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

République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option: *Réseaux et Systèmes Distribués (R.S.D)*

Thème

Mise en place d'un réseau de capteurs sans fil pour l'irrigation intelligente

Réalisé par :

- ***M^{elle} BOUZIDI Zeyneb***
- ***M^{elle} BENAMEUR Amina***

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

- *M^{r} BENMAMMAR Badr* *(Président)*
- *M^{me} LABRAOUI Nabila* *(Encadreur)*
- *M^{r} LEHSAINI Mohammed* *(Examinateur)*
- *M^{r} BENAISSA Mohammed* *(Examinateur)*

Année universitaire : 2011-2012

Remerciements

Nous tenons à remercier en cette occasion tout le corps enseignant qui a contribué à notre formation, plus particulièrement le chef du département informatique Monsieur Benammar pour son dynamisme et la qualité de l'enseignement qui nous a été dispensé, notre encadreur Mme Labraoui Nabila pour son aide et les précieux conseils qu'elle a bien voulu nous donner.

Nos vifs remerciements vont à Monsieur Benmammar Badr, Monsieur Lehsaini Mohammed Et Monsieur Benaissa Mohammed pour avoir accepté de jurer notre mémoire.

Nous n'oublions pas nos parents pour leur contribution, leur soutien et leur patience, nos proches et amis qui nous ont soutenu et encouragé.

Nous remercions aussi vivement tous les responsables de l'université de Tlemcen ainsi que toutes les personnes qui nous ont aidé de près ou de loin de nos études.

Dédicace

Je remercie Dieu de m'avoir donner le courage pour accomplir ce modeste travail que je dédie :

A mes très chers parents qui sont la bougie qui illumine ma vie

A Mes neveux Abd elAdim et Abd elMounim

A mon frère Didou et sa femme Khouloud

A mes sœurs Fatima Zohra, leyla, Abir et Nadia et son mari Adnane

A mes très chers amies Amina et Imene

A ma tante Khadidja

A toute ma grande famille, à tous mes amis et à tous mes enseignants

Et à tous ceux que j'aime et qui m'aiment

BOUZIDI Zeyneb

Dédicace

Tout d'abord je tiens à remercier Dieu le tout puissant de m'avoir donner la santé, la volonté et le courage.

Ensuite, a mes parents pour leur soutien et leurs encouragements et qu'aucune dédicace ne pourra compenser leur sacrifice pour mon bien être et mon bonheur.

A mon cher frère Chems Eddine pour lequel j'ai beaucoup d'estimes.

A mes sœurs Assia et Meryem que je leur souhaite une vie pleine de joie et de réussite.

A mes grands pères, grands-mères, mes oncles, mes tantes et toute ma famille.

A mes amies, en particulier ma collègue du présent projet Zeyneb qui n'a jamais manqué de travailler et ceux de la promotion de la 2ème année master réseaux et systèmes distribués.

A notre encadreur Mme Labraoui Nabila.

Et à tous ceux qui nous ont soutenus pour l'achèvement de ce projet avec réussite.

Je dédie ce modeste travail.

BENAMEUR Amina

Table des matières

Introduction générale

Les réseaux de capteurs sans fil (RCSF)- Wireless Sensor Networks (WSN) - sont considérés comme un type spécial de réseaux ad hoc. Les nœuds de ce type de réseaux consistent en un grand nombre de micro-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 sont dispersés aléatoirement à travers une zone géographique, appelée champ de captage, qui définit le terrain d'intérêt pour le phénomène capté. Les données captées sont acheminées à un nœud considéré comme un "point de collecte", appelé nœud puits (ou sink). Ce dernier peut être connecté à l'utilisateur du réseau via Internet ou un satellite. Ainsi, l'usager peut adresser des requêtes aux autres nœuds du réseau, précisant le type de données requises et récolter les données environnementales captées par le biais du nœud puits.

Les applications des réseaux de capteurs sans fil sont nombreuses. Elles comprennent différents domaines : agricole, militaire,...etc. Notre projet de fin d'étude proposé consiste à mettre en place un réseau de capteurs sans fil pour l'irrigation intelligente et donc l'économie de l'eau. Dans le cadre d'exploitation de technologie de surveillance de l'environnement, le projet cible le secteur de l'agriculture et consiste à développer une nouvelle méthodologie efficace pour le déploiement de ces réseaux pour superviser des champs agricoles pour la détection des zones sèches afin de bien contrôler le processus d'irrigation, selon le taux d'humidité du sol. Un autre but de ce projet est la maîtrise de cette nouvelle technologie à savoir les réseaux de capteurs sans fil, dans le but d'améliorer et d'évoluer les techniques agricoles pour un meilleur rendement.

On propose dans ce PFE, d'aborder les réseaux de capteurs, et donc d'installer le système d'exploitation TinyOS qui est conçu spécialement pour ce type de réseau et qui est écrit dans un langage spécifique optimisé pour les capteurs « le langage NesC ». Afin d'aborder tous les aspects ayant trait au fonctionnement de ces réseaux, notre mémoire est organisé comme suit :

Chapitre I : Généralités sur les réseaux de capteurs sans fil, dans ce chapitre on va présenter les réseaux des capteurs, ou nous dresserons un état de l'art sur ces réseaux.

Chapitre II : Mise en place de la plateforme TinyOS, ou nous décrirons le fonctionnement de ce système ainsi la description de son langage de programmation.

Chapitre III : L'irrigation intelligente pour économiser de l'eau, on montre dans ce chapitre l'importance de l'eau ainsi les systèmes d'irrigation automatique utilisé actuellement et à la fin on propose une nouvelle technologie pour économiser de l'eau.

Chapitre IV : Implémentation et évaluation de l'application, cette partie est consacrée à la présentation du domaine de simulation de réseaux des capteurs sans fil, et les différentes phases de conception et d'implémentation de l'application.

Ce travail est terminé par une conclusion générale et une bibliographie.

Chapitre I :

Généralités sur les Réseaux de capteurs sans fil (WSN: Wireless Sensor Networks)

Ce chapitre consiste à exposer les réseaux des capteurs sans fil : leur caractéristiques, leur topologies, leur systèmes d'exploitations et les technologies utilisées, algorithmes de routage ainsi les applications de ces réseaux.

1. Introduction

De nombreux systèmes nécessitent de prendre en compte l'environnement pour mesurer les phénomènes physiques afin de prendre les décisions nécessaires. Les progrès de ces dernières années en microélectronique et micromécanique ont permis de concevoir des capteurs de plus en plus petits, de plus en plus performants, autonomes et dont les capacités énergétiques ont évolué avec le temps. D'autre part, les techniques de réseaux mobiles permettent d'affranchir des fils et donc de déployer facilement des réseaux de capteurs dans les endroits même difficiles à y accéder.

De plus, dans la vie courante, l'utilisation des capteurs sans fil est en demande croissante pour la supervision et la sécurité. Les industries proposent alors des capteurs sans fil qui peuvent renseigner l'utilisateur sur plusieurs données. Ces capteurs peuvent être reliés formant ainsi un réseau sans fil se basant sur des protocoles pour se communiquer et proposer des programmes et des réseaux embarqués [KP09].

2. Histoire des réseaux de capteurs

Les récents progrès des nouvelles techniques ont provoqué une énorme importance dans le domaine des réseaux sans fil. La technologie des réseaux de capteurs sans fil est devenue une des merveilleuses technologies dans le 21ème siècle ; les réseaux de capteurs ont montré leur impact sur notre vie quotidienne.

CHONG, et al. [SN10] ont parlé de trois générations des nœuds de capteurs:

Génération	Période	Taille	Poids	Batterie
1ère	Les années 80 et 90	Grande boîte à chaussures	Kilogrammes	Grosse
2ème	Entre 2000 et 2003	Boîte de cartes	Grammes	AA
3ème	2010	Particule de poussière	Négligeable	Solaire

Tableau 1. 1 : Les trois générations des nœuds de capteurs

Grâce aux progrès des techniques sans fil, les réseaux de capteurs seront aussi communs dans notre vie quotidienne comme les ordinateurs et l'internet.

3. Présentation des réseaux de capteurs

3.1 Définition d'un capteur

Les capteurs sont des dispositifs capables de générer des données relatives à leur environnement physique. Ces dispositifs embarquent un système de communications afin d'échanger des données formant ainsi un réseau implicite.

Le thème « réseaux de capteurs » n'est pas d'inventer de nouveaux capteurs mais de découvrir de nouveaux modèles de communication, leurs limites et leurs performances. La durée de vie des capteurs est limitée par la durée de vie de leur batterie, le tableau suivant montre une petite comparaison entre les réseaux de capteurs et les réseaux Ad-Hoc :

Senseurs	Ad-Hoc
1. Objectif ciblé	1.Générique/communication
2. Nœuds en collaboration pour remplir un objectif	2. Chaque nœud à son propre objectif
3.Flot de données (many to one)	3.Flot (any to any)
4. Très grand nombre de nœuds n'ayant pas tous un ID	4. Notion d'ID
5. Energie (facteur déterminant)	5. Débit majeur
6. Utilisation du broadcast	6. Communication point à point

Tableau 1. 2 : Comparaison entre capteurs et Ad-Hoc [EF04].

3.2 Définition d'un RCSF ou WSN (Wireless Sensor Network)

Un réseau de capteurs sans fil est un type spécial de réseaux ad hoc avec un grand nombre de nœuds qui sont des micro-capteurs capables de recevoir et de transmettre des données environnementales d'une manière autonome sans intervention humaine [NB04].

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 (par exemple : lâchée de capteurs sur un volcan pour étudier les phénomènes vulcanologiques et leurs évolutions).

Le réseau possède en général un nœud particulier, la base (ou sink), connectée avec les autres nœuds par un réseau filaire est reliée à une alimentation électrique.

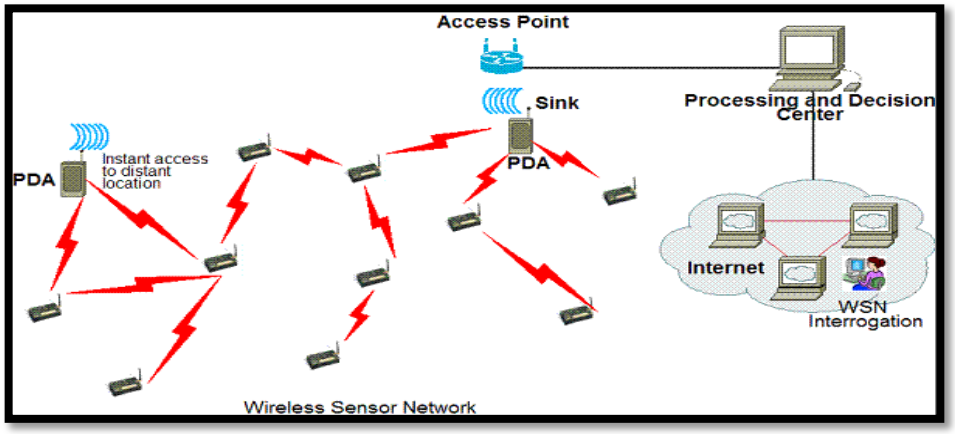


Figure I. 1 : Réseaux de capteurs.

3.3 Les sous-systèmes d'un nœud

Un réseau de capteurs est un réseau informatique composé de nœuds appelés aussi motes, généralement un mote est composé d'un microprocesseur, RAM, interface radio (pour la communication avec les voisins), micro-capteurs capables de collecter et de traiter les informations provenant de l'environnement et une source d'énergie.

Chaque nœud du réseau de capteurs est composé de quatre sous systèmes :

3.3.1 Sous système de calcul

Il comprend un processeur associé généralement à une petite unité de stockage et fonctionne à l'aide de système d'exploitation TinyOS spécial pour les micros capteurs. Ce sous-système est chargé d'exécuter les protocoles de communication permettant de faire collaborer le nœud avec les autres nœuds du réseau.

3.3.2 Sous système de communication

Il est considéré comme un système radio à courte portée et peut fonctionner en quatre modes : Transmit, Receive, Idle, Sleep.

3.3.3 Sous système de capteurs

C'est l'ensemble de capteurs et d'actionneurs (détecteurs) qui relient le nœud au monde extérieur.

3.3.4 Sous système de génération de courant

On veut ici augmenter au maximum la durée de vie de la batterie.

3.4 Architecture d'un nœud

Tous les capteurs respectent globalement la même architecture basée sur un noyau central autour duquel s'articulent les différentes interfaces d'entrée sortie, de communication et d'alimentation.

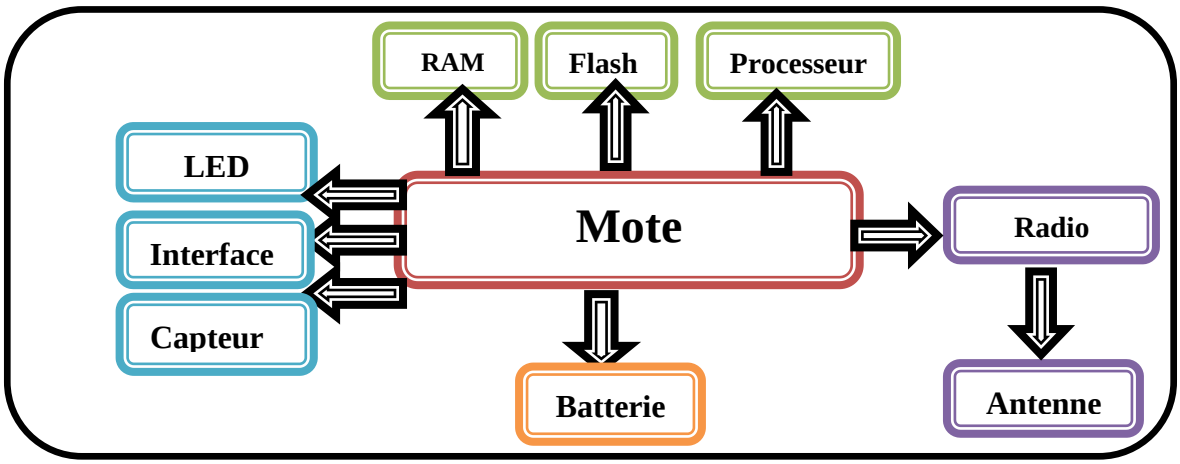


Figure I. 2 : Architecture générale d'un capteur

On peut voir sur la figure I-3 les différents composants qui constituent un capteur, pour être plus précis chaque groupe de composants possède son propre rôle:

- **Mote, processeur, RAM et Flash** : On appelle généralement mote la carte physique utilisant le système d'exploitation pour fonctionner. Celle-ci a pour cœur le bloc constitué du processeur et des mémoires RAM et Flash. Cet ensemble est à la base du calcul binaire et du stockage, temporaire pour les données et définitif pour le système d'exploitation.
- **Radio et antenne** : Un capteur est conçu pour mettre en place des réseaux sans fils, les équipements étudiés sont donc généralement équipés d'une radio ainsi que d'une antenne afin de se connecter à la couche physique que constitue les émissions hertziennes.
- **LED, interface, capteur** : Prévus pour mettre en place des réseaux de capteurs, on retrouve donc des équipements bardés de différents types de détecteurs et autres entrées.
- **Batterie** : Comme tout dispositif embarqué, ils disposent d'une alimentation autonome telle qu'une batterie, et parfois d'un panneau solaire pour permettre de recharger cette batterie, ce qui lui permet d'être disposé dans un endroit parfois inaccessible.

Cependant quelques différences existent suivant les fabricants. Chacun d'eux développe son type de capteurs, ces types peuvent être mica, mica2, telos ou telosb par exemple.

Avec des contraintes hardware aussi strictes dues à la miniaturisation des capteurs, la partie software doit être la plus adaptée possible, d' où un lien très fort entre ces deux parties [SB11].

4. Architecture de communication dans les réseaux de capteurs

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.

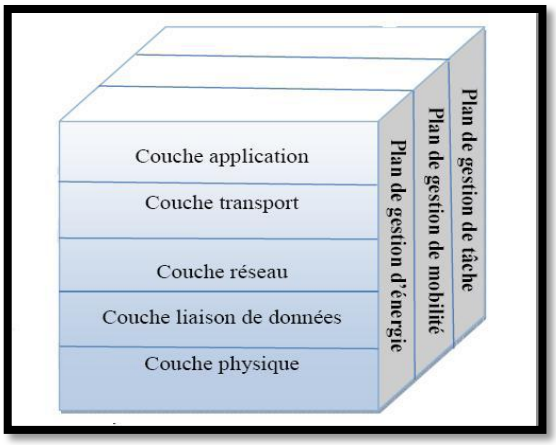


Figure I. 3 : Modèle en couches du réseau de capteurs sans fil

Rôles des couches :

- ✓ **Couche physique :** Matériels pour envoyer et recevoir les données.
- ✓ **Couche liaison de données :** Gestion des liaisons entre les nœuds et les stations de base, contrôle d'erreurs.
- ✓ **Couche réseau :** Routage et transmission des données.
- ✓ **Couche transport :** Transport des données, contrôle de flux.
- ✓ **Couche application :** Interface pour les applications au haut niveau.
- ✓ **Plan de gestion d'énergie :** Contrôle l'utilisation d'énergie.
- ✓ **Plan de gestion de mobilité :** Gestion des mouvements des nœuds.
- ✓ **Plan de gestion de tâche :** Balance les tâches entre les nœuds afin d'économiser de l'énergie.

5. Types de réseaux de capteurs sans fil

5.1 Réseaux de poursuite

Ces réseaux sont généralement développés par l'armée, ils peuvent servir à surveiller toutes les activités d'une zone stratégique ou d'accès difficile, ainsi on pourra détecter des agents chimiques, biologiques ou des radiations avant des troupes. On peut aussi penser à des capteurs embarqués sur les soldats pour faciliter leur guidage et le contrôle de leur position depuis la base.

5.2 Réseaux de collection des données d'environnement

Les nœuds de ce type de réseau peuvent avoir plusieurs fonctionnalités et différents types de capteurs. Ce type de réseau nécessite généralement un flux de données faible, une durée de vie importante ; il sert à la collecte périodique des données environnementales puis leur transmission vers la station de base.

5.3 Réseaux de surveillance et sécurité

La différence entre ce réseau et le réseau de collection d'environnement est que les nœuds ne transmettent pas l'ensemble des données collectées mais seulement les rapports concernant une violation de la sécurité. Ce sont en général des nœuds fixes qui contrôlent d'une façon continue la détection d'une anomalie dans le fonctionnement d'un système. Ainsi les altérations dans la structure d'un bâtiment, suite à un séisme, pourraient être détectées par des capteurs intégrés dans les murs ou dans le béton, sans alimentation électrique ou autres connexions filaires.

6. Fonctionnement d'un réseau de capteurs

Les données captées par les nœuds sont acheminées grâce à un routage multi-saut à un nœud considéré comme un "point de collecte" appelé nœud-puits (ou sink). Ce dernier peut être connecté à l'utilisateur du réseau via Internet, un satellite ou un autre système. L'usager peut adresser des requêtes aux autres nœuds du réseau, précisant le type de données requises pour récolter les données environnementales captées par le biais du nœud-puits.

Les progrès conjoints de la microélectronique, microtechnique, des technologies de transmission sans fil et des applications logicielles ont permis de produire à coût raisonnable des micro-capteurs de quelques millimètres cubes de volume, susceptibles de fonctionner en réseaux.

Ils intègrent : une unité de captage chargée de capter des grandeurs physiques (chaleur, humidité, vibrations, rayonnement...) et de les transformer en grandeurs

numériques, une unité de traitement informatique et de stockage des données et un module de transmission sans fil.

7. Topologies des réseaux de capteurs

Un réseau de capteurs sans fil est composé d'un ensemble de nœuds capteurs et des Gateway qui s'occupent de collecter les données des capteurs et de les transmettre à l'utilisateur via l'internet ou le satellite, 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.

Avantage : simplicité et faible consommation d'énergie des nœuds, moindre latence de communication entre les nœuds et la station de base.

Inconvénient : la station de base est vulnérable, car tout le réseau est géré par un seul nœud.

7.2 Topologie en toile (Mesh Network)

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 les 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.

Avantage : Possibilité de passer à l'échelle du réseau, avec redondance et tolérance aux fautes.

Inconvénient : Une consommation d'énergie plus importante est induite par la communication multi-sauts. Une latence est créée par le passage des messages des nœuds par plusieurs autres avant d'arriver à la station de base.

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.

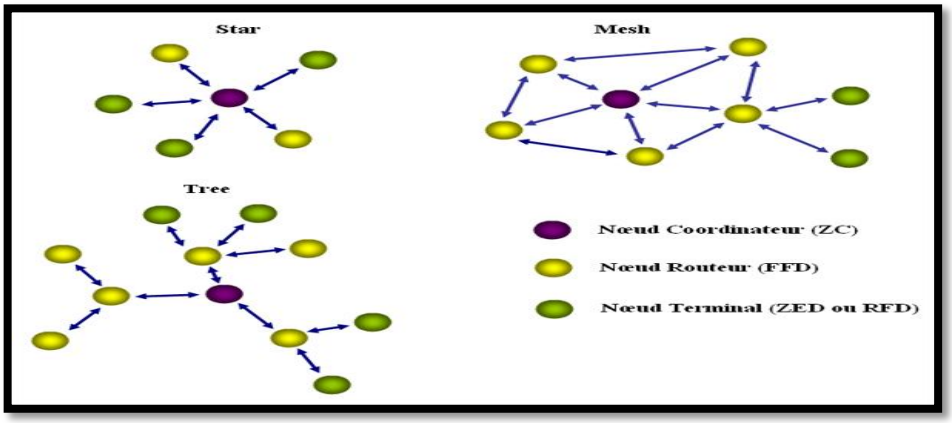


Figure I. 4 : Topologies des réseaux de capteurs

8. Les systèmes d'exploitation pour les réseaux de capteurs

Plusieurs systèmes d'exploitation ont été développés pour répondre aux contraintes particulières des réseaux de capteurs, ils sont les suivants :

8.1 TinyOS

TinyOS (que l'on va étudier dans le chapitre suivant) est un système d'exploitation open source conçu pour les capteurs sans fils et développé par l'Université de Berkeley. Il est basé sur une architecture à base de modules : pilotes pour les capteurs, les protocoles réseau et les services distribués. Les composants sont programmés en NesC, un langage de programmation dérivé du C adapté aux faibles ressources physiques des capteurs. Un certain nombre de plateformes sont directement programmables comme par exemple les tmote ou les MicaZ (ces deux modèles sont compatibles avec ZigBee). TOSSIM est un simulateur de capteurs pour les programmes TinyOS, tout programme en NesC peut être compilé de manière à être exécuté dans TOSSIM, ce qui permet de simuler le comportement d'un ou plusieurs capteurs ainsi de les programmer.

8.2 MOS

MOS est un système d'exploitation open-source pour les capteurs développés par le MANTIS group à l'université du Colorado. MOS est développé en C. Il supporte les plateformes de la famille MICA et de la famille Telos.

8.3 SOS

SOS est un système d'exploitation open-source pour les réseaux de capteurs développés par le laboratoire réseaux et systèmes embarqués de l'université de Los Angeles, il est basé sur une architecture modulaire.

SOS est écrit en langage C, il supporte les plateformes de la famille MICA. Avrora est un simulateur pour les programmes SOS. SOS à l'avantage de posséder une implémentation complète de la topologie en étoile.

- En conclusion, TinyOS est le système d'exploitation le plus utilisé pour les réseaux de capteurs sans fils car il a l'avantage par rapport à MOS et SOS d'être écrit dans un langage spécifique optimisé pour les capteurs [VJ06].

9. Technologies utilisés

9.1 Bluetooth / IEEE 802.15.4

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.

9.2 ZigBee

Beaucoup moins connue que Bluetooth, ZigBee est une norme de transmission de données sans fil permettant la communication de machine à machine. Sa très faible consommation électrique et ses coûts de production très bas en font une candidate idéale pour la domotique ou le matériel de type capteur, télécommande ou équipement de contrôle dans le secteur industriel.

Les débits autorisés sont relativement faibles, entre 20 et 250 Kbits/s, mais c'est véritablement sa très faible consommation électrique qui en fait son atout principal. ZigBee fonctionne sur la bande de fréquences des 2,4 GHz et sur 16 canaux, sa portée était au début d'une dizaine de mètres, elle est désormais de 100 mètres.

9.3 Algorithmes de routage

ZigBee alliance (association visant à promouvoir la technologie) intègre dans sa norme Zigbee1.0, des recommandations pour les algorithmes de routage.

Il existe 2 types de routage au niveau de la couche réseau :

- **Le routage direct** : lorsqu'un dispositif voulant transmettre des données connaît l'adresse réseau du destinataire. Cette adresse est donc transmise dans la trame pour atteindre et agir sur le dispositif prévu.
- **le routage indirect** : se fait lorsqu'un dispositif ne connait pas l'adresse du destinataire. Un équipement de type routeur ou coordinateur fait la relation avec le vrai destinataire d'après la table de routage et la table de découverte des routes.

De plus il ya deux protocoles de routages :

- **Protocole de routage réactif AODV :**

Utilisé pour les réseaux maillés, une route est établie uniquement sur demande c'est-à-dire les protocoles de routage réactif vont chercher la route à utiliser à chaque demande d'émission, l'avantage ici est qu'il ne surcharge pas le trafic.

- **Protocole de routage proactif OLSR :**

C'est un protocole de routage pour objet, il permet d'échanger des informations sur la topologie et la régularité du réseau avec les autres nœuds. On y retrouve donc une approche des problèmes liées à l'usage d'un réseau hertzien, notamment la gestion de liens non fiables. Cela permet aux machines d'un réseau ad-hoc de communiquer entre elles pour qu'elles s'échangent des informations sur leur état. Et ainsi cela permet de créer dynamiquement la topologie du réseau en fonction des machines qui y sont disponibles. Ce système va en temps réel maintenir l'état de toutes les routes qui lui sont disponibles selon les règles propres à ce protocole.

9.4 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 à petite pile, des supports de capteurs et de sécurité et un débit de transmission allant jusqu'à 200kbits/s.

10. Exemples sur les capteurs

Ici, on présente quelques capteurs utilisés mais il existe plusieurs :

- **MIB510CA :**

Carte d'interface série pour la programmation de toutes les plates formes MICA. Elle possède deux connecteurs de 51 pins pour raccorder la carte aux capteurs.

Figure I. 5 : le capteur MIB510CA

- **MCS410 :**

Système de localisation, destiné surtout pour usage intérieur, fournit des informations de positionnement précises, identification des espaces, coordonnées de position et d'orientation. C'est un produit de recherches et d'études. Sa précision varie

entre 1cm et 3 cm. Combine les technologies RF et d'ultrason, se monte sur les murs ou les plafonds et peut être utilisé en émission comme en réception.

Figure I. 6 : le capteur MCS410

11. Applications

La diminution de taille et de coût des micro-capteurs, l'élargissement de la gamme des types de capteurs disponibles (thermique, optique, vibrations,...) et l'évolution des supports de communication sans fil, ont élargi le champ d'application des réseaux de capteurs. Ils s'insèrent notamment dans d'autres systèmes tels que le contrôle et l'automatisation des chaînes de montage.

Ils permettent de collecter et de traiter des informations complexes provenant de l'environnement (météorologie, étude des courants, de l'acidification des océans, de la dispersion de polluants, etc.).

Certains prospectivistes pensent que les réseaux de capteurs pourraient révolutionner la manière même de comprendre et de construire les systèmes physiques complexes, notamment dans les domaines militaire, environnemental, domestique, sanitaire et de la sécurité, etc.

- **Applications militaires :** On peut penser à un réseau de capteurs déployé sur un endroit stratégique ou d'accès difficile, afin de surveiller toutes les activités des forces ennemies, ou d'analyser le terrain avant d'y envoyer des troupes (détection d'agents chimiques, biologiques ou de radiations).
- **Applications domestiques :** En plaçant, sur le plafond ou dans le mur, des capteurs, on peut économiser l'énergie en gérant l'éclairage ou le chauffage en fonction de la localisation des personnes.
- **Applications environnementales :** Les réseaux de capteurs sont beaucoup appliqués dans ce domaine pour détecter des incendies, surveiller des catastrophes naturelles, détecter des pollutions et suivre des écosystèmes.
- **Applications agricoles :** Dans les champs agricoles, les capteurs peuvent être semés avec les graines. Ainsi, les zones sèches seront facilement identifiées et l'irrigation sera donc plus efficace.

- **Applications médicales :** Les réseaux de capteurs ont aussi des développements dans le domaine de diagnostic médical. Par exemple, des micro-caméras sont capables, sans avoir recours à la chirurgie, de transmettre des images de l'intérieur d'un corps humain avec une autonomie de 24 heures.
- **Applications transportés :** Il est possible d'intégrer des nœuds capteurs au processus de stockage et de livraison. Le réseau ainsi formé, pourra être utilisé pour connaître la position, l'état et la direction d'un paquet ou d'une cargaison.

Figure I. 7 : Applications des réseaux de capteurs

12. Agrégation de données dans les réseaux de capteurs

Dans les capteurs, la problématique principale concerne la consommation d'énergie : en effet, ces derniers doivent rester opérationnels le plus longtemps possibles, comme il n'est pas possible de recharger leur énergie ni changer les piles il est nécessaire d'économiser au maximum l'énergie consommée par ces derniers.

On estime que la transmission des données d'un capteur représente environ 70% de sa consommation d'énergie.

De plus, les réseaux de capteurs étant assez denses en général, cela signifie que des nœuds assez proches en terme de distance (voisins) peuvent capter les mêmes données et donc il apparaît nécessaire d'introduire le mécanisme d'agrégation de données afin d'éviter la duplication d'information au sein du réseau de capteurs et donc de préserver leur énergie et d'augmenter la durée de vie du réseau.

L'agrégation est une technique utilisée pour réduire la transmission d'informations redondantes et consiste à remplacer les lectures individuelles de chaque capteur par une vue globale, collaborative sur une zone donnée (clustering), et Avec

cette technique, les nœuds intermédiaires agrègent l'information reçue de plusieurs sources. Cette technique est connue aussi sous le nom de fusion de données.

13. Les attaques sur les réseaux de capteurs :

Les réseaux de capteurs se développant et faisant partie des infrastructures critiques, il est tout à fait naturel de penser à la menace que posent les virus et vers sur ces réseaux.

Dans l'internet, un attaquant peut compromettre des machines en exploitant des vulnérabilités résultant souvent d'un dépassement d'un buffer en mémoire, permettant d'écrire dans la pile car un capteur étant un mini-ordinateur, possédant un CPU, une mémoire, des Entrées/Sorties.

Cependant, les capteurs ont plusieurs caractéristiques qui rendent leurs compromissions à distance par un virus très délicates :

- Les mémoires (programmes et données) sont souvent physiquement séparées, il est alors pratiquement impossible d'exécuter le code qui serait inséré dans la pile, comme c'est souvent le cas dans les attaques qui exploitent un dépassement de buffer pour écraser la pile.
- Le code application est souvent protégé en écriture. Un attaquant ne peut pas modifier les programmes présentés en mémoire.
- La taille des paquets que peut recevoir un capteur est très limitée (typiquement 28 octets), ce qui rend l'injection du code difficile.
- Les techniques qu'utilisent les vers pour compromettre une machine sur internet, ne peuvent donc pas être utilisées directement sur les capteurs. Nous avons cependant montré, en concevant un des premiers virus/vers pour capteurs de type MicaZ/TinyOS, que la conception de virus, bien que difficile mais n'est pas impossible.

En conclusion, Toutes les applications des réseaux de capteurs ont des contraintes de sécurité très différentes. Cependant, dans la plupart d'entre elles, l'intégrité et l'authenticité des données doivent être fournies pour s'assurer que des nœuds non-autorisés ne puissent pas injecter des données dans le réseau. Le chiffrement des données est souvent requis pour des applications sensibles telles que les applications militaires ou les applications médicales.

14. Caractéristiques et limites des réseaux de capteurs

14.1 Caractéristiques

Absence d'infrastructure : Les réseaux ad hoc, en général, se distinguent des autres réseaux mobiles par la propriété d'absence d'infrastructure préexistante et de tout genre d'administration centralisée. Les hôtes mobiles sont responsables d'établir et de maintenir la connectivité du réseau d'une manière continue**.**

Taille : Un grand nombre de nœuds dispersés aléatoirement (des réseaux de 10000 nœuds peuvent être envisagés)

Contrainte d'énergie : Dans plusieurs applications, les nœuds de capteurs sont placés dans des surfaces distantes, le service du nœud peut ne pas être possible, dans ce cas la durée de vie du nœud peut être déterminée par la durée de vie de la batterie, ce qui exige la minimisation des dépenses d'énergies.

Topologie dynamique : Les capteurs peuvent être attachés à des objets mobiles qui se déplacent d'une façon libre et arbitraire rendant ainsi, la topologie du réseau fréquemment changeante.

Auto organisation du réseau : Ceci peut être nécessaire dans plusieurs cas. Par exemple, un réseau comportant un grand nombre de nœuds placés dans des endroits hostiles où la configuration manuelle n'est pas faisable, doit être capable de s'auto-organiser. Un autre cas est celui où un nœud est inséré ou retiré (à cause d'un manque d'énergie ou de destruction physique), ainsi le réseau doit être capable de se reconfigurer pour continuer sa fonction.

Sécurité physique limitée : Les RCSF mobiles sont plus touchés par les paramètres de sécurité que les réseaux filaires classiques. Cela se justifie par les contraintes et limitations physiques qui font que le contrôle des données transférées doit être minimisé.

14.2 Limites

- Les ressources de calcul et de mémoire des nœuds sont relativement faibles, par exemple les nœuds de capteur de type "mote" sont composés d'un microcontrôleur 8-bits 4MHz, 40 KOctets de mémoire et une radio avec un débit d'environ 10 kbps.

  Cela reste vrai même pour les nœuds de moyenne gamme, comme les UCLA/ROCKWELL'S WINS, qui ont un processeur StrongARM 1100 avec une mémoire flash de 1 MO, une mémoire RAM de 128 KO et une radio de 100 Kbps.

- Non seulement les capacités des nœuds sont faibles, mais en plus ils opèrent sur des piles et par conséquent ont une durée de vie limitée.
- L'énergie limitée des capteurs est probablement la caractéristique la plus pénalisante, le plus grand des défis dans le domaine des réseaux de capteurs reste de concevoir des protocoles, entre autre de sécurité, qui minimisent l'énergie afin de maximiser la durée de vie du réseau. En d'autres mots, l'énergie est sans aucun doute la ressource qui convient pour gérer avec la plus grande attention.

15. Conclusion

Dans ce chapitre, nous avons présenté les réseaux de capteurs, en parlant sur l'architecture, composants, fonctionnement, topologies utilisés, applications, les systèmes d'exploitation ainsi que les caractéristiques et limites. Dans le chapitre suivant, nous introduirons en détail le système d'exploitation TinyOS qui est destiné aux WSN afin de faciliter l'implémentation de ces applications et détailler le langage NesC qui est utilisé pour l'implémentation de TinyOS.

Chapitre II :

Mise en place de la plateforme TinyOS

Ce chapitre décrit, le système d'exploitation TinyOS du réseau de capteurs sans fil en présentant son architecture et ses propriétés, ainsi que les outils de simulation. De plus un Choix de langage de programmation s'impose (NesC et ses caractéristiques).

1. Problématique

Les OS classiques sont généralement conçus pour un usage générique, ils sont ainsi conçus en supposant une disponibilité sans limite de ressources. Leurs objectifs est la facilité d'usage, la rapidité et l'efficacité. Parmi leurs caractéristiques, on peut citer:

- Enormes, modèle E/S.
- Architecture Multithread (Mémoire importante).
- Séparation entre espace noyau et utilisateur.
- Pas de contraintes d'énergie, ressources disponibles.

Les contraintes matérielles d'un Mote sont les suivantes :

- Ressources énergétiques basses.
- Mémoire limitée.
- CPU lente et petite taille.
- Parallélisme matériel limité.
- Communication radio et bande-passante faible.
- Portée radio courte.

Alors quelles sont les propriétés d'OS désirées ?

Pour le bon fonctionnement, l'OS doit avoir les propriétés suivantes :

- Image mémoire petite.
- Efficacité en calcul et consommation d'énergie.
- La communication est fondamentale.
- Temps-réel et construction efficace d'applications.

La problématique ici est quel est le type du système d'exploitation à choisir pour assurer les propriétés citées au dessus, ce qui nous allons voir par la suite.

2. Introduction

Parmi les domaines d'utilisation des systèmes embarqués on retrouve les réseaux de capteurs, ces derniers(les capteurs) disposent d'une alimentation autonome et leur durée de vie est limitée par celle de la batterie. Cette forte contrainte a une influence majeure sur l'ensemble des techniques mises en place pour le déploiement de tels réseaux, c'est pourquoi il faut mettre en place un système d'exploitation qui soit le plus léger possible et doit être spécialisé pour fonctionner sur ce type de réseaux.

Donc Les capteurs fonctionnent à basse tension et ceci est géré par un système d'exploitation spécialisé qui se nomme TinyOS et qui répond à la problématique citée précédemment ; il possède les caractéristiques suivantes :

- **Concurrence** : utilise une architecture orientée événement.
- **Modularité** : application composée de composants, SE + applications compilés en un seul exécutable.
- **Communication** : utilise un modèle event/command, ordonnancement FIFO non préemptif.
- **Pas de séparation** kernel/application.

TinyOS est le plus utilisé actuellement pour les réseaux de capteurs et pour le développement des applications légères, il n'existe qu'un langage de programmation capable d'interagir avec l'OS TinyOS c'est le NesC. Ce langage dédié est proche du C traditionnel mais il est orienté composants.

Dans ce chapitre, nous présenterons tout d'abord le système d'exploitation TinyOS, et ensuite nous décrirons le langage de programmation NesC.

3. Le système d'exploitation pour les RCSF : TinyOS

3.1 Historique de TinyOS

TinyOS a commencé comme projet chez Berkeley UC en tant qu'élément du DARPA Programme de NID. Il est devenu depuis sa création impliquant des milliers d'universitaires, lotisseurs et utilisateurs commerciaux dans le monde entier.

On va montrer ici l'historique de TinyOS au cours des années :

- **1999:** La première plateforme de TinyOS (WeC) et les réalisations d'OS sont développées chez Berkeley.
- **2000:** Berkeley conçoit la plateforme et les associés de rene avec Crossbow et Inc qui produit le matériel. La version 0.43 de TinyOS est rendue disponible au public par l'intermédiaire de SourceForge. Les versions Pré-1.0 de TinyOS sont un mélange de C et Perl manuscrits.
- **2001:** Berkeley développe la plateforme de mica et libère la version 0.6 de TinyOS.
- **Février 2002 :** Berkeley distribue 1000 nœuds de mica à d'autres participants au projet de NID.
- **Avril 2002 :** Le travail sur le langage de programmation de NesC a commencé en collaboration entre la recherche d'Intel et le Berkeley UC.

- **Septembre 2002 :** La version 1.0 de TinyOS, mise en application dans le NesC est libérée.
- **Août 2003 :** La version 1.1 de TinyOS est libérée et a inclut de nouveaux dispositifs de NesC comprenant la détection de course de données.
- **Septembre 2003 à décembre 2005 :** TinyOS commence un processus mineur périodique de dégagement.
- **Juin 2004 :** Le groupe de travail lance d'autres étapes pour TinyOS, basées sur des expériences mettant en communication de nouvelles plateformes. Le groupe est d'accord sur le travail de début sur 2.0.
- **Décembre 2005 :** TinyOS 1.1.15, les dernières versions de 1.1 sont libérées.
- **Juillet 2005 :** Le projet de NID conclut.
- **Février 2006 :** TinyOS 2.0 beta1 a libéré au 3ème échange de technologie de TinyOS dans Stanford, CA.
- **Juillet 2006 :** TinyOS 2.0 beta2 libéré.
- **Novembre 2006 :** TinyOS 2.0 est libéré à la conférence de SenSys à Boulder, Cie.
- **Avril 2007 :** TinyOS 2.0.1 est libéré au 4ème échange de technologie de TinyOS à Cambridge.
- **Juillet 2007 :** TinyOS 2.0.2 libéré. Travailler à TinyOS 2.1 implique de légers changements à quelques interfaces.

3.2 Présentation de TinyOS

L'objectif des environnements exécutifs est de fournir une couche d'abstraction au moment du développement, avec la possibilité d'écrire un code portable et réutilisable, sans perdre de performance au moment de l'exécution puisque l'application monolithique exploite au mieux les ressources disponibles en étant statique [GG10].

TinyOS a été créé pour répondre aux caractéristiques et aux nécessités des réseaux de capteurs, telles que :

- Une taille de mémoire réduite et une basse consommation d'énergie.
- Des opérations d'assistance intensive et robustes.
- Il est optimisé en termes d'usage de mémoire et d'énergie.

TinyOS est un système principalement développé et soutenu par l'université américaine de Berkeley en Californie et est devenu le standard de facto lequel le propose en téléchargement sous la licence BSD, une licence libre utilisée pour la

distribution de logiciels en assurant le suivi. Ainsi, l'ensemble des sources sont disponibles pour de nombreuses cibles matérielles [SA10].

TinyOS est un système d'exploitation open source spécialement conçu pour les applications embarquées et plus précisément les réseaux de capteurs sans-fil. Sa conception a été entièrement réalisée en NesC, langage orienté composant syntaxiquement proche du C [UC04].

La particularité principale de cet OS est sa taille extrêmement réduite en termes de mémoire (quelques kilo-octets seulement), grâce à une architecture basée sur une association de composants, réduisant ainsi la taille du code nécessaire à sa mise en place. Ceci permet le respect des contraintes de mémoire qu'observent les réseaux de capteurs. La bibliothèque des composants de TinyOS est particulièrement complète puisqu'on y retrouve des protocoles de réseaux, des pilotes de capteurs et des outils d'acquisition de données. L'ensemble des composants peut être utilisé tel quel ou adapté à une application précise. De plus, en s'appuyant sur une fonction événementielle, TinyOS propose à l'utilisateur un modèle de programmation orienté évènement ainsi qu'une gestion très précise de la consommation du capteur et permet de mieux s'adapter à la nature aléatoire de la communication sans fil entre les interfaces physiques [KP09].

Un programme s'exécutant sur TinyOS est constitué d'une sélection de composants systèmes et de composants développés spécifiquement pour l'application à laquelle il sera destiné (mesure de température et du taux d'humidité comme c'est le cas de notre application…etc.) [TA08].

Figure II. 1 : Signe de TinyOS

3.3 Architecture générale des cibles utilisant TinyOS

TinyOS est prévu pour mettre en place les réseaux sans fil, les équipements étudiés sont donc généralement équipés d'une radio et d'une antenne afin de se connecter à la couche physique et constituent les émissions hertziennes, on retrouve donc des équipements bardés de différents types de détecteurs et autres entrées [DN10].

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

3.4 Propriétés de TinyOS

Le fonctionnement d'un système basé sur TinyOS s'appuie sur la gestion des évènements. Ainsi, l'activation des 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 s'oppose au fonctionnement dit temporel où les actions du système sont gérées par une horloge donnée [YM08].

TinyOS a été programmé en langage NesC, 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.

Lorsqu'un système est dit « temps réel » celui-ci gère des tâches caractérisées par des priorités et par des échéances à respecter dictées par l'environnement externe. Dans le cas d'un système strict, aucune échéance ne tolère des dépassements contrairement à un système temps réel mou. TinyOS se situe au-delà de ce second type car il n'est pas prévu pour avoir un fonctionnement temps réel.

TinyOS a été conçu pour réduire au maximum la consommation en énergie du capteur. Ainsi, lorsqu'aucune tâche n'est active, il se met automatiquement en veille.

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

Tableau II. 1 : Les propriétés de TinyOS

Donc, TinyOS est basé sur quatre grandes propriétés (présentées ci-dessous) qui font que ce système d'exploitation s'adapte particulièrement bien aux systèmes à faible ressources :

- Disponibilité et sources.
- Event-driven(Evènementiel).
- Langage : TinyOS a été programmé en langage NesC.
- Non préemptif et non temps réel.

- Réduction d'énergie.

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

- ✓ **Les évènements :** ils sont utilisés pour réaliser de petits processus. De plus ils peuvent interrompre les tâches qui sont exécutées.
- ✓ **Les tâches :** les tâches sont pensées pour réaliser une plus grande quantité de traitements et elles ne sont pas critiques dans le temps. Les tâches sont exécutées complètement, mais l'initialisation et la terminaison d'une tâche sont des fonctions séparées [JW94].

3.5 Allocation de la mémoire

Il est très important d'aborder la façon avec laquelle un système d'exploitation gère la mémoire et plus spécialement quand celui-ci travaille dans un espace restreint. TinyOS ne nécessite pas beaucoup de place mémoire puisqu'il n'a besoin que de 300 à 400 octets dans le cadre d'une distribution minimale [LD08]. Il est nécessaire d'avoir 4 Ko de mémoire libre qui se répartissent entre les différents besoins suivants :

- **pile** : Elle sert de mémoire temporaire pour l'empilement et le dépilement des variables locales qui sont sauvegardées sur la pile (stack) et déclarées dans une méthode.
- **La mémoire libre** : Pour tout le reste du stockage temporaire. La notion d'allocation dynamique de mémoire n'est pas présente dans le système, ce qui simplifie l'implémentation mais, par ailleurs, il n'existe pas de mécanisme de protection de la mémoire, ce qui rend le système plus vulnérable au crash et aux corruptions de mémoire [HR08].
- **Les variables globales** : Elles réservent un espace mémoire pour stocker les valeurs pouvant être accessibles depuis différentes tâches. Les variables globales sont disponibles et assurent la conservation de la mémoire et l'utilisation de pointeurs.

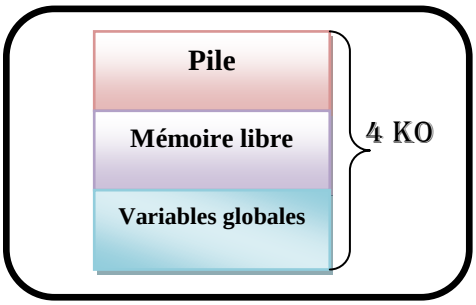


Figure II.2 : Organisation de mémoire dans TinyOS

3.6 Allocation de ressources

3.6.1 L'ordonnanceur

Le choix d'un Ordonnanceur déterminera le fonctionnement global du système et le dotera de propriétés précises telle que la capacité à fonctionner en temps réel qui effectue la gestion des tâches et des évènements du système [JM11], il comporte :

- Deux niveaux de priorité : bas pour les tâches et haut pour les évènements.
- une file d'attente FIFO disposant d'une capacité de 7 places.

On a un niveau de priorité entre les tâches pour permettre de se classer.

Lors de l'arrivée d'une nouvelle tâche, celle-ci sera placée dans la file d'attente en fonction de sa priorité. Dans le cas où la file d'attente est pleine, la tâche dont la priorité est la plus faible est enlevée de la file FIFO.

3.6.2 Package TinyOS

TinyOS est prévu pour fonctionner sous plusieurs plates-formes comme 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 et la version de développement.

- **la version stable (v. 1.1.0) :** elle présente moins de risques mais elle est nettement moins récente.
- **la version actuellement en cours de test (v. 1.1.15) :** nécessite l'installation de l'ancienne version.

3.7 Structure logicielle

L'OS TinyOS se base sur le langage NesC qui lui propose une architecture basée sur des composants, permettant de réduire considérablement la taille mémoire du système et de ses applications. Chaque composant correspond à un élément matériel (LEDs, timer, ADC …) et peut être réutilisé dans différentes applications. Ces applications sont des ensembles de composants associés dans un but précis [RK11].

Les composants peuvent être des concepts abstraits ou bien des interfaces logicielles aux entrées-sorties matérielles de la cible étudiée (carte ou dispositif électronique). L'implémentation des composants s'effectue en déclarant les tâches, les commandes ou les évènements.

Les tâches sont utilisées pour effectuer la plupart des blocs d'instructions d'une application. A l'appel d'une tâche, celle-ci va prendre place dans une file d'attente de type FIFO pour y être exécutée. Comme nous l'avons vu, il n'y a pas de mécanisme de

préemption entre les tâches et une tâche activée s'exécute en entier. Par ailleurs, lorsque la file d'attente des tâches est vide, le système d'exploitation met en veille le dispositif jusqu'au lancement de la prochaine.

Les évènements sont prioritaires par rapport aux tâches et peuvent interrompre la tâche en cours d'exécution. Ils permettent de faire le lien entre les interruptions matérielles (pression d'un bouton, changement d'état d'une entrée, …) et les couches logicielles que constituent les tâches [MK09].

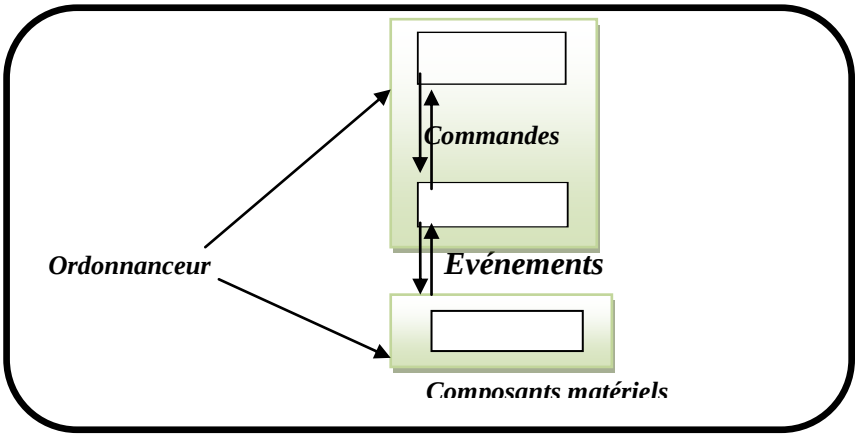


Figure II.3 : Schéma des interactions internes au système TinyOS

4. Les outils de simulation

TinyOS offre des outils de Simulation : TOSSIM, PowerTOSSIM, TinyViz.

4.1 TOSSIM

TOSSIM permet de simuler le comportement d'un capteur au sein d'un réseau de capteurs, il est un simulateur discret basé sur la programmation par événement, de même qu'il est conçu et désigné pour simuler les réseaux de capteurs qui utilisent la plateforme TinyOS. Le principal but de TOSSIM est de créer une simulation très proche de ce qui ce passe dans ces réseaux dans le monde réel.

TOSSIM simule le comportement des applications de TinyOS à un niveau très bas. Le réseau est simulé au niveau des bits et chaque interruption dans le système est capturée. TOSSIM fourni deux modèles de radios pour la communication : Le modèle par défaut « simple » où les paquets sont transmis dans le réseau sans aucune erreur et ils sont reçus par chaque nœud. Avec ce modèle il est ainsi possible que deux nœuds différents peuvent envoyer un paquet en même temps avec la conséquence que ces deux paquets seront alors détruit à cause du chevauchement des signaux. Le deuxième modèle est le modèle « lossy », dans ce modèle les nœuds sont placés dans un graphe

direct formé d'un couple (a, b) ce qui signifie qu'un paquet envoyé par le nœud a peut être été reçu par le nœud b [WZ06].

TOSSIM est équipé aussi d'un simulateur graphique TinyViz. Cette application est équipée par plusieurs API plugins qui permet d'ajouter plusieurs fonctions à notre simulateur comme par exemple contrôler les entrées de notre radio ou bien suivre la dépense d'énergie en utilisant un autre simulateur qui s'appelle PowerTOSSIM [BA11].

4.2 PowerTOSSIM

L'outil PowerTOSSIM permet de faire des simulations de la même manière que TOSSIM sauf que celui-ci prend en considération la consommation d'énergie, ainsi le nœud qui ne possède plus d'énergie s'arrête de fonctionner, ce qui nous permet d'exécuter la simulation jusqu'à la mort du réseau [MD07].

4.3 TinyViz

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

5. Langage de programmation NesC

5.1 Présentation

NesC est un langage conçu pour incarner les concepts structurants et le modèle d'exécution de TinyOS. C'est une extension du langage C orientée composant ; il supporte alors la syntaxe du langage C et il est compilé vers le langage C avant sa compilation en binaire.

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

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

Premièrement, les applications NesC sont construites à partir de composants ayant des interfaces bidirectionnelles bien définies, deuxièmement NesC définie un modèle de concurrence basé sur les notions de tâches, les "handlers" d'événements matériels, et la détection des conditions d'accès concurrent aux données au moment de la compilation [CY08].

5.2 Concepts et définitions

Application NesC = ensemble de composants

- **Composant :** l'unité de code de base de NesC est le composant, component en anglais. Un composant exécute des commandes, lance des Events, dispose d'un Frame pour stocker l'état local et utilise la notion de Tasks pour gérer la concurrence. Il implémente des interfaces utilisées par d'autres composants pour se communiquer, il possède deux types: modules et configurations, une application NesC consiste en un ou plusieurs composants assemblés pour former un exécutable.
- **Interface d'un composant :** un composant offre et utilise des interfaces. Ces interfaces sont l'unique point d'accès au composant et sont bidirectionnelles. Une interface définie un ensemble de fonctions appelées commandes (qui doivent être implémentées par le composant qui offre cette interface), et un autre ensemble de fonctions appelées événements (qui doivent êtres implémentées par le composant utilisant l'interface).

Pour qu'un composant puisse appeler les commandes d'une interface il doit implémenter l'événement définit dans l'interface. Un même composant pourra offrir et utiliser plusieurs interfaces différentes ou plusieurs instances d'une même interface.

- **Modules et configurations :** il existe deux types de composants dans NesC: modules et configurations. Les modules définissent le code de l'application en implémentant une ou plusieurs interfaces.

Les configurations sont utilisées pour assembler d'autres composants en reliant les interfaces utilisées par des composants aux interfaces offertes par d'autres composants. Ceci est appelé "wiring" (câblage). Toute application NesC est décrite par une configuration qui assemble les composants de l'application.

- **L'extension de fichier ".nc" :** NesC utilise l'extension de fichier ".nc" pour tous les fichiers sources, interfaces, modules, et configurations.

- **Modèle de concurrence :** TinyOS exécute seulement un programme consistant d'un ensemble de composants requis par l'application. Il existe deux chemins d'exécution : les tâches, et les "handlers" d'événements matériels.

✓ **Tâche :** Les tâches sont des fonctions dont l'exécution est différée. Une fois lancée, une tâche s'exécute jusqu'à sa fin et ne peut pas interrompre une autre tâche.

✓ **Handler d'événement matériel :** Les handlers d'événements matériels sont exécutés en réponse à une interruption matérielle et s'exécutent aussi jusqu'à leurs fins, mais peuvent interrompre l'exécution d'une tâche ou d'un autre Handler.

Les commandes et les événements qui s'exécutent dans un Handler d'événement matériel doivent être déclarés avec le mot clé async.

- **Complément : c**omme les tâches et les Handlers sont préemptifs par d'autres codes asynchrones, les programmes NesC son susceptibles à certaines conditions de concurrences d'accès aux données.

Pour éviter ces conditions d'accès concurrents, il faut soit accéder aux données partagées exclusivement à travers des tâches, soit en ayant tous les accès aux données partagées dans des instructions "atomic".

Le compilateur signale les accès potentiellement concurrents aux données partagées au programmeur. Il se peut que le compilateur signale des faux positifs .Dans ce cas, une variable peut être déclarée avec le mot clé norace.

Mais attention : le mot clé norace doit être utilisé avec une extrême précaution.

5.3 Développement

5.3.1 L'interface

Une interface définit les interactions entre deux composants, les interfaces sont bidirectionnelles, elles spécifient un ensemble de fonctions à implémenter par les commandes (composants fournisseurs de l'interface), et un ensemble à implémenter par les évènements (composants utilisateurs de l'interface) [DG03].

Afin de distinguer les fonctions concernant un évènement de celles concernant une commande, les en-têtes des fonctions sont précédés par les mots clés respectifs event ou command, donc Les interfaces sont des fichiers décrivant les commandes et évènements proposés par le composant qui les implémente.

La différence entre les commandes et les évènements est que Les commandes font typiquement des appels du haut vers le bas (des composants applicatifs vers les composants plus proches du matériel), alors que les évènements remontent les signaux du bas vers le haut.

✓ Voici un exemple simple d'interface :

```
interface SendMsg {
command result_t send(uint16_t address, uint8_t length,TOS_MsgPtr msg);
event result_t sendDone(TOS_MsgPtr msg, result_t success);
}
```

✓ Voici deux exemples, l'un pour Appeler une commande et l'autre pour signaler un événement :

- Appeler une commande :

```
call Send.send(1, sizeof(Message), &msg);
```

- Signaler un évènement :

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

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

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

5.3.2 Module

Un module contient le code d'un composant élémentaire et implémente une ou plusieurs interfaces et peut les utiliser. Cette partie du code est généralement plus étendue et c'est dans celle-ci que l'on programme réellement le comportement qu'on souhaite voir réalisé par l'application. Cette partie là est à son tour divisée en trois sous-sections : Uses, Provides, Implementation.

- La syntaxe d'un module est la suivante :

```
module NomModuleM {
{
provides {
//liste des interfaces fournies
Exemple : interface NomInterfaceFournie1 ;}
uses {
//liste des interfaces requises, ex :
interface NomInterfaceRequise1 ;}
}
implementation {
//déclarations des variables, exemple :
Int state ;
//implémentations des fonctions décrites par les
interfaces fournies// ;}
```

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

La sous-section uses informe le compilateur que nous allons faire usage d'une interface (on pourra donc effectuer des appels aux méthodes de cette interface). Pour faire cela, on a besoin de respecter quelques règles : si nous utilisons une interface, il nous faut avoir dans la section implementation un lien (''wiring'') reliant cette interface avec un composant qui la fournit. Finalement, utiliser une interface oblige implicitement à gérer les évènements pouvant se produire du fait d'avoir utilisé cette interface précise.

La sous-section implementation, est celle qui contiendra toutes les méthodes nécessaires pour fournir le comportement souhaité à notre composant ou à notre application. Cette sous-section doit contenir au moins : les variables globales que va utiliser notre application, les fonctions qu'elle doit mettre en œuvre pour les interfaces qui nous fournissons et les évènements qu'elle doit mettre en œuvre venant des interfaces que nous utilisons.

5.3.3 La configuration

Les interfaces du composant utilisateur sont reliées aux mêmes interfaces du composant fournisseur.

Il existe deux possibilités de connexion dans NesC:

- endpoint1 = endpoint2.
- endpoint1 -> endpoint2.

Les éléments connectés doivent être compatibles : "Interface" à "Interface", "Command" à "Command", "event" à "event". Il faut toujours connecter un utilisateur d'une interface à un fournisseur de l'interface.

La configuration se fait de la manière suivante :

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

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

5.3.4 Types de 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 valeurs SUCCESS et FAIL,c'est un retour de fonction.
- bool : valeur booléenne qui peut être TRUE ou FALSE.

5.3.5 Types de fonctions en NesC

En NesC les fonctions peuvent être de types très variés. Il y a d'abord les fonctions classiques avec la même sémantique qu'en C et la façon de les invoquer est aussi la même qu'en C. Il y a aussi des types supplémentaires de fonctions : task, event, command. Les fonctions command sont principalement des fonctions qui sont exécutées de manière synchrone [HA08], c'est-à-dire que lorsqu'elles sont appelées elles sont exécutées immédiatement. La manière d'appeler ce genre de fonction est :

```
Call interface.nomFonction
```

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

```
post interface.nomTâche
```

Immédiatement après son invocation, l'exécution du programme qui l'a invoqué se poursuit. Les fonctions event sont des fonctions qui sont appelées quand on relèvera

un signal dans le système, elles possèdent principalement la même philosophie que la programmation orientée évènements, de sorte que, lorsque le composant reçoit un évènement, on effectue l'invocation de cette fonction. Il existe une méthode pour pouvoir invoquer manuellement ce type de fonctions :

```
signal interface.nomEvenement
```

5.4 Construction d'une application en NesC

Toutes les applications ont besoin d'un fichier de configuration de haut niveau qui est normalement nommé d'après l'application elle-même. Dans ce cas, **NomApplication.nc** est le fichier de configuration de l'application et le fichier source que le compilateur NesC utilise pour générer l'exécutable. **NomApplicationM.nc** quant à lui correspond à l'implémentation de l'application.

Comme nous l'avons expliqué dans la partie précédente, les composants NesC peuvent se relier les uns aux autres, et ici c'est le fichier **NomApplication.nc** qui permet de faire cette connexion entre le module **NomApplicationM.nc** et les autres composants auxquels l'application fait appel.

Dans une application, nous avons donc une configuration et un module qui vont ensemble. La convention de TinyOS veut alors que **NomApplication.nc** représente la configuration et **NomApplicationM.nc** représente le module correspondant, il faut donc respecter cette convention pour utiliser le Makefile livré avec la distribution de TinyOS.

5.5 Compilation et exécution d'une application en NesC

Le compilateur traite les fichiers NesC en les convertissant en un fichier C très grand, ce fichier contiendra l'application et les composants de l'OS utilisés par l'application. Ensuite un compilateur spécifique à la plateforme cible compile ce fichier C, il deviendra alors un seul exécutable puis le chargeur installe le code sur le mote comme par exemple Mica2, Telos …etc.

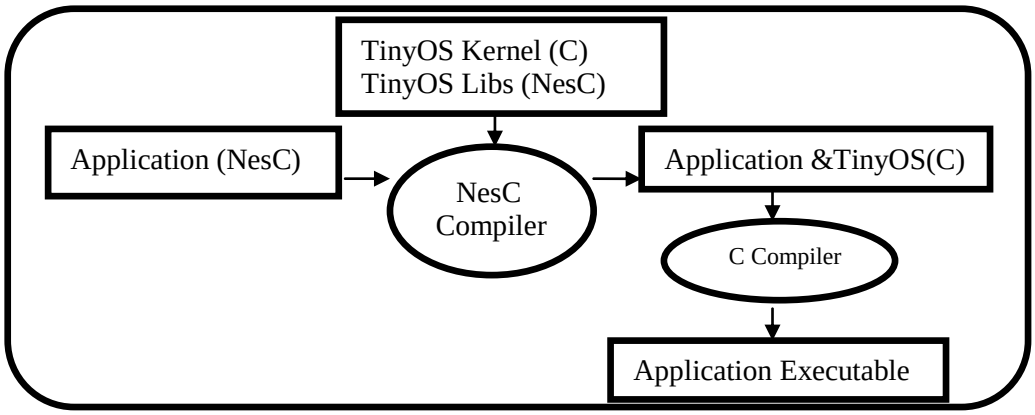


Figure II. 4 : processus de compilation

6. Commentaires sur NesC et TinyOS

Plusieurs remarques peuvent être faites sur TinyOS et le langage NesC au cœur de ce système événementiel. Tout d'abord on ne constate que la conception d'un code séquentiel simple qui nécessite l'emploi d'une structure rigide et complexe.

Certes, la modularité est forte ce qui favorise l'abstraction et la réutilisabilité du code. La programmation modulaire laisse entendre une réutilisabilité forte des modules pour concevoir des applications rapidement, cela est effectivement très simple dans le cas d'une réutilisation des plates-formes développées par les concepteurs de TinyOS (plate-forme Mica2, MicaZ, etc.).

Dans le cas d'un portage de TinyOS vers d'autres plates-formes, cela devient alors plus difficile parce que la programmation en NesC événementielle est particulièrement ardue. Les traitements doivent le plus souvent être décrits sous forme de machines d'états ce qui n'est pas naturel. Pour illustrer cela, un code de modules par exemple, qui ne fait qu'invoquer de façon séquentielle trois sous-procédures, nécessite une machine d'état complète et le code résultant est très lourd.

De plus, une application relativement simple nécessite tout de même un grand nombre de composants simples et le nombre d'interfaces reliant ces composants est rapidement très grand.

NesC est un langage de programmation qui présente de grands avantages pour le développement d'applications pour des systèmes embarqués, et particulièrement pour les réseaux de capteurs sans fil. Celui-ci apporte des outils pour une bonne programmation d'applications, d'abord en abstrayant les caractéristiques de modularité que présentent les réseaux de capteurs, et en suite en facilitant la mise en œuvre de programmes pour ces systèmes.

Le compilateur qui génère un code exécutable à partir de ce langage de description NesC effectue des opérations automatiques qui ne peuvent permettre d'explorer des optimisations mémoire les plus fines. C'est la seconde approche qu'on peut faire à TinyOS : une sous-optimalité qui est due à l'utilisation d'un langage de description intermédiaire transformé de façon automatique par un compilateur.

Nous considérons que TinyOS est fortement lié aux plateformes de la société Crossbow ce qui leur permet d'offrir un package matériel et logiciel flexible pour l'étude de mécanismes de haut-niveau, mais nous pensons que la rigidité de la programmation et le fait de se reposer sur la qualité du compilateur pour l'obtention de

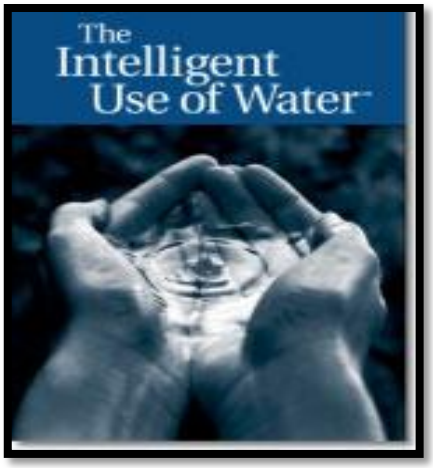
codes performants n'est pas le meilleur point de départ pour la conception d'un système extrêmement contraint en mémoire et en particulier en mémoire RAM.

7. Conclusion

Dans ce chapitre, nous avons présenté le système d'exploitation TinyOS, avec ses propriétés, caractéristiques, sa structure logicielle, et nous avons parlé un peu sur l'allocation de mémoire et de ressources et les simulateurs utilisés par la plateforme TinyOS pour créer une simulation très proche de ce qui ce passe dans le monde réel. Nous avons également présenté le langage de programmation NesC qui est créé pour la conception des applications embarquées et plus précisément les réseaux de capteurs sans fil et enfin quelques commentaires sur TinyOS et le NesC. Dans le chapitre suivant, nous allons étudier le contexte de notre travail de projet de fin d'étude. Nous aborderons l'irrigation intelligente et ses avantages dans l'économie de l'eau.

Chapitre III:

L'irrigation intelligente pour économiser l'eau



Dans ce chapitre on présente l'objectif de notre application, l'irrigation intelligente pour économiser l'eau, des exemples de système utilisé pour l'irrigation automatique ainsi que des statistiques réelles de la récupération de l'eau.

1. Introduction

L'eau c'est ce qui garde le monde en vie, l'importance de l'eau pour la vie humaine n'est plus à démontrer car il est parmi les composants essentiels dans l'équilibre de l'écosystème dans le monde. Cette ressource qui répond aux besoins fondamentaux de l'homme est un élément clé du développement, en particulier pour générer et entretenir la prospérité par le biais de l'agriculture, de la pêche, de la production d'énergie, de l'industrie, des transports et du tourisme. En outre, l'eau est vitale pour tous les écosystèmes du monde.

On tente de présenter dans ce chapitre l'irrigation intelligente en montrant l'intérêt de l'eau et le besoin de ce dernier selon les sols, l'analyse des espaces verts, ainsi que la présentation d'un exemple de système utilisé dans l'irrigation intelligente.

2. Vue d'ensemble: la crise mondiale de l'eau

A première vue, l'eau semble la ressource la plus abondante sur terre. Mais en réalité, l'eau douce ne représente que 1% de toute l'eau terrestre. Les 99% restants sont indisponibles pour l'utilisation humaine, car on les retrouve sous forme d'eau de mer ou d'eau saumâtre, de neige ou de glace. Alors que les ressources en eau douce ne sont pas extensibles, la demande explose sous l'effet d'une croissance rapide de la population mondiale qui puise dans les réserves de la planète, et d'une augmentation exponentielle de la consommation.

Le problème ne se limite pas aux pays en voie de développement. Aux Etats-Unis, pourtant moins densément peuplés, l'offre peine aussi à satisfaire la demande, en raison notamment d'une consommation très élevée liée au mode de vie américain.

Depuis 1900, la population des Etats-Unis a doublé, mais la consommation d'eau par personne a été multipliée par huit, car la technologie et l'amélioration du niveau de vie ont entraîné une multiplication par deux de la consommation d'eau tous les 20 ans. Aujourd'hui, les Américains utilisent en moyenne 382 litres d'eau par personne et par jour, dépassant de beaucoup le minimum de 78 litres par jour estimé nécessaire pour les besoins vitaux, l'hygiène et la production de nourriture [CE02].

Dans le livre blanc « L'irrigation pour un monde en croissance » publié par Rain et Bird, des options telles que le dessalement, la ré-tarification, le recyclage de l'eau et l'amélioration de l'infrastructure et des systèmes de distribution d'eau requièrent une mobilisation des pouvoirs publics et des organisations internationales.

Dans de nombreux cas, les technologies ne sont pas encore suffisamment au point pour pouvoir être utilisées actuellement de manière efficace.

Il y a cependant une chose que vous pouvez déjà faire pour aider à résoudre cette crise mondiale en pleine croissance: économiser l'eau grâce à une irrigation efficace.

3. Economie d'eau des ménages: un élément de solution

Au début, les initiatives d'économie d'eau des ménages ont visé des réductions de consommation à l'intérieur des habitations, telles que la mise au point de modèles de WC plus économes dans les années 1960. Des études ont révélé que les chasses d'eau consommaient jusqu'à 50% du budget des ménages consacré à l'eau. Une décennie plus tard, l'étalement généralisé des villes aux Etats-Unis et les pénuries d'eau résultantes ont incité les sociétés de distribution d'eau à imposer de nouvelles mesures d'économie d'eau domestique et mener de grandes campagnes d'éducation.

La prise de conscience de la nécessité d'économiser l'eau aussi à l'extérieur, et l'organisation de campagnes de sensibilisation en ce sens, sont plus récentes.

Aujourd'hui encore, la plupart des propriétaires connaissent mieux "les bonnes pratiques" d'économie d'eau intérieures, telles que l'utilisation de WC à chasse économique, de douches à pompe économique ou de lave-vaisselles ou machines à laver économiques, que les mesures d'économie d'eau extérieures.

Etant donné que l'irrigation du jardin peut représenter 20 à 50% de la consommation annuelle de 359.614 litres d'eau d'un ménage américain moyen, la réduction de l'arrosage peut contribuer de manière importante à la résolution des problèmes de pénurie d'eau.

Ceci dit, les particuliers ont souvent bien du mal aujourd'hui à concilier les impératifs d'économie d'eau et leur désir d'avoir un beau jardin. Pour beaucoup, un espace vert favorisant l'économie d'eau évoque des images de sable, de rochers et de cactus, voire même de béton. Cette conception minimaliste constitue peut-être le summum de l'économie d'eau, mais ne fait généralement pas partie des choix envisageables par les personnes concernées, pour des raisons de climat ou de préférences personnelles.

4. Le livre blanc « L'irrigation pour un monde en croissance »

Le Guide du Particulier pour une irrigation efficace des espaces verts, leur apporte des informations pratiques sur les moyens d'économiser l'eau grâce à des techniques d'irrigation plus performantes. Si l'on tient compte de tous les aspects d'une utilisation

optimale de l'eau, depuis le choix de l'heure d'irrigation et des volumes d'eau à appliquer jusqu'à l'utilisation de systèmes d'irrigation plus économes, on voit qu'il est possible de garder tous les avantages d'un espace vert bien arrosé en consommant nettement moins d'eau.

5. L'eau dans le sol

Sous nos climats, l'apport d'eau au sol se fait sous forme de pluie, neige, rosée et brouillard. Toute l'eau des précipitations n'atteint pas le sol: une part est évaporée directement pendant et après la pluie; les gouttes peuvent être interceptées en partie par le feuillage. L'eau qui atteint le sol ruisselle, s'infiltre et ré humecte le sol. Les racines absorbent cette eau que la tige et les feuilles évaporent par transpiration. Une fraction réduite finalement gagne la profondeur et atteint la nappe. Un profil habituel de la quantité d'eau contenu dans une coupe du sol et du sous-sol habituel de la quantité d'eau contenu dans une coupe du sol et du sous-sol montre une augmentation de la teneur en eau avec la profondeur [JB03].

6. Besoin d'eau selon les sols

Le type de sol est un facteur important à considérer lors de la conception et de la gestion d'un système d'irrigation. Un sable faiblement pourvu en matière organique retiendra à peine 5% de son poids en eau. Par contre, un sol limoneux pourra en retenir prés de 30 %. Cette quantité d'eau retenue influence directement sur les fréquences d'irrigation. Par exemple, une argile aura assez d'eau en réserve en début de saison pour approvisionner une culture pendant un mois, alors qu'un sable grossier n'aura de réserve que pour quelques jours.

La matière organique joue un rôle déterminant dans la rétention d'eau. Comme une éponge, elle retient l'eau des précipitations pour la restituer à la demande des besoins à la culture [AW09].

7. Analyse des espaces verts

La conception correcte d'un espace vert dépend pour beaucoup d'une analyse correcte de ses différentes zones. Les systèmes d'irrigation les plus économiques divisent le terrain en zones d'irrigation distinctes, correspondant à des végétations dont les besoins en eau sont différents.

De nombreux jardins comprennent par exemple des pelouses, des parterres de fleurs, des massifs, des arbres et même des plantes en pot, chacun de ces types de végétation a des besoins en eau différents, et doit être traité comme une unité séparée.

Des différences d'ensoleillement (zone ombragée ou plein soleil) influencent aussi les besoins d'irrigation.

Le gazon et certaines autres plantes ont généralement besoin d'un apport d'eau plus important pour rester en bonne santé. Diviser un espace vert en différentes zones évite d'imposer les besoins du gazon à toute la végétation et de donner ainsi trop d'eau aux massifs et aux arbres, et réduit d'autant la consommation d'eau totale.

On néglige trop souvent de tenir compte de caractéristiques naturelles préexistantes telles que le relief du terrain et la nature du sol plus ou moins perméable, rocheuse, sablonneuse ou argileuse. Tenir compte de la perméabilité du sol et de l'écoulement naturel de l'eau à travers le terrain permet d'inclure des zones d'irrigation adaptées à la compensation des défauts de drainage.

8. Systèmes d'irrigation automatiques

Les systèmes d'irrigation automatiques sont très pratiques: lorsqu'ils sont bien réglés, ils fournissent juste la quantité d'eau requise, au bon endroit, pratiquement sans effort pour l'utilisateur. La plupart des systèmes automatiques combinent différentes méthodes d'application de l'eau. Il s'agit principalement d'arroseurs escamotables, qui se rétractent dans le sol quand l'irrigation est terminée, et de dispositifs d'irrigation localisée tels que des goutteurs et micro-asperseurs, qui débitent l'eau plus lentement, à l'endroit où la plante en a le plus besoin: au niveau du sol, juste au-dessus des racines [Al06].

L'irrigation ou bien l'arrosage à la main, encore pratiqué par de nombreux particuliers au moyen de tuyaux perforés ou d'arroseurs à déplacer raccordés à un tuyau d'arrosage, ne permet pas d'adapter le débit avec précision à la capacité d'absorption du sol. Beaucoup d'utilisateurs ouvrent le débit au maximum et gaspillent l'eau par une application excessive. Le surplus, qui ne peut être absorbé, est entraîné par ruissellement vers les égouts et les écoulements pluviaux. Ces méthodes entraînent généralement une irrigation excessive du terrain, l'eau est gaspillée par évaporation, ruissellement, ou simplement par l'apport d'une quantité d'eau supérieure à ce qui est nécessaire pour garder la plante en bonne santé.

Un grand avantage des systèmes d'irrigation automatique est la possibilité de fournir des volumes d'eau différents à des plantes ayant des besoins différents, à un débit adapté pour permettre l'absorption de cette eau.

Les systèmes les plus économiques peuvent combiner des conduites enterrées et des rampes d'irrigation localisée, en vue notamment d'une irrigation multizones.

Les parterres de fleurs peuvent ainsi se trouver dans une zone nécessitant moins d'eau qu'un gazon, où la meilleure solution sera une rampe d'arrosage localisé équipée de goutteurs à faible débit, tandis que l'irrigation du gazon s'effectuera plutôt au moyen de tuyères ou d'arroseurs à turbine.

Quelles que soient les économies d'eau autorisées en théorie par la conception d'un système d'irrigation, les économies réelles dépendront en grande partie d'une installation et d'une gestion correcte du système. Le gaspillage d'eau dû à des systèmes d'arrosage inefficaces et à des programmes d'irrigation inadéquats peut atteindre 30% des volumes appliqués sur les pelouses et la végétation.

Un système d'irrigation automatique efficace, qu'il soit de type enterré ou localisé ou une combinaison des deux, garantit l'apport précis de la quantité d'eau requise dans chaque zone distincte du terrain. Un tel système est constitué d'un certain nombre d'éléments différents.

9. Exemple d'un système utilisé pour l'irrigation automatique

S.Sense est un système sans-fil qui fournit les informations nécessaires pour contrôler l'irrigation automatique. Il se compose d'un capteur placé dans le sol et d'un contrôleur connecté au système d'irrigation, Le capteur contrôle en permanence l'état du sol pour déterminer à quel moment l'irrigation est nécessaire, et quelle quantité doit être appliquée. Le système est entièrement automatisé, il n'est pas utile de l'arrêter lorsqu'il pleut ou de regarder la végétation mourir en cas de forte chaleur. S.Sense déclenchera même une alarme s'il détecte une tête d'arrosage cassée ou obstruée, de sorte que le problème puisse être réparé avant qu'une perte substantielle se produise.
Pour installer ce système on utilise certains dispositifs qui sont les suivants :
Système sans-fil facile à installer, sonde diélectrique d'humidité, réseau sans-fil extensible qui fonctionne n'importe où (espaces verts, pelouses, etc.) et se branche sur les systèmes neufs ou existants.

N'importe qui peut installer un système S.Sense en quelques minutes avec juste un tournevis. Le système peut également fournir un contrôle souple des zones d'irrigation par le réseau des sondes. Le réseau sans-fil permet à chaque sonde de transmettre par relais des messages à d'autres sondes, ainsi le système est exploité sur de grandes surfaces.

Voici deux exemples [P04] :

Produit 1 :

S.Sense Starter Kit

Le "**S.Sense Starter Kit**" contient tout ce dont nous avons besoin pour commencer à économiser l'eau et de l'argent. Il contient une sonde "S.Sense Sensor Probe" que nous insérons dans le sol et un contrôleur "S.Sense Controller" qui se relie à notre système d'irrigation existant.

Contenu :

1- S.Sense Sensor Probe.
2- S.Sense Controller.
3- 3 piles AAA Alcalines.
4- 1 pile AA au Lithium.
5- Un guide d'installation.

Produit 2 :

Sonde S.Sense Sensor Probe

Pour de plus grands systèmes ou des zones complexes d'irrigation, des sondes additionnelles S.Sense Probe Sensor peuvent être ajoutées. Chaque sonde que nous ajoutons fournit un contrôle plus précis de différentes zones d'irrigation.

Caractéristiques :

1- Contrôle le taux d'humidité et la température.
2- Fréquence : 916MHz.
3- Distance maxi entre chaque sonde : environ 60m.
4- Type de réseau : maillé.
5- Alimentation : 1 pile AA (durée de vie moyenne : 1 an).
6- Dimensions : 26 cm de haut par 4cm de diamètre

10. Statistiques des superficies irriguées par la direction des services d'agriculture dans la wilaya de Tlemcen en 2011

Dans cette section, nous allons montrer trois tableaux qui représentent des statistiques sur les superficies irriguées dans la wilaya de Tlemcen ainsi leur interprétation, à noter que la SAT est de 901 769 ha et la SUT dans la wilaya de Tlemcen est de 352 610 ha, ce qui représente la moitié de la surface totale plus quelques hectares.

Le tableau 1 nous montre les superficies irriguées par types de cultures et systèmes d'irrigation, chaque culture à des types précis de systèmes. A Tlemcen, on trouve généralement cinq types de cultures : Maraichers pour les cultures des légumes, Arboricultures pour les cultures des arbres fruitiers, Fourragères pour les cultures des

plantes utilisés pour la nourriture du bétail, les cultures industrielles et enfin les cultures céréales. Ainsi, on trouve trois systèmes d'irrigation : Gravitaire, Aspersion et localisé.

Tableau 1 :

Types de Cultures / Systèmes d'irrigation	**Cultures Maraichers (ha)**	**Arboricultures (ha)**	**Cultures Fourragères (ha)**	**Cultures Industrielles (ha)**	**Céréales (ha)**	**Autres Cultures (ha)**	**Surface Irriguée Totale (ha)**
Gravitaire	6630	4284	0	0	0	0	10914
Aspersion	4000	0	630	0	2055	0	6685
Localisé	0	5300	0	0	0	865	6165
Surface Totale	10630	9584	630	0	2055	865	23764

Tableau III. 1 : Superficies irriguées par types de cultures et système d'irrigation

La figure 3.1 suivante nous illustre les données du tableau 3.1

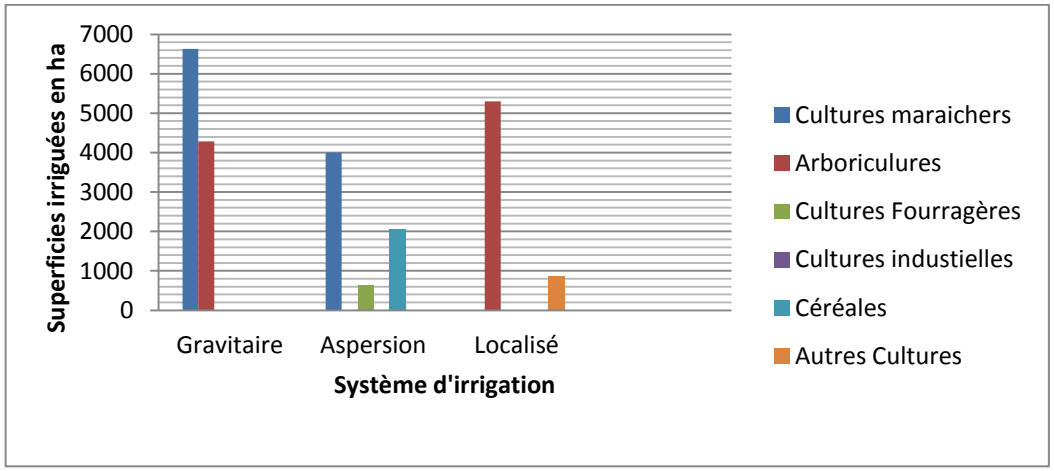


Figure III. 1: Les superficies irriguées par types de cultures et systèmes d'irrigation

D'après la figure III.1, les cultures maraichères utilisent beaucoup plus le système gravitaire et ils utilisent aussi l'irrigation par aspersion. Le système localisé est utilisé par l'arboriculture en une forte proportion suivi par le système gravitaire, par contre les cultures fourragères et céréales n'utilisent que le système par aspersion. On trouve aussi d'autres cultures qui emploient le système localisé. Les cultures industrielles n'utilisent aucun système, le tableau III.2 nous montre les superficies irriguées totales de chaque système et la somme des surfaces en hectares.

Systèmes d'irrigation	**Gravitaire (ha)**	**Aspersion (ha)**	**Localisé (ha)**	**SurfaceTotale (ha)**
Superficies irriguées	10914	6685	6165	23764

Tableau III. 2 : Superficies irriguées par système d'irrigation

La répartition en pourcentage des surfaces irriguées présentée par la figure 3.2 est la suivante :

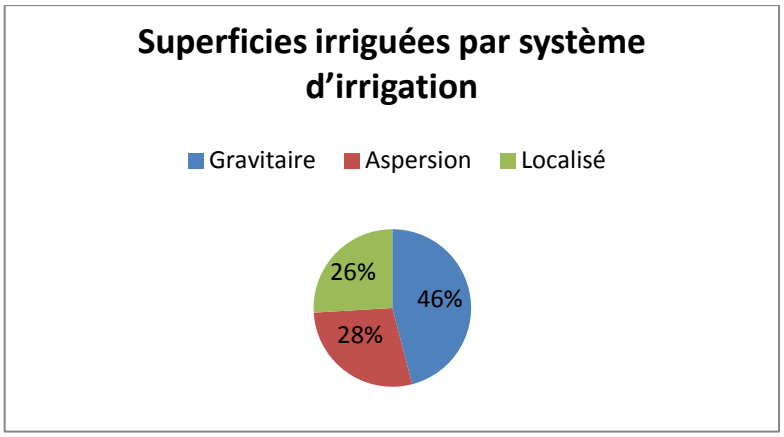


Figure III. 2 : Taux de superficie irriguée par type d'irrigation.

D'après le secteur le système gravitaire est le plus utilisé, on irrigue 46% de la surface utile ensuite le système d'aspersion (28%), le système localisé est le moins utilisé (26%).

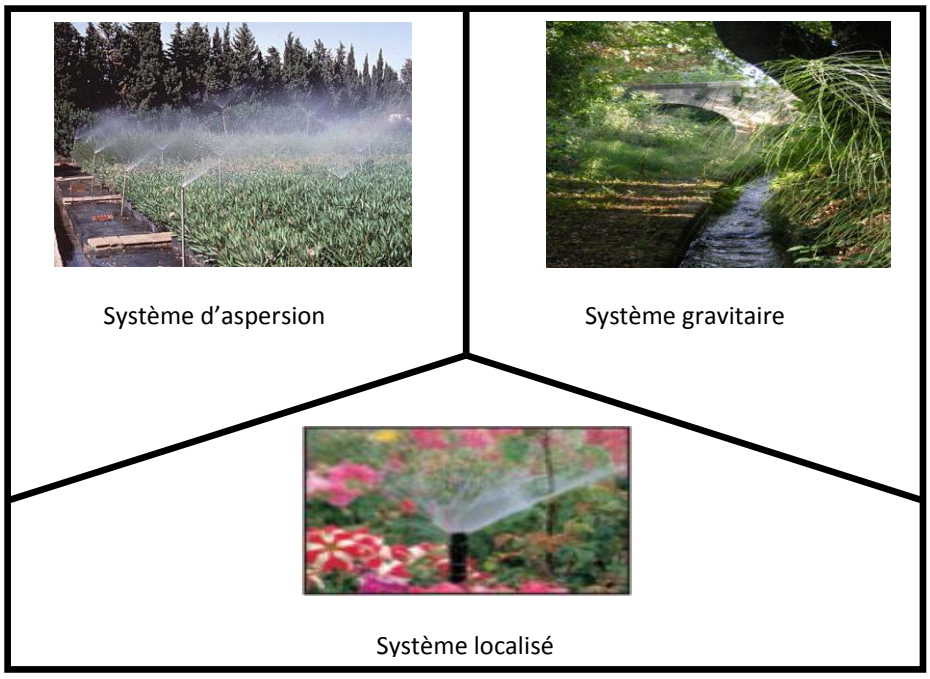


Figure III. 3 : Les systèmes d'irrigation

Le tableau III.3 nous montre les types d'ouvrages employés pour l'irrigation. On utilise un grand nombre de forages (1481) pour une surface importante et environ 1403 puits pour la moitié de la surface irriguée par forage, le pompage au fil d'eau représente un nombre important. On utilise aussi 207 sources pour 2241 hectares, les petits barrages pour une surface assez petite et seulement 3 barrages pour irriguer une surface de 2985ha.

Forages		Puits		Barrages		R -collinaire- Petits barrages	
Nombre	Surface (ha)	Nombre	Surface (ha)	Nombre	Surface (ha)	Nombre	Surface (ha)
1481	8486	1403	4042	3	2985	9	500
Sources		**Pompage au fil de l'eau**		**Surface totale irriguée (ha)**			
Nombre	Surface (ha)	Nombre	Surface (ha)	23764			
207	2241	591	5510				

Tableau III. 3 : Superficies irriguées par types d'ouvrage.

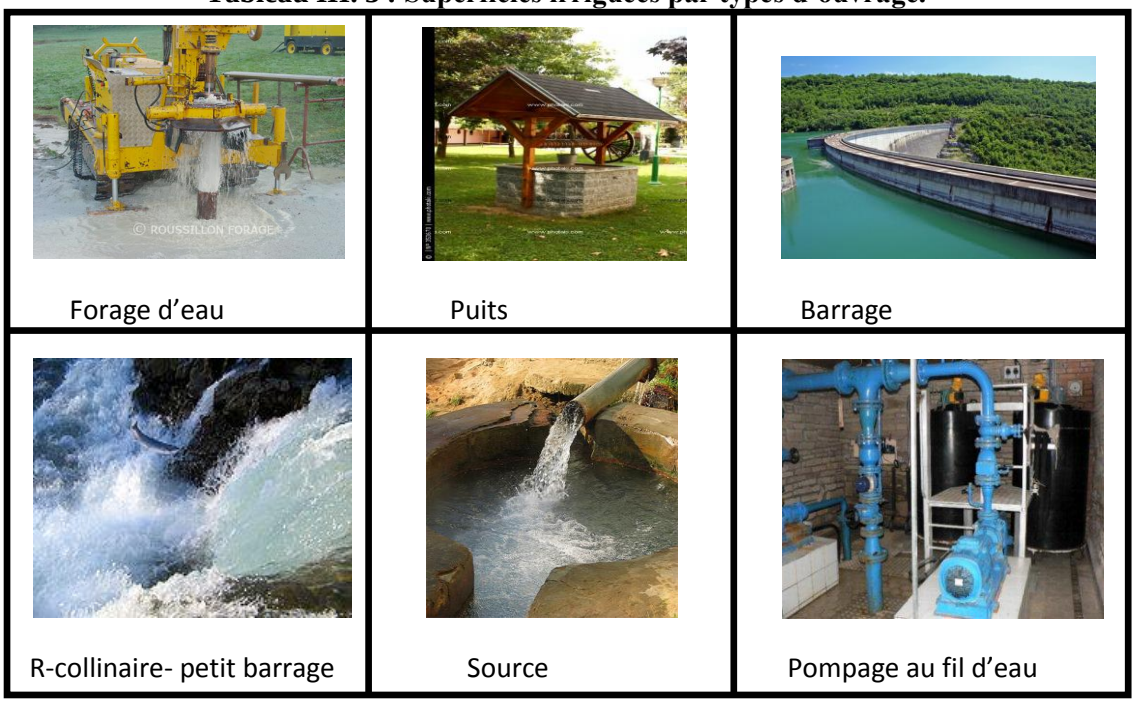


Figure III. 4 : Les types d'ouvrages.

Le tableau III.4 nous donne des statistiques sur les volumes d'eau utilisés à partir de chaque type d'ouvrage en hm^3 par année selon les surfaces en ha.

On remarque que les forages ont plus de volumes que les autres, c'est pour cette raison qu'ils sont les plus utilisés.

Forages		Puits		Barrages		R -collinaire- Petits barrages	
Nombre	Volume Hm3/an	Nombre	Volume Hm3/an	Nombre	Volume Hm3/an	Nombre	Volume Hm3/an
8486	50,916	4042	24,252	2985	16,25	500	3
Sources		**Pompage au fil de l'eau**		**Surface totale irriguée(ha)**			
Nombre	Volume Hm3/an	Nombre	Volume Hm3/an	23764			
2241	13,446	5510	33,06				

Tableau III. 4 : Volume d'eau utilisé à partir de chaque type d'ouvrage (en hm3/an)

Note : statistiques globales dans l'Algérie en 2011

Superficies irriguées nationales : 1 004 530 ha

Dont : Le système par aspersion irrigue 242 494 ha de la surface totale utile, le système localisé irrigue 206 730 ha alors la superficie équipée va être 449 224 ha et le système gravitaire : 555 306 ha. On déduit que le système gravitaire est toujours le plus utilisé.

En conclusion, le système d'irrigation utilisé actuellement se fait par des voies traditionnelles, chaque fois on arrose l'ensemble des terres y compris les zones humides. De ce fait on tente dans notre projet de concevoir des systèmes d'irrigation optimisés, on va utiliser une nouvelle technologie pour économiser de l'eau, on installe des capteurs qui se déclenchent une fois ils détectent une alarme, et le système d'aspersion deviendra automatique, on se propose de lui conférer l'appellation de « système d'irrigation par aspersion intelligente ».

11. Conclusion

Dans ce chapitre, nous avons parlé de l'irrigation automatique intelligente pour économiser de l'eau avec des statistiques sur les superficies irriguées par types de cultures, systèmes d'irrigation et types d'ouvrages avec le volume d'eau utilisé selon chaque type d'ouvrage , ce qui nous permet de passer au dernier chapitre qui est le cœur de notre PFE, l'implémentation d'une application d'irrigation automatique.

Chapitre IV:

Implémentation et évaluation de l'application

Dans ce chapitre, on présente l'environnement du travail, le choix du simulateur et ses objectifs ainsi que la description de l'interface graphique TinyViz et les différentes étapes pour la réalisation de l'application.

1. Introduction

Les avancées technologiques dans les réseaux de capteurs ont permis de développer l'irrigation des terres d'une manière intelligente, cette technologie permet, entre autres, de diviser une terre en zones afin d'effectuer les mesures d'humidité, température... dans chaque zone et de les transmettre à distance à un maitre central appelé le sink, tout ça pour économiser de l'eau.

Avant concrétisation de notre projet, le déploiement d'un réseau de capteurs nécessite une phase de simulation, afin d'assurer le bon fonctionnement de tous les dispositifs. En effet, pour les 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 le cadre du projet, nous avons mis en place une plateforme d'expérimentation qui a pour but de tester, de valider et de simuler le fonctionnement d'un réseau de capteurs. Sa principale fonction est de vérifier le comportement des capteurs développés avant même de les avoir déployé en situation réelle.

Les domaines d'utilisation des réseaux de capteurs sont variés, dans le cadre de notre projet nous nous sommes intéressées plus particulièrement à leur utilisation dans la modélisation et la simulation d'un système d'irrigation intelligent des zones sèches, c'est ce que nous allons présenter dans ce chapitre et à la fin de l'évaluation de l'application les résultats obtenus.

2. Environnement de travail

2.1 Installation de TinyOS

Pour pouvoir travailler avec un réseau de capteurs sans fil, il est nécessaire d'avoir un environnement dédié à celui-ci, l'utilisation des systèmes d'exploitation traditionnels comme Windows ou Linux est impossible car ils ont beaucoup trop de fonctionnalités inutiles pour un réseau de capteurs, c'est pourquoi il faut se tourner vers TinyOS.

On a installé une application all in one qui contient à la foi le système, le simulateur, l'application graphique et le Cygwin qui est utilisé pour la compilation et l'exécution des applications NesC.

2.2 Commandes de base utilisées dans Cygwin pour lancer le simulateur

Cygwin est une couche d'émulation de l'API Linux qui permet d'avoir une interface Unix sous Windows.

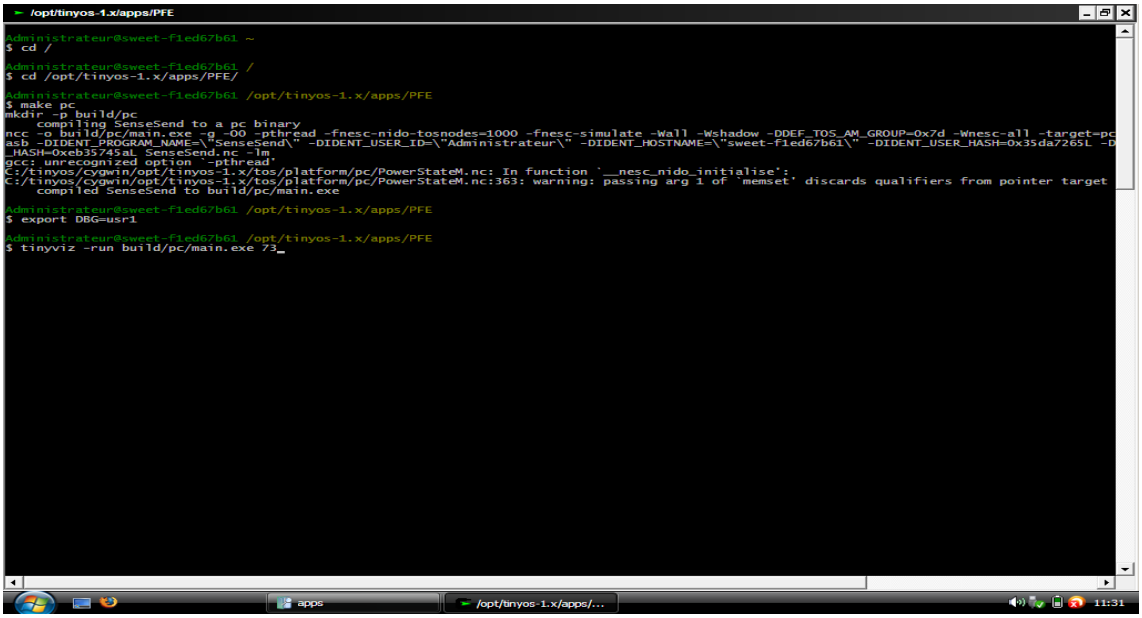


Figure IV-1 l'interface Cygwin

- Tout d'abord, on accède au fichier home par la commande suivante : **cd /**
- Après, on met le chemin de notre application : **cd opt/tinyos-1.x/apps/PFE** pour accéder à l'application « PFE».
- Ensuite, on la compile par la commande : **make pc** (dans notre cas il n'ya pas de capteurs, on l'exécute seulement sur pc).
- Et enfin on l'exécute par : la commande **export DBG=usr1**, et la commande **tinyviz –run build/pc/main.exe nbre_capteurs**

On a configuré les variables d'environnement pour faciliter l'utilisation des commandes et accéder au TinyViz sans chemin.

3. Simulation des réseaux de capteurs

Une simulation d'un réseau de capteurs est nécessaire afin de mieux comprendre le comportement individuel des capteurs et le fonctionnement global du réseau.

Cette phase doit être par la suite être complémentée afin d'estimer les performances temporelles et le comportement d'un grand réseau [GF08].

L'objectif de la plateforme est de simuler un réseau de capteurs, ce qui sous entend que nous n'utilisons pas de capteurs réels. Dans cette optique, nous présentons un des simulateurs utilisé.

3.1 Choix du simulateur

Afin de simuler le comportement d'un capteur (envoi / réception de messages via les ondes radios, traitement de l'information, …) au sein d'un réseau de capteurs, un outil très puissant a été développé et proposé sous le nom de Tossim.

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.

3.1.1 Description de TinyViz

L'outil TinyViz est une application graphique qui nous permet d'avoir un aperçu de notre réseau sans avoir à déployer les capteurs dans la nature.

Une économie d'effort et une préservation du matériel sont possibles grâce à cet outil, l'application permet une analyse étape par étape en activant les différents modes disponibles [FA08].

Comme nous remarquons sur la figure ci-dessous, les capteurs sont disposés dans (en rouge) la fenêtre du logiciel, ils sont placés dans l'espace selon notre topologie.

La partie du haut (en bleu) rassemble toutes les commandes permettant d'intervenir sur la simulation. Nous allons détailler un peu ce que fait chaque bouton présent dans l'interface :

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

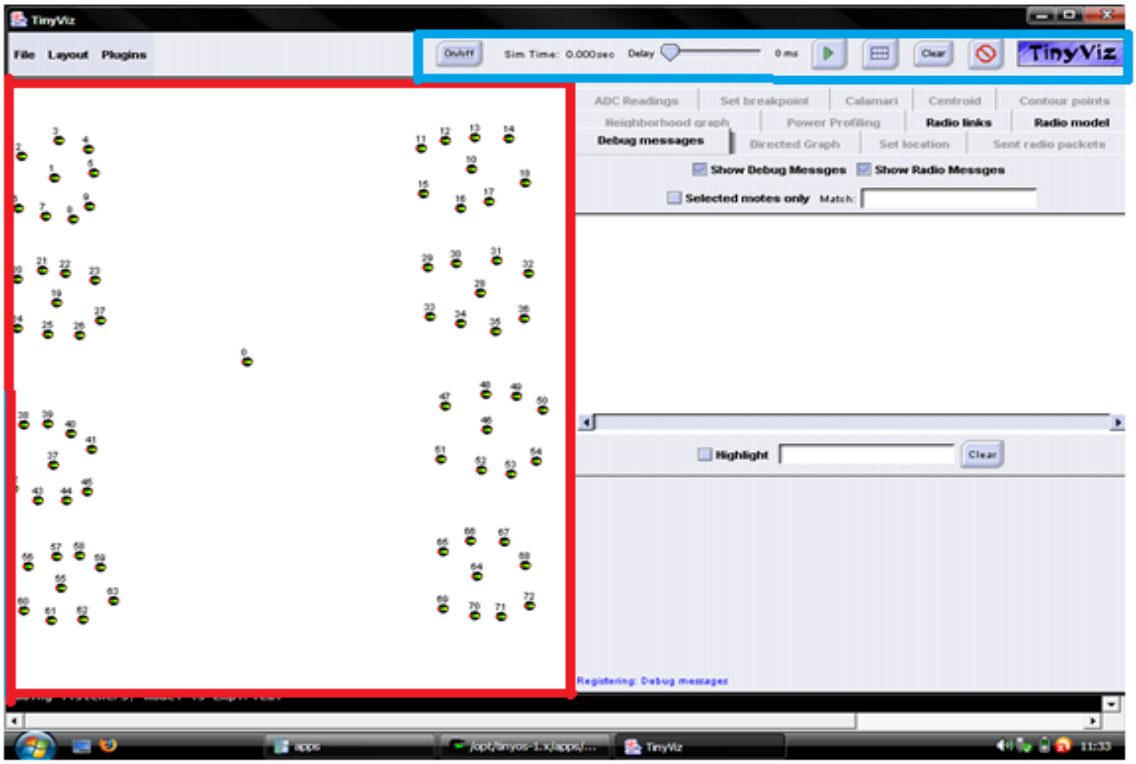


Figure IV-2 La fenêtre du logiciel TinyViz avec la répartition dans l'espace

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

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

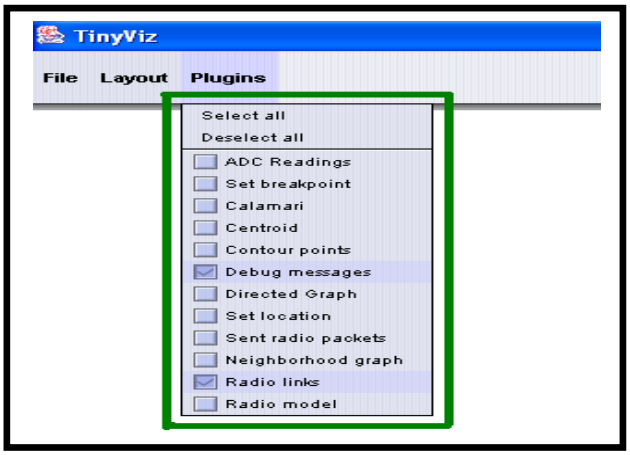


Figure VI-3 Les différents plugins

3.2 Propriétés de Tossim [GF08]

- Le premier but de Tossim est de fournir une simulation des applications de TinyOS, pour cette raison, il simule TinyOS et son exécution.
- Tossim est un simulateur d'évènement, plutôt que de compiler une application pour un capteur, l'utilisateur compile dans cet environnement et tourne sur PC.
- Les utilisateurs peuvent tester et analyser leurs algorithmes dans un environnement contrôlé et identique.
- Tossim reste néanmoins seulement une approche d'un comportement de réseau réel.
- Tossim ne prend pas en compte par exemple la propagation des ondes radios, la consommation d'énergie. Il propose un modèle simplifié qui ne sait pas simuler plusieurs programmes NesC simultanément, ni un réseau de capteurs ayant des nœuds hétérogènes.
- Tous les mauvais fonctionnements d'un réseau de capteurs ne peuvent pas être résolus par Tossim.
- Il reproduit l'envoi et la réception des messages avec le bon routage.

3.3 Objectifs de la simulation

Le but principal de notre projet est de réaliser un système d'irrigation par aspersion intelligente pour évoluer la méthode utilisée, gain du temps et de l'argent et surtout la consommation d'eau.

La conservation obtenue grâce à ce système d'irrigation efficace reste une des alternatives les plus faciles à réaliser et à mettre en place immédiatement pour obtenir une économie d'eau significative. Alors on tente d'aborder de façon approfondie l'irrigation à consommation d'eau réduite par une conception, installation et maintenance importante et qui va aider beaucoup dans notre pays.

4. Les fichiers de l'application

Notre application nommée PFE est formée des composants suivants :

- un module, appelé « humiditeM.nc ».
- une configuration, appelée « humidite.nc ».
- le fichier d'entête, appelé « MessageHumidite.h ».
- le fichier de configuration « capteurs.mps »

Le fichier de configuration portant le nom de l'application elle-même avec une extension particulière (dans notre cas **humidite.nc)** permet au compilateur NesC de

générer un fichier exécutable. **humidite.nc** est utilisé pour câbler le module aux autres composants que l'application **PFE** nécessite.

5. Topologie du réseau

Notre application consiste à simuler un réseau de capteurs qui permet de détecter la sècheresse d'une terre agricole, les nœuds de ce réseau sont des capteurs d'humidité, de température et de luminosité. On divise la terre en huit zones géographiques comme il est présenté dans la figure ci-dessous, dans chaque zone on installe huit capteurs (nœuds ordinaires). Quand chacun des huit nœuds mesure la valeur d'humidité et de la température autour de lui, il va l'envoyer à son maitre (cluster Head) et ce dernier va calculer la moyenne des valeurs reçues et l'envoyer au sink qui récupère la moyenne de chaque zone et va localiser si la zone humide ou sèche, donc on irrigue seulement la partie sèche de la terre pour minimiser la consommation de l'eau.

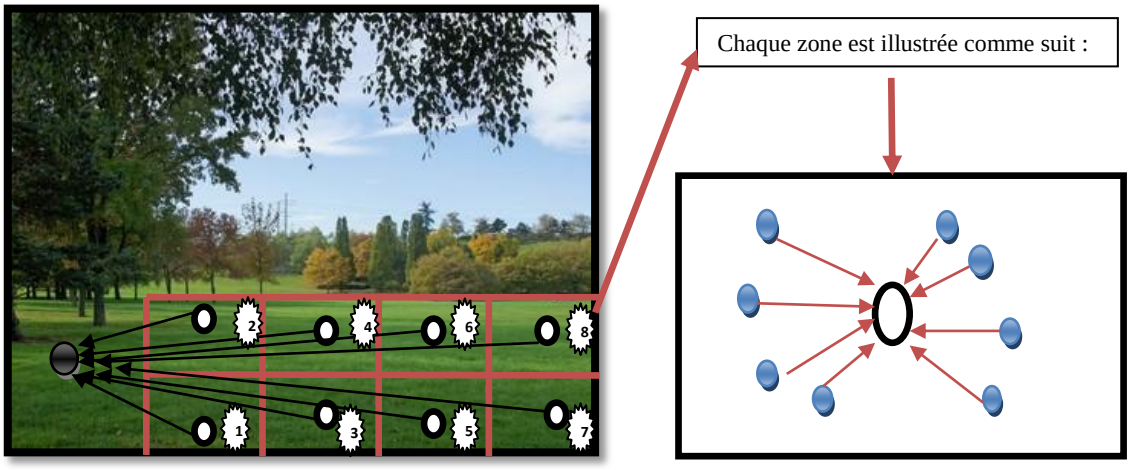


Figure IV-4 topologie de l'application

: nœud ordinaire pour capter l'humidité et température.

: Le maitre.

: Le sink.

: Zone géographique.

: Envoi les valeurs captées au cluster head.

: Envoi des calculs au sink.

: Numéro de la zone.

6. Les étapes d'exécution de notre application

- L'envoi des messages des nœuds ordinaires vers leur cluster head

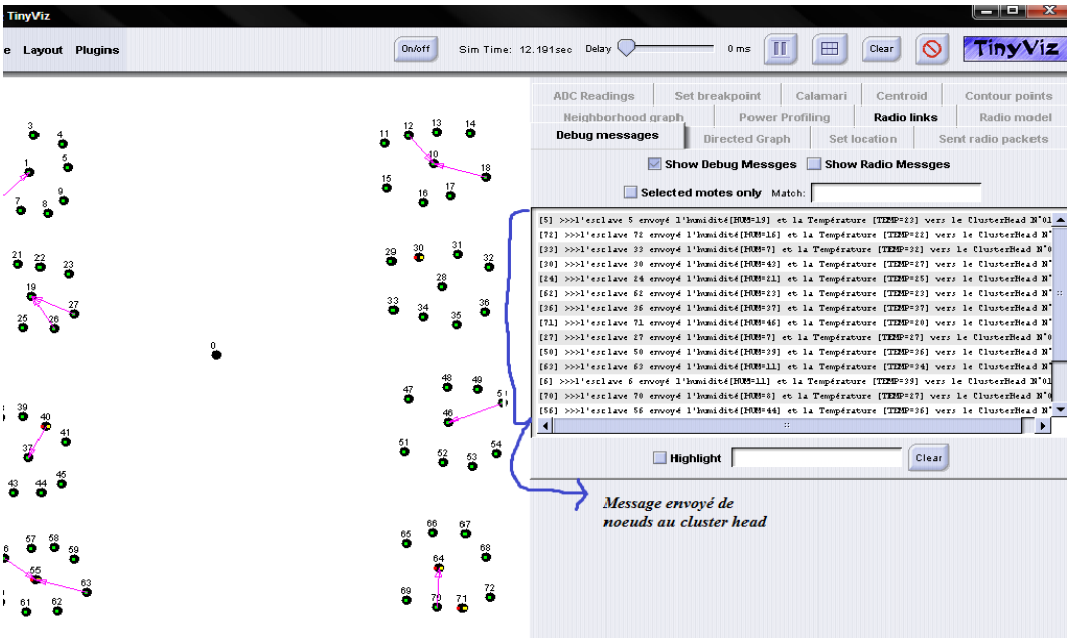


Figure IV-5 Envoi des messages au CH

- L'envoi des messages de chaque cluster head vers le sink

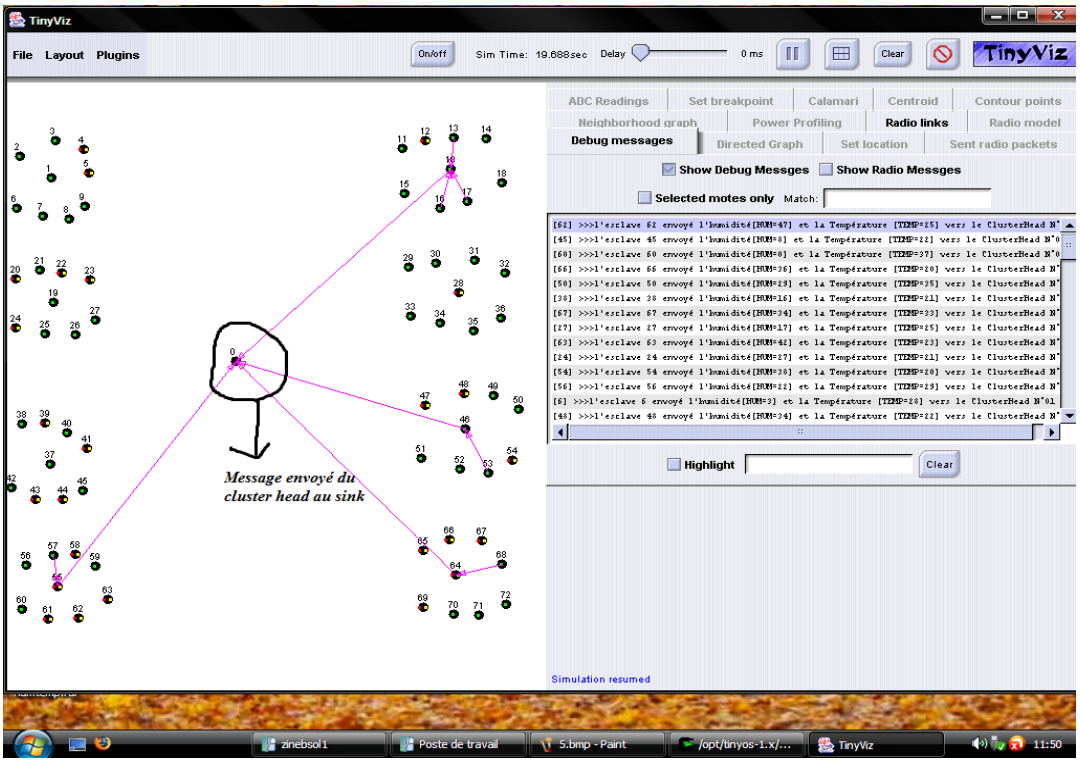


Figure IV-6 envoi des messages au sink

- Détection des zones sèches et humides par le sink

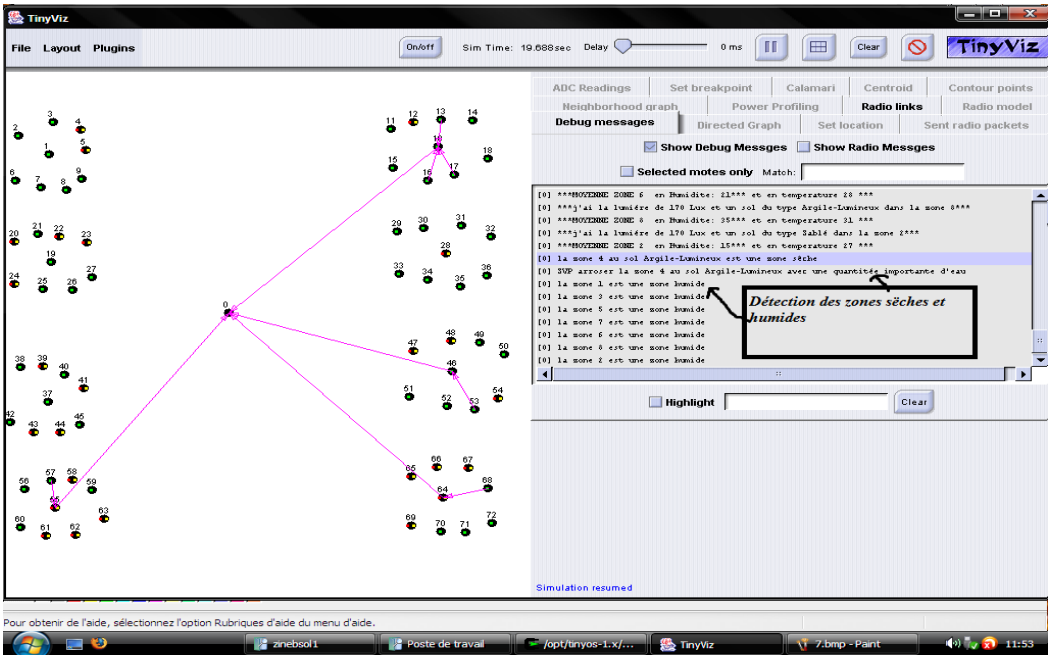


Figure IV-7 Détection de nature des zones

7. Les métriques d'évaluation de l'application

7.1 Première étude (si tous les capteurs fonctionnent bien)

Pour détecter si une zone est sèche ou bien assez irriguée, il est nécessaire d'implanter des capteurs en se servant d'une sonde à plusieurs niveaux du sol. Ces capteurs ont pour objectif de collecter l'humidité du sol dans une zone géographique restreinte, et par la suite, prendre une décision, qui est « arroser ou ne pas arroser la zone cible ».

Bien sûr cette décision relève des résultats de fonctions statistiques. Dans la première étape de notre étude, nous avons effectué une comparaison de deux fonctions statistiques, à savoir la moyenne et la médiane, afin de déterminer la fonction qui donne la meilleure décision concernant l'état du sol.

7.1.1 Estimation entre moyenne et médiane

Le cluster head reçoit des valeurs en entrées venant des nœuds ordinaires, il doit faire un calcul pour l'envoyer au sink et décider par la suite sur la sécheresse d'une zone ou non. Parmi les méthodes de calcul nous distinguons deux méthodes, il faut choisir entre la moyenne qui est la somme de l'ensemble des valeurs mesurées divisée par le nombre des capteurs ou la médiane qui est basée sur le stockage des valeurs représentées dans un tableau trié en ordre croissant, la valeur médiane correspond à la valeur qui se trouve au point milieu de cette liste ordonnée.

Les valeurs reçues par le CH sont normalisées, elles sont comprises entre 00 et 100 (valeurs d'humidité en %).

Pour une zone sèche, l'humidité du sol est comprise entre 00 et 24 alors que pour une zone bien irriguée, l'humidité est comprise entre 25 et 100.

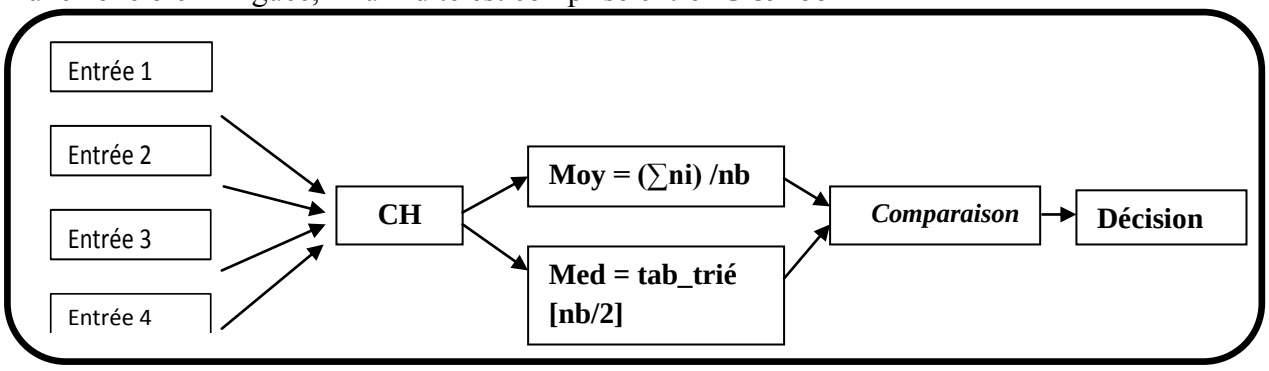


Figure IV-8 comparaison entre moyenne et médiane

Dans cette figure, « **ni** » est la valeur captée par le nœud ordinaire **i** et « **nb** » représente le nombre des nœuds ordinaires (quatre dans notre cas).

Valeurs / zones	Valeur 1	Valeur2	Valeur3	Valeur4
Zone1	13	70	20	80
Zone2	10	07	98	42
Zone3	03	91	96	89

Tableau IV-1 Des exemples de valeurs captées dans trois zones

Le tableau ci-dessus représente les valeurs obtenues dans chaque zone après la simulation, le tableau ci-dessous représente leurs moyennes et médianes.

Façon de calcule / zone	moyenne	Médiane
Zone1	45	20
Zone2	39	10
Zone3	50	89

Tableau IV-2 la moyenne et la médiane de chaque zone

La figure suivante nous illustre les données du tableau IV-2 :

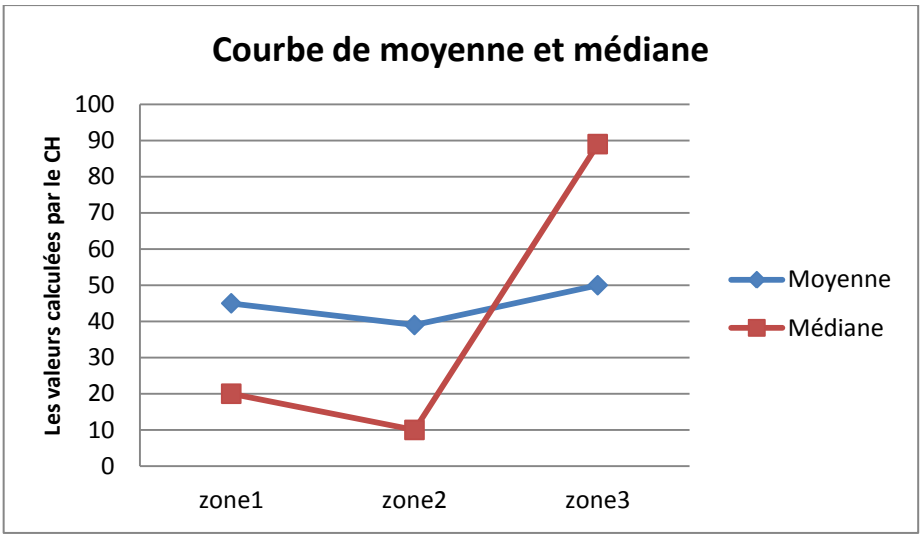


Figure IV-9 courbe de comparaison entre moyenne et médiane

7.1.2 Discussion sur les résultats

On remarque que les résultats obtenus en utilisant séparément la moyenne et la médiane ne sont pas les même. Nous considérons dans ces résultats que tous les nœuds d'une zone cible fonctionnent correctement. La médiane, récupère toujours la deuxième valeur du tableau trié même si cette valeur n'englobe pas l'état de la zone, par exemple la médiane du zone 3 est 89, ce qui signifie que cette zone n'a pas besoin d'irrigation, malgré qu'il ya une valeur détectée égal à 03 qui nécessite l'irrigation.

La valeur de la moyenne est approximativement proche des autres valeurs alors que la médiane peut avoir des fois une valeur éloignée par rapport aux autres valeurs captées. Nous pouvons dire que les valeurs de la moyenne sont plus objectives que ceux de la médiane.

7.1.3 Décision de la discussion

Après avoir analysé les données numériques acquises de la simulation et après avoir analysé les points négatifs du calcul par la médiane qui restreint la deuxième valeur pour la décision. On a jugé que le calcul par la moyenne est plus raisonnable du de point de vue pratique que la médiane, et nous donne une bonne estimation de l'état de la zone.

7.2 Impact de la densité de la zone

Nous entendons dire par densité, le nombre des nœuds ordinaires installé dans chaque zone. Dans cette étude le cluster head va calculer seulement avec la moyenne (on a décidé de faire ça précédemment). Dans cette section on va comparer une zone qui utilise 4 nœuds ordinaires et une autre qui utilise 8 nœuds ordinaires.

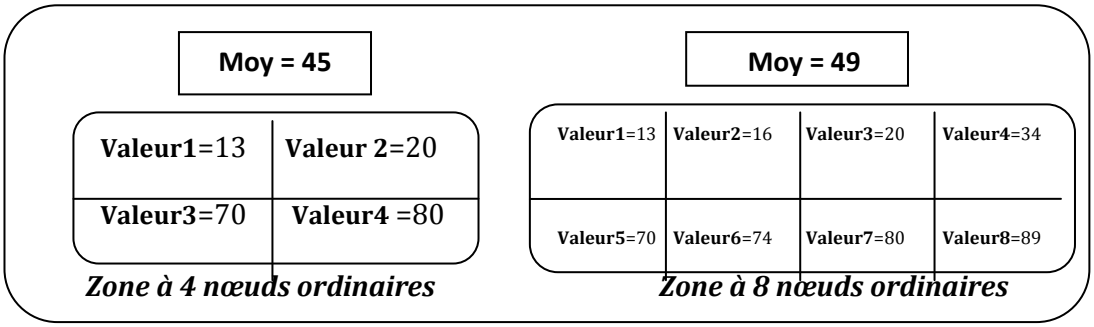


Figure IV-10 valeurs obtenues dans une zone à 4 nœuds ordinaires et dans une autre à 8 nœuds ordinaires

La figure IV-11 suivante nous montre les valeurs acquises par la simulation :

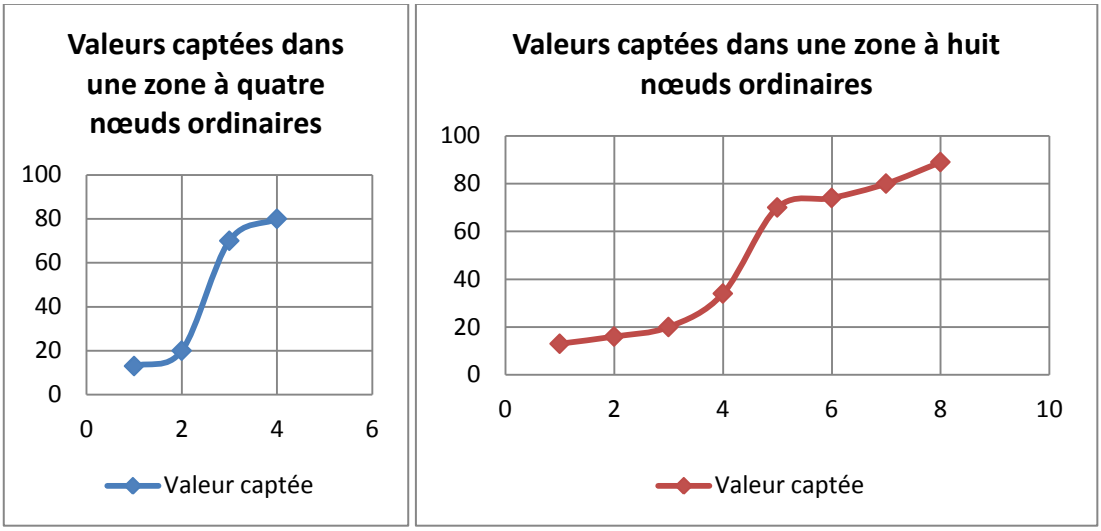


Figure IV-11 valeurs captées dans une zone à quatre nœuds ordinaires et dans une autre à huit nœuds ordinaires

A travers cette figure on remarque que l'étude faite sur une zone à 8 nœuds ordinaires est plus significative que celle à 4 nœuds ordinaires, ainsi une zone à huit nœuds ordinaires améliore bien l'étude et précise parfaitement l'état de la zone et donc plus on augmente le nombre des nœuds ordinaires dans la zone, plus on arrive à des résultats meilleurs.

La question qui se pose maintenant est « est ce que l'humidité est suffisante pour la détection de la sécheresse d'un sol ? » Nous allons discuter cela dans ce qui suit.

7.3 Les paramètres à ajouter pour une meilleure détection

Dans une terre agricole, il existe plusieurs paramètres à prendre en compte en fonction des plantes qui y sont cultivées. Cependant, il existe des périodes pendant lesquelles ces paramètres telle que la température, l'humidité…etc. a l'intérieur de la terre, deviennent dangereuses pour la plante, la capacité de l'homme pour avoir des jugements précis et rapides dans ce cas est limitée.

C'est pourquoi que beaucoup de systèmes de contrôle automatique et de régulation ont été introduits dans une terre agricole pour se substituer à l'intervention de l'homme en le libérant des contraintes contraignantes de surveillance de la terre.

Notons qu'un climat est un ensemble de conditions atmosphériques qui caractérise un espace homogène de la terre [LZ09] et qui contribue au déroulement de la croissance et du développement des différents organes et plantes. Ainsi, le climat est un facteur de rendement qu'il est nécessaire de maitriser pour obtenir des rendements optimaux des cultures. La température, l'hygrométrie, la luminosité et le type de sol sont des éléments principaux sur lesquels il faut veiller pour commander au mieux l'atmosphère d'une terre ou d'un champ. Ceci a des conséquences sur la croissance, la qualité des récoltes et les problèmes parasitaires [CL07].

Cependant, l'humidité peut empêcher le développement de certains parasites des plantes qui prolifèrent dans le champ. De plus dans un champ, l'humidité augmente en l'absence de renouvellement d'air et durant la nuit, elle augmente encore plus avec le phénomène de condensation de la température qui tend à diminuer cette humidité [YP04], un paramètre très important à prendre en compte est la lumière (une quantité maximale de lumière est nécessaire pour que le processus photosynthétique puisse s'opérer).

7.3.1 Description du type de sol utilisé

Le type de sol dans lequel toutes les données expérimentales ont été recueillies : Parmi dix types de sol, on a choisit dans notre étude les trois types suivants : sol sablé, sol limoneux et argile- limoneux.

Le système de pilotage assure les opérations suivantes : « refroidissement », « humidification » et « éclairage », ces opérations permettent de fixer la température et l'humidité pendant la journée et la lumière pendant la nuit.

7.3.2 Méthode de modélisation utilisée

Nous proposons ici, l'utilisation d'une fonction qui est composée de quatre paramètres : *température*, *hygrométrie*, *luminosité* et *type de sol*.

On la note comme suit : $\Omega = \{H, T, L, S\}$ ou H : hygrométrie, T : température, L : luminosité et S : type de sol

7.3.3 Bornes de chaque paramètre

Dans notre application, les bornes de mesures dans un champ agricole sont recueillies auprès de la direction d'irrigation et sont les suivantes :

- La Température : [25°, 40°]

- L'Humidité : [00%, 30%]
- Luminosité est fixée à 170 lux.
- Sol : {sablé, limoneux, argile-limoneux}

Le calcul respecte les bornes inférieures et supérieures et aussi les trois types de sol.

7.3.4 Mesures effectuées et prise de décision

Après l'exécution de l'application qui utilise la moyenne pour le calcul, et huit nœuds ordinaires dans chaque zone. Les résultats obtenus des six exemples de simulation sur les trois types de sols sont les suivants:

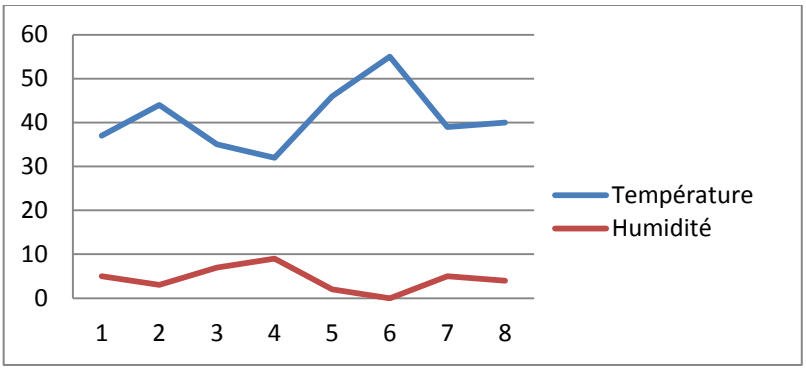


Figure IV-12 Exemple d'une zone sèche de type sablé

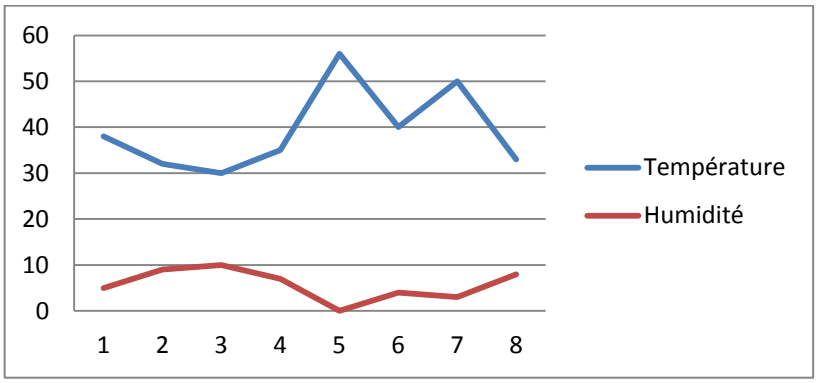


Figure IV-13 Exemple d'une zone humide de type sablé

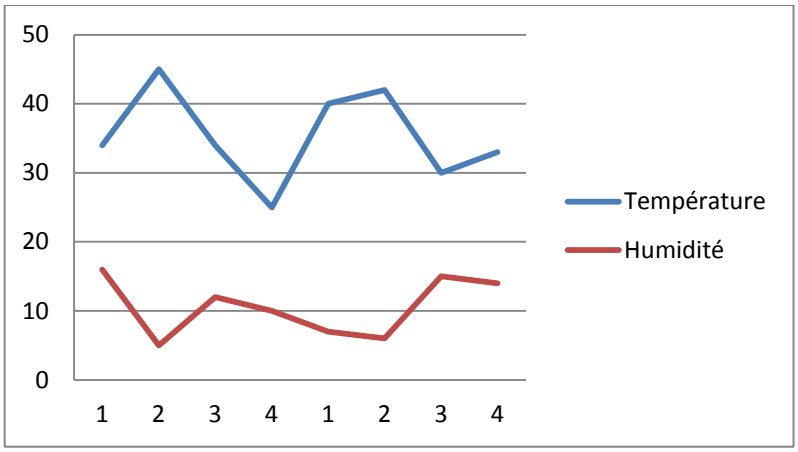


Figure IV-14 Exemple d'une zone humide de type limoneux

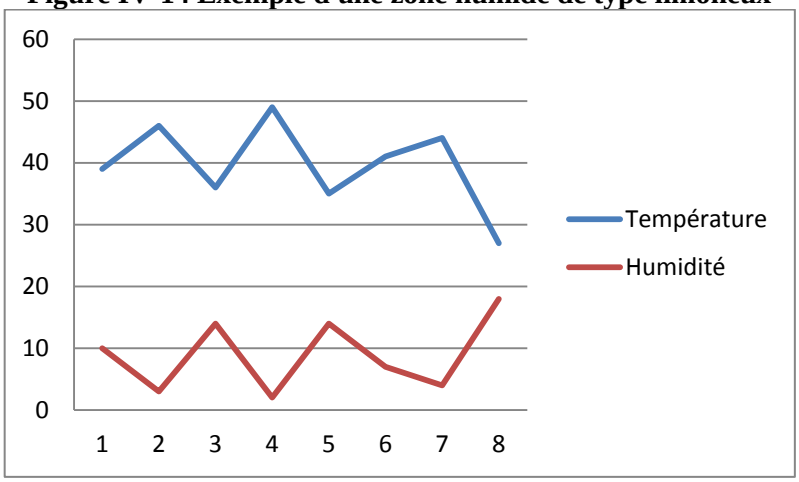


Figure IV-15 Exemple d'une zone sèche de type limoneux

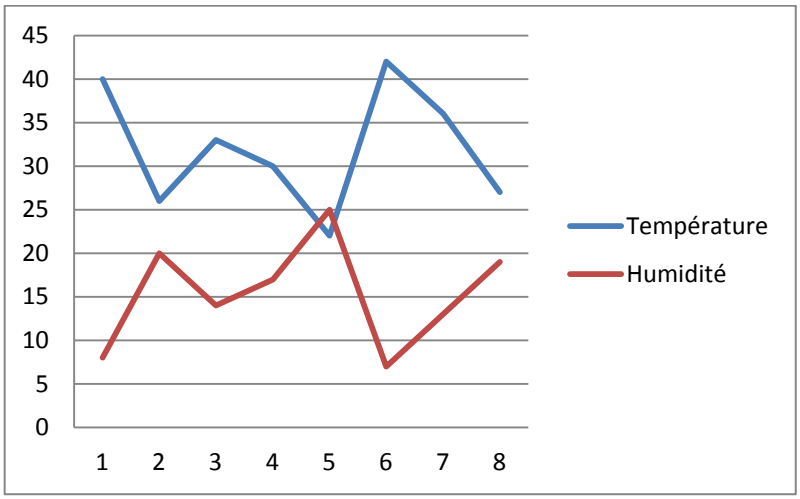


Figure IV-16 Exemple d'une zone sèche de type argile- limoneux

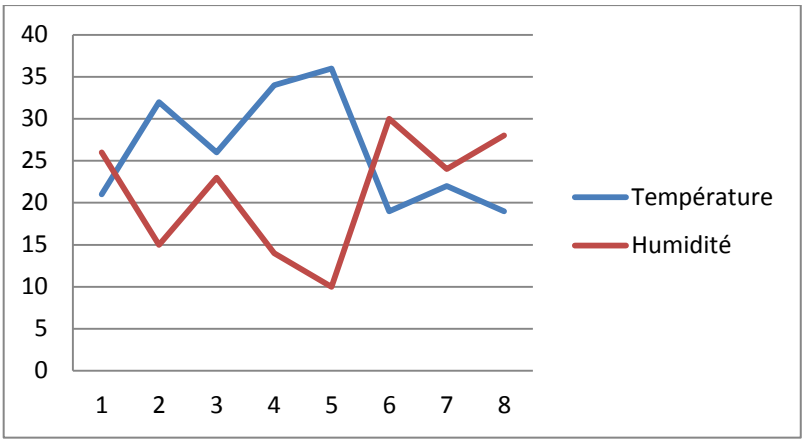


Figure IV-17 Exemple d'une zone humide du type argile- limoneux

D'après ces résultats, nous pouvons fixer les seuils optimaux correspondant à chaque type de sol :

- On dit qu'une zone de type « sablé » est humide si l'humidité du sol va être égale ou supérieure à 5% et la température entre 35ºet 40º, sinon elle est sèche.
- On dit qu'une zone de type « limoneux » est humide si l'humidité du sol varie entre 10 à 15% et la température entre 30ºet35º, sinon elle est sèche.
- On dit qu'une zone de type « argile-limoneux » est humide si l'humidité du sol va être égale ou supérieure à 20% et la température entre 25ºet 30º, sinon elle est sèche.

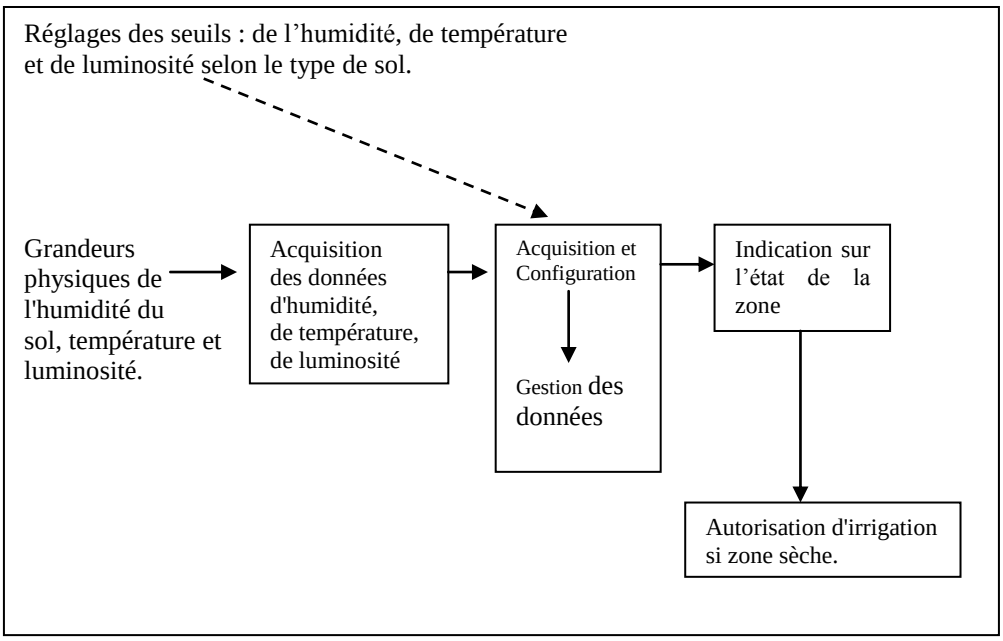


Figure IV-18 les étapes faites par le sink

8. Deuxième étude (en cas de panne des capteurs)

Les capteurs peuvent être utilisés dans des milieux hostiles comme, par exemple, dans un champ de bataille, en présence d'incendies, d'inondations…etc. Dans ces environnements les capteurs peuvent tomber en panne même au cours d'un fonctionnement normal. Par ailleurs, la communication entre les capteurs se fait généralement par diffusion radio et cette diffusion ne peut être supposée fiable. Il est donc intéressant et nécessaire d'étudier la tolérance aux pannes pour ce type de réseau.

On a deux types de défaillances qui peuvent être considérées, les pannes de capteurs dues par exemple à un défaut d'énergie et les défaillances de communication. En ce qui concerne les premières, nous les restreindrons aux pannes « crash » où le capteur s'arrête définitivement. Pour le deuxième type de défaillance, la communication de ces réseaux se fait généralement par diffusion radio mais la particularité de ce mode de communication est qu'il n'est pas fiable.

On va étudier tout les types de pannes avec cet exemple : zone humide de type sablé.

Paramètres / Num_capteur	Humidité	Température
1	33	7
2	37	6
3	32	8
4	50	1
5	43	3
6	44	2
7	35	5
8	24	10

Tableau IV-3 tableau d'étude

8.1 Pannes crash

On les sépare en deux types :

- Le message n'est pas diffusé (le capteur s'arrête).
- Le capteur peut émettre un message vide.

8.1.1 Cas 1 : le message n'est pas diffusé grâce à un défaut d'énergie

1- Si le capteur de plus haute valeur d'humidité et plus basse valeur de température tombe en panne: dans le cas de l'exemple qu'on a pris c'est le capteur 8.

   Le sink va diviser la somme des valeurs qu'il a reçu sur le nombre des capteurs qui ont envoyé les mesures et les moyennes pour les deux paramètres vont être comme suit : 4 pour l'humidité et 39 pour la température. Les résultats qu'on a obtenus sans le capteur 8 désignent que la zone est sèche alors qu'elle est humide. Mais si le sink fait ses calculs avec la médiane, on arrive à des résultats positifs, il nous affiche que la zone est humide (la valeur du capteur 7).

2- Si le capteur de plus basse valeur d'humidité et plus haute valeur de température tombe en panne : dans le cas de l'exemple qu'on a pris c'est le capteur 4, les résultats de simulation ne changent pas car on a éliminé une des basses valeurs d'humidité, il nous indique que la zone est sèche alors qu'elle est humide (c'est favorable).

3- Si deux capteurs (un de plus haute et l'autre de plus basse valeurs d'humidité) tombent en panne (capteur 4 et capteur 8) : la zone reste humide.

4- Si deux capteurs de plus hautes valeurs d'humidité tombent en panne (capteur 3 et capteur 8) : la zone devient sèche selon les deux façons de calcul : médiane et moyenne car les deux capteurs qui indiquent que la zone est humide ont chutés.

5- Si deux capteurs de plus basses valeurs d'humidité tombent en panne (capteur 4 et capteur 6) : la zone reste humide.

En conclusion, la panne des capteurs ayant capté des valeurs identiques à l'état réel de la zone, entraîne surement une mauvaise décision, puisque leurs valeurs ne seront pas prises en compte dans le calcul. Ceci peut être évité en utilisant la fonction médiane, qui est plus robuste aux pannes des capteurs lorsque le nombre des capteurs en panne ne dépasse pas la moitié des capteurs d'une zone cible.

8.1.2 Cas 2 : le capteur peut émettre un message vide

On suppose que le sink attribut un zéro au message vide et que cela va affecter le calcul. Le tableau suivant nous illustre les résultats des cas cités précédemment avec ce type de panne :

	Calcule avec moyenne	Calcule avec médiane	L'état réel de la zone	L'état de la zone après la panne
Cas 1	Négative	Positive	Humide	Sèche
Cas 2	Positive	Positive	Humide	Humide
Cas3	Négative	Positive	Humide	Sèche si le calcul est fait par la moyenne et humide avec la médiane
Cas4	Négative	Négative	Humide	Sèche
Cas5	Négative	Positive	Humide	Sèche si le calcul est fait par la moyenne et humide avec la médiane.

Tableau IV-4 résultats obtenus

D'après le tableau ci-dessus, on remarque que le calcul avec la médiane nous donne les vrais résultats dans ce type de panne.

8.2 Pannes de communication

On distingue trois cas :

8.2.1 Le capteur envoi une haute valeur qui est normalement une valeur moyenne ou basse

- Le capteur envoi une valeur haute qui est normalement une valeur basse :

Le capteur 4, au lieu qu'il envoi 1 pour l'humidité et 50 pour la température, il envoi respectivement 9 et 27. Les résultats ne vont pas changer car la zone est humide et le capteur augmente la basse valeur, le problème se pose quand la zone est sèche.

- Le capteur envoi une valeur haute qui est normalement une valeur moyenne :

Le capteur 7, au lieu qu'il envoi 5 pour l'humidité et 35 pour la température, il envoi respectivement 8 et 30. Les résultats ne vont pas changer car la zone est humide et le capteur augmente la valeur moyenne en une valeur haute, le problème se pose quand la zone est sèche.

8.2.2 Le capteur envoi une basse valeur qui est normalement valeur haute ou moyenne

- Le capteur envoi une valeur basse qui est normalement une valeur haute :

Le capteur 8, au lieu qu'il envoi 10 pour l'humidité et 24 pour la température, il envoi respectivement 3 et 45. Les résultats vont changer bien sur car la zone est humide et le capteur diminue la valeur haute en une valeur basse, on ne rencontre pas de problème se pose quand la zone est sèche.

- Le capteur envoi une valeur basse qui est normalement une valeur moyenne :

Le capteur 7, au lieu qu'il envoi 5 pour l'humidité et 35 pour la température, il envoi respectivement 3 et 45. Les résultats ne vont pas changer bien sure car la zone est

humide et le capteur diminue la valeur moyenne en une valeur basse en une faible proportion, le problème se pose quand la zone est sèche.

8.2.3 Le capteur envoi une valeur moyenne qui est normalement valeur haute ou basse

- Le capteur envoi une valeur basse qui est normalement une valeur moyenne :

Le capteur 7, au lieu qu'il envoi 5 pour l'humidité et 35 pour la température, il envoi respectivement 3 et 45. Les résultats ne vont pas changer bien sur car la zone est humide et le capteur diminue la valeur haute en une valeur basse, on ne rencontre pas de problème quand la zone est sèche.

- Le capteur envoi une valeur haute qui est normalement une valeur moyenne :

Dans le capteur 7, au lieu qu'il envoi 5 pour l'humidité et 35 pour la température, il envoi respectivement 9 et 24. Les résultats ne vont pas changer bien sure car la zone est humide et le capteur augmente la valeur moyenne en une valeur haute, le problème se pose quand la zone est sèche.

Pour conclure, nous avons clairement démontré que dans un scénario non idéal, où les nœuds capteurs peuvent tomber en panne, la médiane représente de meilleurs résultats pour une bonne décision par rapport à la moyenne. Nous consolidons l'idée que la moyenne est très sensible en cas des pannes alors que la médiane, c'est une fonction statistique robuste et tolérante aux pannes à un certain degré.

9. Conclusion

Dans ce chapitre, Nous avons implémenté notre application, qui consiste à détecter les zones sèches. Deux méthodes de calcul se basant sur la moyenne des valeurs captées et la médiane, ont été utilisées dans deux scénarios : un scénario idéal, et un scénario non idéal.

Le réglage des seuils pour la prise de décision, n'est tout de même pas facile, puisqu'il faut considérer chaque type de sol. De même la prise par exemple de la mesure de l'humidité à différents niveaux de profondeurs peut varier selon que la sonde est implantée près de la surface ou plus loin. Tous ces problèmes ne peuvent malheureusement pas être réglés par simulation, mais par expérimentation. Il est donc nécessaire de procéder à une expérimentation pour bien régler les seuils des paramètres de calcul et valider le modèle de calcul utilisé.

Conclusion générale

L'agriculture est un secteur économique majeur, une importance qui se mesure en termes d'espaces, d'emplois et d'économie. L'apport économique de l'agriculture de précision incite les entreprises agricoles à mettre le paquet dans la recherche scientifique. Jusqu'à présent les moyens utilisés sont précaires car le thème de l'agriculture de précision est relativement nouveau, mais des avancées significatives ont été faites que d'autres peuvent utiliser. Chaque jour, les technologies de l'information les plus avancées sont apparus sur le marché, tels que les capteurs et les contrôleurs pour une meilleur détection comme c'est le cas de notre application.

Ce travail s'inscrit dans le cadre du projet ANDRU d'Alger ainsi dans les travaux de l'Université de Abou Bekr Belkaid de Tlemcen, c'est un travail de fin d'étude pour l'obtention du diplôme master II en informatique option réseaux et systèmes distribués, concernant le domaine des réseaux de capteurs.

Les réseaux de capteurs sont une nouvelle technologie qui a surgit après les grands progrès technologie concernant le développement des capteurs intelligents, des processeurs puissants et des protocoles de communications sans fil. Ils ont été classés parmi les 21 technologies les plus importantes du 21éme siècle. En effet, la recherche dans le domaine des capteurs est en train de vivre une révolution importante, ouvrant des perspectives d'impacts significatifs dans de nombreux domaines tel que le domaine d'agriculture comme c'est le cas de notre travail, pour que ces réseaux puissent mener à bien leurs missions ils doivent assurer un certain niveau de contrôle qui diffèrent selon l'application déployée.

Au cours de notre mémoire, nous nous sommes intéressées à mettre en place un réseau de capteurs pour l'irrigation intelligente, Il a fallut pour cela prendre en main beaucoup de nouvelles technologies, comme TinyOS et le nesC.

Dans ce travail nous avons débuté par une étude approfondie sur les RCSF dont le but est de mettre en évidence les services critiques du réseau, donc nous avons donné un aperçu sur les RCSF et certaines de leurs applications. Nous avons atteint les objectifs fixés dés le début de ce projet car on a réussi à installer de TinyOS, et prendre en main le langage NesC. Puis on a discuté brièvement sur l'irrigation intelligente pour une meilleure consommation d'eau.

D'un point de vue personnel, ce projet nous a apporté des bénéficies personnels, il a permis de découvrir un nouveau domaine, une nouvelle manière de programmer et de

concevoir une application, avec des contraintes techniques et matérielles très importantes, la découverte de TinyOS et le NesC. Ce projet été particulièrement intéressant par le fait que les réseaux de capteurs sans fil sont vraiment en plein expansion de nos jours, mais encore trop peu connu des personnes extérieures à ce domaine, la recherche sur les réseaux de capteurs est actuellement en pleine essor. Il a été pour nous un grand plaisir de s'intégrer à la recherche dans ce domaine. Travailler sur une activité possédant des enjeux aussi enthousiasmants nous a motivés pour mener à terme ce projet.

On a atteint le but principale de notre application qui est la surveillance de l'environnement pour détection des zones sèches et les irriguer par la suite.

Les résultats qu'on a obtenus ne sont pas une fin en soi, au contraire ils nous mènent à poser des questions sur d'autres perspectives (faire une expérimentation réelle afin de valider nos résultats, utilisation d'un grand nombre de capteurs), d'affiner cette recherche et utiliser cette application pour tirer partie de toutes ses ressources et arriver à des meilleurs résultats.

Cependant on n'a donné qu'un mince aperçu sur les possibilités énormes des réseaux de capteurs, il est donc nécessaire de saisir chaque opportunité et travailler davantage pour pouvoir utiliser toutes ses fonctions.

Bibliographie

[Al06] Académie de Lyon, « système d'arrosage Automatique », session 2006.

[AW09] Anne Weill et Jean Duval, « gestion du sol et de l'eau », manuscrit du guide de gestion globale de la ferme maraichère biologique, 2009.

[BA11] Elboughdiri Abdessalem, « Conception et modélisation d'un réseau de capteur sans fil », mémoire de master, option : Technique de communication radiofréquence, juin 2011.

[CE02] Extrait de la publication de la communauté européenne en 2002, « l'eau c'est la vie ».

[CL07] CaiChun li, «research on intelligent greenhouse environment control system, journal of chogping institute of technology (Natural Science Edition) », Octobre2007.

[CY08] Challal Yacine, « Réseaux de capteurs Sans Fils », 17 Décembre 2008.

[DG03] David Gay, Philip Levis, «The nesC Language, an approach to Networked Embedded Systems in proceedings of programming Language », Juin 2003.

[DN10] Douma Nabila « la localisation dans un réseau de capteurs », projet de première année master RTM, 2010

[EF04] E. Fleury « réseaux de capteurs », 16 décembre 2004.

[FA08] Fares Abdelfatah, « Développement d'une bibliothèque de capteurs », projet de master en informatique, 25/04/2008.

[GF08] Grégory Faye, « rapport de stage ESIES mise en œuvre d'un réseau de capteurs sans fil et développement d'applications », 2008

[GG10] Gwenhael Goavec « Etude d'un système d'exploitation pour microcontrôleur à faible consommation », juin 2010.

[HA08] H.Alatrista, S.Aliaga, K.Gouaich, J.Mathieu, «Implémentation de protocole sur une plateforme de réseaux de capteurs sans-fils », 25 avril 2008.

[HR08] Hafdaoui Rabab, « le routage dans les réseaux de capteurs sans fil », mémoire de licence, université de khenchela, 2010.

[JB03] Jacques Beauchamp, « La lutte contre l'érosion des sols dans les régions de grandes cultures », université de Picardie jules verne, 17/11/2003.

[JM11] Jerroudi Mohammed « systèmes embarqués », rapport de projet, 2011.

[JW94] Jutta Wiliamowski « modélisation des taches pour la résolution des problèmes en coopération » mémoire de doctorat, université Joseph Fourier, 06/04/1994.

[KP09] Kadionik Patrice, « Projet avancé: Réseau de capteurs sans fils », 3eme année électroniques, option : Systèmes embarqués, juin 2009.

[LD08] L.Daumas « implémentation d'un protocole de routage dans un réseau de capteurs sans fil », mémoire de master, université d'avignon, 2008.

[LZ09] Liang Zhu, «application of the enviromental monitoring technology in the protected agriculture», 2009.

[MD07] Mehdi Damou, « simulation d'un réseau de capteur avec TinyOS », projet de fin d'étude, 2007.

[MK09] Medjhoum Khaled, « le système embarqué Tinyos », 2009.

[NB04] Nadjib Badache « les réseaux de capteurs », rapport de recherche dans les systèmes informatiques, février 2004.

[RK11] Rahim Kacimi « Premiers pas avec TinyOS », projet en télécommunications et Réseaux, janvier2011.

[P04] Exemples de produits non disponibles de système S.Sense en France, juin 2004.

[SA10] Samir Athmani, « protocole de sécurité dans les réseaux de capteurs sans fil », mémoire de magister option ingénierie des systèmes d'informations, université de Betna, 15/7/2010.

[SB11] Sahraoui Belkheyr, « La Géo-localisation dans les Réseaux de Capteurs sans Fil », projet d'ingénieur d'Etat en Informatique, Juillet 2011.

[SN10] Chee-Yee Chong, Srikanta P. Kumar « Sensor Networks: Evolution, Opportunities, and Challenges », 2010.

[TA08] T.Abdelkrim « mise en place d'un réseau de capteurs sans fil », projet de master1, 2008.

[UC04] Site officiel de TinyOS « http://www.tinyos.net» ,2004.

[VJ06] Julien Vaudour, « Élaboration de couches MAC et réseau pour un réseau de capteurs », projet d'ingénieur I.I.E, 29 juin 2006.

[WZ06] Wassim Znaidi «modélisation formelle de réseaux de capteurs à partir de TinyOS » rapport de projet de fin d'étude option signaux et systèmes, 10 juin 2010.

[YM08] Yannis Mazzer « Réseaux de microcontrôleurs à faible consommation d'énergie embarquant des capteurs » projet participant dans l'Institut National De Recherche En Informatique et en Automatique INRIA, 2008.

[YP04] Yuwei Ping, « establishment of energy saving solar green house and technology of protective cultivation on sandy land», 2004.

Liste des Figures

Liste des Tableaux

Annexe 1 : Abréviations

ADC	**A**nalog to **D**igital **C**onverter
AODV	**A**d hoc **O**n **D**emand **V**ector Routing
BSD	**B**erkeley **S**oftware **D**istribution licence
CH	**C**luster **H**ead
ID	**ID**entifiant
JDK	**J**ava **D**evelopment **K**it
LED	**L**ight **E**mitting **D**iode complement
NesC	**N**etwork **E**mbedded **S**ystem **C**
OLSR	**O**ptimized **L**ink **S**tate **R**outing Protocol
OS	**O**perating **S**ystem
PDA	**P**ersonal **D**igital **A**ssistant
RAM	**R**andom **A**ccess **M**emory
RCSF	**R**éseaux de **C**apteurs **S**ans **F**il
SAT	**S**uperficie **A**gricole **T**otale
SE	**S**ystème d'**E**xploitations
SUT	**S**uperficie **A**gricole **U**tile
TCP	**T**ransport **C**ontrol **P**rotocol
TinyOS	**Tiny** micro threading **O**perating **S**ystem
TinyViz	**T**iny **V**isualization **S**oftware
Tossim	**T**inyos **o**perating **s**ystem **sim**ulator
WSN	**W**ireless **S**ensor **N**etwork

Annexe 2 : Mots clés en NesC

interface	A collection of event and command definitions
module	A basic component implemented in nesC
configuration	A component made from wiring other components

implementation	Contains code & variables for module or configuration
components	List of components wired into a configuration

provides	Defines interfaces provided by a component
uses	Defines interfaces used by a module or configuration
as	Alias an interface to another name

command	Direct function call exposed by an interface
event	Callback message exposed by an interface

call	Execute a command
signal	Execute an event
post	Put a task on the execution queue
task	A function to be executed in the background

includes	Include a header file

async	For commands, events executed asynchronously
atomic	Guarantees atomic execution of a statement
norace	Eliminates warnings of race conditions nesC detected

Annexe 4 : Code source de l'application

```
// Projet de fin d'étude réalisé par Benameur Amina et Bouzidi Zeyneb

includes HumidMessage;
 module humiditeM {
  provides {
    interface StdControl;
          }
  uses {
    interface Timer;
    interface Leds;
    interface ADC;
    interface SendMsg;
   interface ReceiveMsg;
  //interface Random; on utilise random mais pas interface une fonction nommée random
      }          }
implementation {
// les variables utilisées
                                    TOS_Msg data;
                                    uint16_t valueHum,valueTemp,ClusterHead,typeSol;
                                    uint16_t somme = 0;
                                    uint16_t somme1 = 0;
                                    uint16_t somme2 = 0;
                                    uint16_t somme3 = 0;
                                    uint16_t somme4 = 0;
                                    uint16_t somme5 = 0;
                                    uint16_t somme6 = 0;
                                    uint16_t somme7 = 0;
                                    uint16_t sommeT1 = 0;
                                    uint16_t sommeT2 = 0;
                                    uint16_t sommeT3 = 0;
                                    uint16_t sommeT4 = 0;
                                    uint16_t sommeT5 = 0;
                                    uint16_t sommeT6 = 0;
                                    uint16_t sommeT7 = 0;
                                    uint16_t sommeT8 = 0;
                                    uint16_t moy[8];
                                    uint16_t moyTemp[8];
                                    uint16_t zonn[8];
                                    uint16_t sol[8];
                                    int cmp = 0;
                                    int cmp1 = 0;
                                    int cmp2 = 0;
                                    int cmp3 = 0;
                                    int cmp4 = 0;
                                    int cmp5 = 0;
                                    int cmp6 = 0;
                                    int cmp7 = 0;
                                    uint16_t humidite;
                                    uint16_t temperature;
                                    uint16_t sole;
//*************************random*************************************************************
  int random (int min, int max){
  return (min + (rand() % (max - min + 1)));}
  //*************************initialisation*****************************************************
  command result_t StdControl.init() {
```

```
                    call Leds.init();
                    humidite = random (00,50);// humidité compris entre 00 et 50
                    temperature = random (20,40); // un facteur important dans la détection du sècheresse d'une zone
                   sole = random (00,02);// les trois types de sol                    }
 //************************starting*************************************************
 command result_t StdControl.start() {
 if (TOS_LOCAL_ADDRESS !=0){
    call Leds.greenOn();
    return call Timer.start(TIMER_REPEAT, 10000);}  }
//************************stoping*************************************************************
 command result_t StdControl.stop() {
   return call Timer.stop(); }
//************************connexion*************************************************
  task void connexion() {
  HumidMessage *message = (HumidMessage *)data.data;
                                        atomic message->valueHum = valueHum;
                                        atomic message->valueTemp = valueTemp;
                                        atomic message->typeSol = typeSol;
                                        atomic message->source = ClusterHead;          }
//*************************************************************************************
 event result_t Timer.fired()  {
                                    call ADC.getData();
                                     return SUCCESS;
                                    }
//*************************************************************************************
 async event result_t ADC.dataReady(uint16_t val) {
   if (TOS_LOCAL_ADDRESS !=0){
                                        atomic valueHum = humidite;//random
                                        atomic valueTemp = temperature;
                                        atomic typeSol = sole;
                                        if (valueHum>25 && valueTemp< 35){
                                                                        call Leds.redToggle();
                                                                        call Leds.greenToggle();
                                                                         call Leds.yellowToggle();}}
//********************************** la première zone*************************************
if ((TOS_LOCAL_ADDRESS >1)&&(TOS_LOCAL_ADDRESS<10)){
ClusterHead = 1;
post connexion(); //Appeler la fonction connexion()
call SendMsg.send(1, sizeof(HumidMessage), &data);
dbg(DBG_USR1, ">>>l'esclave %d envoyé l'humidité[HUM=%d] et la Température [TEMP=%d] vers le
ClusterHead N°01  \n", TOS_LOCAL_ADDRESS,valueHum,valueTemp);  }
//********************************** la  deuxième zone*************************************
if ((TOS_LOCAL_ADDRESS >10)&&(TOS_LOCAL_ADDRESS<19)){
ClusterHead =  02;
post connexion ();//Appeler la fonction connexion ()
call SendMsg.send(10, sizeof(HumidMessage), &data);
dbg(DBG_USR1, ">>>l'esclave %d envoyé l'humidité[HUM=%d] et la Température [TEMP=%d] vers le
ClusterHead N°02  \n", TOS_LOCAL_ADDRESS,valueHum,valueTemp); }
//********************************** la troisièmme zone*************************************
if ((TOS_LOCAL_ADDRESS >19)&&(TOS_LOCAL_ADDRESS<28)){
ClusterHead =  03;
post connexion ();//Appeler la fonction connexion ()
call SendMsg.send(19, sizeof(HumidMessage), &data);
dbg(DBG_USR1, ">>>l'esclave %d envoyé l'humidité[HUM=%d] et la Température [TEMP=%d] vers le
ClusterHead N°03  \n", TOS_LOCAL_ADDRESS,valueHum,valueTemp);  }
//********************************** la quatrièmme zone*************************************
if ((TOS_LOCAL_ADDRESS >28)&&(TOS_LOCAL_ADDRESS<37)){
ClusterHead = 04;
```

```
post connexion ();//Appeler la fonction connexion ()
call SendMsg.send(28, sizeof(HumidMessage), &data);
dbg(DBG_USR1, ">>>l'esclave %d envoyé l'humidité[HUM=%d] et la Température [TEMP=%d] vers le
ClusterHead N°04  \n", TOS_LOCAL_ADDRESS,valueHum,valueTemp);   }
//********************************** la cinquièmme zone***********************************
 if ((TOS_LOCAL_ADDRESS >37)&&(TOS_LOCAL_ADDRESS<46)){
ClusterHead =  05;
post connexion ();//Appeler la fonction connexion ()
call SendMsg.send(37, sizeof(HumidMessage), &data);
dbg(DBG_USR1, ">>>l'esclave %d envoyé l'humidité[HUM=%d] et la Température [TEMP=%d] vers le
ClusterHead N°05  \n", TOS_LOCAL_ADDRESS,valueHum,valueTemp); }
//********************************** la sixièmme zone***********************************
if ((TOS_LOCAL_ADDRESS >46)&&(TOS_LOCAL_ADDRESS<55)){
ClusterHead =  06;
post connexion ();//Appeler la fonction connexion ()
call SendMsg.send(46, sizeof(HumidMessage), &data);
dbg(DBG_USR1, ">>>l'esclave %d envoyé l'humidité[HUM=%d] et la Température [TEMP=%d] vers le
ClusterHead N°06  \n", TOS_LOCAL_ADDRESS,valueHum,valueTemp);    }
//********************************** la septièmme zone***********************************
if ((TOS_LOCAL_ADDRESS >55)&&(TOS_LOCAL_ADDRESS<64)){
ClusterHead =  07;
post connexion ();//Appeler la fonction connexion ()
call SendMsg.send(55, sizeof(HumidMessage), &data);
dbg(DBG_USR1, ">>>l'esclave %d envoyé l'humidité[HUM=%d] et la Température [TEMP=%d] vers le
ClusterHead N°07  \n", TOS_LOCAL_ADDRESS,valueHum,valueTemp);    }
//********************************** la huitièmme zone***********************************
if ((TOS_LOCAL_ADDRESS >64)&&(TOS_LOCAL_ADDRESS<73)){
ClusterHead =  64;
post connexion ();//Appeler la fonction connexion ()
call SendMsg.send(64, sizeof(HumidMessage), &data);

dbg(DBG_USR1, ">>>l'esclave %d envoyé l'humidité[HUM=%d] et la Température [TEMP=%d] vers le
ClusterHead N°08  \n", TOS_LOCAL_ADDRESS,valueHum,valueTemp);  }

call Timer.stop();
return SUCCESS;                }
  //*****************************************************************************************
  event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success)  {
   // dbg(DBG_USR1, ">>> envoi effectué pour le capteur %d\n", TOS_LOCAL_ADDRESS);
    return SUCCESS;                }
  //*****************************************************************************************
  event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr m) {
HumidMessage *message = (HumidMessage*)m->data;
if (TOS_LOCAL_ADDRESS ==1) {
      cmp=cmp+1;
      somme =somme +message->valueHum;
      sommeT1 =sommeT1 +message->valueTemp;
                  if (cmp ==8){
                                    HumidMessage *message1 = (HumidMessage *)data.data;
                                    atomic message1->valueHum = somme/8;
                                    atomic message1->valueTemp = sommeT1/8;
                                    atomic message1->source = 1;// la zone N° 01
                                    atomic message1->typeSol = typeSol;
                                    call SendMsg.send(0, sizeof(HumidMessage), &data);     }}
else
    if (TOS_LOCAL_ADDRESS ==10) {
        cmp1=cmp1+1;
       somme1=somme1 +message->valueHum;
```

```
        sommeT2 =sommeT2 +message->valueTemp;
         if (cmp1 ==8){
              HumidMessage *message2 = (HumidMessage *)data.data;
              atomic message2->valueHum = somme1/8;
              atomic message2->valueTemp = sommeT2/8;
              atomic message2->source = 10; // la zone N° 02
             atomic message2->typeSol = typeSol;
             call SendMsg.send(0, sizeof(HumidMessage), &data);     } }
else
      if (TOS_LOCAL_ADDRESS ==19) {
         cmp2=cmp2+1;
         somme2=somme2 +message->valueHum;
         sommeT3 =sommeT3 +message->valueTemp;
         if (cmp2 ==8){
             HumidMessage *message3 = (HumidMessage *)data.data;
             atomic message3->valueHum = somme2/8;
             atomic message3->valueTemp = sommeT3/8;
             atomic message3->source = 19; // la zone N° 03
             atomic message3->typeSol = typeSol;
             call SendMsg.send(0, sizeof(HumidMessage), &data);   } }
if (TOS_LOCAL_ADDRESS ==28) {
   cmp3=cmp3+1;
  somme3 =somme3 +message->valueHum;
  sommeT4 =sommeT4 +message->valueTemp;
 if (cmp3 ==8){
       HumidMessage *message4 = (HumidMessage *)data.data;
       atomic message4->valueHum = somme3/8;
       atomic message4->valueTemp = sommeT4/8;
       atomic message4->typeSol = typeSol;
      atomic message4->source = 28; // la zone N° 04
      call SendMsg.send(0, sizeof(HumidMessage), &data); } }
else
      if (TOS_LOCAL_ADDRESS ==37) {
        cmp4=cmp4+1;
        somme4=somme4 +message->valueHum;
        sommeT5 =sommeT5 +message->valueTemp;
       if (cmp4 ==8){
           HumidMessage *message5 = (HumidMessage *)data.data;
           atomic message5->valueHum = somme4/8;
           atomic message5->valueTemp = sommeT5/8;
           atomic message5->source = 37; // la zone N° 05
          atomic message5->typeSol = typeSol;
         call SendMsg.send(0, sizeof(HumidMessage), &data);             } }
else
     if (TOS_LOCAL_ADDRESS ==46) {
     cmp5=cmp5+1;
     somme5=somme5 +message->valueHum;
     sommeT6 =sommeT6 +message->valueTemp;
     if (cmp5==8){
         HumidMessage *message6 = (HumidMessage *)data.data;
         atomic message6->valueHum = somme5/8;
        atomic message6->valueTemp = sommeT6/8;
       atomic message6->source = 46; // la zone N° 06
       atomic message6->typeSol = typeSol;
 call SendMsg.send(0, sizeof(HumidMessage), &data);       } }
if (TOS_LOCAL_ADDRESS ==55) {
 cmp6=cmp6+1;
 somme6 =somme6 +message->valueHum;
```

```
 sommeT7 =sommeT7 +message->valueTemp;
 if (cmp6 ==8){
      HumidMessage *message7 = (HumidMessage *)data.data;
      atomic message7->valueHum = somme6/8;
      atomic message7->valueTemp = sommeT7/8;
      atomic message7->typeSol = typeSol;
      atomic message7->source = 55; // la zone N° 07
      call SendMsg.send(0, sizeof(HumidMessage), &data); } }
else
     if (TOS_LOCAL_ADDRESS ==64) {
     cmp7=cmp7+1;
     somme7=somme7 +message->valueHum;
     sommeT8 =sommeT8 +message->valueTemp;
     if (cmp7 ==8){
        HumidMessage *message8 = (HumidMessage *)data.data;
        atomic message8->valueHum = somme7/8;
        atomic message8->valueTemp = sommeT8/8;
        atomic message8->source = 64; // la zone N° 08
        atomic message8->typeSol = typeSol;
       call SendMsg.send(0, sizeof(HumidMessage), &data);              } }
else
     if (TOS_LOCAL_ADDRESS ==0) { // au niveau de sink
     int i;
     moy[cmp]=message->valueHum;
     moyTemp[cmp]=message->valueTemp;
     zonn[cmp]=message->source;
    sol[cmp]=message->typeSol;
    cmp=cmp+1;
   if (cmp==8){
         for (i=0;i<8;i++){
            if (sol[i] == 0) // un sol sablé
             dbg(DBG_USR1, "***j'ai la lumiére de 170 Lux et un sol du type Sablé dans la zone
                                %d***", zonn[i]);
          else
                if (sol[i] == 1) // un sol Limoneux
                dbg(DBG_USR1, "***j'ai la lumiére de 170 Lux et un sol du type Limoneux dans la zone      %d***",
zonn[i]);
                else  // c-à-d (sol[i] == 2) un sol argile- Limoneux
                dbg(DBG_USR1, "***j'ai la lumiére de 170 Lux et un sol du type Argile- Limoneux dans la zone
%d***", zonn[i]);
dbg(DBG_USR1, "***MOYENNE ZONE %d  en Humidite: %d*** et en temperature %d ***",
zonn[i],moy[i],moyTemp[i]);  }
for (i=0;i<8;i++){
   if(((moy[i])<10) && (((moyTemp[i])<40) &&(moyTemp[i])>35) && ((sol[i] == 0 ))){
  dbg(DBG_USR1, "la zone %d au sol Sablé est une zone sêche",zonn[i]);
  dbg(DBG_USR1, "SVP arroser la zone %d au sol Sablé avec une quantitée importante d'eau",zonn[i]);}
else
if((((moy[i])>10)&&((moy[i])<15)) && (((moyTemp[i])<30) &&(moyTemp[i])>35) && ((sol[i] == 1 ))){
    dbg(DBG_USR1, "la zone %d au sol Limoneux est une zone sêche",zonn[i]);
    dbg(DBG_USR1, "SVP arroser la zone %d au sol Limoneux avec une quantitée importante d'eau",zonn[i]);}
if(((moy[i])<20) && (((moyTemp[i])>25) &&(moyTemp[i])<30) && ((sol[i] == 2 ))){
   dbg(DBG_USR1, "la zone %d au sol Argile- Limoneux est une zone sêche",zonn[i]);
   dbg(DBG_USR1, "SVP arroser la zone %d au sol Argile-Lumineux avec une quantitée importante d'eau",zonn[i]);}
else
    dbg(DBG_USR1, "la zone %d est une zone humide",zonn[i]);      }}}
 return m;
}}
```

Résumé

Les réseaux de capteurs sont des réseaux formés d'un grand nombre de nœuds capteurs qui se collaborent entre eux pour fournir un service bien déterminé. Cependant l'impossibilité d'une intervention humaine, a poussé les utilisateurs à s'intéresser à ces réseaux pour la surveillance et la sécurité de l'environnement ainsi la collection des données. Dans ce mémoire nous présentons une approche pour mettre en place un réseau de capteurs pour détection des zones sèches et les irriguer par la suite, notre application a pour but de tester, de valider et de simuler le fonctionnement du réseau et sa principale fonction est de vérifier le comportement des capteurs développés avant même de les avoir déployé en situation réelle.

Abstract

Sensor networks are networks containing a large number of sensor nodes that collaborate to provide a service that have specifieds properties. But the impossibility of human intervention, prompted users to be interested in these networks for surveillance and safety of environment and collection of data too. In this memory we present an approach to establish a network of sensors for detection of dryland and irrigate them thereafter, our application was designed to test, validate and simulate the network and its main function is to verify the behavior of sensors developed even before you deploy in a real situation.

ملخص

شبكات الاستشعار والشبكات تحتوي على عدد كبير من أجهزة الاستشعار التي تتعاون على توفير خاصية الخدمة المحددة. لكن استحالة التدخل البشري، تدفع المستخدمين للاهتمام بهذه الشبكات لمراقبة وسلامة البيئة وجمع البيانات أيضا. في هذه المذكرة نقدم نهجا لإنشاء شبكة من أجهزة الاستشعار للكشف عن الأراضي الجافة وريها بعد ذلك، تطبيقنا يهدف لاختبار والتحقق من صحة ومحاكاة الشبكة التي وظيفتها الرئيسية التحقق من سلوك أجهزة الاستشعار المتقدمة حتى قبل أن تنشر في الوضع الحقيقي.