REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU
FACULTE DU GENIE ELECTRIQUE ET INFORMATIQUE
DEPARTEMENT INFORMATIQUE

MEMOIRE

DE FIN D'ETUDES
En vue de l'obtention du diplôme de
Master en informatique
Spécialité : Systèmes informatiques

Thème :

Implémentation et simulation d'un algorithme de routage plat pour les RCSF

Cas : « Diffusion Directe »

- ***Proposé et dirigé par : M^r^ DEMRI.M***
- ***Réalisé par : M^lle^ AKRETCHE Lydia***
  &
  M^lle^ AREZKI Nabila

Année Universitaire 2014/2015

REMERCIEMENTS

Notre profonde gratitude et sincères remerciements vont à notre promoteur Mr DEMERI.M pour son suivi, sa disponibilité, son orientation et ses remarques pertinentes et précieuses.

Nous remercions chaleureusement les membres du jury pour l'honneur qu'ils nous font en acceptant de juger ce mémoire de fin d'études.

Enfin, nous remercions toutes les personnes ayant contribué de prés ou de loin au bon accomplissement de notre travail.

Lydia & Nabila

DÉDICACES

JE DÉDIE CE MODESTE TRAVAIL

- ✓ À mes chers parents qui m'ont soutenu tout au long de mes études et surtout pour réaliser ce mémoire et dont le rêve était toujours de nous voir réussir.
- ✓ À mes chères sœurs adorées: Tinhinane & Wissam.
- ✓ À mon unique « malih » & à la mémoire de mon grand père.
- ✓ À mon adorable oncle Samir, à qui je souhaite une réussite totale dans ses projets.
- ✓ À mes chères tantes, leurs époux et leurs enfants.
- ✓ À mes amis qui m'ont aidé de prés ou de loin en particulier Celia, Ghiles et Moumouh.

Lydia

DÉDICACES

JE DÉDIE CE MODESTE TRAVAIL

- ✓ A mes chers parents qui m'ont soutenu tout au long de mes études et surtout pour réaliser ce mémoire et dont le rêve était toujours de nous voir réussir.

- ✓ A mes chères frères : Sonia & Samir.

- ✓ A mes chères tantes, leurs époux et leurs enfants.

- ✓ A mes amis qui m'ont aidé de prés ou de loin.

Nabila

Sommaire

Table des figures

Introduction générale

Depuis leur création, les réseaux de communication sans fil ont connu un succès sans cesse croissant au sein des communautés scientifiques et industrielles. Grâce à ses divers avantages, cette technologie a pu s'instaurer comme acteur incontournable dans les architectures réseaux actuelles. Le média hertzien offre en effet des propriétés uniques, qui peuvent être résumées en trois points : la facilité du déploiement, l'ubiquité de l'information et le coût réduit d'installation. Au cours de son évolution, le paradigme sans fil a vu naître diverses architectures dérivées, telles que : les réseaux cellulaires, les réseaux locaux sans fils et autres. Durant cette dernière décennie, une architecture nouvelle a vu le jour : les réseaux de capteurs sans fil. Ce type de réseaux résulte d'une fusion de deux pôles de l'informatique moderne : les systèmes embarqués et les communications 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", communiquant via des liens sans fil. 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. Les RCSF sont souvent considérés comme étant les successeurs des réseaux ad hoc. En effet, les RCSF partagent avec les MANET (Mobile Ad hoc NETworks) plusieurs propriétés en commun, telles que l'absence d'infrastructure et les communications sans fil. Mais l'une des différences clés entre les deux architectures est le domaine d'application. Contrairement aux réseaux MANET, qui n'ont pas pu connaître un vrai succès, les RCSF ont su attirer un nombre croissant d'industriels, vu leur réalisme et leur apport concret. En effet, le besoin d'un suivi continu d'un environnement donné est assez courant dans diverses activités de la société.

Les processus industriels, les applications militaires de tracking, le monitoring d'habitat, ainsi que l'agriculture de précision ne sont que quelques exemples d'une panoplie vaste et variée d'applications possibles du suivi continu offert par les RCSF. Grâce à ce potentiel riche en applications, les RCSF on su se démarquer de leur origine MANET et attirer de grandes firmes à travers le monde, telles que IBM, Sun, Intel et Philips.

Malheureusement, les RCSF ne sont pas parfaits ! A cause de leur faible coût et leur déploiement dans des zones parfois hostiles, les motes sont assez fragiles et vulnérables à

diverses formes de défaillances : cassure, faible énergie, ... etc. Ces problèmes rendent les RCSF des systèmes à fragilité innée, qui doit être considérée comme une propriété normale du réseau.

L'objectif du travail en cours est d'étudier le fonctionnement du protocole de routage Diffusion Directe en termes d'énergie à travers des études de simulation. Ensuite, ce travail permettra de comparer ce protocole avec un autre protocole qui est « LEACH » toujours en termes d'énergie.

Notre étude s'étale sur trois chapitres, un premier chapitre qui présente d'une manière générale le routage dans les réseaux de capteurs sans fil ainsi que les protocoles de routages.

Un second chapitre qui détaille le fonctionnement du protocole de routage Diffusion Directe .

Un dernier chapitre est consacré à l'implémentation et simulation du protocole Diffusion Directe et sa comparaison avec LEACH. Ce dernier chapitre explique également les outils utilisés pour l'environnement d'exécution et de compilation.

Enfin on terminera avec une conclusion générale, une annexe et une Bibliographie.

CHAPITRE I

Le routage dans les réseaux de capteurs sans fil

I. Introduction :

Dans les réseaux sans fil, les protocoles de routage permettent d'établir des routes entre les nœuds pour acheminer les paquets entre eux. Cependant, dans les réseaux de capteurs, les protocoles de routage établissent des routes entre tout nœud du réseau et la station de base pour assurer la fidélité du routage. Dans cette optique, plusieurs protocoles ont été proposés dans la littérature. Il y en a de nouveaux protocoles qui ont été développés comme il y en a des protocoles qui sont des améliorations des autres et essaient de combler les limitations des versions originales.

Le routage dans les RCSF doit prendre en compte deux contraintes. Premièrement, remontrer rapidement l'information d'un nœud à la station de base en empruntant le plus court chemin. Deuxièmement, remonter l'information à la station de base à moindre coût en termes de consommation d'énergie.

Dans les premiers protocoles de routage conçus pour les RCSF, chaque nœud détecte et participe au routage de l'information. De ce fait, quand un nœud reçoit un paquet pour la première fois, il le retransmet dans son voisinage sinon il l'ignore. Ce processus d'acheminement de l'information se répète jusqu'à l'aboutissement à la station de base.

II. Qu'est-ce qu'un capteur :

Un capteur sans fil est un petit dispositif électronique capable de mesurer une valeur physique environnementale (température, lumière, pression, etc.) et de la communiquer à un centre de contrôle via une station de base. Les progrès conjoints de la microélectronique, des technologies de transmission sans fil et des applications logicielles ont permis de produire à coût raisonnable des micro-capteurs de quelques millimètres cubes de volume, susceptibles de fonctionner en réseaux .

Chaque capteur assure les fonctions suivantes:

►Acquisition des données.

►Calculer des informations à l'aide des valeurs collectées.

►Communiquer ces données à travers le réseau.

Un capteur est composé de quatre unités de base :

1. ***Unité de capture*** : permet la mesure de grandeurs physiques ou analogiques et leurs conversions en données numériques. Contient le capteur lui-même et des convertisseurs analogique-numérique (ADC). Les capteurs obtiennent des mesures sur les paramètres environnementaux et les transforment en signaux analogiques. Les ADCs convertissent ces signaux analogiques en signaux numériques.

2. ***Unité de traitement de données*** *:* constitué d'un processeur couplé à une mémoire vive et mémoire flash, elle peut embarquer un système d'exploitation pour faire fonctionner le capteur.
Elle contrôle les procédures permettant au nœud de collaborer avec les autres nœuds pour réaliser les tâches d'acquisition et stocker les données collectées.

3. ***Unité de communication)*** *:* elle est équipée d'un couple émetteur/récepteur, appelé transceiver (transmitter et receiver).
Elle permet la communication entre les différents nœuds du réseau via un support de communication radio.

4. ***Unité d'alimentation :*** c'est l'élément principal de l'architecture du capteur, elle fournit en énergie toutes les autres unités. Elle correspond le plus souvent à une batterie ou une pile alimentant le capteur, dont les ressources limitées en font une problématique propre à ce type de réseaux.

- Il existe des capteurs qui sont dotés d'autres composants additionnels tels que les systèmes de localisation GPS (Global Position System), la mobilité,…, etc.

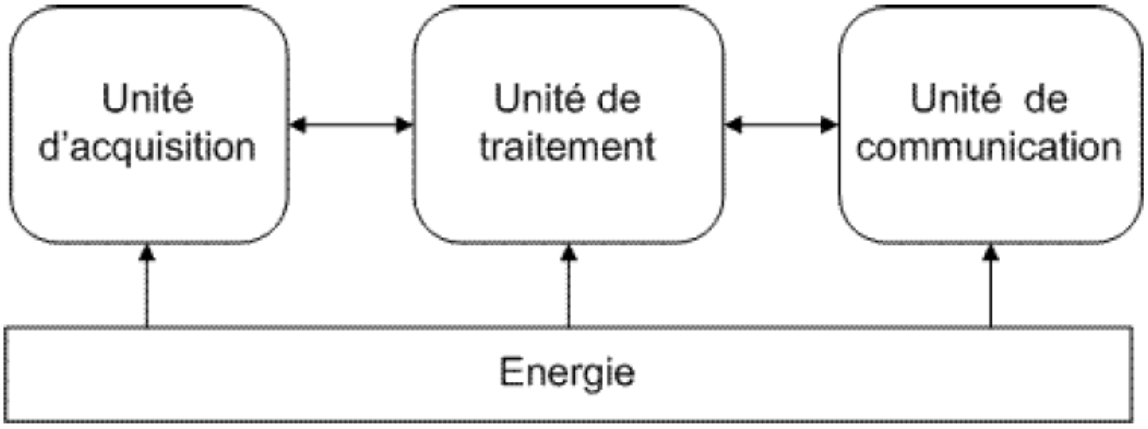


Figure I.1 : Les composants d'un nœud capteur [01]

III. Caractéristiques d'un RCSF :

Un réseau de capteurs présente des caractéristiques particulières comparativement aux autres réseaux sans fil. Dans cette section, nous présentons les principales caractéristiques de ces réseaux:

- **Sans infrastructure :** Les RCSF appartiennent à la famille des réseaux sans fil sans infrastructure dit ad-hoc de type many-to-one. Les capteurs sont généralement déployés aléatoirement dans des zones hostiles ce que nécessite qu'ils doivent s'auto-configurer et s'auto-organiser sans intervention humaine.

- **Scalabilité (Passage à l'échelle) :** Dans les RCSF, les capteurs sont déployés généralement en grand nombre pour garantir la couverture totale de la zone d'intérêt et faire face aux pannes puisque les capteurs peuvent cesser de fonctionner pour différentes causes. Nous pouvons avoir dans certains cas des RCSF de haute densité dont la taille dépasse mille capteurs voire un million de capteurs.

- **Interférences :** La notion d'interférences apparaît dans la plus part des réseaux sans fil en particulier dans les RCSF où deux capteurs voisins peuvent transmettre dans la même bande de fréquences ce qui peut causer des interférences.

- **Topologie dynamique :** Les capteurs sans fil peuvent être placés sur des objets mobiles par exemple sur des animaux pour les surveiller à distance sans perturber leur comportement. Ce type de scénario génère une topologie qui n'est pas statique dite dynamique [01].

- **Contrainte d'énergie, de stockage et de calcul :** La caractéristique la plus critique dans les RCSF c'est la limite des ressources énergétiques car la plupart des capteurs sont dotés de piles à énergie limitée.

IV. Domaines d'applications des RCSF :

Le faible coût et la communication sans fil ont permis aux réseaux de capteurs d'envahir plusieurs domaines d'applications. Ils permettent aussi d'étendre le domaine des applications existantes. Parmi ces domaines où ces réseaux se révèlent très utiles et peuvent offrir de meilleures contributions, on peut citer :

- ***Découverte 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 les feux de forêts, les tempêtes ou les 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.

- ***Contrôle de la pollution*** *:* des capteurs au-dessus d'un emplacement industriel offrent la possibilité de détecter et de contrôler des fuites de gaz ou de produits chimiques. Ces applications permettent 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 et on peut interroger le réseau sur l'état du champ et 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 berger.

- ***Surveillance médicale :*** en implantant sous la peau de mini capteurs vidéo, on peut recevoir des images d'une partie du corps en temps réel sans aucune chirurgie. 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.

V. L'architecture des RCSF :

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. Par ailleurs, les nœuds capteurs comme tout autre composant de télécommunication adhèrent à une architecture protocolaire spécifique. La réalisation de cette dernière requiert la mise en œuvre de techniques développées pour les réseaux ad hoc. Cependant, de nouveaux problèmes apparaissent engendrés entre autre par la sévérité des contraintes dues aux limitations de ressources physiques des RCSF. C'est la raison pour laquelle, il est commode que la conception des protocoles de communication soit faite d'une manière optimale.

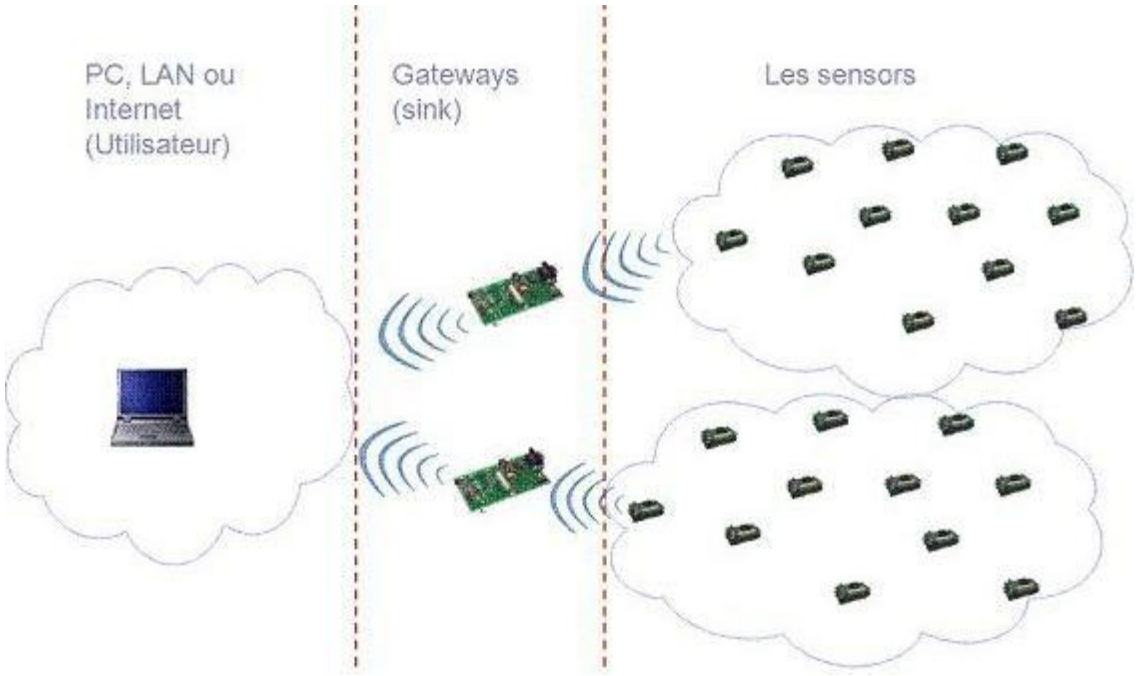


Figure I.2 : Exemple de réseaux de capteurs [02].

V.1. Architecture de communication :

Un réseau de capteurs est composé d'un grand nombre des nœuds dispersés dans une zone géographique. Le nombre des capteurs déployés est défini selon la taille de la zone de couverture et chaque nœud a la possibilité de détecter ou collecter certaine information (température, humidité, …) et de la transmettre aux voisins jusqu'à l'aboutissement à la station de base ou nœud puits [02]. La distance entre deux nœuds est estimée en fonction de la puissance de transmission via les ondes radio.

Le nœud puits est un nœud qui n'est pas comme les autres en termes d'énergie. Son but est de recevoir les informations qui y arrivent des autres nœuds et les transmettre à son tour à un poste de contrôle par le biais d'une transmission satellitaire ou Internet. La figure I.3 illustre un schéma de communication de données dans un RCSF.

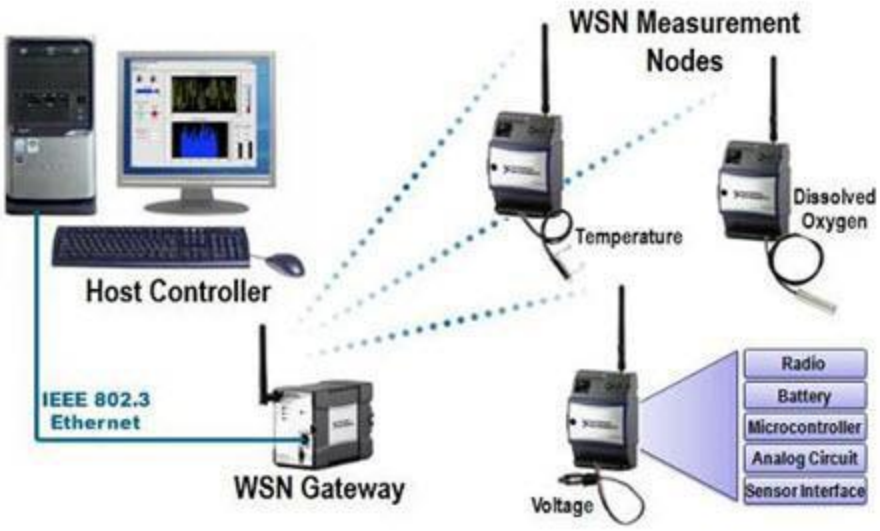


Figure I.3: Architecture de communication. [02]

V.2. Collecte d'informations :

Il y a deux méthodes pour collecter les informations dans un réseau de capteurs.

- **Collecte d'informations à la demande** : Lorsque l'on souhaite connaitre l'état de la zone d'intérêt à l'instant t, le nœud puits émet une requête en broadcast vers tous les nœuds déployés dans la zone d'intérêt pour que ces derniers remontent leur dernier relevé vers le nœud puits. Les informations sont alors acheminées vers le nœud puits par le biais d'une communication multi-sauts. La figure I.4 illustre ce mode de communication [03].

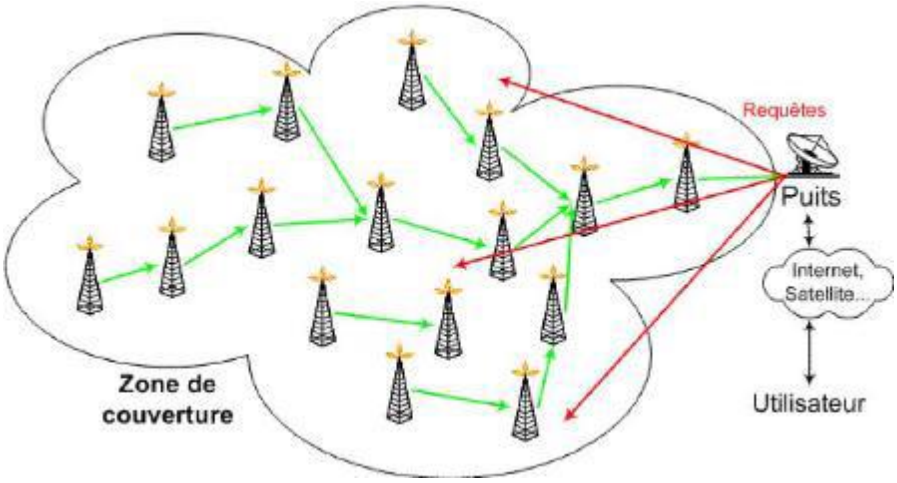


Figure I.4: Collecte d'informations à la demande [03].

- **Suite à un événement :** Un événement se produit en un point de la zone d'intérêt (changement brusque de température, détection d'un mouvement...), les capteurs situés à proximité de ce point cible remontent alors les informations relevées et les acheminent jusqu'au nœud puits. La figure I.5 symbolise comment se fait la communication de l'information au centre de contrôle dans ce type de réseaux [03].

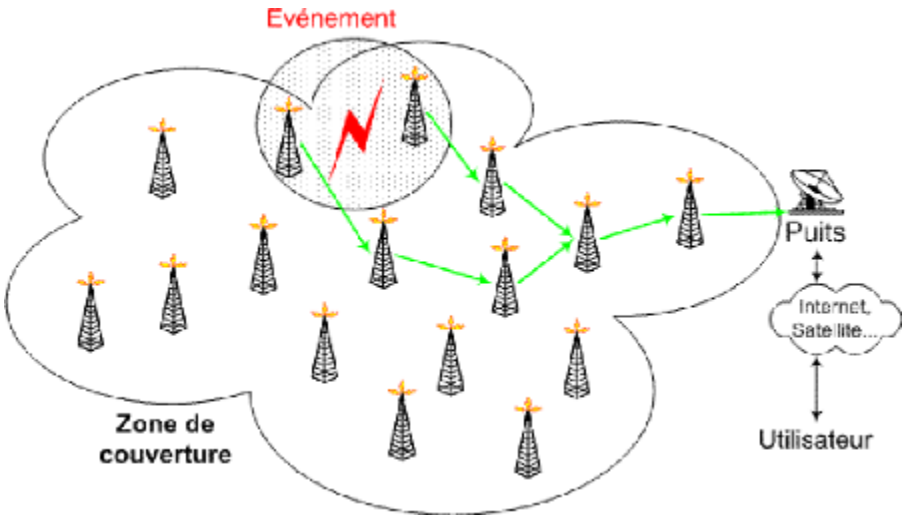


Figure I.5 : Application orientée événement [03]

V.3. Architecture protocolaire [04] :

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 : 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 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 garde une image sur la localisation des nœuds pendant la phase de routage, et le plan de gestion de l'énergie qui permet de préserver le maximum d'énergie.

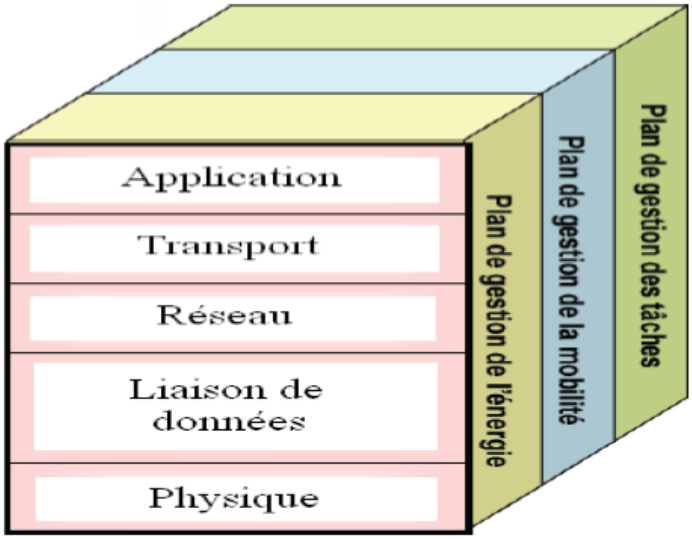


Figure I.6: Pile protocolaire dans les RCSF. [04]

❖ ***La couche physique :*** s'occupe de la spécification des caractéristiques matérielles, des fréquences porteuses,... assure les techniques d'émission, de réception et de modulation de données d'une manière robuste.

❖ ***La couche liaison :*** Spécifie comment les données sont expédiées entre deux nœuds/routeurs dans une distance d'un saut.

- Responsable du multiplexage des données, du contrôle d'erreurs, de l'accès au média,...
- Assure la liaison point à point et multipoints dans le réseau.
- Garantie une faible consommation d'énergie et minimiser les collisions entre les données diffusées par les nœuds voisins.

❖ ***La couche réseau :*** gère l'adressage et le routage des données. Ce routage diffère de celui des réseaux ad-hoc sans fils :

- Inexistence d'un système d'adressage global pour le grand nombre de nœuds.
- Les applications exigent l'écoulement des données mesurées de sources multiples à un puits particulier.
- Des capteurs peuvent produire les mêmes données redondantes.
- Gestion rigoureuse des ressources.
- D'où, la nécessité de nouveaux algorithmes de routage.

- ***La couche transport :*** Elle est chargée du transport des données, de leur découpage en paquets, du contrôle de flux, de la conservation de l'ordre des paquets et de la gestion des éventuelles erreurs de transmission.

- ***La couche application :*** Elle assure l'interface avec les applications. Il s'agit donc du niveau le plus proche des utilisateurs, géré directement par les logiciels.

  Les plans de gestion d'énergie, de mobilité et de tâche contrôlent l'énergie, le mouvement et la distribution de tâche dans 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, un retour arrière vers l'utilisateur est toujours maintenu et le nœud peut garder trace de ses nœuds voisins. En déterminant leurs voisins, les capteurs peuvent balancer l'utilisation de leur énergie et la réalisation de tâche.

- ***Plan de gestion de tâche :*** balance et 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 en même temps ; certains nœuds exécutent cette tâche plus que d'autres selon leur niveau de batterie.

VI. Contraintes de conception des RCSF :

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

- **La tolérance aux fautes** : la tolérance aux fautes est la capacité de maintenir les fonctionnalités du réseau en présence de fautes. La fiabilité des réseaux de capteurs sans fil est affectée par des défauts qui se produisent à cause de diverses raisons telles que le mauvais fonctionnement du matériel ou à cause d'un manque d'énergie. Ces problèmes n'affectent pas le reste du réseau.

- **Le facteur d'échelle** (Scalability) : le nombre de nœuds de capteurs augmente sur un réseau sans fil et ce nombre peut atteindre le million. Un nombre aussi important de nœuds engendre beaucoup de transmissions entre les nœuds et peut imposer des difficultés pour le transfert de données.

- **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.

- **L'environnement** : les capteurs sont souvent déployés en masse dans des endroits tels que des champs de bataille, 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,...) et redéploiement de nœuds additionnels.

- **Les contraintes matérielles** : la principale contrainte matérielle est la taille du capteur. Les autres contraintes sont la consommation d'énergie qui 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 standardisé. On utilise le plus souvent l'infrarouge, le Bluetooth et les communications radio Zig Bee.

- **La consommation d'énergie** : un capteur, en plus de 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 ré-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.

VII. Routage dans les RCSF :

Dans les RCSF, les capteurs sont déployés en grand nombre pour surveiller un phénomène et faire remonter l'information à un centre de contrôle distant. Pour atteindre cette finalité, les capteurs ont la capacité de communiquer et collaborer entre eux pour acheminer l'information collectée à la station de base en garantissant sa fiabilité et en empruntant le plus court chemin entre le nœud qui a détecté ce phénomène et la station de base. Cette opération permet le routage de l'information entre le nœud détecteur et le nœud puits et elle consiste à trouver les routes les plus courtes. Dans cette optique, plusieurs protocoles de routage ont été proposés dans la littérature.

Les contraintes présentées dans les RCSF ont donné naissance à des protocoles de routage différents que ceux des autres réseaux sans fil puisque la contrainte énergétique se

pose avec force dans les RCSF. De ce fait, les protocoles de routage conçus pour les RCSF doivent garantir l'acheminement de l'information entre tout nœud du réseau et la station de base à moindre coût en termes d'énergie.

Dans ce qui suit, nous présentons quelques approches et techniques sur lesquelles se basent les protocoles de routage dans les réseaux sans fil et en particulier dans les réseaux de capteurs.

VII.1- Approches d'établissement de routes :

a. Approche basée sur l'état du lien :

Chaque nœud du réseau maintient une vue globale de la topologie du réseau qui lui permet de calculer les chemins vers toutes les destinations. Les nœuds diffusent périodiquement (par inondation) l'état des liens avec leurs voisins à tous les nœuds du réseau. Le problème de cette approche, les messages de contrôle surchargent le réseau et surtout quand il s'agit d'un réseau avec un grand nombre de nœuds.

b. Approche basée sur le vecteur à distance :

Dans cette approche, chaque nœud transmet à ses voisins la distance en termes de nombre de sauts qui le sépare de chaque destination dans le réseau et le nœud voisin utilisé pour atteindre cette destination. Donc, un nœud établit les routes vers les autres destinations en se basant sur les informations reçues depuis tous ses voisins. Ce dernier calcule le chemin le plus court vers n'importe quelle destination dans le réseau. Si la distance séparant deux nœuds change on répète le processus de calcul.

L'approche du vecteur des distances évite l'inondation, mais elle est moins précise que l'approche basée sur l'état du lien puisqu'il est aussi difficile de trouver des routes alternatives en cas de coupure de routes.

VII.2 Les principaux protocoles de routage dans les RCSF :

a) Selon la structure du réseau :

La figure suivante résume les principaux protocoles de routage selon la structure du réseau. :

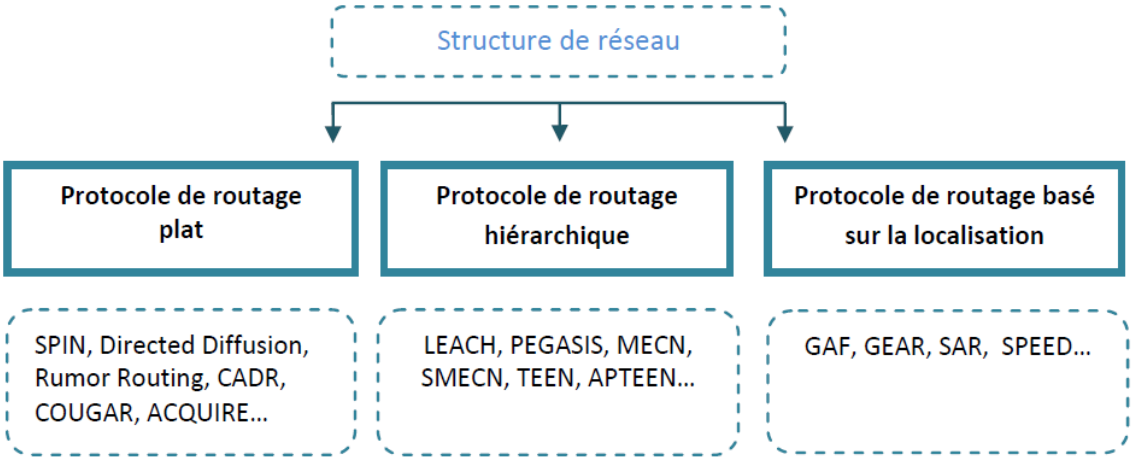


Figure I.7 : Protocoles de routage pour les RCSF selon la structure du réseau [07].

➢ *Les protocoles de routage plat :*

Ces protocoles considèrent que tous les nœuds sont identiques, c'est à dire ont les mêmes fonctions à exécuter sauf le nœud de contrôle (*sink*) qui est chargé de collecter toutes les informations issues des différents nœuds capteurs pour les transmettre vers l'utilisateur final.

➢ *Les protocoles de routage hiérarchique :*

L'objectif principal du routage hiérarchique est de maintenir efficacement la consommation d'énergie des nœuds capteurs en les impliquant dans la communication multi-saut au sein d'un cluster et en effectuant l'agrégation et la fusion des données afin de diminuer le nombre de messages transmis à la destination. La formation de clusters est généralement fondée sur la réserve d'énergie des capteurs et sur les capteurs qui sont à proximité de cluster-head (voir figure ci-dessous). LEACH (Low Energy Adaptive Clustering Hierarchical) est l'une de premières approches de routage pour les réseaux de capteurs. L'idée proposée par LEACH a été une inspiration pour de nombreux protocoles de routage hiérarchique, bien que certains protocoles aient été développés de manière indépendante.

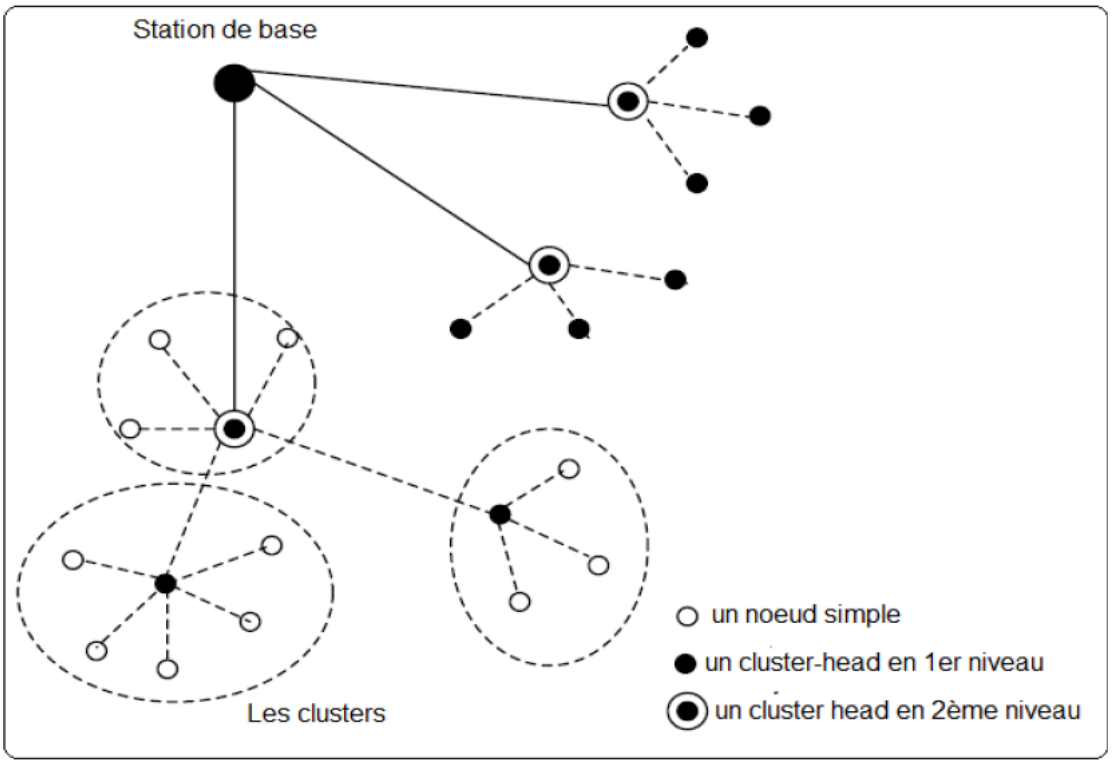


Figure I.8 : Topologie hiérarchique [08].

- ***Les protocoles de routage basés sur la localisation :***

Les protocoles de routage basés sur la localisation utilisent les informations d'emplacement pour guider la découverte de routage et la transmission des données. Ils permettent la transmission directionnelle de l'information en évitant l'inondation d'information dans l'ensemble du réseau. Par conséquent, le coût de contrôle de l'algorithme est réduit et le routage est optimisé. De plus, avec la topologie du réseau basée sur des informations de localisation de nœuds, la gestion du réseau devient simple.

L'inconvénient de ces protocoles de routage est que chaque nœud doit connaître les emplacements des autres nœuds.

b) *Selon le type du protocole :*

La figure suivante résume les principaux protocoles de routage selon le type du protocole :

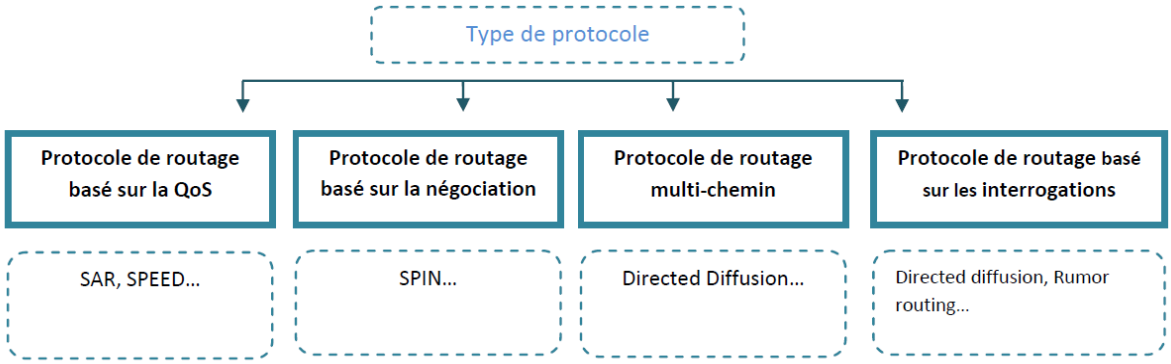


Figure I.9 : Protocoles de routage pour les RCSF selon le type du protocole [07].

➢ *Protocole de routage basé sur la QoS :*

Ce type de routage est utilisé dans les applications qui ont des exigences temps réel. Par exemple, le système de sécurité ou la détection d'intrusion et les applications de surveillance (centrales nucléaires, applications militaires). Dans ces protocoles, le réseau doit équilibrer entre la consommation d'énergie et la qualité des données. En particulier, le réseau doit satisfaire certaines métriques de QoS comme : le retard, l'énergie et la largeur de bande passante.

➢ *Protocole de routage basé sur la négociation des données :*

En détectant le même phénomène, les nœuds capteurs inondent le réseau par les mêmes paquets de données. Ce problème de redondance peut être résolu en employant des protocoles de routage basés sur la négociation. En effet, avant de transmettre, les nœuds capteurs négocient entre eux leurs données en échangeant des paquets de signalisation spéciales, appelés *META-DATA*. Ces paquets permettent de vérifier si les nœuds voisins disposent des mêmes données à transmettre [04]. Cette procédure garantit que seules les informations utiles seront transmises et élimine la redondance des données.

➢ *Protocole de routage multi-chemin :*

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

DD (Directed Diffusion) est l'un des protocoles de cette classe, qui utilise lui aussi un schéma de nommage sous forme de paire (Attribut-valeur) pour les requêtes et les données. Chaque nœud qui recense un évènement crée et diffuse un gradient au voisinage direct. Chaque nœud puits diffuse l'intérêt (qui est une demande ou une interrogation) vers tous les voisins, par la suite le gradient spécifie la direction dans laquelle les données répondant à l'intérêt seront envoyées. De cette manière, plusieurs routes reliant la station de base à la source de données peuvent être trouvées, puis la meilleure route sera renforcée pour éviter la redondance.
Son fonctionnement se résume en trois phases :

- La propagation des intérêts.
- Établissement des gradients.
- La livraison des données et renforcement des chemins.

Le DD définit des règles de renforcement positif et négatif pour la sélection des chemins.
On détaillera bien le fonctionnement de ce protocole dans le chapitre suivant.

➢ *Protocole de routage basé sur les interrogations :*

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

VII.3. Les critères de performance des protocoles de routage dans les RCSF :

La performance des réseaux de capteurs sans fil est fondée sur les facteurs suivants :

- **Évolutivité** : l'évolutivité est un facteur important dans les réseaux de capteurs sans fil. Une zone de réseau n'est pas toujours statique, elle change selon les besoins des utilisateurs. Tous les nœuds dans le domaine du réseau doivent être évolutifs ou être en mesure de s'adapter aux changements dans la structure du réseau en fonction de l'utilisateur.
- **L'énergie** : chaque nœud utilise peu d'énergie pour des activités telles que la détection, le traitement, le stockage et la transmission. Un nœud dans le réseau doit savoir combien d'énergie sera utilisée pour effectuer une nouvelle tâche à laquelle il est soumis. L'énergie consommée peut varier selon le type de fonctionnalité ou l'activité qu'il a à accomplir.
- **Le temps de traitement**: il se réfère au temps pris par le nœud dans le réseau pour assurer l'ensemble de l'opération commençant par la détection, le traitement des données ou le stockage de données, la transmission ou la réception sur le réseau.
- **Le schéma de transmission**: la transmission de données par les nœuds capteurs vers la destination ou la station de base se fait par un schéma de routage à un seul saut ou à multi-saut.
- **La capacité du réseau** : tous les nœuds du réseau de capteurs utilisent certaines ressources du réseau qui les aident à accomplir certaines activités comme la détection ou la transformation.
- **Synchronisation :** dans les communications radio entre les nœuds capteurs d'un WSN, les capteurs écoutent en permanence les transmissions et consomment de l'énergie s'ils ne sont pas synchronisés les uns les autres.
- **Contrôle de paquets**: un paquet envoyé avant la transmission entre deux nœuds est appelé le paquet de contrôle. Le paquet de contrôle contient le nombre de bits de données envoyés, l'adresse du nœud de destination et certaines informations qui contribuent à éviter les collisions pendant la transmission.

VII.4. Les limite des protocoles de routage dans les RCSF :

La plupart des protocoles de routage proposés pour les réseaux sans fil en particulier pour les RCSF, ont été conçus pour des réseaux de taille modeste. Cependant, lorsque le nombre de noeuds augmente les performances de ces protocoles commencent à se dégrader et on parle de la contrainte de passage à l'échelle.

Le passage à l'échelle est le fait de mettre en place un grand nombre de nœuds dans une zone de couverture. Or, quand le nombre de nœuds augmente cela pourra avoir une influence sur les performances du réseau en termes de latence, etc. En outre, la majorité des protocoles de routage ont été développés pour des réseaux de taille modeste et nous avons remarqué que lorsque le nombre de nœuds augmente les performances de ces protocoles se dégradent grandement.

Dans les applications orientées événements (Event-Driven), le temps nécessaire pour remonter une information au centre de contrôle doit être réduit pour que l'équipe d'intervention arrive à temps et minimise l'occurrence d'une catastrophe. Cependant, nous avons remarqué que lorsque le nombre de nœuds augmente dans certains protocoles, le temps pour envoyer l'information à la station de base devient assez grand, ce qui provoque un retard dans l'intervention. De ce fait, les protocoles doivent garantir un temps réduit pour alerter le centre de contrôle. Pour se faire, nous avons proposé d'utiliser des heuristiques pour réduire ce temps dans les réseaux denses.

VIII. Consommation d'énergie dans les RCSF :

La première étape dans la conception de système énergétique de capteurs consiste à analyser les caractéristiques de consommation d'énergie d'un nœud capteur sans fil. Cette analyse systématique de l'énergie d'un nœud capteur est extrêmement importante pour identifier les problèmes dans le système énergétique pour permettre une optimisation efficace. L'énergie consommée par un capteur est principalement due aux opérations suivantes : la détection, le traitement et la communication.

Énergie de capture :

Les sources de consommation d'énergie des nœuds pour les opérations de détection ou de capture sont : l'échantillonnage, la conversion analogique-numérique, le traitement de signal et l'activation de la sonde de capture.

Énergie de traitement :

L'énergie de traitement est composée de deux sortes d'énergie: l'énergie de commutation et l'énergie de fuite. L'énergie de commutation est déterminée par la tension d'alimentation et la capacité totale commutée au niveau logiciel (en exécutant un logiciel). Par contre, l'énergie de fuite correspond à l'énergie consommée lorsque l'unité de calcul n'effectue aucun traitement. En général, l'énergie de traitement est faible par rapport à celle nécessaire pour la communication.

Énergie de communication :

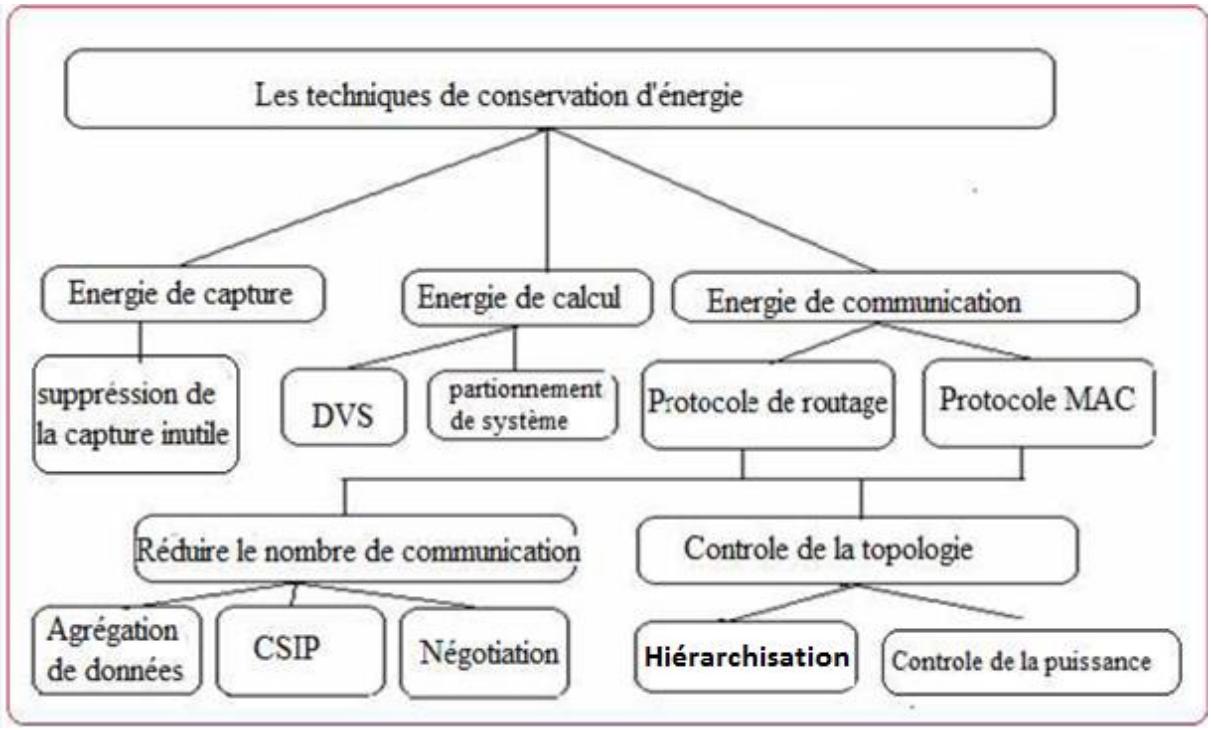
L'énergie de communication se décline en trois parties : l'énergie de réception, l'énergie de l'émission et l'énergie en état de veille. Cette énergie est déterminée par la quantité des données à communiquer et la distance de transmission, ainsi que par les propriétés physiques du module radio. L'émission d'un signal est caractérisée par sa puissance ; quand la puissance d'émission est élevée, le signal aura une grande portée et l'énergie consommée sera plus élevée. Notons que l'énergie de communication représente la portion la plus grande de l'énergie consommée par un nœud capteur.

IX. Techniques de minimisation de la consommation d'énergie :

La consommation d'énergie est très importante puisque généralement les capteurs sont déployés dans des zones inaccessibles. Ainsi, il est difficile, voire impossible, de remplacer les batteries après leur épuisement. De ce fait, la consommation d'énergie au niveau des capteurs a une grande influence sur la durée de vie du réseau.

Après la description des principales causes de consommation d'énergie dans les RCSF, nous présentons dans ce qui suit les différentes techniques utilisées pour minimiser cette consommation. Ces techniques sont appliquées soit au niveau de la couche liaison soit au niveau de la couche réseau.

Le schéma suivant donne un aperçu global de ces mécanismes :



FigureI.10 : Les techniques de conservation d'énergie [05].

L'énergie du capteur peut être économisée soit au niveau de la capture, au niveau du traitement ou au niveau de la communication.

- La seule solution apportée pour la minimisation de la consommation d'énergie au niveau de la capture consiste à réduire les fréquences et les durées de captures.
- L'énergie de calcul peut être optimisée en utilisant deux techniques :
    1. L'approche DVS (Dynamique Voltage Scaling) [05] qui consiste à ajuster de manière adaptative la tension d'alimentation et la fréquence du microprocesseur pour économiser la puissance de calcul sans dégradation des performances.
    2. L'approche du partitionnement de système qui consiste à transférer un calcul prohibitif en temps de calcul vers une station de base qui n'a pas de contraintes énergétiques et qui possède une grande capacité de calcul [06].
- La minimisation de la consommation d'énergie pendant la communication est étroitement liée aux protocoles développés pour la couche réseau et la sous-couche MAC. Ces protocoles se basent sur plusieurs techniques : l'agrégation de données, la négociation et à la technique CSIP (Collaborative Signal and Information Processing). Cette dernière technique est une discipline qui combine plusieurs domaines [07] : la communication et le calcul à basse puissance, le traitement du signal, les algorithmes

distribués, la tolérance aux fautes, les systèmes adaptatifs et la théorie de fusion des capteurs et des décisions. Ces techniques ont pour but de réduire le nombre d'émission/ réception des messages.

X. Conclusion :

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

Dans ce premier chapitre, nous avons présenté les réseaux de capteurs sans fil, leurs architectures de communication, la pile protocolaire des capteurs et leurs diverses applications. Cependant, nous avons remarqué que plusieurs facteurs et contraintes compliquent la gestion de ce type de réseaux. En effet, les réseaux de capteurs se caractérisent par une capacité énergétique limitée.

CHAPITRE II

Fonctionnement du protocole ***« Diffusion Directe »***

I- Introduction :

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

Le protocole de routage nommé « Directed Diffusion » est un protocole dont le principe est de trouver le meilleur chemin pour diriger les données d'un nœud source vers le nœud « sink » ou station de base.

II- Définition du protocole « diffusion directe » :

Directed Diffusion est un protocole de propagation de données, permettant d'utiliser plusieurs chemins pour le routage d'information. Le puits diffuse un intérêt sous forme de requête, afin d'interroger le réseau sur une donnée particulière. Il se base sur le modèle publish/subscribe.

Les capteurs envoient (publient) des notifications d'événements au collecteur, qui va souscrire son intérêt pour certaines de ces informations. Les capteurs concernés publient par la suite l'information désirée.

III- Fonctionnement du protocole « diffusion directe » :

Directed Diffusion (DD) fut l'un des premiers protocoles centrés-données. Il a été proposé par C.Intanagonwiwat, R.Govindan et D.Estrin [10] et est devenu l'un des protocoles les plus répandus dans les réseaux de capteurs. Sa création représente une importante avancée dans le domaine du routage. Il a été une base pour la conception de plusieurs protocoles de routage pour les réseaux de capteurs [11]. Ci-après les étapes de son fonctionnement :

III.1- Nomination des données :

L'adressage dans DD utilise un schéma attribut-valeur afin de décrire les intérêts et les rapports de données.

Exemple : dans une application de protection de forêts, une requête peut être effectuée sous cette forme : [16]

```
Type = GetTemperature
Zone = [100, 100, 120, 120]
Interval = 10 ms
Durartion = 1 mn
```

Une réponse d'un capteur pourra être formulée ainsi :

```
Type = GetTemperature
Location = (110, 115)
Temperature = 32
Timestamp = 11:32:10
```

L'exemple ci-dessous définit les différentes étapes d'établissement de chemins de routage de ce protocole.

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

III.2- Propagation des intérêts et établissement des gradients :

Le collecteur (nœud sink) commence par envoyer périodiquement un message d'intérêt. Cette tâche est définie par un ensemble de paires "attributs-valeurs".
Une description simplifiée de l'exemple précédent pourrait être de la forme suivante : [16]

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

Le message ainsi décrit permet de spécifier les données par lesquelles l'utilisateur est intéressé. Pour cette raison, il est appelé intérêt. Il peut contenir différents attributs. L'exemple précédent spécifie :

- le type de la cible détectée.
- le débit (l'intervalle) avec lequel les réponses doivent être envoyées.
- la durée de validité de l'intérêt
- et la sous-région de captage concernée par l'intérêt.

- Au départ, l'intérêt est diffusé par le nœud collecteur à tous ses voisins. Le débit demandé initialement par l'intérêt envoyé est plus faible que celui demandé par l'utilisateur.

- Chaque nœud du réseau maintient à son niveau un cache qui a pour but de garder trace des intérêts reçus.

- Chaque entrée du cache correspond à un intérêt distinct et contient plusieurs champs de gradients qui identifient les voisins depuis lesquels l'intérêt a été reçu.

  - Un gradient spécifie une valeur et une direction dans laquelle les données répondant à l'intérêt seront envoyées.
  - Le choix de la sémantique de la valeur associée au gradient dépend de l'application du réseau de capteurs. Dans l'exemple cité précédemment, cette valeur représente le débit avec lequel les données seront envoyées. Dans d'autres applications, cette valeur peut, par exemple, être une probabilité p avec laquelle une donnée est envoyée vers le voisin.
  - Chaque gradient possède une certaine durée de vie qui représente la durée de vie de l'intérêt associé. Quand un gradient expire, il est retiré de l'entrée d'intérêt. Quand tous les gradients d'une entrée d'intérêt expirent, l'entrée elle-même est supprimée du cache.

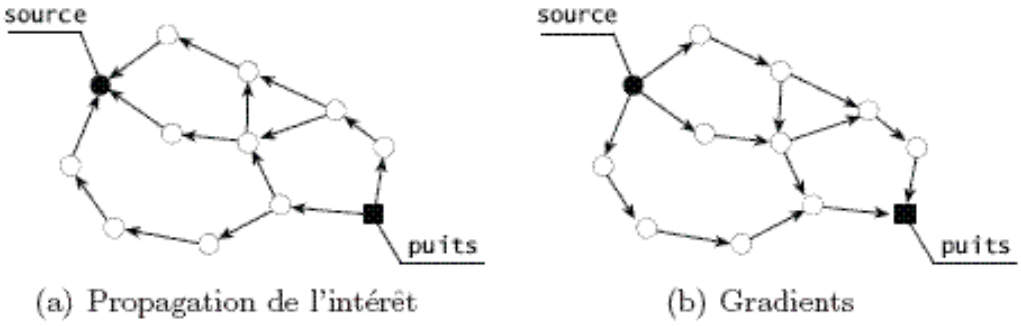


Figure II. 1 : Propagation des intérêts et établissement des gradients [17]

Le nœud source (NS) détecte un évènement.

Quand un nœud reçoit un intérêt, il vérifie son cache d'intérêts.

- Si aucune entrée correspondante à l'intérêt reçu n'existe (il est différent des intérêts du cache), le nœud crée une nouvelle entrée dans son cache, dans laquelle il va définir un gradient vers le voisin à partir duquel l'intérêt a été reçu.

- S'il existe une entrée correspondante, mais aucun gradient pour l'émetteur de l'intérêt, le nœud ajoute un gradient vers le voisin émetteur.

- Finalement, s'il existe déjà une entrée et un gradient pour l'émetteur, le nœud les met simplement à jour.

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

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

Ainsi, un intérêt est disséminé à partir du nœud collecteur sur tout le réseau. Cette dissémination installe des gradients dans chaque nœud du réseau pour orienter les données de réponses à l'intérêt inondé, vers le nœud collecteur ; telle que montre la figure ci-dessous

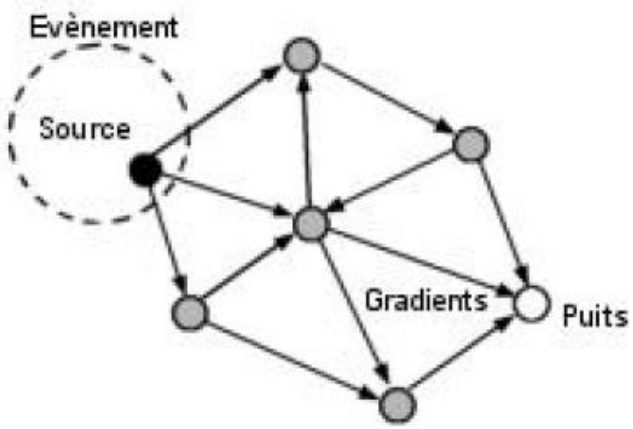


Figure II. 2 : **Établissement des gradients [17]**

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

III.3- Propagation des données :

Quand un nœud capte un évènement dans son champ de déploiement, il consulte son cache d'intérêts pour rechercher un enregistrement correspondant aux données captées. Si l'intérêt est trouvé, le nœud (qui devient désormais source de données pour cet intérêt) commence à envoyer l'évènement capté à tous ses voisins (constitués par les gradients établis dans la phase d'inondation d'intérêt) avec un faible débit appelé débit exploratoire. Suivant l'intérêt donné dans l'exemple adopté, l'évènement suivant forme une réponse pour le nœud collecteur : [16]

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

Au départ, les données sont considérées comme exploratoires, et sont envoyées à tous les voisins, pour lesquels un gradient a été établi, avec un faible débit.

Ces données exploratoires voyagent vers le nœud collecteur dans le sens inverse de celui de la propagation des intérêts. Elles sont prévues pour la construction et la réparation des chemins qui seront adoptés par la suite.

Un nœud intermédiaire qui vient de recevoir un message de donnée exploratoire d'un nœud voisin, cherche dans son cache d'intérêts une entrée correspondante à l'évènement capté.

- Si aucune entrée n'est trouvée, le message de données est écarté.
- Sinon, le nœud vérifie alors son cache de données (associé à l'entrée de l'intérêt).

Ce cache de données garde trace des données récemment envoyées et permet, entre autres, l'empêchement des boucles.

- Si le message reçu correspond à une entrée du cache de données, il est abandonné car il a déjà été envoyé.

- Sinon, le message est inséré dans le cache puis diffusé aux voisins pour lesquels un gradient a été établi.

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

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

Remarque : Comment éviter les boucles ?

Avant de relayer une donnée à ses voisins, un nœud utilise son cache de données. Ce cache enregistre les données récemment émises par les voisins. Cela évite la création de boucles, en supprimant les données déjà rencontrées.

III.4- Renforcement des chemins :

Renforcement Positif :

La phase de propagation des données exploratoires a pour but d'explorer les chemins existants entre la source et le collecteur.

Le renforcement positif sert à choisir une route (selon un certain critère) parmi celles découvertes, afin d'obtenir les données à haut débit appelées données renforcées.

- Pour cela, quand les données exploratoires atteignent le nœud collecteur, ce dernier choisit l'un de ses voisins appropriés et lui envoie un message de renforcement positif.

- Un nœud intermédiaire qui reçoit ce message de renforcement, renforce son gradient vers l'émetteur. Dans l'exemple précédent, un gradient est considéré comme étant renforcé si sa valeur (c'est-à-dire le débit de données demandé) est supérieure à 1 donnée / sec.

- Par la suite, le nœud doit, à son tour, envoyer le message de renforcement à l'un de ses voisins (en suivant le même critère de sélection).

- De cette manière, l'un des chemins explorés est récursivement renforcé (figure II.3).

- Pour le choix du chemin à renforcer, le nœud collecteur, et par la suite les nœuds intermédiaires, appliquent localement la règle de renforcement positif.

- Grâce au cache de données, un nœud choisit le premier voisin à partir duquel il a reçu une donnée exploratoire correspondant à l'intérêt.
  Par conséquent, un chemin ayant la plus faible latence est établi entre la source et le collecteur.
  D'autres règles peuvent être appliquées. Par exemple, un nœud peut choisir le voisin à partir duquel il a reçu le plus grand nombre de messages de données. Ainsi, le chemin le plus fiable est élu.

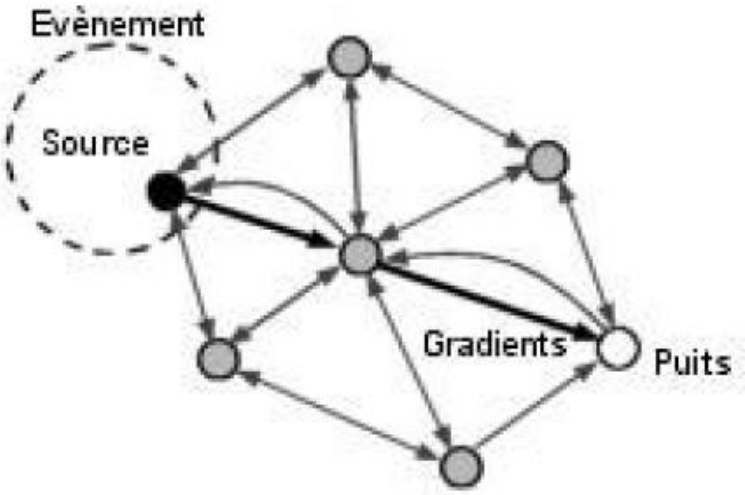


Figure II. 3 : Renforcement d'un chemin. [17]

- Une fois qu'un chemin reliant le nœud source au nœud collecteur soit renforcé, les données générées par la source sont envoyées avec un débit plus élevé à travers ce chemin.

- Un nœud qui vient de recevoir une donnée renforcée enverra celle-ci uniquement aux voisins pour lesquelles il possède un gradient renforcé.

Remarque :

La latence : est un délai minimum de transmission dans les communications informatiques. Il désigne le temps nécessaire à un paquet de données pour passer de la source à la destination à travers un réseau.

- **Renforcement négatif :**

Dans le cas de panne d'un lien (perte de paquet, débit réduit, etc.) le puits peut envoyer un renforcement négatif sur le chemin en panne en spécifiant le débit de base (exploratoire), et en procédant à un renforcement positif d'un chemin alternatif.

Remarque : Cette étape parle des chemins défaillants et de pannes.

VI. Conclusion :

La contribution au routage dans les réseaux de capteurs sans fil est :

Routage plat : choix du chemin optimal.

Dans DD c'est la station de base qui diffuse les requêtes sous forme d'intérêt ;

La communication s'effectue de voisin à voisin ou chaque nœud agrège les données et garde en mémoire le chemin de provenance.

CHAPITRE III

Implémentation

&

Simulation

I. Introduction :

Suite aux différents problèmes vécus par les réseaux de capteurs (problème énergétiques et de ressources), l'université de Berkeley a développé alors un système d'exploitation minime destiné pour ces réseaux : *TinyOS,* Il est orienté "*composants*" afin de faciliter l'implémentation de ces réseaux, tout en minimisant la taille du code afin de respecter les contraintes de mémoire des composants matériels.

TinyOS a été programmé en *NesC.* Ce langage a été inventé pour répondre aux attentes des systèmes embarqués. Il possède une syntaxe proche du C, supporte le système multitâche de TinyOS et défini des mécanismes pour structurer et lier des composants logiciels en un système embarqué robuste [12].

Dans ce chapitre, nous introduirons le langage NesC qui va nous permettre par la suite d'implémenter le protocole de routage diffusion directe.

II. Description de l'application :

- ***Les Structures de données :***

La structure des messages « Interet » et « Data» définit dans le fichier : « app.h » sont montrés ci-dessous :

****Interet ****

```
typedef struct Interet{
uint16_t seqNum ; // numero de seq (32- bits)
uint16_t pred; // dernier saut
uint16_t NBsaut;//nombre de sauts
uint8_t numAttrs ; //nombre d'attributs contenus dans un packet

bool recuInt;
}  Interet ;
```

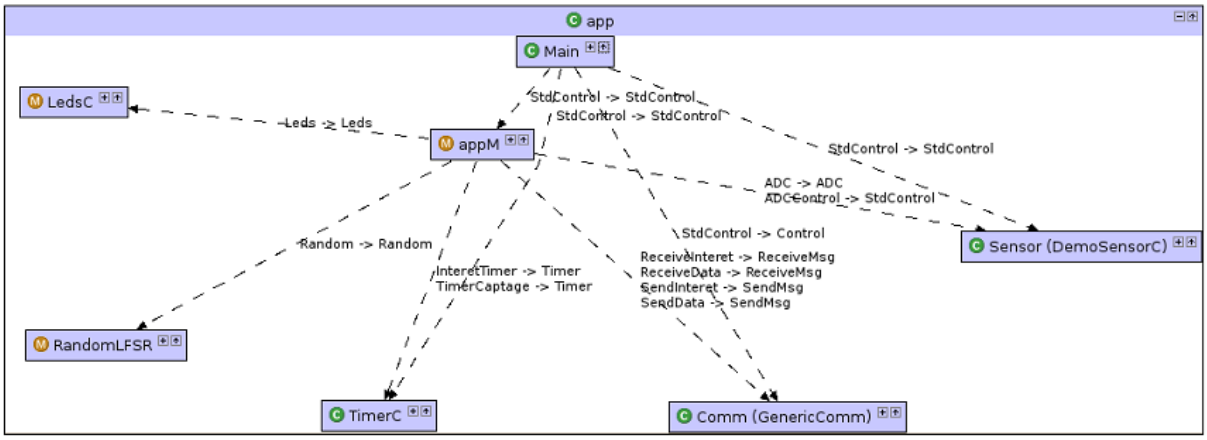
****Data****

```
typedef struct Data{
uint16_t Val;//la donnée
uint16_t Source;//nœud qui génère la donnée
uint16_t prédécesseur;
uint16_t destination;
bool recuData;
}Data ;
```

- ***Aspect algorithmique du programme :***
  1. ***Le Composant configuration:*** mettre en relation les composants via leurs interfaces.

```
configuration app{ }
implementation{
//composants utilisés
components Main, appM, LedsC, TimerC, DemoSensorC as Sensor,
 GenericComm as Comm, RandomLFSR;

 //initialisation
Main.StdControl -> TimerC.StdControl ;
Main.StdControl -> appM.StdControl ;
Main.StdControl -> Comm.Control;
Main.StdControl -> Sensor;

//Timer
appM.InteretTimer    -> TimerC.Timer[(uint8_t)unique("Timer")];
appM.TimerCaptage    -> TimerC.Timer[(uint8_t)unique("Timer")];

//communication
appM.ADC      -> Sensor;
appM.ADCControl ->Sensor;
appM.ReceiveInteret->Comm.ReceiveMsg[AM_DDMSG];
appM.ReceiveData->Comm.ReceiveMsg[AM_DDMSG_D];
```

```
appM.SendInteret->Comm.SendMsg[AM_DDMSG];
appM.SendData->Comm.SendMsg[AM_DDMSG_D];
appM.Random->RandomLFSR;
appM.Leds     -> LedsC.Leds;

}
```

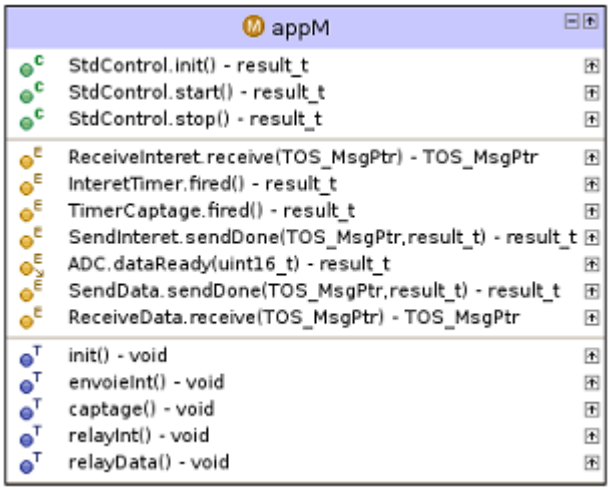


Les composants utilisés sont :

- **Main :** est le premier composant exécuté dans une application TinyOS.
- **appM :** nom du module.
- **LedsC :** ce composant implémente une interface Led qui Permet l'utiliation des leds
- **RandomLFSR :** ce composant implémente une interface Random permet de générer des nombres aléatoires.
- **TimerC :** ce composant implémente une interface Timer qui permet d'attendre et de lancer un événement après le délai d'attente.
- **GenericComm :** implémente les interfaces **SendMsg** et **ReceiveMsg** permettant d'envoyer et recevoir des messages respectivement.
- **DemoSensorC :** ce composant est un Capteur de démonstration auquel on associe le nom Sensor implémente une interface ADC et StdControl.

2. ***Le Composant module :*** Implémente les commandes et les événements avec du pseudo-C

```
module appM{

 provides {interface StdControl;}   //interface fournie

  uses  { // interfaces utilisées

        interface Timer as InteretTimer;
        interface Timer as TimerCaptage;
        interface ADC;
        interface StdControl as ADCControl;
        interface Leds ;
        interface ReceiveMsg as ReceiveInteret ;
        interface ReceiveMsg as ReceiveData ;
       interface SendMsg as SendData ;
       interface SendMsg as SendInteret;
       interface Random;

          }
}
```



3. Les commandes :

```
command result_t StdControl.init(){
                return SUCCESS;
                }
```

```
        command result_t StdControl.start(){
        return SUCCESS;
                }
```

```
        command result_t StdControl.stop(){
   return SUCCESS ;
                }
```

4. Les évènements :

```
        event TOS_MsgPtr ReceiveInteret.receive(TOS_MsgPtr pmsg) {
                return pmsg;
                }
```

```
        event result_t InteretTimer.fired()  {
                return SUCCESS;
                }
```

```
        event result_t TimerCaptage.fired()  {
                return SUCCESS;
                }
```

```
event result_t SendInteret.sendDone(TOS_MsgPtr pmsg, result_t success ){
        return SUCCESS;
        }
```

```
async event result_t ADC.dataReady(uint16_t this_data){
        return SUCCESS ;
        }
```

```
event result_t SendData.sendDone(TOS_MsgPtr pmsgD, result_t success ){
        return SUCCESS;
        }
```

```
event TOS_MsgPtr ReceiveData.receive(TOS_MsgPtr pmsgD) {
        return pmsgD;
        }
```

III. Description du langage NesC :

Le langage NesC est une extension du langage de programmation C qui a été conçu pour faciliter l'implémentation des structures et des composants TinyOS.

Figure III.1: symbole de langage NesC [14].

III.1. Présentation :

Le langage NesC (Network embedded system C) est un dialecte du C basé sur des composants. NesC est orienté pour satisfaire les exigences des systèmes embarqués.

De plus, il supporte un modèle de programmation qui agrège l'administration des communications, la concurrence et les événements ainsi que la capacité de réactions à ces événements.

NesC réalise aussi une optimisation dans la compilation du programme, en détectant les carrières possibles de données qui peuvent produire des modifications concurrentes au même état, à l'intérieur du processus d'exécution de l'application. Une carrière de données se produit quand plus d'une tâche peuvent simultanément accéder à la même section de mémoire (concurrence d'accès mémoire entre tâches), et quand au moins l'un des accès est un "write".

NesC simplifie aussi le développement d'applications et réduit la taille du code [14].

III.2 Les Principales caractéristiques de NesC :

NesC est constitué d'*interfaces* et de *composants*.

- Une interface peut être utilisée ou fournie.
- Les composants sont des ***modules*** ou des ***configurations***.

Une application est représentée comme un ensemble de composants, regroupés et rattachés entre eux (figure III.2). Les interfaces sont bidirectionnelles : le fournisseur de l'interface doit implémenter les commandes, alors que l'usager de l'interface doit implémenter ses événements.

Deux types de composants existent :

- Les modules, qui mettent en application des spécifications d'un composant.
- Les configurations, qui se chargeront de relier différents composants en fonction des interfaces (commandes ou événements).

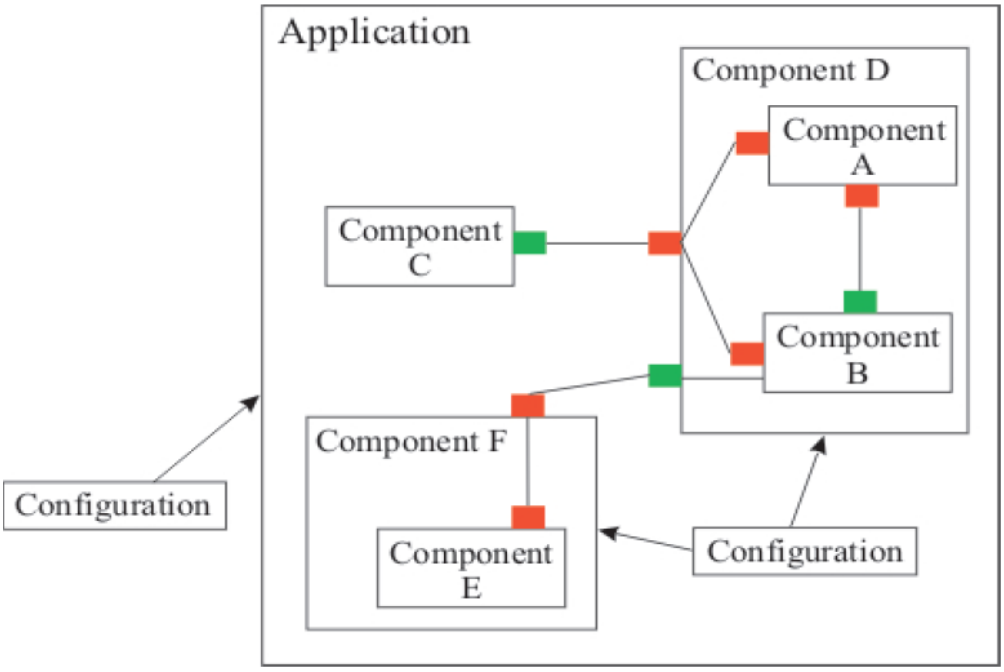


Figure III.2 : Architecture d'une application NesC. [15]

III.3. Les fichiers dans NesC :

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

1. Interface : Ce type de fichier déclare les services fournis et les services qui seront utilisés.

- Ils se trouvent dans le répertoire **/tos/interface**. (Exemple : StdControl.nc).

2. Module : Le type Module contient le code de l'application, en mettant en œuvre une ou plusieurs interfaces. (Exemple : appM.nc)

3. Configuration : Dans ces fichiers on déclare la manière de lier les différents composants et comment effectuer le contrôle des flux. (Exemple : app.nc).

II.4. Concepts principaux dans NesC :

1. Composants :

TinyOS définit un nombre important de concepts qui sont exprimés dans NesC.

Les applications NesC sont construites par des composants avec des interfaces bidirectionnelles définies.

NesC définit un modèle basé sur les tâches et les gestionnaires d'événements matériels, et détecte les accès concurrents aux informations pendant la compilation.

Un composant, du point de vue de la programmation, est composé de plusieurs sections et l'ensemble de toutes ces sections constitue le composant [12].

2. Configurations :

C'est à cet endroit que l'on déclare les autres composants dont se servira l'application. Cette possibilité offerte par le langage permet de faire de la programmation modulaire et de réutiliser des composants préalablement définis. La structure de la partie Configurations est la même que celle de la partie Implémentation.

Exemple :

Configuration Monapp { ……}

3. Implémentations :

Cette section définit les connexions entre les différents composants qu'utilise l'application.

Exemple :

```
implementation {
components Main, MonappM ;          //sont des composants utilisés par l'application
Main . StdControl -> MonappM. StdControl ;
//relier le composant MonappM avec le composant Main pour démarrer
}
```

4. 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 réaliser dans l'application. Cette partie est à son tour divisée en trois sous-sections :

Exemple :

```
module MonappM {
provides {... }   //interfaces fournies
uses {... }       //interfaces utilisées
}
implementation {... }
```

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

```
provides {
interface    //interface que nous fournissons ;
}
```

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). Pour faire cela, on a besoin de respecter quelques règles : si nous utilisons une interface, il nous faut avoir dans la section « implementation » un lien « wiring » reliant cette interface avec un composant qui la fournit.
Finalement, utiliser une interface oblige implicitement à gérer les événements pouvant se produire du fait d'avoir utilisé cette interface précise.

```
uses {
interface          //interface que nous utilisons ;
}
```

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

- Les variables globales utilisées dans l'application.
- Les fonctions qu'il implémenter pour les interfaces fournies.
- Les événements qu'il doit implémenter pour les interfaces utilisées.

III.5. Types des données :

Les types de données qui peuvent être utilisés en NesC sont tous ceux que fournit le langage C standard plus quelques autres qui sont très utiles pour la construction de paquets puisqu'ils fournissent à l'utilisateur le nombre de bits qu'ils occupent (ceci est important au

moment de la transmission des informations par l'intermédiaire des ondes radio). Ces types additionnels sont :

- uint16_t : entier non signé sur 16 bits.
- uint8_t : entier non signé sur 8 bits.
- result_t : utilisé pour savoir si une fonction a été exécuté avec succès ou non, c'est comme un booléen mais avec les valeurs SUCCESS et FAIL. (retour de fonction)
- bool : valeur booléenne qui peut être TRUE ou FALSE.

En NesC il est possible de faire une utilisation dynamique de la mémoire mais ce n'est pas très recommandé (à moins que cela ne soit absolument nécessaire). Pour pouvoir l'utiliser il existe un composant spécial appelé MemAlloc qui permet une gestion dynamique de la mémoire [12].

III.6. Types de fonctions en NesC :

En NesC les fonctions peuvent être de types très variés [15].

- Des fonctions classiques avec la même sémantique qu'en C et la façon de les invoquer est aussi la même qu'en C.
- Des types supplémentaires de fonctions : task, event, command.

  - **Les fonctions « command »** sont principalement des fonctions qui sont exécutées de manière synchrone, c'est- à-dire que lorsque elles sont appelées elles sont exécutées immédiatement. La manière d'appeler ce genre de fonction est : ***call interface . nomFonction () ;***

  - **Les fonctions « task »** sont des fonctions qui sont exécutées dans l'application, utilisant la même philosophie que les threads, c'est principalement une fonction normale qui est invoquée de la maniéré suivante : ***post nomTache () ;***

    Immédiatement après son invocation, l'exécution du programme qui l'a invoqué se poursuit.

- **Les fonctions « event »** sont des fonctions qui sont appelées quand on relèvera un signal dans le système, elles possèdent principalement la même philosophie que la programmation orientée événements, de sorte que, lorsque le composant reçoit un événement, on effectue l'invocation de cette fonction.

  Il existe une méthode pour pouvoir invoquer manuellement ce type de fonctions : ***signal interface . nomEvenement () ;***

IV. Simulateur de réseaux de capteur :

L'objectif de la plateforme est de simuler un réseau de capteur, ce qui sous-entend que nous n'utilisons pas de capteurs réels. Dans cette optique, nous présentons un simulateur que nous avons étudié, TOSSIM.

IV.1. Définition :

TOSSIM est le simulateur de TinyOs. Il permet de simuler le comportement d'un capteur (envoie/réception de messages via les ondes radios, traitement de l'information, etc.…) au sein d'un réseau de capteurs.

Pour une compréhension moins complexe de l'activité d'un réseau, TOSSIM peut être utilisé avec une interface graphique, TinyViz, permettant de visualiser de manière intuitive le comportement de chaque capteur au sein du réseau.

IV.2.Description :

TinyViz est une application graphique qui donne un aperçu du réseau de capteurs à tout instant, ainsi que les divers messages qu'ils émettent/reçoivent. Il permet de déterminer un délai entre chaque itération des capteurs afin de permettre une analyse pas à pas du bon déroulement des actions [13].

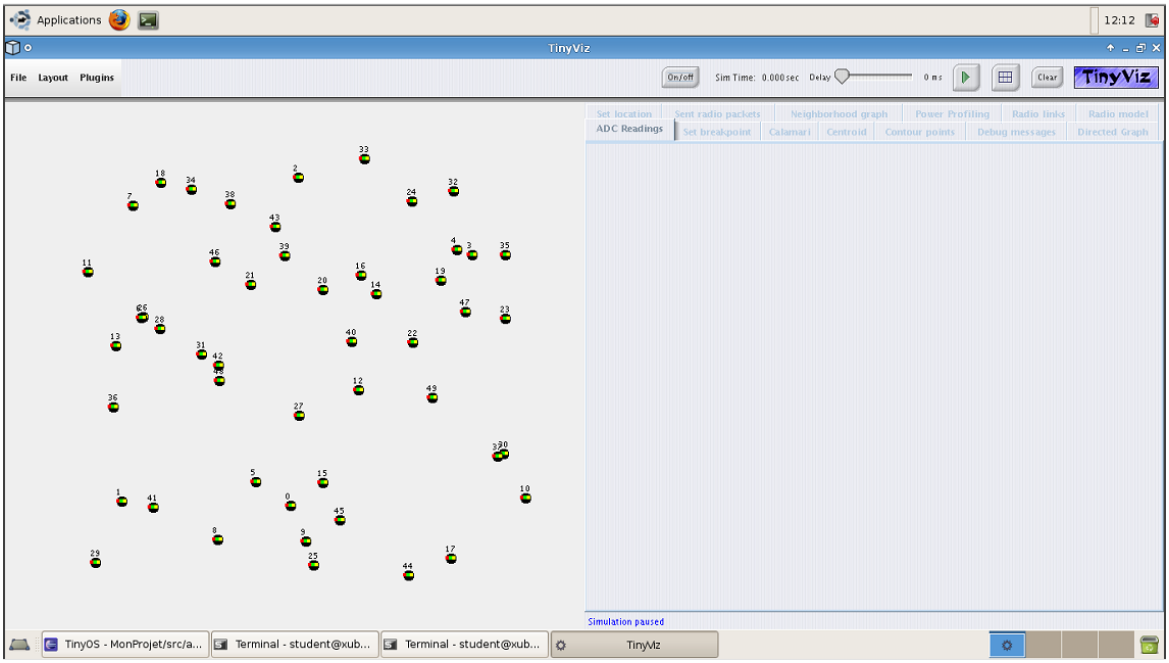


Figure III.3 : Fenêtre graphique TinyViz.

On peut voir dans la partie de gauche de la **figure III.3** tous les capteurs. Ils sont déplaçables dans l'espace, ou si l'on possède une configuration particulière, on peut charger un fichier qui positionnera chaque capteur à l'emplacement spécifié.

La partie du haut rassemble toutes les commandes permettant d'intervenir sur la simulation (dans l'ordre) :

- On/Off : met en marche ou éteint un capteur. Le délai du timer : permet de sélectionner la durée au bout de laquelle se déclenche le timer.
- Le bouton « Play » : il permet de lancer la simulation où de la mettre en pause.
- Les grilles : il affiche un quadrillage sur la zone des capteurs afin de pouvoir les situer dans l'espace.
- Le bouton « Clear » : il efface tous les messages qui avaient été affichés lors de la simulation. Le bouton de fermeture : arrête la simulation et ferme la fenêtre.

Chaque onglet contient un plugin qui permet de visualiser la simulation de façon plus ou moins détaillée. Le plugin « Radio Links » permet de visualiser graphiquement par des

flèches, les échanges effectués entre deux capteurs (unicast), ainsi que les messages émis par un capteur à l'ensemble du réseau (broadcast).

Figure III.4: Visualisation des messages radio.

Dans l'exemple de la **figure III.4**, le capteur 1 a envoyé un broadcast (repéré par un cercle) à tous les capteurs dans son rayon d'action. Tous les capteurs de sa zone de couverture lui répondent (flèches) par un message direct.

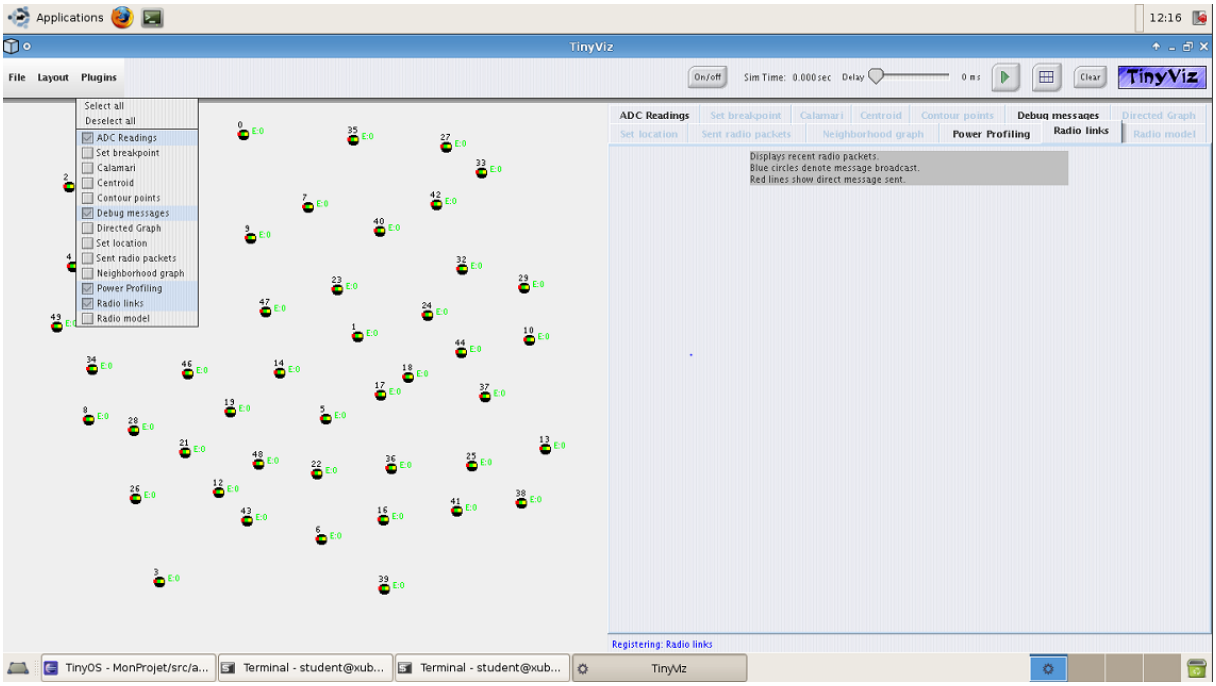


Figure III.5 : plugins de TinyViz .

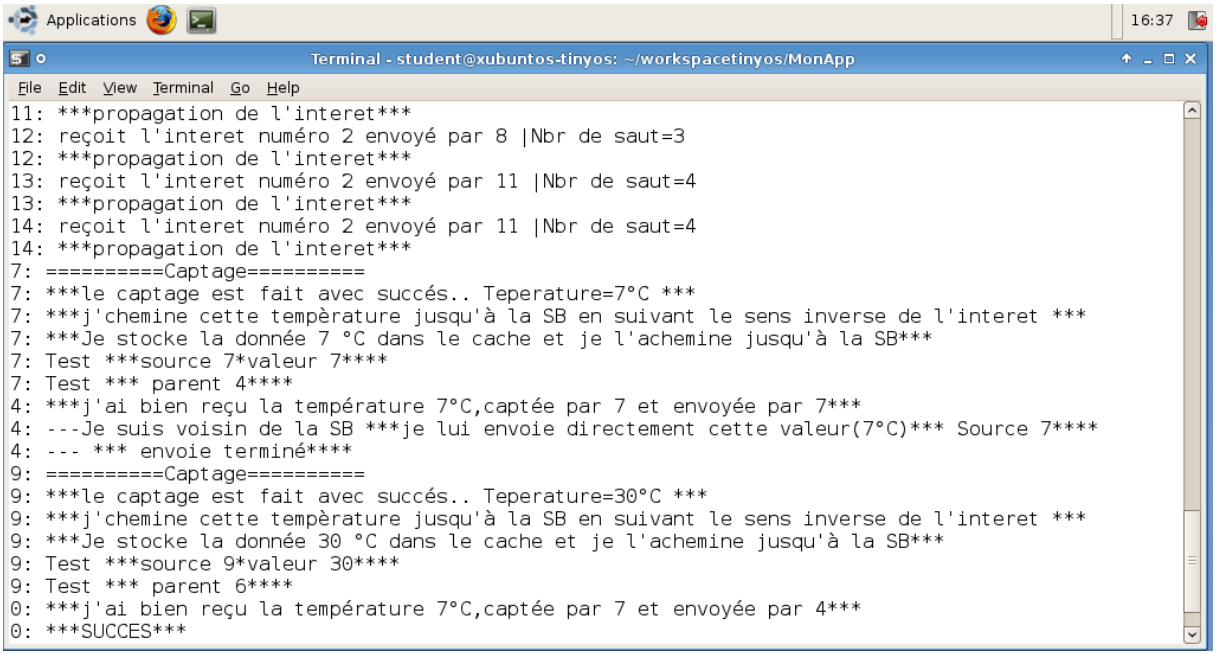


Figure III.6 : Simulation de « Diffusion Directe ».

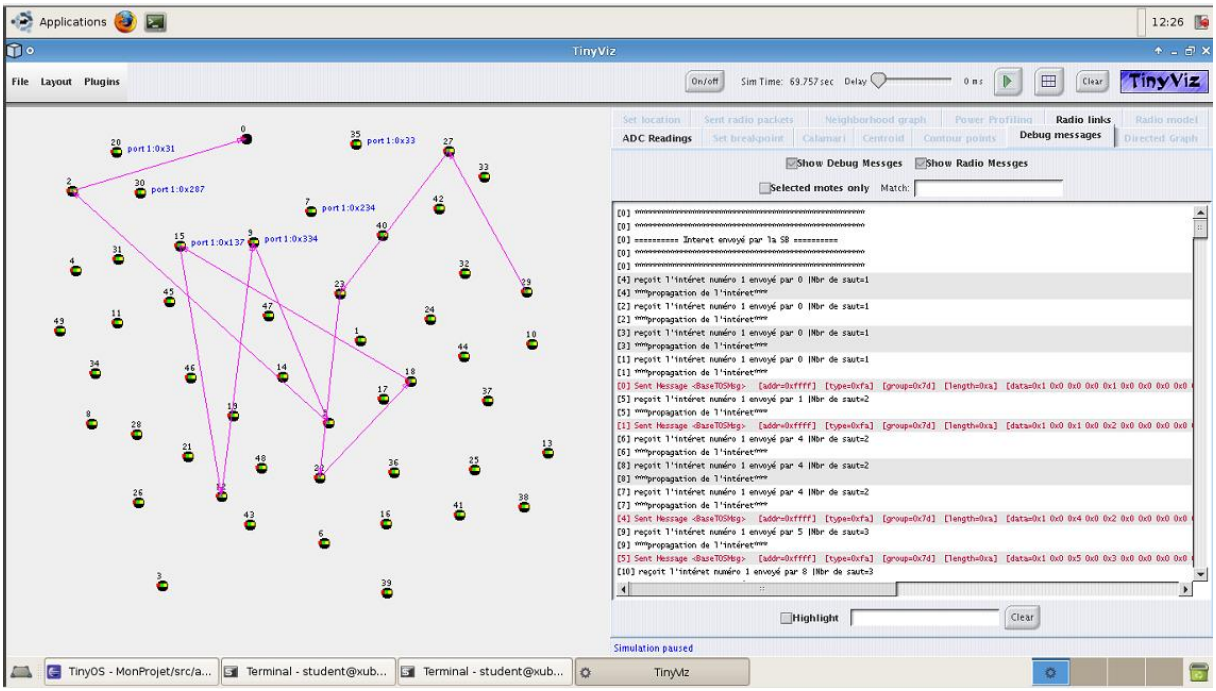


Figure III.7 : Debug message de TinyViz.

IV. 3.Avantages :

TOSSIM permet de simuler fidèlement le comportement d'un réseau de capteurs. Il permet également l'affichage des évènements et des messages de débogage pour chaque capteur mais aussi simultanément pour l'ensemble des capteurs [13].

A cela se rajoute l'interface graphique TinyViz, qui permet la visualisation des échanges radios, conjointement aux messages de débogage, ce qui permet, à chaque instant de la simulation, d'avoir une vue globale de l'activité du réseau étudié.

TinyViz offre la possibilité de ralentir la simulation par un délai, afin d'observer le déroulement des évènements, ce qui est très intéressant lorsque le réseau est surchargé de messages.

En plus, il a une extension PowerTOSSIM, qui permet de modéliser la consommation énergétique des différents nœuds du réseau.

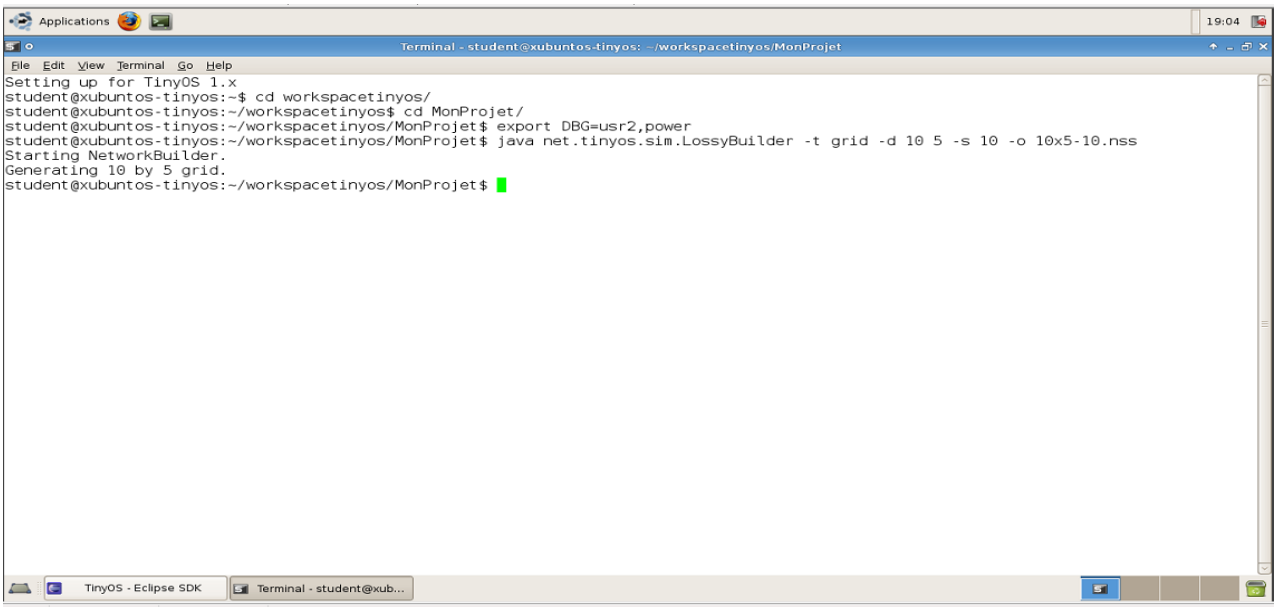


Figure III.8 : va générer une topologie lossy dans une grille de 10X2, avec des espacements de 10 pieds (3m), et la stocker dans le fichier "10x2-10.nss"

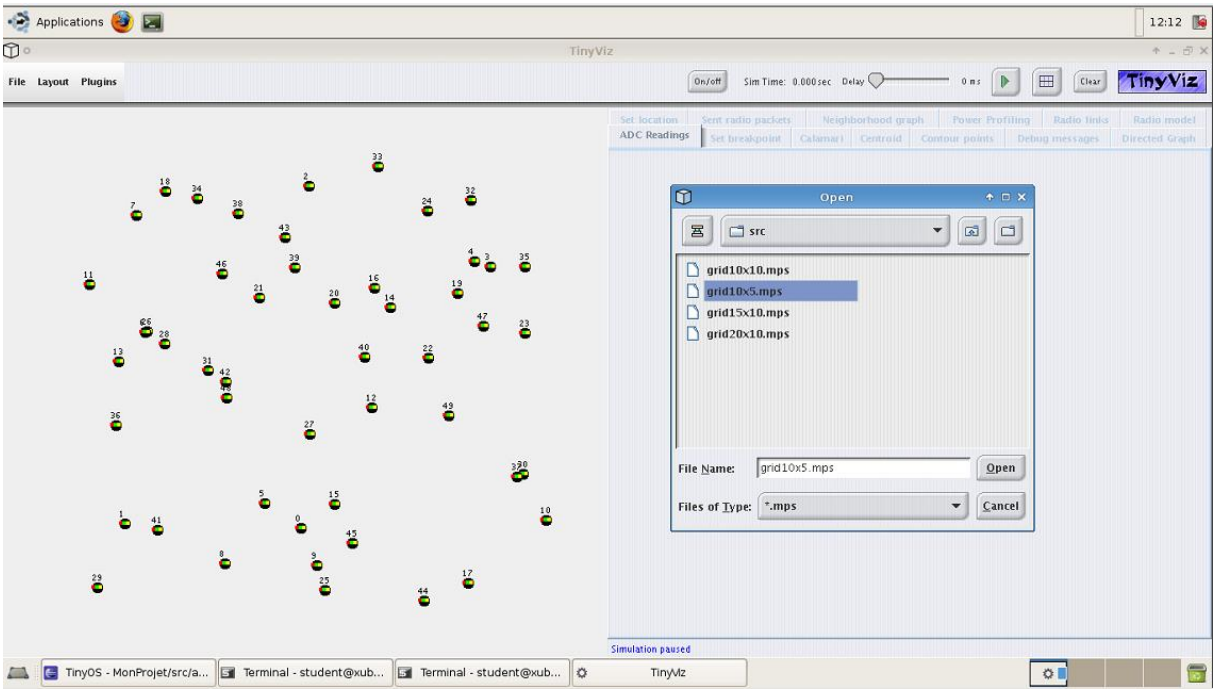
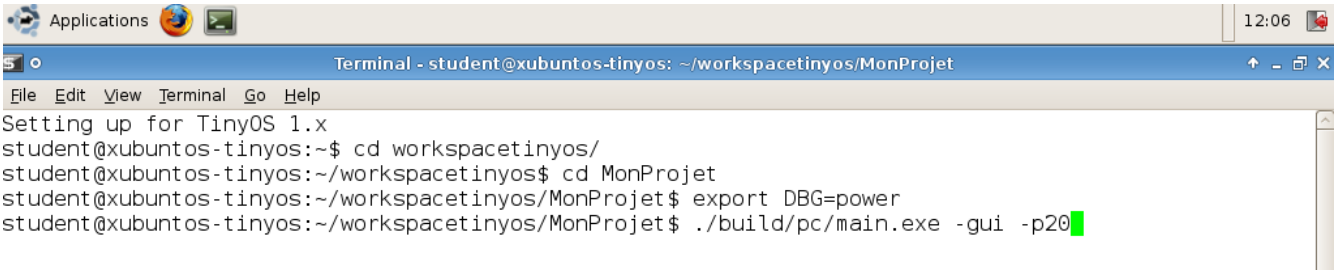


Figure III.9 : Télécharger une topologie pour les nœuds capteurs.

Le plugin Power Profiling de TinyViz nous permet de voir l'état de la consommation de l'énergie de l'application en cours d'exécution. En Tapant dans une Console Linux, les commandes ci-dessous:



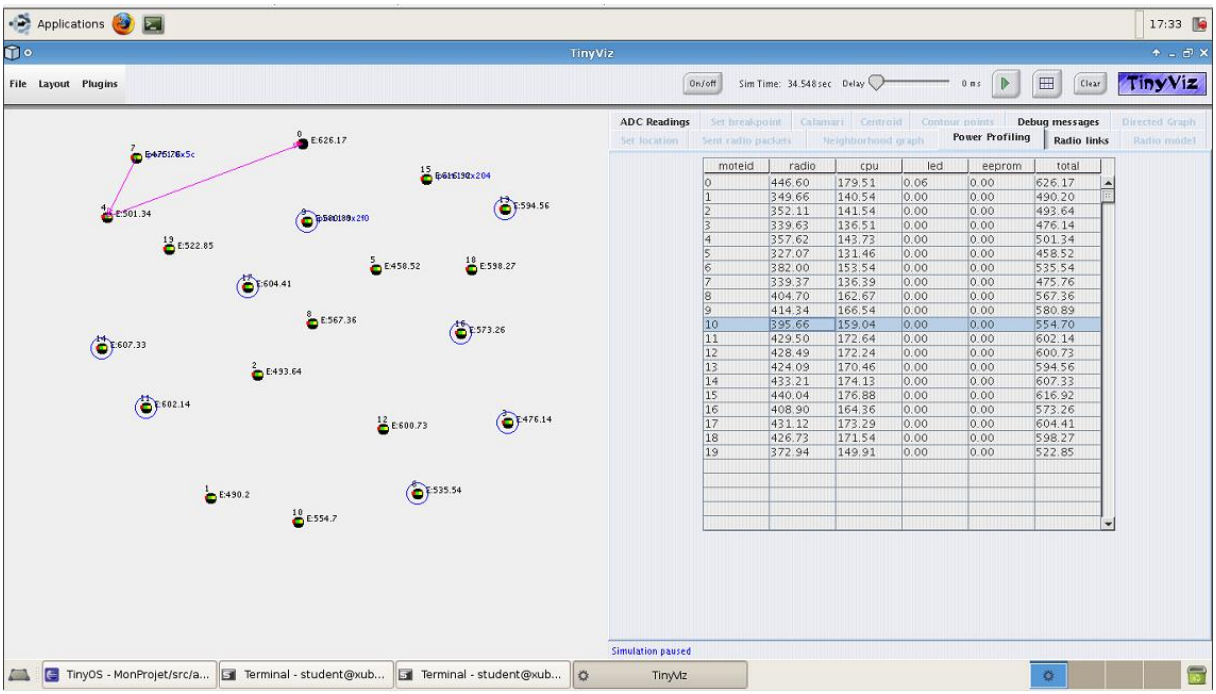


Figure III.10 : Simulation de la consommation énergétique « Diffusion Directe ».

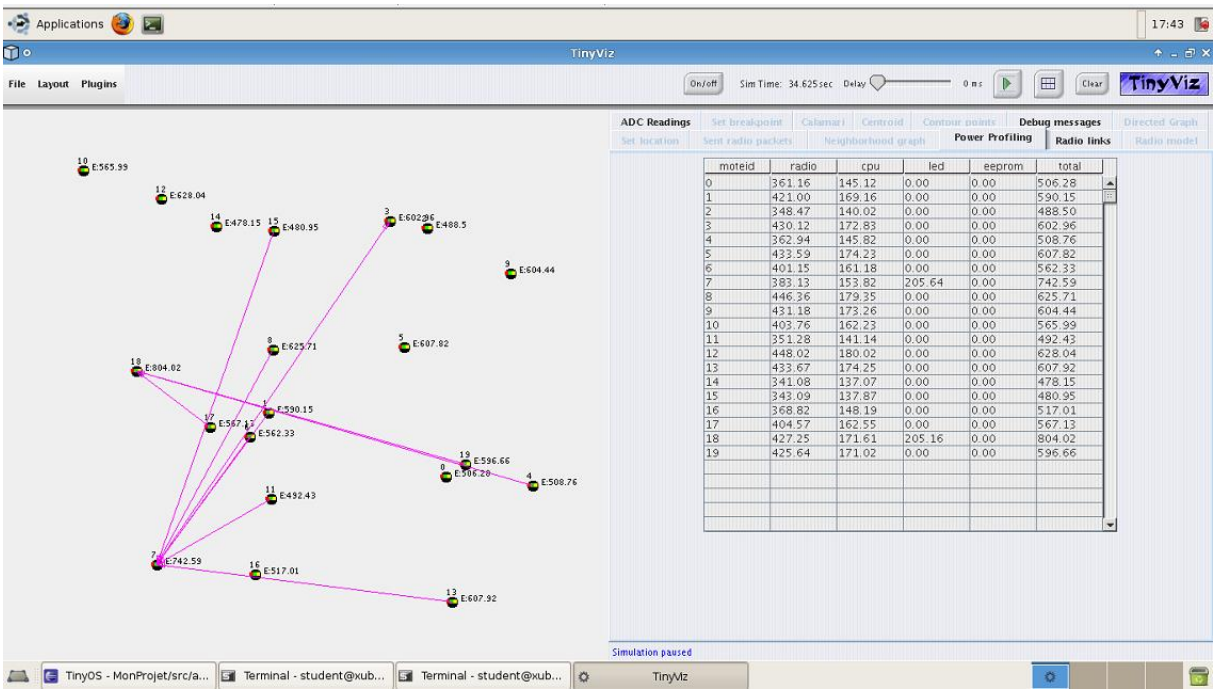


Figure III.11: Simulation de la consommation énergétique « LEACH ».

V- Comparaison entre LEACH et Diffusion Directe en termes d'énergie :

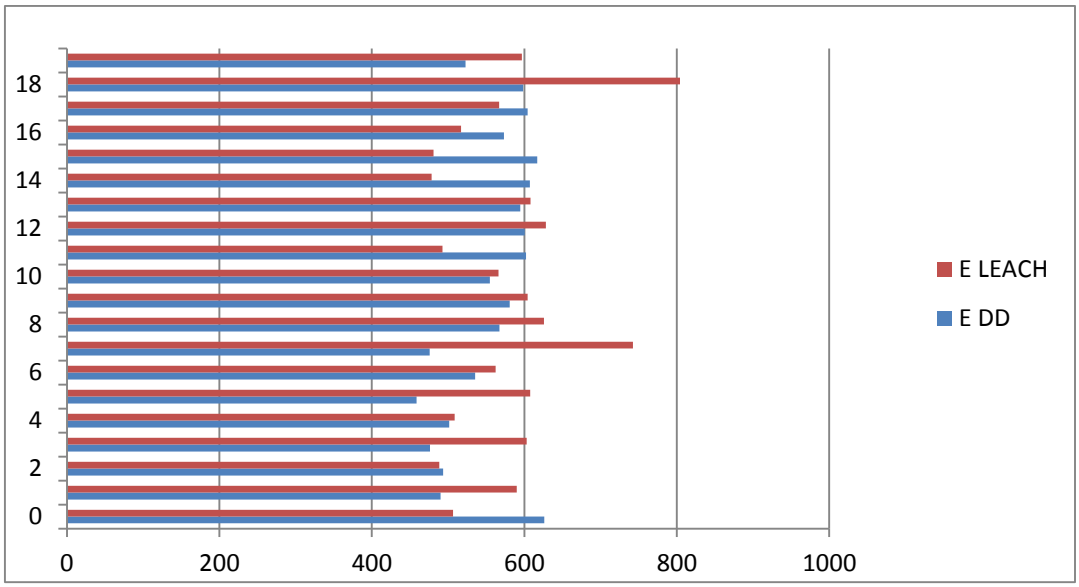


Figure III.12: Comparaison entre LEACH et Diffusion Directe en termes d'énergie

D'après les résultats de simulation de l'énergie, on constate que le protocole Diffusion Directe à un taux d'énergie consommée moins important que LEACH .Cela est normal vu qu'on a pallié au problème :

« Chaque nœud du réseau reçois un même message plusieurs fois de différents voisins » ;

par la solution suivante :

Chaque nœud doit envoyer qu'une seule fois un même message :

- Chaque message a un identifiant unique.
- Si les nœuds reçoivent à nouveau un message avec cet identifiant, ils ne le renvoient pas.

Concernant LEACH, la consommation d'énergie est élevée car :

- La communication directe avec le puits exige une consommation d'énergie importante des nœuds lointains.
- Les CHs sont les maillons les plus faibles car l'agrégation de données est centrée à leur niveau.

- La rotation du rôle de CHs permet d'équilibrer la consommation d'énergie mais ça nécessite des phases d'initialisation.
- La distribution des CHs n'est pas homogène.

VI. Conclusion :

Dans ce chapitre, nous avons présenté le langage NesC qui est un langage de programmation qui présente de grands avantages pour le développement d'applications pour des systèmes embarqués, et particulièrement pour les réseaux de capteurs sans fil.

Les simulations réalisées au moyen du simulateur TOSSIM ont mené à étudier les différentes caractéristiques de l'algorithme de Diffusion Directe et LEACH.

Conclusion générale

L'objectif de notre travail était d'implémenter et de simuler un algorithme de routage plat pour les RCSF cas : « Diffusion Directe », ensuite cet algorithme sera comparé avec un autre algorithme de routage hiérarchique qui est « LEACH » en termes d'énergie.
Ce travail peut ce résumer en deux parties :

- Dans la première partie, nous avons fait une étude théorique sur le routage dans les réseaux de capteurs sans fil. Nous avons mis l'accent sur les différents domaines d'application et les différents protocoles de routage ensuite on a présenté le fonctionnement du protocole Diffusion Directe.

- Dans la deuxième partie, nous avons simulé le protocole Diffusion Directe et LEACH par rapport à leurs consommations énergétiques.

Les bénéfices de notre travail :

Le travail que nous avons accompli nous a permis :

- D'avoir les notions de bases sur les différents algorithmes de routage et le fonctionnement détaillé de diffusion directe.

- De nous initier aux différentes étapes à suivre pour l'implémentation de « Diffusion Directe ».

- D'acquérir de nouvelles connaissances sur le langage nesC et le système d'exploitation TinyOS.

En comparant ces deux protocoles nous avons vu et observé des résultats pour un réseau de capteurs dont le nombre est peu important, une topologie de 20 nœuds, que Diffusion Directe consomme moins d'énergie par rapport à LEACH parce qu'on a fait en sorte que chaque nœud du protocole diffusion directe doit renvoyer qu'une seule fois le message.

Les réseaux de capteurs constituent un domaine de recherche très vaste. Ils ont de nombreuses perspectives d'application dans des domaines très variés. Il reste encore de

nombreux problèmes à résoudre dans ce domaine afin de pouvoir les utiliser dans des conditions réelles.

Afin d'être adaptés à un environnement réel, nos algorithmes peuvent être toujours améliorées en introduisant la mobilité des nœuds et la maintenance des chemins.

Les performances de nos algorithmes ont été prouvées à l'aide du simulateur TOSSIM, il serait préférable de réaliser ces études dans des conditions réelles de manière à comparer les performances réelles par rapport à celles effectuées par des simulations.

ANNEXE

Le Système d'exploitation TinyOS

I. Présentation de la plate-forme TinyOS:

TinyOS (**Tiny O**perating **S**ystem), est un *système d'exploitation* et un *environnement de développement* ***Open Source***, conçu pour les réseaux de capteurs sans fil. Le caractère open source, permet à ce système d'être régulièrement enrichie par une multitude d'utilisateurs. Cette plate-forme logicielle, et les applications tournant dessus ont été originalement réalisées en NESC (**N**etworks **E**mbedded **S**ystem **C**), un langage orienté composant (component) syntaxiquement proche du C. Il respecte une architecture basée sur une association statique de composants à la compilation, réduisant ainsi la taille du code nécessaire à sa mise en place. Cela s'inscrit dans le respect des contraintes de mémoires qu'observent les senseurs, pourvus de ressources très limités dues à leur miniaturisation.

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

TinyOS s'appuie sur un fonctionnement évènementiel, c'est-à-dire qu'il ne devienne actif qu'à l'apparition de certains évènements. Le reste du temps, le senseur se trouve en état de veille, vu les faibles ressources énergétiques des senseurs, garantissant ainsi une durée de vie maximale. Ce type de fonctionnement permet une meilleure adaptation à la nature aléatoire de la communication sans fil entre les capteurs.

L'objectif de TinyOS est de faciliter l'accès aux ressources matérielles par les processus applicatifs, et de permettre une meilleure portabilité des applications, en rendant l'utilisation des ressources matérielles la plus transparente possible.

II. Propriétés de TinyOS :

TinyOS a été créé pour répondre aux caractéristiques, et aux nécessités des WSNs 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.
- ✓ **Event-driven** : Le plus gros avantage de TinyOS est qu'il est basé sur un fonctionnement événementiel (concurrence), c'est à dire s'appuie sur la gestion des évènements qui se produisent instantanément. Ainsi, l'activation de tâches, leur interruption ou encore la mise en veille du senseur, s'effectue à l'apparition d'évènements, ceux-ci ayant la plus forte priorité. Ce fonctionnement évènementiel (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 évènements :*** 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, mais l'initialisation et la terminaison d'une tâche sont des fonctions séparées. Les tâches ne peuvent pas prendre de paramètre en entrée.

- ✓ **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. Dans le cas d'un système strict, aucune échéance ne tolère de dépassement, contrairement à un système temps réel. TinyOS se situe au-delà de ce second type, car il n'est pas prévu pour avoir un fonctionnement temps réel.

- ✓ **Consommation d'énergie :** TinyOS a été conçu pour réduire au maximum la consommation en énergie du senseur. Ainsi, lorsqu'aucune tâche n'est pas active, il se met automatiquement en veille.

- ✓ **Communication :** TinyOS utilise un modèle event/command, un ordonnancement FIFO non préemptif, en plus pas de séparation kernel/application.

- ✓ **Langage :** TinyOS a été programmé en langage NesC, que nous allons détailler plus loin.

- ✓ **Disponibilité et sources :** TinyOS est un système principalement développé et soutenu par l'université américaine de Berkeley, qui le propose en téléchargement sous la licence BSD, et en assure le suivi; Ainsi, l'ensemble des sources sont disponibles pour de nombreuses cibles matérielles.

III. Cibles possibles pour TinyOS :

TinyOS peut être implémenté sur, un PC, un senseur (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.

IV. Gestion, Allocation de la mémoire :

Il est très important d'aborder la façon avec laquelle un système d'exploitation gère la mémoire, c'est encore plus significatif lorsque ce système travaille dans un espace restreint. TinyOS a une empreinte mémoire très faible, puisqu'il n'a besoin que de 300 à 400 octets

dans le cadre d'une distribution minimale. Il est primordial d'avoir 4 Ko de mémoire libre qui se repartissent entre les différents besoins suivant :

- **Allocation statique de la mémoire :** TinyOS ne gère pas l'allocation dynamique de la mémoire, et ne permet pas ni le heap (malloc), ni les pointeurs sur fonction.

- **La Pile (stack) :** elle sert de mémoire temporaire pour l'empilement, et le dépilement des variables locales (qui sont déclarées dans une méthode).

- **Les variables globales:** elles réservent un espace mémoire pour stocker des valeurs pouvant être accessibles depuis différentes tâches (disponibles per-frame, conservation de la mémoire, utilisation de pointeurs).

- **La mémoire libre:** pour tout le reste du stockage temporaire.

La notion d'allocation statique de la mémoire, simplifie grandement l'implémentation mais, par ailleurs, il n'existe pas de mécanisme de protection de la mémoire, ce qui rend TinyOS plus vulnérable aux crashs, et aux corruptions de la mémoire.
La figure 1 illustre le modèle mémoire de TinyOS

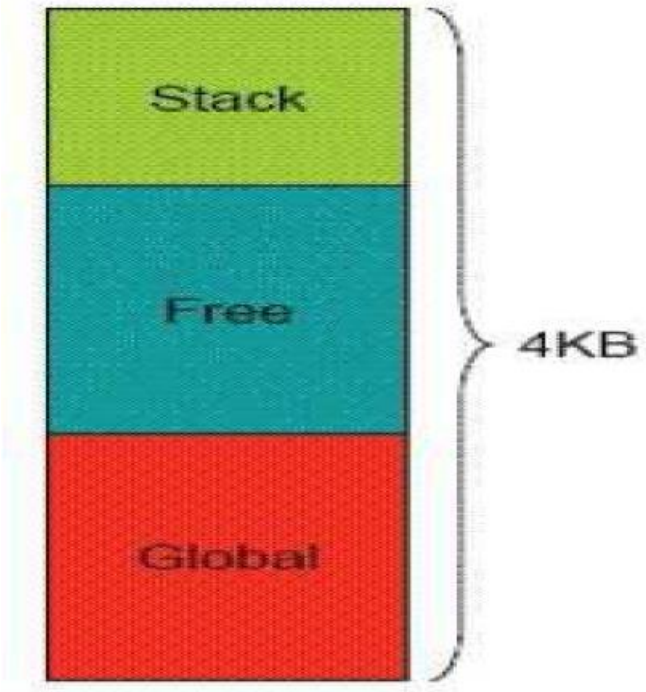


Figure 1 : Organisation de la mémoire dans TinyOS.

V. Modèle d'exécution de TinyOS :

V.1. Description de scheduler TinyOS:

Pour maintenir une grande efficacité requise par les réseaux de senseurs, TinyOS utilise une programmation orientée évènement. Ce modèle permet un très haut niveau de concurrence pour un espace très réduit de mémoire. D'autre part, le choix d'un scheduler déterminera le fonctionnement global du système.

TinyOS fournit une hiérarchie d'ordonnancement à deux niveau, composé de :

- ***Tâches ;***
- ***Handler d'évènements matériels*** (Préemptif) ;

Et définie **une politique FIFO** (disposant d'une capacité de 7).

Il est très significatif d'évoquer que le mot-clé ***async*** déclare une *commande* ou un *événement* qui peut être exécuté par un gestionnaire d'événements matériel (pouvoir être exécuté à tout moment i.e. Préemption d'un autre code).

Le tableau suivant récapitule les différentes actions dans TinyOS :

Action	*Utilisation*
Tâche	Travaux de *longue durée (*background de traitement de données)
Commande	Exécution d'une fonctionnalité précise dans un autre composant
Evénement	Equivalent logiciel à une interruption matérielle

Tableau 1 : Différentes actions dans TinyOS

Abréviations

RCSF : Réseau de **C**apteurs **S**ans **F**il

WSN: Wireless **S**ensor **N**etwork

MANET: Mobile **A**d hoc **NET**works

IBM: International **B**usiness **M**achines

ADC: Analog **D**igital **C**onverter

GPS: Global **P**osition **S**ystem

LEACH: Low **E**nergy **A**daptive **C**lustering **H**ierarchical

QoS: Quality **o**f **S**ervice

DD: Directed **D**iffusion

SQL: Structured **Q**uery **L**anguage

SPIN: Sensors **P**rotocols for **I**nformation via **N**egotiation

DVS : Dynamique **V**oltage **S**caling

CSIP: Collaborative **S**ignal and **I**nformation **P**rocessing

NesC: Network **e**mbedded **s**ystem **C**

Bibliographie

&

Sitographie

[01] Kamel BEYDOUN, "Conception d'un protocole de routage hiérarchique pour les réseaux de capteurs", Thèse de Doctorat, U.F.R des Sciences et Techniques de l'université de Franche-Comté, Décembre 2009.

[02] Gérard CHALHOUB, "Les réseaux de capteurs sans fil", Conférence, Université de Clermont-Ferrand, Novembre 2010.

[03] http://igm.univ-mlv.fr/~dr/XPOSE2006/Bunel/Presentation.html#unreseau, le 11/06/2013.

[04] Samir ATHMANI, Protocole de sécurité Pour les Réseaux de capteurs Sans Fil, Pour l'obtention du Magistère en Informatique, le : 15/07/2010.

[05] S. Ziane and A. Mellouk, "A swarm intelligent scheme for routing in mobile ad networks". Systems Communications, IEEE, Aug 2005.

[06] Paolo Santi, "Topology Control in Wireless Ad Hoc and Sensor Networks". Hardcover, july 2005.

[07] S. Kumar, D. Shepherd, and F. Zhao, "Collaborative signal and information processing in micro-sensor networks". IEEE Signal Processing Magazine, March 2002.

[08] N. Li and J. C. Hou, "BLMST: a decentralized, power-efficient broadcast algorithm for wireless sensor networks". Accepted for publication in ACM Baltzer Wireless Networks (WINET), 2005.

[09] M. Younis, M. Youssef, and K. Arisha, "Energy-Aware Routing in Cluster-Based Sensor Networks". In Proceedings of the 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS2002), (Forth Worth, TX), 2002.

[10] C. Intanagonwiwat, R. Govindan, and D.Estrin. Directed Diffusion : a scalable and robust communication paradigm for sensor networks. In ACM Mobicom, pages 56Ŕ67. ACM, Aout 2000.

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

[12] Wassim ZNAIDI **, «** Modélisation formelle de réseaux de capteurs à partir de TinyOS »,projet fin d'etude, école polytechnique de tunisie,2006.

[13] Fares Abdelfatah, « Développement d'une bibliothèque de capteur sans fil », diplôme de master en informatique, université Montpellier 2, avril 2008.

[14] Cedric BASTIEN ,« developpement d'un outil permettant le suivi d'un objet mobile », projet de fin d'etude,2008.

[15] H. Alatrista, S. Aliaga, K. Gouaich, J. Mathieu, « Implémentation de protocole sur une plateforme de réseaux de capteurs sans-fils», mémoire master, Université de Montpellier

[16] nesC Language Reference Manual. D-Gay, D-Culler, Ph-Levis. September 2002.

[17] http://moodle.utc.fr/