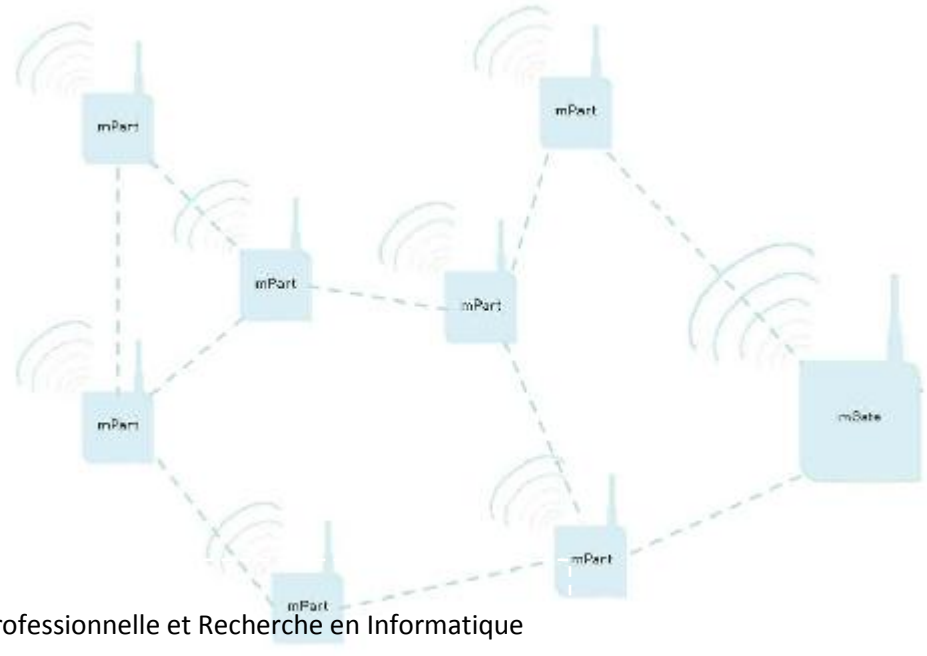
UNIVERSITÉ MONTPELLIER 2
SCIENCES ET TECHNIQUES

Développement d'une bibliothèque de capteurs

Rapport



Master Informatique
Spécialité : Informatique professionnelle et Recherche en Informatique
Parcours : CASAR

Encadré par :
Anne-Elisabeth Baert
Vincent Boudet
Réalisé par :
FARES Abdelfatah

25/04/2008

Remercîment

Je tiens à remercier en premier lieu les membres du jury. Michel LECLERE maitre de conférences en informatique et Marc NANARD m'ont fait l'honneur de juger mes travaux.

Je remercie particulièrement mes encadrant Anne-Elisabeth Baert maître de Conférences en Informatique et Vincent boudet maitre de conférences en informatique qui m'ont accueilli, accompagné et conseillé tout au long de ce parcours.

Un merci très respectueux à Mr julien Champ de sa patience et ses conseilles précieux.

Merci à tous mes amis qui m'ont poussé à dépasser tous les problèmes et aller jusqu'au bout et terminé ce projet.

Un grand merci a ma famille qui ma toujours soutenue tous le long de mon parcours.

I. Introduction général

Ce travail s'inscrit dans le cadre des travaux de l'équipe de recherche de l'Université de Montpellier II, c'est un travail d'étude et de recherche pour le master informatique, concernant le domaine des réseaux de capteurs. L'objectif du projet est d'explorer un certain nombre de technologies concernant la mise en œuvre de ces réseaux, afin de mettre en place une bibliothèque de test et d'expérimentation pouvant reproduire les échanges d'informations entre les différents capteurs.

Les réseaux de capteurs sans fil, sont destinés à relever des informations dans des environnements hostiles auxquels l'homme n'a pas toujours accès. C'est pourquoi on considère qu'une fois qu'ils sont déployés, les capteurs sont autonomes. Leur durée de vie est donc la durée de vie de leur batterie. Le facteur énergie est donc au centre de toutes les préoccupations sur les capteurs : protocoles de routage "économiques", technologie sans fil adaptée, redondance des équipements. Il faut minimiser les dépenses énergétiques, car l'énergie est une contrainte clé dans les réseaux de capteurs.

Notre travail entre dans le cadre de l'étude du problème de routage dans les réseaux de capteurs sans fils. Notre étude offre principalement, une étude synthétique des travaux de recherche qui ont été fait, et qui se font à l'heure actuelle, dans le but de résoudre le problème d'acheminement de données entre les hôtes du réseau avec la prise en compte de consommation d'énergie possible. Comme nous allons voir le problème de routage au sein des WSN est très compliqué, cela est dû essentiellement à l'absence d'infrastructure fixe (foret, mer,…), de toute administration centralisée et à d'autres défis qui doivent être vérifié.

La suite de ce document s'articule de la façon suivante : après avoir bien présenté le domaine des réseaux des capteurs, ou nous dresserons un état de l'art sur les réseaux de capteurs. Puis nous parlerons des différents algorithmes de routage, utilisé pour acheminer l'information et du protocole à état de liens dans un réseau. La partie suivante sera 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. Nous achèverons cette étude par une conclusion dressant le bilan du travail réalisé et abordant les perspectives suite à ce projet.

II. Contexte

Depuis quelques décennies, le besoin d'observer et de contrôler des phénomènes physiques tels que la température, la pression ou encore la luminosité est essentiel pour de nombreuses applications industrielles et scientifiques. Dans le domaine de l'écologie, la surveillance de polluants pourrait considérablement augmenter la qualité de vie dans les villes. Il n'y a pas si longtemps, la seule solution pour acheminer les données du capteur jusqu'au contrôleur central, était le câblage qui avait comme principaux défauts d'être coûteux et encombrant.

Les capteurs sont des dispositifs de taille extrêmement réduite avec des ressources très limité, autonomes, capables de traiter des informations et de les transmettre, via les ondes radio, à une autre entité (capteurs, unité de traitements...) sur une distance limitée à quelques mètres. Les réseaux de capteurs utilisent un très grand nombre de ces capteurs, pour former un réseau sans infrastructure établie. Un capteur analyse son environnement, et propage les données récoltées aux capteurs appartenant à sa zone de couverture.

Les capteurs sans fils coutent cher, et sont difficile a maintenir. C'est pour cela on doit intervenir pour implémenter une application, capable de simuler le comportement des ses entités, bien définir l'architecture de déploiement, choisir l'algorithme de routage qui permet de bien acheminer l'information toute en consommant le moindre cout d'énergie et maintenir les réseaux opérationnels le maximum possible.

III. Cahier de charge

➢ Prestation

A fin de bien simuler les WSN on est amené à reproduire le comportement et le fonctionnement d'un capteur sans fil dans un environnement informatique. Des choix sur les techniques de modélisations, de communication entre capteurs, étaient à faire. Le développement d'un environnement de simulation et d'un exemple simple de réseau de capteurs s'imposa. Il nous fallait prévoir un environnement modulaire permettant de choisir le type de composant à intégrer à la simulation afin de rendre cette simulation fiable. On sera capable de prévoir le bon fonctionnement du réseau, ses performances, son organisation, sa consommation d'énergie, etc.

Tous les capteurs d'une application sont considérés homogènes (i.e. même capacité de calcul, de communication et d'énergie). Dans notre l'application, certains capteurs on des rôles différents, générant une architecture hétérogène.

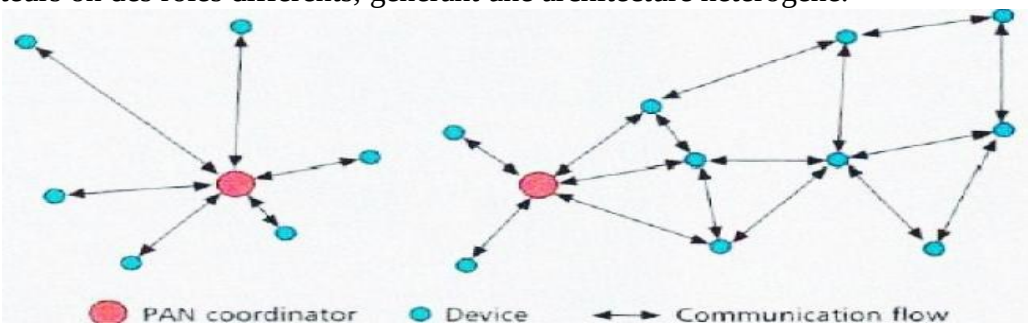


Figure 1 : WSN hiérarchique

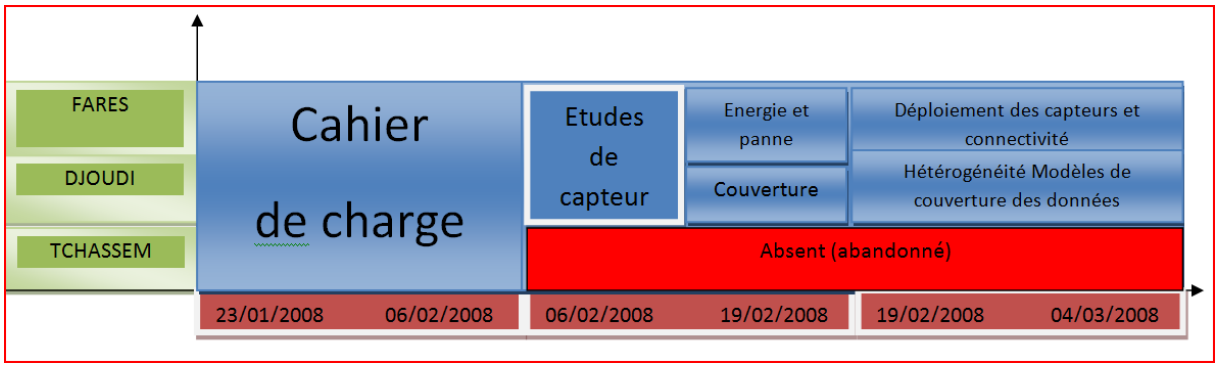
Dans un WSN hiérarchique (figure 1), certains capteurs sont déclarés « chef » de leur groupe. Le routage vers les stations de base est alors traité par ces derniers.

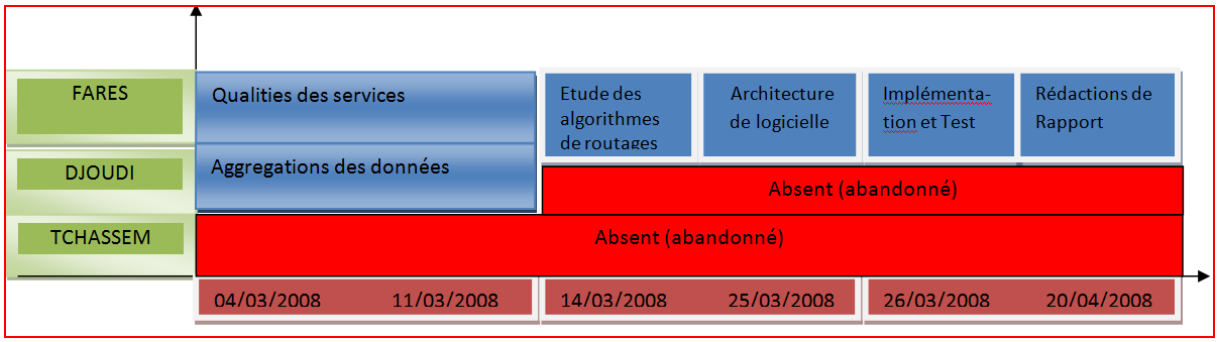
➢ Méthode de travail

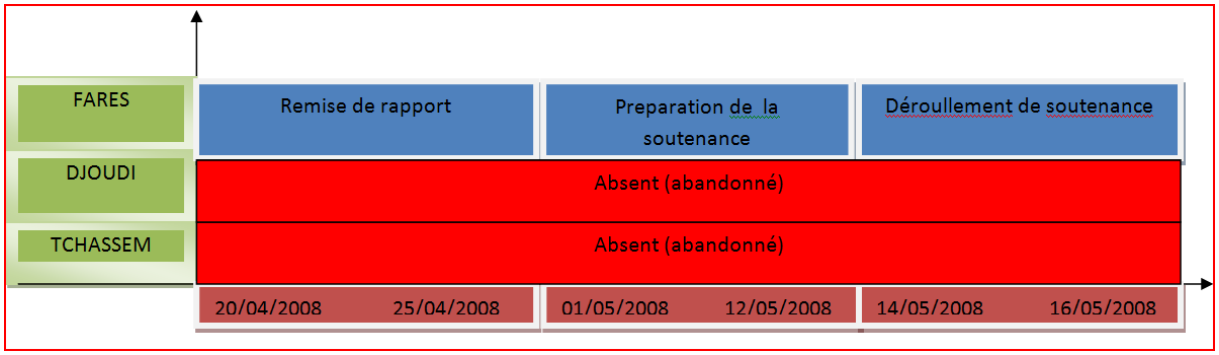
En accord avec mes tuteurs nous avons décidé de nous organiser de la manière qui nous semblait la plus appropriée dans le cadre de ce projet.

- J'ai créé une adresse mail spécifique pour le projet «projet_capteur@hotmail.fr » les titreurs « ter.univmontp2@gmail.com ».
- Etant donné que le projet contient des phases clés, un rapport écrit était remis à mes encadrant à la fin de chaque partie, suivi d'une présentation orale, pour conserver un bon axe de travail.
- Un diagramme de Gantt a été réalisé montrant le déroulement de projet et la répartition de tache. Ce diagramme peut évoluer suivant le déroulement de projet.

IV. Organisation de travail







Chapitre 1 : Présentation de réseaux des capteurs sans fils

Ce chapitre consiste à exposer les réseaux des capteurs sans fil : ses caractéristiques, les propriétés de système d'exploitation, l'énergie des capteurs, le déploiement et l'ordonnancement de réseaux des capteurs sans fil.

V. Réseaux de capteur sans fils

1. Introduction

Les avancées technologiques récentes confortent la présence de l'informatique et de l'électronique au cœur du monde réel. De plus en plus d'objets se voient ainsi équipés de processeurs et de moyens de communication mobiles, leur permettant de traiter des informations mais également de les transmettre.

Les réseaux de capteurs sans-fil entrent dans ce cadre. En effet, ceux-ci sont constitués d'un ensemble de petits appareils, ou capteurs, possédant des ressources particulièrement limitées, mais qui leur permettent néanmoins d'acquérir des données sur leur environnement immédiat, de les traiter et de les communiquer.

2. Capteurs sans fils

Les capteurs sont des dispositifs de taille extrêmement réduite avec des ressources très limitées, autonomes, capable de traiter des informations et de les transmettre, via les ondes radio, à une autre entité (capteurs, unité de traitements...) sur une distance limitée à quelques mètres.

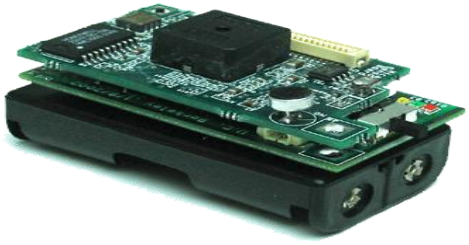


Figure 2 : Capteur sans fils

Les réseaux de capteurs utilisent un très grand nombre de ces capteurs, pour former un réseau sans infrastructure établie. Un capteur analyse son environnement, et propage les données récoltées aux capteurs appartenant à sa zone de couverture. Chaque capteur relayant l'information sur sa propre zone de couverture, le réseau se trouve entièrement couvert.

3. Architecture de capteur sans fils

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

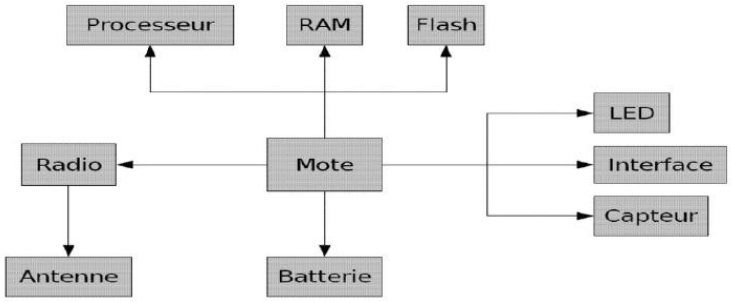


Figure 3 : Architecture d'un capteur sans fils

On peut voir sur la figure 1 les différents composants qui constituent un capteur. Pour être plus précis chaque groupe de composants possède son propre rôle :

- **Unité de traitement**

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

- **Unité de transmission**

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

- **Unités de captage**

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

- **Unités de control d'énergie**

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

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.

4. Cas d'utilisation de WSN

Comme beaucoup de technologie, le développement des *WSN* a été suscité par des besoins militaires. L'absence de câbles entre les nœuds, leur faible taille, le nombre élevé de *motes* déplorables pour couvrir une zone étendue répondent à ces critères

Puis, la diminution des coûts de fabrication des capteurs, ainsi que la réduction de leur taille a entraîné une utilisation dans des applications civiles. Ils peuvent par exemple être Utilisés à des fins de surveillance environnementale. En effet, des capteurs peuvent être placés Dans des régions glaciaires ou tropicales afin de suivre de manière précise les effets du réchauffement de la planète, les changements climatiques ou l'augmentation de la pollution. Ils peuvent également être employés pour une surveillance de l'habitat, car leur déploiement, par exemple en montagne, pourrait permettre de recenser les animaux fréquentant un territoire donné.


Surveillance militaire


Surveillance environnementale

Figure 4 : Cas d'utilisations

Des applications industrielles peuvent également les adoptes. Une idée d'utilisation pourrait être de placer ces instruments à des points spécifiques, par exemple aux points d'usure des machines ce qui permettrait d'émettre des alertes lorsque leur état général se dégrade. Enfin, certains envisagent d'implanter des senseurs dans le corps humain. Ce qui permettrait de contrôler l'état de santé de patients, et ainsi d'adapter leur traitement, de prévenir la dégradation de leur état de santé, et par conséquent d'être capable d'anticiper une hospitalisation en urgence. Mais ceci n'est pour l'instant qu'une utilisation future, qui dépend encore des progrès à venir de cette technologie.

VI. Caractéristiques des réseaux des capteurs

1. Système d'exploitation TinyOS

➢ Présentation

TinyOS est un système d'exploitation Open Source pour les réseaux des capteurs, conçu par l'université américaine de BERKELEY. Le caractère open source permet à ce système d'être régulièrement enrichie par une multitude d'utilisateurs. Sa conception a été entièrement réalisée en NesC, langage orienté composant syntaxiquement proche du C. Il respecte une architecture basée sur une association de composants, réduisant ainsi la taille du code nécessaire à sa mise en place. Cela s'inscrit dans le respect des contraintes de mémoires qu'observent les capteurs, pourvus de ressources très limités dues à leur miniaturisation.

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

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

➢ Propriétés de OS

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

- **Evénementiel** : Le fonctionnement d'un système basé sur TinyOS s'appuie sur la gestion des évènements qui ce déclenche. Ainsi, l'activation de tâches, leur interruption ou encore la mise en veille du capteur s'effectue à l'apparition d'évènements, ceux-ci ayant la plus forte priorité. Ce fonctionnement évènementiel (event driven) s'oppose au fonctionnement dit temporel (time driven), où les actions du système sont gérées par une horloge donnée.
- **Non préemptif** : Le caractère préemptif d'un système d'exploitation précise si celui-ci permet l'interruption d'une tâche en cours. TinyOS ne gère pas ce mécanisme de préemption entre les tâches, 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.
- **Pas de temps réel** : Lorsqu'un système est dit « temps réel » celui ci gère des niveaux de priorité dans ses tâches, permettant de respecter des échéances données par son environnement. Dans le cas d'un système strict, aucune échéance ne tolère de dépassement contrairement à un système temps réel. TinyOS se situe au-delà de ce second type, car il n'est pas prévu pour avoir un fonctionnement temps réel.

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

2. Energie de capteur sans fils

➢ Introduction

Les capteurs sans fils sont des éléments indépendants les uns des autres, comme leur nom l'indique. Par conséquent, ils doivent également disposer d'une alimentation autonome. Leur durée de vie est limitée par la durée de vie de leur batterie.

Cette contrainte forte a une influence majeure sur l'ensemble des techniques, mises en place pour le déploiement de tels réseaux. Un effet majeur de cette limitation énergétique est la limitation maximale des transmissions par voie hertzienne, très coûteuses. Il est donc primordial d'effectuer tant que possible le traitement de l'information localement au niveau du nœud. L'enjeu est donc d'étendre la durée de vie du système et sa robustesse, en cas de chute de certains nœuds seulement. Les problématiques sont donc très éloignées de celles des réseaux classiques, telle la maximisation du débit.

Dans les réseaux de capteur sans fils, il faut assurer une consommation répartie de l'énergie au sein du réseau. Cet énergie et consommé par les diverses fonctionnalités de réseaux qui sont donc par ordre décroissant de consommation d'énergie

Radio (Communication)
Protocoles (MAC, routage)
CPU (calcul, agrégation)
Acquisition

➢ Problématique

L'enjeu d'énergie et capital dans les réseaux de capteur, pour augmenter l'autonomie de capteur il faut agir sur plusieurs paramètres :

Sur quels paramètres est-il possible d'agir ?

Point de vue « circuit »

➢

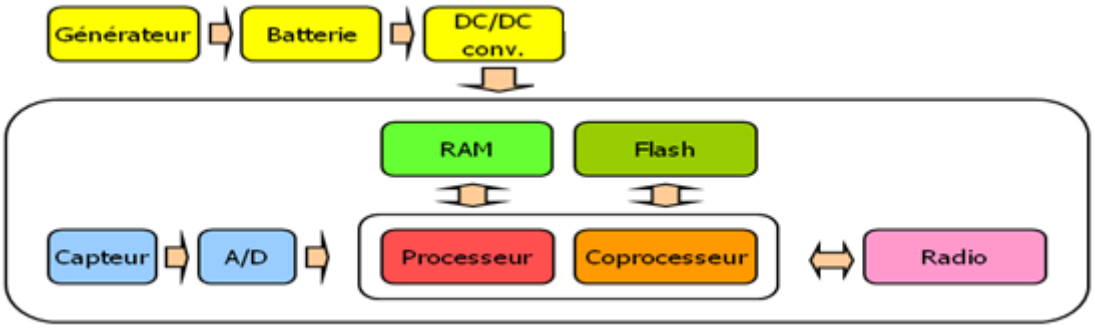


Figure 5 : Circuit de capteur

L’interdépendance des paramètres est un casse-tête pour réaliser une optimisation

- Si on augmente le taux de transmission
  - La probabilité de collision diminue
  - Le taux d’erreur augmente
  - La consommation augmente
- Si on augmente la puissance de la correction d’erreur
  - Le taux d’erreur diminue
  - La probabilité de collision augmente
  - La consommation augmente
- Si on augmente la puissance d’émission
  - Le taux d’erreur diminue
  - La probabilité de collision augmente
  - La consommation augmente

La consommation énergétique du module de surveillance dépend énormément du matériel employé et de la nature du phénomène observé. L’économie d’énergie obtenue par la mise en veille de certains nœuds pour l’observation est donc très variable.

VII. Ordonnancement de réseaux de capteur

Les réseaux de capteurs sont généralement denses et redondants. En effet, suivant l’application, on déploiera plus ou moins des capteurs dans un souci d’allongement de la durée de vie de l’application. À tout moment, il existe donc des capteurs qui observent une même portion de la zone de déploiement. Cette redondance est exploitée par l’ordonnancement d’activité : Ordonnancer l’activité dans un réseau de capteurs consiste à alterner les charges de façon à épuiser les nœuds équitablement. Pendant qu’une partie participe à l’application, les autres sont dans un mode passif, économisant ainsi leur énergie.

1. Gestion d'ordonnancement

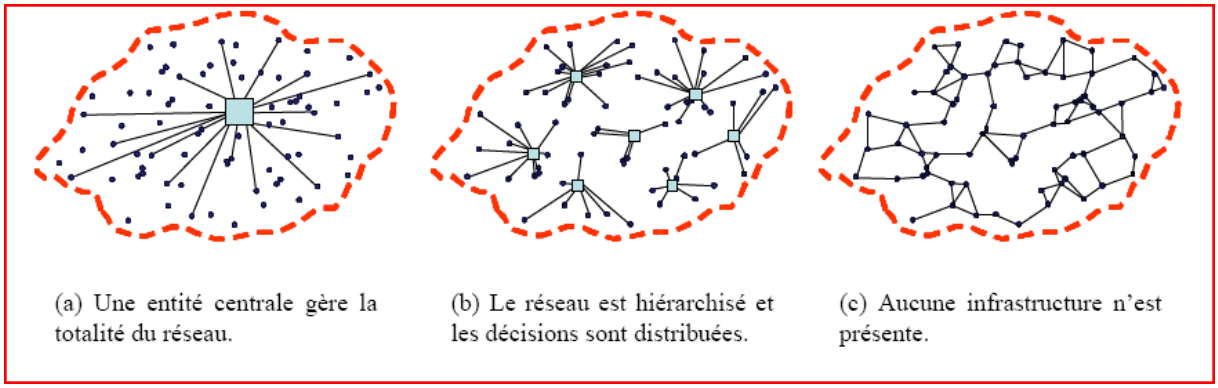


Figure 6 : Trois approches algorithmiques pour l'ordonnancement d'activité

L'ordonnancement d'activité peut se faire de diverses façons. Nous distinguons ici les approches centralisées (où une entité centrale connaît chaque nœud et est capable d'influer sur chacun pour lui assigner ses tâches) des approches hiérarchiques (une vision hiérarchisée du réseau où l'autorité centrale est démultipliée selon plusieurs entités responsables d'une sous-partie du réseau) et des approches localisées, par conséquent totalement décentralisées, dans lesquelles un comportement global cohérent doit être obtenu à partir de décisions prises localement.

VIII. Modélisation de réseaux de capteur

Un réseau sans fils est représenté par un graphe $G = (V,E)$, avec V l'ensemble des sommets (les *nœuds* du réseau) et E *(appartient à)* V^2 l'ensemble des arêtes donnant les communications possibles :

il existe une paire ordonnée $(u,v) \in E$ si le nœud u est capable de transmettre des messages à v.L'ensemble de voisins $N(u)$ d'un nœud u est défini par :

$$N(u) = \{v \in V \mid v \neq u \wedge (u,v) \in E\}.$$

La taille de $N(u)$ correspond au degré d'un nœud. Etant donné un graphe $G = (V,E)$, le modèle du disque unitaire définit l'ensemble E des arêtes :

$$E = \{(u,v) \text{ (appartient à) } V_2 \mid u \neq v \wedge \text{dist}(u,v) \leq CR\}$$

CR étant la portée de communication des nœuds et dist(u,v) donnant la distance euclidienne entre u et v. Deux nœuds sont dits connectés si la distance qui les sépare est inférieure à CR ou s'il existe un chemin multi-sauts composé de nœuds intermédiaires capables de communiquer deux à deux. Lorsque plusieurs chemins, k par exemple, existent entre deux nœuds, ces derniers sont dits k-connectés. Un réseau est k-connexe si chaque paire de nœuds est k-connectées.

Chaque capteur possède également une portée de surveillance SR. Nous notons $S(u)$ la surface observable par un nœud u. La forme de $S(u)$ dépend du capteur à proprement dit, et il est extrêmement difficile de la caractériser, tant elle dépend du matériel utilisé et de l'information qui est relevée. Comme dans la grande majorité des travaux existants dans la littérature, nous supposons ici que $S(u)$ est un disque de centre u et de rayon SR :

$$S(u) = \{p \in A \mid \text{dist}(u, p) \leq SR\}$$

Où p est un point physique de la zone de déploiement notée A. La surface couverte par un ensemble de nœuds $V = \{v_1, v_2, \ldots, v_n\}$ est notée $S(V)$: $S(V) = \bigcup_{i=1}^{i=|V|} S(v_i)$

Enfin, nous noterons $S_{-1}(p)$ l'ensemble des nœuds appartenant à V capables de surveiller le point physique p :

$$S^{-1}(p) = \{u \in V \mid p \in S(u)\}.$$

IX. Déploiement de capteur

Les capteurs sont au préalable déployés sur une zone à surveiller. Pour satisfaire de nouvelles contraintes ou pour pallier des pannes, un déploiement de nœuds supplémentaires, dit *itératif*, peut être requis. Différents modes de déploiement sont envisageables et dépendent essentiellement de l'application de surveillance. Une fois déployés, nous supposons que les capteurs sont statiques.

1. Déploiement déterministe

Lorsque l'environnement est accessible ou connu, il est possible de placer précisément les nœuds sur la zone. Dans le problème que nous étudions, il est alors possible de programmer leurs activités au préalable. C'est ainsi, par exemple, que sont mis en place les capteurs chargés de réguler la climatisation d'un immeuble ou de surveiller les constantes médicales de malades à distance. On parle alors de déploiement déterministe.

2. Déploiement aléatoires

L'utilisation des capteurs dans des zones inaccessibles ou sensibles rend impossible un déploiement déterministe, au cours duquel chaque objet serait placé à une position prédéterminée.Les nœuds peuvent alors être déployés à l'aide de moyens divers. Il a souvent été question d'un déploiement aléatoire des capteurs, effectué comme un jeté de graines. Il semble pourtant difficile, vu la fragilité des capteurs existants, d'envisager un déploiement par avion par exemple.

Néanmoins, un déploiement aléatoire peut être obtenu à partir d'une distribution de capteurs à des individus. Dans ce travail, nous supposerons des réseaux déployés aléatoirement. Une fois disséminés, il est couramment admis que les capteurs sont statiques.

X. Conclusion

Après avoir étudié les réseaux des capteurs sans fil et de bien les présenter, détailler l'architecture, et expliquer le fonctionnement de son système d'exploitation TinyOs. Un choix de protocole de routage et d'un algorithme adéquat pour le routage d'information dans notre réseau ainsi qu'une approche algorithmique pour l'ordonnancement d'activité s'impose pour bien simulé les réseaux de capteur sans fils. Ceci va être étudié et présenté dans le prochain chapitre.

Chapitre 2 : Protocole et Algorithmes de routage + langage de programmation

Ce chapitre décrit, les algorithmes de routage dans un réseau de capteur sans fils Dijkstra, ASTAR A et Bellman-Ford : présenté leurs principes d'exécution et les schémas algorithmiques, ainsi que la Description de protocole *ospf*, ses caractéristiques et le principe de fonctionnement. De plus un Choix de langage de programmation s'impose.

XI. Algorithmes de routage

1. Algorithme de DIJKSTRA

Cet algorithme n'est utilisable que dans le cas où les coûts des arcs sont tous positifs ou nuls. Il calcule le plus court chemin entre une source « s » et tous les sommets accessibles depuis « s ». On obtient une arborescence, de racine « s » formée par ces plus courts chemins.

➢ Principe

Soit G = <S, A, C> un graphe orienté de « n » capteurs. Soit s le capteur qui va émettre, et G un graphe dont on veut trouver les plus courts chemins aux autres sommets. Cet algorithme doit donc déterminer pour chaque sommet m le chemin le plus court entre s et m dans le graphe G.

L'algorithme de Dijkstra est un algorithme de type glouton[1] : à chaque nouvelle étape, on traite un nouveau sommet. Reste à définir le choix du sommet a traité, et le traitement a lui infligé.

Tout au long du calcul, on va donc maintenir deux ensembles :

- C, l'ensemble des sommets qui restent à visiter ; au départ C=S-{source}.
- D, l'ensemble des sommets pour lesquels on connaît déjà leur plus petite distance à la source ; au départ, D = {source}.

L'algorithme se termine bien évidemment lorsque C est vide.

Comme on connaît maintenant un plus court chemin de « s » vers « m », il faut redéfinir les plus courts chemins vers certains autres sommets dont on connaît déjà. Ce sont les sommets « i » pour lesquels « m » est un prédécesseur.

- Dijkstra peut trouve le plus court chemin d'un nœud source à tous les autres.
- Dijkstra peut trouver le plus court chemin entre 2 sommets, en lançant l'algorithme depuis un nœud source bien définie « x », jusqu'à ce qu'il atteigne un point « y » spécifique.
- Dijkstra peut trouver le plus court chemin entre tous les nœuds d'un graphe en le lançant sur tous ces nœuds.

➢ Algorithmique

Titre: Dijkstra
Entrées: *R = (X;U;d)* un réseau, *s* un sommet.

[1] Un algorithme glouton est un algorithme qui confronté à un choix, choisit ce qui lui semble le

Sorties: *p()* une fonction indiquant la plus courte distance qui sépare un point de *s*, *pred()* une fonction indiquant par quel arc on arrive à un point si on emprunte le plus court chemin.
Variables intermédiaires: *S* et *S'* deux sous-ensembles de sommets, *x* et *y* deux noeuds, *sortie* un booléen.

```
Début
S ← ∅;
S' ← X;

pour tout x ∈ X faire
p(x) ← +oo;
pred(x) ← nil;
fin pour;

p(s) ← 0;
sortie ← faux;

tant que S' ≠ ∅ et non sortie faire
choisir x ∈ S' tel que p(x) = min{p(y) | y ∈ S'};
sortie ← (p(x) = +oo);
S ← S ∪ {x};
S' ← S' - {x};

pour chaque arc u = (x;y) | y ∈ S' faire
si p(y) > p(x) + d(u) alors
p(y) ← p(x) + d(u);
pred(y) ← u;
fin si;
fin pour;
fin tant que;
Fin
```

Figure 7 : Algorithme de Dijkstra SPF

2. Algorithme de ASTAR A*

➢ Description

Cet algorithme n'est qu'une variante de Dijkstra, et subit donc les mêmes restrictions que ce dernier. Mais les modifications qu'il apporte en font un algorithme très puissant et tout aussi flexible que Dijkstra.

La principale différence, est dans le fait qu'il se charge de trouver le plus court chemin entre tous sommets quelconques. Sa puissance, et aussi sa faiblesse comme on le verra, est du au fait qu'il fournit le même résultat que Dijkstra mais en un temps inférieur en sélectionnant les sommets qu'il explore pour aller du sommet source au sommet destination. Il est clair qu'ainsi certains sommets ne sont même pas visité d'ou le gain.

Comme nous l'avons dit, A* doit être en mesure de sélectionner les sommets qui lui semble les plus prometteurs et de les explorer en priorité. Pour cela il fait une estimation du coût restant à partir d'un sommet pour se rendre au sommet de destination.

Ainsi pour chercher le plus court chemin entre deux sommets la différence entre Dijkstra et A* peut être réellement très importante. En général, plus le nombre de sommets du Graphe est grand, plus le gain est important.

➢ Algorithmique

```
procédure A*(G : Graphe, s : sommet)
Init(G, s)
f(s)   h(s)
x   s
TantQue x 6= p faire
Examiner _ (x)
Choisir un sommet ouvert x d'approximation f(x) minimale
FinTantQue
procédure Examiner*(x : Sommet)
Pour tout successeur y de x faire
Si _(x) + c(x, y) < _(y) alors
Ajuster − Arborescence(x, y)
Ouvrir(y)
f(y)   _(y) + h(y)
Finsi
Finpour
Fermer(x)
procédure Ajuster-Arborescence(x, y : Sommet)
_(y)   _(x) + c(x, y)
p(y)   x
```

Figure 8 : Algorthme de A*

3. Algorithme de Bellman-Ford

- **Description**

Cet algorithme à la particularité de trouver comme Dijkstra, le plus court chemin de source vers les autres sommets pour des graphes possédant des coûts négatif. Mais les restrictions quant aux conditions d'utilisation, en font un algorithme peu polyvalent.

- Le graphe ne doit pas posséder de circuit qu'il soit absorbant ou non.
- Le sommet source doit être racine du Graphe.

Néanmoins si ces conditions sont remplies, il s agit d'un algorithme performant.

Comme l'algorithme de Dijkstra, Bellman-Ford utilise la technique de relâchement, diminuant progressivement une estimation du poids d'un plus court chemin depuis l'origine « s » vers chaque sommet « x » du graphe, jusqu'à atteindre la valeur réelle du poids de plus court chemin.

L'algorithme retourne **VRAI** si et seulement si, le graphe ne contient aucun circuit de poids négatif accessible depuis l'origine.

Son principe est assez similaire à Dijkstra.

- **Algorithmique de Bellman-Ford**

```
booléen Bellman_Ford( G, s)
  initialisation ( G, s)  // les poids de tous les sommets sont mis à +infini
                          // le poid du sommet initial à 0 ;
  pour i=1 jusquà Nombre de sommets -1 faire
      pour chaque arc (u, v) du graphe faire
          paux := poids(u) + poids(arc(u, v));
          si paux <poids(v) alors
              pred(v) := u ;
              poids(v) := paux;

  pour chaque arc (u, v) du graphe faire
      si poids(u) + poids(arc(u, v)) <poids(v) alors
      retourner faux
  retourner vrai
```

Figure 9 : Algorithme de Bellman-Ford

4. Conclusion et choix d'algorithmes

Après avoir étudié le trois algorithme de recherche de plus court chemin il faut retenir, qu'il existe une multitude d'algorithmes et que chacun possède ses restrictions ses forces et ses faiblesses.

Un informaticien doit connaître ses caractéristiques plus que l'algorithme lui-même, pour pouvoir être capable de choisir celui qui convient. Donc j'ai essayé, d'après ce que je veux faire dans la simulation de réseau des capteurs sans fils de choisir l'algorithme qui convient le plus a mon application.

L'algorithme de Dijkstra est le plus adapté à notre application, vu les coûts des arcs sont tous nuls, de plus cet algorithme peut trouver le plus court chemin entre tous les nœuds d'un graphe en le lançant sur tous ces nœuds. C'est proprement ce que j'ai fait pour bien simuler les réseaux de capteur : exécuter l'algorithme sur chaque capteur en utilisant **OSPF** « *Open shortest path first* », une implémentation bien connue du monde réel utilisée dans le routage internet.

XII. Protocol de routage « OSPF »

1.Introduction

Le rôle du capteur est d'acheminé des alertes a la station d'émission. Pour cela, il se base sur sa table de routage comportant les éléments nécessaires au transfert du paquet. De plus dans notre application chaque capteur doit posséder une table de routage, contenant tous les numéros de capteur pour lesquels il doit être capable de faire du routage (ses voisins de même surface).

Il existe deux principales familles : les protocoles internes (IGP : Interior Gateway Protocol) qui établissent les tables de routage des routeurs appartenant à une entité unique appelée AS (Autonomous System : système autonome) et les protocoles externes (EGP : Exterior Gateway Protocol) permettant l'échange des informations entre ces systèmes autonomes. Au sein des protocoles internes, il existe deux types : les protocoles à vecteur de distance (Distant Vector Protocol) qui utilisent le saut de routeur comme métrique, et les protocoles à états de liens (Link State Protocol), beaucoup plus performants que les précédents, que nous allons voir en détail à travers le protocole OSPF

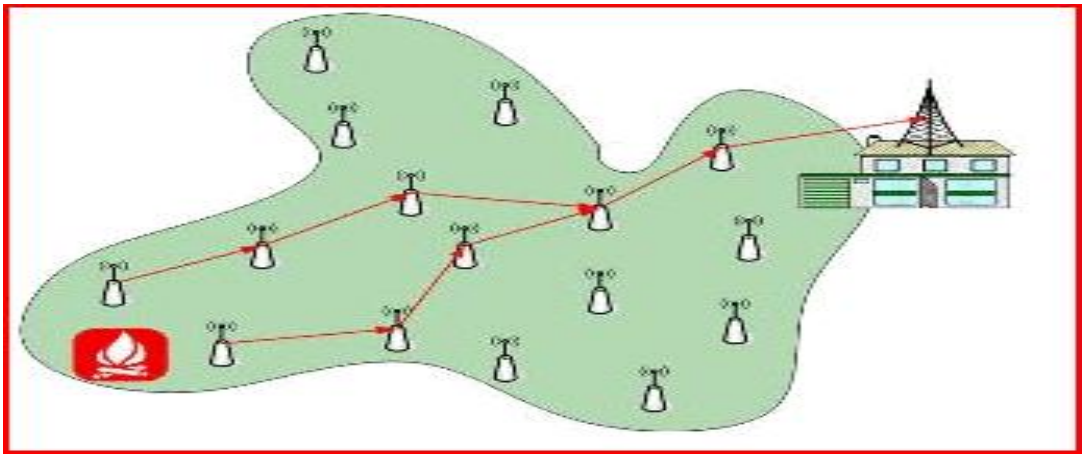


Figure 10 : Routage dans un réseau des capteurs sans fils

2.Description de OSPF

OSPF « *Open shortest path first »* est un protocole de routage dynamique défini par l'IETF. Il a fait l'objet d'un historique relativement complexe de RFCs (Voir ospf RFC List). Ce protocole a deux caractéristiques essentielles :

- Il est ouvert : c'est le sens du terme *Open* de OSPF. Son fonctionnement est connu de tous.
- Il utilise l'algorithme SPF pour *Shortest Path First*, plus connu sous le nom d'algorithme de Dijkstra, afin d'élire la meilleure route vers une destination donnée.

Son principe est simple, chaque capteur détermine l'état de ses connections (liens) avec les capteurs voisins, il diffuse ses informations à tous les capteurs appartenant à une même zone. Ces informations forment une base de données, qui doit être identique à tous les capteurs de la même zone. Sachant qu'un système autonome (AS) est constitué de plusieurs zones, l'ensemble de ces bases de données représente la topologie de l'AS. A partir de cette base de données, chaque capteur va calculer sa table de routage grâce à l'algorithme,SPF(Short-Path-First).

Pour que les capteurs s'échangent les informations de routage, ils doivent être adjacents, c'est à dire qu'ils doivent se découvrir les uns les autres. Cette adjacence va se construire grâce au protocole.

3.Fonctionnement avancé de OSPF

La table de routage est obtenue à la fin grâce à l'application de l'algorithme SPF (Short Path First), sur une base d'information décrivant les liens qui unissent les capteurs d'une area.

Un lien est une interface de capteur et son état est la description de cette interface. Cette base est nommée Table de convergence, elle est identique à tous les capteurs de la zone. Au démarrage, un capteur doit se faire connaître des autres, puis il génère table représentant tous les états de liens de voisinage du capteur.

Cet échange d'état de lien entre les capteurs se fait par inondation (flooding). Des mises à jour d'état de lien (Lin State Update) permettent de mettre à niveau tous les capteurs. Lorsque les bases de données sont synchronisées (identiques entre tous les capteurs de l'area), chaque capteur va calculer " l'arbre du chemin le plus court " en appliquant l'algorithme SPF (algorithme de Dijkstra).Il construira ainsi sa table de routage (routing table ou forwarding table).

➢ Déroulement du processus :

- **ETAPE 1 :** Découverte des voisins : (adjacence des capteurs-Down state) Pas d'échange d'informations, attente d'un paquet HELLO. (Init state) Les capteurs envoient des paquets HELLO (toutes les 10s) pour établir une relation avec son voisin. Dès qu'il reçoit un HELLO il passe à l'état suivant. (Two-way state) Deux possibilités : soit il n'y a que deux

capteurs (liaison point à point), alors les routeurs deviennent adjacents (on passe à l'étape 3), soit il y a plusieurs capteurs, on passe à l'étape 2.

- **ETAPE 2** : Découverte de route : Etablissement d'une relation entre les capteurs, (Exchange state) Les capteurs décrivent leurs Link-Database aux autres. C'est le serveur qui initie l'échange de paquets contiennent une description de la LDB (Link-State Database) avec un numéro de séquence. Les routeurs confirment la réception par des paquets comportant le numéro de séquence. Chacun compare ses informations avec les informations reçues, si ces dernières sont plus récentes une mise a joue est effectué.

- **ETAPE 3** : Solution des routes (table de routage) Lorsque le capteur a établit sa Link-state database, il peut créer sa table de routage. Il utilise pour cela l'algorithme SPF qui tient compte de la bande passante du lien (voir algorithme SPF Dijkstra).

- **ETAPE 4 :** Maintien des tables de routage : Lorsqu'il y a un changement d'état de lien (par exemple si une capteur ne reçoit plus de paquet HELLO d'une autre capteur, elle considère le lien " down "), le routeur envoie une LSU avec les nouvelles informations à ses voisins. Ceux-ci inondent alors de LSU les autres capteurs, de nouvelles tables de routage sont crées. Si aucun changement topologique n'intervient, les informations sont rafraichies, les LSA ont, par défaut, une durée de vie de 30 minutes.

4.Conclusion

OSPF a été développé pour palier aux nombreux problèmes de RIP (protocole de routage) et répondre au besoin de routage sur des grands réseaux.

Ses principaux avantages sont :

- convergence rapide, pas de limite des capteurs.
- supporte VLSM et CIDR pour réduire les routes.
- métrique précise (en fonction de la bande passante).
- répartition de charge (load balancing) grâce à une gestion de plusieurs routes pour une même destination.
- utilisation du multicast et mise à jour incrémentielle et non entières.

Par contre OSPF nécessite pour ses calculs une consommation de ressources processeur et mémoire très importante sur les routeurs.

5. Langage de programmation

Java	C++
Généralités	
sans préprocesseur	avec préprocesseur
sans opérateur,	avec opérateur,
sans fonction à nombre variable d'arguments,	avec fonction à nombre variable d'arguments,
avec étiquette sur le *break* et le *continue*	sans étiquette sur le *break* et le *continue*
sans *const*	avec *const*
sans *goto*	avec *goto*
sans variable globale	avec variable globale
A propos de l'objet	
langage objet "pur"	langage orienté objets
toute fonction est une méthode d'instance ou de classe	il peut-y avoir des fonctions non rattachées à une classe
sans héritage multiple	avec héritage multiple
sans type paramétré	avec types paramétrés (templates)
sans surcharge d'opérateurs	avec surcharge d'opérateurs
"liaison dynamique" de toutes les méthodes (sauf celles déclaréees final).	seules les *virtual* fonctions sont liées dynamiquement
À propos des types primitifs	
tout est objet sauf les types primitifs les types primitifs sont portables (big-Endian)	il existe aussi des types struct, union, enum, tableaux. . . les types primitifs sont "plateforme-dépendants"
caractère 16-bits Unicode	caractère 8-bits ASCII
avec type booléen	sans type booléen
toute conditionnelle est une expression booléenne	un résultat entier est assimilé à une expression booléenne
initialisation automatique	initialisation à la charge du programmeur
À propos des pointeurs et structures de données	
les objets sont manipulés par référence, pas de manipulation explicite ni d'arithmétique de pointeur les références sur les tableaux ne peuvent pas être manipulées comme des pointeurs.	il existe les opérateurs * -> &, les tableaux manipulés avec l'arithmétique sur les pointeurs sans test automatique de débordement, tableaux multidimensionnel dont la taille est fixée à la declaration.
Avec test automatique de débordement de tableaux.	les chaînes de caractères sont des tableaux de caractères
objet *String*	terminées par un zero
concaténation via l'opérateur +	concaténation via une primitive
sans *typedef*	avec *typedef*
En vrac	
API réseau et support du *multithreading* en natif	pas d'API réseau ni de support du *multithreading* en natif
avec *garbage collector*	sans *garbage collector*
pré-compilé puis interprété	compile
portable	dependant

Tableau 1 : Comparaison entre Java et C++

Java est très proche du langage C++ étant donné qu'il a quasiment la même syntaxe. Toutefois Java est plus simple que le langage C++ bien qu'il s'en inspire, car les caractéristiques critiques du langage C++ (celles qui sont à l'origine des principales erreurs) ont été supprimées.

Toutefois Java est beaucoup moins rapide que le langage C++, il perd en rapidité ce qu'il gagne en portabilité.

Je décidé d'implémenté l'application avec le langage C++ vue mon bon niveaux de connaissance de ce langage.

6. Architecture adopté

Apres avoir étudié les différents algorithmes de routage, le protocole de reconnaissance de réseaux, et fait le choix de langage de programmation, le choix d'architectures de déploiement des capteurs sans fils s'avers difficile,

J'ai décidé de choisir une architecture centralisé basé sur une entité centrale qui gère la totalité du réseau. Cette architecture est illustré comme le montre la figure suivante.

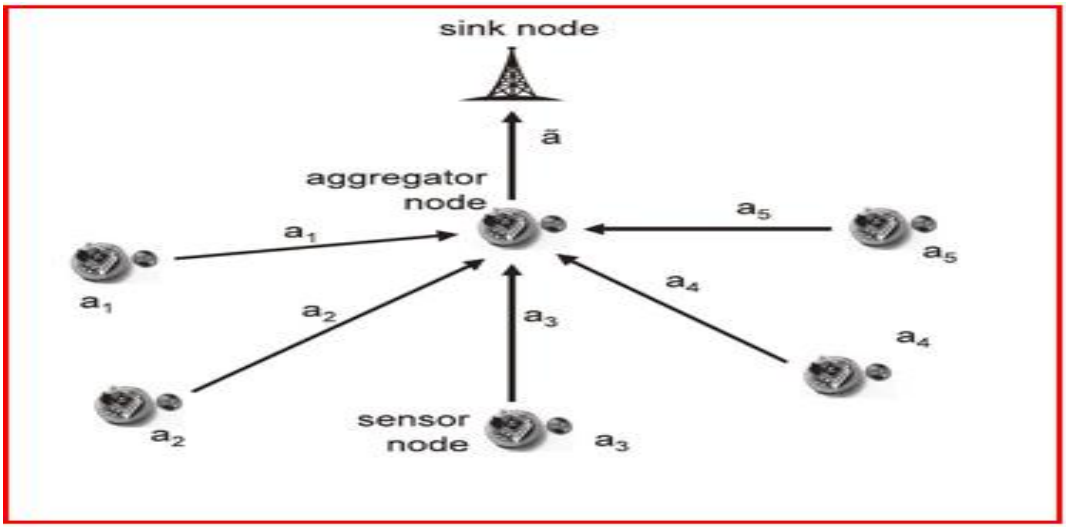


Figure 11 : Architecture centralisé

Dans les solutions d'ordonnancement centralisées, il faut pouvoir bénéficier d'une entité centrale ayant vue sur tout le réseau. Cette entité, représentée par un carré sur la figure 16, peut ainsi assigner à chaque objet un rôle. Les changements de topologie dus aux éventuelles pannes ou à la mobilité des nœuds sont scrutés de façon à pouvoir informés les objets concernés.

XIII. Conclusion

Dans ce chapitre je choisie le principaux mécanisme de fonctionnement, on choisissant l'algorithme de plus court chemins Dijkstra, le protocole de routage *ospf,* le langage de programmation C++ et l'architecture de déploiement « centralisé ». le prochain chapitre va présenté les étapes de mise en place de la bibliothèque de capteurs sans fils

Chapitre 3 : Réalisations de logiciel →Simulateur d'un réseau des capteurs sans fils.

Dans ce chapitre on présente des simulateurs qui existent sur le marché, leurs avantages et les limites, les contraintes matériels et les différentes étapes pour la réalisation de logiciel.

XIV. Mise en place de projet

1. Introduction

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

Dans 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.

Pour la réalisation de ce projet nous avons procédé de la manière suivante.

2. Simulateur de réseaux de capteur

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

- **ATEMU**

ATEMU est un émulateur de capteur basé sur un type de capteur existant, le mica2. Il possède une interface graphique nommé xatdb qui permet de visualiser les évènements qui se produisent sur chaque capteur et qui simplifie les actions de débogage.

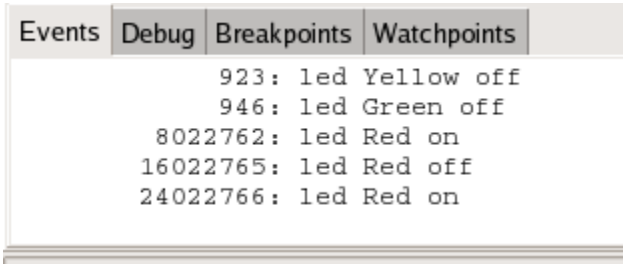


Figure 12 : Simulateur d'une application avec ATEMU

La simulation permet de lancer plusieurs applications pouvant être différentes, et ainsi de simuler un réseau de capteurs, et d'afficher les événements se déroulant sur chaque capteur.

a. Avantages

ATEMU permet de simuler un réseau de capteurs relativement fidèlement. Il peut simuler un grand nombre de capteurs. Xadtb, la fenêtre graphique d'ATEMU permet de visualiser les évènements se produisant sur un capteur choisi, d'afficher les messages de

débogage et d'insérer des points d'arrêts ce qui permet de concevoir des applications fiables.

b. Limites

Lorsque le réseau simulé est composé de beaucoup de capteurs, il devient très difficile de visualiser tous les échanges entre les capteurs composant le réseau. La visualisation ne permet pas d'afficher simultanément les évènements de plusieurs capteurs. Il faut donc naviguer entre les onglets pour avoir un aperçu global de l'activité du réseau.

➢ TOSSIM

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

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

a. Description

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.

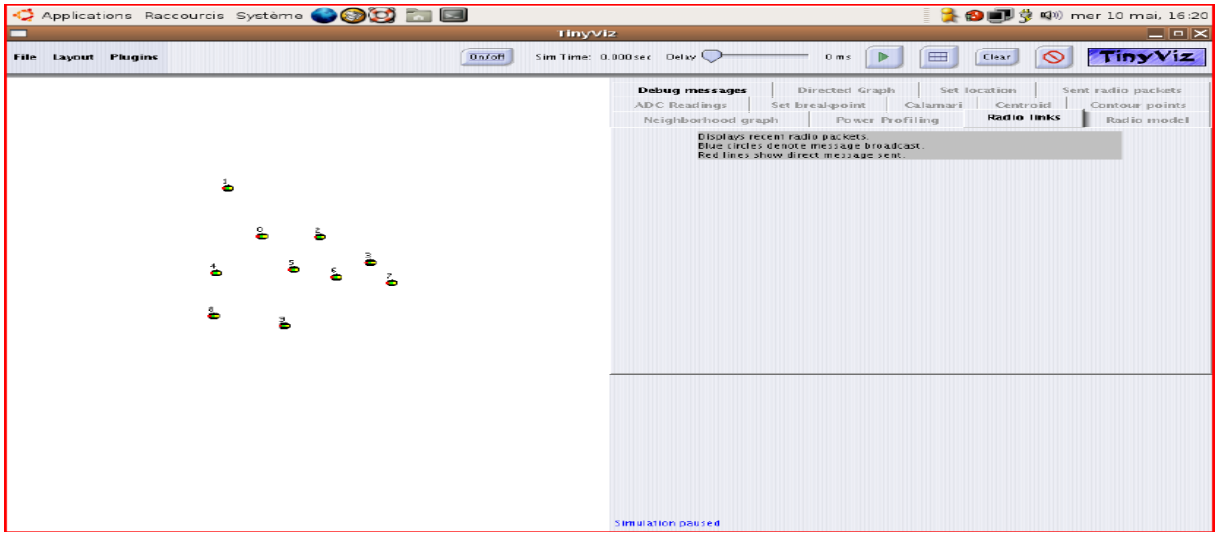


Figure 13 : fenêtre graphique TinyViz

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

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

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

Chaque onglet contient un plugin qui permet de visualiser la simulation de façon plus ou moins détaillée. Le plugin « Radio Links » permet de visualiser graphiquement par des flèches, les échanges effectués entre deux capteurs (unicast), ainsi que les messages émis par un capteur à l'ensemble du réseau (broadcast).

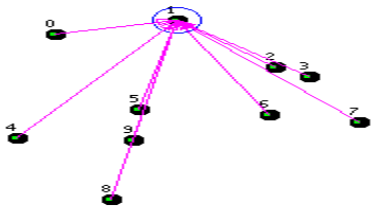


Figure 14 : Visualisation des messages radio

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

b. Avantages

Comme ATEMU, TOSSIM permet de simuler fidèlement le comportement d'un réseau de capteurs. Il permet également l'affichage des évènements et des messages de débogage pour chaque capteur mais aussi simultanément pour l'ensemble des capteurs. A cela se rajoute l'interface graphique TinyViz, qui permet la visualisation des échanges radios, conjointement aux messages de débogage, ce qui permet, à chaque instant de la simulation, d'avoir une vue globale de l'activité du réseau étudié. TinyViz offre la possibilité de ralentir la simulation par un délai, afin d'observer le déroulement des évènements, ce qui est très intéressant lorsque le réseau est surchargé de messages.

c. Limites

Malgré nos recherches, nous n'avons pas trouvé comment deux capteurs peuvent exécuter deux applications différentes. En effet, il serait plus intéressant de distinguer les capteurs maîtres des capteurs esclave, en leur demandant d'exécuter des codes différents. Pour pallier ce problème, nous avons eu recours à une astuce. Chaque capteur est

identifiable par un numéro unique (dans la plateforme), nous arrivons donc à différencier le rôle de chaque capteur par son numéro.

3. Contrainte matériels

Un capteur est composé de quatre éléments principaux :

- Un élément qui se charge de mesurer l'environnement extérieur (unité de capteur),
- Une unité de calcul,
- Un élément émetteur / récepteur,
- Une alimentation.

Trois composants additionnels peuvent être implantés dans un capteur :

- Un système de recherche d'emplacement,
- Un générateur d'alimentation,
- Une unité mobile (permettant de bouger le capteur).

L'unité de captures où l'élément capteur est composé de deux sous éléments:

- Le capteur récupérant des données analogiques
- un convertisseur passant de données analogique du capteur à des données numériques (appelée ADCs) envoyées à l unité de calcul.

Unité de calcul : Le composant regroupe :

- Un processeur,
- Une unité de mémoire réduite.

Il permet de gérer les données reçues par notre propre capteur, ou les données reçues par les autres capteurs du réseau.

Emetteur/Récepteur : L'élément permet de connecter le capteur au réseau.

Alimentation : Celle ci est souvent supplée par un récupérateur d'énergie (Exemple une cellule solaire).

Un capteur devra regrouper tous ses éléments dans une structure de moins de un centimètre cube. Les contraintes primordiales sont les suivantes :

- consommer le moins possible,
- fonctionner dans une aire de surface étendue,
- avoir un coût de production faible et dont on peut se passer,
- être autonome,
- être adapté à son environnement.

Dans l'absolu un capteur aura une structure comme celle présentée ci-dessous

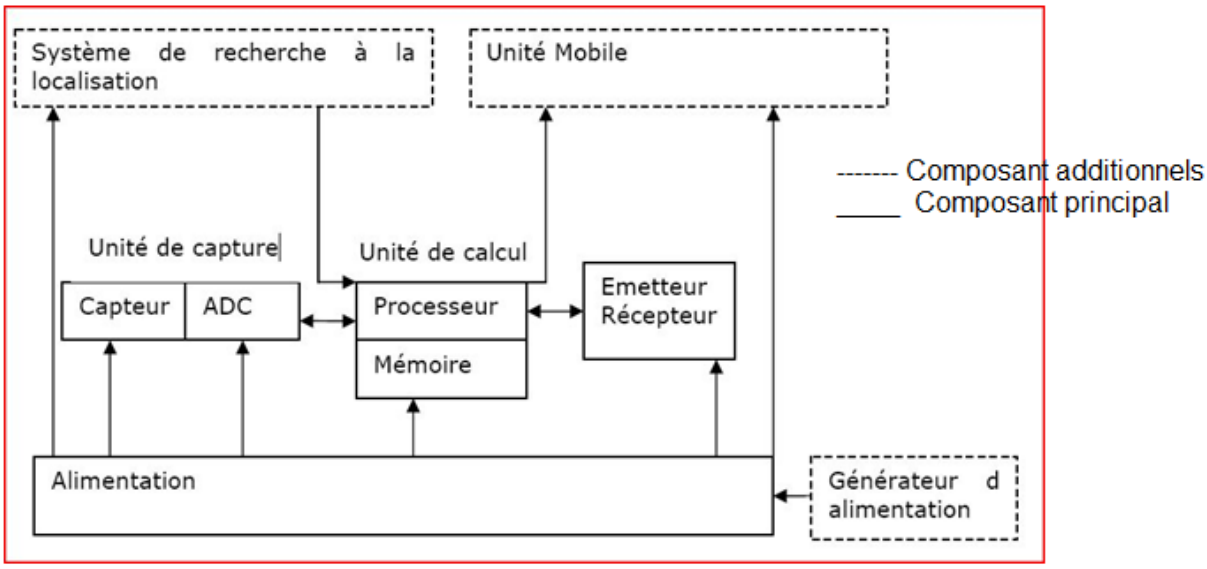


Figure 15 : Structure approché d'un capteur

Apres une étude de l'état de l'art des réseaux de capteur nous avons extraits un certain nombre de composants nécessaire à la bonne marche d'un capteur.

On peut séparer ces composants en deux catégories :

- Les composants nécessaires à la bonne marche d'un capteur.
- Les composants optionnels que nous avons du implémenter pour des raisons de fonctionnement.

Liste des composants nécessaires	La liste des composants facultatifs :
Capteur Batterie : Energie_total Emission/Réception	Energie d'émission Energie de réception Rayon d'émission Rayon de couverture Position …

Tableau 2 : Composant d'un capteur pour la simulation

4. Architecture de logiciel et modulation UML

Pour mettre en œuvre cette modularité, j'ai employé un panel de techniques différentes exposées ci dessous. Pour une découverte progressive de l'architecture associée aux technologies, plutôt que de présenter un seul bloc diagramme (UML), nous les avons découpés en fonction des notions abordées.

N'ayant pas énormément d'expérience dans la création d'architecture objet complexe, nous avons tenté de construire la structure de base (la plus importante des itérations). Les problèmes ont d'abord été posés sur papier puis modéliser. Etant incapable de représenter en une fois l'ensemble de l'architecture de base du logiciel, j'ai suivi la méthode de travail suivant.

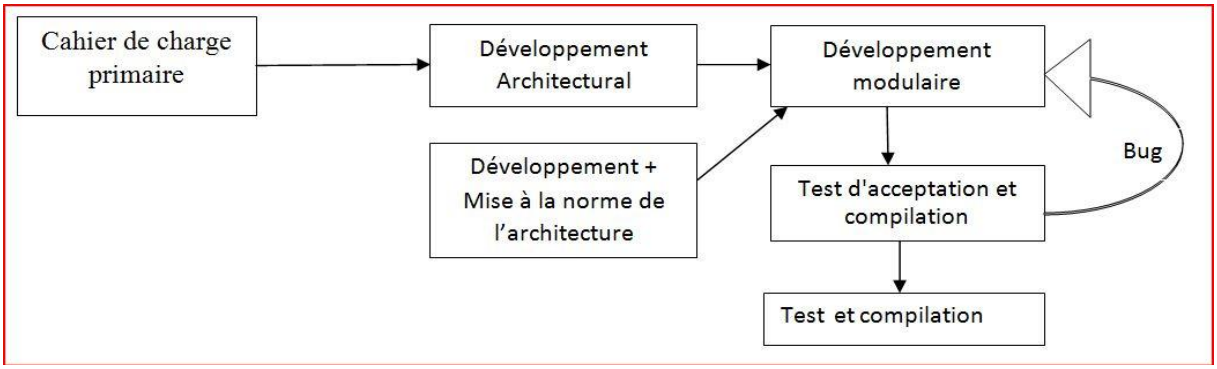


Figure 16 : Méthode de travail

5. Architecture centralisé de l'application

Pour mettre en œuvre l'architecture, j'ai choisi de définir 2 classes, une classe capteur qui représente les entités simple de notre approche centralisée, et une classe serveur « nœud principal » qui représente le nœud cardinale dans le réseau en se basant sur la notion de thread.

➢ Notion de Thread et principe de fonctionnement

Dans le contexte du développement d'applications modernes, il est très rare de ne pas avoir besoin de déclencher des activités en parallèle (Multithread). Cela s'oppose à la problématique du développement des programmes linéaires. Un thread, parfois appelé "processus léger", est en quelque sorte un processus à l'intérieur d'un processus. Les ressources allouées à un processus (temps processeur, mémoire) vont être partagées entre les threads qui le composent. Un processus possède au moins un thread (qui exécute le programme principal).

Contrairement aux processus, les threads partagent la même zone mémoire (espace d'adressage), ce qui rend très facile (et périlleux !) la communication entre les threads. Chaque thread possède son propre environnement d'exécution (valeurs des registres du processeur) ainsi qu'une pile (variables locales).L'utilisation des threads et le détail de leur comportement est différente dans chaque système d'exploitation (Windows NT, UNIX). Certains langages, comme JAVA, définissent leur propre mécanisme de threads, afin de permettre l'utilisation facile et portable des threads sur tous les systèmes.

Le serveur aura plusieurs tâches à exécuter en parallèle : il devra être à l'écoute des clients qui souhaitent se connecter, et il devra aussi s'occuper des clients déjà connectés. Je choisis d'utiliser un contexte multithreads. Nous pourrions aussi utiliser un contexte multiprocessus mais l'utilisation des threads est suffisante, de plus, ils sont moins lourds à gérer pour le système d'exploitation.

Notre serveur fonctionnera de la manière suivante : le processus principal sera en charge d'écouter les connexions entrantes et pour chacune d'elles, il va créer un thread qui sera dédié à cette connexion. J'ai choisi de créer une classe représentant notre serveur (nœud principal). Cette classe aura les méthodes relatives à l'initialisation et au démarrage de notre serveur.

Les threads de LINUX sont une implementation normalisée des MIT-threads. Ces threads sont gérés à la fois par le système, et par une librairie au niveau utilisateur. Pour créer des threads, LINUX applique la stratégie de l'One-For-One (une entrée dans la table des processus pour chaque thread).

pthread create(thread, attribut, routine, argument) Creation d'un thread. Le nouveau flot d'exécution démarre en se branchant à la routine spécifiée.

pthread exit(r´esultat) Suicide d'un thread.

pthread join(thread, résultat) Attendre la terminaison d'un autre thread.

pthread kill(thread, nu du signal) Envoyer un signal (UNIX) `a un thread. C'est un moyen dur pour tuer un thread. Il existe des méthodes plus conviviales.

- **Modélisation de l'architecture**

Le figure suivant met en évidence l'architecture déployé pour représenté l'architecture de notre application. Il ya un échange de messages entre les capteurs et le nœud central.

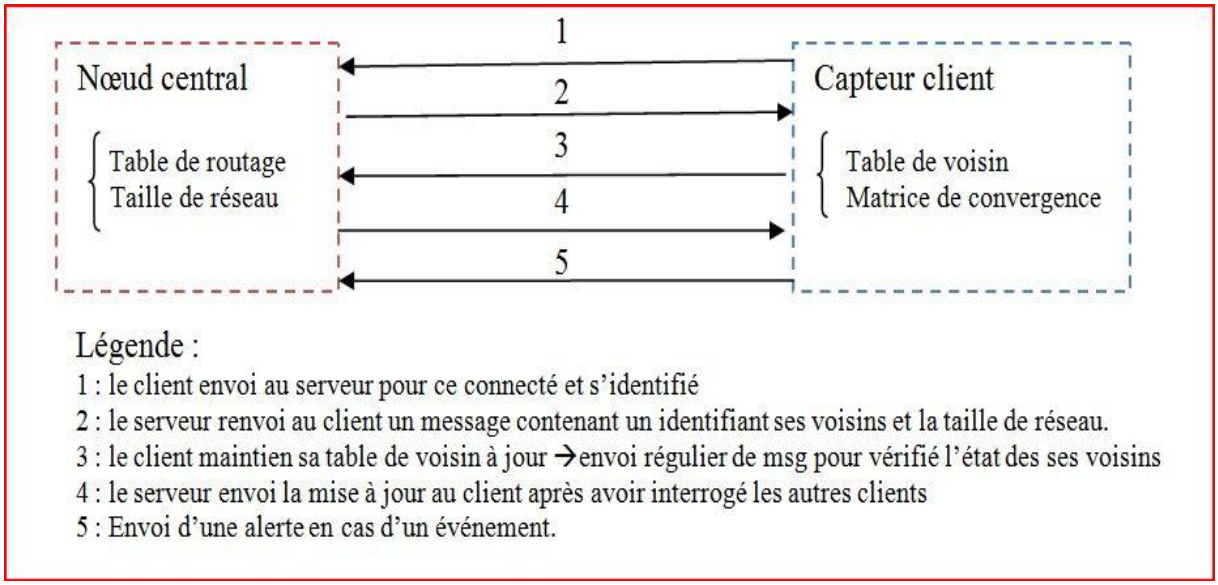


Figure 17 : Architecture de l'application

- **Description de l'entité Nœud principal**

Le programme « Nœud principal ou serveur» est le processus qui permet de gérer tous les réseaux. Il permet de relier tous les capteurs entre eux, à travers un échange de message et de vérifier l'état (connecté/déconnecté) de chaque capteur. Le capteur central ou serveur, a pour rôle d'identifier chaque capteur, et de l'informer de ses voisins à l'aide de la matrice de graphe déjà définie.

Ce processus utilise les fonctions suivantes pour bien administrer ses clients « capteurs »

Exemple :

Le serveur multitâche implémenté dans notre logiciel, vérifie l'état des capteurs toute les x secondes. Si un événement ce produit, le capteur sera capable de transmettre cette information à travers le nœud principal. Comme tous les autres composants, celui ci absorbe une certaine quantité d'énergie à son démarrage et à chaque action effectuée.

Fonction	Description
void *listen(void *prt)	Cette fonction permet de rester à l'écoute des messages envoyés par les capteurs, des événements produits, et de les analyser.
void *traitement(void *prt)	Permet de spécifié le type de message
int conecter()	Cette fonction permet d'attribué un identifiant unique, la liste de voisins et la taille de réseaux à chaque capteur connecté puis les envoyés
int ping ()	Permet de vérifier l'état des voisins et le renvoie vers les capteurs connectés avec un message de type 2 →Mise à jour effectuée
int decouvete_reseaux()	Cette fonction permet d'obtenir les voisins actuellement connectés.
int message()	Cette fonction permet d'envoyer un message l'or du déclanchement d'un événement en calculant le plus cours chemin.

Tableau 3 : description des fonctions de noeud principal

A travers ces fonctions définie, on a put implémenter un serveur qui nous avons jugé nécessaire pour le fonctionnement de la simulation. La déclaration d'un graphe pour des clients « capteurs » et nécessaire pour pouvoir renvoyer les voisins de chaque capteur qui ce connecte.

Le processus « serveur » et considéré comme le nœud qui administre tous les réseaux des capteurs, elle permet : de créer un réseau de capteurs, identifier chaque capteur et permettre Emission/Réception de l'information… .

- **Description de l'entité capteur « client »**

Ce processus capteur représente une simulation pour un capteur, ce dernier va être capable de s'identifier au près du nœud principal, de savoir les capteurs voisins qui appartiennent a son diamètre d'émission et la taille de réseau, en interrogeant le serveur qui a les informations nécessaires sur le réseau. On sera capable de lance plusieurs capteurs en exécution, grâce à la notion de thread.

L'or du déclanchement d'un événement à un emplacement bien précis, le capteur qui couvre l'espace va émettre un message à la station (borne de transmission).

Méthode	Description
void *listen(void *prt)	fonction « listen » ce met a l'écoute des messages échangé, analyse le type de message et exécute les instructions adéquat.
void *ping(void *prt)	envoi de message « ping » avec numéro 2 et son identifiant pour recevoir la mise à jour de ses voisins.
int voisinage()	Cette fonction permet d'obtenir les voisins de capteur actuellement connecté à partir de serveur.
void *message(void *prt)	La fonction capable d'envoyé des messages à la station principale (borne de transmission) en calculant le plus court chemin (Dijkstra).
int message()	Cette fonction permet de transféré un message envoyé ver la station (exécuté sur tous les capteurs de plus courts chemins)

Tableau 4: description de l'entité capteur

Dans le fichier client (ou capteur) et pour le bon fonctionnement, chaque client va se lancer sur un terminal pour effectuer une liaison d'identification, avec le serveur qui va lui attribuer un identifiant et lui renvoi ses voisins. Apres être identifié le capteur se met a l'écoute des événements et des messages a échanger.

Cette classe entité capteur représente bien un réseau des capteurs sans fil car, elle permet de lancer plusieurs capteur « client ».

-------> La communication capteurs/Nœud principal ce fait a travers un échange de message bien déterminé. Cet échange et mis en place entre processus à travers des sockets, la communication par socket utilise un descripteur pour désigner la connexion sur laquelle on envoie ou on reçoit les données. Ainsi la première opération à effectuer, consiste à appeler une fonction créant un socket et retournant un descripteur (un entier), identifiant de manière unique la connexion. Ainsi ce descripteur est passé en paramètres des fonctions, permettant d'envoyer ou recevoir des informations à travers le socket.

➢ Echange des messages

La communication entre les capteurs et le nœud principal, se fait à travers des messages qui sont spécifiés par un numéro.

- **Coté serveur « Nœud principal »**

1. Message de numéro « 1 » : Permet d'établir une connexion entre le capteur te le serveur à travers la fonction connecter.

```
if(nb == 1 )
 {conecter();
```

Type Msg 1	monIdf:2	Voisin 1	Voisin 2	……	Voisin n

2. Message de numéro « 2 » : l'hors de la réception d'un message de numéro 2 on utilise la fonction « ping » .Cette fonction permet de tester la connectivité des clients pour faire la mise à jour de la table du routage et renvoi un message de type 3 au client.

```
if(nb == 2 )
 {ping();
```

3. Message de numéro « 3 » : L'hors de la réception d'un message de numéro 3 on fait appel à la fonction découverte_reseaux() qui permet d'obtenir les voisins actuellement connecter.

```
if(nb == 3 )
 {decouverte_reseaux();}
```

4. Message de numéro « 4 » : l'hors de la réception d'un message de numéro 4, cela signifie l'envoi d'un message due au déclanchement d'un événement en utilisant la fonction message().

```
if(nb == 4 )
 {message();}
```

Tous ces message sont envoyés à travers des socket
Exemple :

```
struct sockaddr_in serveur;
struct sockaddr_in locale; // declaration de socket
sendto(sock,buf,30,0,(struct sockaddr*)&adresse_client,sizeof(adresse_client));//envoi de message
(recvfrom(sock,cas,30,0,(struct sockaddr*)&sin,&lg_adresse)==-1); // reception dun message
```

- **Coté capteur « client »**

1. Message de numéro « 1 » : l'envoi de message de numéro 1 a pour but de demander - Demander le numéro.
   - Demander la taille de réseaux.
   - Demander les voisins.

```
sendto(sock,"1",30,0,(struct sockaddr*)&sin,sizeof(sin));  // message qui porte le numero 1
```

2. Message de numéro « 2 » : L'hors de la réception d'un message de numéro 2 envoyé par le capteur vers le nœud principal de réseau, cela ce traduit par une demande des voisins connectés.

```
sendto(sock,"2",30,0,(struct sockaddr*)&sin,sizeof(sin)); // message de numéro 2
```

3. Message de numéro « 3 » : Ce message permet de renvoyer les voisins d'un capteur qui appartient à la même surface, vers le serveur en utilisant la fonction voisinage().

```
sendto(sock,"3",30,0,(struct sockaddr*)&sin,sizeof(sin));// message de numéro 3
```

4. Message de numéro « 4 » : la socket portant le numéro 3 signifie l'envoi d'un message vers la stations d'émission. Cela se fait après avoir calculer le plus court chemin à travers l'algorithme de Dijkstra

```
sendto(sock,"4",30,0,(struct sockaddr*)&sin,sizeof(sin));// message de numéro 4
```

L'échange client/serveur ce traduit par le schémas suivant.

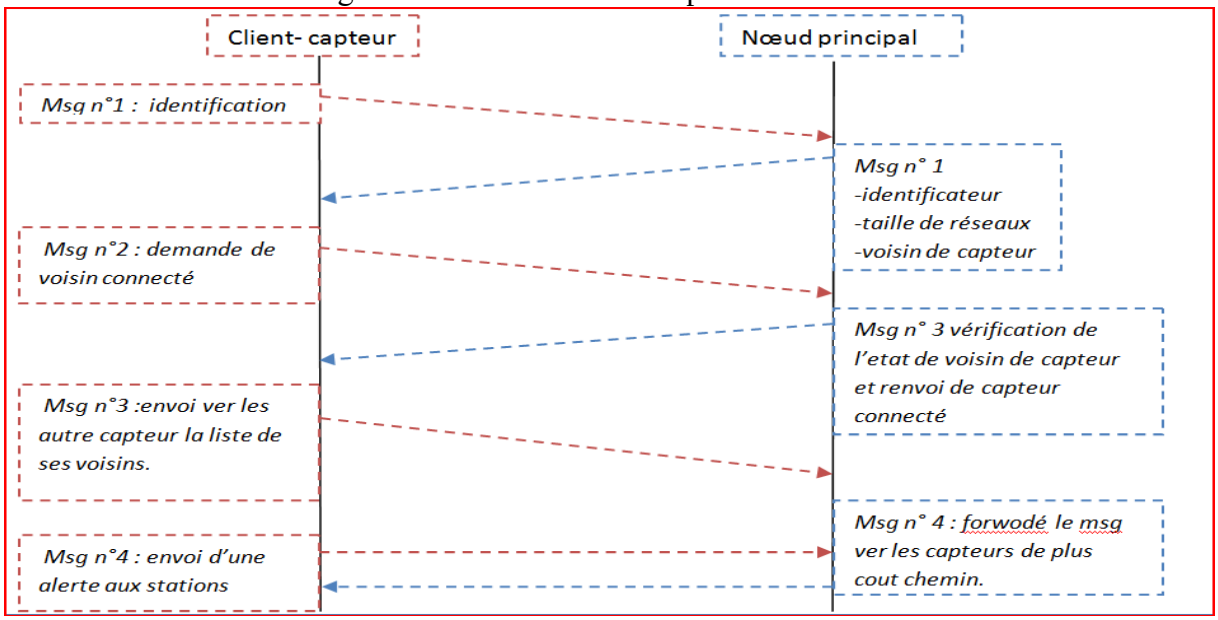


Figure 18 : échange de message entre capteur et nœud principal

XV. Conclusion et perspective

Au cours de ce projet « Développement d'une bibliothèque de capteurs » il nous a été demandé de mettre en place une plateforme de test et d'expérimentation pour les réseaux de capteurs. La recherche sur les réseaux de capteurs est actuellement en pleine essor. Les réseaux de capteurs sans fil sont une technologie récente. Les progrès de miniaturisation et d'allongement de la durée de vie des batteries, annoncent un futur prometteur à cette technologie. De plus, le développement de nouveaux capteurs permettra d'étendre les domaines d'applications déjà nombreux. Il a été pour moi un grand plaisir de s'intégrer à la recherche dans ce domaine. Travailler sur une activité possédant des enjeux aussi enthousiasmants, ma donné la motivation pour mener à terme ce projet. La mise en place de cette plateforme de test est une tache difficile, elle nécessite un choix d'une architecture à adopter. Dans mon projet j'ai du avoir recours à une architecture centralisé. La contrainte de l'énergie et l'une des tache critique dans le projet car l'un des but de ce projet et de maximiser la duré de vie des capteurs sans fils toute en gardant nos réseaux opérationnels. Les réseaux de capteurs peuvent être mobiles. Par exemple, on peut imaginer un réseau de capteurs pour surveiller le déplacement d'animaux. Il faut donc s'attendre à ce que la topologie du réseau soit en permanente modification. Cela signifie qu'il faut trouver des protocoles de routage adaptés (efficaces dans le cas de la mobilité, tout en étant "économiques"). Mon application ne supporte pas ce critères, est cela peut être repris par d'autre groupe qui vont continuer la recherche dans ce domaine.

Compte tenu de l'étendue du sujet à traiter, le logiciel de simulation que j'ai réalisé cette année ne sera probablement qu'un prototype parmi d'autres. Il est donc important pour moi de présenter un travail pouvant être transmis et repris facilement par un autre groupe de projet, l'année prochaine par exemple. Cette modularité ne s'applique pas qu'au programmeur suivant, mais également à l'utilisateur qui doit pouvoir faire évoluer son application comme défini par le cahier des charges. La modularité dans un contexte client doit inclure :

- La possibilité de programmer des composants sur un modèle générique bien défini.
- La possibilité de faire évoluer nos composants de capteur.
- La possibilité de faire évoluer l'application.

Une amélioration de déroulement de l'application peut être effectuée en diminuant le nombre de message échangé, pour économiser en énergie. Pouvoir mètre en place un processus de générations de panne, pour savoir si notre déploiement de capteur peut rester toujours opérationnelle, et imaginer une interface graphique ou l'utilisateur peut effectuer son choix d'architecture et spécifier d'avantage le type de capteur qu'il parfaire testé.

Bibliographie

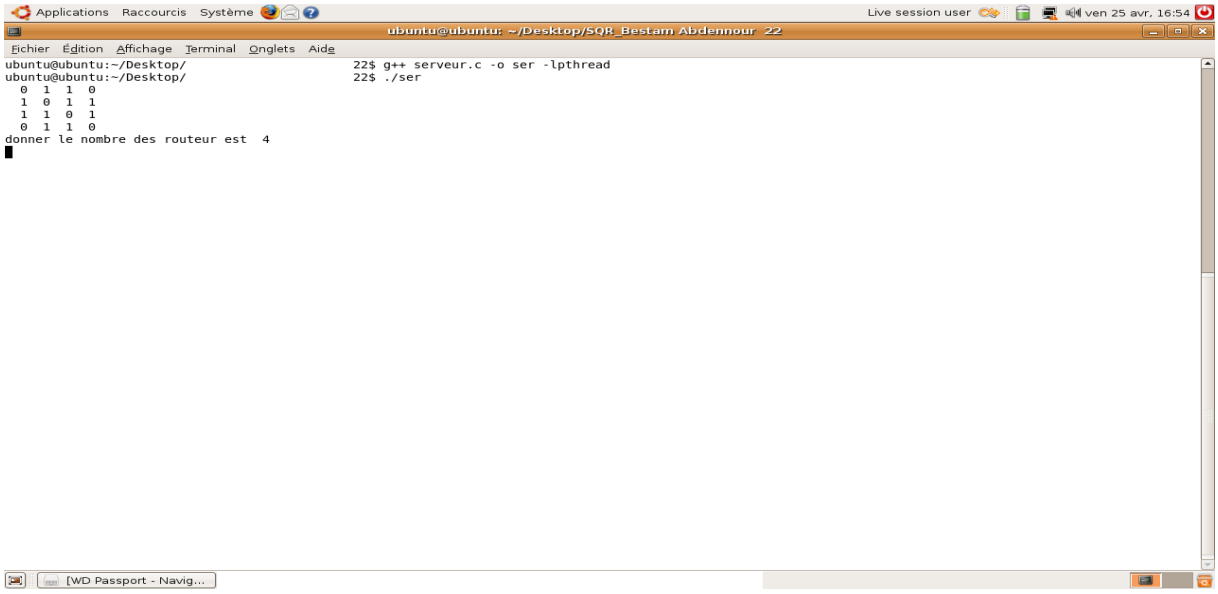
http://www.techno-science.net/?onglet=glossaire&definition=11711
www.iict.ch/actu/publications/module3435/download?id=3436
http://www.tinyos.net/
http://wwwmaster.ufrinfop6.jussieu.fr/2005/IMG/pdf/presentationZigBeeTinyOS.pdf
http://membres.lycos.fr/faucheur/reseauxdecapteurs/index.htm
http://www.hynet.umd.edu/research/atemu/
http://liuppa.univ-pau.fr/
http://www.supinfo-projects.com/cn/
http://ospf.cs.st-andrews.ac.uk/index.shtml?index&040000
http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson5.html
http://fr.wikipedia.org/wiki/Algorithme_de_Dijkstra
http://fr.wikipedia.org/wiki/Processus_l%C3%A9ger
http://fr.wikipedia.org/wiki/R%C3%A9seau_de_capteurs_sans-fil
http://www.aromath.net/Page.php?IDP=665&IDD=0
http://www-b2.is.tokushima-u.ac.jp/%7Eikeda/suuri/dijkstra/Dijkstra.shtml
http://www-b2.is.tokushima-u.ac.jp/%7Eikeda/suuri/maxflow/Maxflow.shtml
http://www.tinyos.net/windows-1_1_0.html
http://www2.lifl.fr/~carle/dnld/dea/mr2_im4_carle_6ppp.pdf
http://www.julienvaudour.fr/fichiers/memo.pdf

Table de figure :

XVI. Annexe

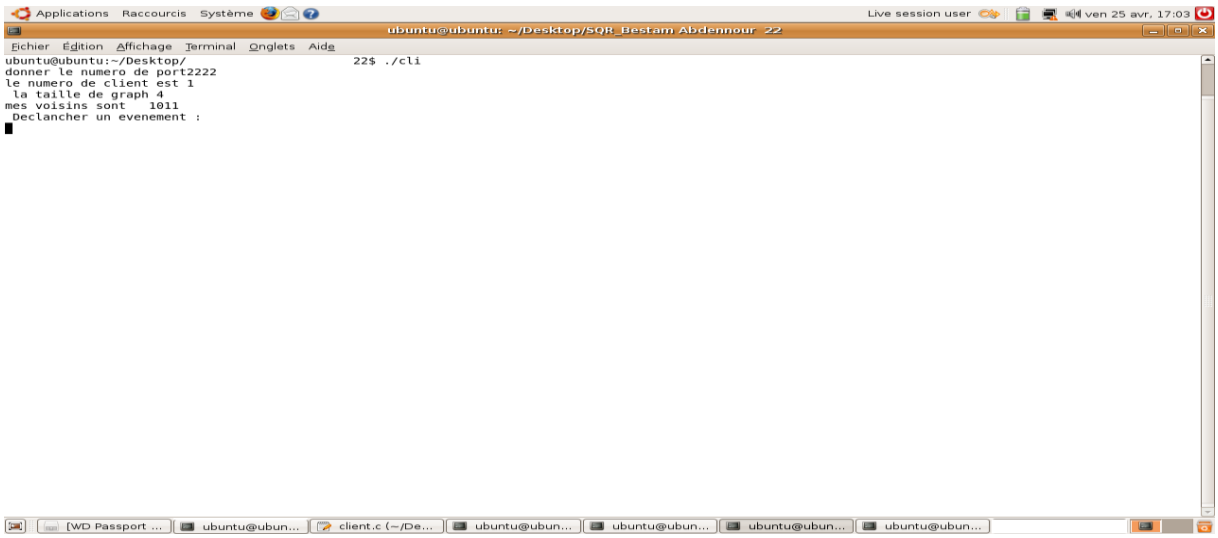
1. Manuel d'utilisation

Pour exécuté le programme il faut ouvrir un terminal ou on lance le serveur « nœud principal » g++ serveur.c –o ser –lpthread



1.1 Terminal de lancement de serveur

Après avoir lance le serveur, la matrice de grahpe de reseaux de capteur et affiché avec le nombre de client a lancé, le nœud principal va se mètre à l'écoute.
on doit lancer les clients (capteur).
g++ client.c –o cli –lpthread

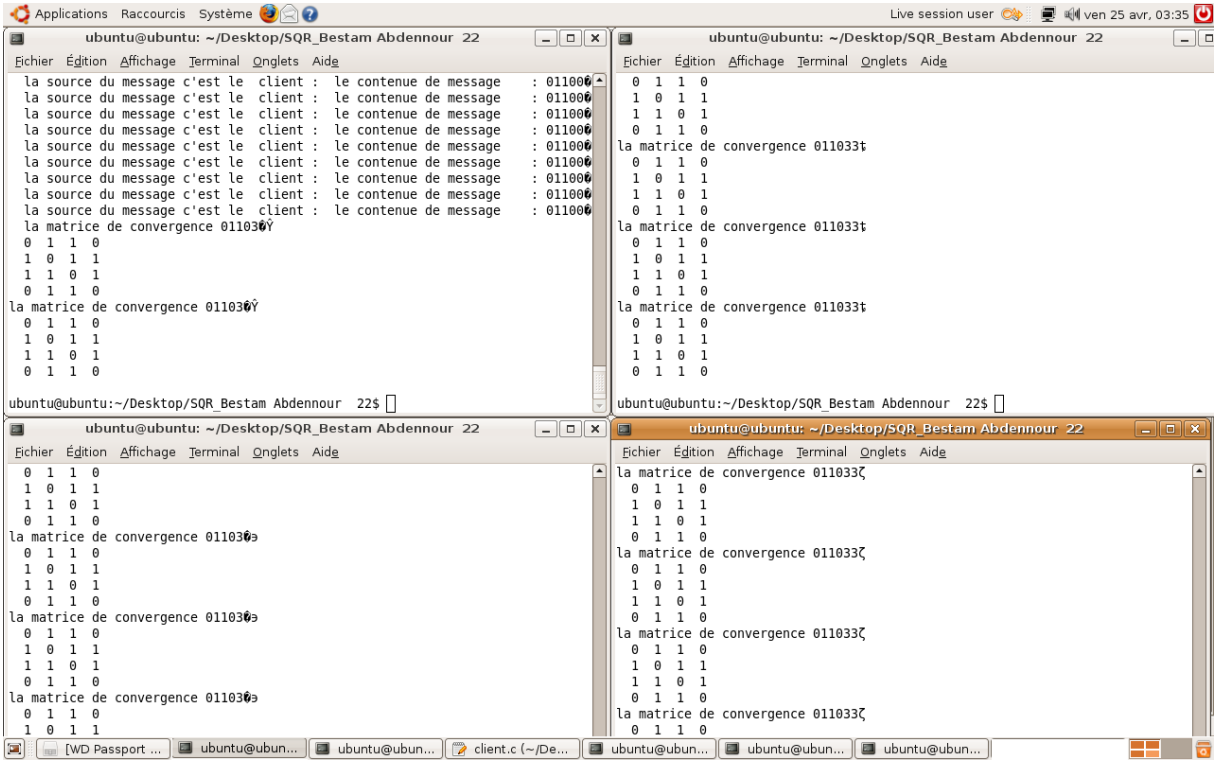


1.2 Figure de lancement d'un client

L'hors de lancement, chaque client 1 a un identifiant qui va interroger le serveur qui lui renvoi
- un identifiant
- ses voisins
- la taille de réseaux

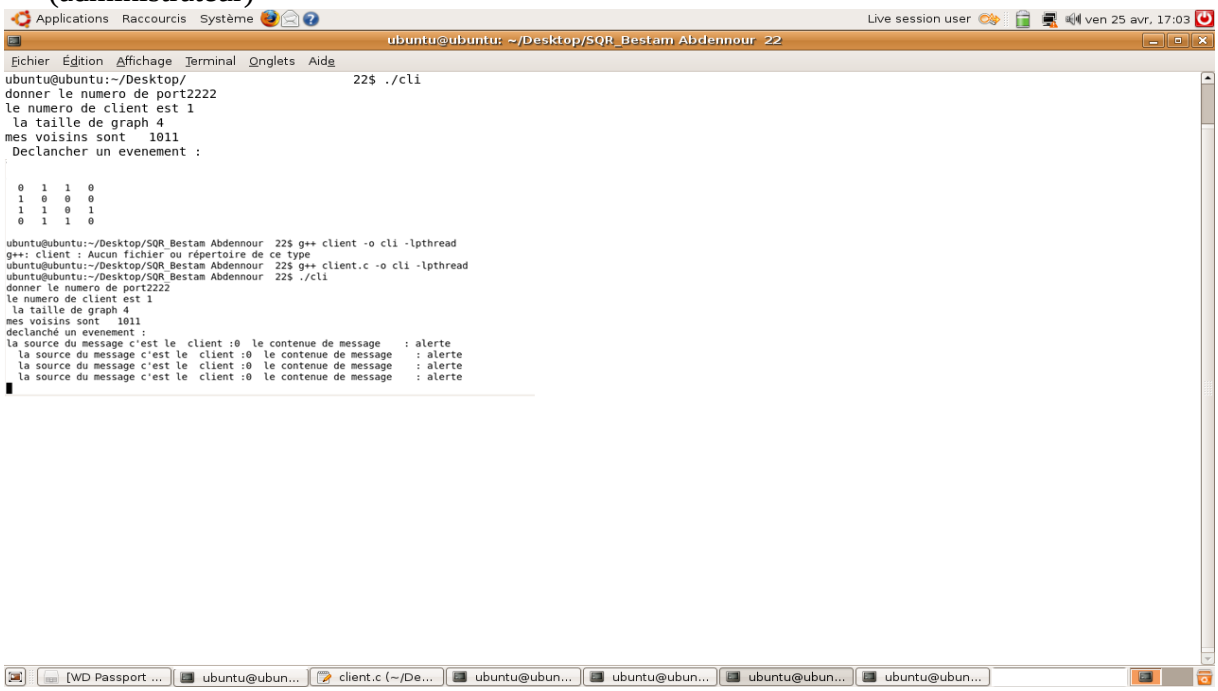
Et reste a déclenché un événement

Apres avoir lancé tous le capteur qui vont s'identifié au prés de serveur le figure suivant nous montre la convergence de la mise a jour de table de routage de chaque client.

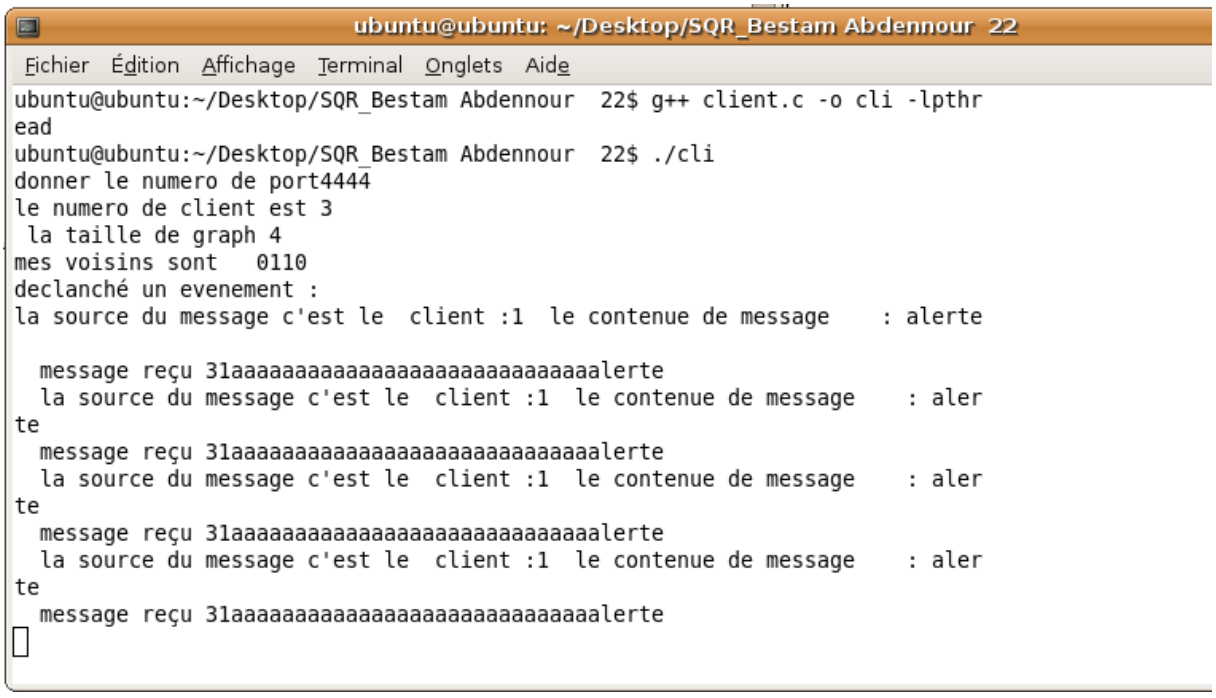


1.3 Figure qui montre la matrice de convergence de client

Le déclanchement d'un événement ce fait par un capteur qui va émettre vers la station de reception, ce ou le message va être reçu et acheminé vers la destination final (administrateur)



1.4 Figure de transmission d'alerte âpre calcule de plus court chemin



Ici ce une alerte envoyé par le capteur d'identifiant 0 à la station central, l'acheminement d'alerte ce fait a travers le capteur n° 1 et transféré ver la station (capteur de n°4).

- **Compte rendu réunion 1**

Date	**24 Janvier 2008**
Lieu	Université de Montpellier 2
Présences	Anne-Elisabeth Baert Vincent Boudet
	FARES Abdelfatah DJOUDI Mohamed Lakhdar MBOUKEM Tchassem Jules

Les sujets abordés :

Présentation approfondie du sujet de projet « Développement d'une bibliothèque de capteurs ».

Les points développés :

- ✓ Présentation de l'état de l'art des réseaux de capteurs sans fils pour comprendre rapidement les enjeux et les contraintes du système.
- ✓ Discussion sur la chronologie de projet et les phases critique.
- ✓ Présenter pour la prochaine réunion une étude des réseaux de capteurs ainsi qu'un premier cahier des charges.

- **Compte rendu réunion 2**

Date	**05 Février 2008**
Lieu	Université de Montpellier 2
Présences	Anne-Elisabeth Baert
	FARES Abdelfatah DJOUDI Mohamed Lakhdar

Les sujets abordés :

Cahier de charge

Les points développés :

- ✓ Discussion et correction de cahier de charge remise au par avons par email
- ✓ Modification de diagramme de Gantt.
- ✓ Remise de cahier de charge par email pour une dernier vermification.

Le cahier des charges prévoyait :

- reproduire le comportement et le fonctionnement d'un capteur sans fil dans un environnement informatique.
- prévoir un environnement modulaire permettant de choisir le type de composant à intégrer à la simulation afin de rendre cette simulation fiable.
- On sera capable de prévoir le bon fonctionnement du réseau, ses performances, son organisation, sa consommation d'énergie.

- **Compte rendu réunion 3**

Date	**06 Mars 2008**
Lieu	Université de Montpellier 2
Présences	Anne-Elisabeth Baert
	FARES Abdelfatah DJOUDI Mohamed Lakhdar

Les sujets abordés :

Présentation de premier chapitre «Présentations des réseaux de capteur sans fils ».

Les points développés :

- ✓ Présentations de réseaux de capteur sans fils
  - o Architecture de capteur sans fils.
  - o Système d'exploitation TinyOS.
  - o Ordonnancement et modalisation de réseaux de capteur sans fils.
- ✓ Algorithme de routage « choix d'algorithme de routage ».
- ✓ Remise d'un rapport écrit sur le chapitre 1.

- **Compte rendu réunion 4**

Date	**03 Avril 2008**
Lieu	Université de Montpellier 2
Présences	Anne-Elisabeth Baert
	FARES Abdelfatah

Les sujets abordés :
Implémentation de logiciel de simulation de réseaux de capteur sans fils
Les points développés :

- ✓ Choix de langage de programmation java
- ✓ Modalisation UML et architecture de logiciel
- ✓ Implémentations de classe nécessaires au bon fonctionnement de la simulation.
- ✓ Rédaction de rapport.