Université Abou Bekr Belkaid 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 de Master en Informatique

Option : *Réseaux et Systèmes Distribués*

Thème

Etude et évaluation du protocole de localisation DV-HOP dans un réseau de capteurs sans fil

Réalisé par :

- **MEKIDICHE Mohammed**
- **RAIS Hichem**

Présenté le 01 Juillet 2012 devant le jury composé de MM.

- *Mr BENMAMMAR Badr* *(Président)*
- Mme LABRAOUI Nabila *(Encadreur)*
- *Mr LEHSAINI Mohammed* *(Examinateur)*
- *Mr BENAISSA Mohammed* *(Examinateur)*

Année universitaire : 2011-2012

Table de matières

Introduction générale

Le besoin d'observer et éventuellement de contrôler des phénomènes physiques tels que la température, la pression ou encore la luminosité est essentiel pour de nombreuses applications industrielles, scientifiques, et même grand public. Cette tâche est déléguée aux capteurs dont la fonction est l'acquisition de l'information sur les phénomènes observés et, le cas échéant, l'exécution des traitements qui s'y attachent. L'utilisation des capteurs n'est pas une nouveauté en soi. En effet, grâce aux récents progrès des technologies sans fil, les capteurs peuvent communiquer non seulement de proche en proche mais aussi d'acheminer de l'information à tous les nœuds connectés au réseau. On s'est ainsi affranchi de la contrainte de câblage, qui limitait considérablement le déploiement d'un grand nombre de nœuds. Il est donc devenu tout à fait possible de déployer un réseau constitué d'un grand nombre de capteurs collaboratifs afin de surveiller une zone plus large.

Les réseaux de capteurs sans fil sont considérés comme un type spécial de réseaux ad hoc. Ils apportent une perspective intéressante : celle de réseaux capables de s'auto-configurer et de s'auto-gérer sans qu'il y ait besoin d'interventions humaines. Les nœuds sont généralement déployés de manière aléatoire à travers une zone géographique, appelée zone d'intérêt.

Les données récoltées sont acheminées grâce à des communications sans fil en multi-saut (c.-à-d. de proche en proche) à une station de base dont le rôle est entre autre d'agréger/exploiter les données récoltées. Elle représente en quelque sorte le point d'entrée du réseau de capteurs.

Parmi les problèmes cruciaux, deux d'entre eux peuvent être cités :

- **celui de la Localisation** : une grande majorité des applications dans les réseaux de capteurs utilise un déploiement aléatoire d'un grand nombre de capteurs, en raison soit de l'hostilité de la zone à surveiller, soit de son immensité. La phase de localisation est donc nécessaire non seulement au fonctionnement du réseau (routage géographique par exemple), Il est donc nécessaire de localiser, avec la meilleure précision possible, tous les noeuds du réseau. Cette problématique, malgré les nombreux travaux de recherche qui s'y étaient attachés, reste une problématique ouverte.

- **Couverture** : une des conséquences du déploiement aléatoire est la redondance des capteurs sur la zone surveillée. Il est donc tout à fait judicieux, afin de prolonger au maximum la durée de vie du réseau, de mettre en veille un certain nombre de capteurs redondants tout en assurant une couverture totale de la zone surveillée et en maintenant la connectivité du réseau.
- **Fusion de données :** dans certains cas de figures, il est nécessaire que tous les nœuds du réseau aient à leur disposition un agrégat tel que la moyenne de tous les prélèvements effectués. Ceci dans le but d'effectuer une action concertée par exemple. Cette tâche, d'ordinaire si facile à effectuer dans un réseau fiable, devient rapidement problématique dans le cas d'un réseau sujet à la fois à des perturbations environnementales constantes et à des défaillances fréquentes.

Bien d'autres problèmes tels que 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 précision.

Nous avons également proposé une amélioration de l'algorithme DV-HOP, afin de minimiser l'erreur d'estimation. Nos résultats de simulation ont démontré l'efficacité de notre proposition par rapport à la version basique de DV-HOP.

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 le langage NesC.

Le chapitre 4 : constitue le coeur de notre travail, dans ce chapitre nous présentons le simulateur TOSSIM avec une brève description de ses composants, ses principales caractéristiques et fonctionnalités. Par la suite nous donnons les résultats de simulation sous forme de graphes de plusieurs

simulations, effectuées pour obtenir des mesures pour évaluer la précision de calcul des positions.

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

Chapitre 1

Les Réseaux de capteurs sans fil:

Description, protocole

1. Introduction

Les réseaux de capteurs sans fil sont un cas particulier des réseaux sans fil sans infrastructure (réseaux ad hoc). En effet, ceux-ci sont constitués d'un ensemble de petits appareils, ou capteurs, possédant des ressources particulièrement limitées mais qui leur permettent d'acquérir des données sur leur environnement immédiat, de les traiter et de les communiquer. Ils présentent des intérêts considérables pour le secteur industriel, mais aussi pour les organisations civiles où la surveillance et la reconnaissance de phénomènes physiques sont une priorité. En effet, un réseau de capteurs peut être mis en place dans le but de surveiller une zone géographique plus ou moins étendue pour détecter l'apparition de phénomènes ou mesurer une grandeur physique (température, pression, vitesse...).

2. Définitions

Un réseau de capteur sans fil (Wireless Sensor Network ; WSN, ou RCSF) est un type spécial de réseau ad-hoc défini par un ensemble coopérant de nœuds capteurs dispersés dans une zone géographique appelée zone de captage afin de surveiller un phénomène et récolter ses données d'une manière autonome. Les nœuds capteurs utilisent une communication sans fil pour acheminer les données captées avec un routage multi sauts vers un nœud collecteur appelé nœud puits (ou sink) qui va transmettre, via internet ou satellite, ces informations à l'utilisateur du réseau (Figure I.1). Ainsi, l'usager peut adresser des requêtes aux autres nœuds du réseau, précisant le type de données requises, puis récolter les données environnementales captées par le biais du nœud collecteur.

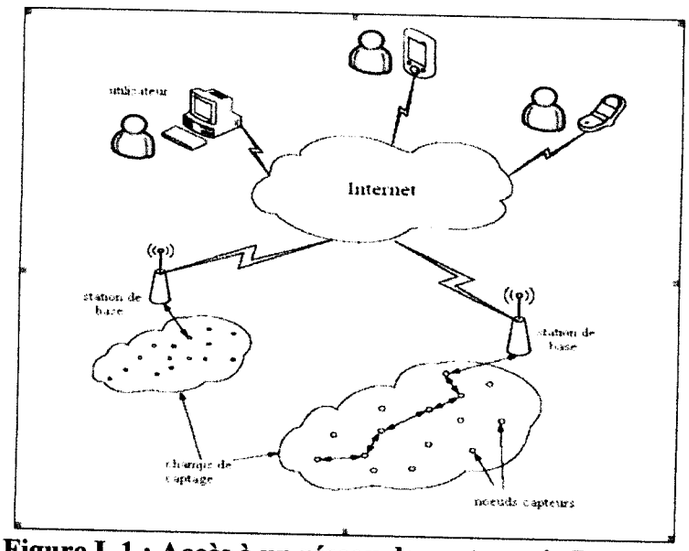

Figure I. 1 : Accès à un réseau de capteur via Internet.

3. Domaines d'application des réseaux de capteurs

La miniaturisation des micro-capteurs, le coût de plus en plus faible, la large gamme des types de capteurs disponibles (thermique, optique, vibrations, etc.) ainsi que le support de communication sans fil utilisé, permettent l'application des réseaux de capteurs dans plusieurs domaines parmi lesquels :

3.1 Domaine militaire

Comme pour de nombreuses autres technologies, le domaine militaire a été le moteur initial pour le développement des réseaux de capteurs.

Le déploiement rapide, le coût réduit, l'auto organisation et la tolérance aux pannes des réseaux de capteurs sont des caractéristiques qui font de ce type de réseaux un outil appréciable dans un tel domaine. Actuellement, les RCSFs peuvent être une partie intégrante dans le commandement, le contrôle, la communication, la surveillance, la reconnaissance, etc.

3.2 Domaine médical

Les réseaux de capteurs sont également largement répandus dans le domaine médical. Cette classe inclut des applications comme : fournir une interface d'aide pour les handicapés, collecter des informations physiologiques humaines de meilleure qualité, facilitant ainsi le diagnostic de certaines maladies, surveiller en permanence les malades et les médecins à l'intérieur de l'hôpital.

3.3 Domaine architectural :

Transformation des bâtiments en environnements intelligents capables de reconnaître des personnes, interpréter leurs actions et y réagir.

3.4 Domaine environnemental

Dans ce domaine, les capteurs peuvent êtres exploités pour détecter les catastrophes naturelles (feux de forêts, tremblements de terre, etc.), détecter des fruits des produits toxiques (gaz, produits chimiques, pétrole, etc.) dans des sites industriels tels que les centrales nucléaires et les pétrolières.

3.5 Domaine commercial

Parmi les domaines dans lesquels les réseaux de capteurs ont aussi prouvé leur utilité, on trouve le domaine commercial. Dans ce secteur on peut énumérer plusieurs applications comme : la surveillance de l'état du matériel, le contrôle et l'automatisation des processus d'usinage, etc.

4. Architecture physique d'un capteur

Un capteur est composé de quatre composants de base: Unité de capture, unité de traitement, unité d'émission/réception, et une unité d'énergie. Il se peut aussi qu'il existe d'autres composants additionnels dépendant de l'application, par exemples: un générateur d'énergie, un système de localisation, et un mobilisateur [1].

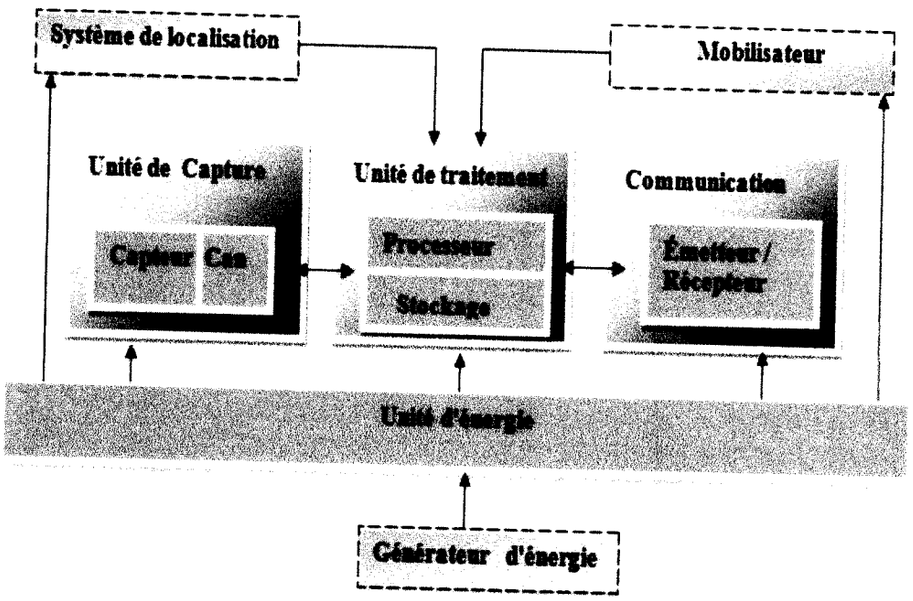

Figure I. 2: les composants d'un capteur

4.1 Unité de capture

La fonction principale de l'unité de capture est de capturer ou mesurer les données physiques à partir de l'objet cible. Le signale analogique correspondant aux évènements observés par le capteur est ensuite transformé en données numériques qui peuvent être utilisées par l'unité de traitements.

4.2 Unité de traitements

L'unité de traitement joue un rôle majeur dans la collaboration entre les nœuds afin d'accomplir les tâches prédéfinies. Actuellement, il existe plusieurs familles d'unités de traitement incluant les microcontrôleurs, microprocesseurs, et FPGAs (Field Programmable Gate Arrays).

Les FPGAs consomment plus d'énergie et ne sont pas compatibles aux méthodologies de programmation traditionnelles, mais le fait qu'ils soient programmables et reconfigurables [2] présente un réel avantage.

L'unité de traitement a besoin de stocker les informations pendant le traitement local et l'agrégation des données, une mémoire flash (mémoire non volatile servant au stockage stable d'informations) est généralement utilisée vu son coût et capacité de stockage

4.3 Unité d'émission/réception

Il existe trois schémas de communication pour les réseaux de capteurs: la communication optique (Laser), l'infrarouge, et la radiofréquence (RF: Radio Frequency).

Le Laser consomme moins d'énergie que la RF et fournit une haute sécurité, mais exige l'utilisation d'une ligne optique et il est sensible à la perturbation physique.

L'infrarouge n'a pas besoins d'antennes, mais sa capacité de diffusion est limitée.

La RF est la plus simple à utiliser mais elle exige l'utilisation des antennes [2].Plusieurs stratégies de réduction de la consommation d'énergie sont développées, comme la modulation/démodulation et le filtrage. La modulation en amplitude est plus simple par rapport à celle en fréquence, mais elle est susceptible au bruit

4.4 Unité d'énergie

La consommation d'énergie est un point très important pour les réseaux de capteurs. Les batteries utilisées sont soit rechargeables ou non. Souvent, dans les environnements sensibles, il est impossible de recharger ou changer une batterie, donc avoir une meilleure gestion de la consommation d'énergie est primordial pour augmenter la durée de vie du réseau.

Il existe deux grandes politiques pour conserver la consommation d'énergie. Dans la première, appelé DPM (Dynamic Power Management), les composants inactifs sont mis en veille. Dans ce cas une analyse stochastique pour prédire les prochains évènements est Généralités sur les réseaux de capteurs nécessaires.

Pour la deuxième, appelé DVS (Dynamic Voltage Scheduling), l'énergie est fournie aux composants selon leur charge de travail

Les nouveaux capteurs peuvent contenir des générateurs d'énergie renouvelable, par exemple: l'énergie solaire, et l'énergie mécanique (vibration, l'aire…)

Certaines applications ont besoin de savoir l'emplacement du capteur. Pour cette raison, le capteur doit avoir un système de localisation telle qu'un GPS (Global Positioning System).

Pour les réseaux de capteurs mobiles, des nœuds doivent se déplacer, donc un mobilisateur doit exister dans les composants du capteur [1].

5. Exemple de capteur sans fil

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

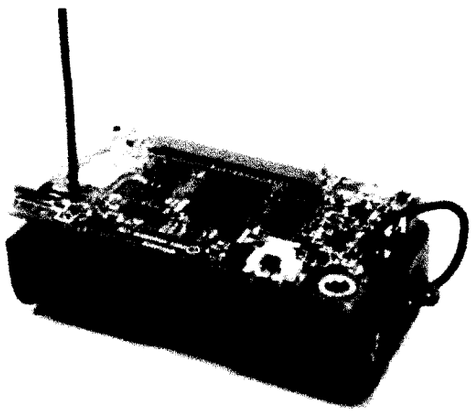

Figure I. 3: Capteur Mica2.

5.1 Le capteur Mica2

Chaque Mica2 est équipé d'un processeur cadencé à 7,37 Mhz et doté de 4ko de RAM, de 128 ko de mémoire flash et d'un transmetteur radio à 433 MHz.

Les Mica2 peuvent être équipés de plusieurs types de circuits intégrés permettant d'effectuer des mesures de phénomènes naturels. Chaque carte supporte plusieurs capteurs sous forme de composants électroniques. L'équipe SOD du LaBRI dispose de Mica2 munis de deux types de cartes [3]:

A.La carte Crossbow MTS300

MTS300 est équipée d'un capteur de température, d'intensité lumineuse, d'un microphone et d'un buzzer à 4kHz (figure I.4)

B.La carte Crossbow MTS420

MTS420, beaucoup plus complète, comporte un capteur d'humidité et de température, un capteur de lumière ambiante, un baromètre, un accéléromètre à deux axes et une puce GPS (figure I.5)

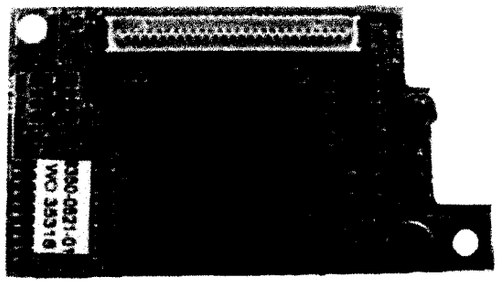

Figure I. 4: carte Crossbow MTS300

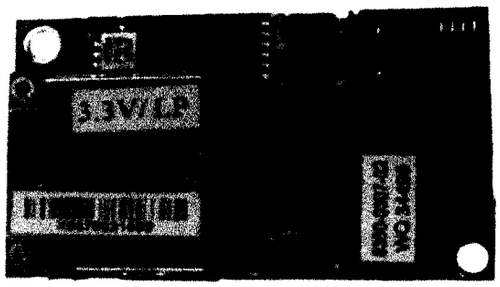

Figure I. 5: carte Crossbow MTS420

6. Architecture des réseaux de capteurs sans fil

Un réseau de capteurs est constitué essentiellement de : plusieurs nœuds capteurs, un nœud Sink ou plusieurs et un centre de traitement des données.

6.1 Nœuds

Un réseau de capteurs sans fil est un réseau ad hoc avec un grand nombre de nœuds qui sont des micros capteurs capables de récolter et de transmettre des données environnementales d'une manière autonome. La position de ces nœuds n'est pas obligatoirement prédéterminée. Ils peuvent être aléatoirement dispersés dans une zone géographique, appelée « champ de captage » correspondant au terrain d'intérêt pour le phénomène capté.

En plus d'applications civiles, il existe des applications militaires aux réseaux de capteurs (détection d'intrusions, localisation de combattants, véhicules, armes, etc. sur un champ de bataille, sous l'eau, dans l'espace, dans le sol...)

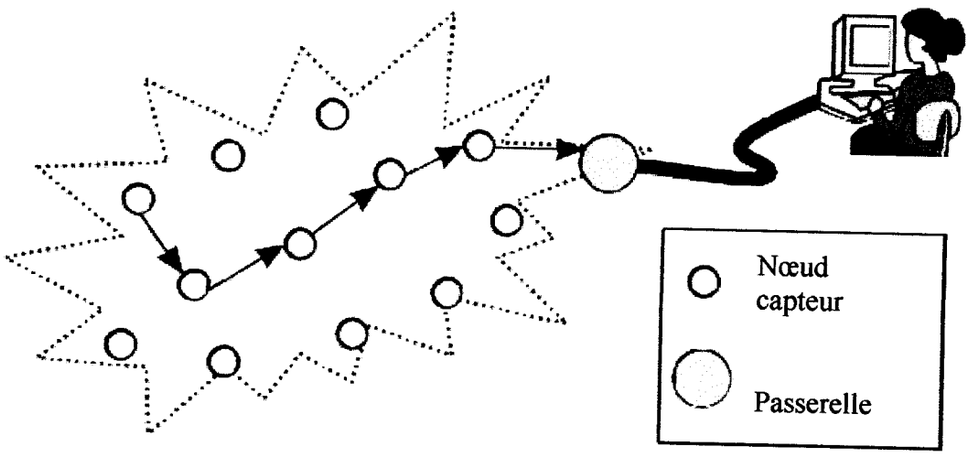

Figure I. 6: les Nœuds d'un réseau capteur.

6.2 Sink

Et c'est un nœud particulier dans le réseau et est le responsable de la collecte de données issues des différents nœuds de réseau. Doit être active et illimitée d'énergie, nous pouvons trouver deux ou plusieurs sinks pour alléger la charge.

6.3 Centre de traitement des données

Est le centre, qui est une collection de toutes les données envoyées par les sinks, cet endroit à le rôle d'extraire des informations utiles afin d'exploiter

7. topologie d'un réseau de capteur

Il existe plusieurs topologies pour les réseaux de capteurs

7.1 Topologie en étoile

La topologie en étoile est un système uni saut. Tous les nœuds envoient et reçoivent seulement des données avec la station de base. Cette topologie est simple et elle demande une faible consommation d'énergie, mais la station de base est vulnérable et la distance entre les nœuds et la station est limitée.

7.2 Topologie en toile (en grille)

La topologie en toile est un système multi saut. La communication entre les nœuds et la station de base est possible. Chaque nœud a plusieurs chemins pour envoyer des données. Cette topologie a plus de possibilités de passer à l'échelle du réseau, avec redondance et tolérance aux fautes, mais elle demande une consommation d'énergie plus importante

7.3 Topologie hybride

La topologie hybride est un mélange des deux topologies ci-dessus. Les stations de base forment une topologie en toile et les nœuds autour d'elles sont en topologie étoile. Elle assure la minimisation d'énergie dans les réseaux de capteurs.

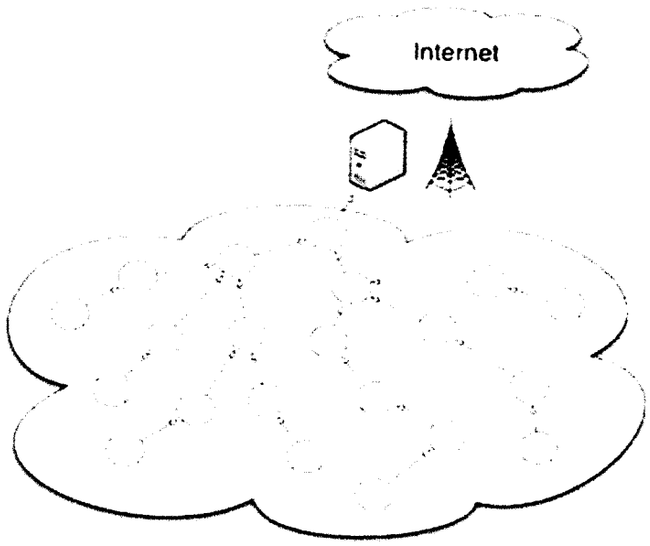

Figure I. 7: Topologie hybride d'un réseau de capteurs sans fil

8. Communication dans les réseaux de capteurs

8.1 Architecture de communication basée sur le modèle OSI

Le modèle de communication comprend cinq couches qui ont les mêmes fonctions que celles du modèle OSI ainsi que trois couches pour la gestion d'énergie, la gestion de la mobilité et la gestion des tâches[4].

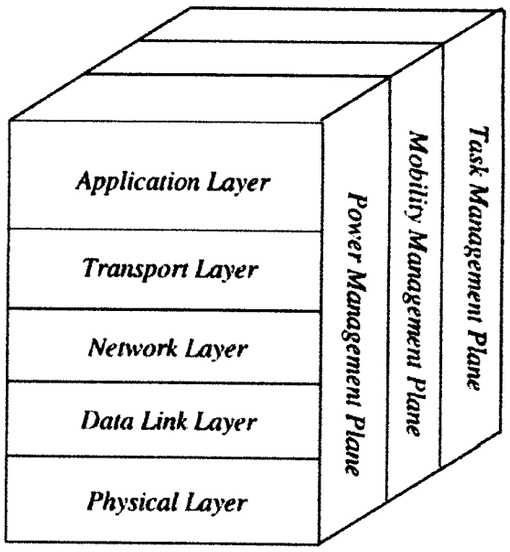

Figure I. 8: Modèle en couches du réseau de capteurs sans fil.

Rôles des couches :

A. Couche physique : Matériels pour envoyer et recevoir les données

B .Couche liaison : Gestion des liaisons entre les nœuds et les stations de base, contrôle d'erreurs

C. Couche réseau : Routage et transmission des données

D. Couche transport : Transport des données, contrôle de flux

E. Couche application : Interface pour les applications au haut niveau

F.Plan de gestion d'énergie : Contrôle l'utilisation d'énergie

G. Plan de gestion de mobilité : Gestion des mouvements des nœuds

H.Plan de gestion de tâche : Balance les tâches entre les nœuds afin d'économiser de l'énergie.

8.2 Technologies de la communication dans les réseaux de capteurs

A. Bluetooth / IEEE 802.15.1

Bluetooth est une spécification de l'industrie des télécommunications. Elle utilise une technique radio courte distance destinée à simplifier les connexions entre les appareils électroniques. Malheureusement, un grand défaut de cette technologie est sa trop grande consommation d'énergie. [17]

B. ZigBee / IEEE 802.15.4

ZigBee est une norme de transmission de données sans fil permettant la communication de machine à machine. Zigbee offre des débits de données moindres,

mais sa très faible consommation électrique et ses coûts de production très bas en font une candidate idéale pour la domotique ou les matériels de type capteur, télécommande ou équipement de contrôle dans le secteur industriel.

C. Dash 7 / ISO/IEC 18000-7

Dash7 est une nouvelle technologie de réseaux de capteurs sans fil en utilisant la norme ISO/IEC 18000-7. Sa consommation électrique est très faible, la durée de vie de batterie peut arriver à plusieurs ans.

Sa distance de communication est 2km. Elle fournit une faible latence pour le suivi des objets en mouvement, un protocole petit pile, des supports de capteurs et de sécurité et un débit de transmission allant jusqu'à 200kbits/s.

9. 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

De nombreuses applications pour les réseaux de capteurs comme la surveillance de feu de forêt, le suivi de véhicule, etc... Ont besoin d'avoir une information géographique pour fonctionner efficacement. Les protocoles de routage géographique ou orienté position vont pouvoir fonctionner sans le coûteux mécanisme de découverte de route proactive et ainsi économiser de l'énergie et améliorer le taux d'acheminement. De plus, dans les protocoles de contrôle de topologie, où chaque capteur doit ajuster sa puissance de transmission pour minimiser sa consommation énergétique, les algorithmes ont le plus souvent besoin d'information sur la position des voisins. [9]

2. Les systèmes de localisation

2.1 Les principaux systèmes de localisation

A. GPS (Global Positioning System)

Système de géo localisation par satellite. Le réseau de 24 satellites (plus 4 satellites en réserve) actuellement en fonctionnement, développé par l'armée américaine, est mis à disposition des civils. Il permet de déterminer les coordonnées géographiques de n'importe quel point situé à la surface du globe. Sa précision peut atteindre 1 mètre. [18] Le GPS s'utilise en association avec une carte pour se repérer et se positionner : randonnées, voile, trek...

B. Galileo

Est le futur système de positionnement européen par satellite. Ce système est en phase de test depuis 2004, commencera à être utilisable en 2010 et pleinement en 2013. Ce système vise à supprimer la dépendance de l'Europe à l'utilisation du système américain GPS (Global Positioning System). Cette indépendance est essentielle car le système américain souffre de restrictions sur la précision de positionnement, sur la fiabilité et sa continuité. De plus, le positionnement dans certaines régions du globe n'est pas possible avec le GPS pour des raisons techniques ou politiques.

Le système Galiléo sera entièrement sous contrôle civil, contrairement aux autres systèmes de positionnement existants. [9]

C. le system IRNSS (Indian Regional Nafigational Satellite System)

Est une proposition de système de positionnement par satellites qui serait construit et contrôlé par le gouvernement Indien. Il fournirait la position absolue à une

précision de 20 mètres à travers toute l'Inde et à une distance de 1500 à 2000 km des frontières. Un but de contrôle complet de la part du gouvernement Indien a été cité, ce qui implique que toutes les parties du projet soit construites en Inde.

D. Glonass

Signifie en russe GLObal'naya NAvigatsionnaya Sputnikovaya Sistema soit système mondial de navigation par satellite. C'est un système de positionnement développé par l'actuelle Union Soviétique et contrôlé pour le gouvernement russe par l'agence spatiale russe. Il est une alternative et complémentaire du GPS américain et du futur Galiléo européen.

Le développement de Glonass a débuté en 1976 avec pour but une couverture mondiale en 1991. Le premier lancement de satellite a eu lieu le 12 octobre 1982 et la totalité des satellites furent mis en orbite en 1995. Cependant, après l'achèvement du projet, le système se délabra avec l'effondrement de l'économie russe. Les Russes lancèrent un grand projet de restauration du système en 2001, en introduisant notamment le gouvernement indien en tant que partenaire, et accélérant ainsi le programme de restauration du système a été achevé en 2009. [9]

2.2 Le système GPS

A. La structure du système GPS

Le système GPS peut être séparé en trois parties : le segment de contrôle, le segment spatial et le segment utilisateur.

- ✓ **Le segment de contrôle**

Le segment de contrôle est formé par six stations de contrôle appartenant aux forces armées américaines de l'air (USAF), reparties tout autour du globe en fonction de la longitude.

Le but de ces stations est de contrôler la santée du segment spatial et de maintenir le temps du système, le GPS-time (GPST). De manière plus précise, ces stations permettent de contrôler l'état de santé des satellites ainsi que leur trajectoire, prédire les éphémérides des satellites et les paramètres de l'horloge, mettre a jour les messages de navigation des satellites, commander de petites manoeuvres afin de réinitialiser une orbite.

- ✓ **Le segment spatial**

Le système GPS est formé de 31 satellites (situation au 27 août 2009) en orbite quasi circulaire (excentricité <0,01) a une altitude de 20200 km. Leur période est de

11h58 minutes, soit un demi jour sidéral. Ces satellites sont repartis sur six plans orbitaux inclinés 55° par rapport au plan équatorial.

Chaque satellite contient plusieurs horloges atomiques, certains ont quatre horloges (deux au rubidium et deux au césium), certains en ont trois au césium et, dans les plans de modernisation du GPS, les prochains satellites auront des maser a hydrogène, qui sont extrêmement précis. Ce sont les stations au sol qui sélectionnent l'horloge la plus précise. En effet, ces horloges perdent ou gagnent moins d'une nanoseconde par jour !

Les satellites sont lancés par blocs qui ont tous des spécifications.

✓ **Le segment utilisateur**

Le segment utilisateur rassemble l'ensemble des utilisateurs du système, du simple utilisateur aux géomètres et aux militaires. L'ensemble de ces utilisateurs peut être sépare en deux catégories, selon la prestation du systéme qu'ils utilisent. [10]

B. Le principe de GPS

Voici une illustration en deux dimensions du principe de la détermination de la position d'un récepteur GPS:

Chacun des trois satellites S1, S2 et S3 envoie un signal indiquant sa position ainsi que le moment de l'envoi du signal.

A partir des temps de parcours de ces trois signaux, le récepteur GPS placé en M calcule sa distance à chacun des trois satellites.

Le point M est l'unique point d'intersection de ces trois cercles.

Dans l'espace, les signaux de quatre satellites sont nécessaires pour déterminer de façon univoque la position de M. Le point M est alors l'unique point d'intersection de trois sphères

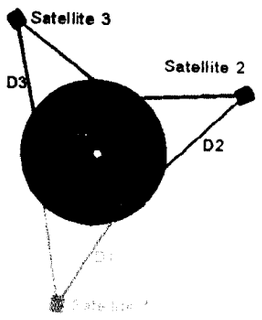

Figure II. 1: principe de mesure GPS.

3. la localisation matérielle

La localisation dans les réseaux de capteurs dépend de plusieurs dispositifs matériels. Cette dépendance se relève à l'utilisation des ancres (beacons), et à

l'estimation des distances entre les nœuds. Dans cette section nous présentons la définition d'une ancre et son mode d'utilisation, et nous introduisons également les différentes techniques d'estimation de la distance. [11]

3.1 Ancres/Beacons

L'objectif de la localisation est de déterminer les coordonnées physiques d'un groupe de nœuds. Ces coordonnées peuvent être globales, c'est à dire qu'elles sont alignées avec un système extérieur comme le système GPS par exemple, ou bien relatives, ce qui signifie qu'elles forment une transformation rigide (rotation, réflexion, translation) des coordonnées du système global. Dans le deuxième cas, on n'a pas besoin de la position des nœuds pour fonctionner, une carte relative est suffisante. Les méthodes qui créent une carte relative des coordonnées sans recours aux ancres sont appelées « anchor-free ». Par contre d'autres méthodes ne fonctionnent pas sans connaître la position d'un certain nombre d'ancres à priori, sont appelées « anchor-based ». [11]

Les ancres (souvent appelées aussi beacons) sont au préalable nécessaires pour localiser les nœuds d'un réseau dans un système de coordonnées global. Les ancres sont simplement des nœuds ordinaires qui connaissent leurs coordonnées à priori. Cette connaissance pourrait être difficilement codée, ou bien facilement acquise par un certain matériel supplémentaire comme un récepteur GPS. Au minimum, trois ancres non-colinéaires sont nécessaires pour définir un système de coordonnées en deux dimensions.

Les ancres peuvent être utilisées de plusieurs façons. Certains algorithmes de localisation trouvent une carte arbitraire relative pour les coordonnées des nœuds, puis ils utilisent les ancres pour déterminer une transformation rigide des coordonnées relatives vers les coordonnées globales. D'autres algorithmes, partant des positions des ancres, calculent les positions des nœuds non-ancres dans un système global. [11]

3.2 Estimation de distances

Parmi les méthodes de localisation dans les réseaux de capteurs nous distinguons entre deux catégories, les méthodes qui ne sont pas basées sur la distance inter-nœuds et d'autres qui y sont. Les premières sont celles qui ne calculent pas de distances entre voisins. Elles utilisent d'autres informations telles que la connectivité pour estimer la position des nœuds.

Les deuxièmes sont des méthodes qui estiment les distances entre les nœuds pour calculer les positions. Plusieurs techniques sont développées pour les estimations des distances entre les nœuds voisins. Parmi lesquelles nous trouvons celles qui sont basées sur les dispositifs radio comme la méthode de la force du signal reçu « Received Signal Strength Indication (RSSI) » et la technique d'estimation de la distance par le nombre de sauts radios « Radio hop count » ; et celles qui sont fondées sur l'utilisation d'autres matériels (microphones, etc) comme la technique de la différence entre les temps d'arrivée de deux signaux « Time Différence of Arrival (TDoA) » Et celle qui estime l'angle d'arrivée du signal « Angle of Arrival (AoA) ».

4. Critères de localisation

Un algorithme de localisation est évalué selon une liste de critères dont nous citons :

4.1 Précision de la localisation

L'erreur de la localisation est souvent définit comme étant, la distance euclidienne entre les vraies positions des nœuds et celles estimées par l'algorithme.
L'objectif d'un algorithme de localisation est de minimiser cette erreur pour augmenter la précision de localisation. Généralement, cette imprécision vient de l'imprécision des methodes d'estimation de la distance. Les obstacles environnementaux et les terrains irréguliers peuvent influencer la précision des algorithmes de localisation. Des obstacles comme de gros rochers peuvent interférer avec les ondes radios, et empêcher l'utilisation de « TDoA » du fait qu'on n'a plus une ligne droite.

4.2 Contraintes de ressources

Les nœuds capteurs possèdent généralement des resources très limitées. Ils possèdent de faibles processeurs et de petites mémoires, ce qui rend les grands calculs irréalisables. Par conséquent, un algorithme de localisation doit être simple et non complexe et son développement n'exige pas de grands calculs ni de grande capacité de stockage de mémoire. De plus, nous ajoutons la rapidité de l'algorithme. Avec quelle rapidité le système de localisation renvoie-t-il les positions des noeuds ? Ceci est particulièrement important, surtout lors du traçage d'un chemin d'une cible.

4.3 Contraintes énergétiques

La seule source d'énergie d'un nœud capteur est sa batterie. Pour cela, dans les réseaux de capteurs, une gestion de l'énergie très économique est nécessaire. Comme le facteur dominant de la consommation d'énergie est la communication radio, il faut trouver un algorithme de localisation qui communique le moins possible via la radio.

4.4 Passage à l'échelle

Les réseaux de capteurs sont généralement envisagés à large échelle, avec des centaines voir des milliers de nœuds. La question qui se pose, est-ce qu'un algorithme de localisation fonctionne sur un réseau de plusieurs milliers de nœuds ? Et si oui, est-il toujours aussi efficace ? Ce critère est en rapport avec le fait qu'un algorithme soit implémentable de façon distribuée ou non.

Pratiquement, il est impossible de tenir compte de tous ces critères lors du développement d'un algorithme de localisation. Néanmoins, il peut être intéressant de les garder à l'esprit afin de pouvoir rendre notre méthode meilleure selon tel ou tel critère.

5. Algorithmes de localisation

Cette section est destinée aux algorithmes de localisation. Nous distinguons deux façons d'implémenter un algorithme de localisation selon leur organisation de calcul : les algorithms centralisés et les algorithmes distribués.

5.1 Algorithmes centralisés

Les algorithmes centralisés sont conçus pour fonctionner sur une machine centrale très puissante au niveau ressources. Les nœuds capteurs recueillent des informations (signal, voisins, distances, etc) de leur environnement et les transmettent à une station de base qui à son tour les analyse, calcule les positions et les transmet aux nœuds. Les algorithmes centralisés contournent le problème des ressources limitées des nœuds en acceptant des coûts de communications très élevés pour envoyer les informations à la machine centrale. Ces algorithms deviennent de plus en plus coûteux quand la taille du réseau augmente, car ça épuise les nœuds qui sont trop proches de la station de base qui subissent un très grand nombre de communications. En outre, les algorithmes centralisés exigent qu'une station de base puissante soit déployée parmi les nœuds, ce qui n'est pas toujours possible. Dans le cas où c'est possible, le problème de la mise à l'échelle peut être résolu en déployant plusieurs stations de bases.

Cependant, La centralisation permet à un algorithme d'être plus complexe, car les calculs se font sur la machine centrale et non pas par les nœuds eux mêmes. [11].

5.2 Algorithmes distribués

Dans le cas d'un algorithme distribué, tous les nœuds communiquent avec leurs voisins pour estimer les distances et échanger les informations de voisinage, afin de dériver leur position. Par conséquent, à la fin du processus de localisation, chaque nœud

doit connaître sa position ainsi que celles de ses voisins sans l'aide d'aucune unité centrale. Les algorithmes distribués, extrapolent généralement les positions des nœuds à partir de celles des ancres. Ainsi, ils localisent les nœuds directement dans le système de coordonnées global de ces ancres.

Comme le calcul des positions se fait par les nœuds eux mêmes, les algorithmes distribués ne sont pas complexes. Pour les réseaux à grande échelle, 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.

5.3 Comparaison

Une comparaison entre les algorithmes distribués et centralisés est présentée dans le tableau II.1. Cette comparaison montre les différentes caractéristiques de ces algorithmes en tenant compte des critères de localisation. [7]

Algorithmes	Centralisés	Distribués
Precision	Bonne à cause du calcul complexe.	Moyenne/faible.
Contraintes resources	Non, les calculs se font sur la machine puissante.	Oui, les nœuds font le calcul.
Complexité	Très complexes.	Non complexes.
Consommation	d'énergie Forte, grand nombre de communications.	Faible.
Passage à l'échelle	Non robustes	Robustes

Tableau II. 1: Comparaison entre algorithmes centralisés et distribués.

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 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 (Global Positioning System).

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.

6.2 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 resolution d'un système d'équations dont les distances sont les inconnues.

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 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. [12]

7. Les méthodes libres de mesure

Dans cette famille de méthodes, les capteurs cherchant à déterminer leurs positions s'appuient uniquement sur les positions des ancres. Aucune mesure de distance ou d'angle n'est utilisée. Par conséquent, ces méthodes ne peuvent fournir que des positions estimées aux capteurs. Les méthodes, pour ne citer qu'elles, sont des exemples de méthodes libres de mesure.

Il existe deux techniques courantes dans ce type de méthodes. La première consiste à définir des zones contenant les capteurs dont les centres de gravité correspondent à leurs positions estimées. Par exemple, dans les auteurs proposent le raisonnement suivant : soit n étant le nombre d'ancres dans le réseau, chaque ancre diffuse sa position. Lorsqu'un capteur obtient les positions des n ancres, il calcule tous les triangles possibles qu'il peut former avec ces positions et obtient alors un ensemble de triangles. Ensuite, pour chaque triangle, le capteur détermine s'il se situe à l'intérieur ou à l'extérieur. Après avoir déterminé son appartenance ou non à chacun des triangles, il en déduit une zone le contenant et calcule sa position estimée comme étant le centre de gravité de cette zone. En présence d'un pourcentage faible d'ancres dans le réseau, cette solution est intéressante. En revanche, si l'augmentation de ce pourcentage améliore la précision des positions, elle engendre dans le même temps des calculs importants pour les capteurs.

Dans, les auteurs définissent des zones en utilisant non pas des triangles mais des polygones.

Dans la seconde technique, chaque capteur estime les distances qui le séparent des ancres et applique la multilatération pour calculer sa position estimée. La méthode HTRefine utilise ce procédé. Ce schéma est également adopté par des méthodes basées mesures.

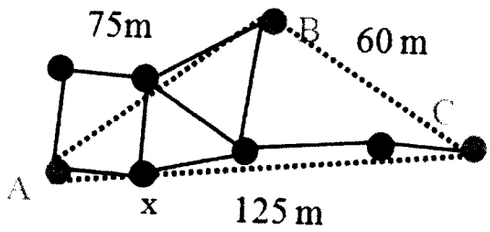

Figure II. 2 : DV-hop

Au commencement de la méthode HTRefine, toutes les ancres diffusent leurs positions.

Lorsqu'un capteur reçoit la position d'une ancre, il estime la distance qui le sépare d'elle. Pour ce faire, HTRefine utilise la technique d'estimation des distances DV-Hop. Dans cette technique, lors de l'inondation des positions des ancres, chaque capteur calcule le nombre de sauts minimum qui le sépare de chacune des ancres. Une deuxième vague d'inondation fournit suffisamment d'informations au capteur pour qu'il puisse convertir ces nombres de sauts en estimations de distances. La conversion consiste à multiplier le nombre de sauts séparant le capteur d'une ancre par une distance moyenne entre deux capteurs voisins. Lors de la première vague, lorsqu'une ancre A reçoit la position d'une ancre B, elle calcule la distance euclidienne qui les sépare et la divise par le nombre de sauts. Elle obtient ainsi une moyenne des distances des sauts entre elle et B et la communique aux capteurs. Lorsque A reçoit d'autres positions d'ancres, elle calibre à nouveau sa distance moyenne et diffuse cette mise à jour aux capteurs afin qu'ils puissent affiner leurs estimations de distance **DV-Hop**

La figure II.2 est une illustration de DV-Hop où l'ancre A estime la distance moyenne d'un saut. Les nœuds noirs représentent les ancres et les nœuds blancs les capteurs non localisés. Trois sauts séparent A et B alors que quatre séparent A de C. L'ancre A calcule les distances euclidiennes d AB = 75m et d AC = 125m. La distance moyenne d'un saut est donnée par la fraction (125+75)/ (3+4)= 28.75 m.

Le capteur X estimera les distances avec B et C comme suit : $d_{XB} = 2 * 28.57$

et $d_{XC} = 3 * 28.57$.

Pour obtenir leurs positions, les capteurs utilisent ensuite la multilatération. En reprenant l'exemple de la figure II.2, X obtiendra sa position en résolvant le système suivant :

$$\begin{cases} d^2_{AX} = (x_X - x_A)^2 + (y_X - y_A)^2 \\ d^2_{BX} = (x_X - x_B)^2 + (y_X - y_B)^2 \\ d^2_{CX} = (x_X - x_C)^2 + (y_X - y_C)^2 \end{cases}$$

Finalement, un processus de raffinement des positions est effectué. En effet, après avoir estimé leurs positions, les capteurs les diffusent à leurs voisins. En fonction de ces données et grâce aux relations de voisinage, les capteurs calculent à nouveau leurs positions qui se rapprochent de leurs positions réelles. Après un nombre d'itérations défini, les capteurs fixent leurs positions estimées. [12]

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 (DV-Hop).

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

Chapitre 3 :
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 [5]

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 [6].

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, intégré, modulaire, destiné aux réseaux de capteurs. Il respecte une architecture basée sur une association de composants, réduisant la taille du code nécessaire à sa mise en place afin de respecter les contraintes de mémoires qu'observent les réseaux de capteurs.

En effet, TinyOS est constitué de plusieurs modules disponibles pour les applications et offrant des fonctions de capture de mesures ou de communication. Il n'existe pas d'exécutable pour le noyau du système, il est construit au moment de la compilation de l'application en fonction des composants qu'elle utilise. Le langage de programmation associé, le nesC, qui est une extension du langage C, permet de déclarer les composants ainsi que les liens qui les unissent et de faire l'association code/composants.

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 [7].

2.2 Principes de TinyOS

Un **composant** est constitué d'au moins un **module** utilisant et fournissant des **interfaces**. S'il contient plusieurs modules les liens entres eux sont décrits par un fichier de **configuration**.

Une application complète est un composant contenant plusieurs modules liés eux dont un module **Main** qui permet de démarrer.

Le SE offre une centaine de composants que l'on peut utiliser pour écrire des applications. Quand le programme est généré par le compilateur, seuls les composants utilisés (y compris ceux du système) sont présents. **Main** est lui-même connecté à certains composants du système (comme l'ordonnanceur par exemple) qui seront donc chargés en mémoire puis lancés par **Main** On va décrire dans des fichiers portant le suffixe **.nc** les modules, les interfaces et les configurations. Une application comporte donc plusieurs de ces fichiers.

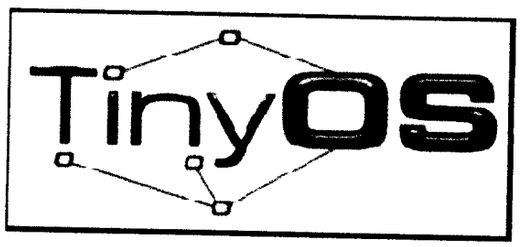

Figure III. 1: Symbole de system TinyOs

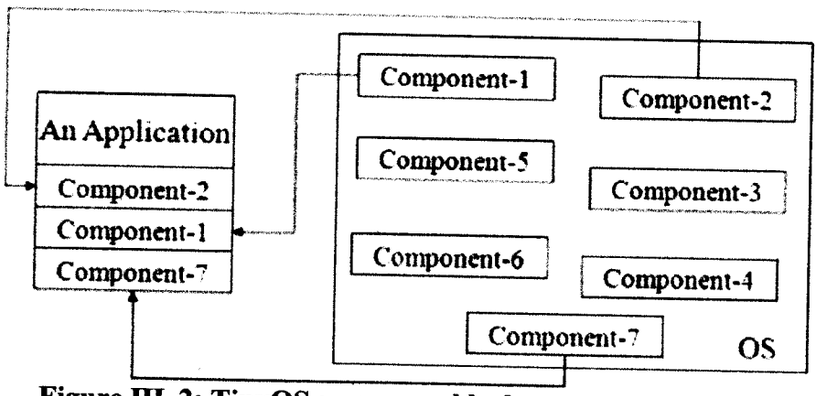

Figure III. 2: TinyOS : un ensemble de composants logiciels

2.3 Propriétés de la plateforme TinyOS

TinyOS est basé sur des propriétés qui font que ce système d'exploitation, s'adapte particulièrement bien aux systèmes à faible ressources :

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

B.Event-driven : Le fonctionnement d'un système basé sur TinyOS s'appuie sur la gestion des évènements se produisant. 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 évènementiel (event-driven) s'oppose au fonctionnement dit temporel (time-driven) où les actions du système sont gérées par une horloge donnée.

C.Non preemptif : 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 peut stopper l'exécution d'une tâche.

D.Pas de temps réel : Lorsqu'un système est dit « temps réel » celui ci gère des niveaux de priorité dans ses tâches, permettant de respecter des échéances données par son environnement. Dans le cas d'un système strict, aucune échéance ne tolère de dépassement contrairement à un système temps réel. TinyOS se situe au-delà de ce second type, car il n'est pas prévu pour avoir un fonctionnement temps réel.

E.Langage : TinyOS a été programmé en langage NesC que nous allons détailler Plus tard :

F.Consommation d'énergie : TinyOS a été conçu pour réduire au maximum la consommation en énergie du capteur. Ainsi, lorsqu'aucune tâche n'est pas active, il se met automatiquement en veille.[6]

2.4 Allocation de la mémoire

TinyOS a une empreinte mémoire très faible puisqu'il ne prend que 300 à 400 octets dans le cadre d'une distribution minimale. En plus de cela, il est nécessaire d'avoir 4 Ko de mémoire libre qui se répartissent entre les différents besoins suivant :

La pile : sert de mémoire temporaire, on y empile et dépile les variables locales.

Les variables globales : réservent un espace mémoire pour le stockage de valeurs pouvant être accessible depuis des applications différentes.

La mémoire libre : pour le reste du stockage temporaire.

Autres particularités, il n'y a pas d'allocation dynamique de mémoire et pas de mécanisme de protection de la mémoire, ce qui rend le système vulnérable à des crashs ou des corruptions de mémoire.

2.5 Fonctionnement

Les composants déclarent des tâches, des commandes ou des événements.

Les tâches sont des travaux de "longue durée". Lors de l'appel d'une tâche, cette dernière est ajoutée dans une file de type FIFO. Les tâches s'exécutent dans l'odre de la file et en entier car TinyOS ne dispose pas de mécanisme de préemption entre les tâches.

Les commandes sont des exécutions d'une fonctionnalité précise dans un autre composant

Les événements sont les équivalents logiciels aux interruptions matérielles, ils sont prioritaires par rapport aux tâches et peuvent donc les interrompre.

Lorsque la file d'attente est vide, cela signifie qu'aucune tâche n'est exécutée, et TinyOS met en veille le capteur, afin d'économiser la batterie.

2.6 Cibles possibles pour TinyOS

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

Mote, processeur, RAM et Flash : On appelle généralement Mote la carte physique utilisant TinyOS pour fonctionner. Celle-ci a pour cœur le bloc constitué du processeur et des mémoires RAM et Flash. Cet ensemble est à la base du calcul binaire et du stockage, à la fois temporaire pour les données et définitif pour le système Tinyos

Radio et antenne : TinyOS est prévu pour mettre en place des réseaux sans fils, les équipements étudiés sont donc généralement équipés d'une radio ainsi que d'une antenne afin de se connecter à la couche physique que constituent les émissions hertziennes.

LED, interface, capteur : TinyOS est prévu pour mettre en place des réseaux de capteurs, on retrouve donc des équipements bardés de différents types de détecteurs et autres entrées.

Batterie : Comme tout dispositif embarqué, ceux utilisant TinyOS sont pourvus d'une alimentation autonome telle qu'une batterie.

2.7 Ordonnancement

L'ordonnancement détermine l'ordre dans lequel les tâches sont exécutées sur un CPU. Dans les systèmes informatiques traditionnels, l'objectif d'un ordonnanceur est de minimiser la latence, pour optimiser l'utilisation de la bande passante et des ressources, et pour assurer l'équité. Le choix d'un algorithme d'ordonnancement adapté aux réseaux de capteurs dépend généralement de la nature de l'application. Pour les applications ayant des exigences temps réel, un algorithme d'ordonnancement en temps réel doit être utilisé. Pour d'autres applications, les algorithmes d'ordonnancement non temps réel sont suffisants. Les capteurs pour réseau sont utilisés dans des environnements à la fois en temps réel et non temps réel, Le système d'exploitation pour capteur en réseau doit fournir des algorithmes d'ordonnancement qui répondent aux exigences des applications. Par ailleurs, l'algorithme d'ordonnancement à un impact sur la gestion de la mémoire et l'efficacité de la consommation énergétique. Il existe deux catégories d'ordonnancement des taches :

- **l'ordonnancement en temps réel**

Ce mode de fonctionnement permet de répondre aux besoins d'application en temps réel pour la surveillance d'environnement de manière périodique

- **l'ordonnancement en temps partagé**

L'ordonnancement des tâches en mode événementiel, convient pour des évènements asynchrones

2.8 Tâches, événements et applications

TinyOS est basé sur la gestion de tâches et d'événements. Une tâche est un bloc d'instruction, un événement est l'équivalent logiciel d'une interruption matérielle et a priorité sur les tâches. Chaque tâche est activées ou interrompue en fonction de

l'apparition d'un ´événement et TinyOS n'étant pas préemptif les tâches ne peuvent pas s'interrompre entre elles, mais peuvent l'être par un événement.

Gestion des tâches

Chaque tâche active est mise en attente dans une file d'attente de type FIFO (First In First Out : première arrive première sortie), lorsque la file des tâches est vide le système se met en veille en attendant le prochain événement. Ce mécanisme de tâches a pour avantage d'empêcher une tâche d'en interrompre une autre, pouvant bloquer le système, mais il a aussi pour inconvénient de ne pas permettre une gestion en temps réel.

Pour les tâches de longue durée TinyOS possède un mécanisme permettant de fragmenter l'exécution d'une tâche nomme split-phase qui permet de ne pas bloquer le système. Ce mécanisme est utilise dans l'initialisation de composants qui demandent du temps au démarrage, comme la radio par exemple.

Les Evénements

Lorsqu'une interruption matérielle a lieu, l'événement correspondant reçoit un signal et prend la main de manière asynchrone, c'est à dire qu'il n'attend pas la fin de la tache courante pour s'exécuter. Des événements peuvent être Signalés ne correspondant pas directement a une interruption matérielle. Il existe également des événements synchronises : ils sont mis en attente dans la liste des tâches, avec une priorité supérieure aux tâches en attente mais n'interrompent pas la tâche courante (cas de certains Timers).

Les applications basées sur TinyOS sont formées de composants réutilisables et portables, (comme les Timers, les convertisseurs de signal ou la radio) qui sont relies entre eux. Ces composants peuvent être directement lies au matériel (composant gérant les LED par exemple) ou un regroupement de plusieurs composants de bas niveau (composants gérant les envois de données par la radio). Les composants sont implémentés en utilisant les tâches, les événements et des commandes qui permettent de faire appel aux fonctions n'alitent d'autres composants auxquels ils sont lies.

Exemple d'application

Une application devant mesurer la température d'une pièce régulièrement et transmettre les données à un ordinateur utilisera plusieurs composants :

– un composant de mesure de température

– un composant qui se chargera de l'envoi des données par le port USB

– un composant de mesure du temps

– un composant de gestion des LED pour afficher la fréquence des mesures

2.9 Package TinyOS

TinyOS est prévu pour fonctionner sur une multitude de plateformes, disponibles dès l'installation, TinyOS peut être installé à partir d'un environnement Windows (2000 et XP) ou bien GNU/Linux (Red Hat essentiellement, mais d'autres distributions sont également possibles). Deux principales versions de TinyOS sont disponibles : la version stable (v. 1.1.0) et la version actuellement en cours de tests (v.1.1.15); la première présente moins de risques mais est nettement moins récente.

Windows : un guide propose l'installation de tous les principaux outils nécessaires au bon fonctionnement du système, notamment Cygwin (couche d'émulation de l'API Linux) qui permet d'avoir une interface Unix sous Windows. Le JDK Java 1.4 de Sun est nécessaire an d'exécuter la procédure d'installation.

GNU/Linux : des packages RPM sont proposés au téléchargement, un guide explique la marche à suivre. Les distributions Linux ayant un autre gestionnaire de paquet peuvent utiliser un programme (comme Alien) pour installer les packages, ou compiler directement à partir des sources. Le JDK de IBM est nécessaire.

Par la suite, des packages supplémentaires peuvent être ajoutés en passant par le site Source Forge, qui met à disposition le code open source de TinyOS et d'un ensemble de programmes dédiés.

3. Description du langage NesC

Il doit exister un environnement de développement logiciel afin d'importer des applications sur les capteurs. Le langage utilisé pour le développement dans TinyOS est le nesC.

Figure III. 3:symbole de langage Nesc

3.1 Présentation

Le langage NesC a une architecture basée sur des composants. Cela permet de réduire la taille mémoire du système et de ses applications, chaque composant correspondant à un élément matériel du type LED ou timer par exemple. Les composants peuvent être réutilisés dans n'importe quelle application. L'implémentation

des composants, dans le but d'élaborer des applications, s'effectue en déclarant des tâches, des commandes ou des 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 pour y être exécute.

3.2 Concepts principaux dans NesC

Application : un ou plusieurs composants reliés ensemble pour former un exécutable

Composant : un élément de base pour former une application nesC. Il existe deux types de composants : modules et configurations

- ✓ **Module** : composant qui implémente une ou plusieurs interfaces
- ✓ **Configuration** : composant qui relie d'autres composants ensemble

Interface : définie d'une manière abstraite les interactions entre deux composants

3.3 Développement

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

```
interface Timer {
command result_t  start(char type, unit32_t  interval):
command result_t stop();
event result_t fired();
}
```

TinyOS n'autorise pas les pointeurs de fonctions. Afin de néanmoins proposer un mécanisme alternatif, NesC utilise des interfaces paramétrées. Celles-ci permettent à l'utilisateur de créer un ensemble d'interfaces identiques et d'en sélectionner une seule à appeler grâce à un identifiant.

```
interface SendMsg[unit8 t id]
```

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

```
configuration NonModule {}
implementation {
//liste des modules et configurations utilises, ex :
components Main,Module1.....ModuleN,Config1,.....,ConfigiM ;

//descriptifs des liaisons
//Interface fournie <- Interface requise ou
//Interface requise -> interface fournie, ex :
Main.StdControl -> Module1.StdControl ;
}
```

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

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

```
module NomModuleM {
provides {
//liste des interfaces fournies, ex :
interface NominterfaceFourniel ;
}
uses {
//liste des interfaces requises, ex :
interface NomInterfaceRequisel ;
}
}
implimentation {
//déclarations des variables, ex :
int state ;
//implementations des fonctions décrites par les interfaces fournies :
}
```

3.4 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 [6].

3.5 Compiler et exécuter une application nesC

La première étape de ce processus consiste à compiler les fichiers nécessaires à l'application et au système d'exploitation. Celle-ci est réalisée via le compilateur NesC fourni par TinyOS. Son rôle est premièrement de transformer les fichiers NesC en fichier C et deuxièmement d'y intégrer les fichiers du noyau de TinyOS. Ce qui permet d'obtenir un fichier source C unique. Une fois cette étape accomplie, il ne reste alors qu'à utiliser un compilateur C traditionnel qui va utiliser le fichier précédemment créé afin de générer une application exécutable. Celle-ci sera donc constituée par la «fusion» du système d'exploitation et du code applicatif. Ces différentes phases peuvent être synthétisées comme l'illustre la figure III.4 Pour effectuer une compilation, les fichiers doivent se situer dans le même répertoire contenant aussi un makefile.

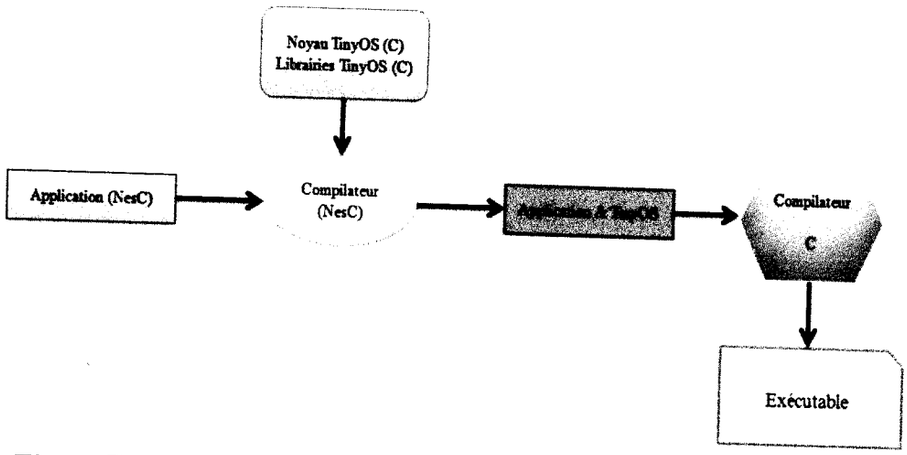

Figure III. 4: Etapes De Compilation D'une Application Sous Tinyos.

4. Conclusion

Le système TinyOS est un système embarqué, dédié spécialement pour les réseaux de capteurs, qui présente un environnement de calcul particulier soumit a plusieurs contraintes tel que l'énergie, l'espace mémoire, etc.

TinyOS permet le développement d'application simplifiée par la mise en relation de composants cibles, tout en offrant des primitives ainsi que des bibliothèques réseaux qui le rondent complet. Il est classé parmi les systèmes nom temps réel, nom préemptif, événementiel.

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, présenter la plate-forme logicielle que nous avons utilisée pour les simulations TOSSIM, ensuite, nous allons présenter les contextes de simulation et les résultats pour le protocole de localisation étudié avec discussion de résultat.

2. Simulateur de réseaux de capteur

2.1 Définition

Afin d'évaluer les protocoles de localisation, TOSSIM a été utilisé. 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.

2.2 Description

A.Tinyviz

TinyViz est une interface graphique Java. Elle permet de donner un aperçu des capteurs à tout instant ainsi que des divers messages qu'ils émettent. Elle détermine un délai entre chaque itération des capteurs afin de permettre une analyse pas à pas du bon déroulement des actions en activant différents modes comme Radio, CPU, etc. **[15]**

b. PowerTOSSIM

Le simulateur TOSSIM n'a pas la capacité de vérifier le taux d'énergie dissipée pendant l'exécution des applications. Cependant, le besoin de vérifier la consommation énergétique dans un RCSF a un intérêt primordial. L'université de Harvard a conçu le simulateur PowerTOSSIM qui surmonte

ce problème. Ce nouveau simulateur est intégré dans TOSSIM. Il permet de générer un fichier de l'extension .trace qui enregistre les détails de la simulation comme l'énergie consommée dans le réseau. **[16]**

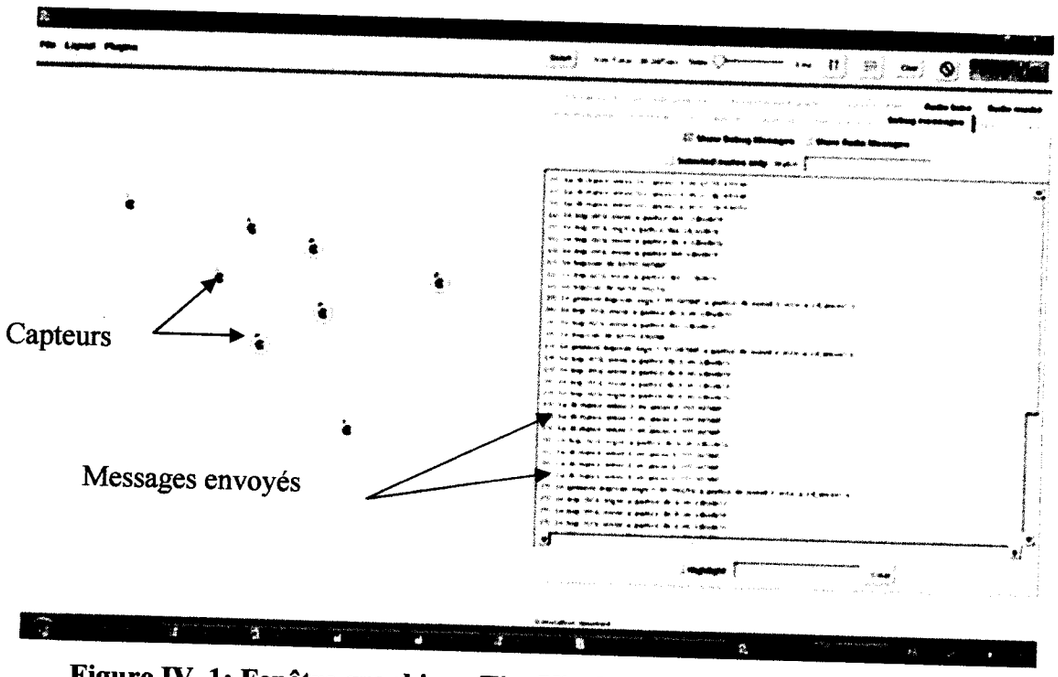

Figure IV. 1: Fenêtre graphique TinyViz de notre application

3. Estimation des coordonnées

DVHOP est un algorithme de localisation. Son but est de permettre aux capteurs de trouver leurs positions à l'aide des positions connues de 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 noeuds, et deuxièmement la multilatiration.

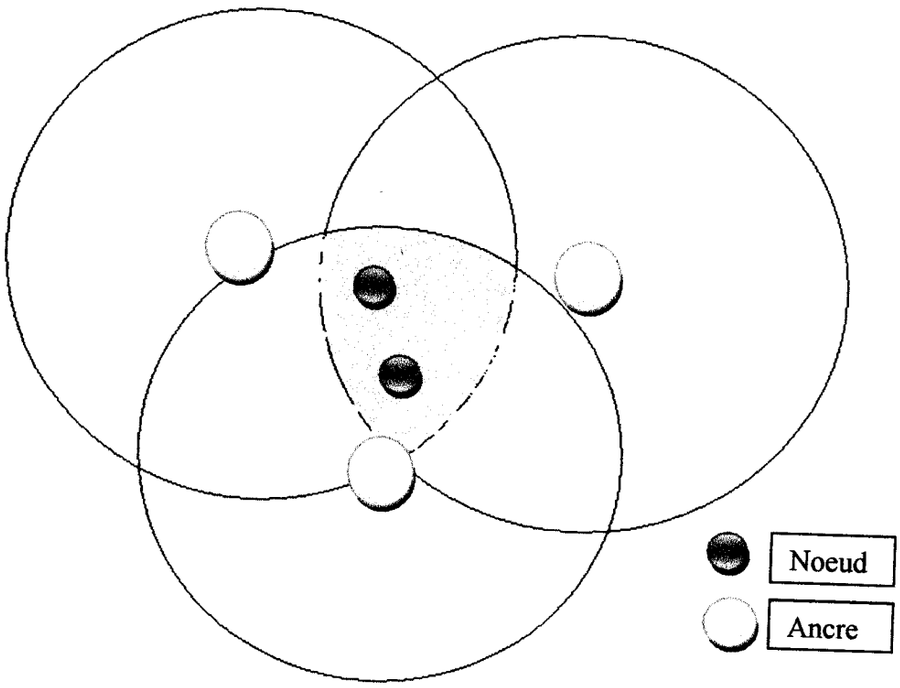

Figure IV. 2: Figure représentant le modèle réel.

La Figure IV.2 illustre un modèle de réseaux des capteurs sans fil en utilisant la méthode de multilatiration pour estimer les positions des noeuds, les boules violées représentent les ancres (leur position est connue) et les oranges représentent les nœuds (leur position est inconnue).

Les cercles représentent les zones de couvertures des ancres, pour que l'erreur d'estimation soit minimale, il faut placer les nœuds dans la zone d'intersection des cercles (zone bleu).

A- Estimation 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.

B- Multilatération

La Multilatération est une méthode relativement simple et intuitive, en utilisant plus que trois points de référence (ancres). La position d'un nœud est calculée en connaissant les positions d'un certains ancres et les distances estimées de ces nœuds aux différents ancres. Nous formons le système suivant :

Soit une cible a dont on veut trouver la position Xa, et soit m ancres i dont nous connaissons les positions x_i, $1 < i < m$.

Nous supposons que nous connaissons aussi une estimation des distances D_{ai} avec $1 < i < m$ 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 :

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

$$B=\begin{cases} x^2_{11}-x^2_{m1}+\ldots+x^2_{1p}-x^2_{mp}+d^2_{ma}-d^2_{1a} \\ \quad\ldots \\ \ldots \\ x^2_{(m-1)1}-x^2_{m1}+\ldots+x^2_{(m-1)p}-x^2_{mp}+d^2_{ma}-d^2_{(m-1)a} \end{cases}$$

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

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

Où $x_a=\begin{pmatrix} x_{a1} \\ x_{a2} \\ \\ X_{ap} \end{pmatrix}$ 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 Delphi.

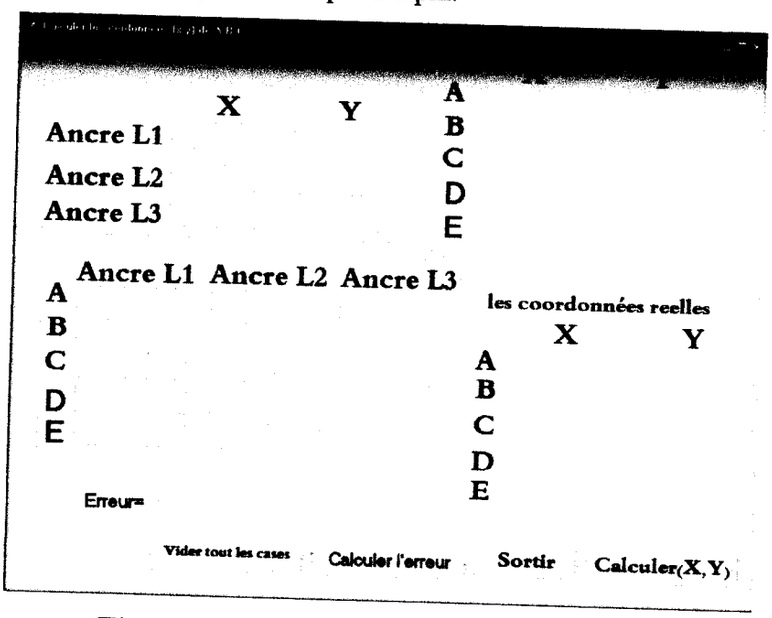

Figure IV. 3: Notre application de trilateration

4. Objectifs et métriques de la simulation

Le but général de ce chapitre est d'implémenter et d'analyser DVHOP (détaillé en chapitre 2) un algorithme de localisation sur les réseaux de capteurs sans fil. Nous avons considéré deux métriques d'évaluation, à savoir, la précision de la localisation et la consommation d'énergie.

- **Evaluer la précision de la localisation**

La précision de la localisation est une métrique très importante dans le processus de localisation. Puisque dans celui-ci nous n'avons pas la vraie localisation mais nous allons estimer une localisation. Donc plus cette estimation se rapproche de la localisation réelle, plus nous avons une méthode de localisation plus précise.

Nous allons évaluer la précision de la localisation en fonction du nombre de nœuds ordinaires dans le réseau et en fonction du nombre des ancres dans le réseau.

Cette formule représente la précision de localisation de l'algorithme lors d'estimation des distances entre les noeuds et les ancres. Nous supposons que nous connaissons également les vraies positions des noeuds. La précision d'un nœud N_i est calculée comme suivant:

$$erreur\ Ni = \frac{\sqrt{(Xi-Xr)^2+(Yi-Yr)^2}}{Xr+Yr}$$

$$\text{Erreur totale} = \sum_{i=0}^{N} \frac{erreur\ Ni}{N} * 100\% \qquad [13]$$

Tel que :

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

Xi, Yi : les positions d'un nœud trouvées par DV-hop.

N : le nombre des nœuds ordinaires.

Erreur Ni : l'erreur d'un nœud.

Erreur totale: la Moyenne d'erreur de tous les nœuds de réseau.

- **La consommation d'énergie**

L'énergie est une ressource rare et critique dans un réseau de capteurs sans fil. Il est donc très important aussi d'évaluer la consommation d'énergie de l'algorithme de localisation pour faire un compromis entre précision d'estimation et consommation d'énergie.

Cette métrique est évaluée en fonction de la taille du réseau (changement du nombre de nœuds dans le réseau).

5. Version amélioré de DV-hop (Improved DV-hop)

Dans la version basique de DV-Hop, le nœud voulant estimer sa position, prend en compte le premier hop size reçu d'un des ancres et ignore les autres.

Dans la version proposée par [14] et nommée IDV-hop, les auteurs ont gardé le même principe que la version basique de DV-hop, cependant, un recalcule des distances entre les nœuds et les ancres est effectué, si un nœud reçoit un hop inférieur à l'ancien hop du même ancres.

6. Résultat de la simulation

6.1 La métrique précision de localisation

Cette évaluation montre l'efficacité de la phase de calcul de distance entre chaque nœud avec toutes les autres ancres.

Le but est de calculer la précision de localisation d'un nœud par rapport à sa position exacte.

La précision de la localisation est évaluée en fonction de deux paramètres: le nombre de nœuds ordinaires et le nombre des ancres.

On a utilisé deux versions d'algorithme : la première est la version basique **DV-hop** et la deuxième est la version améliorée **IDV-hop**.

a. Erreur d'estimation pour la version basique

Scénario 1 :

En fonction de la densité des nœuds ordinaires, on a obtenu les résultats suivants avec une topologie constituée en 3 ancres et de 5, 10 puis 20 nœuds ordinaires.

Nombre de nœuds ordinaires	5	10	20
Erreur (%)	54.08	50.26	34.57

Tableau IV. 1: précision de l'algorithme dvhop en foction de densité des nœuds

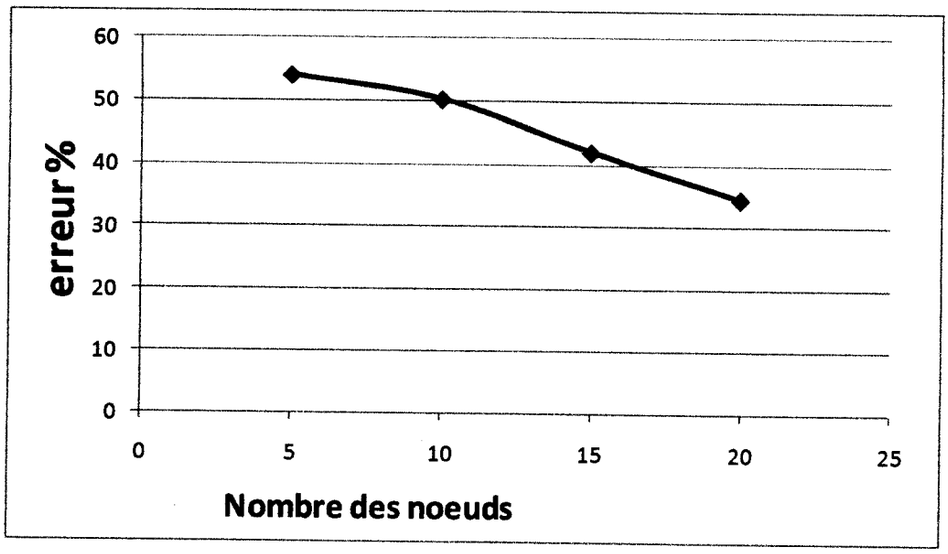

Figure IV. 4: Estimation d'erreur de localisation en fonction de nombre des nœuds

Observation et discussion : On remarque clairement dans le graphe de la Figure IV.4, que l'erreur diminue lorsque le nombre des nœuds ordinaires augmente. Ceci s'explique par le fait que le nombre de sauts augmente donc le hop-size va diminuer.

Scénario2 :

En fonction de la densité des ancres, on a obtenu les résutats suivants avec une topologie constituée en 5 nœuds ordinaires et 3 ancres puis 4, 5, 6 et 7 ancres.

Nombre d'ancres	3	4	5	6	7
Erreur (%)	54.08	40.1	36.71	30	24.73

Tableau IV. 2: Precision de l'algorithme Dvhop en Fonction de densité d'ancres

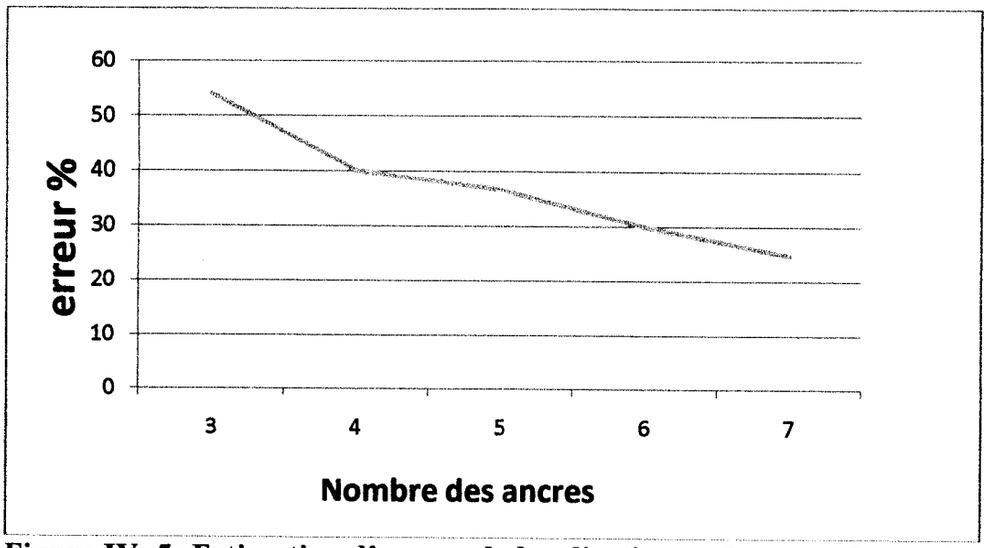

Figure IV. 5: Estimation d'erreur de localisation en fonction de nombre des ancres

Observation et discussion :

On remarque clairement dans le graphe de la Figure IV.5, que l'erreur diminue lorsque le nombre des ancres augmente, car dans la phase de multilateration, la zone d'intersection des cercles qui représentent les zones de couvertures, va diminuer et les coordonnées X et Y sont incluses dans cette zone, donc l'erreur va diminuer.

Sauf que pour des raisons économiques, un nœud ancre est généralement plus chèr s'il possède un GPS, et donc le coût total du réseau peut augmenter si le nombre de nœuds ancre augmente.

b. Erreur d'estimation pour la version améliorée

Scénario 1 :

En fonction de la densité des nœuds ordinaires, on a obtenu les résutats suivants avec une topologie constituée de 3 ancres et 5 puis 10 puis 20 nœuds ordinaires

Nombre de nœuds ordinaires	5	10	20
Erreur (%)	30.22	27.31	23.76

Tableau IV. 3 : Précision de l'algorithme IDV-hop en fonction de densité de nœuds

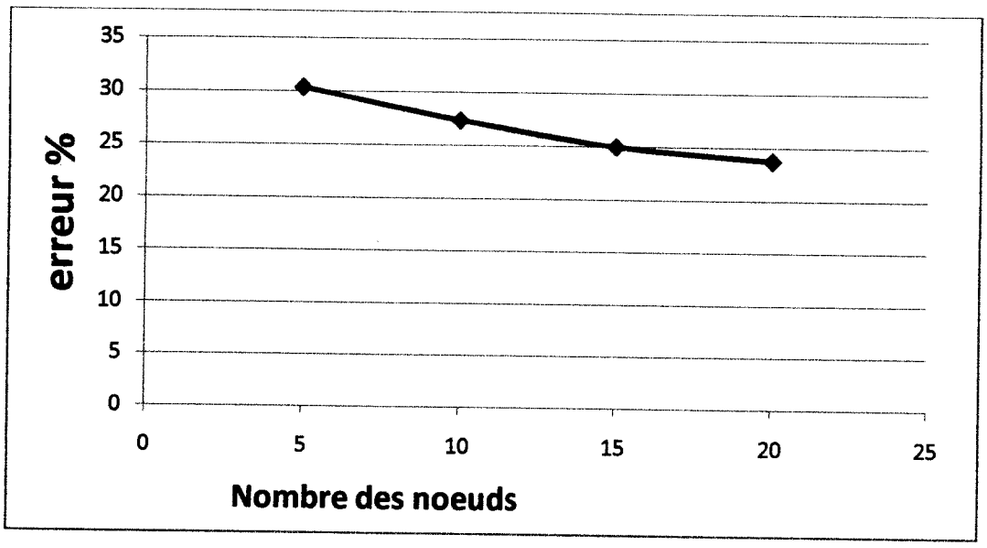

Figure IV. 6 : Estimation d'erreur de localisation en fonction de nombre de nœuds.

Observation et discussion : On remarque clairement dans le graphe de la Figure IV.6, que l'errer diminue lorsque le nombre des nœuds ordinaires augmente. Ceci s'explique par le fait que le nombre de sauts augmente donc le hopsize va diminuer.

Scénario 2 :

En fonction de la densité des ancres, on a obtenu les résutats suivants avec une topologie constituée en 5 nœuds et 3 ancres puis 4, 5, 6 et 7 ancres.

Nombre d'ancres	3	4	5	6	7
Erreur (%)	30.22	27.33	24.45	23.25	22.15

Tableau IV. 4: Precision de l'algorithme IDV-hop en foction de la densite d'ancres

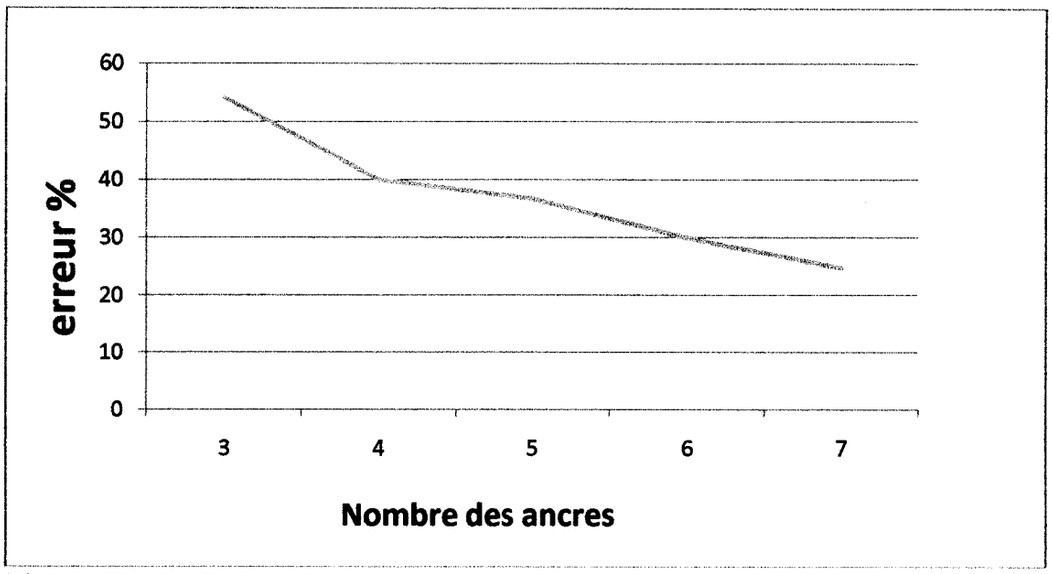

Figure IV. 7 : Estimation d'erreur de localisation en fonction de nombre des ancres.

Observation et discussion : On remarque clairement dans le graphe de la Figure IV.7, que l'erreur diminue lorsque le nombre des ancres augmente, car dans la phase de multilateration la zone d'intersection des cercles qui représentent les zones de couvertures va diminuer et les coordonnées X et Y sont inclues dans cette zone, donc l'erreur va diminuer.

C. Comparaison entre DV-hop et IDV-hop en fonction d'erreur d'estimation

En fonction de la densité des ancres, on a obtenu les résutats suivants avec une topologie constituée en 3 ancres et 5 puis 10 puis 15 puis 20 nœuds ordinaires en utilisant Matlab, on a obtenu les courbes suivantes :

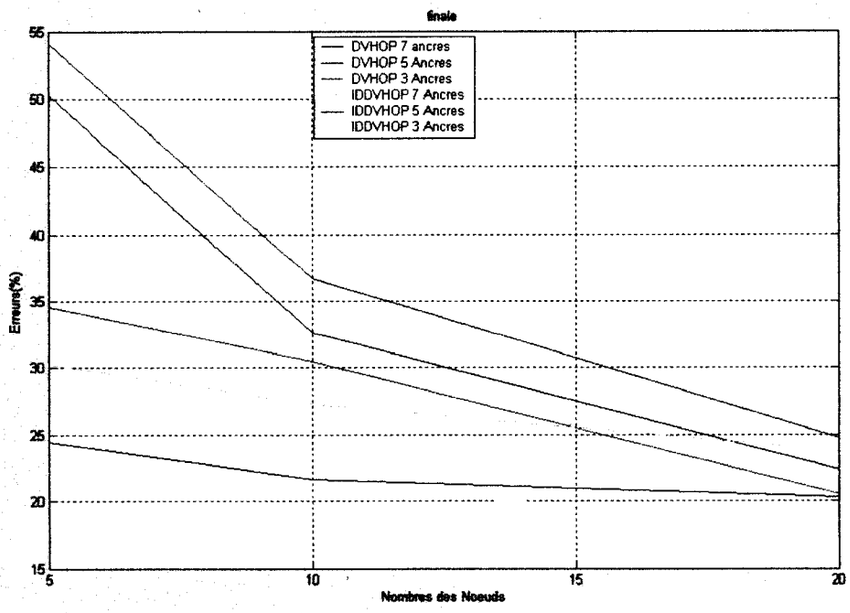

Figure IV. 8 : Comparaison entre DV-hop et IDV-hop en fonction d'erreur d'estimation

Observation et discussion :

On remarque clairement dans le graphe de la Figure IV.8, que l'erreur diminue lorsque le nombre des ancres ou le nombre de nœuds ordinaires augmente. On remarque aussi que l'erreur dans le IDV-hop est moins que le DV-hop soit en fonction de nombre de nœuds ordinaire ou de nombre d'ancre.

Nos résultats de simulation démontrent que la version améliorée IDV-Hop donne de meilleurs résultats en termes de précision que la version basique de DV-Hop.

6.2. La métrique de consommation d'énergie

Pour évaluer cette métrique, on a utilisé pour chaque simulation les paramètres de configuration propres aux réseaux de capteurs du tableau IV.5:

IDV-hop	DV-hop	Nombre d'ancres	Nombre de nœuds
11239.57	11141.53	3	5
12185.14	12091.54	4	5
13001.79	12869.22	5	5
13773.52	13434.02	6	5
13957.14	13816.4	7	5

Tableau IV. 5: Compraison entre DV-hop et IDV-hop en fonction de consomation d'energie

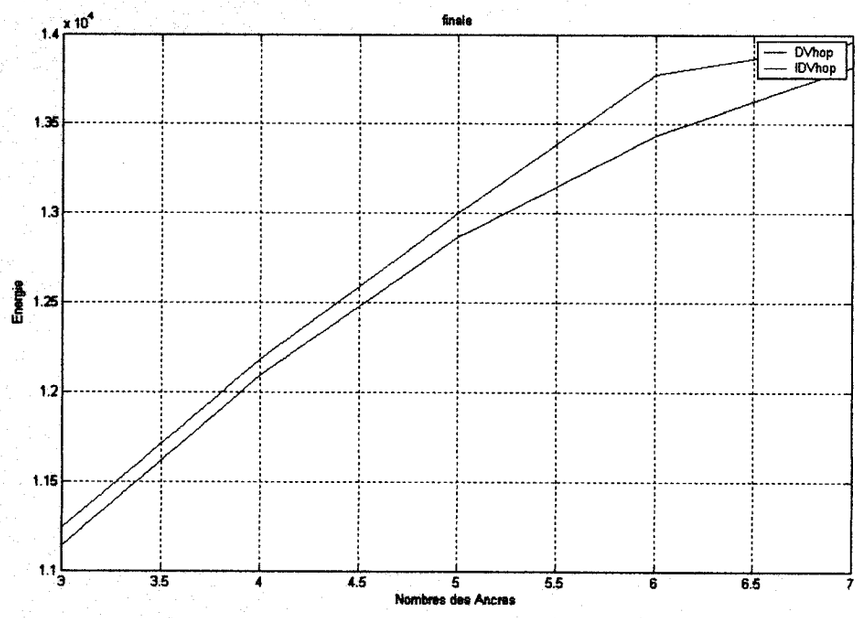

Figure IV. 9: Compraison entre DV-hop et IDV-hop en fonction de consomation d'energie.

Observation et discussion :

A partir du graphe de la Figure IV.9, on remarque que l'énergie totale consommée augmente avec l'augmentation de nombres de nœuds dans le réseau. L'augmentation d'énergie totale consommée est expliquée par l'augmentation de nombre de nœuds en échangeant de différents messages entre eux. On constate aussi que l'énergie consommée dans *IDV-hop* est légèrement supérieure à celle du *DV-hop*, cela est dû au fait que dans *IDV-hop*, un nœud fait plus d'émission /réception des messages par rapport au DV-hop.

Nous pouvons constater que IDV-Hop donne de meilleures estimations de positions par rapport à DV-HOP mais au pris d'une légère consommation d'énergie due au surcoût additionnel en communication.

7. Conclusion

Dans ce chapitre, nous avons présenté l'environnement de simulation avec lequel nous avons travaillé : simulateur TOSSIM et ses caractéristiques. Ensuite, nous avons décrit en détail le protocole de localisation DV-hop ainsi que sa version améliorée *IDV-hop* où le but a été d'étudier et d'évaluer leurs performances sous deux métriques à avoir : L'erreur de localisation ainsi que l'énergie totale consommée par ses derniers.

Par ailleurs, nous avons constaté que l'algorithme de localisation IDV-hop est plus efficace que DV-hop en termes de précision de localisation mais il consomme plus d'énergie.

Conclusion générale

Les réseaux de capteurs sans fil (RCSF) sont une nouvelle technologie qui a surgit après les grands progrès technologiques concernant le développement des capteurs intelligents capables de detecter des évenements et de préciser sa position, c'est-à-dire la localisation. Plusieurs recherches qui ont été faites pour la conception de l'algorithme tiennent compte de cette contrainte et en plus 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'étude. Dans ce projet, nous avons donné un aperçu sur les réseaux de capteurs et présenté certaines de leurs applications. Nous avons aussi exposé les différents défis qui doivent surmonter la conception des protocoles de communication dans ces réseaux. Nous nous sommes intéressés aux protocoles liés à la localisation, et en particulier à l'algorithme DV-hop et sa version améliorée IDV-hop.

Dans ce projet nous avons étudié les deux algorithmes de localisation dans les RCSFs (*DV-hop, IDV-hop*) dans le but de faire une comparaison entre eux à l'aide du simulateur *TOSSIM*. Après avoir effectué plusieurs simulations et analysé les résultats obtenus, nous avons constaté que IDV-hop est plus précis que le DV-hop, au pris d'un surcoût additionnel.

Tout de même, DV-Hop est un algorithme range free, qui se base uniquement sur la connectivité des nœuds capteurs. La précision des algorithmes range-free est souvent inférieure aux algorithmes range-based, qui eux utilisent se basent sur d'autres mesures. Comme perspective, il serait très intéressant de joindre les avantages des algorithmes range-based en termes de précision aux algorithmes range-free. Par exemple rajouter le RSSI à l'algorithme DV-hop afin de minimiser l'erreur de précision tout en gardant sa simplicité. Une bonne hybridation permettra d'avoir un algorithme de géolocalisation ayant comme avantage une meilleure précision, à moindre coût.

Références :

[1] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci « A Survey on Sensor Networks ».Georgia Institute of Technology. Pages 102-114,IEEE Communications Magazine. August 2002

[2] Benahmed Khelifa. « La sécurité dans les réseaux de capteurs sans fil». Centre universitaire de Béchar, Institut des sciences exactes BP: 417-08000 Béchar 2005.

[3] Mokhtar Aboelaze, Fadi Aloul. « Current and Future Trends in Sensor Networks:A Survey». IEEE 2005

[4] Mounir Achir. « Technologies basse consommation pour les réseaux Ad Hoc. Thèse pour obtenir le grade de Docteur de l'INPG ». Institut National Polytechnique de Grenoble. juillet 2005.

[5] sahraoui Belkheyr « La Géo-localisation dans les Réseaux de Capteurs sans Fil » Mémoire de fin d'études, Université Abou Bakr Belkaid– Tlemcen, 2011

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

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

[8]: KAREL Heurtefeux, FABRICE Valois. « Localisation collaborative pour reseaux capteurs » . Lyon (France).

[9]: ARONDEL Olivier, PONPARDIN Thomas. « Systeme de positionnement Galileo/Glonass ». Ecole superieur d'ingenieurs 01/09.

[10]: ROCH Jonas. « Le Global Positioning System(GSM) : structure et fonctionnement». Travail de maturité 2009.

[11]: MAKHOUL Abdallah. Réseaux de capteurs : « localisation couverture et fusion de données». Thèse Doctorat. Université de Franche-comté :14-11-2008.

[12]: SAAD Clement. Quelques contributions dans les réseaux de capteurs sans fil : Localisation et Routage. Thèse Doctorat École Doctorale 166 « I2S Mathématiques et Informatique » Laboratoire d'Informatique (EA 4128). présentée à l'Université d'Avignon et des Pays de Vaucluse. 10-07-2008.

[13] Haiyang Zhangm et Shuang Tian, « A Selective Anchor Node Localization Algorithm for Wireless Sensor Networks»,septembre 2007,pages:58-62

[14]Nan Jiang, Xiao Xiang, Chen Huan « An Iterative Boundary Node Localization Algorithm based on DV-hop Scheme in WSN».

Journal of Convergence Information Technology, Volume6, Number 7, July 2011.

[15] H. Alatrista, J. Mathieu, K. Gouaïch S. Aliaga, « Implémentation de protocoles sur une plateforme de réseaux de capteurs sans fil », TER master 1 informatique,Université de Montpellier II, 29 Avril 2008.

[16] Borrong Chen, Geoff Werner Allen, Mark Hempstead, Matt Welsh, Victor Shnayder, « Simulating the Power Consumption of LargeScale Sensor Network Applications», Proceedings of the 2nd international conference on Embedded networked sensor systems, Pages: 188 – 200, Harvard University, 2004.

[17] Abdelraouf Ouadjaout, « La Sécurité et la Fiabilité du Routage dans les Réseaux de Capteurs Sans Fil», Mémoire de magistère, USTHB, Algérie, 2006.

[18] N. Bulusu, J. Heidemann, and D. Estrin. « GPS-less low cost outdoor localization for very small devices». Technical report 00-729, Computer science department, University of Southern California, Apr. 2000.

Annexe1

Code source de notre application :

```
//Développeurs Rais Hichem & Mekidiche Mohamed
//cet algorithme traite 3 ancres (les ancres sont 0 et 1 et
2) et n nœuds tq n>=1
//lancer le shell et taper les cammandes suivantes
//    cd /opt/tinyos-1.x/apps/dvhop3_n    pour accéder au
répertoire qui contient notre application
//make pc    pour compiler l'application
//export DBG=usr1     pour afficher notre message
//export PATH="$TOSROOT/tools/java/net/tinyos/sim:$PATH"
//TinyViz -run build/pc/main.exe 8    pour lancer la
simulation avec 8 noeuds
//cliquer sur le menu "Layout" puis "file load"  et
sélectionner un fichier qui prend l'extention ".mps" (c'est
une topologie déja créée)
//pour un capteur qui envoie des messages seulement aux
capteurs qui ont dans sa zone de couverture: cliquer sur le
menu plugins et sélectionner "radio Model" puis décocher "out
edges" puis cliquer sur "update"
//cocher debug messages dans le menu plugins pour visualiser
les messages
includes DvhopMsg;//appler le fichier DvhopMsg.h qui contient
la structure de notre message

module DVhopM
{
  provide interface StdControl;//interface fournie
 //interface utilisée
 uses {
    interface Timer;
    interface Leds;
    interface SendMsg as envoiMsg;
    interface ReceiveMsg as RecoieMsg;
  }
}
implémentation
{
float premier_hop_arriver;
  DVhopMsg *  pemission;//pointeur sur le message emis
  DVhopMsg * poi;//pointeur sur le message reçu
  TOS_Msg msg;//message emis
  int hop[3][2];//tableau utilisé par chaque ancre pour
stoquer les hop  d'autres ancres
 bool ancre_recu[3][2];//tableau des boolean chaque ancre
reçoit les hop d'autre ancres
DVhopMsg * p;//pointeur sur le message emis aprés la
réception d'un message
bool noeud_reçu_avant[3];//par exemple si un noeud reçoit un
message d'ancre 1  neud_reçu_avant[1] devient true(avant de
calculer le hopsize)
```

```
bool affiche1,v,vv,vvv,r;//variable utilser pour afficher un
message une seule fois
float tab[3][2];//tableau pour stoquer les x et les y  des
ancres
 float hopsize[3];//tableau pour stoquer les hopsizes
int  hop_noeud[3];//tableau utiliser par les noeuds pour
stoquer les hop des ancres
float L1L2,L0L2,L0L1;//les distances entre les ancres
int i,j,k;
bool neud_recu_apres[3];//par exemple si une noeud recoi un
message d'ancre 1  neud_recu_apres[1] devient true(apres de
calculer le hopsize)
  command result_t StdControl.init() {
// initialisation des noeuds
    pemission =( DVhopMsg* ) msg.data;// pointer sur le
message emis qui est de type TOS_Msg sur le champs data
    switch(TOS_LOCAL_ADDRESS) {

       case 0 : //si l'adreese du capteur=0
(TOS_LOCAL_ADDRESS=0)
              pemission->id_ancre = 0;
              pemission->hop =0;
              pemission->x = 81.75;
              pemission->y = 41;
              pemission->id_node=TOS_LOCAL_ADDRESS;
                    pemission->hopsize= 0;
               break;

        case 1 :    //si l'adreese du capteur=1
(TOS_LOCAL_ADDRESS=1)
              pemission->id_ancre = 1;
              pemission->hop =0;
              pemission->x = 16.54;
              pemission->y =  33.01;
              pemission->id_node=TOS_LOCAL_ADDRESS;
                    pemission->hopsize= 0;
             break;
case 2 : //si l'adreese du capteur=2 (TOS_LOCAL_ADDRESS=2)
              pemission->id_ancre =2;
              pemission->hop =0;
              pemission->x = 47.76;
              pemission->y = 66.02;
              pemission->id_node=TOS_LOCAL_ADDRESS;
              pemission->hopsize= 0;
               break;

default: pemission->id_node=TOS_LOCAL_ADDRESS;
pemission->id_ancre =100;// les autres noeuds prends la valeur
100 par exemple
pemission->hopsize= 0;

    call Leds.init();
    return SUCCESS;
  }

}
```

```
command result_t StdControl.start() {


v=TRUE;vv=TRUE;vvv=TRUE;
//stocker les x et les y des ancres dans un tabeau
 tab[0][0]=81.75;
 tab[0][1]=41;

 tab[1][0]=16.54;
 tab[1][1]= 33.01;

 tab[2][0]=47.76;
 tab[2][1]=66.02;

//le calcule des distances entres les ancres
L0L1= sqrt(pow((tab[0][0]-tab[1][0]),2)+ pow((tab[0][1]-
tab[1][1]),2));
L0L2= sqrt(pow((tab[0][0]-tab[2][0]),2)+ pow((tab[0][1]-
tab[2][1]),2));
L1L2= sqrt(pow((tab[1][0]-tab[2][0]),2)+ pow((tab[0][1]-
tab[2][1]),2));

if(TOS_LOCAL_ADDRESS==0){
dbg(DBG_USR1, "La distance entre les ancres 0 et 1=
%f\n",L0L1);
dbg(DBG_USR1, "La distance entre les ancres 0 et 2=
%f\n",L0L2);
dbg(DBG_USR1, "La distance entre les ancres 1 et 2=
%f\n",L1L2);
                        }
for(i=0;i<=2;i++)
for(j=0;j<2;j++)
 ancre_recu[i][j]=FALSE;

for(i=0;i<=2;i++){
neud_recu_avant[i]=TRUE;
neud_recu_apres[i]=FALSE;
                  }
affiche1=TRUE;
k=0;r=FALSE;
premier_hop_arriver=0;
       call Timer.start(TIMER_REPEAT, 2000);// envoyer un
message apres chaque 2 secondes

     return SUCCESS;
  }

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

```
event result_t envoiMsg.sendDone(TOS_MsgPtr sent, result_t
success) {
   call Leds.redOff();
    return SUCCESS;
  }

  event result_t Timer.fired() {
    atomic {
      if( k<15){
      k=k+1;
      //dbg(DBG_USR1, "bien envoyer\n");
    if (call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg),
&msg)== SUCCESS)
      {
            call Leds.yellowToggle();
      }else {call Leds.redToggle();
   dbg(DBG_USR1, "echec\n");
             }
 }
  }
   return SUCCESS;
  }

event TOS_MsgPtr RecoieMsg.receive(TOS_MsgPtr m) {
  // reception  d'un message
  call Leds.greenToggle();
    atomic {
    poi = (DVhopMsg*) m->data;// pointeur sur le message reçu
      p =( DVhopMsg* ) msg.data;// pointeur sur le message emis
       switch(TOS_LOCAL_ADDRESS) {
         case 0 :
 atomic{
if( (ancre_recu[0][0]==FALSE) && (poi->id_ancre==1) ){//si
l'ancre 0 reçoit un message d'ancre 1
  ancre_recu[0][0]=TRUE;
       hop[0][0]=poi->hop+1;
         dbg(DBG_USR1, "le hop 00=%d reçu a partir de%d
id_node=%d\n", hop[0][0],poi->id_ancre,poi->id_node);
         }
         else if( (ancre_recu[0][1]==FALSE)
&&(poi>id_ancre==2)){    //si l'ancre 0 reçoit un message
d'ancre 2
           ancre_recu[0][1]=TRUE;
              hop[0][1]=poi->hop+1;
               dbg(DBG_USR1, "le hop 01=%d recue a partir de%d
idnode=%d\n", hop[0][1],poi->id_ancre,poi->id_node);
         }
if( (ancre_recu[0][0]==TRUE) &&(ancre_recu[0][1]==TRUE)&&
(v==TRUE)){//si l'ancre 0 reçoit un message d'ancre 1 et 2
alors on calcule le hopsize
```

```
v=FALSE;
 hopsize[0]=((L0L1+L0L2)/(hop[0][0]+hop[0][1]));
dbg(DBG_USR1, "le hopsize de L0=%f\n",hopsize[0]);
 //envoyer un broadcast avec les info suivants
 p->hopsize= hopsize[0];
 //p->hop=0;
 p->id_ancre=0;
 for(i=0;i<12;i++)
if (call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg),
&msg)== SUCCESS)
call Leds.redOn();
}

}
 break;
case 1 :
 atomic{
if( (ancre_recu[1][0]==FALSE) && (poi->id_ancre==0) ){
  ancre_recu[1][0]=TRUE;
       hop[1][0]=poi->hop+1;
dbg(DBG_USR1, "le hop 10=%d recue a partir de %d
idnode=%d\n", hop[1][0],poi->id_ancre,poi->id_node);
         }

else if( (ancre_recu[1][1]==FALSE) &&(poi->id_ancre==2)){
          ancre_recu[1][1]=TRUE;
             hop[1][1]=poi->hop+1;
  dbg(DBG_USR1, "le hop 11=%d recue a partir de%d
idode=%d\n", hop[1][1],poi->id_ancre,poi->id_node);
         }

if( (ancre_recu[1][0]==TRUE) &&(ancre_recu[1][1]==TRUE)&&
(vv==TRUE)){
vv=FALSE;
 hopsize[1]=((L0L1+L1L2)/(hop[1][0]+hop[1][1]));
dbg(DBG_USR1, "le hopsize de L1=%f\n",hopsize[1]);
p->hopsize= hopsize[1];
 //p->hop=0;
 p->id_ancre=1;

 for(i=0;i<8;i++)
if (call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg),
&msg)== SUCCESS)

call Leds.redOn();

}

}
 break;
```

```
case 2 :
 atomic{
if( (ancre_recu[2][0]==FALSE) && (poi->id_ancre==0) ){
  ancre_recu[2][0]=TRUE;
       hop[2][0]=poi->hop+1;
dbg(DBG_USR1, "le hop 20=%d recue a partir de%d idnode=%d\n",
hop[2][0],poi->id_ancre,poi->id_node);
        }

 else if( (ancre_recu[2][1]==FALSE) &&(poi->id_ancre==1)){
          ancre_recu[2][1]=TRUE;
             hop[2][1]=poi->hop+1;
dbg(DBG_USR1, "le hop 21=%d recue a partir de %d
idnode=%d\n", hop[2][1],poi->id_ancre,poi->id_node);
        }

if( (ancre_recu[2][0]==TRUE) &&(ancre_recu[2][1]==TRUE)&&
(vvv==TRUE)){
vvv=FALSE;
 hopsize[2]=((L0L2+L1L2)/(hop[2][0]+hop[2][1]));
dbg(DBG_USR1, "le hopsize de L2=%f\n",hopsize[2]);
p->hopsize= hopsize[2];
// p->hop=0;
 p->id_ancre=2;

 for(i=0;i<12;i++)
if (call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg),
&msg)== SUCCESS)

call Leds.redOn();
}

}
 break;
default :
atomic{
 if( poi->hopsize!=0  ){// si le hopsize est  calculer
 if((r==FALSE)  && (poi->id_ancre<3) ){
 r=TRUE;
premier_hop_arriver=poi->hopsize;
 dbg(DBG_USR1, "le premier hopsize reçu = %f a partir de
noeud %d elle a id_ancre= %d\n",premier_hop_arriver,poi-
>id_node,poi->id_ancre);
 }

if( (neud_recu_avant[0]==TRUE) && (poi->id_ancre==0) ){//  si
le massage reçue a id_ancre=0
 neud_recu_avant[0]=FALSE;
 hop_noeud[0]=poi->hop+1;

//dbg(DBG_USR1, "le hopsize reçu=%f a partir de %d et
idnode=%d\n",poi->hopsize,poi->id_ancre,poi->id_node);
```

```
dbg(DBG_USR1, "le hop 30=%d recue a partir de %d et
idnode=%d\n", hop_noeud[0],poi->id_ancre,poi->id_node);
hopsize[0]=poi->hopsize;
 p->hopsize= poi->hopsize;
 p->id_ancre=0;
 p->id_node=TOS_LOCAL_ADDRESS;
 p->hop=poi->hop+1;

 for(i=0;i<8;i++)
 if (call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg),
&msg)== SUCCESS)
call Leds.redOn();
         }

else if( (neud_recu_avant[1]==TRUE) &&(poi->id_ancre==1)){
         neud_recu_avant[1]=FALSE;
       hop_noeud[1]=poi->hop+1;

dbg(DBG_USR1, "le hop 31=%d reçue a partir de %d et
idnode=%d\n", hop_noeud[1],poi->id_ancre,poi->id_node);
 p->hopsize= poi->hopsize;
 hopsize[1]=poi->hopsize;
 p->id_ancre=1;
 p->id_node=TOS_LOCAL_ADDRESS;
 p->hop=hop_noeud[1];
 for(i=0;i<12;i++)
 if (call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg),
&msg)== SUCCESS)
call Leds.redOn();
  }
else if( (neud_recu_avant[2]==TRUE) &&(poi->id_ancre==2)){
         neud_recu_avant[2]=FALSE;
       hop_noeud[2]=poi->hop+1;
 //dbg(DBG_USR1, "le hopsize reçu=%f a partir de %d et
idnode=%d\n",poi->hopsize,poi->id_ancre,poi->id_node);
 dbg(DBG_USR1, "le hop 32=%d recue a partir de %d et
idnode=%d\n", hop_noeud[2],poi->id_ancre,poi->id_node);

 hopsize[2]=poi->hopsize;
 p->hopsize= poi->hopsize;
 p->id_ancre=2;
 p->hop=hop_noeud[2];
 p->id_node=TOS_LOCAL_ADDRESS;

  for(i=0;i<12;i++)
 if (call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg),
&msg)== SUCCESS)

call Leds.redOn();

 }
```

```
if(affiche1==TRUE && neud_recu_avant[0]==FALSE &&
neud_recu_avant[1]==FALSE && neud_recu_avant[2]==FALSE ){//si on
a reçoit les hop de tous les ancres
 affiche1=FALSE;//variable pour afficher le min une seule fois
 dbg(DBG_USR1, "la distance entre %d et ancre 0
=%f\n",TOS_LOCAL_ADDRESS,premier_hop_arriver*hop_noeud[0]);
  dbg(DBG_USR1, "la distance entre %d et ancre 1
=%f\n",TOS_LOCAL_ADDRESS,premier_hop_arriver*hop_noeud[1]);
   dbg(DBG_USR1, "la distance entre %d et ancre 2
=%f\n",TOS_LOCAL_ADDRESS,premier_hop_arriver*hop_noeud[2]);
  }
}

  else  if (poi->hopsize==0) {//le hopesize n'est pas encore
calculer
if(poi->id_ancre<3) {
if((poi->id_ancre==0)&&(neud_recu_apres[0]==FALSE)){
neud_recu_apres[0]=TRUE;
 p->hopsize=0;
 p->id_ancre=poi->id_ancre;
 p->hop=poi->hop+1;
  p->id_node=TOS_LOCAL_ADDRESS;
  for(i=0;i<12;i++)
 call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg), &msg);
}

else if((poi>id_ancre==1)&&(neud_recu_apres[1]==FALSE)){
neud_recu_apres[1]=TRUE;
 p->id_ancre=poi->id_ancre;
 p->hopsize=0;
  p->id_node=TOS_LOCAL_ADDRESS;
 p->hop=poi->hop+1;

 for(i=0;i<8;i++)
 call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg), &msg);
                              }
else if((poi>id_ancre==2)&&(neud_recu_apres[2]==FALSE)){
neud_recu_apres[2]=TRUE;

 p->id_ancre=poi->id_ancre;
 p->hopsize=0;
 p->id_node=TOS_LOCAL_ADDRESS;
 p->hop=poi->hop+1;
 for(i=0;i<12;i++)
 call envoiMsg.send(TOS_BCAST_ADDR, sizeof( DVhopMsg), &msg);
                     }
  }
  }
  }
break;
      }
    }
   return m;
  }
}
```

Annexe 2

Procédure d'installation sous Windows XP et exemple d'execution

Ce guide propose l'installation du principal outil nécessaire au bon fonctionnement du système, notamment Cygwin (couche d'émulation de l'API Linux) qui permet d'avoir une interface Unix sous Windows. Cygwin est un environnement d'émulation Linux qui permet d'avoir un shell et de compiler et exécuter les programmes Linux (On dispose ainsi de gcc, apache, bash, etc.).

1- Télécharger le fichier **tinyos-1.1.0-1is.exe** de la source **http://www.tinyos.net/dist-1.1.0/tinyos/windows/ .**

2- Exécuter ce fichier pour installer la version 1.1.0 sous windows XP. L'installation se fait automatiquement. Un raccourci de Cygwin est sauvegardé sur le bureau.

3- Accéder à **C:\tinyos\cygwin\opt\tinyos-1.x\doc\tutorial\verifyhw.html** et suivre les

étapes que contient cette page afin de vérifier si l'installation est bien réussie.

4- Accéder à: **cd /opt/tinyos-1.x/tools/java**

ET taper : **make**

5- Installer les mises à jour de NesC1.1.1 and TinyOS1.1.15.

Pour se faire, rechercher sur le net http://www.tinyos.net/dist-1.1.0/tinyos/windows/ ces mises à jour en téléchargeant le **rpm** et le mettant dans **C:\tinyos\cygwin\home\Nom de votre répertoire**

Et taper dans le **shell**:

rpm -ivh --ignoreos nesc-1.1.2b-1.cygwin.i386.rpm

rpm -ivh --ignoreos --force tinyos-1.1.15Dec2005cvs-1.cygwin.noarch.rpm

6- Aller à **opt/tinyos-1.x/tools/java/net/tinyos/sim** et verifier si ces fichiers sont presents:

SimObjectGenerator.java et **MoteSimObjectGenerator.java**

S'ils existent, alors les supprimer de ce répertoire.

7- Editer le **makefile** qui est dans **C:\tinyos\cygwin\opt\tinyos-1.x\tools\java\net\tinyos\sim**

et écrire cette instruction si elle n'existe pas : **net/tinyos/message/avrmote/*.class**

Le makefile

.....

.....

net/tinyos/sim/plugins/plugins.list \
net/tinyos/sf/*.class \
net/tinyos/util/*.class \
net/tinyos/packet/*.class \
net/tinyos/message/*.class \
**net/tinyos/message/avrmote/*.class **
org/apache/oro/text/regex/*.class \
org/python/compiler/*.class \
org/python/core/*.class \
org/python/modules/*.class \
org/python/parser/*.class \
org/python/parser/ast/*.class \
org/python/rmi/*.class \

.....

8- Aller à **shell** et taper:

cd /opt/tinyos-1.x/tools/java/net/tinyos/sim

make clean

make

9- Accéder à l'application qui va être simulée. On prend par exemple, l'application Blink.

Accéder au **shell** et faire:

cd /opt/tinyos-1.x/apps/blink

make pc

puis

export PATH="$TOSROOT/tools/java/net/tinyos/sim:$PATH"

puis

TinyViz -run build/pc/main.exe 20 ///Insérer le nombre de noeuds. Par exemple 20.

Liste des Figures :

Liste des tables :