBG_ERROR}, \
         {"time", DBG_TIME|DBG_ERROR}, \
         {"power", DBG_POWER|DBG_ERROR}, \
\
         {"sim", DBG_SIM|DBG_ERROR}, \
         {"queue", DBG_QUEUE|DBG_ERROR}, \
         {"simradio", DBG_SIMRADIO|DBG_ERROR}, \
         {"hardware", DBG_HARD|DBG_ERROR}, \
         {"simmem", DBG_MEM|DBG_ERROR}, \
\
         {"usr1", DBG_USR1|DBG_ERROR}, \
         {"usr2", DBG_USR2|DBG_ERROR}, \
         {"usr3", DBG_USR3|DBG_ERROR}, \
         {"temp", DBG_TEMP|DBG_ERROR}, \
      {"error", DBG_ERROR}, \
\
         {"none", DBG_NONE}, \
         { NULL, DBG_ERROR }
#define DBG_ENV       "DBG"
#endif
```

Options de main.exe

Les options disponibles pour TOSSIM (main.exe) sont :

Usage: / Build / pc / main.exe [options] num_nodes

Tel que les [options] sont les suivantes:

- **-h, - help**: afficher l'aide ;
- **-gui:** la simulation en mode pause et attendre la fênetre GUI pour se connecter ;
- **-a = <model>:** spécifie le modèle ADC ;
- **-b = <sec>:** Motes démarrés en <sec> secondes (par défaut 10) ;
- **-ef = <file>:** l'utilisation <file> EEPROM, autrement un fichier anonyme est utilisé ;
- **-l = <scale>:** lancez sim à <scale> en temps réel ;
- **-r = <modèle>:** précise le modèle de radio (simple par défaut), ex: simple, static, lossy ;
- **-rf = <fichier>:** spécifie le fichier d'entrée pour le modèle lossy (par défaut lossy.nss) ;
- **-s = <num>:** <num> démarrage de nœuds ;
- **-t = <sec>:** faire marcher la simulation pour <sec> seconde virtuelles ;
- **-num_nodes:** nombre de nœuds à simuler ;
- **all:** permettre tous les messages disponibles ;
- **boot:** simuler le démarrage et StdControl ;
- **clock:** l'horloge matérielle ;
- **Task:** Les tâche enqueueing / dequeueing / running;
- **sched:** l'ordonnanceur TinyOS ;
- **sensor:** lectures des capteurs ;
- **LED:** les leds des motes ;
- **Crypto:** les opérations de chiffrement (par exemple, TinySec) ;
- **route:** les systèmes de routage ;
- **am:** active la transmission/ réception des messages ;
- **crc:** contrôle CRC sur des messages actifs ;
- **packet:** transmission / réception au niveau des paquets ;
- **code:** paquet de codage / décodage ;
- **radio:** la radio ops de bas niveau: bits et octets ;
- **logger:** stockage non volatile ;
- **adc:** l'ADC ;
- **i2c:** le bus I2C ;
- **uart:** l'UART (bus de série) ;
- **prog:** la programmation réseau ;

- **sounder:** la sirène sur la carte de capteurs de mica ;
- **usr1:** le mode de sortie utilisateur 1 ;
- **usr2:** le mode de sortie utilisateur 2 ;
- **usr3:** le mode de sortie utilisateur 3 ;
- **Temp:** pour une utilisation temporaire ;

Pour plus d'éclaircissement et de détails sur la compilation et l'exécution de la simulation sous TOSSIM comme : les différents modes dbg, la surveillance de réseau et l'injection de paquets, les modèles radio comme l'utilisation de LossyBuilder, les erreurs sur les bits et les erreurs de paquets, l'actionnement modèle Lossy, etc, se référez au pdf *TOSSIM: A Simulator for TinyOS Networks* se trouvant sur le chemin:/opt/tinyos-1.x/docs/nido.pdf7

TOSSIM TINYVIZ

TinyViz fournit une interface graphique pour le débogage, la visualisation et l'interaction avec des simulations TOSSIM. TinyViz n'est pas un visualiseur, c'est un cadre dans lequel les plugins fournissent des fonctionnalités souhaité, comme une visualisation du trafic réseau, etc

On peut utiliser la variable DBG en conjonction avec TinyViz

Exemple : $ export DBG = "sim build / pc / main.exe -gui 10"

TOSSIM SIMDRIVER

SimDriver est une plate-forme extensible pour interagir avec TOSSIM / Nido qui est le simulateur de TinyOS. Il est l'évolution de la TinyViz, l'interface graphique GUI qui vous permet de visualiser et de déboguer le fonctionnement des programmes de TinyOS.

SimDriver intègre un environnement de script qui vous permet d'exécuter des scénarios et d'interagir avec le simulateur en cours. GUI est un composant optionnel, permettant à la fois le mode assisté et le fonctionnement en mode batch.

Pour compiler SimDriver

1) Assurez-vous que vous avez compilé les fichiers dans net / TinyOS / message:

$ cd net/tinyos/message
$ make

2) Il suffit de taper "make" dans ce répertoire:

$ cd net/tinyos/sim
$ make

3) SimDriver peut être compilé en un fichier JAR exécutable qui contient toutes les classes, les images, et autres composants nécessaires à son exécution.

L'utilisation de ce fichier JAR élimine toute inquiétude sur la configuration CLASSPATH; car le fichier JAR contient toutes les classes nécessaires.

- Pour construire le fichier JAR, tapez:
  $make jarfile

Pour exécuter SimDriver

1) Démarrez votre simulation avec l'option "-gui" option ligne de commande. Par exemple :

$ apps cd / CndToLeds
$ make pc
$. / build / pc / main.exe-gui 10

La simulation ne commencera pas avant la fenêtre GUI soit connectée à elle. On aura cet affichage : SIM: socket serveur Créer en attente d'une connexion client sur le port 10840.

2) Dans une autre fenêtre, démarrer SimDriver:

$ java-jar simdriver.jar-gui

La fenêtre TinyViz apparaîtra. En appuyant sur "play" le bouton débutera la simulation; appuyant sur «pause» le bouton causera une pause. Faites glisser le "delay" le curseur va ralentir la vitesse à laquelle les événements sont traitées de la simulation.

Actuellement, l'emplacement des motes dans l'affichage TinyViz est dénué de sens. TinyViz place les motes dans des endroits aléatoires sur l'écran. Vous pouvez déplacer le motes tout autour; finalement vous pouvez envoyer vos indications à TOSSIM lui-même comme une «vraie» localisation des motes.

Pour obtenir des informations utiles à partir de l'interface, vous devez activer un ou plusieurs "plugins", en les sélectionnant dans le menu Plugins. Par défaut, il y a plusieurs plugins inclus sous TinyViz comme:

DebugMsgPlugin - Affiche les messages de débogage dans une fenêtre

RadioLinkPlugin - Visualise la connectivité radio des Motes

MotePacketPlugin - Voir les messages radio envoyés par les motes

BreakpointPlugin - Cause à l'interface GUI pour faire une pause lorsqu'un événement survient.

L'activation d'un plugin à partir du menu "Plugins" vous permet de configurer ce plugin en utilisant les contrôles sur la fenêtre de droite.

Par exemple, avec DebugMsgPlugin activé, vous pouvez sélectionner motes de l'affichage et cliquez sur la boîte "motes sélectionnés uniquement" pour voir seulement les messages de débogage des motes sélectionnés.

Notez que les messages de débogage reçu par TinyViz sont ceux sélectionnés en utilisant la variable 'DBG' environnement lorsque vous exécutez TOSSIM. Donc, si vous voulez voir LEDS et AM messages de débogage dans TinyViz, vous avez besoin pour commencer TOSSIM l'utilisation de :

DBG=led,am ./build/pc/main.exe -gui 10

- Par exemple, pour exécuter la démonstration de l'application Surge3 dans TinyViz:

1) On compile le code d'application Surge:

$ cd broken/experimental/mdw/surge3/apps/Surge
$ make pc

2) compiler l'application Surge Java GUI:

$ cd ../../tools/java/net/tinyos/surge
$ make pc

3) Lancez l'application Surge sur TOSSIM:

$ DBG=usr1 /build/pc/main.exe -gui 10

4) Dans une fenêtre séparée, démarrer TinyViz:

$ java -jar tinyviz.jar

5) Enfin, dans une autre fenêtre, lancer l'interface graphique de Surge :

$ java net.tinyos.surge.MainClass 0x11 (Remplacez '0 x11 'avec l'ID du groupe AM utilisé dans l'application Surge elle-même.)

- Il est important de démarrer les applications dans cet ordre, puisque TinyViz doit se connecter à Surge, et l'interface graphique GUI de Surge doit se connecter à TinyViz

6) Cliquez sur le bouton 'play' au TinyViz. Vous devriez voir les motes. Activer plugins si nécessaire.

7) Cliquez sur "balise racine Démarrer" dans le GUI de surtension. Cela va démarrer la balise racine, provoquant les motes de Surge à «réveiller» et commencer à envoyer paquets.

Annexe C :

Format des messages sous TinyOS-1.x

I-Messages de TinyOS:

Pourenvoyer un message dans le réseau, on utilise la structure TOS_Msg de TinyOS. Cette dernière se trouve dans le fichier *opt/tinyos-1.x/tos/types/am.h*

Elle est définie comme suit :

```
/**
* @author Jason Hill
 * @author David Gay
 * @author Philip Levis
 * @author Chris Karlof
 */
#ifndef AM_H_INCLUDED
#define AM_H_INCLUDED
enum {
  TOS_BCAST_ADDR = 0xffff,
  TOS_UART_ADDR = 0x007e,
};

#ifndef DEF_TOS_AM_GROUP
#define DEF_TOS_AM_GROUP 0x7d
#endif

enum {
  TOS_DEFAULT_AM_GROUP = DEF_TOS_AM_GROUP
};

uint8_t TOS_AM_GROUP = TOS_DEFAULT_AM_GROUP;

#ifndef TOSH_DATA_LENGTH
#define TOSH_DATA_LENGTH 29
#endif

#ifndef TOSH_AM_LENGTH
#define TOSH_AM_LENGTH 1
#endif

#ifndef TINYSEC_MAC_LENGTH
#define TINYSEC_MAC_LENGTH 4
#endif
```

```
#ifndef TINYSEC_IV_LENGTH
#define TINYSEC_IV_LENGTH 4
#endif

#ifndef TINYSEC_ACK_LENGTH
#define TINYSEC_ACK_LENGTH 1
#endif

typedefstructTOS_Msg
{/* The following fields are transmitted/received on the radio. */
uint16_taddr;
uint8_t type;
uint8_t group;
  uint8_t length;
int8_t data[TOSH_DATA_LENGTH];
uint16_tcrc;
  /* The following fields are not actually transmitted or received
   * on the radio! They are used for internal accounting only.
   * The reason they are in this structure is that the AM interface
   * requires them to be part of the TOS_Msg that is passed to
   * send/receive operations.
   */
uint16_t strength;
uint8_t ack;
  uint16_t time;
uint8_tsendSecurityMode;
uint8_treceiveSecurityMode;
} TOS_Msg;
typedefstructTOS_Msg_TinySecCompat
{
  /* The following fields are transmitted/received on the radio. */
uint16_t addr;
  uint8_t type;
// length and group bytes are swapped
uint8_t length;
uint8_t group;
int8_t data[TOSH_DATA_LENGTH];
uint16_tcrc;

  /* The following fields are not actually transmitted or received
   * on the radio! They are used for internal accounting only.
   * The reason they are in this structure is that the AM interface
   * requires them to be part of the TOS_Msg that is passed to
   * send/receive operations.
   */
uint16_t strength;
uint8_t ack;
  uint16_t time;
uint8_tsendSecurityMode;
uint8_treceiveSecurityMode;
} TOS_Msg_TinySecCompat;
```

```
typedefstructTinySec_Msg
{ uint16_t addr;
uint8_t type;
uint8_tlength;
// encryption iv
uint8_t iv[TINYSEC_IV_LENGTH];
  // encrypted data
uint8_tenc[TOSH_DATA_LENGTH];
  // message authentication code
uint8_t mac[TINYSEC_MAC_LENGTH];
  // not transmitted - used only by MHSRTinySec
  uint8_t calc_mac[TINYSEC_MAC_LENGTH];
  uint8_t ack_byte;
boolcryptoDone;
boolreceiveDone;
  // indicates whether the calc_mac field has been computed
boolMACcomputed;
} __attribute__((packed)) TinySec_Msg;

enum {
  MSG_DATA_SIZE = offsetof(structTOS_Msg, crc) + sizeof(uint16_t), //
36 by default
  TINYSEC_MSG_DATA_SIZE = offsetof(structTinySec_Msg, mac) +
TINYSEC_MAC_LENGTH, // 41 by default
  DATA_LENGTH = TOSH_DATA_LENGTH,
  LENGTH_BYTE_NUMBER = offsetof(structTOS_Msg, length) + 1,
  TINYSEC_NODE_ID_SIZE = sizeof(uint16_t)
};
enum {
  TINYSEC_AUTH_ONLY = 1,
  TINYSEC_ENCRYPT_AND_AUTH = 2,
  TINYSEC_DISABLED = 3,
  TINYSEC_RECEIVE_AUTHENTICATED = 4,
  TINYSEC_RECEIVE_CRC = 5,
  TINYSEC_RECEIVE_ANY = 6,
  TINYSEC_ENABLED_BIT = 128,
  TINYSEC_ENCRYPT_ENABLED_BIT = 64
} __attribute__((packed));
typedefTOS_Msg *TOS_MsgPtr;
uint8_t TOS_MsgLength(uint8_t type)
{
#if 0
uint8_t i;
for (i = 0; i < MSGLEN_TABLE_SIZE; i++)
if (msgTable[i].handler == type)
returnmsgTable[i].length;
#endif
returnoffsetof(TOS_Msg, strength);
}
#endif
```

Le paquet de données TOS_Msgde TinyOS est décrit dans le tableau suivant [ann1]:

Address		Message Type	Group ID	Data Length	Data	CRC	
0	1	2	3	4	5...n-2	n-1	n

Tel que, le schéma suivant explicite chaque champ de la structure Tos_Msg :

Octets	Champ	Description
0,1	Adresse du message ou Message Address	L'une des trois types de valeurs possibles: - Adresse de diffusion (0xFFFF) - un message à tous les nœuds. - Adresse UART (0x007E) - un message d'un noeud vers port série de la passerelle. Tous les messages entrants auront cette adresse. - Adresse du nœud - le numéro d'identification unique d'un nœud pour recevoir les messages.
2	Type du message ou Message Type	Message actif (AM) identifiant unique pour le type de message qu'il est. Typiquement, chaque application aura son propre type de message. Voici quelques exemples: - AMTYPE_XUART = 0x00 - AMTYPE_MHOP_DEBUG = 0x03 - AMTYPE_SURGE_MSG = 0x11 - AMTYPE_XSENSOR = 0x32 - AMTYPE_XMULTIHOP = 0x33 - AMTYPE_MHOP_MSG = 0xFA
3	ID du groupe ou Group ID	L'identifiant unique pour le groupe de motes(capteurs) participant au réseau. La valeur par défaut est 125 (0x7d).Seulement les motesavec le même ID du groupe pourront parler les uns aux autres.
4	Longueur des données ou Data Length	La longueur (l) en octets de la charge utile des données. Ceci n'inclut pas le CRC ou la trame de synchronisation "synch" en octets.
5…n-2	Données utiles ou Payload data	Le contenu du message actuel. Les données résident à l'octet 5 , plus la longueur des données (l ci-dessus). Les données seront spécifiques au type de message.
n-1, n	CRC (CyclicRedundancy Code)	Deux octets qui assurent l'intégrité du message.

II-Récupération des messages sortants des nœuds capteurs :

On lançant l'exécution d'une application quelconque se trouvant dans *opt/tinyos-1.x/apps,* on peut récupérer les messages échangés entre les nœuds capteurs, et cela on tapant :
$ export DBG=am //pour active message

Suivant le tutorial se trouvant sur le chemin *opt/tinyos-1.x/doc/tutorial/lesson6.html,* un exemple est offert à notre disposition :

- Si par exemple on a un paquet:

```
7e 00 0a 7d 1a 01 00 0a 00 01 00 46 03 8e 03 96 03 96 03 96 03 97 03 97 03
97 03 97 03 97 03
7e 00 0a 7d 1a 01 00 14 00 01 00 96 03 97 03 97 03 98 03 97 03 96 03 97 03
96 03 96 03 96 03
7e 00 0a 7d 1a 01 00 1e 00 01 00 98 03 98 03 96 03 97 03 97 03 98 03 96 03
97 03 97 03 97 03
```

- Nous pouvons donc interpréter les données par paquets comme suit:

destaddr	handlerID	groupID	msglen	source addr	counter	channel	readings
7e 00	0a	7d	1a	01 00	14 00	01 00	96 03 97 03 97 03 98 03 97 03 96 03 97 03 96 03 96 03 96 03

Ainsi on a :
- Adresse de destination (2 octets)
- Handler ID (1 octet)
- Group ID (1 octet)
- La longueur du message (1 octet)
- Données utiles ou Payload(29 octets)
  - Source ID (2 octets)
  - compteur simple (2 octets)
  - canal ADC (2 octets)
  - Lectures de données ADC (10 lectures de 2 octets chacun)

Un autre exemple sur le site :http://cs.acadiau.ca/~shussain/wsn/apps/tos1/

0: Sending message: ffff, a
7e 00 0a 88 18 00 00 00 00 88 00 a6 03 2a 02 be 03 f3 00 6b 00 8b 00 21 03 79 00 ff 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Les champs de paquet sont décrits comme suit:
- 00 7e - adresse de destination (UART)
- 0a - handler ID
- 88 - Groupe ID

- 18 - la longueur du message
- 00 00 - adresse de la source
- 00 00 - nombre de paquets
- 00 88 - lecture 0 (0000 0000 1000 1000 - RGY = 000, les trois derniers bits, comme indiqué par des LED)
- 03 A6 - lecture 1 (0000 0011 1010 0110 - RGY = 110, les trois derniers bits, comme indiqué par des LED)
- 02 2a - lecture 2 (0000 0010 0010 1010 - RGY = 010, les trois derniers bits, comme indiqué par des LED)
- la LED rouge est le moins significative des bits, tandis que le jaune est la plus importante.Le LED rouge s'allume lorsque la lecture du capteur est de plus qu'un un certain seuil. La LED jaune est activée chaque fois qu'un paquet est envoyé sur le port série.

III- Trace récupéré de l'application Surge :

Dans TinyOS, deux algorithmes de routage de base sont la diffusion radio (Bcast) et le routage multi- sauts (MultiHopRouter) comme l'illustre la figure suivante (Fig C-1).
Surge est un exemple d'applicationTinyOS qui utilise le routage ad-hoc MultiHop. Elle est conçue pour être utilisé en conjonction avec l'outil Surge java. Chaque nœud Surge prend des lectures de lumière et les transmet à une station de base. Le nœud peut également répondre à des commandes de diffusion de la base (broadcast)

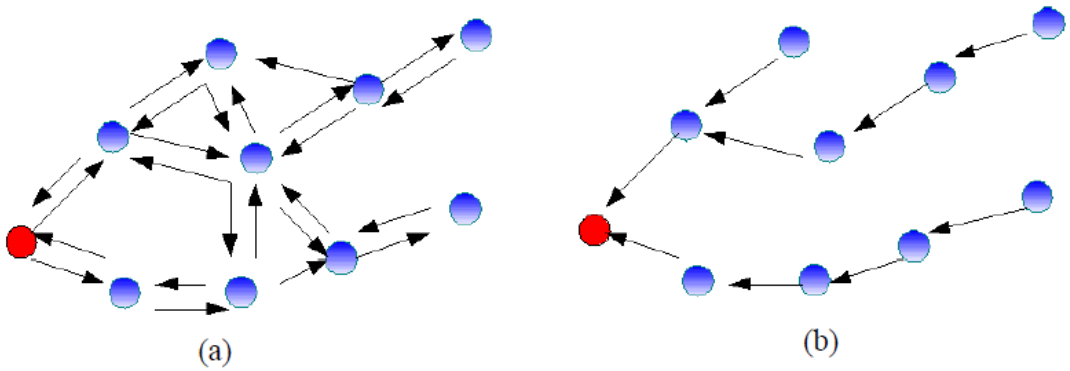


Fig. C-1: types de base de routage dans TinyOS (Surge). Radiodiffusion (a), multi-sauts (b)

- Exécution de l'application Surge dans l'environnement simulé TinyOS qui est lancé depuis les lignes de commande

$ cd opt/tinyos-1.x/apps/Surge
$ make pc
$ export DBG=route //on peut utiliser également DBG=packer, usr1,radio,usr2,sim,etc.
$ ./build/pc/main.exe 5

La trace récupérée de l'exécution de l'application Surge pour le noeud 2 est :

```
4: MultiHopLEPSM Sending route update msg.
2: MultiHopLEPSM timer task.
2: MultiHopLEPSM: Updating Nbr 3. ExpTotl = 5, rcvd= 43, missed = 0
2: MultiHopLEPSM: Updating Nbr 4. ExpTotl = 5, rcvd= 43, missed = 0
2: MultiHopLEPSM: Updating Nbr 1. ExpTotl = 5, rcvd= 39, missed = 0
2: MultiHopLEPSM: Updating Nbr 0. ExpTotl = 5, rcvd= 4, missed = 0

2:  addr    prnt    misd    rcvd    lstS    hop     rEst    sEst
2:  3       0       0       0       43      1       255     255
2:  4       0       0       0       43      1       255     255
2:  1       255     0       0       38      255     255     0
2:  0       126     0       0       4       0       204     25
```

- **addr**– addresse (ID),
- **prnt**– parent,
- **misd**–paquetsmanquants,
- **rcvd** – packets reçus ,
- **lstS**– dernier num de séquence,
- **hop** – hop count (255 if not existing link),
- **rEst**– receiving estimation factor (255 is best),
- **sEst**– sending estimation factor (255 is best).

- Si on veut voir tous les messages actifs reçus. L'adresse de diffusion est égal 0xFFFF :

  $ export DBG = am

```
0: Received message:
    ff ff 11 7d 0c 02 00 02 00 1f 00 ff 00 74 03 ff ff 00 02 00 00 01 00
    00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 bc 5a 00 00 0:
0: AM_type = 17
3: AM_address = ffff, 11; counter:2
3: Received message:
    ff ff 11 7d 0c 02 00 02 00 1f 00 ff 00 74 03 ff ff 00 04 00 00 02 00
    00 01 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 bc 5a 00 00 3:
3: AM_type = 17
2: AM report send done for message to 0xffff, type 17.
ff ff fa 7d 16 02 00 02 00 20 00 ff ff 00 04 04 00 00 03 00 00 01 00 00
00 00 00 00 00 00 00 00 00 00 4e c7 00 00 00 00 94 85 00 00 2:
2: Sending message: ffff, fa
    1: AM_address = ffff, fa; counter:3
1: Received message:
    ff ff fa 7d 16 02 00 02 00 20 00 ff ff 00 04 04 00 00 03 00 00 01 00
    00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 af 92 00 00 1:
1: AM_type = 250
```

ZigBee est un standard de communication sans fil à bas coût pour échanger des données issues d'équipements sans fil simples et faible consommation énergétique dans un milieu industriel.

Avec la convergence de l'informatique, de l'électronique et des télécommunications, il est ainsi possible de mettre en place des réseaux de capteurs sans fil dans un contexte de contrôle industriel.

ZigBee permet de combler plusieurs domaine d'application : surveillance de locaux pour la détection de départ de feu, surveillance de bâtiments contre les intrusions, Gestion Technique de Bâtiment (GTB), aide à la personne, assistance aux personnes…

Ce standard est promu par l'alliance ZigBee qui regroupe un ensemble d'industriels travaillant sur l'élaboration de spécifications afin de pouvoir développer des applications sans fil à faible consommation et sécurisées.

Comme tout standard, ZigBee permet à l'utilisateur final de cette technologie d'être constructeur indépendant ; ce qui lui permet d'acheter ses modules compatibles ZigBee auprès de n'importe quel fournisseur.

La norme ZigBee s'appuie sur la norme IEEE 202.15.4. Elle est librement téléchargeable sur le site de l'Alliance ZigBee : **http://www.zigbee.org/**

I . Architecture protocolaire de ZigBee :

ZigBee s'appuie sur la norme IEEE 802.15.4 pour les niveaux physique et MAC (*Medium ACcess*).

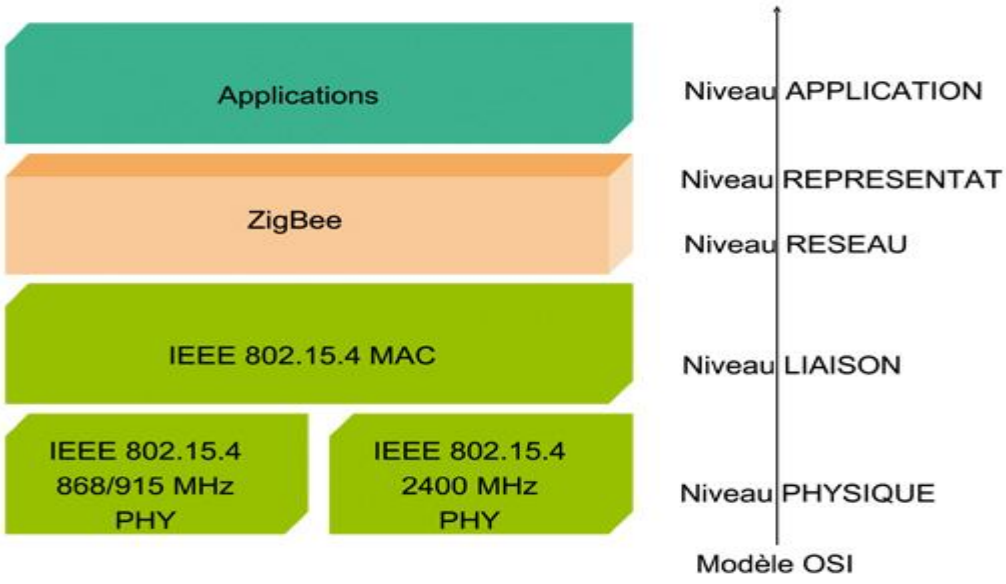


Figure D.1 : la pile protocolaire ZigBee

I.1 Couche physique IEEE 802.15.4

I.1.1 Bandes de fréquences et canaux

La couche physique IEEE 802.15.4 est généralement prise en charge par le module radio. Elle offre quatre débits différents. Le tableau1 résume les débits proposés selon la fréquence et la modulation.

Fréquence (MHz)	Modulation	Débit (kb/s)
868/868.6	BPSK	20
868/868.6	ASK	250
868/868.6	O-QPSK	100
902/928	BPSK	40
902/928	ASK	250
902/928	O-QPSK	250
2400/2483.5	O-QPSK	250

Tab. 1 – Le débit en fonction de la fréquence et de la modulation.

Dans la bande de fréquences des 2.4 GHz, le débit est donc de 250 Kb/s, il lui est associé un seuil de réception qui va jusqu’`a −92 dBm.
Avec les trois plages de fréquences, la couche physique offre 27 canaux de transmission différents dont :

- **16** sur la plage de fréquences de **2400/2483.5 MHz** séparés de 5 MHz,
- **10** sur la plage de fréquences de **902/928** séparés de 2 MHz
- **1 canal** sur la plage de fréquences de **868/868.6 MHz**.

L’usage de ces canaux dépend de la législation des pays dans lesquels des solutions conformes à ce standard sont utilisées (voir la figure D.2 et D.3).

La norme IEEE 802.15.4 supporte les 3 bandes ISM (*Industrial*, *Scientific*, *Medical*) de 868 MHz, 915 MHz et 2,4 GHz.

	Bande	Usage	Débit	Nombre de canaux
2.4 GHz	ISM	Mondial	250kb/s	16
915 MHz	ISM	Amérique	40kb/s	10
868 MHz	ISM	Europe	20kb/s	1

Figure D.2 : Bandes de fréquence pour ZigBee

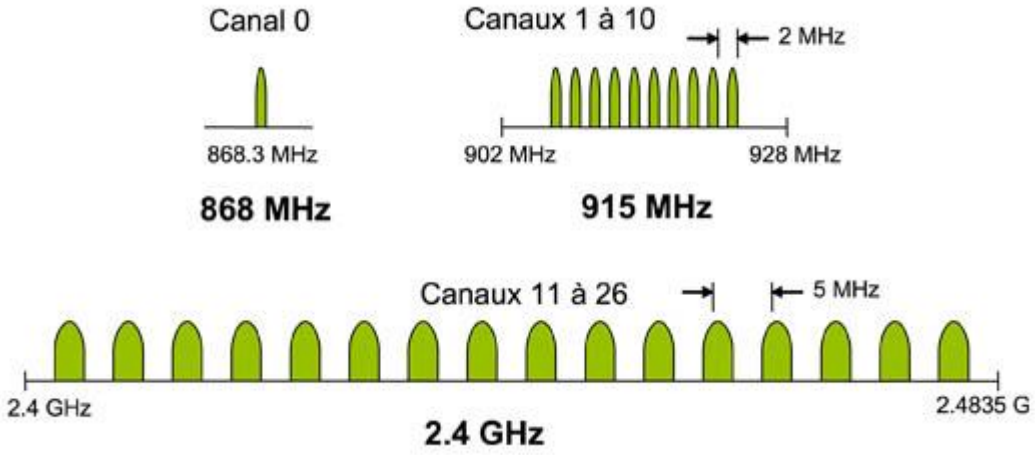


Figure D.3 : Spectre fréquentiel pour ZigBee

I.1.2 Le paquet de niveau physique

La norme prévoit un paquet de niveau physique représenté sur la figure 2.9. Ce paquet comprend un en-tête de synchronisation, un en-tête PHY et les données PHY(voir figure D.4). L'en-tête de synchronisation comprend 6 octets : un préambule d'une longueur de 5 octets qui permet au récepteur de parfaire sa synchronisation et un fanion de START (délimiteur début paquet) sur 1 octet. Ce fanion rompt l'alternance des bits transmis lors du préambule, indiquant ainsi l'imminence de la transmission de données. Après l'en-tête de synchronisation vient l'en-tête PHY dont le rôle est de spécifier la longueur du paquet. Les données suivent cet en-tête, 127 octets maximum par paquet, soit une durée de transmission maximum de 4,256 ms par paquet sur la couche PHY2450 (250 kbits/s).

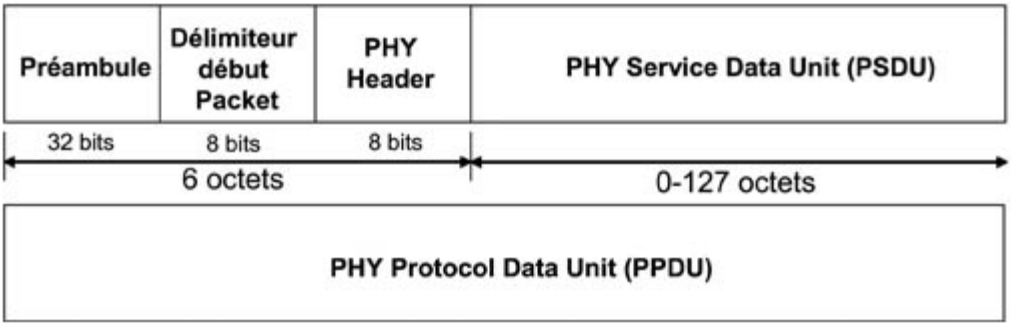


Figure D. 4 : Structure d'un paquet de la couche physique IEEE 802.15.4

I.1.3 Fonctionnalités du niveau physique

Le niveau physique gère les fonctionnalités suivantes :

- Activation/désactivation de l'interface radio.
- Détection de l'énergie dans le canal. Choix du canal radio.
- Qualité du lien radio.
- Évaluation du canal pour la mise en œuvre du protocole d'accès CSMA (Test d'occupation du medium).
- Émission/réception des paquets dans le canal radio.

I.1.3.1 Activation et désactivation du module radio

Le module radio possède trois états de fonctionnement : un état de transmission, un état de réception et un état de sommeil. Le temps de changement d'´etat entre transmission et réception ne doit pas dépasser les 192 µs. Cet état est piloté par la couche MAC. Il est essentiel pour économiser de l'´energie de mettre le module en mode sommeil.

I.1.3.2 Indication de la qualité du lien

Le LQI (Link Quality Indication) caractérise la qualité d'un lien à un instant donné suite à une réception d'une trame. Ce paramètre est essentiel pour les protocoles des couches réseau et application. Par exemple, un protocole de routage peut exploiter cette indication afin d'identifier les meilleurs liens à utiliser dans son choix de routes.

I.1.3.3 Test d'occupation du médium ou CCA (Clear Channel Assessment)

Le CCA permet de savoir l'´etat du canal radio. Il est essentiel pour le fonctionnement de l'algorithme de CSMA/CA de la couche MAC. La couche physique est capable d'effectuer trois modes de CCA différents :

- La détection d'un signal avec une puissance reçue supérieure à un certain seuil.
- La détection d'un signal conforme à la modulation de la couche physique.
- La détection d'un signal qui répond aux deux conditions.

A noter que le seuil de détection d'activité est typiquement -95 dBm.

I.1.3.4 Sélection du canal

Comme la couche physique offre plusieurs canaux de transmission, il est nécessaire de sélectionner un canal précis, ceci à la demande des couches supérieures. Le changement de

canal est aussi fait implicitement par la couche physique suite à une demande de scrutation sur l'ensemble des plages de fréquences chacune constituant un canal de transmission. Cette action est appelée un scan.

I.1.4 Services rendus

La couche PHY de 802.15.4 rend deux services :

- un service de données PHY (PHY data service), qui permet l'´emission et la réception de PPDU sur le médium radio,
- un service de gestion PHY (PHY management service), qui permet l'interfaçage entre le logiciel et l'entité de gestion de la couche physique. Les capacités de cette entité sont les suivantes (liste exhaustive) : activation/désactivation du transceiver, ED, LQI sur les paquets reçus, sélection du canal, CCA3 pour le CSMA/CA de la sous-couche MAC.

I.2 La sous-couche MAC

La couche MAC 802.15.4 supporte deux modes de fonctionnement selon les besoins applicatifs : le mode suivi de beacon et le mode non suivi de beacon. En mode suivi de beacon, le coordinateur envoie périodiquement un beacon pour synchroniser l'activité des entités qui lui sont attachées selon une structure de supertrame donnée sur la figure 4.1. En mode non suivi de beacon, les beacons ne sont utilisés que pour la découverte du réseau.

Par la suite, nous allons décrire les fonctionnalités de gestion essentielles de la couche MAC 802.15.4 :

- l'accès au médium,
- les scans, la création du réseau et l'association,
- la synchronisation avec un coordinateur,
- les échanges de trames.

I.2.1 Accès au médium

Un coordinateur limite l'accès au médium en utilisant la diffusion de beacon délimitant des supertrames. La structure de la supertrame est définie par les deux paramètres BI (Beacon Interval) qui définit l'intervalle qui sépare deux beacons consécutifs et SD (Superframe Duration).

La portion active de la supertrame est découpée en aNumSuperframeSlots, slots de temps égaux et est composée de trois parties : *le beacon, la CAP* (Contention Access Period) et la *CFP* (Contention Free Period).

Durant la CAP toutes les stations sont en compétition pour accéder au médium alors que la CFP est constituée de slots de temps, appelés GTS (Guaranteed Time Slot) alloués à chaque station pour communiquer avec le coordinateur.

Le début du premier slot est l'instant d'envoi du beacon. La CAP suit la fin de transmission du beacon et la CFP, si elle est présente, suit immédiatement la fin de la CAP. Si un coordinateur ne souhaite pas appliquer la structure de la supertrame, il initialise les valeurs de BO et de SO à 15. Dans ce cas, nous nous retrouvons dans un PAN en mode non suivi de beacon.

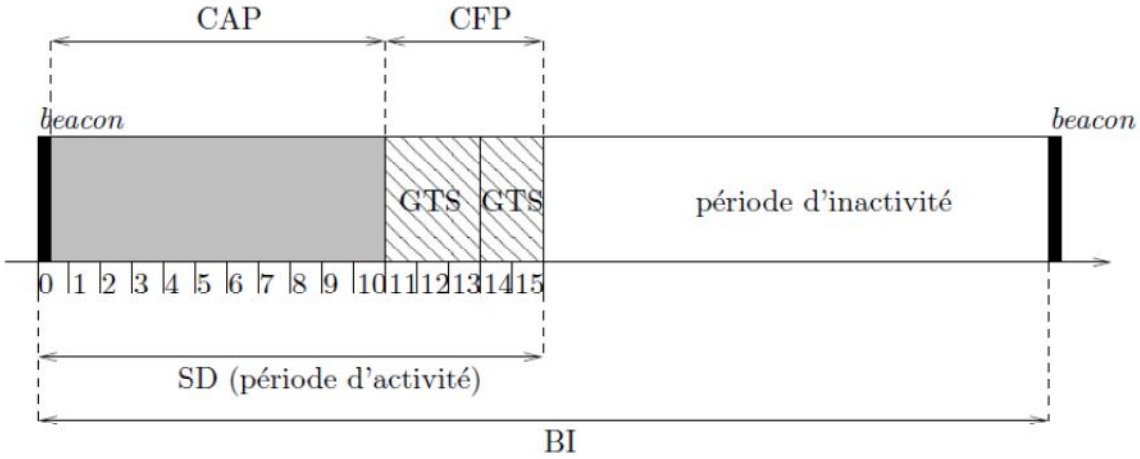


Figure D.5 : structure d'une supertrame

La figure D.5 montre un exemple d'une supertrame incluant un période de CAP, une période de CFP contenant deux GTS de tailles différentes et une période d'inactivité (car BO = SO+1).

L'accès au médium durant la CAP pour toute trame, sauf les acquittements et les beacons, est géré selon l'algorithme de CSMA/CA slotté. Avant tout envoi, l'entité doit s'assurer de pouvoir finir la transaction (incluant la réception d'un acquittement s'il est demandé) et de pouvoir attendre un IFS (InterFrame Space) avant la fin de la CAP. Si ce n'est pas le cas, l'envoi est reporté à la CAP de la supertrame suivante. L'envoi de trames durant les slots réservés au GTS s'effectue sans CSMA/CA.

I.2.2 Les scans, la création du réseau, les associations et la synchronisation

1 Les scans

Afin de découvrir les réseaux existants à portée, de créer un réseau, de s'associer à un réseau ou de se synchroniser avec un réseau, la couche MAC effectue un des quatre types de scans suivants :

- *Le scan d''energie* permet de récupérer l''energie maximum détectée sur chaque canal. Ce scan est utilisé par un coordinateur souhaitant créer un PAN afin de choisir le canal convenable.
- *Le scan actif* permet de récupérer la liste des identifiants de PAN existants. Le scan actif suit la diffusion de requête de beacon. Un coordinateur fonctionnant selon un mode non suivi de beacon répond à cette requête par une diffusion de beacon. Un coordinateur fonctionnant selon un mode suivi de beacon l'ignore et continue à envoyer ses beacons périodiquement. Ce scan est utilisé par un coordinateur souhaitant créer un PAN pour choisir le bon identifiant de PAN, ou par une station qui cherche à s'associer à un PAN (dont elle connaît l'identifiant).
- *Le scan passif* comme pour le scan actif, permet de récupérer la liste des identifiants de PAN existants. En revanche, la station n'envoie pas une requête de beacon mais elle effectue une écoute passive de chaque canal à scanner pour une durée donnée. Ce scan est utilisé par une station qui cherche à s'associer à un PAN.
- *Le scan d'orphelin* permet à une station de retrouver son coordinateur après une perte de synchronisation avec ce dernier.

2Création du réseau

La couche supérieure envoie une requête à la couche MAC pour effectuer un scan actif sur une liste de canaux afin de découvrir les réseaux existants à portée. Une fois le bon canal choisi, la couche

supérieure décide d'un identifiant de PAN et demande `a la couche MAC d'initier un PAN avec cet identifiant.

3 Association et désassociation

Après avoir effectué un scan (passif ou actif) et remonté le résultat dû à la couche supérieure, cette dernière demande à la couche MAC de s'associer à un PAN spécifique en précisant son identifiant PAN et l'adresse du coordinateur correspondant. Suite à cette demande, la couche MAC génère une requête d'association à destination du coordinateur.

A la réception d'une requête d'association, la couche MAC du coordinateur remonte l'information à la couche supérieure et c'est `a celle-ci de décider d'accepter ou pas cette requête en fonction des ressources qu'il lui reste par exemple. La réponse à cette requête d'association est envoyée en mode de transmission indirecte.

Suite à une requête de désassociation envoyée par la couche supérieure, la couche MAC envoie une indication de désassociation au coordinateur pour l'informer que la station souhaite se désassocier du PAN. De même, la couche supérieure d'un coordinateur peut décider de désassocier une station qui lui est associée, elle lui envoie alors une commande de désassociation pour l'informer de ce fait.

I.2.3 Synchronisation avec le beacon du coordinateur

Dans le mode suivi de beacon, les entités restent synchronisées au réseau grâce à la transmission périodique d'une trame de beacon par le coordinateur.

Toute station appartenant à un beacon-enabled PAN doit pouvoir se synchroniser avec le beacon de son coordinateur pour connaître la structure de la supertrame, et aussi pour récupérer les trames la concernant (comme indiqué par la liste des entités ayant des trames en instance).

Si une station ne reçoit pas un nombre aMaxLostBeacons consécutifs de beacons, la couche MAC détecte une perte de synchronisation et indique ce constat à la couche supérieure.

La couche MAC rejette toute trame de beacon dont l'adresse source ou l'identifiant du PAN ne correspondent pas à l'adresse du coordinateur ou à l'identifiant du PAN auquel la station est associée, respectivement.

Les caractéristiques de la sous-couche MAC

- Mise en œuvre de trames avec un adressage IEEE 64 bits ou un adressage court sur 16 bits, soit 65536 équipements adressables au plus dans ce dernier cas.
- Structure de trame simple.
- Mode de transmission fiable.
- Gestion de l'association/déassociation d'équipements.
- 3 niveaux de sécurité possibles sur les communications : aucun, liste ACL, chiffrement AES 128 de la charge (*payload*) de la trame.
- Mode de transfert de type *half duplex*.
- 4 trames sont définies :
  - Trame de données (*Data Frame*) : transfert de données.
  - Trame d'acquittement (*Acknowledgment Frame*) : confirmation de données bien reçues.
  - Trame beacon (*Beacon Frame*) : émise par le coordinateur de réseau en mode beacon.
  - Trame de commande MAC (*MAC Command Frame*) : pour le contrôle des nœuds.

La structure de la trame de données est donnée Figure D.6.

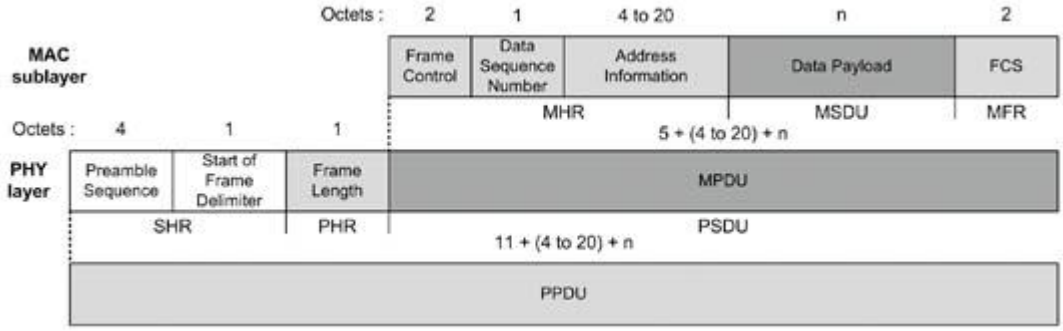


Figure D.6 : Structure d'une trame de la sous-couche MAC IEEE 802.15.4

Un échange d'une trame de données à l'initiative d'un nœud est donnée Figure D.7 dans le cas d'un réseau en mode beacon d'un réseau et en mode non beacon.

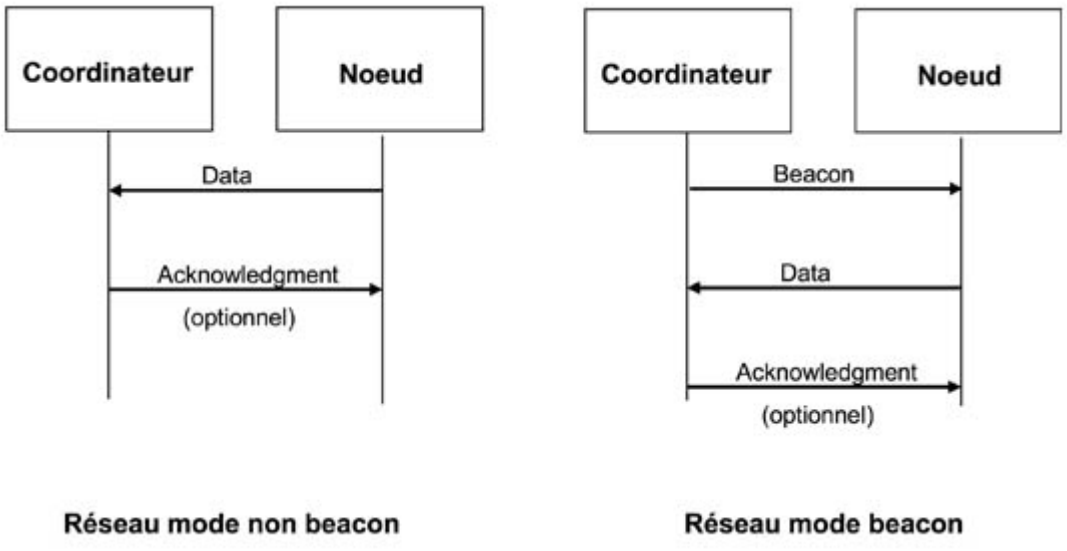


Figure D.7 : Échange d'une trame de données

- La norme IEEE 802.15.4 définit 3 types d'équipements (nœuds du réseau) :
  - Le coordinateur du réseau.
  - L'équipement à fonctionnalités complètes FFD (*Full Function Device*).
  - L'équipement à fonctionnalités réduites RFD (*Reduced Function Device*).

L'équipement FFD peut être soit un coordinateur, soit un routeur, soit un équipement terminal (capteur).

L'équipement RFD est un équipement simplifié comme un équipement terminal (*End Device*) muni de capteurs.

Pour communiquer au sein d'un même réseau, au moins un équipement FFD et des équipements RFD utilisent de concert le même canal radio parmi ceux définis dans la norme.

Un équipement FFD peut dialoguer avec des équipements RFD ou FFD, mais un équipement RFD ne peut dialoguer qu'avec un équipement FFD.

I.3 Couche ZigBee

Le niveau ZigBee précise les algorithmes de routage pouvant être mis en œuvre dans le réseau.

Il autorise aussi le chiffrement des données par AES-128.

Le routage mis en œuvre peut être soit direct, soit indirect.

Dans le cas du routage direct, l'équipement source connaît l'adresse de l'équipement destinataire. L'adresse est celle définie au niveau de la trame MAC.

Dans le cas d'un routage indirect, l'équipement source ne connaît pas l'adresse de l'équipement destinataire. Dans ce cas, un équipement routeur ou coordinateur fera la mise en relation avec l'équipement destinataire en utilisant sa table de routage.

L'algorithme de routage préconisé dans la norme ZigBee pour les réseaux maillés est l'algorithme AODV (*Ad hoc On-Demand Vector Routing*). C'est un algorithme de routage réactif : une route est établie uniquement sur demande.

Si l'on regarde maintenant la topologie réseau et sa taille, le réseau ZigBee est un réseau PAN (*Personal Area Network*). C'est un réseau de quelques dizaines de mètres permettant un échange de données avec des équipements électroniques.

On peut mettre dans la catégorie PAN les normes suivantes : USB, Bluetooth, Infrarouge IR et ZigBee et éventuellement Wifi…

Les topologies mises en œuvre avec ZigBee dépendent de la complexité de l'application utilisée. La figure D.8 présente quelques topologies possibles.

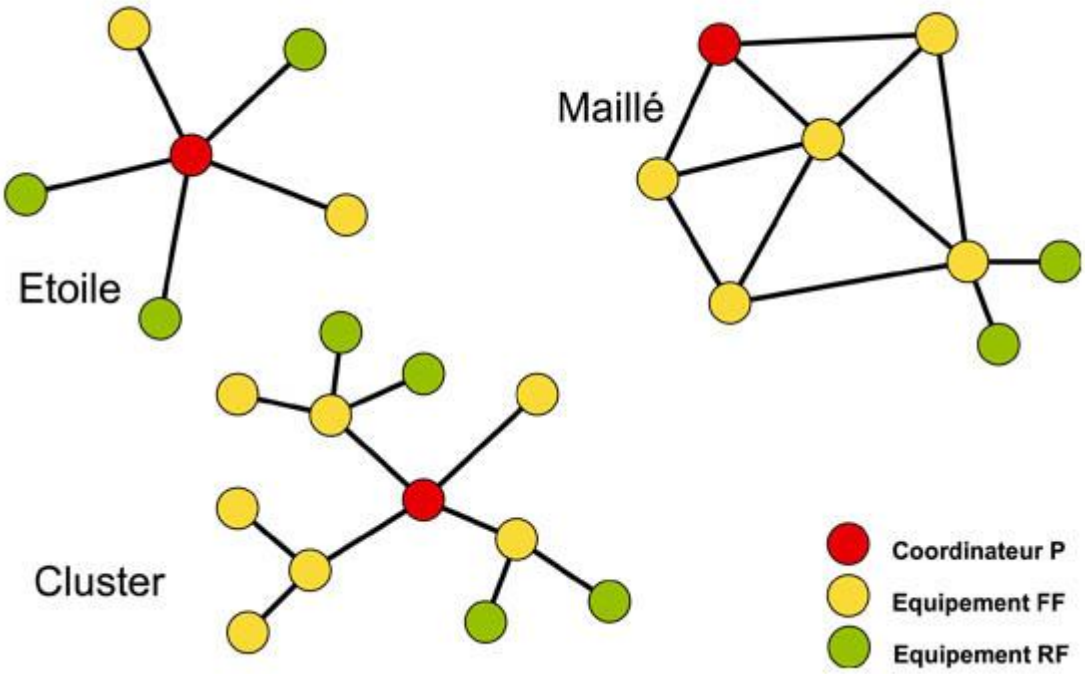
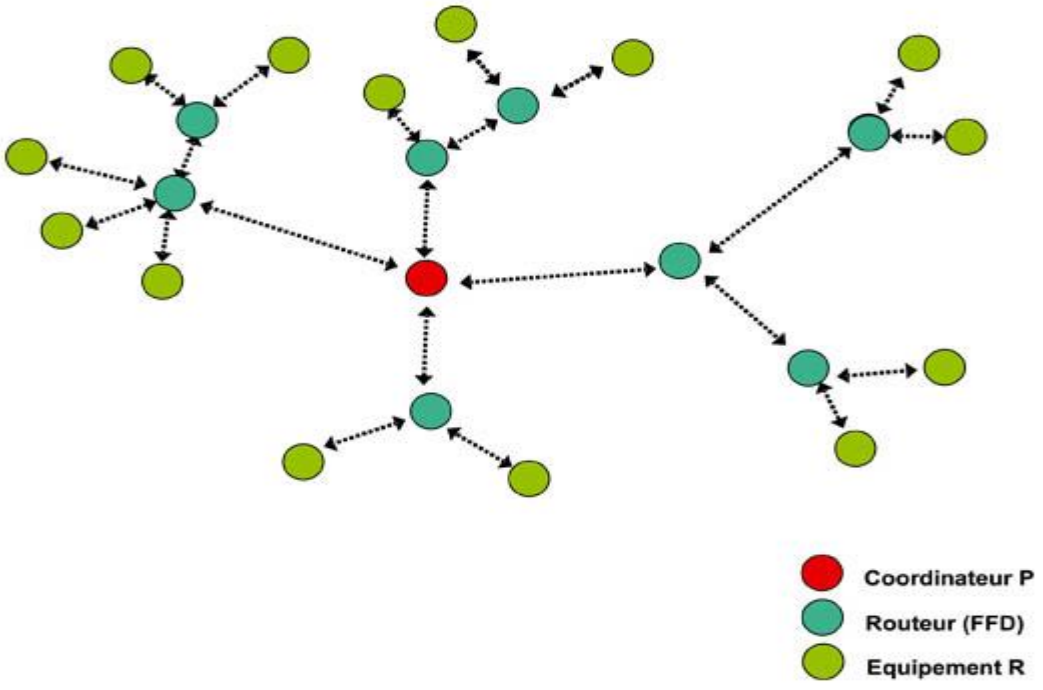


Figure D.8 : Topologies possibles d'un réseau ZigBee

Des topologies plus complexes peuvent être mises en œuvre en utilisant des routeurs ZigBee.

La figure D.9 donne un exemple d'une telle topologie.



II Bilan

ZigBee a toute sa place dans les réseaux de capteurs sans fil. C'est un concurrent sérieux à Bluetooth.

La figure D.10 reprécise ZigBee dans le panorama des réseaux sans fil.

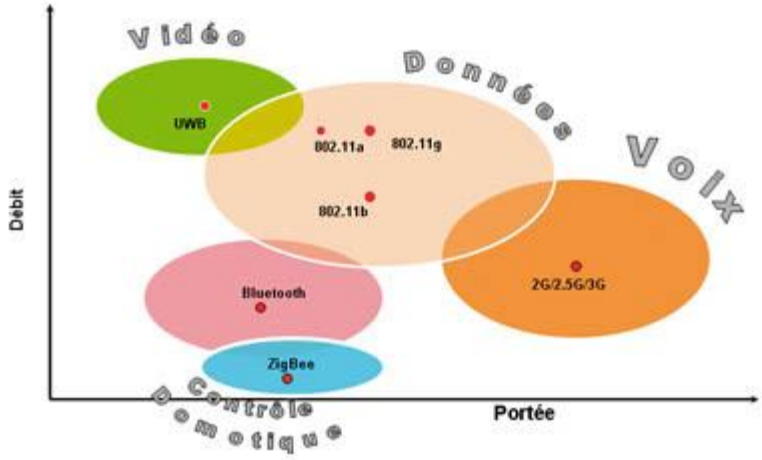


Figure D.10 : ZigBee dans le panorama des réseaux sans fil

ZigBee a donc bien un marché ciblé dans la domotique dans le cadre des réseaux de capteurs sans fil. C'est d'ailleurs un choix judicieux pour ce type de réseau sans fil par rapport à Bluetooth comme le montre le tableau 2.

Caractéristiques	ZigBee	Bluetooth
Portée Réel A l'extérieur	 10 - 100 m jusqu'à 400 m	 10 m 100 m
Débit	20, 40, 250 kb/s	1 Mb/s
Latence Enumération nouvel équipement Réveil équipement	 30 ms 15 ms	 20 s 3 s
Autonomie	En années	En jours
Sécurité	128 bits AES	64 bits, 128 bits
Fréquences	868 MHz, 915 MHz, 2.4 GHz ISM	2.4 GHz ISM
Complexité	Simple	Complexe
Topologie réseau	Ad hoc, étoile, maillé	Ad hoc
Equipements par réseau	jusqu'à 65536	8
Extensibilité	Oui/Très grande	Non/Très faible
Flexibilité	Très grande	Moyenne
Fiabilité	Très grande	Moyenne

Tableau 2 : Comparaison de ZigBee et de Bluetooth

III ZigBee dans le cadre de la réalisation du capteur sans fil

Il est important maintenant de situer ZigBee dans le cadre de la réalisation du capteur sans fil. Il convient de réaliser ce capteur sans fil ZigBee dans le cas d'une topologie simple.

Nous avons donc les caractéristiques suivantes :

● La topologie est en étoile. Le réseau est en mode non beacon.

● Le capteur ZigBee sera un équipement RFD.

● On utilisera un adressage ZigBee court sur 16 bits.

La figure 11 présente donc le cadre de la réalisation.

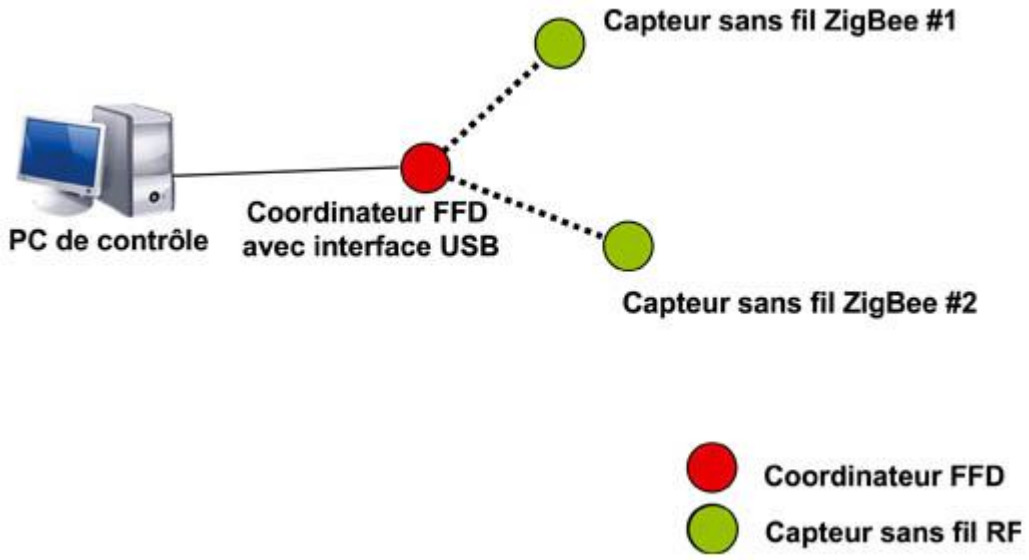


Figure D.11 : Topologie ZigBee mise en œuvre