Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire

Université Abou Bakr Belkaid– Tlemcen

Faculté des Sciences

Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme

Ingénieur d'Etat en Informatique

Option : S I

Thème

La Géo-localisation dans les Réseaux de Capteurs sans Fil.

Réalisé par :

- SAHRAOUI Belkheyr

Présenté le Juillet 2011 devant le jury composé de MM.

- *Mr Benamar Abdel krim* *(Président)*
- *Mme Labraoui Nabila* *(Encadreur)*
- *Mme Belhabi* *(Examinateur)*
- *Mlle Benmansour* *(Examinateur)*

Année universitaire: 2010-2011

REMERCIEMENTS

Grâce à Dieu vers lequel vont toutes les louanges, ce travail s'est accompli.

Je tiens à exprimer mes vifs remerciements envers toutes les personnes qui ont contribué au bon déroulement de ce travail.

En particulier, j'exprime ma gratitude à mon encadreur Mme Labraoui Nabila, Chercheur au laboratoire STIC et maître assistant à l'université ***A. B*** *de Tlemcen.*

Dédicaces

A mes chers parents et ma grande mère pour leur soutien moral durant mes études.

A tout ma famille proche ou lointaine.

Et à tous mes fidèles amis, à tous ceux que j'aime……

Belkheyr SAHRAOUI

Sommaire

La Localisation dans les réseaux de capteurs sans fil

Description de l'architecture de la plateforme TinyOS : un système d'exploitation pour les réseaux de capteurs

Implémentation et Evaluation de l'Algorithme de Localisation DV-Hop

Liste des Figures

Liste des Tableaux

Liste des abréviations

CPU Central Processing Unit

GPS Global Positioning System

IRNSS Indian Regional Navigational Satellite System

TinyOS Tiny Open Source

WSN Wireless Sensor Networks

Wi-Fi Wireless fidelity

WLAN Wireless local area network

RSSI Received Signal Strength Indicator

RCSF Réseaux de capteurs sans fil

AoA Angle of Arrival

TDoA Time Difference of Arrival

Introduction générale

Introduction générale

Dans les premiers âges de l'informatique, la difficulté pour les chercheurs était d'adapter les différents algorithmes aux contraintes matérielles imposées par les ordinateurs (puissance de calcul, mémoire limitée, ...). Même si ces difficultés n'ont pas disparu, leurs impacts tendent à diminuer tant la puissance des ordinateurs s'accroît à une vitesse surprenante. Il faut des applications telles que la cryptographie, les analyses ADN en bioinformatique, la fouille de données, ... pour redonner vie à ces limites imposées par le matériel. Toutefois, les avancées technologiques conduisent à une nouvelle évolution du paradigme. Incontestablement, ce début de vingt et unième siècle est placé sous le signe de la communication. Après le phénomène Internet, la démocratisation des technologies sans fil révolutionne les moyens de communication avec notamment l'apparition de réseaux spontanés ou réseaux ad hoc. L'hétérogénéité de ces réseaux et l'absence d'infrastructure accroissent leurs intérêts et ouvrent de nouvelles perspectives avec, par exemple, l'émergence de réseaux mobiles.

Autre événement majeur de l'informatique, la décomposition et la répartition des calculs sur plusieurs machines a permis d'outrepasser les limites des processeurs. En effet, pourquoi se limiter à une seule et même unité de calcul ? Dès lors qu'un calcul peut se décomposer, sa résolution sera d'autant plus rapide que le nombre d'unités de calcul sera élevé.

Enfin, indépendamment de ces avancées technologiques et algorithmiques, il existe une tendance dans le domaine des microsystèmes électroniques qui s'est accentuée ces dernières années : il s'agit de la miniaturisation. C'est dans ce contexte qu'apparaît une nouvelle génération d'appareils : les capteurs communiquant. Ils sont généralement de petite taille, dotés d'une unité de calcul et capables de communiquer entre eux. On parlera alors de réseaux de capteurs. Les fonctionnalités des capteurs étant limitées de par leurs faibles ressources, les applications doivent être adaptées à leurs caractéristiques.

L'émergence de ce nouveau champ d'étude va pousser les informaticiens à retourner aux sources de leur histoire ...

La tâche première d'un capteur est de détecter un événement (par exemple, un changement de température, des mouvements, des vibrations, ...). Il est donc capable de récolter des données relatives à son environnement, de les traiter, puis, si nécessaire, de les communiquer à des capteurs voisins via un médium sans fil. Le déploiement de ce type d'appareils forme alors un réseau qui peut être utilisé dans des domaines militaires (par exemple, le suivi de déplacement des troupes ennemies), civils (la détection de feux de forêt), médicaux (le suivi des patients), animaliers (l'étude des migrations d'espèces), etc... Dans plusieurs exemples, les capteurs sont mobiles. Il faut donc distinguer deux types de réseaux : les réseaux statiques et les réseaux mobiles.

Si leurs perspectives d'utilisation sont claires et attrayantes, les problématiques qu'engendrent ces réseaux n'en sont pas moins nombreuses. A priori, ils ne dépendent d'aucune infrastructure et les capteurs n'ont aucune information relative au réseau auquel ils appartiennent. De plus, étant construits de façon ad hoc, ces réseaux doivent être auto-organisant. Parmi les problèmes cruciaux, deux d'entre eux peuvent être cités :

- celui de la localisation : il s'agit d'attribuer une position géographique (exacte ou estimée) aux capteurs ; une application telle que la surveillance des feux de forêt, par exemple, n'aurait pas de sens sinon.
- celui du routage : il consiste à acheminer un message d'un capteur vers un autre. Souvent, les réseaux contiennent une station de base chargée de collecter l'ensemble des informations perçues par les capteurs. Il s'agit alors de transmettre ces informations point à point vers cette station de base.

Bien d'autres problèmes tels que l'adressage ou la diffusion sont liés à ce type de réseau. Chaque méthode proposée doit assurer l'auto-stabilisation du réseau en garantissant la convergence vers une solution stable. D'autre part, l'énergie des capteurs étant limitée, cette contrainte doit être prise en compte afin d'allonger la durée de vie du réseau.

Ce mémoire se focalise sur la problématique de la localisation statique dans les réseaux de capteurs sans fil. Nous nous sommes également intéressé de près à

l'algorithme de localisation DV-HOP afin de l'implémenter et d'évaluer ses performances selon la métrique de la précisions.

La suite de ce document est constituée de 4 chapitres :

Le chapitre 1 : nous présentons une description générale des réseaux de capteurs sans fil ainsi que leurs caractéristiques, contraintes et spécificités.

Le chapitre 2 : est consacré à la problématique de la localisation.

Le chapitre 3 : présente une description pour le système TinyOS et la langage NesC.

Le chapitre 4 : constitue le cœur de notre travail, dans ce chapitre nous présentons le simulateur TOSSIM avec une brève description de ces composants, ses principales caractéristiques et fonctionnalités. Par la suite nous donnons les résultats sous forme de graphes de plusieurs simulations, effectuées pour obtenir des mesures pour voir la précision de méthode

En fin de ce mémoire, une conclusion est donnée pour résumer les apports essentiels de notre travail, les ouvertures et les perspectives pour la future.

Chapitre 1

Réseaux de capteurs sans fil: description, protocole

1. Introduction

Les avancées technologiques récentes confortent la présence de l'informatique et de l'électronique au cœur du monde réel. De plus en plus d'objets se voient ainsi équipés de processeurs et de moyens de communication sans fil et mobiles, leur permettant de traiter des informations mais également de les transmettre [FA08]. Les réseaux de capteurs sans fil (RCSF) entrent dans ce cadre. En effet, ceux-ci sont constitués d'un ensemble de petits appareils, ou capteurs, possédant des ressources particulièrement limitées, mais qui leur permettent néanmoins d'acquérir des données sur leur environnement immédiat, de les traiter et de les communiquer à une station de base ou point de collecte.

2. Présentation un capteur sans fil

Les capteurs sont des dispositifs de taille extrêmement réduite avec des ressources très limitées, autonomes, capable de traiter des informations et de les transmettre, via les ondes radio, à une autre entité (capteurs, unité de traitements...) sur une distance limitée à quelques mètres. La figure 1.1 illustre le schéma interne d'un capteur sans fil.

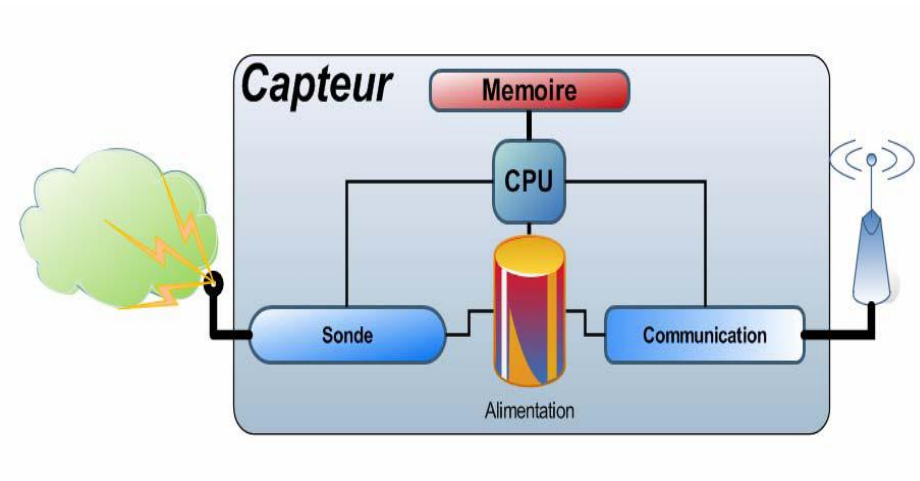

Figure 1. 1 : Schéma interne d'un Capteur sans fil.

Les réseaux de capteurs utilisent un très grand nombre de ces capteurs, pour former un réseau *sans infrastructure* établie. Un capteur analyse son environnement, et propage les données récoltées aux capteurs appartenant à sa zone de couverture. Chaque capteur

relayant l'information sur sa propre zone de couverture, le réseau se trouve entièrement couvert.

3. Composants d'un capteur sans fil

Un nœud capteur contient quatre unités de base : l'unité de captage, l'unité de traitement, l'unité de transmission, et l'unité de contrôle d'énergie. Il peut contenir également, suivant son domaine d'application, des modules supplémentaires tels qu'un système de localisation (GPS), ou bien un système générateur d'énergie (cellule solaire). On peut même trouver des micro-capteurs, un peu plus volumineux, dotés d'un système mobilisateur chargé de déplacer le micro-capteur en cas de nécessité.

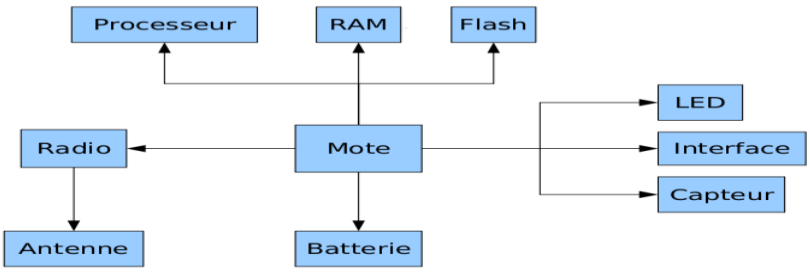

Figure 1. 2: Architecture d'un capteur sans fil.

On peut voir sur la figure 1.2 les différents composants qui constituent un capteur. Pour être plus précis chaque groupe de composants possède son propre rôle :

3.1 Unité de traitement

Mote, processeur, RAM et Flash : On appelle généralement Mote la carte physique utilisant le système d'exploitation pour fonctionner. Celle-ci a pour cœur le bloc constitué du processeur et des mémoires RAM et Flash. Cet ensemble est à la base du calcul binaire et du stockage, temporaire pour les données et définitif pour le système d'exploitation. Cette unité est chargée d'exécuter les protocoles de communications qui permettent de faire collaborer le nœud avec les autres nœuds du réseau. Elle peut aussi analyser les données captées pour alléger la tâche du nœud puits.

3.2 Unité de transmission

Radio et antenne : les équipements étudiés sont donc généralement équipés d'une radio ainsi que d'une antenne. Cette unité est responsable d'effectuer toutes les émissions et réceptions des données sur un medium sans fil. Elle peut être de type *optique* (comme dans les nœuds Smart Dust), ou de type *radiofréquence*. Les communications de type optique sont robustes vis-à-vis des interférences électriques. Néanmoins, elles présentent l'inconvénient d'exiger une ligne de vue permanente entre les entités communicantes. Par conséquent, elles ne peuvent pas établir de liaisons à travers des obstacles.

3.3 Unités de captage

LED, interface, capteur : On retrouve donc des équipements de différents types de détecteur et d'autre entrée. Le capteur est généralement composé de deux sous-unités : le *récepteur* (reconnaissant l'analyste) et le *transducteur* (convertissant le signal du récepteur en signal électrique). Le capteur est responsable de fournir des signaux analogiques, basés sur le phénomène observé, au convertisseur Analogique/Numérique. Ce dernier transforme ces signaux en un signal numérique compréhensible par l'unité de traitement.

3.4 Unités de control d'énergie

Batterie : Un micro-capteur est muni d'une ressource énergétique pour alimenter tous ses composants. Cependant, en conséquence de sa taille réduite, la ressource énergétique dont il dispose est limitée et généralement irremplaçable. Cette unité peut aussi gérer des systèmes de rechargement d'énergie à partir de l'environnement observé telles que les cellules solaires, afin d'étendre la durée de vie totale du réseau.

4. Exemple de capteur sans fil

Le modèle que nous allons présenter est le MicaZ de la marque Crossbow. D'autres modèles existent chez ce fabriquant tel que le Mica2, l'Imote2 ou le TelosB .

Figure 1. 3: Symbole de la compagnie Crossbow.

Figure 1. 4: Capteur MicaZ.

Figure 1. 5: Photographie d'un MIB520.

Le capteur MicaZ

On remarque que les capteurs sont constitués de quatre principaux groupes de composants ayant chacun leur propre rôle :

- ✓ **Processeur et mémoire :** composé du *processeur* qui effectue les traitements, de la mémoire *RAM* qui stocke les données temporaires et de la mémoire *flash* qui contient le système d'exploitation. Le MicaZ est constitue d'une mémoire flash de 512 Kbytes permettant de stocker plus de 100 000 mesures ainsi que d'une mémoire flash de 128 Kbytes

- ✓ ***Communication :*** composé d'une *antenne* et d'un *système de communication radio* an de pouvoir émettre et recevoir des signaux sans fil.

- ✓ ***Alimentation :*** composé de la *batterie* fournissant l'énergie nécessaire au fonctionnement et assurant ainsi l'autonomie du capteur.

- ***Interactions :*** composé des *interfaces*, de *système de capture*, de *LEDs* et qui permet au capteur d'interagir avec son environnement.

Le capteur illustré dans la Figure 1.4, est alimenté par deux piles AA et fonctionne sur une plage théorique allant de 2,7 à 3,3 Volts. La portée de ce capteur peut atteindre une trentaine de mètres en intérieur et s'étendre jusqu' à 100 mètres en milieu ouvert.

Ce type de communication convient pour les réseaux de type WPAN 2 dont la portée est de l'ordre de quelques dizaines de mètres. Celle-ci est gérée par un protocole de haut niveau permettant la communication entre des appareils autonomes restreint en mémoire et énergie : ZigBee.

La machine servant à traiter les données ne possédant aucun moyen de communiquer directement avec le capteur (en radio), il est alors nécessaire d'utiliser un second capteur de même nature ainsi que d'une carte d'interface.

5. Architecture des réseaux de capteurs sans fil

Un réseau de capteur sans fil (RCSF) est constitué d'un nombre plus ou moins grand de nœuds capteurs. Ces nœuds sont autonomes, distribués dans l'espace qui coopèrent pour surveiller des conditions environnementales ou physiques, telle que la *température*, le *bruit*, la *vibration*, la *pression*, le *mouvement*, etc. A l'origine, le développement des réseaux de capteur sans fil a été motivé par des applications militaires telles que la surveillance de champ de bataille. Cependant, ce type de réseau est maintenant employé dans plusieurs domaines d'application civils, comme la surveillance d'environnement, d'habitat, la surveillance médicale, l'automatisation des maisons, le contrôle du trafic [YR07].

5.1 Architecture

L'architecture des réseaux de capteurs sans fils utilise beaucoup de sources. Historiquement, beaucoup du travail relatif a été effectué dans le contexte des réseaux à auto-organisation, mobiles et Ad Hoc Un réseau de capteurs est constitué essentiellement de : plusieurs nœuds capteurs, un nœud Sink et un centre de traitement des données.

- **Nœuds :** sont des capteurs, leur type, leur architecture et leur disposition géographique dépendent de l'exigence de l'application en question. Leur énergie est souvent limitée puisqu'ils sont alimentés par des piles.

- **Sink :** c'est un nœud particulier du réseau. Il est chargé de la collecte des données issues des différents nœuds du réseau. Il doit être toujours actif puisque l'arrivée des informations est aléatoire. C'est pourquoi son énergie doit être illimitée. Dans un réseau de capteur sans fils plus ou moins large et à charge un peu élevée, on peut trouver deux sinks ou plus pour alléger la charge.

- **Centre de traitement des données :** c'est le centre vers lequel les données collectées par le sink sont envoyées. Ce centre a le rôle de regrouper les données issues des nœuds et les traiter de façon à en extraire de l'information utile exploitable. Le centre de traitement peut être éloigné du sink[YR07], alors les données doivent être transférées à travers un autre réseau, c'est pourquoi on introduit une passerelle entre le sink et le réseau de transfert pour adapter le type de données au type du canal (comme c'est illustré dans la figure 1.6).

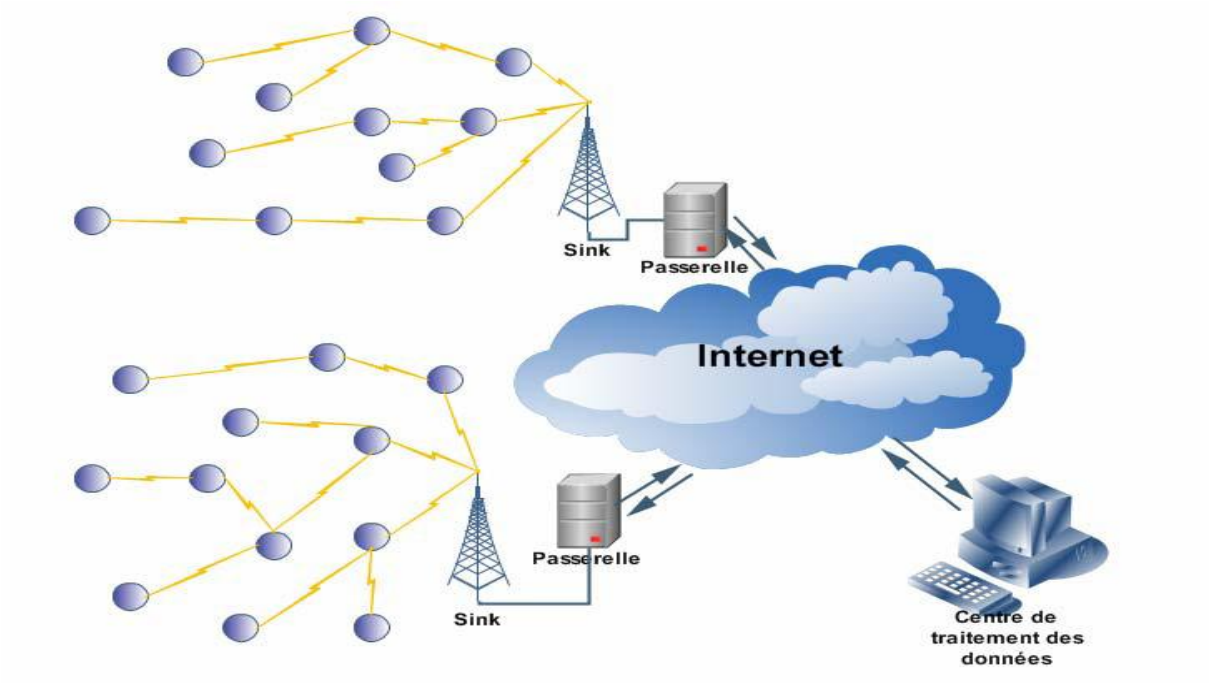

Figure 1. 6: Architecture générale d'un réseau de capteurs sans fil.

5.2 Types des nœuds

Dans un réseau de capteurs il existe deux types de nœuds : nœud source et nœud sink. Un nœud source est n'importe quelle entité dans le réseau qui peut fournir de l'information, c'est à dire un simple nœud capteur [YR07].

Un nœud sink est l'entité où les données sont récupérées. Il y a essentiellement trois types de sink :

- ✓ Un nœud appartenant au réseau comme n'importe quel autre nœud.
- ✓ Une entité extérieure au réseau. Pour ce deuxième cas, le sink peut être un dispositif extérieur, par exemple, un ordinateur portatif ou un PDA interagissant avec le réseau.
- ✓ Une passerelle vers un autre réseau tel qu'Internet, où la demande de l'information vient d'un certain centre de traitement lointain.

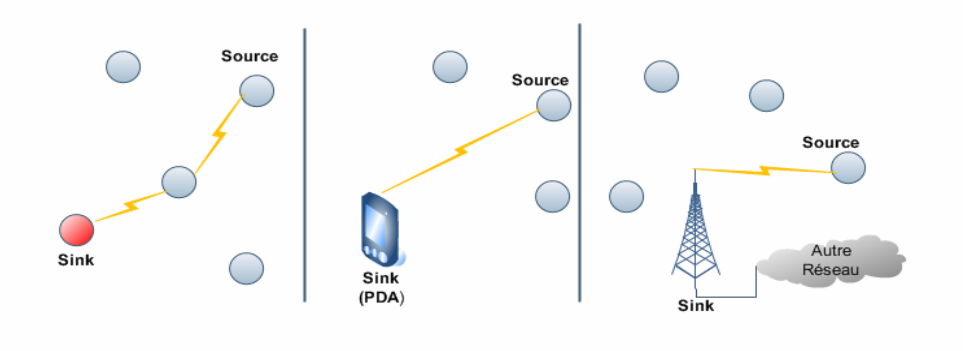

Figure 1. 7: Différents types de sink.

6. Les modèles de transmission des données dans les RCSF

La transmission de données dans les réseaux de capteurs peut se faire suivant plusieurs modèles dont on distingue trois essentiels :

6.1 Le modèle driven-event

Au lieu d'avoir un nœud émetteur et un autre récepteur de l'information, on trouve un nœud récepteur (le nœud de contrôle « *sink* ») et un groupe de nœuds capteurs, se trouvant proche de l'événement, qui sont tous des émetteurs de la même information.

L'avantage pour ce modèle, repose essentiellement sur la détection de l'événement et la rapidité des prises des réactions nécessaires pour assurer l'aspect temps réel des applications.

L'inconvénient majeur de ce modèle est la redondance des données, car les nœuds excités par le même événement envoient la même information au « *sink* » [YR07].

6.2 Le modèle query-driven

Ce modèle est semblable au modèle *driven event* sauf que la collecte des informations sur l'état de l'environnement est initiée par des interrogations envoyées par le « s*ink* », on peut utiliser ce modèle pour contrôler et reconfigurer les nœuds. Par exemple, le « *sink* » peut envoyer des commandes au lieu des interrogations pour modifier le programme d'un noeud capteur, son taux de trafic et son rôle. Seul le nœud capteur jouant le rôle de «*sink*» est autorisé d'émettre des demandes d'interrogations ou des commandes et ce pour assurer l'ordre et l'hiérarchie de réseau de capteur.

6.3 Le modèle continuous

Dans ce modèle, les nœuds capteurs envoient les informations d'une manière continue au nœud « *sink* » suivant un volume de trafic prédéterminé [BS10].

7. Topologies des réseaux de capteurs sans fils

Les topologies des réseaux de capteurs 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.

7.1 Topologie Hiérarchique

Les protocoles à topologie hiérarchique forment des réseaux dans lesquels un nœud central s*ink* (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. Aussi, 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. 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 [WZ06].

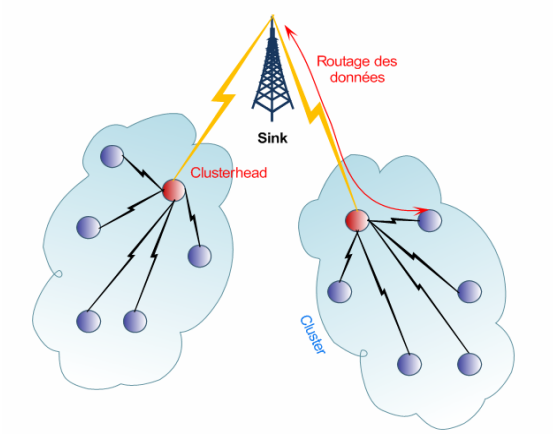

Figure 1. 8 : Topologie Hiérarchique

7.2 Topologie plate

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 s*ink*, est chargé de la collecte des données issues des différents nœuds capteurs afin de les transmettre vers les centres de traitement.

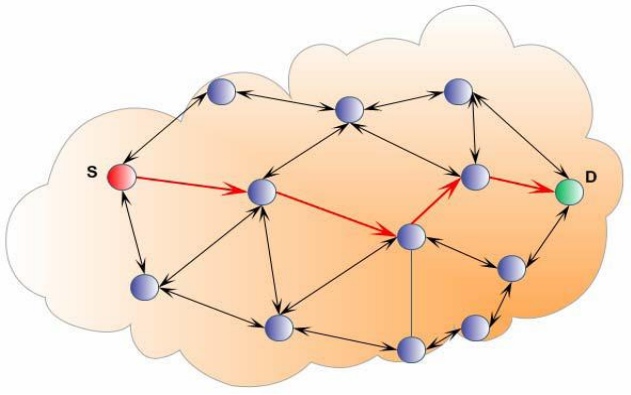

Figure 1. 9: Topologie plate (Flat).

7.3 Topologie basée Localisation

Les protocoles à topologie basée localisation suppose que :

- Le réseau est partitionné en plusieurs zones de localisation.
- Chaque zone a son identifiant.

- Chaque nœud a un identifiant EUI (*End-system Unique Identifier*) et enregistre dynamiquement l'identifiant de la zone à laquelle il appartient temporairement.

L'information temporaire de localisation appelée LDA (*Location Dependent Address*) qui est un triplet de coordonnées géographiques (longitude, latitude, altitude) obtenues, par exemple, au moyen d'un GPS avec une précision dépendant du type de l'application. Une telle topologie exige l'implémentation d'un algorithme de gestion de localisation qui permet aux nœuds de déterminer les endroits approximatifs des autres nœuds.

Ce type de topologie est mieux adapté aux réseaux avec une forte mobilité Avant d'envoyer ses données à un nœud destination, le nœud source utilise un mécanisme pour déterminer la localisation de la destination puis inclus l'identifiant de zone de localisation et du nœud destination dans l'entête du paquet à envoyer [YR07].

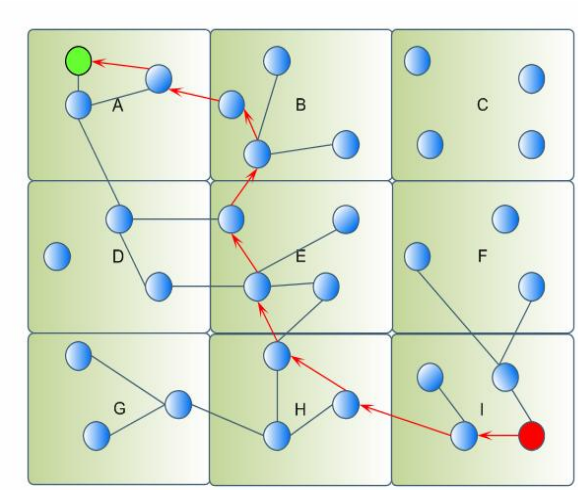

Figure 1. 10 : Topologie Basée Localisation.

8. Application des réseaux de capteurs

Il existe trois grandes classes de réseaux de capteurs : réseau de collection des données d'environnements, réseau de surveillance et sécurité et enfin les réseaux de poursuite.

8.1Réseau de collection des données d'environnements

Ces réseaux ont été conçus en générale pour la collecte périodique des données environnementales puis leurs transmissions vers une station de base (sink). Les nœuds du

réseau peuvent avoir plusieurs fonctionnalités et différents types de capteurs. Ce type de réseau nécessite généralement un flux de données faibles, une durée de vie importante et une topologie statique et par contre elle ne présente pas de contraintes de types temps de latence. Un exemple de tel réseau est **Great Duck Island Monitoring Project** dans lequel les nœuds mesurent la température, l'humidité et les radiations solaires au sein d'un champ agriculture [WZ06].

8.2 Réseau de surveillance et sécurité

Les réseaux de surveillance sont constitués des nœuds fixes et qui contrôlent d'une façon continue la détection d'une anomalie dans le fonctionnement d'un nœud. La différence entre ce réseau et le premier réseau est que les nœuds ne sont pas en train de collecter aucune donnée. Ici les nœuds transmettent seulement les rapports concernant une violation de la sécurité. Ce type de réseau nécessite un excellent temps de latence pour une bonne performance. Un exemple de tel réseau est la détection des incendies dans les forêts et spécialement le projet **FireBug** installé dans les forêts de Californie [WZ06].

8.3 Réseau de poursuite

Cet usage de réseau concerne en générale la poursuite d'un objet dans un lieu contrôlé par un réseau de capteurs. Ce type de réseau est caractérisé par une topologie mobile ce qui engendre une instabilité de la communication des noeuds. Ces réseaux sont générale développé par l'armée comme le projet américain **SmartDust** dans la poursuite du tireur d'élite [WZ06].

9. Contrainte de conception des RCSF

La conception est contrainte par plusieurs paramètres. Ces facteurs servent comme directives pour le développement des algorithmes et protocoles utilisés dans les RCSF.

9.1 Durée de vie du réseau

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

9.2 Ressources limitées

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

9.3 Bande passante limitée

Afin de minimiser l'énergie consommée lors de transfert de données entre les nœuds, les capteurs opèrent à bas débit. Typiquement, le débit utilisé est de quelques dizaines de Kb/s. Un débit de transmission réduit n'est pas handicapant pour un réseau de capteurs où les fréquences de transmission ne sont pas importantes.

9.4 Facteur d'échelle

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

9.5 Topologie dynamique

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

- Les nœuds capteurs peuvent être déployés dans des environnements hostiles (champ de batail par exemple), la défaillance d'un nœud capteur est, donc très probable.
- Un nœud capteur peut devenir non opérationnel à cause de l'expiration de son énergie.
- Dans certaines applications, les nœuds capteurs et les stations de base sont mobiles.

9.6 Agrégation de donnée

Dans les réseaux de capteurs, les données produites par les nœuds capteurs voisins sont très corrélées spatialement et temporellement. Ceci peut engendrer la réception par la station de base d'informations *redondantes*. Réduire la quantité d'informations

redondantes transmises par les capteurs permet de réduire la consommation d'énergie dans le réseau et ainsi d'améliorer sa durée de vie. L'une des techniques utilisée pour réduire la transmission d'informations redondantes est l'agrégation des données. Avec cette technique, les nœuds intermédiaires agrègent l'information reçue de plusieurs sources. Cette technique est connue aussi sous le nom de fusion de données.

10. Caractéristiques des RCSF

Un réseau de capteurs présente les caractéristiques suivantes :

- **absence d'infrastructure** : les réseaux Ad-hoc en général, et les réseaux de capteurs en particulier se distinguent des autres réseaux par la propriété d'absence d'infrastructure préexistante et de tout genre d'administration centralisée.
- **taille importante** : un réseau de capteurs peut contenir des milliers de nœuds.
- **interférences :** les liens radio ne sont pas isolés, deux transmissions simultanées sur une même fréquence, ou utilisant des fréquences proches, peuvent interférer.
- **topologie dynamique** : les capteurs peuvent être attachés à des objets mobiles qui se déplacent d'une façon libre et arbitraire rendant ainsi la topologie du réseau fréquemment changeante.
- **sécurité physique limitée** : les réseaux de capteurs sans fil sont plus touchés par le paramètre de sécurité que les réseaux filaires classiques. Cela se justifie par les contraintes et limitations physiques qui font que le contrôle des données transférées doit être minimisé.
- **bande passante limitée** : une des caractéristiques primordiales des réseaux basés sur la communication sans fil est l'utilisation d'un médium de communication partagé. Ce partage fait que la bande passante réservée à un noeud est limitée.
- **contrainte d'énergie, de stockage et de calcul** : la caractéristique la plus critique dans les réseaux de capteurs est la modestie de ses ressources énergétiques car chaque capteur du réseau possède de faibles ressources en termes d'énergie (batterie). Afin de prolonger la durée de vie du réseau, une minimisation des dépenses énergétiques est exigée chez chaque noeud. Ainsi, la capacité de stockage et la puissance de calcul sont limitées dans un capteur.

11. Protocole ZigBee / IEEE 802.15.4

ZigBee est un protocole de haut niveau qui permet de mettre en place une communication entre des petites radios, avec une consommation énergétique réduite. Ce paramètre est important car ces radios sont souvent implantées sur des objets mobiles de type capteur sans fil comme c'est le cas dans ce projet. Ces capteurs possédant une durée de vie limitée, l'optimisation des couts de communication est primordiale. ZigBee est base sur le standard IEEE 802.15.4 pour des WPAN (Wireless Personal Area Networks). Ce protocole est le centre d'une communauté industrielle : la ZigBee Alliance. Celle-ci a pour but de promouvoir l'utilisation de ce protocole [CB08].

Pour se référer a un mode de communication sans fil plus connu, le ZigBee est fortement inspire de la technologie Bluetooth, mais en étant moins onéreux et plus simple à implémenter en terme de codage. En effet, les nœuds ZigBee sont en moyenne dix fois plus petits en taille de code. Pour mieux comprendre la place de ZigBee au sein du panel de solution de communication sans fil existant, voici le graphe (figure 1.10) qui situe chaque solution selon sa portée et son débit [CB08].

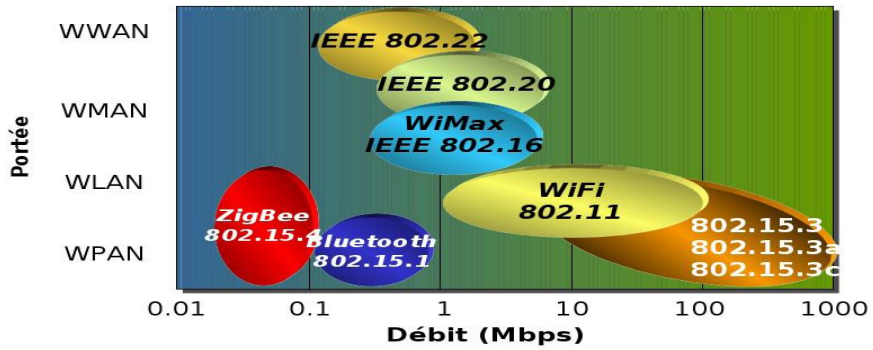

Figure 1. 11 : positionnement de ZigBee.

Comme on peut le voir dans la Figure 1.10, le protocole ZigBee possède une faible portée ainsi qu'un faible débit, mais cela engendre du même coup une faible consommation énergétique. De plus ZigBee possède une fiabilité accrue de part les propriétés du standard IEEE 802.15.4.

Pour montrer l'intérêt du ZigBee dans son aspect économique du point de vue énergétique, voici un tableau comparatif chiffre entre différents protocoles de communication sans fil.

Protocole	ZigBee	Bluetooth	WiFi
IEEE	802.15.4	802.15.1	802.11a/b/g
Ressources systèmes	4 Ko – 32 Ko	+ 1 Mo	+ 250 Ko
Durée de vie de la batterie	100 à + 1000 jours	0,5 à 5 jours	1 à 7 jours
Taille des réseaux	Illimite (2^{64})	32	7
Bande passante	20 à 250 Ko/s	> 11000 Ko/s	720 Ko/s
Portée	1 à > 100 mètres	1 à 100 mètres	1 à > 10 mètres

Tableau 1. 1: des technologies ZigBee, Bluetooth et WiFi.

Au vu de ce tableau on comprend qu'on retrouve naturellement ce protocole dans des environnements embarqués ou la consommation énergétique est primordiale et où le nombre de membres du réseau est important.

11. Conclusion

Dans ce chapitre nous avons présenté les réseaux de capteurs sans fil (RCSF), en exposant leurs architectures, leurs contraintes ainsi que leurs domaines d'applications.

Dans le chapitre suivant, nous allons présenter le principe de localisation utilisé dans des systèmes tels que GPS et Galileo, puis nous abordons la problématique de la localisation dans les réseaux de capteurs sans fil (RCSF).

Chapitre 2
La Localisation dans les réseaux de capteur sans fil

1. introduction

Depuis quelques années, la localisation est devenue de plus en plus importante dans la vie quotidienne. Dans un premier temps, l'utilisation des systèmes de navigation par satellite (GPS), permettant une localisation relativement précise en environnements extérieurs dégagés (avec une précision de l'ordre de quelques mètres), a connu une réelle expansion. Plus récemment, de nombreuses applications émergentes reposent sur la possibilité de poursuivre des dispositifs mobiles dans des environnements où les méthodes de navigation traditionnelles ne sont plus opérantes [CS09].

Avoir sa position exacte n'importe où sur la surface de la terre est rendu possible grâce aux constellations satellitaires de la géo-localisation. Il s'agit d'un ensemble de satellites placés de façon précise en orbite MEO (Medium Earth Orbit). Nous présentons dans ce qui suit le principe de localisation utilisé dans des systèmes tels que GPS et Galileo, puis nous abordons la problématique de la localisation dans les réseaux de capteurs sans fil (RCSF).

2. Les systèmes de localisation

Si autrefois les systèmes de localisation étaient exclusivement réservés aux applications militaires, ils se sont ouverts depuis les années 90 au monde civil. Les grandes puissances ont toutes cernées les impacts politiques et économiques de ces systèmes [CS09].

2.1 Les principaux systèmes de localisation

A. **Le GPS : (Global Positioning System)** est le système de localisation américain. Opérationnel depuis les années 1980, il a été développé pour fournir à l'armée américaine un système de repérage à couverture mondiale et de très grande précision. Son rôle consiste, par exemple, à guider un missile sur des centaines de kilomètres. Son nom officiel est NAVSTAR system (Navigation Satellite Timing and Ranging).

C'est à la fin de l'année 1993 que le département américain de la défense a ouvert l'accès gratuit au GPS pour les utilisateurs civils [CS09].

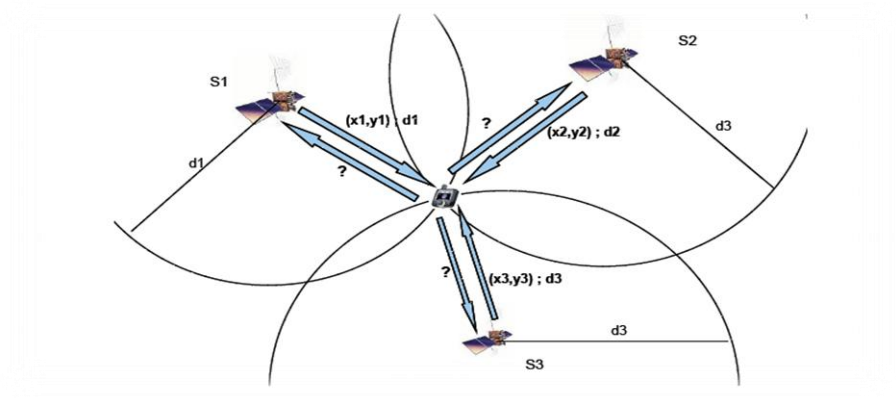

Figure 2. 1: Principe des méthodes de localisation par satellite (en 2D).

B. **Le système russe GLONASS (IAC)** : développé de 1976 à 1982, n'est plus pleinement opérationnel et ce dû aux conditions politiques et économiques du pays. Toutefois, la Russie a entrepris sa remise à niveau et GLONASS devrait être de nouveau fonctionnel en 2011.

C. **Le système IRNSS (Indian Regional Navigational Satellite System)** : De son côté, l'Inde met en œuvre son système IRNSS. Il offrira une précision au sol inférieur à 20 mètres et devrait être prêt en 2012.

D. **Le système Beidou :** développé par la Chine en est à une version expérimentale (Beidou-1). Elle est composée de quatre satellites ayant des fonctionnalités limitées. La version Compass (ou Beidou-2) devra compter 35 satellites opérationnels d'ici 2010 ou 2011.

E. **Le système Galileo** : Enfin, Galileo (CNES et ESA) est le système de localisation européen. Il est composé de 27 satellites et atteindra une précision inférieure au mètre pour les applications du domaine civil [CS09].

2.2 Principe de localisation GPS

Le principe de localisation est en lui même très simple. Des satellites dédiés gravitent autour de la terre. En effet, si on imagine vouloir localiser un point M, de la surface du globe terrestre, il suffit d'entrer en contact avec ces satellites qui lui communiquent leurs

coordonnées. Au moins trois satellites sont nécessaires pour une localisation dans la deuxième dimension, alors qu'au moins quatre le sont pour la troisième dimension.

Le point M applique alors la *multilatération* : en deux dimensions, il s'agira de définir le point d'intersection des cercles dont les centres sont les positions des satellites et les rayons sont les distances entre le module et les satellites. En trois dimensions, les cercles sont remplacés par des sphères dont le point d'intersection correspond à la position du module.

2.2.1 Positionnement simple sur le globe

Le boîtier récepteur, que possède le particulier, procède par mesures de distances; de ce point de vue, le G.P.S. travaille en régime sphérique.

- Le point M est donc sur une sphère de rayon D1 et de centre le satellite S1, l'intersection avec le globe donne un premier cercle C1.
- Le point M est aussi sur une sphère de rayon D2 et de centre le satellite S2, l'intersection avec le globe donne un deuxième cercle C2. Les cercles C1 et C2 se coupent donc en 2 points.
- Le point M est enfin sur une sphère de rayon D3 et de centre le satellite S3, l'intersection avec le globe donne un troisième cercle C3. C'est le troisième satellite "qui lève l'indétermination" et précise de manière unique le point M cherché.

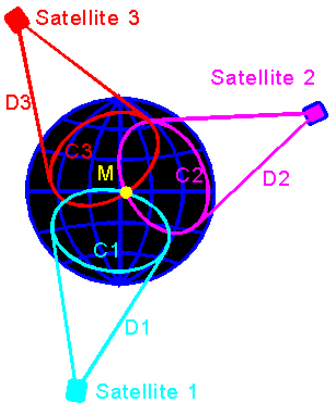

Figure 2. 2 : positionnement simple sur le globe.

2.2.2 Positionnement en altitude

Lorsque l'on veut en plus de la latitude et longitude, l'altitude, on utilise un quatrième satellite. Plus ce dernier sera proche de la verticale de M, plus l'altitude sera fiable. En pratique il arrive que M puisse "voir" 12 satellites. Un algorithme de calcul affine donc la position 3D en utilisant un maximum de satellites. D'ailleurs le récepteur GPS indique de lui même, le nombre de satellites en vue, c'est à dire utilisables.

Par exemple un satellite visible au ras de l'horizon sera inopérant pour calculer l'altitude. Réciproquement un satellite à la verticale de M donnera un mauvais positionnement horizontal. Pour des appareils évolués, le récepteur affiche le positionnement des satellites utilisés, ce qui permet d'apprécier la qualité de l'information calculée. Certains appareils indiquent même la précision de la localisation.

Cette influence de la géométrie de la constellation est caractérisée par un coefficient GDOP (Geometric Dilution Of Precision). L'UERE (User Equivalent Range Error) de standardisation GPS est de l'ordre de 16 à 23 m pouvant aller jusqu'à 400 m après 14 jours sans transfert de données.

2.2.3 La réalité et les calculs

L’horloge du récepteur est moins précise que celle du satellite et n’est jamais parfaitement synchronisée. Le calcul consiste donc à résoudre des équations dont les inconnues sont les trois coordonnées X Y Z de M, et une erreur de temps Dt inconnue mais identique pour toutes les mesures des distances approchées D1, D2, D3,....puisque tous les satellites sont parfaitement synchronisés entre eux. Ainsi le récepteur utilise les données de quatre satellites pour résoudre son problème, soit par mesures successives avec une seule voie de réception, soit par mesures simultanées avec un récepteur à plusieurs voies. Cette dernière méthode est naturellement impérative pour des engins évoluant à grande vitesse.

Le G.P.S. assure en tous points du globe un positionnement et une navigation en trois dimensions, précis à quelques dizaines de mètres près pour les utilisateurs classiques (soumis à une dégradation éventuelle aléatoire des signaux), et approchant 10 mètres dans

le plan horizontal et 15 mètres en altitude pour les usagers privilégiés ou l'armée américaine.

Si certaines méthodes de localisation s'interdisent d'utiliser ces technologies à cause de leurs erreurs de mesure, il est indispensable, pour celles qui les utilisent, de considérer ces erreurs lors du calcul des positions [CS09].

3. La localisation dans les RCSF

Dans bon nombre d'applications, un événement détecté par un capteur n'est utile que si une information relative à sa localisation géographique est fournie. C'est le cas de la surveillance des feux de forêt ou de troupes ennemies dans un contexte militaire.
Sans cette information, ces applications n'auraient aucun sens. Il s'agit donc de déterminer pour chacun des capteurs sa position. La localisation des capteurs est un des principaux problèmes dans ce type de réseaux et nombreuses sont les solutions qui ont été proposées pour le résoudre, chacune faisant des hypothèses diverses sur les capacités des capteurs.

Les méthodes de localisation étudiées dans ce mémoire ont pour but d'estimer ces positions de manière automatique. Une méthode de localisation dans les réseaux de capteurs est composée de deux parties : estimation des distances et dérivation des positions.

3.1 Estimation des distances

Dans cette phase les nœuds communiquent entre eux et collectent différents indicateurs de qualité des communications radios. Le hardware radio peut rapporter diverses informations sur le signal radio entre deux nœuds.

En effet le simple fait qu'ils communiquent entre eux nous indique qu'ils sont à portée radio l'un de l'autre. De plus le hardware radio de nos nœuds peut nous rapporter diverses caractéristiques à propos du signal radio entre les deux nœuds, à partir desquelles les distances séparant les nœuds peuvent être estimées.

2.2 Dérivation des positions

Le but de cette phase est de trouver les positions des nœuds qui respectent au mieux les distances inter-nœuds estimées. Si nous connaissons la position de quelques nœuds du

réseau dans un certain système de coordonnées, les positions des autres nœuds dans ce système de coordonnées peuvent être trouvées.

4. Caractérisations des méthodes

Il existe de nombreuses approches pour résoudre le problème de la localisation et chaque méthode a ses avantages et ses inconvénients.

4.1 Utilisation d'estimations de distances

1. **Les méthodes range-free**

Ces méthodes ne calculent jamais de distances entre voisins. Elles utilisent d'autres informations telles que la connectivité pour identifier la position des nœuds. Dans un objectif de simplicité et de réduction du coût, ces méthodes supposent que le déploiement des nœuds respecte certaines contraintes et propose des calculs plus ou moins complexes pour évaluer la position. Elles semblent donner de bons résultats dans les réseaux denses et réguliers. Dans ce mémoire nous nous intéresserons aux méthodes range-free.

2. **Les méthodes range-based**

Ces méthodes estiment les distances entre les nœuds en utilisant une mesure de la distance inter-nœud obtenue grâce au signal radio, et ensuite dérivent de ces distances les positions des nœuds.

4.2 Nécessité de connaître la position d'ancres

Si une méthode requiert l'encodage au préalable de la position d'un certain nombre d'ancres, cela signifie qu'il faudra qu'une personne intervienne avant un déploiement pour mesurer la position d'un certain nombre de nœuds. Cela est parfois difficile, voire impossible dans certaines situations.

Et donc, le fait que la méthode de localisation requiert ou non de connaître la position d'un certain nombre d'ancres est une caractéristique importante de la méthode.

1. **Les méthodes anchor-based**

Sont celles qui ne fonctionnent pas sans connaître la position d'un certain nombre d'ancres à priori.

2. Les méthodes anchor-free

Sont celles qui n'ont besoin de la position d'aucun nœud pour fonctionner ; elles créent donc une carte relative du réseau. Par relative, nous entendons une carte qui est à une translation, une rotation orthogonale, une réflexion et une dilatation près de la 'vraie' carte. Autrement dit, c'est une carte qui conserve les rapports entre les distances entre tous les points.

4.3 Forme d'implémentation

Nous distinguons plusieurs façons d'implémenter le processus de localisation :

1. Les méthodes centralisées

Tous les nœuds communiquent avec leurs voisins et renvoient à l'ordinateur central soit des informations sur le signal, soit directement les distances. L'ordinateur central s'occupe si nécessaire d'estimer les distance à partir des informations sur le signal et ensuite de localiser les nœuds [BU02, BH00] .

2. Les méthodes distribuées

Ici tous les nœuds communiquent avec leurs voisins pour estimer les distances et échangent leurs informations de voisinage. Ils dérivent ensuite de façon distribuée la position de tous les nœuds dans le réseau. C'est-à-dire qu'à la fin du processus de localisation, chaque nœud doit connaître sa position ainsi que celles de ses voisins et ce sans l'aide d'un ordinateur central qui effectuerait les calculs. Pour les grands réseaux, on considère qu'une méthode distribuée est nécessaire car les méthodes centralisées demanderaient trop de communication pour l'acheminement des informations vers l'unité centrale et consommeraient donc trop d'énergie [JA06].

5. Les technologies de signaux utilisés dans la localisation

On distingue plusieurs types de signaux utilisés dans la localisation selon le type de capteurs, l'environnement ou bien selon la technologie de mesure utilisée.

5.1 Infrarouges

On distingue plusieurs types d'utilisation de l'infrarouge pour la localisation.

Premièrement on peut placer une série d'émetteurs (typiquement des LEDs) infrarouges autour d'une zone et alors un dispositif muni de détecteurs infrarouges peut, en utilisant les caractéristiques des divers signaux infrarouges émis par les différentes sources, se localiser et trouver son orientation dans la zone couverte par les émetteurs [PL04].

On utilise également les infrarouges pour permettre à des robots de se localiser dans un environnement (chaque robot a des émetteurs et des récepteurs infrarouges placés dans plusieurs directions et peut émettre des infrarouges. L'intensité de lumière reçue en retour permet d'estimer la distance entre eux [SR].

5.2 (Ultra) sons

Il y a eu beaucoup de recherches sur la localisation à base de sons ou ultrasons. Le principal avantage de la technologie est que la vitesse de propagation du son est *assez lente*, en comparaison à celle des ondes. Cela permet de mesurer les temps de propagation précisément et ainsi d'obtenir des estimations de distances fiables.

Le système cricket [NP00], par exemple, combine des communications par radio et des émetteurs/récepteurs de sons pour se localiser. Pour calculer les distances jusqu'à ses voisins, un 'cricket' envoie simultanément une onde radio et une onde sonore ayant une certaine forme caractéristique. Les nœuds qui reçoivent l'onde radio démarrent un minuteur et attendent l'arrivée de l'onde sonore. Lorsque celle-ci arrive ils consultent le minuteur et peuvent ainsi déduire le temps de propagation du son. A partir de cela ils calculent les distances de façon assez précise et en déduisent les positions.

La seule restriction de ce système est qu'il ne peut être utilisé sur de trop longues distances et qu'il est parfois difficile à utiliser dans des environnements trop bruyants ou ayant des obstacles ou des murs [KW05].

5.3 Radio frequencies

5.3.1 GPS : Global positioning system

La technologie de localisation la plus répandue est sans doute le GPS. Il remplit très bien son rôle de système de localisation à l'échelle planétaire.

Les satellites émettent en continu des signaux radios. Ces signaux radios contiennent une description des trajectoires de chaque satellite. Les récepteurs GPS reçoivent ces

informations et connaissent ainsi les positions des satellites. Ils calculent également les temps que prends l'onde radio pour arriver de chaque satellite et en déduisent ainsi les distances jusqu'à chacun d'eux. Ils déduisent ensuite, en utilisant la multilatération, leurs positions. Nous détaillerons la trilatération au chapitre 4.

5.3.2 RFID : Radio Frequency Identification

C'est une technologie très en vogue ces derniers temps. Elle permet par exemple de localiser et identifier un objet dans un stock. Les premiers systèmes RFID furent inventés en 1939 par des anglais sous le nom d'IFF (Identification Friend or Foe) et furent utilisé durant la deuxième guerre mondiale par les pilotes alliés afin d'identifier les autres avions comme amis ou ennemis.

RFID est un terme générique désignant les technologies qui utilisent les ondes radios pour identifier des gens ou des objets. Il y a plusieurs méthodes d'identification mais en général on stocke un numéro de série - qui désigne une personne ou un objet - et parfois d'autres informations sur une puce électronique attachée à une antenne (l'antenne et la puce forment ce que l'on appelle un émetteur RFID ou tag RFID).

Cela permet à un lecteur RFID de questionner les tags et de recevoir leurs informations. On distingue les tags passifs et les tags actifs. Les tags passifs n'ont pas de source de courant et utilisent donc l'énergie des ondes émises par le lecteur RFID (devant être à proximité) pour s'activer et répondre. Les tags actifs ont une source de courant propre et peuvent donc être plus actifs et indépendants [JC05].

5.3.3 RF-UWB : Ultra Wide Band

L'Ultra Wide Band est un nouveau type de radio sans fil. Il est caractérisé par une grande largeur de bande par rapport à la fréquence centrale des ondes émises. Il y a deux facteurs derrière le concept UWB : (1) Bande passante relativement large et (2) une fréquence centrale relativement petite. La grande largeur de bande permet des résolutions temporelles précises et dans des systèmes bien architecturés une meilleure confidentialité. La basse fréquence centrale quant à elle devrait permettre un meilleur passage des ondes à travers les différents matériaux.

Divers systèmes de localisation ont été testés sur les radios UWB et semblent donner des résultats très prometteurs.

5.3.4 RF-WIFI, RF-Bluetooth

Les méthodes utilisées pour la localisation dans les réseaux WIFI et Bluetooth sont assez comparables à celles utilisées dans les réseaux de capteurs. Ces 3 types de réseaux utilisent des technologies radio assez comparables (le standard radio des RCSF est un sous-standard de Bluetooth). Nous analyserons les méthodes de localisation basées sur les radios sans fil dans les sections qui suivent.

5.4 Image

Les méthodes de localisation par la vision sont nombreuses, basées sur du traitement d'images et sont très différentes des méthodes de localisation habituelles. La difficulté consiste à extraire des informations pertinentes des pixels formant l'image.

L'extraction d'informations depuis des images forme un domaine de recherche à part entière et qui sort du cadre de ce mémoire. Divers travaux de recherche sont en cours dans ce domaine.

Technologie	**Remarque**
Infrarouges	Doit être en ligne de vue Inutilisable en lumière de soleil Précision : 5-10m
Ultrasons	Pas d'obstacle Précision : 1-10m
Radio Fréquence	Très utile GPS précision : 5-10m RFID précision : 5cm-5m Cellulaire précision : 50-100m Wifi précision : 2-50m Bluetooth, ZigBee précision :2-10m
Optique	Doit être en ligne de vue Bonne précision
UWB Radio	Expérimentale Précision :6-10m

Tableau 2. 1: Les technologies de localisation [MK05].

6. Les technologies de mesure

Plusieurs technologies permettent à un capteur de mesurer la distance qui le sépare d'un capteur voisin (ToA, TDoA, RSSI) ou bien de mesurer l'angle qu'il forme avec celui-ci (AoA).

6.1 Différence des temps d'arrivée

La technologie TDoA (Time Difference of Arrival) se base sur la différence des dates d'arrivée d'un ou plusieurs signaux et suppose également que la vitesse de propagation des signaux est connue. Cette technologie s'applique dans les cas suivants :

- un émetteur envoie des signaux de natures différentes (par exemple, l'ultrason, l'onde radio, ...) à un récepteur ;
- un récepteur reçoit des signaux d'une même nature d'au moins trois émetteurs ;
- un émetteur envoie un signal reçu par au moins trois récepteurs (dans ce dernier cas une vue globale des signaux sera connue).

Dans chacun des cas, les récepteurs mettent en corrélation leurs informations et en déduisent les distances qui les séparent des émetteurs. Il s'agit d'une simple résolution d'un système d'équations dont les distances sont les inconnues.

6.2 Temps d'arrivée

La technologie ToA (Time of Arrival) suppose que les nœuds du réseau sont synchrones. La distance qui sépare deux capteurs se déduit de la vitesse de propagation du signal et de la différence entre les dates d'émission et de réception du message. Cette technologie est celle utilisée par le système GPS.

Lorsque les nœuds ne sont pas synchrones, l'envoi d'un message aller-retour est nécessaire. En fonction de son horloge, de la vitesse de propagation du signal et du temps de traitement du signal reçu, un capteur récepteur obtient la distance qui le sépare du capteur émetteur en calculant la différence entre les dates d'émission et de réception, en y soustrayant le temps de traitement du signal, puis en divisant le résultat par deux. Cela suppose que les nœuds du réseau ont un temps de traitement du signal identique

6.3 Puissance du signal

La puissance d'émission et de réception d'un signal peut être également exploitée pour obtenir la distance entre deux capteurs. La technologie RSSI (Received Signal Strength Indicator) considère la perte de puissance d'un signal entre son émission et sa réception. Cette perte (path loss) varie en fonction de la distance entre les deux capteurs : plus les capteurs sont éloignés, plus la perte est importante. Cette perte sera alors traduite en une distance.

6.4 Angle d'arrivée

La technologie AoA (Angle of Arrival) calcule l'angle formé entre deux capteurs. Chaque capteur est doté d'antennes orientées de sorte à déduire l'angle qu'il forme avec

un voisin lorsque ce dernier lui envoie un signal. Cet angle est reporté par rapport à un axe propre au capteur. Toutefois, un capteur peut être équipé d'une boussole et, dans ce cas, l'angle sera reporté sur un des axes nord, sud, est ou ouest.

7. Algorithmes de localisation

7.1 Les algorithmes basés sur les méthodes Range-based

Les méthodes Range-based utilisent les technologies ToA, RSSI, AoA et autres afin de mesurer les distances ou les angles entre deux capteurs voisins. Grâce à cette capacité de mesure, un capteur pourra, sous certaines conditions, obtenir sa position exacte. Autrement, une position estimée lui sera attribuée. Les méthodes Range-based sont les plus répandues.

SumDistMinMax :

Dans la méthode SumDistMinMax, chaque ancre commence par diffuser sa position. Lorsqu'un capteur reçoit cette position, il estime la distance qui le sépare de cette ancre en appliquant la technique SumDist qui est la plus simple pour l'estimation de la distance entre un capteur et une ancre. Elle consiste à ajouter les distances mesurées entre chaque paire de capteurs voisins séparant l'ancre et le capteur qui cherche à estimer sa position. Lorsqu'une ancre envoie sa position, elle joint au message la distance qui la sépare d'elle même, c'est à dire 0. A la réception de ce message, chaque capteur voisin mesure la distance avec l'émetteur, l'enregistre, ajoute cette distance à celle contenue dans le message et fait suivre la position de l'ancre avec la distance mise à jour. Le même processus est répété pour tous les capteurs.

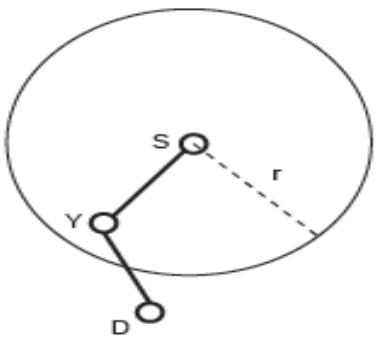

Figure 2. 3 :Algorithme SumDistMinMax

Au final, chaque capteur obtient la position de chacune des ancres et calcule une estimation de la distance qui le sépare de celle-ci. La figure 2.4 illustre le fonctionnement de SumDist. Sur cette figure, la distance estimée entre les noeuds S et D est égale à

$d_{SY} + d_{YD}$ sachant que de par l'inégalité triangulaire $d_{SD} \leq d_{SY} + d_{YD}$.

Soit x1, x2, ..., xq, a étant un chemin d'un capteur x1 vers une ancre a, l'estimation de la distance entre x1 et a (notée dx_1a) est définie de façon récursive comme suit :

$$dx_1a = dx_1x_2 + dx_2a$$

Après cette phase d'estimation des distances avec les ancres, les capteurs calculent leurs positions estimées en utilisant la méthode MinMax. Le principe de cette méthode est de déterminer, pour chaque capteur, une « boîte » le contenant dont le centre de gravité correspond à sa position estimée.

- **DV-Distance :**

Cette méthode est similaire à la précédente sauf que la distance entre voisins est calculée en se référant à la puissance du signal reçu et est exprimée en mètres et non en nombre de sauts. Dans ce cas, l'algorithme à vecteur de distance utilise comme métrique la distance cumulative exprimée en unité de longueur

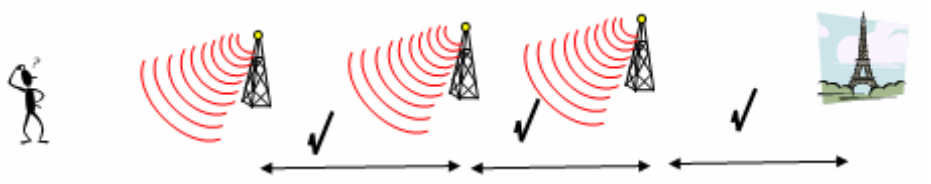

Figure 2. 4:Algorithme DV-Distance

Cette méthode est plus concrète que la précédente car elle ne considère pas que tous les nœuds sont équidistants les uns des autres. Elle est, par contre, sensible à la propagation d'erreur de mesure.

- **DV-Euclidean :**

Cette troisième méthode repose sur la propagation de la distance euclidienne séparant le nœud de l'ancre.

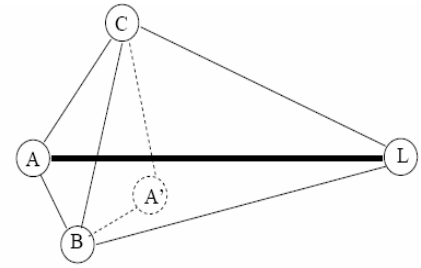

Figure 2. 5 :Algorithme DV-Euclidean

Soit un nœud A du réseau, pour déterminer sa position, il doit recevoir la position d'au moins deux de ses voisins : B et C. (figure 2.5). A doit également estimer les distances AB et AC par mesure de puissance du signal reçu, et BC qui peut être déduite à partir de la reconstitution du voisinage de A. Si on considère le quadrilatère ABCL, où les cotés sont tous connus, la diagonale BC l'est aussi, on peut donc calculer AL qui représente la distance euclidienne de A au ancre L. Cette méthode de propagation a l'avantage d'utiliser les mesures réelles, coordonnées GPS par exemple, ce qui évitera la propagation d'erreur de mesure.

7.2 Les algorithmes basés sur les méthodes Range-free

Il existe plusieurs algorithmes appartenant aux méthodes Range-free nous citons.

DV-HOP

C'est le schéma le plus basique, il utilise un échange de vecteur de distance afin que tous les nœuds du réseau parviennent à calculer la distance les séparant du ancre. Chaque ancre maintient une table {Xi, Yi, hi} où (Xi, Yi) sont les coordonnées des autres ancres du réseau et hi est le nombre de sauts séparant ce dernier du nœud en question.

Chaque ancre calcule la distance le séparant des autres ancres dans le réseau, en utilisant les informations de localisation obtenues à partir d'un système de positionnement, il en déduit une approximation de la distance par saut. C'est la distance par saut qui va constituer l'information de correction pour tout le réseau.

Chaque nœud ancre calcule : $hopSize = \frac{\sum \sqrt{(Xi-Xj)^2-(Yi-Yj)^2}}{\text{Hij}}$ $\quad i \neq j$

Avec (Xi, Yi) représentent les coordonnés d'un nœud, (Xj, Yj) les coordonnés d'un ancre et Hij le nombre de saut.

Illustration du DV-Hop

L'exemple suivant illustre le calcul de DV-Hop :

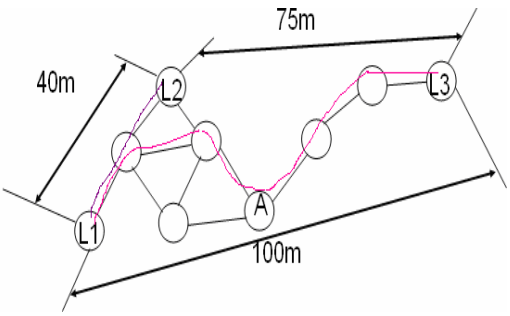

Figure 2. 6: algorithme DV-HOP

Soit dans la figure 2.6, les nœuds L1, L2 et L3 représentent des ancres. Et A le nœud voulant calculer sa position. Chaque ancre calcule la correction (hop size) et le diffuse.

$L1 \rightarrow \frac{(100+40)}{(6+2)} = 17.50$

De même pour L2 et L3 :

$L1 \rightarrow \frac{(75+40)}{(5+2)} = 16.42$

$L1 \rightarrow \frac{(100+75)}{(6+5)} = 15.90$

Un nœud quelconque du réseau obtient une mise à jour de l'ancre le plus proche. La diffusion des corrections à travers le réseau est contrôlée : quand un nœud reçoit ou émet une mise à jour, il doit tout d'abord supprimer les anciennes. Lorsque le réseau est étendu, un champ TTL (Time To Leave) est utilisé pour localiser les mises à jour au voisinage de l'ancre.

Dans l'exemple précédent, le nœud A peut estimer la distance le séparant à L1, L2, et L3 à travers la mise à jour de correction reçue de L2 ainsi A calcule la distance qui le sépare des ancres L1, L2 et L3:

$A \rightarrow L1 = 16.42 \times 3$

$A \rightarrow L2 = 16.42 \times 2$

$A \rightarrow L3 = 16.42 \times 3$

A partir de ces trois valeurs, le nœud A peut déterminer sa position relative en appliquant le même principe qu'en GPS. Pour obtenir leurs positions, les capteurs utilisent ensuite la multilatération (expliquée après dans chapitre 4). En reprenant l'exemple de la figure 2.6, X obtiendra sa position en résolvant le système suivant :

$$\begin{cases} d^2_{L1L2}=(x_A - x_{L1})^2 + (y_A - y_{L1})^2 \\ d^2_{L2L3}=(x_A - x_{L2})^2 + (y_A - y_{L2})^2 \\ d^2_{L1L3}=(x_A - x_{L3})^2 + (y_A - y_{L3})^2 \end{cases}$$

L'avantage du DV-Hop est sa simplicité. L'inconvénient est qu'il n'est adapté qu'au réseau isotrope où les propriétés du graphe sont indépendantes de la direction.

Notre étude s'intéresse particulièrement et de près à l'algorithme DV-Hop.

8. conclusion

Dans ce chapitre nous avons abordé la problématique de la localisation plus particulièrement dans les réseaux de capteurs sans fil. Nous avons également présenté les différentes technologies utilisées ainsi que quelques algorithmes existants tels que APS, DV-Hop et SumDistMinMax. Il est vrai que l'utilisation de telle ou telle technologie ou méthode est fortement liée à l'environnement du réseau : obstacles, degré d'humidité, vitesse du vent, par exemple, les technologies RSSI ou AoA sont fortement déconseillées dans des milieux fermés (In door). Des études telles que Venkatraman et al. [CS09] ont analysé l'impact de ces paramètres sur les mesures de distances et d'angles. Enfin, les auteurs K et al. [CS09] ont montré que les mesures de distances étaient moins perturbées que celles des angles.

Dans le chapitre suivant, nous allons présenter le system exploitation TinyOs un system open source embarqué pour les RCSF afin de pouvoir comprendre son fonctionnement et présenter le langage NESC pour l'implémentation des applications.

Chapitre III

Description de l'architecture de la plateforme TinyOS : un système d'exploitation pour les réseaux de capteurs

1. Introduction

Suite aux différents problèmes vécus par les réseaux de capteurs (problème énergétiques et de mémoire), 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, comme les applications tournant dessus, a été écrit en *NesC.* Ce langage a été inventé pour répondre aux attentes des systèmes embarqués. Il possède une syntaxe proche de C, supporte le système multitâche de TinyOS et défini des mécanismes pour architecturer et "linker" des composants logiciels en un système embarqué robuste [WZ06].

Dans ce chapitre, nous introduirons le mode de fonctionnement de la plateforme TinyOS, ainsi que le langage NesC et Cette description va nous permettre par la suite (dans le chapitre 4) d'implémenter un algorithme de localisation dans les RCSF.

2. TinyOS: Tiny Micro threading Operating System

2.1 Présentation

TinyOS est un système d'exploitation Open Source pour les réseaux des capteurs, conçu par l'université américaine de BERKELEY. Le caractère open source permet à ce système d'être régulièrement enrichie par une multitude d'utilisateurs. Sa conception a été entièrement réalisée en NesC, langage orienté composant syntaxiquement proche du C. Il respecte une architecture basée sur une association de composants, 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 capteurs, 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 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, du taux d'humidité…).

TinyOS s'appuie sur un fonctionnement évènementiel, c'est à dire qu'il ne devient actif qu'à l'apparition de certains évènements, par exemple l'arrivée d'un message radio. Le reste du temps, le capteur se trouve en *état de veille*, garantissant une durée de vie maximale connaissant les faibles ressources énergétiques des capteurs. Ce type de fonctionnement permet une meilleure adaptation à la nature aléatoire de la communication sans fil entre capteurs [FA08].

Figure 3. 1: Symbole de system TinyOS

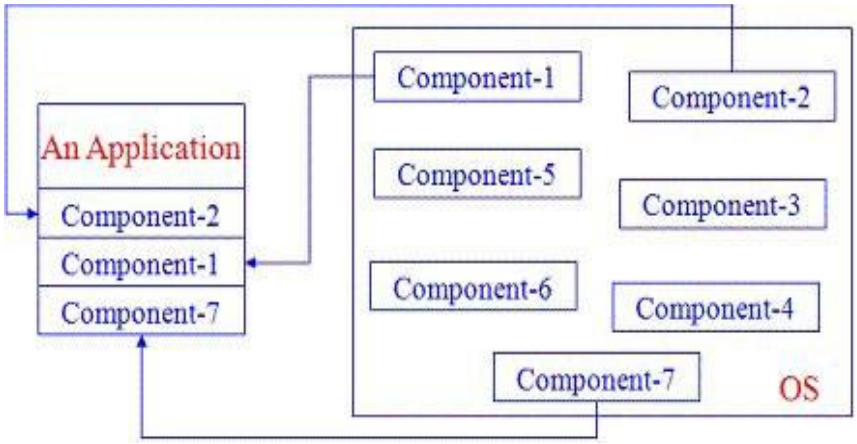

Figure 3. 2: TinyOS : un ensemble de composants logiciels

2.2 Propriétés de la plateforme TinyOS :

TinyOS a été conçu pour la programmation des réseaux de capteurs et il est caractérisé par :

- **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.

- **Event-driven** : Le fonctionnement d'un système basé sur TinyOS 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 capteur s'effectue à l'apparition d'évènements, ceux-ci ayant la plus forte priorité. Ce fonctionnement basé sur les évènements (Event-driven) s'oppose au fonctionnement dit temporel (time-driven) où les actions du système sont gérées par une horloge donnée .

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

- **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 mais donne la priorité aux interruptions matérielles. Ainsi, les tâches entre-elles ne s'interrompent pas mais une interruption (sous forme d'un évènement) peut stopper l'exécution d'une tâche .

- **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.

- **Consommation** : TinyOS a été conçu pour réduire au maximum la consommation en énergie d'un nœud capteur. Ainsi, lorsqu'aucune tâche n'est active, il se met automatiquement en mode veille [CB08]. Le Tableau 3.1 ci-dessous résume ses propriétés :

Propriété	Valeur pour TinyOS
Type	Event-driven
Disponibilité	Open source
Langage	NesC
Préemptif	Non
Temps réel	Non
Source	Fournies

Tableau 3. 1: Propriété de TinyOS

2.3 Allocation de la mémoire

Il est très important d'aborder la façon avec laquelle un système d'exploitation gère la mémoire et plus spécialement quand celui-ci travaille dans un espace restreint. TinyOS ne nécessite pas beaucoup de place mémoire 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 répartissent entre les différents besoins suivant :

1. **La Pile :** Elle sert de mémoire temporaire pour l'empilement et le dépilement des variables locales.
2. **Les variables globales :** Elles réservent un espace mémoire pour stocker des valeurs pouvant être accessibles depuis différentes tâches.
3. **La mémoire libre :** Pour tout le reste du stockage temporaire. La notion d'allocation dynamique de mémoire n'est pas présente dans le système, ce qui simplifie l'implémentation mais, par ailleurs, il n'existe pas de mécanisme de protection de la mémoire, ce qui rend le système plus vulnérable au crash et aux corruptions de mémoire [HS08].

2.4 Modèle d'exécution de TinyOS

Pour maintenir une grande efficacité requise par les réseaux de capteurs, TinyOS utilise une programmation par évènement. Ce modèle permet un très haut niveau de concurrence pour un espace très réduit de mémoire [WZ06].

L'implémentation des composants de TinyOS s'effectue en déclarant des tâches, des commandes ou des évènements. Nous allons détailler ceux-ci dans le Tableau 3.2 suivant :

Action	Utilisation
Tâche	Travaux de « longue durée »
Commande	Exécution d'une fonctionnalité précise dans un autre composant
Evènement	Equivalent logiciel à une interruption matérielle

Tableau 3. 2 : différents actions dans TinyOS

2.4.1 Programmation par évènement

Dans un système basé sur la programmation par évènement, chaque exécution est partagée entre les différentes taches de traitement. Dans TinyOS, chaque module est désigné pour fonctionner en attendant continuellement de répondre aux évènements inattendus. Quand un évènement 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 évènement 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 évènements et l'utilisation du CPU. Dans TinyOS, les taches associées avec un évènement sont exécutées très rapidement après leurs signalisations. Une fois terminé, le CPU entre en veille en attendant une nouvelle réception d'évènement [WZ06].

2.4.2 Les Tâches

Un des facteurs limitant la programmation par évènement est la longue exécution des tâches qui peut interrompre d'autres programmes importants. Si l'exécution d'un évènement 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 évènements. Les tâches sont utilisées pour effectuer la plupart des blocs d'instruction 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 exécuté que lorsque il n'y a plus d'évènements. En plus les tâches peuvent être interrompues à tout moment par des évènements. 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) [WZ06].

2.4.2 L'ordonnanceur TinyOS

La gestion des tâches et des évènements est vitale pour TinyOS. Le choix d'un ordonnanceur déterminera le fonctionnement global du système et le dotera de propriétés précises telles que la capacité à fonctionner en temps réel.
L'ordonnanceur TinyOS c'est :

- **Deux niveaux de priorité** (bas pour les tâches, haut pour les évènements) ;
- **Un file d'attente FIFO** (disposant d'une capacité de sept).

Par ailleurs, entre les tâches, un niveau de priorité est défini permettant de classer les tâches, tout en respectant la priorité des interruptions (ou évènements). Lors de l'arrivée d'une nouvelle tache, celle-ci sera placée dans la file d'attente en fonction de sa priorité (plus elle est grande, plus le placement est proche de la sortie). Dans le cas ou la file d'attente est pleine, la tâche dont la priorité est la plus faible est enlevée de la FIFO [WZ06].

2.5 Spécificités des cibles possibles de TinyOs

Les cibles de TinyOs sont essentiellement des cibles embarquées telles que les capteurs Micaz (Figure 1.4) que nous utilisons pour notre projet. Toutefois, toutes ces cibles possèdent une architecture commune basée sur un noyau central autour duquel s'articulent les différentes interfaces d'entrées/sorties, de communication et d'alimentation. L'architecture a déjà été décrite dans le chapitre 1 (Figure 1.2).

1. **Le capteur :** On appelle généralement capteur (mote en anglais) la carte physique intégrant TinyOS pour son fonctionnement. Le cœur du capteur est constitue du bloc processeur et des mémoires RAM (Random Access Memory) et Flash. Ce cœur est à la base du calcul binaire et du stockage, ce dernier pouvant être temporaire pour les données et définitif pour le système d'exploitation TinyOS [HS08].
2. **La radio :** TinyOs étant prévu pour mettre en place des réseaux sans fils, les équipements l'utilisant sont donc souvent munis d'une radio ainsi que d'une antenne pour pouvoir se connecter à la couche physique que constituent les émissions hertziennes.

3. **Les LEDs, capteurs et interfaces :** Pour avoir tout de même une utilité, les capteurs sont équipes de différents détecteurs (température, luminosité, ...) et de 3 LEDS pour pouvoir vérifier la bonne communication.
4. **La batterie :** TinyOs s'utilisant des systèmes embarqués, il gère donc une alimentation autonome grâce à une politique de basse consommation en énergie.

3. Description du langage NesC

Comme nous l'avons déjà mentionné, le développement dans TinyOS s'effectue à travers NesC. Le langage NesC est une extension du langage de programmation C qui est été désigné pour faciliter l'implémentation des structures et des composants TinyOS.

Figure 3. 3:symbole de langage NesC.

3.1 Présentation

Le langage NesC (network Embedded system C) est un dialecte de 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, les concurrences provoquant les tâches et les événements ainsi que la capacité de réagir par rapport 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'un fils peuvent simultanément accéder à la même section de mémoire (concurrence d'accès mémoire entre threads), et quand au moins l'un des accès est un "write". NesC simplifie aussi le développement d'applications et réduit la taille du code [CB08].

3.2 Les Principales caractéristiques de NesC

NesC est constitué d'*interfaces* et de *composants*. Une interface peut être utilisée ou peut être fournie. Les composants sont des *modules* ou des *configurations*. Une application est représentée comme un ensemble de composants, regroupés et rattachés entre eux (figure 3.4). Les interfaces sont utilisées pour les opérations qui d'écrivent l'interaction bidirectionnelle. Le fournisseur de l'interface doit mettre en application des commandes, alors que l'usager de l'interface doit mettre en application des événements.

Deux types de composants existent : Les modules, qui mettent en application des spécifications d'un composant. Les configurations, qui se chargeront d'unir différents composants en fonction des interfaces (commandes ou événements). La figure 3.5 montre un diagramme de blocs dans lequel est d'écrit le processus de compilation pour une application TinyOS écrite en NesC [HS08].

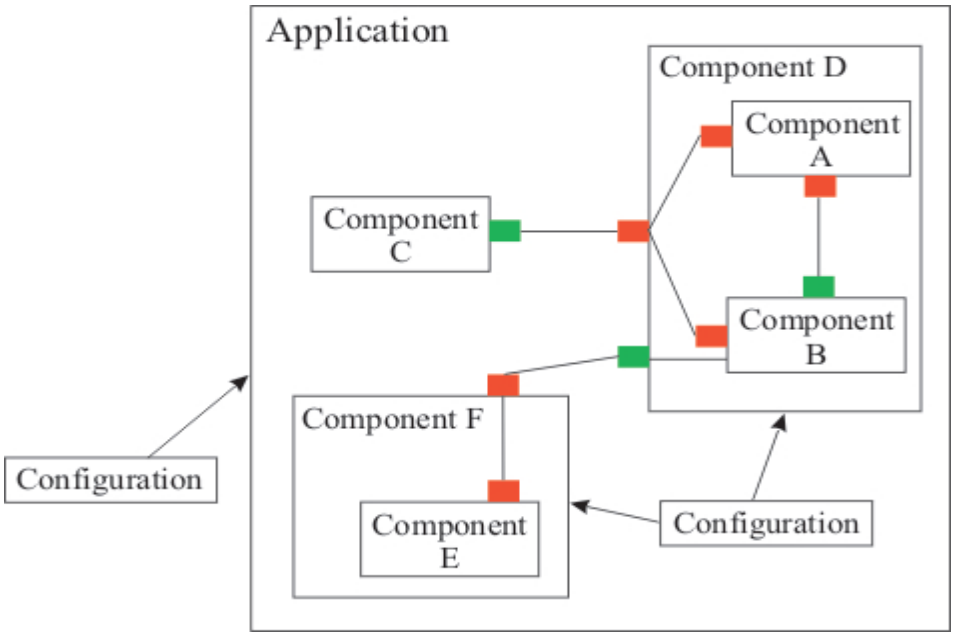

Figure 3. 4: Architecture d'une application NesC.

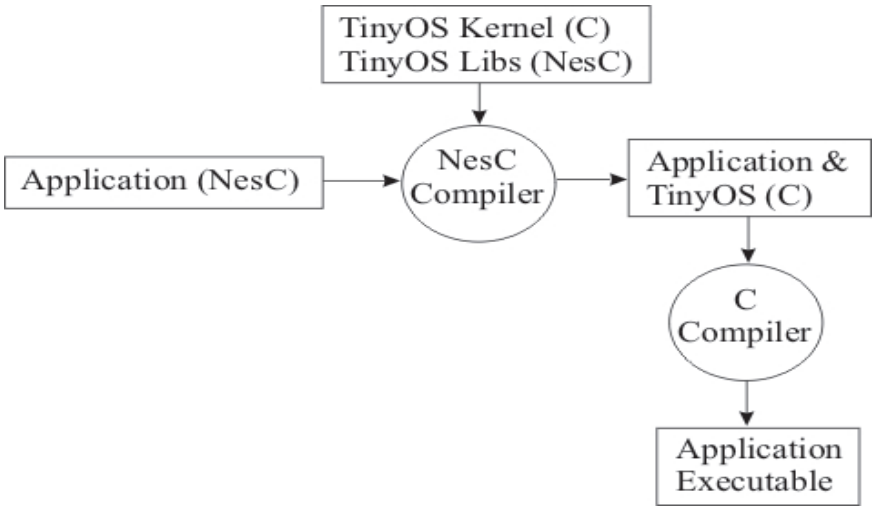

Figure 3. 5: Processus de compilation

3.3 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 : BlinkM.nc)
3. **Configuration :** Dans ces fichiers on déclare la manière d'unir les différents composants et comment effectuer le contrôle des flux. (Exemple : Blink.nc).

3.4 Concepts principaux dans NesC

1. Composants

TinyOS définit un nombre important de concepts qui sont exprimés dans NesC. D'abord, les applications NesC sont construites par des composants avec des interfaces bidirectionnelles définies. Aussi, NesC définit un modèle basé sur les tâches et les captures d'événements matériels, et détecte des éclatements d'information pendant la compilation. Un composant, du point de vue de la programmation, est composé de

plusieurs sections et l'ensemble de toutes ces sections donne lieu à la création de ce composant [WZ06].

2. Implémentations

Cette section définit les connections entre les différents composants qu'utilise l'application. Dans cette section « implémentation » sont donc principalement définis quels sont les composants qui fournissent les interfaces à notre application (ils seront généralement des composants primitifs). Généralement nous devons utiliser les interfaces que nous fournissent d'autres composants primitifs ou non primitifs, et en définitive pour chacune de ces interfaces, que nous utiliserons dans la création de notre composant, on doit obligatoirement définir des relations avec les composants qui fournissent ces interfaces. Le processus définissant ces relations s'appelle « wiring » [HS08].

Exemple :

```
1 implementation {
2 components Main, MonAppliM ;//sont des composant
3 Main . StdControl -> MonAppliM. StdControl ;
//relier le composant MonappliM avec la composant Main pour booter
4 }
```

3. 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 :

```
1 Configuration MonAppli {
2 implementation {
Components <mettre les composant nécessaire>
.....
5 }
6 }
```

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 voir réalisé par l'application. Cette partie l à 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). 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.

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

La sous-section « implementation », 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 que vont utiliser notre application

– Les fonctions qu'elle doit mettre en œuvre pour les interfaces qui nous fournissons

– Les événements qu'elle doit mettre en œuvre venant des interfaces que nous utilisons.

3.5 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 n'apportent pas de puissance de calcul mais 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 [WZ06].

3.6 Types de fonctions en NesC

En NesC les fonctions peuvent être de types très variés [HS08].Il y a d'abord les fonctions classiques avec la même sémantique qu'en C et la façon de les invoquer est aussi la même qu'en C. Il y a aussi 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 exécutes immédiatement. La manière d'appeler ce genre de fonction est :

```
1 call interface . nomFonction
```

Les fonctions « task » sont des fonctions qui sont exécutées dans l'application, utilisant la même philosophie que les fils ou « threads », c'est principalement une fonction normale qui est invoquée de la maniéré suivante :

```
1 post interface . 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 :

```
1 signal interface . nomEvenement
```

3.7 Différences entre NesC, C, C++ et Java

Les langages C, C++ et Java ont une composition dynamique et ont un espace de noms (namespace) global. Appeler une fonction requiert d'utiliser un nom qui est unique pour cette fonction dans l'espace de noms. NesC a un autre objectif. D'abord, le code est divisé en composants, unités discrètes de fonctionnalités. Un composant peut seulement faire référence à des variables de son propre espace de noms. D'une certaine manière, les composants NesC sont semblables à des objets. La principale différence est la portée des appels par référence. Tandis que les objets C++ et Java font référence à des fonctions et des variables au niveau global, les composants NesC utilisent uniquement un niveau local. Ceci signifie que, non seulement il faut déclarer les fonctions mises en œuvre, mais que de plus un composant doit déclarer les fonctions qui appellent ces fonctions [HS08].

4. Conclusion

Dans ce chapitre, nous avons étudié l'architecture du système TinyOS ; nous avons examiné son mode d'exécution et son mode d'ordonnancement des différents types d'action qui sont évènement, commande et tâche. Nous avons également présenté le langage NesC 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.

Dans le chapitre suivant, nous allons implémenter, analyser un algorithme de localisation nommé DV-Hop pour les RCSF dans la plate forme TinyOS en utilisant le langage NesC.

Chapitre 4

Implémentation et Evaluation de l'Algorithme de Localisation DV-Hop

1. Introduction

Avant sa mise en place, le déploiement d'un réseau de capteurs nécessite une phase de simulation, afin de s'assurer du bon fonctionnement de tous les dispositifs. En effet, pour de grands réseaux le nombre de capteurs peut atteindre plusieurs milliers et donc un coût financier relativement important. Il faut donc réduire au maximum les erreurs de conception possibles en procédant à une phase de validation.

Dans ce chapitre, nous allons en premier lieu, présenté la plate-forme logicielle que nous avons utilisée pour les simulations TOSSIM, ensuite, nous allons présenté les contextes de simulation et les résultats pour le protocole de localisation étudié avec discussion de résultat.

2. Simulateur de réseaux de capteur

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

2.1 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.

2.2Description

TinyViz est une application graphique qui donne un aperçu de notre réseau de capteurs à tout instant, ainsi que des divers messages qu'ils émettent. Il permet de déterminer un délai entre chaque itération des capteurs afin de permettre une analyse pas à pas du bon déroulement des actions [FA08] .

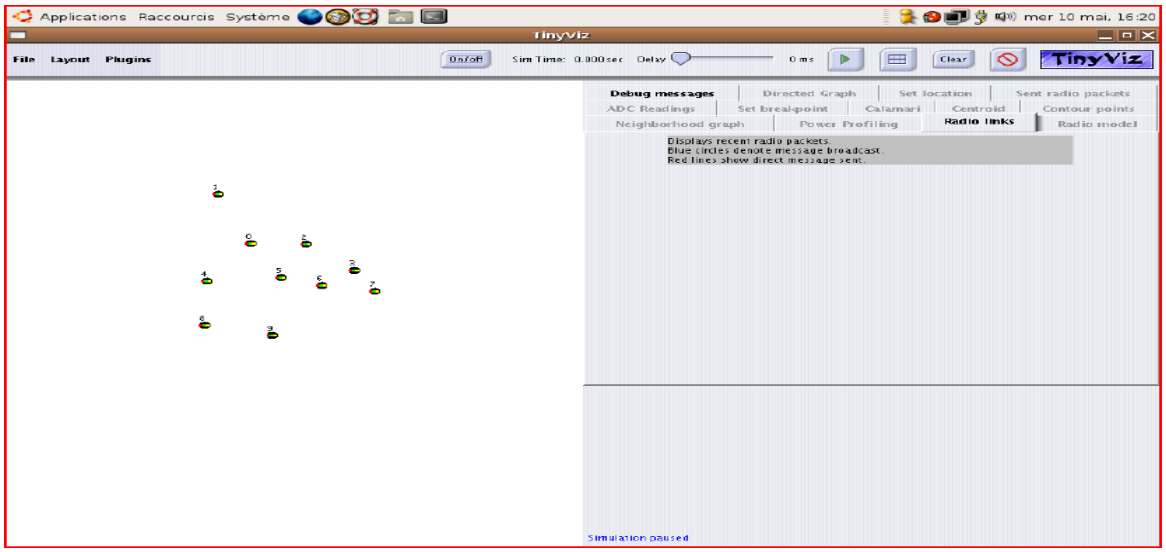

Figure 4.1 : Fenêtre graphique TinyViz

On peut voir dans la partie de gauche de la figure 4.1 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 : il 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 4.2 : Visualisation des messages radio

Dans l'exemple de la figure 4.2, 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.

2.3 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 [FA08]. 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.

2.4 Limites

Malgré nos recherches, nous n'avons pas trouvé comment deux capteurs peuvent exécuter deux applications différentes. En effet, il serait plus intéressant de distinguer les capteurs maîtres des capteurs esclave, en leur demandant d'exécuter des codes différents. Pour pallier ce problème, nous avons eu recours à une astuce. Chaque capteur est identifiable par *un numéro unique* (dans la plateforme) nous arrivons donc à différencier le rôle de chaque capteur par son numéro [FA08].

3. Objectif et simulation

Le but général dans ce chapitre est d'implémenté et d'analyser DVHOP (détaille dans chapitre 2) un algorithme de localisation sur les réseaux de capteurs sans fil en particulier:

- Evaluer la précision de la localisation.

Ces métriques sont évaluées en fonction de la taille du réseau (changement du nombre de nœud dans le réseau).

3.1 Scénario

DVHOP un algorithme de localisation. Son but est de permettre aux capteurs de trouver leur position à l'aide des positions connues de seulement quelques capteurs spécifiques appelés (ancres).

En plus, la localisation par moyens propres est donc indispensable. Elle se fait en deux étapes : premièrement l'estimation de la distance aux autres nœuds, et ensuite la multilatiration.

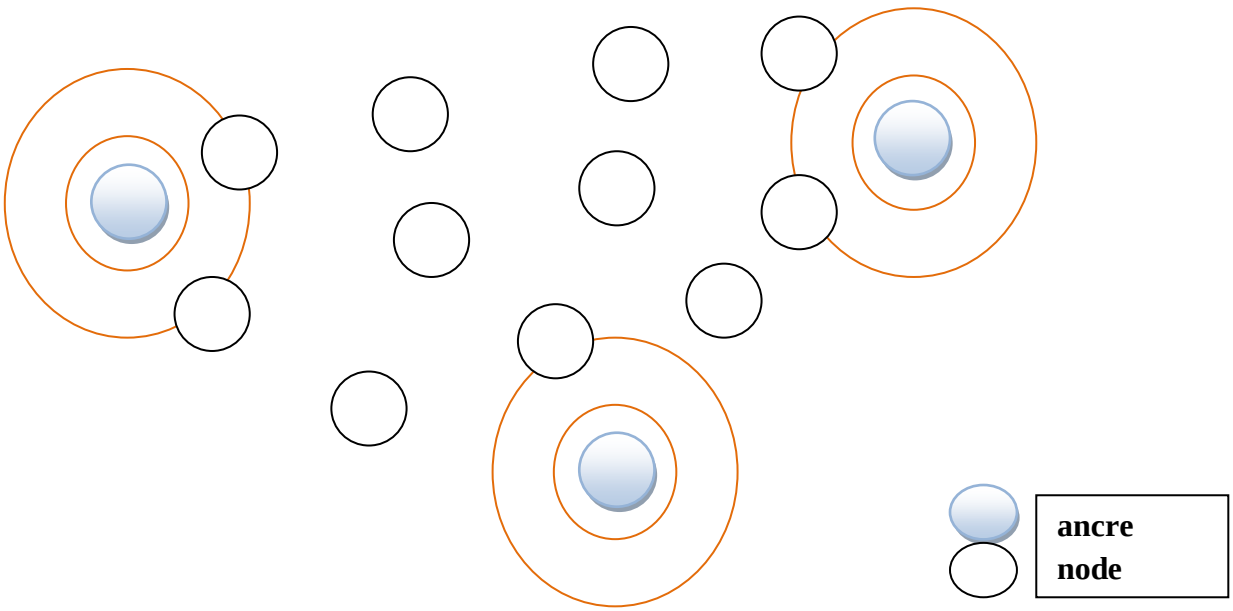

Figure 4.3 :Figure representant la modèle reel.

Figure 4.3 illustre un modèle de réseaux des capteurs sans fil, les boule bleu représentent les ancres et les blanches représentent les nœuds aveugles.

3.1.1Estimation de la distance

L'estimation de distance peut se faire sur base de différents indicateurs :

- Le temps de propagation d'une onde.

- La puissance du signal à la réception.
- Le taux d'erreurs corrigées lors des transmissions.
- Le nombre de saut

De manière générale : Si Cij est la valeur de l'indicateur entre les nœuds i et j et si la fonction f nous permet de déduire une estimation de la distance entre i et j (que nous notons dij), nous avons donc :

$$Dij = f(Cij).$$

3.1.2Multilatération

La Multilatération est une méthode relativement simple et intuitive.

Théorie :

Le fonctionnement de la multilatération va être décrit en détail. Une description détaillée de cet algorithme peut être trouvée dans. La multilatération nous permet de retrouver la position d'un nœud à partir des positions d'un certain nombre d'ancres et des distances entre ces ancres et le nœud à localiser [HM07].

Soit une cible a dont on veut trouver la position X_a, et soit m ancres i dont nous connaissons les positions $Xi \in R, 1 < i < m$. Nous supposons que nous connaissons aussi une estimation des distances $Dia, 1 < i < m$ entre chaque ancre i et le noeud a.

Nous pouvons alors poser :

$$\begin{cases} (x_{11} - x_{a1})^2 + (x_{12} - x_{a2})^2 + \ldots + (x_{1p} - x_{ap})^2 = d^2_{1a} \\ \ldots \\ (x_{m1} - x_{a1})^2 + (x_{m2} - x_{a2})^2 + \ldots + (x_{1p} - x_{ap})^2 = d^2_{ma} \end{cases}$$

Le système peut être linéaire en soustrayant la dernière équation des m – 1 équations précédentes.

$$\begin{cases} x^2_{11} - x^2_{m1} - 2(x_{11} - x_{m1})x_{a1} \\ +x^2_{12} - x^2_{m2} - {}^2(x_{12} - x_{m2})x_{a2} \\ +\ldots \\ +x^2_{1p} - x^2_{mp} - 2(x_{1p} - x_{mp})x_{ap} = d^2_{1a} - d^2_{ma} \\ \ldots \\ \ldots \\ x^2(m-1)1 - x^2_{m1} - 2(x(m-1)1 - x_{m1})x_{a1} \\ +x^2(m-1)2 - x^2_{m2} - 2(x(m-1)2 - x_{m2})x_{a2} \\ +\ldots+x^2(m-1)p - x^2_{mp} - 2(x(m-1)p - x_{mp})x_{ap} = d^2(m-1)_a - d^2_{ma} \end{cases}$$

En réordonnant les termes, nous obtenons un système d'équations linéaires de la forme Ax = b où :

$$A = \begin{pmatrix} 2(x_{11} - x_{m1}) \ldots 2(x_{1p} - x_{mp}) \\ \ldots \\ \ldots \\ 2(x_{(m-1)1} - x_{m1}) \ldots 2(x_{(m-1)p} - x_{mp}) \end{pmatrix}$$

$$B = \begin{pmatrix} x_{211} - x_{2m1} + \ldots + x_{21p} - x_{2mp} + d^2_{ma} - d^2_{1a} \\ \ldots \\ \ldots \\ x_{2(m-1)1} - x_{2m1} + \ldots + x_{2(m-1)p} - x_{2m}p + d^2_{ma} - d^2_{(m-1)a} \end{pmatrix}$$

Comme nous avons des erreurs dans les estimations de distances, nous ne pouvons pas trouver de solution exacte à ce système d'équationsEt donc :

$$\mathbf{X_a = = (A^T A)^{-1} A^T b.}$$

Où xa = $\begin{pmatrix} xa1 \\ xa2 \\ \\ \\ xap \end{pmatrix}$ C'est-à-dire notre estimation de la position du nœud **a**.

Le processus peut être recommencé avec tous les nœuds inconnus du réseau. Nous obtenons ainsi les positions de tous les nœuds dans le réseau. Cette méthode sera implémentée par matlab.

3.2 Métrique d'évaluation

Dans notre simulation, nous nous intéresserons essentiellement à la précision de localisation de l'algorithme puisqu'elle constitue le paramètre le plus critique dans la détermination de l'importance d'un réseau de capteur

La métrique précision de localisation

Cette métrique représente la précision de localisation de l'algorithme lors d'estimation des distances entre les nœuds et des positions. Nous supposons que nous connaissons également les vraies positions des nœuds. La précision est calculée comme suit:

$$PE_L = (1 - \frac{\sum_i^N \sqrt{(Xti - Xri)^2 + (Yti - Yri)^2}}{N}) * 100\ \%$$

Tel que :

Xr,Yr :les positions réelles d'un nœud .

P_L : la précision de localisation

Xt,Yt :les positions d'un nœud trouvées par DvHop.

N :nombre des nœuds.

3.4 Résultat de simulation

Evaluation la métrique de précision de localisation

Cette évaluation montre l'efficacité de la phase de calcule la distance entre les ancres et distance (hop-size) pour chaque nœud. Le but est de savoir combien de nœuds du réseau peuvent être localisés dans une position qui se trouve la plus approche à la position exacte. Pour cela nous avons défini la précision de localisation comme le pourcentage de la position trouvée des nœuds sur la position réelle des nœuds. La précision de la localisation est évaluée en fonction d'un seul paramètre : le nombre de nœuds.

Nombre nœud	5	10	15	20	30
Erreur de localisation(%)	63 ,45	47 ,35	41,89	32,78	19,23

Tableau 4.1 : le pourcentage de précision de l'algorithme.

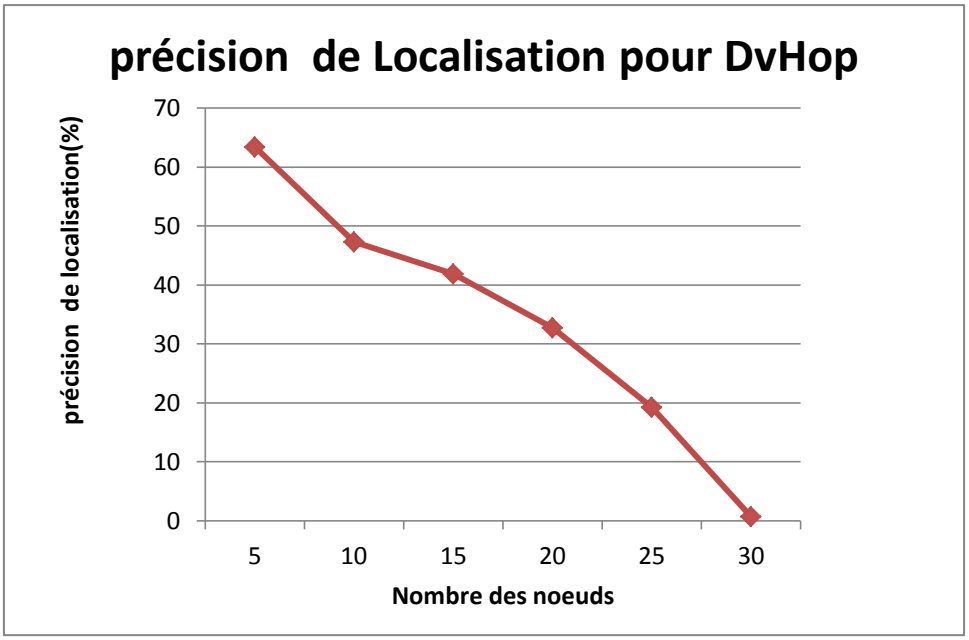

Figure 4.4 : La précision de localisation pour algorithme DvHop

Les figures (Figure 4.4) montre l'erreur de localisation pour des réseaux constitués respectivement de 5, 10 et 15… nœuds. Le graphe montre que l'erreur augmente dans le cas où le nombre des nœuds augmente. Ceci s'explique par le fait que algorithme DVHOP n'est pas adapté pour un réseaux qui contient un grand nombre de nœuds .

4. Conclusion

Dans ce chapitre, nous avons présenté l'environnement de simulation avec lequel nous avons travaillé : simulateur TOSSIM et sa caractéristique. Nous avons également définit les paramètres de simulation ainsi que la métrique d'évaluation prise en compte dans notre étude.

Les simulations réalisées au moyen du simulateur TOSSIM ont mené à étudier des différentes caractéristiques de l'algorithme de localisation des réseaux de capteurs sans fil. Cependant, des cas non prévisibles peuvent se présenter. D'où, une émulation réelle semble indispensable pour mieux évaluer ces différentes caractéristiques.

Conclusion générale

Les réseaux de capteurs ont connu une grande évolution au cours des dernières années. Cette évolution a rencontré plusieurs contraintes dont la plus importante était la localisation. Plusieurs recherches ont été faites pour la conception de l'algorithme qui tiennent compte de cette contrainte et en plus qui minimisent la consommation d'énergie. En effet, c'est dans le cadre de ce thème que s'oriente l'objectif de notre projet de fin d'études.

Au cours de ce projet « Géo-localisation dans les réseaux de capteur sans fil » il nous a été demandé d'implémenté un algorithme de localisation pour les réseaux de capteurs. La recherche sur les réseaux de capteurs est actuellement en pleine essor. Les réseaux de capteurs sans fil sont une technologie récente. Les progrès de miniaturisation et d'allongement de la durée de vie des batteries, annoncent un futur prometteur à cette technologie. De plus, le développement de nouveaux capteurs permettra d'étendre les domaines d'applications déjà nombreux. Il a été pour moi un grand plaisir de s'intégrer à la recherche dans ce domaine.

La documentation sur TinyOS et le langage NesC reste très succincte. L'étude des divers tutoriaux permet une prise en main rapide. Par contre, chaque tutorial aborde de multiples sujets en parallèle et il faut donc réussir à séparer tous ces sujets et de comprendre comment utiliser chacun d'eux séparément. Pour obtenir d'avantage d'information, il est préférable de s'inscrire à la mailing-list de TinyOS et de parcourir les questions précédemment posées, ce qui représente un travail fastidieux. Une fois ce premier pas franchi, il est assez aisé de développer une application combinant toutes les possibilités ouvertes par NesC et les capteurs. Le simulateur TOSSIM apporte une aide non négligeable en permettant de valider l'exécution du code, mais est vite limité par le fait qu'il ne gère pas tous les capteurs de capteurs. De plus, les calculs sont vite limités à de simples opérations, car le langage NesC ne comporte pas de module de calcul mathématique. De surcroit NesC n'implémente que des types entiers, donc une erreur de calcul peut vite être introduite lors d'une division.

D'un point de vue personnel ce projet m'a permis de découvrir la programmation sur des systèmes embarqués, qui est beaucoup plus limitée en termes de ressources matérielles et de calculs. Les réseaux de capteurs sont le monde de demain, avec un très large éventail d'utilisation tel que la domotique, pour commander l'allumage de chauffage en fonction de la température, la localisation de personnes telles que des pompiers dans des bâtiments lors d'interventions.

Comme perspective, il serait très intéressant à la consommation d'énergie dans les noeuds puisqu'elle constitue le paramètre le plus critique dans la détermination de la durée de vie d'un réseau de capteur et de joindre les avantages des algorithmes range-based en terme de précision aux algorithmes range-free. Par exemple rajouter le RSSI à l'algorithme DVHOP afin de minimiser l'erreur de précision tout en gardant sa simplicité. Une bonne hybridation permettra d'avoir un algorithme de géo-localisation ayant comme avantage une meilleure précision, à moindre coût.

Bibliographie

[CS09] Mr.Clément SAAD, « Quelques contributions dans les réseaux de capteurs sans fil: Localisation et Routage »,thèse pour obtenir un diplôme de DOCTORAT, l'Université d'Avignon et des Pays de Vaucluse ,2009.

[FA08] : Mr. fares Abdelfatah, « Développement d'une bibliothèque de capteur sans fil », diplôme de master en informatique, université Montpellier 2, avril 2008

[YR07]M. Yasser ROMDHANE, « Evaluation des performances des protocoles S-MAC et Directed Diffusion dans les réseaux de capteurs », Projet De Fin d'Etudes, école supérieur de communication de Tunis,2007

[JA06] J.A. Costa, N. Patwari, and A.O. Hero III. «Distributed weighted multidimensional scaling for node localization in sensor networks». ACM Transactions on Sensor Networks (TOSN), 2(1) :39_64, 2006.

[PL04] J. Pugh. «Local Range and Bearing Sensing Using Infrared Transceivers in Mobile Robotics». 2004.

[WZ06]Wassim ZNAIDI **, «** Modélisation formelle de réseaux de capteurs à partir de TinyOS »,projet fin d'etude, école polytechnique de tunisie,2006.

[SR] Seattle Robotics Society. « Implementing infrared object detection».

[NP00] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. «The Cricket location-support system. Proceedings of the 6th annual international conference on Mobile computing and networking, pages 32_43, 2000.

[KW05] K. Whitehouse, C. Karlof, A. Woo, F. Jiang, and D. Culler. «The effects of ranging noise on multihop localisation : an empirical study». Proceedings of the fifth international conference on Information processing in sensor networks, 2005.

[JC05] J.P. Curty, M. Declercq, C. Dehollain, and N. Joehl. « Design and Optimization of Passive UHF RFID Systems». Springer, 2007.

[MK05] M. Kushwaha, K. Molnar, J. Sallai, P. Volgyesi, M. Maroti, and A. Ledeczi. , «Sensor Node Localization Using Mobile Acoustic Beacons». PhD thesis, Vanderbilt University, 2005.

[HM07]V.Haegen Mathieu, « Réseaux de senseurs sans fil : problèmes de localisation », Mémoire pourl'obtention du grade de licencié, Université Libre de Bruxelles ,Département d'informatique,2007.

[BU02] N. Bulusu. « Self-Con_guring Location Systems». PhD thesis, University of California, 2002.

[BH00] N. Bulusu, J. Heidemann, and D. Estrin. « GPS-less low-cost outdoor localization for very small devices». Personal Communications, IEEE, 7(5) :28_34, 2000.

[CB08]Cedric BASTIEN ,« developpement d'un outil permettant le suivi d'un objet mobile », projet de fin d'etude,2008.

[HS08]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 2,25.5.2008.

[BS10] Belaidi Soufiane, « Les Protocoles de Routage dans les Réseaux de Capteurs sans Fil », diplôme d'ingénieur en informatique, Université A.B Tlemcen, Juillet 2010.

Annexe1

Code source de l'application :

DVhopM.nc

```
//
// Projet PFE d' ingénieur en informatique



includes DvhopMsg;

module DvhopM {
        provides {
        interface StdControl;
        }
        uses {
        interface Leds;
        interface Timer as dvtime;
        interface SendMsg as dvsend;
        interface ReceiveMsg as dvrecev;
        }
        }
implementation {
 TOS_Msg message;
 TOS_MsgPtr buff;
 bool etatg,etatr;
```

```
bool anchorone,anchortwo;
int inf[1][3];
int tab [2][3];
int  x,y,i,j,num;
```

```
norace float distance[2];
command result_t StdControl.init(){
    dvhop_msg * pointt;

pointt =( dvhop_msg* ) message.data;
   switch(TOS_LOCAL_ADDRESS){
      case 0 : x=2;
               y=2;
      pointt->id_anchor = 0;
            pointt->hop =0;
            pointt->x = 2;
            pointt->y = 2;
             pointt->hopsize=0;

             break;
       case 1 :x=2;
               y=8;
       pointt->id_anchor = 1;
            pointt->hop =0;
            pointt->x = 2;
            pointt->y = 8;

           pointt->hopsize=0;

           break;
       case 2 :x=5;
```

```
            y=6;
        pointt->id_anchor = 2;
           pointt->hop =0;
           pointt->x = 5;
           pointt->y = 6;
            pointt->hopsize=0;


            break;
       default :
           atomic distance [0]=distance [1]=distance [2]= 0;
      for(i=0;i<=1;i++){
      for(j=0;j<=3;j++){
      inf[i][j]=6;


             }}
             for(i=0;i<=2;i++){
      for(j=0;j<=3;j++){
      inf[i][j]=0;
             }}


                        }
     call Leds.init();
     atomic num = 0;
          atomic etatg = etatr = TRUE ;
          atomic anchorone=anchortwo= TRUE;
     return SUCCESS;
}}}

ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
command result_t StdControl.start(){
     if (TOS_LOCAL_ADDRESS == 0 || TOS_LOCAL_ADDRESS == 1 ||
TOS_LOCAL_ADDRESS == 2){
```

```
      call dvtime.start( TIMER_REPEAT,500);


     call Leds.greenToggle();


      }
      else {
     // call dvtime.start(TIMER_ONE_SHOT,1000);


     call Leds.redOn();


      }
      return SUCCESS;
      }



      //****************************** command pour
stop***********************************************
 command result_t StdControl.stop(){
   call dvtime.stop();
   return SUCCESS;
    }

//**********************************************timer*************************
*****************


   event result_t dvtime.fired(){
   //call send cammand
   dvhop_msg * poi;
   poi=(dvhop_msg *)message.data;
   switch (TOS_LOCAL_ADDRESS){
   case 0:
   poi->id_node = TOS_LOCAL_ADDRESS;
```

```
//poi->id_anchore = 0;

if (call dvsend.send(TOS_BCAST_ADDR,sizeof(dvhop_msg),&message)== SUCCESS){
call Leds.greenOn();
break;
}



case 1: poi->id_node = TOS_LOCAL_ADDRESS;
//poi->id_anchore = 0;

if (call dvsend.send(TOS_BCAST_ADDR,sizeof(dvhop_msg),&message)== SUCCESS){
call Leds.greenOn();
break;
}




case 2: poi->id_node = TOS_LOCAL_ADDRESS;
//poi->id_anchore = 0;

if (call dvsend.send(TOS_BCAST_ADDR,sizeof(dvhop_msg),&message)== SUCCESS){
call Leds.greenOn();
break;
}
}
return SUCCESS;



}
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


  event result_t dvsend.sendDone(TOS_MsgPtr sentMessage,result_t result){

   if (TOS_LOCAL_ADDRESS == 0 || TOS_LOCAL_ADDRESS == 1 || TOS_LOCAL_ADDRESS == 2){

  dbg(DBG_USR1, "anchor send de message:%d\n",TOS_LOCAL_ADDRESS);

  return SUCCESS;}

  else{ dbg(DBG_USR1, "node send de message:%d\n",TOS_LOCAL_ADDRESS);

  return SUCCESS;}

  return FAIL;

  }


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

task void verification(){

int m;

uint16_t k = 0;


for (m=0;m >= 2;m++){

k=k+tab[m][3];}

if(k==3){

atomic etatr= FALSE;

}

}

//*****************************etape reception************************


 event TOS_MsgPtr dvrecev.receive(TOS_MsgPtr recv_packet) {

   dvhop_msg * poi;

    dvhop_msg * po;

  dbg(DBG_USR1, "evenement declenche de message\n");

  if ( TOS_LOCAL_ADDRESS == 2 && etatg )
```

```
{ dbg(DBG_USR1, "entrer dans partie 1 de message\n");
// dvhop_msg * poi;
 //poi=(dvhop_msg *)message.data;
  poi = (dvhop_msg*) recv_packet->data;
 po =(dvhop_msg*) message.data;
   if (poi->id_node != 0 && poi->id_node != 1 && poi->id_node != 2)
   {dbg(DBG_USR1, "anchor entrer dans partie premeir condition\n");

      if (poi->id_anchor != 2 )

    {dbg(DBG_USR1, "anchor entrer dans partie deuxieme condition\n");
      switch ( poi->id_anchor){

      case 0:if (anchorone){
    atomic{
      inf[0][0]=poi->id_anchor;
      inf[0][1]=poi->x;
      inf[0][2]=poi->y;
      inf[0][3]=poi->hop;
      anchorone= FALSE;
      }
      }
     case 1:if(anchortwo){
      atomic{
      inf[1][0]=poi->id_anchor;
      inf[1][1]=poi->x;
      inf[1][2]=poi->y;
      inf[1][3]=poi->hop;
      anchortwo= FALSE;
      }
                    }
```

```
                    }


        dbg(DBG_USR1, "anchor entrer dans partie 1et perndre les inf de message\n");
         if (!anchorone &&!anchortwo)
                { //calculhopsize et envoi à les node
                dbg(DBG_USR1, "anchor calcul hopesize \n");
                atomic {
                float s =pow((x-inf[0][1]),2)+ pow(y-inf[0][2],2);
                 float ss =pow((x-inf[1][1]),2)+ pow(y-inf[1][2],2);
                 float h=(sqrt(s)+sqrt(ss))/(inf[0][3]+inf[1][3]);
                 po->hopsize = h;
                 dbg(DBG_USR1, "anchor calcul hopesize=%f\n",h);
                 po->id_node =TOS_LOCAL_ADDRESS;
                 }
                 if   (call   dvsend.send(TOS_BCAST_ADDR,sizeof(dvhop_msg),&message)==
SUCCESS){
                  dbg(DBG_USR1, "anchor envoyer le hopesize\n");
                atomic etatg= FALSE;
                 //call Leds.redOff();

                 //call Leds.greenOff();

                 call Leds.yellowOn();
                 }
                }

      }
    }
   }
```

```
 if (TOS_LOCAL_ADDRESS != 0 && TOS_LOCAL_ADDRESS != 1
&&TOS_LOCAL_ADDRESS != 2 && etatg)


 {dbg(DBG_USR1, "node receptioner une message\n");
  poi = (dvhop_msg*) recv_packet->data;
po =(dvhop_msg*) message.data;
   if (etatr)
   {dbg(DBG_USR1, "entrer dans etatr\n");


 for(i=0;i<=2;i++)
 {dbg(DBG_USR1, "entrer dans boucle for\n");
 if(poi->id_anchor==i && tab[i][3]!=1 )
 {
atomic  { inf[i][0]=poi->id_anchor;
  tab[i][1]=poi->id_node;
  tab[i][2]=poi->hop;
  tab[i][3]=1;
  po->hop =poi->hop+1;
  po->id_node=TOS_LOCAL_ADDRESS;
  dbg(DBG_USR1, "node entrer dans partie 1 perndre les inf de message\n");
      }
  if (call dvsend.send(TOS_BCAST_ADDR,sizeof(dvhop_msg),&message)== SUCCESS){
  //call Leds.greenOn();
   dbg(DBG_USR1, "entrer dans partie 2et envoyer les inf de message\n");
                                                  }
  post verification();
 break;


  }
  }
  }
```

```
    else
    {
    if(poi->hopsize > 0 && poi->id_anchor ==2)
    { dbg(DBG_USR1, "node entrer dans partie 3 pour calcul distance\n");
     for(j=0;j <= 2;j++){

   atomic distance[j]=tab[j][2]* poi->hopsize;
                    }
     dbg(DBG_USR1, "node calcul: distance est d1:%f llllllllD2:f llllllllD3:f",distance[0],distance[1],distance[2]);
     po->hopsize=poi->hopsize;
     po->id_anchor=poi->id_anchor;
     po->id_node=TOS_LOCAL_ADDRESS;
     if (call dvsend.send(TOS_BCAST_ADDR,sizeof(dvhop_msg),&message)== SUCCESS){
    atomic  etatg= FALSE;
    call Leds.yellowOn();
}    }
    }return recv_packet;
  }
}
```

Annexe2

Caractéristiques techniques des capteurs Micaz et de la station réceptrice MIB610 (2/2)

Crossbow

Processor/Radio Board	MPR2400CA	Remarks
Processor Performance		
Program Flash Memory	128K bytes	
Measurement (Serial) Flash	512K bytes	> 100,000 Measurements
Configuration EEPROM	4K bytes	
Serial Communications	UART	0-3V transmission levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	Digital I/O,I2C,SPI	
Current Draw	8 mA	Active mode
	< 15 µA	Sleep mode
RF Transceiver		
Frequency band[1]	2400 MHz to 2483.5 MHz	ISM band, programmable in 1 MHz steps
Transmit (TX) data rate	250 kbps	
RF power	-24 dBm to 0 dBm	
Receive Sensitivity	-90 dBm (min), -94 dBm (typ)	
Adjacent channel rejection	47 dB	+ 5 MHz channel spacing
	38 dB	- 5 MHz channel spacing
Outdoor Range	75 m to 100 m	1/2 wave dipole antenna, LOS
Indoor Range	20 m to 30 m	1/2 wave dipole antenna
Current Draw	19.7 mA	Receive mode
	11 mA	TX, -10 dBm
	14 mA	TX, -5 dBm
	17.4 mA	TX, 0 dBm
	20 µA	Idle mode, voltage regular on
	1 µA	Sleep mode, voltage regulator off
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 V - 3.3 V	Molex connector provided
User Interface	3 LEDs	Red, green and yellow
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51-pin	All major I/O signals

Notes
[1]5 MHz steps for compliance with IEEE 802.15.4/D18-2003.
Specifications subject to change without notice

MIB520CB Mote Interface Board

Base Stations

A base station allows the aggregation of sensor network data onto a PC or other computer platform. Any MICAz Mote can function as a base station when it is connected to a standard PC interface or gateway board. The MIB510 or MIB520 provides a serial/USB interface for both programming and data communications. Crossbow also offers a stand-alone gateway solution, the MIB600 for TCP/IP-based Ethernet networks.

Résumé

Comme le firent Internet et les communications sans fil il y a de cela quelques décennies, l'avènement des réseaux de capteurs s'apprête à révolutionner notre mode de vie., un certain nombre de problématiques doit être résolu. Aux contraintes traditionnelles des réseaux ad hoc s'ajoutent les limites très strictes liées aux caractéristiques matérielles des capteurs telles que la puissance de calcul, la mémoire et surtout l'alimentation en énergie, rendant les algorithmes existants inadaptés, La localisation est une problématique importante pour les réseaux de capteurs sans fil, en particulier en intérieur, là où le GPS est inutilisable. Les algorithmes de localisation existants se rangent en deux catégories : «range-based» et «range-free». Les techniques «range-based » partent d'une évaluation de la distance entre émetteur et récepteur radio, la technique «range-free» est plus économique en matériel car elle se contente de l'information de connectivité liée à la portée radio, Ce mémoire aborde à problématiques: celle de la localisation et nous allons devoir implémentera l'algorithme DVHOP qui permettra de localiser les capteurs par le NesC un langage de programmation pour system exploitation :TinyOS

Mots clés : réseaux de capteurs, localisation, DVHOP,TinyOS,NesC

Abstract

As did it Internet and wireless communications some decades ago, the advent of sensor networks prefaces a revolution in our way of life, some problems must be solved. To traditional constraints of ad hoc networks, should be added hardware limitations of sensors such as computing power, memory size and especially energy consumption. In wireless Sensor Networks, localization is an important issue, especially indoors, where GPS is unavailable. Two categories of localization algorithms exist, range-based and range-free. Range-based techniques measure the distance between two nodes, while range-free method use the connectivity information, this memory address to problem: localization and we have to write algorithm DVHOP to locate the mote by NesC programmation language for operating system : TinyOS.

Keywords : sensor networks, localization,DVHOP,TinyOS,NesC

ملخص

كما فعلت الإنترنت اللاسلكية ومنذ عقود قليلة ، وظهور شبكات الاستشعار على وشك أن تحدث ثورة في أسلوب حياتنا. يجب حل عدد من القضايا. تضاف القيود التقليدية للشبكات مخصصة حدود صارمة تتعلق الخصائص الفيزيائية للاستشعار مثل القدرة الحاسوبية ، والذاكرة ، وخصوصا إمدادات الطاقة ، مما يجعلها غير صالحة خوارزميات القائمة ، والموقع هو مسألة هامة بالنسبة شبكات الاستشعار اللاسلكية ، وخصوصا في الداخل ، حيث GPS غير قابل للاستخدام. خوارزميات الموقع الحالي تنقسم إلى قسمين : "مجموعة مقرها" و "حر المدى". تقنيات "مجموعة مقرها" بداية من تقييم المسافة بين المرسل والمتلقي والراديو ، وتقنية "المدى الحر" المعدات لأنها أرخص من المعلومات الاتصال الوحيدة ذات الصلة لمجموعة الراديو ، وهذا عناوين لمشاكل في الذاكرة : موقع وسيكون لدينا لتنفيذ DVHOP خوارزمية لتحديد مكان أجهزة الاستشعار بواسطة لغة البرمجة nesC لنظام التشغيل : TinyOS

الكلمات الرئيسية : شبكات الاستشعار ، التعريب ، DVHOP، TinyOS، nesC