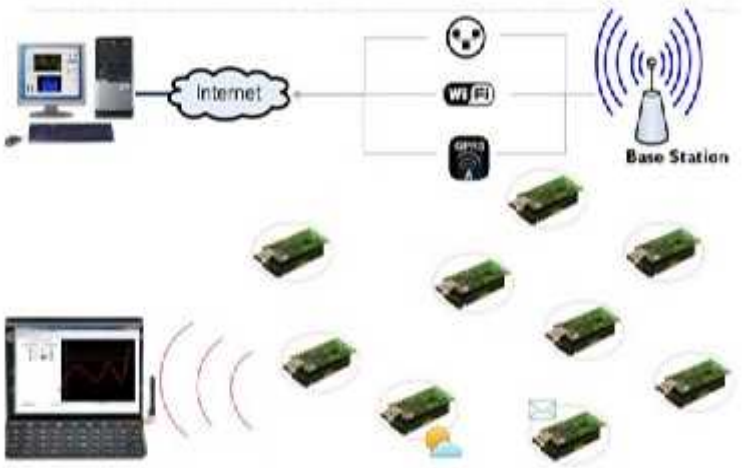
PARIS 8
UNIVERSITE
VINCENNES - SAINT-DENIS

Projet Tuteuré

Implémentation d'un protocole de routage hiérarchique sous TinyOS



Encadré par **:** Mr.Touati Youcef

Réalisé par : Maknassi Asmaa
Jaiti Sana

Année universitaire 2015-2016

Sommaire

1 Introduction aux Réseaux de capteurs sans fils.....5

1.1 Qu'est-ce qu'un réseau de capteur sans fil.....5
1.2 Architecture des réseaux de capteurs sans fil.....5
1.3 Domaines d'application des RCSF.....6
1.3.1 Domaine médical 6
1.3.2 Domaine militaire 6
1.3.3 Domaine architectural 6
1.3.4 Domaine environnemental.....6
1.4 Conclusion.....6

2 Les protocoles de routage dans les réseaux de capteurs sans fils 7

2.1 Définition des protocoles de routage dans les réseaux de capteurs 7
2.2 Contraintes de conception des protocoles de routage pour RCSF.....7
2.3 Les différents protocoles de routage dans les réseaux de capteurs.....7
2.3.1 LEACH.....7
2.3.2 PEGASIS.....8
2.3.3 TEEN.....9
2.3.4 APTEEN.....9
2.3.5 HEED.....9
2.4 Conclusion 10

3 Présentation de l'environnement de travail.....11

3.1 Le système d'exploitation TinyOS.....11
3.2 Le langage de programmation de TinyOS : NesC.....11
3.3 Interface.....12
3.4 Modèle de concurrence.....12
3.4.1 Les tâches.....12
3.4.2 Les handlers d'événements matériels 12
3.5 Processus de compilation.....13
3.5.1 NESCDT : Un éditeur pour NesC dans Eclipse 13
3.5.2 Débogage du programme.....13
3.6 Les différents outils de simulation 14
3.6.1 Tossim.....14
3.6.2 PowerTossim.....14
3.6.3 TinyViz.....14
3.7 Conclusion 15

4 Implémentation du protocole TEEN.....16

4.1 Organigramme du protocole TEEN.....17
4.2 Résultats obtenus de la compilation…….....18
4.2.1 LEACH….....19
4.2.2 TEEN…….....20
4.2.3 Comparaison entre LEACH et TEEN…….....21
4.3 Simulation avec TinyViz.....21
4.4 Conclusion…….....23
4.5 Problèmes rencontrés…….....23

Conclusion Générale.....24

Références.....25

Annexe 1 & Annexe 2

Remerciement

Nous tenons à remercier notre encadrant Mr TOUATI Youcef pour toute l'aide apportée tout au long de ce travail, son aide permanente et ses conseils éclairés, nous a permis de mener à bien ces recherches. Nous tenons également à remercier l'université Paris 8 de nous avoir accordé les salles pour pouvoir faire notre projet.

On remercie aussi le membre de jury qui fera l'honneur d'examiner et de juger notre travail.

Le but de notre projet, Problématique et Solution

- **Le but du projet :** Il consiste à étudier les protocoles de routage dans les réseaux de capteurs sans fil (RCSF) et d'implémenter le protocole TEEN (Threshold-sensitive Energy Efficient sensor Network Protocol).

- **Problématique :** L'énergie est un grand problème dans la conception d'un RCSF, les applications exigent que la durée de vie du réseau soit de l'ordre de plusieurs mois ou même des années et les nœuds capteurs restent en permanence en activité, leurs batteries peuvent fonctionner juste pour quelques jours ou pour quelques semaines.

- **Solution :** Implémenter un protocole de routage qui a pour objectif la minimisation de la consommation d'énergie des nœuds capteurs et étendre la durée de vie du réseau.

Introduction

Dans la vie courante, l'utilisation des capteurs sans fils est de plus en plus demandée pour la supervision et la sécurité. Les industries proposent alors des capteurs sans fils qui peuvent renseigner l'utilisateur sur plusieurs données. Ces capteurs peuvent aussi être reliés ensemble pour former un réseau sans fil se basant sur des protocoles pour se communiquer et proposant des programmes et des réseaux embarqués. Les capteurs fonctionnent donc à basse tension et ceci est géré par un système d'exploitation spécialisé : TinyOS.

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, unités de traitement...) sur une distance limitée à quelques mètres. Les RCSF (Réseaux de capteurs sans fils) sont souvent considérés comme étant les successeurs des réseaux ad hoc (réseaux sans fil capables de s'organiser sans infrastructure définie préalablement). En effet, les RCSF partagent avec lesMANET (Mobile Ad hoc Networks) plusieurs propriétés telles que l'absence d'infrastructure et les communications sans fils. Mais l'une des différences clés entre les deux architectures est le domaine d'application. Contrairement aux réseaux MANET,qui n'ont pas pu connaître un vrai succès, les RCSF ont su attirer un nombre croissant d'industriels, vu leur réalisme et leur apport concret. Notre rapport est composé de quatre chapitres :

- Dans le premier chapitre nous présentons des Généralités.
- Le deuxième chapitre présente les différents protocoles de routage hiérarchique des réseaux de capteurs sans fils.
- Le troisième chapitre décrit l'environnement de travail : TinyOS, NesC, NESCDT et les différents outils de simulation.
- Le quatrième chapitre présente l'implémentation du protocole TEEN et les résultats obtenus.

1. Introduction aux Réseaux de capteurs sans fils

1.1 Qu'est-ce qu'un réseau de capteur sans fil ?

Les réseaux de capteurs sans fil (RCSF) sont un type particulier de réseau Ad-hoc (réseaux sans fil capables de s'organiser sans infrastructure définie préalablement), dans lesquels les nœuds sont des "capteurs intelligents". Ils se composent généralement d'un grand nombre de capteurs communicantsentre eux via des liens radio et capables de récolter et de transmettre des données environnementales d'une manière autonome. Dans ce type de réseau, les capteurs échangent des informations par exemple sur l'environnement pour construire une vue globale de la région contrôlée, qui est rendue accessible à l'utilisateur externe par un ou plusieurs nœuds.

1.2 Architecture des réseaux de capteurssans fils

Un RCSF est composé d'un ensemble de nœuds capteurs. Ces nœuds capteurs sont habituellement dispersés dans une zone de capture organisé en champs « sensorfields ». Chacun de ces nœuds à la capacité de collecter des données et de les transférer au nœud passerelle (puits) par l'intermédiaire d'une architecture multi-sauts. Le puits transmet ensuite ces données par Internet ou par satellite à l'ordinateur central "Gestionnaire de tâches " (Voir1.1).

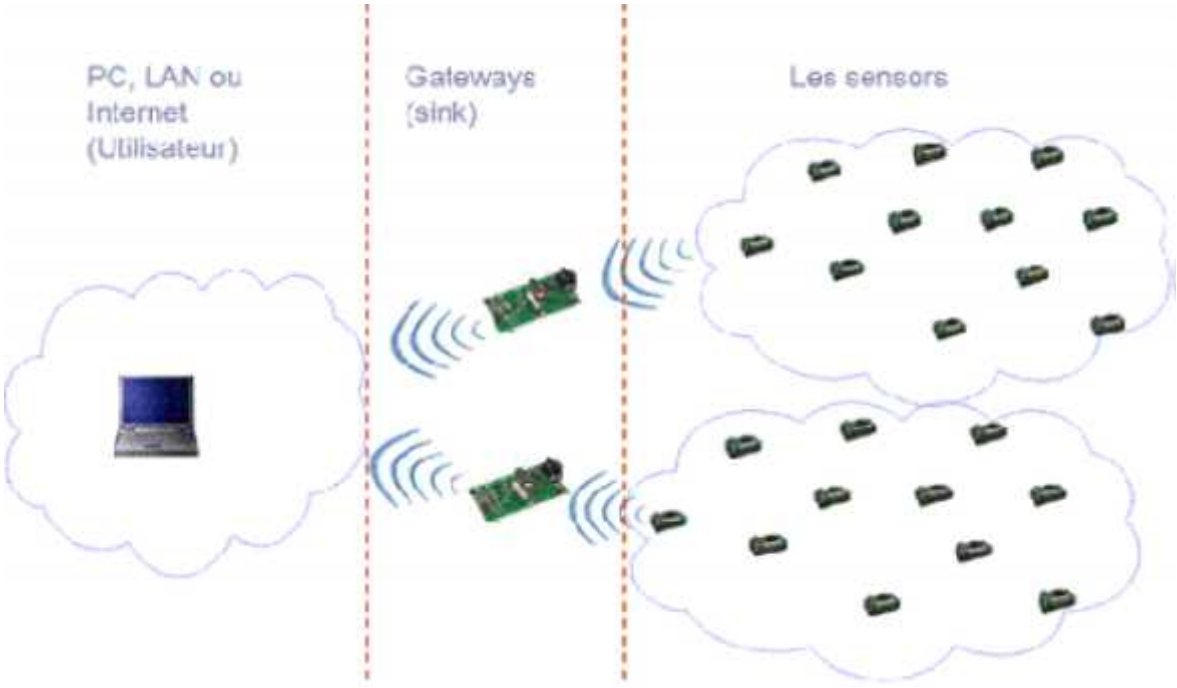


Figure 1.1 Architecture des réseaux de capteurs

1.3 Domaines d'application des RCSF

1.3.1 Domaine médical

Les réseaux de capteurs sont largement répandus dans le domaine médical. Cette classe inclut des applications comme : fournir une interface d'aide pour les handicapés, collecter des informations physiologiques humaines de meilleure qualité, facilitant ainsi le diagnostic de certaines maladies grâce à des micro-capteurs qui pourront être ingérés ou implantés sous la peau, surveiller en permanenceles malades et les médecins à l'intérieur de l'hôpital.

1.3.2 Domaine militaire

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

1.3.3 Domaine architectural

La détection des altérations dans la structure d'un bâtiment, suite à un séisme ou au vieillissement.

1.3.4 Domaine environnemental

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

1.4 Conclusion

Dans ce chapitre on a commencé par une présentation des généralités sur les protocoles de routages dans les réseaux de capteurs sans fils et on a décrit aussi le fonctionnement, l'architecture et les domaines d'application des réseaux de capteurs sans fils. Dans le prochain chapitre nous allons définir les différents protocoles de routage hiérarchique utilisés avec une étude comparative entre quelques protocoles de routage dans les réseaux de capteurs sans fils.

2. Les protocoles de routage dans les réseaux de capteurs

2.1 Définition des protocoles de routage dans les RCSF

Le routage est une méthode d'acheminement des informations vers une destination donnée dans un réseau de connexion. Il est fondamental dans les réseaux de capteurs sans fils, vu qu'il n'existe pas d'infrastructure qui gère les informations échangées entre les différents nœuds du réseau. En effet, c'est à chaque nœud du réseau de jouer le rôle d'un routeur. Comme nous l'avons déjà vu, l'architecture des réseaux Ad-hoc est caractérisée par l'absence d'infrastructure fixe préexistante, à l'inverse des réseaux de télécommunication classiques. Un réseau Ad-hoc doit s'organiser automatiquement de façon à être déployé rapidement et pouvoir s'adapter aux conditions du trafic et aux différents mouvements pouvant intervenir au sein des nœuds mobiles.

2.2 Contraintes de conception des protocoles de routage pour RCSF

- Offrir un support pour pouvoir effectuer des communications multi-sautsfiables.
- Assurer un routage optimal si possible.
- Offrir une bonne qualité concernant les temps de latence.
- Auto-organiser le réseau.

2.3 Les différents protocoles de routage dans les RCSF

2.3.1 LEACH

LEACH (Low-Energy Adaptive ClusteringHierarchy) est un protocole qui choisit aléatoirement les nœuds cluster-Head et attribue ce rôle aux différents nœuds selon la politique de gestion Round-Robin (c'est-à-dire tourniquet) pour garantir une dissipation équitable d'énergie entre les nœuds. Dans le but de réduire la quantité d'informations transmises à la station de base, les cluster-Head agrègent les données capturées par les nœuds membres qui appartiennent à leur propre cluster, et envoient un paquet agrégé à la station de base (voir 2.1).

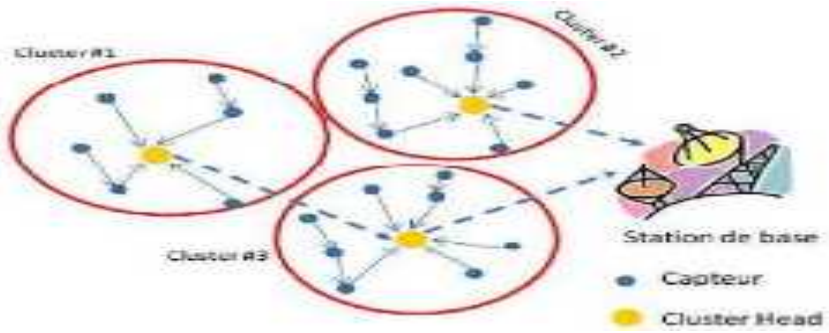


Figure 2.1 Le Clustering dans un RCSF

LEACH est exécuté en deux phases : la phase « set-up » et la phase « steady-state » suivant la figure 2.2. Dans la première phase, les clusters Head sont sélectionnés et les clusters sont formés, et dans la seconde phase, le transfert de données vers la station de base aura lieu. Durant la première phase, le processus d'élection des clusters Head est déclenché pour choisir le futur cluster Head. Ainsi, une fraction prédéterminée de nœuds s'élise comme cluster Head selon le schéma d'exécution suivant : durant une période T, un nœud n choisit un nombre aléatoire nb dont la valeur est comprise entre 0 et 1 (0 < nb < 1). Si nb est inférieure à une valeur seuil alors le nœud n deviendra cluster Head durant la période courante,sinon le nœud n devrait rejoindre le cluster Head le plus proche dans son voisinage.

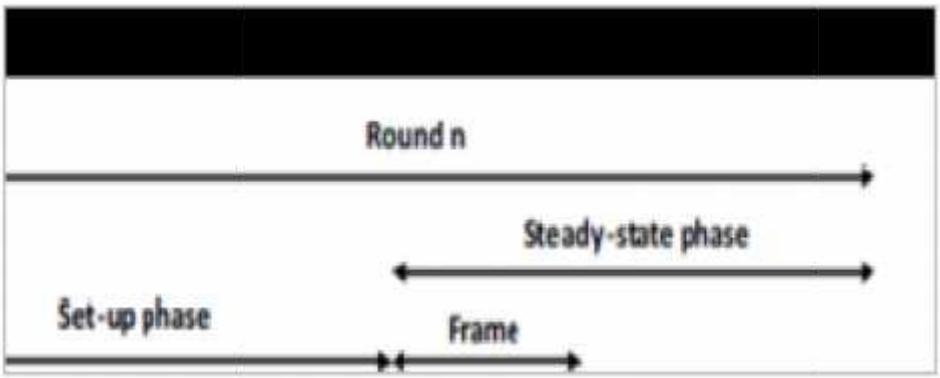


Figure 2.2 Illustration des deux fonctions de LEACH

2.3.2 PEGASIS

Power-Efficient Gathering in Sensor Information Systems (PEGASIS) est une version améliorée du protocole LEACH. PEGASIS forme des chaînes plutôt que des clusters de nœuds de capteurs afin que chaque nœud transmette et reçoiveuniquement des données d'un voisin. Un seul nœud est sélectionné à partir decette chaîne pour transmettre à la station de base. L'idée de PEGASIS est qu'ilutilise tous les nœuds pour transmettre ou recevoir des données avec ses plusproches voisins. Il déplace les données reçues de nœud à nœud, puis les donnéesseront agrégées jusqu'à ce qu'elles atteignent tous la station de base. Donc, chaquenœud du réseau est tour à tour un chef de file de la chaîne, ainsi que responsable pour

transmettre l'ensemble des données recueillies et fusionnées par la chaîne de nœuds au niveau de la station de base.
Une variante de PEGASIS appelée Hierarchical PEGASIS a été conçue afin d'améliorer PEGASIS.

2.3.3 TEEN

TEEN est conçu pour être sensible à des changements soudains des attributs tels que la température. La réactivité est importante pour les applications critiques dont le réseau fonctionne dans un mode réactif. L'architecture du réseau de capteurs est basée sur un groupement hiérarchique où les nœuds forment des clusters et ce processus va se répéter jusqu'à ce que la station de base soit atteinte.

Au début, les nœuds écoutent le médium continuellement et lorsque la valeur captée du paramètre contrôlé dépasse le seuil Hard, le nœud transmet l'information. La valeur captée est stockée dans une variable interne appelée SV. Puis, les nœuds ne transmettront des données que si la valeur courante du paramètre contrôlé est supérieure au seuil hard HT ou diffère du SV d'une quantité égale ou plus grande que la valeur du seuil Soft ST.

2.3.4 APTEEN

APTEEN est une extension du TEEN basée sur la capture périodique des données et la réaction aux événements temps-réel. Quand la station de base forme les clusters, les cluster-heads diffusent les attributs, les seuils et le plan de transmission à tous les nœuds et effectuent également l'agrégation des données afin d'économiser de l'énergie.

- **Avantages du protocole APTEEN :**

Il offre une grande flexibilité qui permet à l'utilisateur de choisir l'intervalle de temps CT, et les valeurs seuils HT et ST pour que la consommation d'énergie soit contrôlée par la variation de ces paramètres.

- **Inconvénients du protocole APTEEN :**

Il nécessite une complexité supplémentaire pour implémenter les fonctions de seuils et de périodes de temps CT. Ainsi, le surcoût et la complexité associés à la formation des clusters à plusieurs niveaux par APTEEN sont assez élevés.

2.3.5 HEED

Younes et Fahmy ont proposé un algorithme de clustering distribué appelé HEED pour les réseaux de capteurs sans fils. HEED ne fait aucune restriction sur la distribution et la densité des nœuds. Il ne dépend pas de la topologie du réseau mais il suppose que les capteurs ont la possibilité de modifier leur puissance de transmission. HEED sélectionne les cluster-heads selon un critère hybride regroupant l'énergie restante des nœuds et un second paramètre tel que le degré des nœuds. Il vise à réaliser une distribution uniforme des clusters Head dans

le réseau et à générer des clusters équilibrés en taille. Son problème réside dans la détermination du nombre optimal des clusters. De plus, HEED ne précise pas de protocole particulier à utiliser pour la communication entre les clusters-head et le sink. A l'intérieur du cluster, le problème ne se pose pas car la communication entre les membres du cluster et le cluster head est directe (à un saut). D'autre part, avec HEED, la topologie en clusters ne réalise pas de consommation minimale d'énergie dans les communications intra-cluster et les clusters générés ne sont pas équilibrés en taille.

2.4 Conclusion

Dans ce chapitre on a décrit les différents protocoles de routage hiérarchique dans les réseaux de capteurs sans fils et nous avons effectué une étude comparative entre le protocole LEACH et PEGASIS, et entre LEACH et TEEN.

Dans le prochain chapitre nous allons définir le système d'exploitation TinyOs, le langage de programmation utilisé dans les réseaux de capteurs sans fils NESC et les différents outils de simulations.

3. Présentation de l'environnement de travail

3.1 Le système d'exploitation TinyOS

TinyOS est un système d'exploitation intégré, modulaire, destiné aux réseaux de capteurs miniatures. Cette plate-forme logicielle ouverte et une série d'outils développés par l'Université de Berkeley est enrichie par une multitude d'utilisateurs. En effet, TinyOS est le plus répandu des systèmes d'exploitation pour les réseaux de capteurs sans fils. Il est utilisé dans les plus grands projets de recherches sur le sujet (plus de10.000 téléchargements de la nouvelle version). Un grand nombre de ces groupes de recherches ou entreprises participent activement au développement de cet OS (Operating System)en fournissant de nouveaux modules, de nouvelles applications, ...Cet OS est capable d'intégrer très rapidement les innovations en relation avec l'avancement des applications et des réseaux eux même tout en minimisant la taille du code source en raison des problèmes inhérents de mémoire dans les réseaux de capteurs. La librairie TinyOS comprend les protocoles réseaux, les services de distribution, les drivers pour les capteurs et les outils d'acquisition de données. TinyOS est en grande partie écrit en C mais on peut très facilement créer des applications personnalisées en langages C, NesC, et Java.

3.2 Le langage de programmation de TinyOS:NesC

Une application NesC consiste en un ou plusieurs composants assemblés pour former un exécutable. Un composant, du point de vue programmation, est composé de plusieurs sections, il offre et utilise des interfaces qui sont son unique point d'accès. Il existe deux types de composants : modules et configurations.

- **Modules :** sont des composants qui définissent le code de l'application en implémentant une ou plusieurs interfaces.
- **Configurations :** sont des composants qui assemblent les composants de l'application, en reliant les interfaces utilisées par des composants aux interfaces offertes par d'autres composants. Les éléments connectés doivent être compatibles : "Interface" à "interface", "command" à "command", "event" à "event" (Voir 3.1).

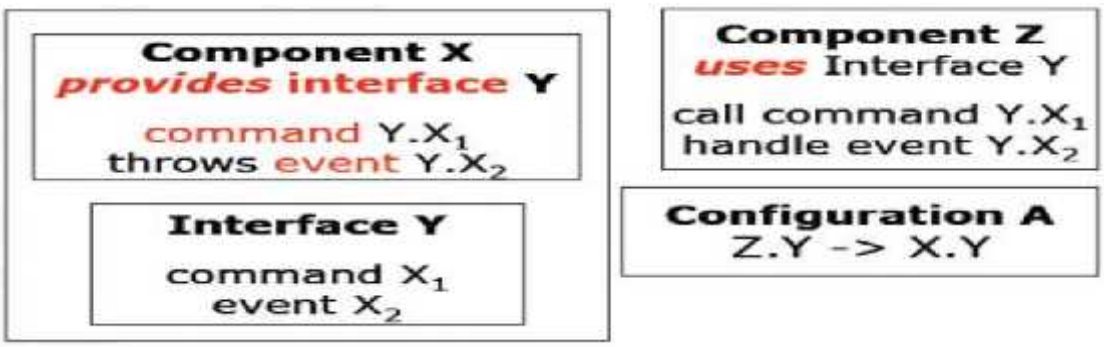


Figure 3.1 Application TinyOS

3.3 Interface

- Une interface définie les interactions entre deux composants.
- Elle définit un ensemble de fonctions appelées commandes (qui doivent êtres implémentées par le composant qui offre cette interface) et un autre ensemble de fonctions appelées événements (qui doivent êtres implémentées par le composant utilisant cette interface). Les interfaces sont utilisées pour les opérations qui décrivent des interactions bidirectionnelles.
- Pour qu'un composant puisse appeler les commandes d'une interface, il doit implémenter les événements qui y sont définis. Un même composant pourrait offrir et utiliser plusieurs interfaces différentes ou plusieurs instances d'une même interface.

3.4 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.

3.4.1 Les tâches

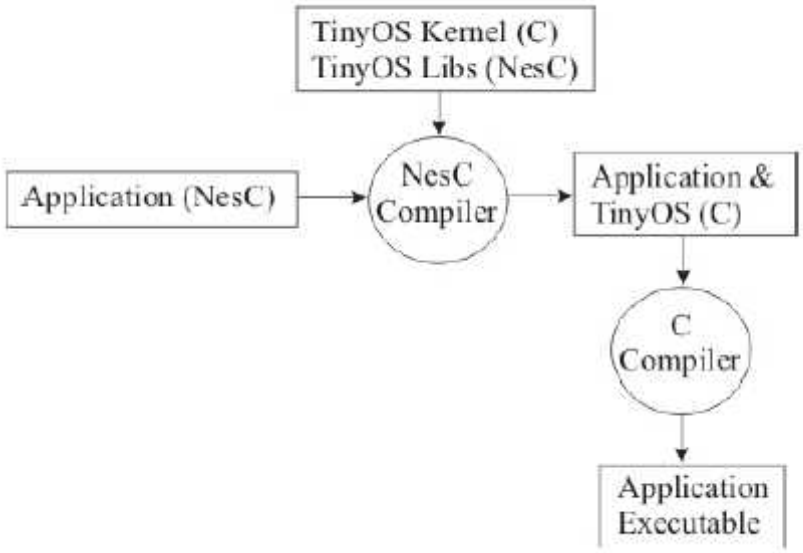
Sont des fonctions dont l'exécution est différée. Une fois lancée, une tâche s'exécute jusqu'à sa fin et ne peut pas interrompre une autre tâche.

- Une tâche est un élément de contrôle indépendant défini par une fonction retournant void et sans arguments : task void myTask() ...

3.4.2 Les handlers d'événements matériels

Sont exécutés en réponse à des interruptions matérielles (pression d'un bouton, changement d'état d'une entrée...) et ils permettent de faire le lien entre ces dernières et les couches logicielles que constituent les tâches. Ils sont prioritaires par rapport aux tâches et peuvent interrompre la tâche en cours d'exécution ou un autre "handler".

3.5 Processus de compilation



3.5.1 NESCDT : Un éditeur pour NesC dans Eclipse

C'est est un éditeur pour travailler avec NesC code dans Eclipse (comme une alternative à d'autres éditeurs). Fondamentalement, il suffit de créer un nouveau Nescdt projet dans Eclipse et lier ces dossiers à partir de l'installation de TinyOS et les arbres d'application qui sont nécessaires.

3.5.2 Débogage du programme

Il existe de nombreuses façons de débugger un programme et notamment pour NesC, on pourra utiliser Gdb qui est un debugger s'exécutant en mode console ou Eclipse plugin. EclipseIDE est conçu pour appuyer sur Gdb pour examiner l'état des programmes en cours d'exécution.

Gdb gère quatre missions principales :

- Lancement de programmes précisant les paramètres susceptibles d'influer sur son comportement.
- Arrêter le programme sur les conditions prédéfinies (points d'arrêt, points d'observation).
- Examen de l'état du programme quand il est arrêté (Stack trace, examiner les variables).
- Changer l'état du programme (pas à pas unique dans le code).

3.6 Les différents outils de simulation

TinyOs offre des outils de Simulation (TinyViz/Tossim/PowerTossim) :

3.6.1 Tossim

TOSSIM est le simulateur de TinyOs. Il permet de simuler le comportement d'un capteur (envoie/réception de messages via les ondes radios, traitementdel'information, ...) au sein d'un réseau de capteurs sans fils.Ce dernier est souvent utilisé avec une interface graphique (TinyViz) pour une meilleure compréhension et visualisation de l'´état du réseau.

3.6.2 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.

3.6.3 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. Comme nous le voyons sur la figure 3.2. Les capteurs sont disposés dans la partie gauche de la fenêtre du logiciel.

- **Mode d'emploi**

Dans cette section 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.

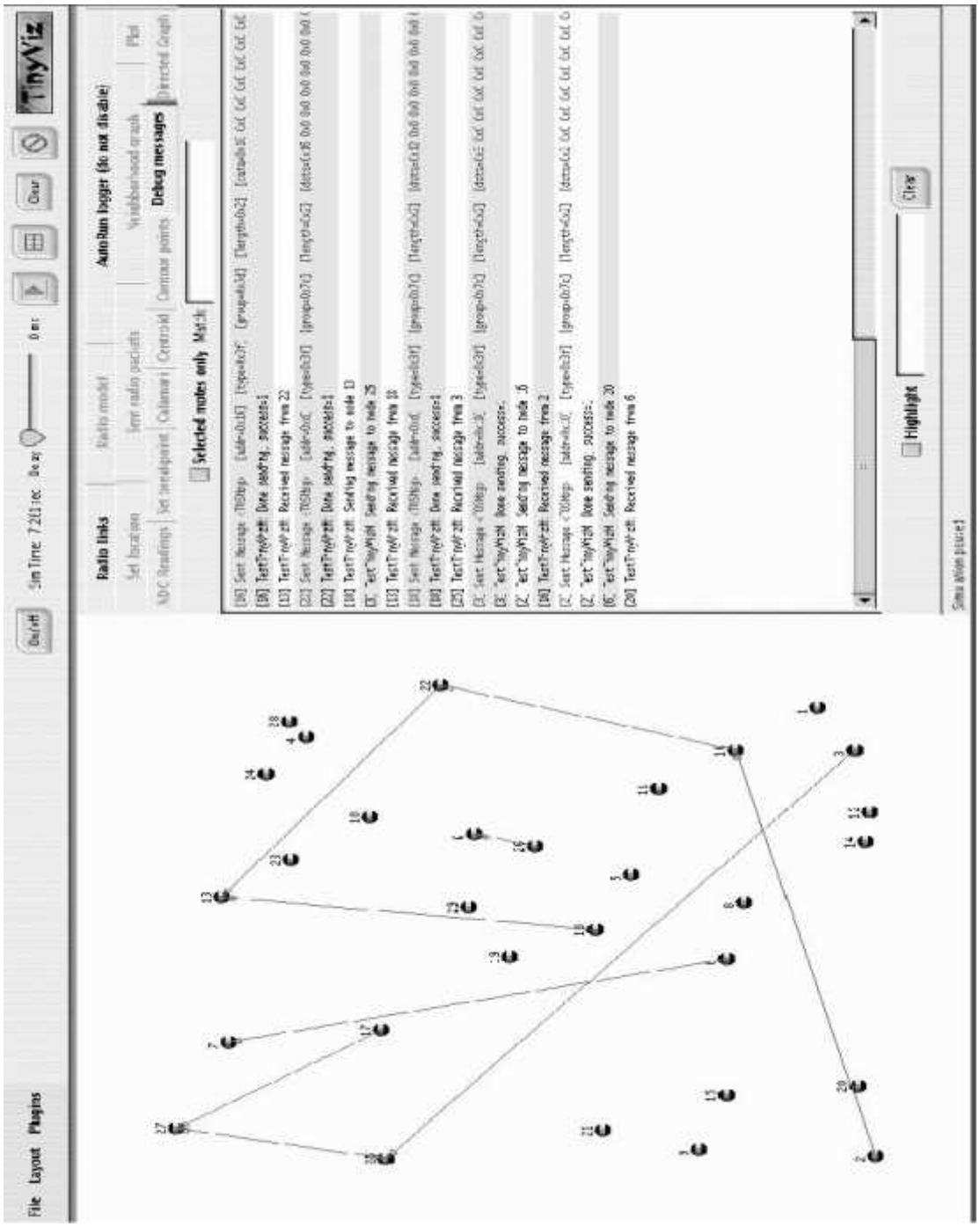


Figure 3.2 TinyViz

3.7 Conclusion

Dans ce chapitre on a présenté l'environnement du travail utilisé : le système d'exploitation TinyOs, le langage de programmation utilisé NesC et les différents outils de simulations. Dans le prochain chapitre nous allons procéder à l'étape d'implémentation du protocolede routage qu'on a choisi (le protocole TEEN).

4. Implémentation du protocole TEEN

On a choisi de travailler sur le protocole TEEN vue les perspectifs et les contraintes d'énergie fixé avec notre tuteur et vue sa capacité d'économiser l'énergie par rapport au protocole Leach.

Le Protocole TEEN minimise la consommation d'énergie des nœuds capteurs et étendre la durée de vie du réseau.

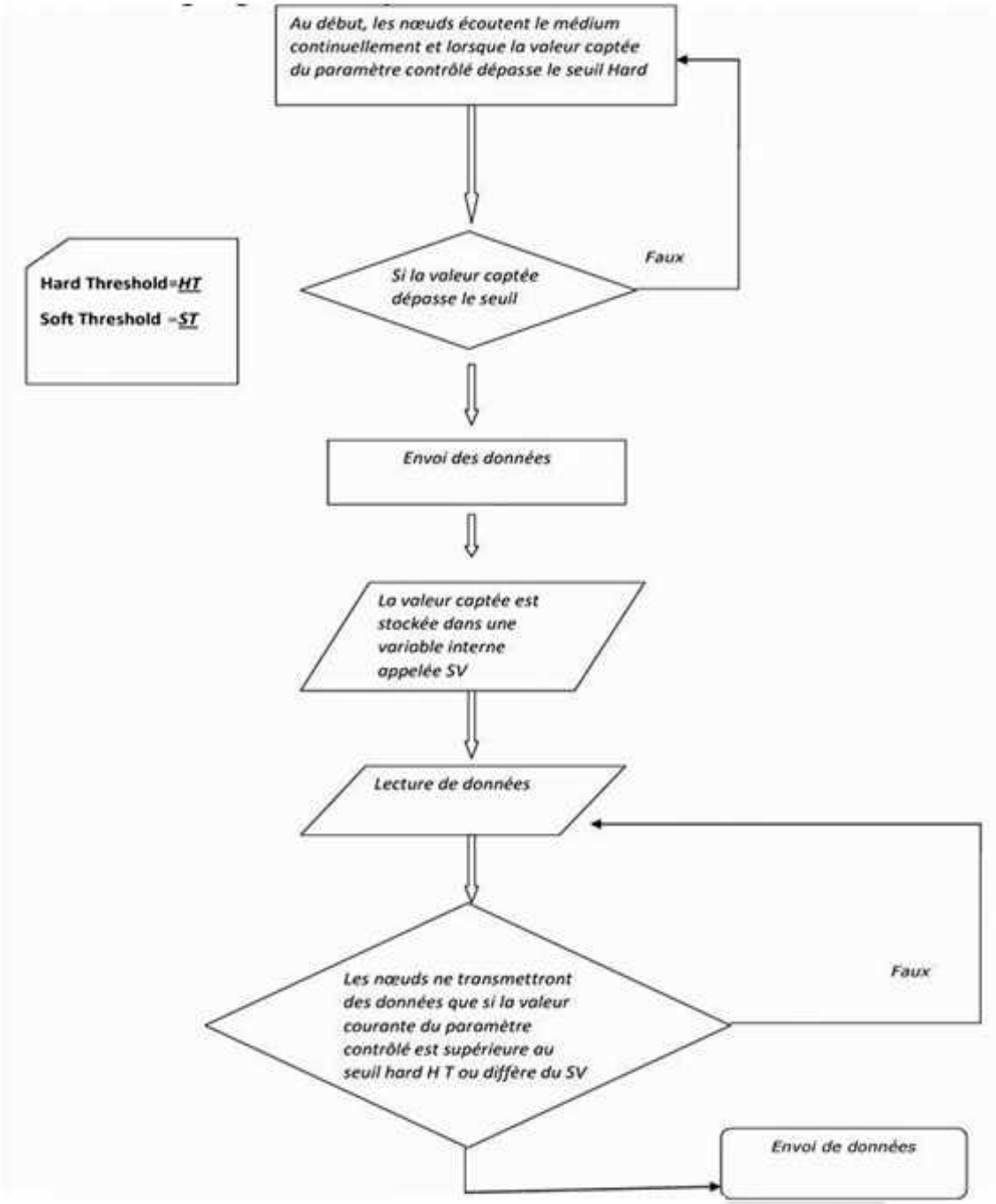
- **Avantages du protocole TEEN :**

Puisque la transmission d'un message consomme plus d'énergie que la détectiondes données, alors la consommation d'énergie dans TEEN est moins importante que dans les protocoles proactifs ou ceux qui transmettent des données périodiquement tels que LEACH.

- **Inconvénients du protocole TEEN :**

L'inconvénient principal de ce protocole est que, si les seuils HT et ST ne sont pas reçus, les nœuds ne communiqueront jamais, et aucune donnée ne sera transmise à l'utilisateur, ainsi la station de base ne connaît pas les nœuds qui ont épuisé leur énergie. TEEN ne convient pas aux applications qui nécessitent desenvois périodiques de données.

4.1 Organigramme du protocole TEEN



4.2 Résultats obtenus de la compilation

Rappel : Architecture d'un capteur

L'architecture des capteurs, en général, est commune pour tous. On peut voir sur la figure 4.1 les différents composants qui constituent un capteur.

- **Mote, Processeur, RAM, Flash :**

On appelle Mote la carte physique utilisant le système d'exploitation. Le processeur est à la base des calculs binaires. Les mémoires RAM et Flash servent pour le stockage, définitif ou non, des données.

- **Radio, Antenne:**

Afin d'émettre, un capteur a besoin d'une antenne et d'une radio pour ajuster les fréquences hertziennes.

- **LED, interfaces, capteur :**

Composants prévu pour mettre en place un réseau de capteurs.

Batterie : Indispensable pour l'autonomie du capteur.

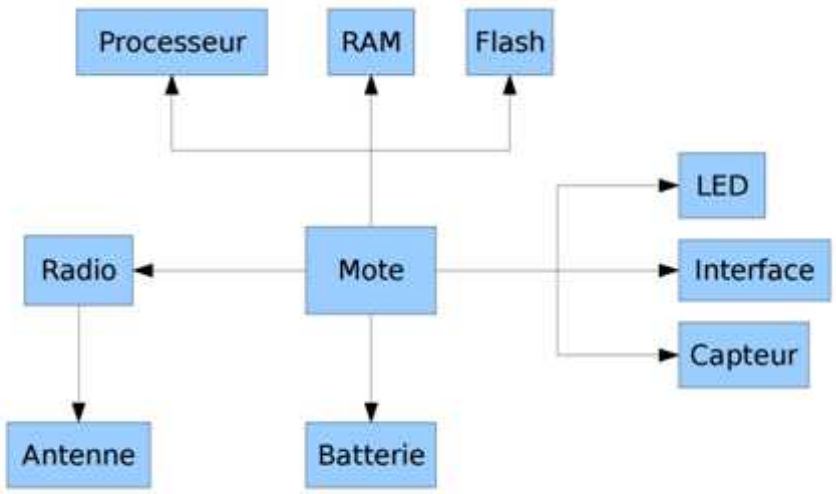
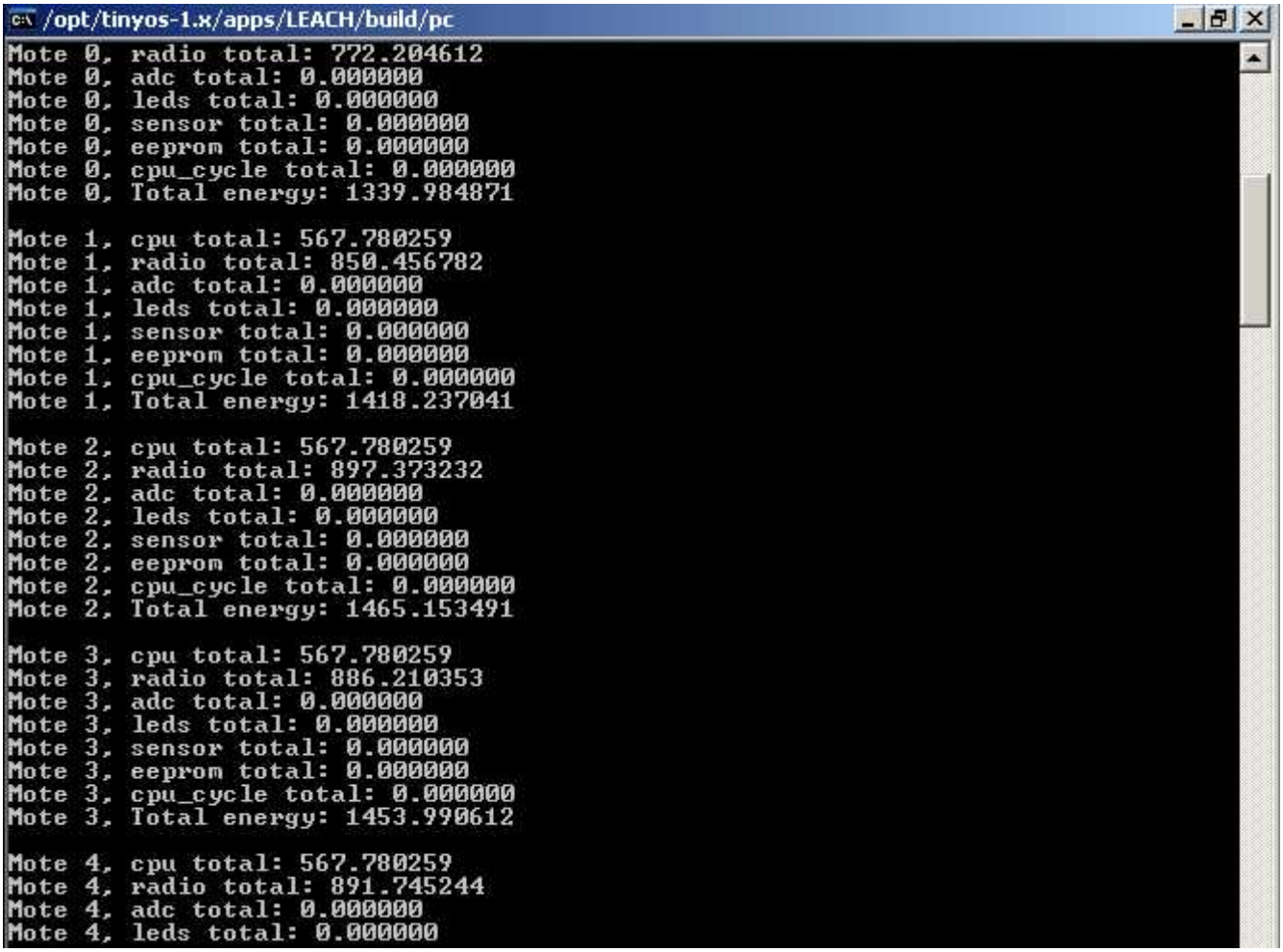
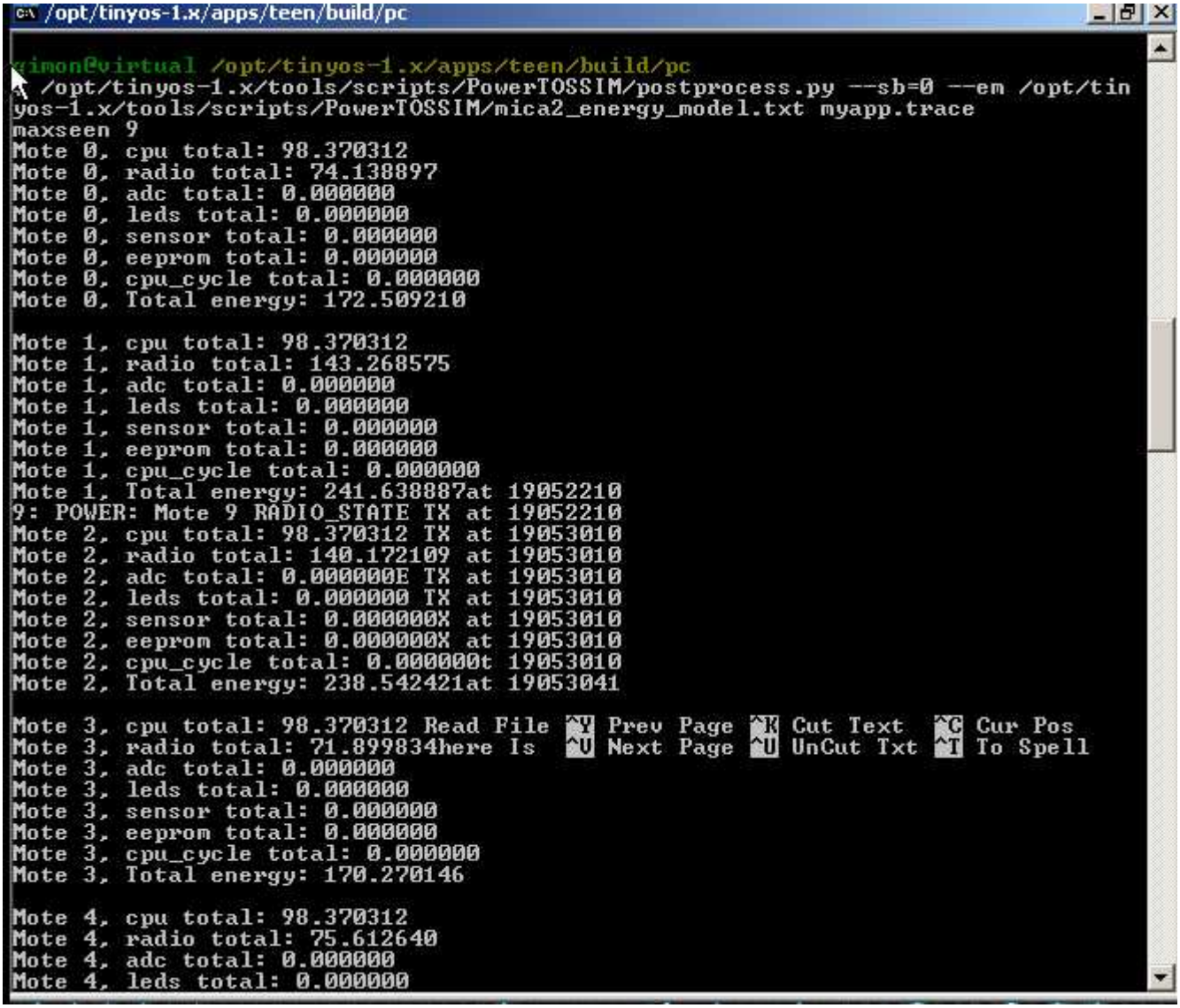


Figure 4.1 Architecture générique d'un Mote

4.2.1 LEACH


/opt/tinyos-1.x/apps/LEACH/build/pc
Mote 0, radio total: 772.204612
Mote 0, adc total: 0.000000
Mote 0, leds total: 0.000000
Mote 0, sensor total: 0.000000
Mote 0, eeprom total: 0.000000
Mote 0, cpu_cycle total: 0.000000
Mote 0, Total energy: 1339.984871
Mote 1, cpu total: 567.780259
Mote 1, radio total: 850.456782
Mote 1, adc total: 0.000000
Mote 1, leds total: 0.000000
Mote 1, sensor total: 0.000000
Mote 1, eeprom total: 0.000000
Mote 1, cpu_cycle total: 0.000000
Mote 1, Total energy: 1418.237041
Mote 2, cpu total: 567.780259
Mote 2, radio total: 897.373232
Mote 2, adc total: 0.000000
Mote 2, leds total: 0.000000
Mote 2, sensor total: 0.000000
Mote 2, eeprom total: 0.000000
Mote 2, cpu_cycle total: 0.000000
Mote 2, Total energy: 1465.153491
Mote 3, cpu total: 567.780259
Mote 3, radio total: 886.210353
Mote 3, adc total: 0.000000
Mote 3, leds total: 0.000000
Mote 3, sensor total: 0.000000
Mote 3, eeprom total: 0.000000
Mote 3, cpu_cycle total: 0.000000
Mote 3, Total energy: 1453.990612
Mote 4, cpu total: 567.780259
Mote 4, radio total: 891.745244
Mote 4, adc total: 0.000000
Mote 4, leds total: 0.000000

4.2.2 TEEN


/opt/tinyos-1.x/apps/teen/build/pc
gimon@virtual /opt/tinyos-1.x/apps/teen/build/pc
/opt/tinyos-1.x/tools/scripts/PowerTOSSIM/postprocess.py --sb=0 --em /opt/tinyos-1.x/tools/scripts/PowerTOSSIM/mica2_energy_model.txt myapp.trace
maxseen 9
Mote 0, cpu total: 98.370312
Mote 0, radio total: 74.138897
Mote 0, adc total: 0.000000
Mote 0, leds total: 0.000000
Mote 0, sensor total: 0.000000
Mote 0, eeprom total: 0.000000
Mote 0, cpu_cycle total: 0.000000
Mote 0, Total energy: 172.509210
Mote 1, cpu total: 98.370312
Mote 1, radio total: 143.268575
Mote 1, adc total: 0.000000
Mote 1, leds total: 0.000000
Mote 1, sensor total: 0.000000
Mote 1, eeprom total: 0.000000
Mote 1, cpu_cycle total: 0.000000
Mote 1, Total energy: 241.638887at 19052210
9: POWER: Mote 9 RADIO_STATE TX at 19052210
Mote 2, cpu total: 98.370312 TX at 19053010
Mote 2, radio total: 140.172109 at 19053010
Mote 2, adc total: 0.000000E TX at 19053010
Mote 2, leds total: 0.000000 TX at 19053010
Mote 2, sensor total: 0.000000X at 19053010
Mote 2, eeprom total: 0.000000X at 19053010
Mote 2, cpu_cycle total: 0.000000t 19053010
Mote 2, Total energy: 238.542421at 19053041
Mote 3, cpu total: 98.370312 Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
Mote 3, radio total: 71.899834here Is ^U Next Page ^U UnCut Txt ^T To Spell
Mote 3, adc total: 0.000000
Mote 3, leds total: 0.000000
Mote 3, sensor total: 0.000000
Mote 3, eeprom total: 0.000000
Mote 3, cpu_cycle total: 0.000000
Mote 3, Total energy: 170.270146
Mote 4, cpu total: 98.370312
Mote 4, radio total: 75.612640
Mote 4, adc total: 0.000000
Mote 4, leds total: 0.000000

4.2.3 Comparaison entre LEACH et TEEN

LEACH	TEEN
* LEACH est un protocole orienté temps * suppose que tous les nœuds puissent transmettre des données avec une grande puissance pour atteindre la station de base.	*TEEN est un protocole orienté événement * TEEN n'utilise pas TDMA qui est inadapté pour les applications orientées événements. Les chefs de zones élus dans TEEN ne transmettent donc pas de schedule TDMA à leurs membres, mais un message qui contient la tâche demandée au capteur

4.3 Simulation avec TinyViz

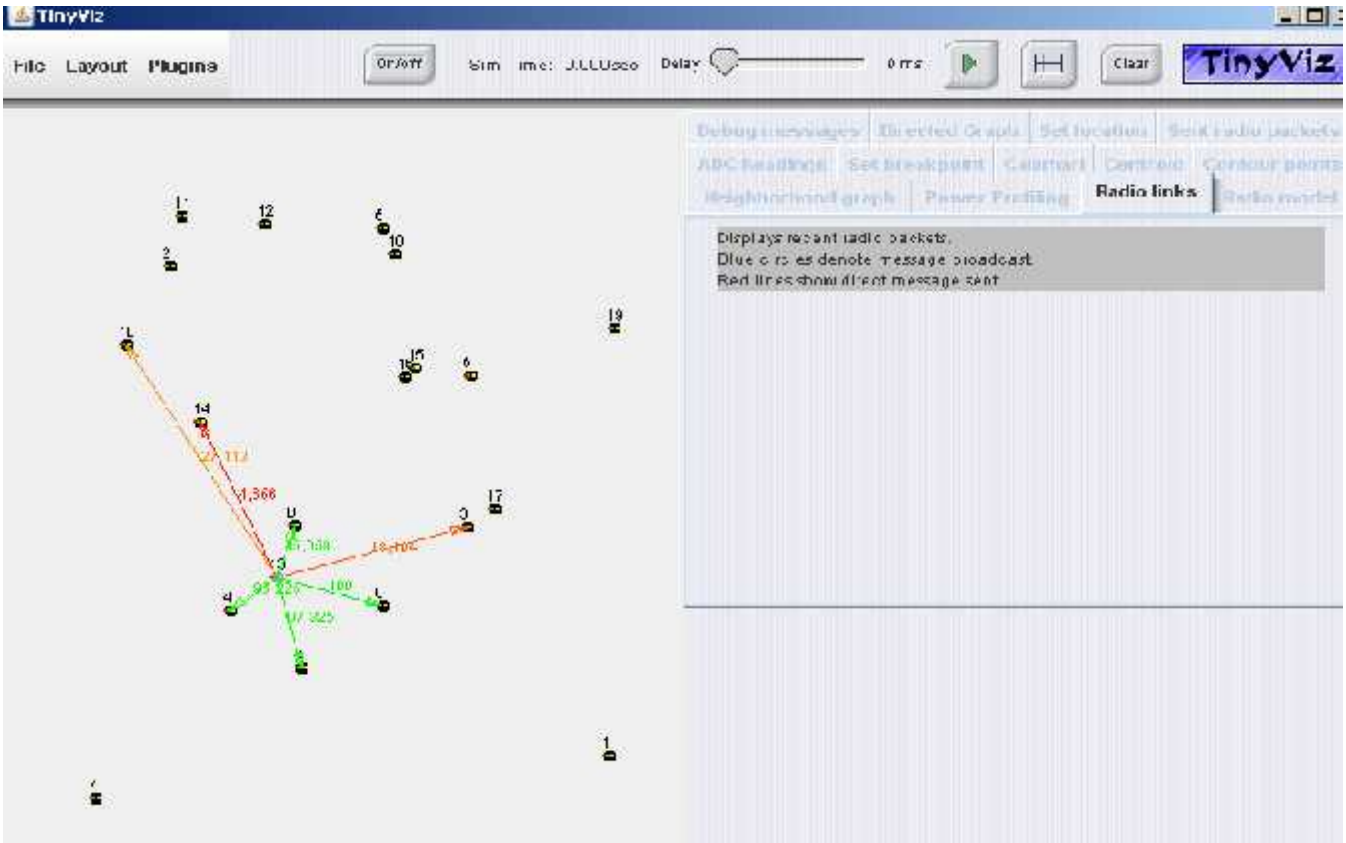


Figure 4.2 LEACH

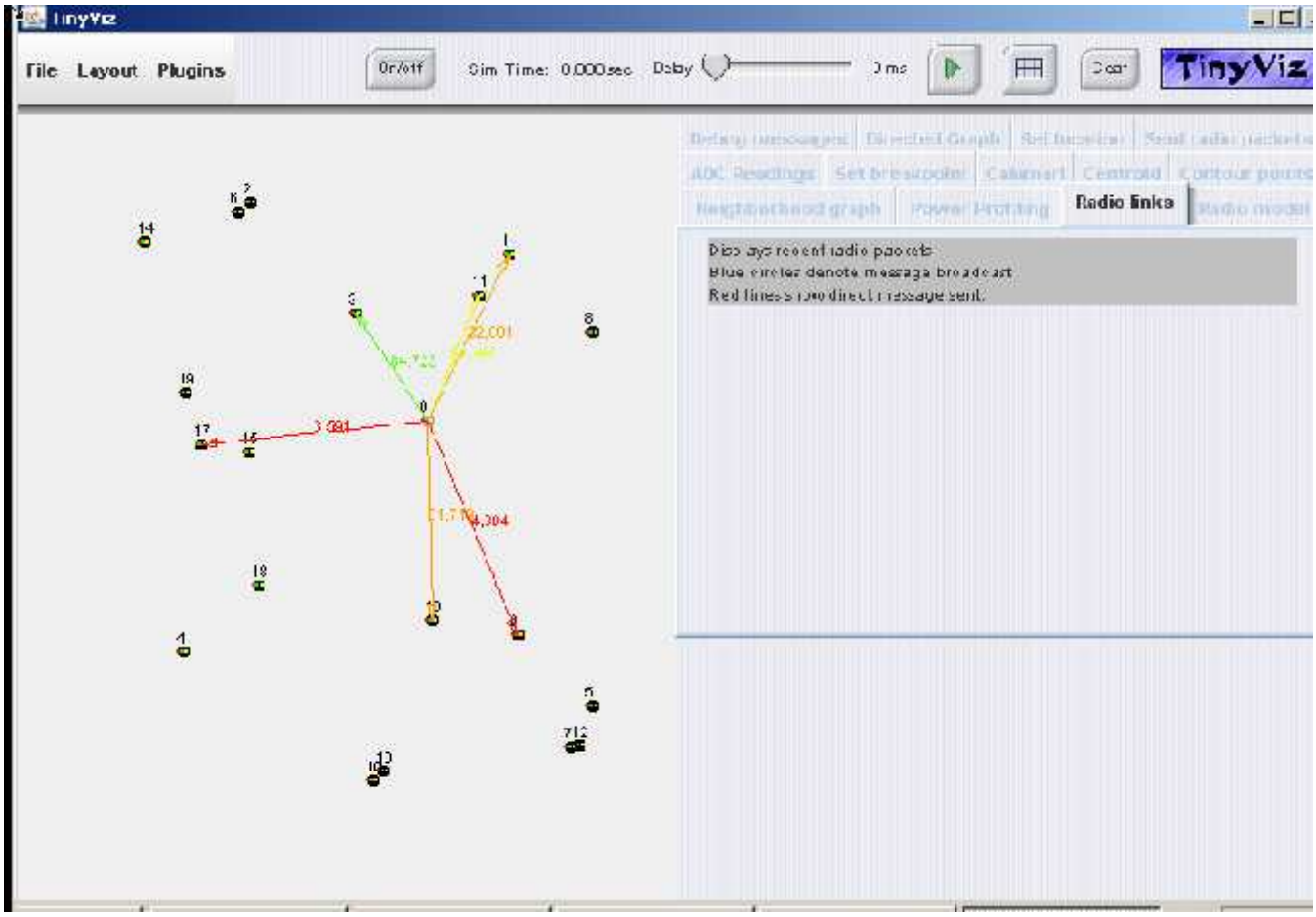


Figure 4. 3 TEEN

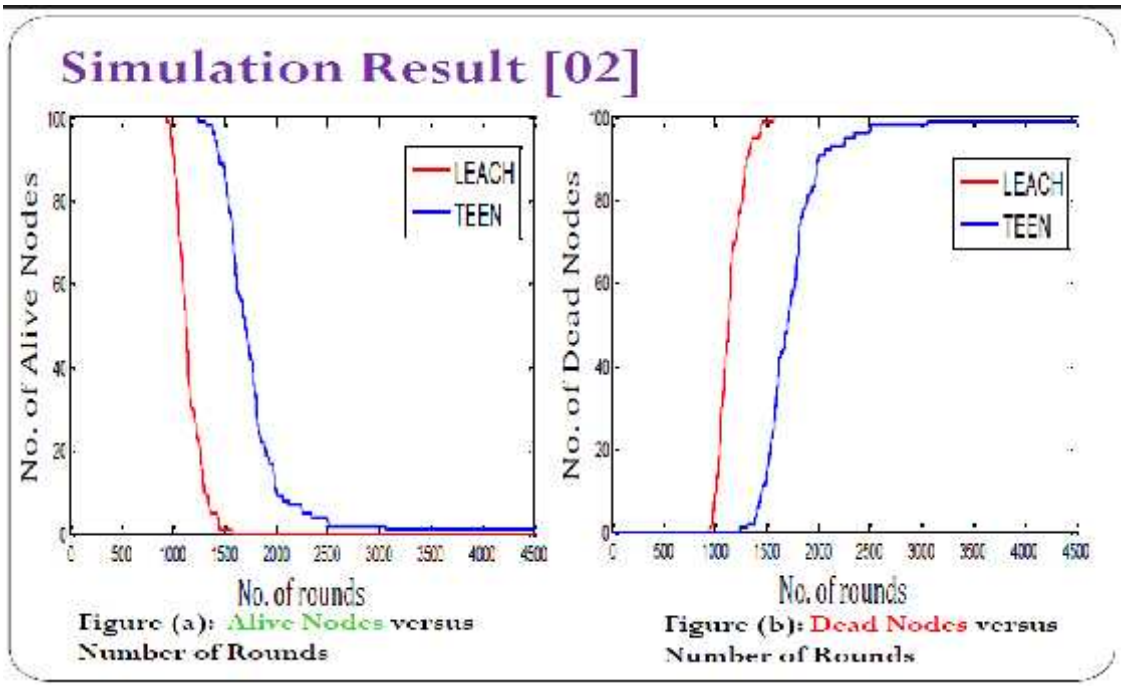


Figure 4.4 Simulé avec Matlab

4.4 Conclusion

La majorité du comportement de TEEN est semblable au protocole LEACH. Cependant, quelques différences existent, les chefs élus ne transmettent pas un Schedule TDMA, mais émettent un message contenant les informations suivantes :

• **Attributs** : représentent la tâche demandée au capteur.

• **Hard threshold (HT)** : détermine la valeur critique après laquelle les membres doivent envoyer leurs rapports de données.

• **Soft threshold (ST)** : spécifie le changement minimal obligeant le nœud à envoyer un nouveau rapport. Donc, lorsqu'un nœud s'aperçoit que la valeur captée a dépassé HT, il doit émettre un rapport au chef. Il ne réémet un nouveau rapport que si la valeur change radicalement, i.e. : la différence dépasse ST. Ce mécanisme permet d'implémenter un comportement réactif,tout en limitant le nombre de messages utilisés.

4.5 Problèmes rencontrés

- L'installation de TinyOS nous a pris beaucoup de temps (voir annexe).
- Manque de documentation sur le protocole que nous avons implémenté (TEEN).
- La plupart de la documentation sur les réseaux de capteurs sans fil est en anglais.

Conclusion Générale

Les réseaux de capteurs sans fil nous intéressaient déjà bien avant de choisir ce sujet et maintenant notre passion ne cesse d'augmenter Ce projet nous a permis aussi d'apprendre plus de choses et d'acquérir de nouvelles connaissances théoriqueset pratiques dans les réseaux de capteurs sans fil (Système d'exploitation TinyOs et le langage de programmation NesC), et de travailler en équipe puisqueon devait partager les tâches pour avancer le plus vite possible.

La compréhension du fonctionnement des modèles TinyOS ainsi que le développement sous NesC, nous a permis de comprendre de manière plus efficace la mise en œuvre d'un RCSF.

Références

- UC Berkeley, “Smart dust: Autonomous sensing and communication in acubic millimeter.”robotics.eecs.berkeley.edu/ pister/SmartDust.

- P. Levis, N. Lee, M. Welsh, and D. Culler, “Tossim: accurate and scalablesimulation of entire tinyos applications,” in SenSys ’03: Proceedings of the 1st international conference on Embedded networked sensor systems. NewYork, NY, USA : ACM Press, pp. 126–137.

- TinyOS Team, “Tinyos.” www.tinyos.net

Annexe 1 :

1 Guide d'installation

Deux principales versions de TinyOS sont disponibles : la version stable (1.1.0) et la version en développement (2.0.2) qui nécessite l'installation de l'ancienne version pour fonctionner. TinyOS peut être installé sur Windows (2000 et XP), GNU/Linux, Mac OS ou sur un capteur. Nous avons procédé à l'installation de la première version de TinyOS sur Windows XP.

1.1. Procédure d'installation sous Windows XP

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

1- Télécharger le fichier tinyos-1.1.0-1is.exe de la source http://www.tinyos.net/dist-1.1.0/tinyos/windows/ .
2- Exécuter ce fichier pour installer la version 1.1.0 sous Windows XP. L'installation se fait automatiquement. Un raccourci de Cygwin est sauvegardé sur le bureau.
3- Accéder à **C:\tinyos\cygwin\opt\tinyos-1.x\doc\tutorial\verifyhw.html** et suivre les étapes que contient cette page afin de vérifier si l'installation est bien réussie.

2. Installation de TinyViz

Les concepteurs développent au fur et à mesure l'outil TinyViz sans mettre à jour les fichiers sources déjà existants dans les anciennes versions. Cela ne permet pas de lancer TinyViz dans des conditions normales. Pour pouvoir le lancer, il est nécessaire de passer par les étapes suivantes :

1- Installer TinyOS-1.0
2- Accéder à :**cd /opt/tinyos-1.x/tools/java ET** taper : **make**
3- Installer les mises à jour de NesC1.1.1 and TinyOS1.1.15.
Pour se faire, rechercher sur le net http://www.tinyos.net/dist-1.1.0/tinyos/windows/ ces mises à jour en téléchargeant le **rpm** et le mettant dans **C:\tinyos\cygwin\home\PLANETE PC**
Et taper dans le **shell**:
rpm -ivh --ignoreos nesc-1.1.2b-1.cygwin.i386.rpm
rpm -ivh --ignoreos --force tinyos-1.1.15Dec2005cvs-1.cygwin.noarch.rpm
4- Aller à **opt/tinyos-1.x/tools/java/net/tinyos/sim** et vérifier si ces fichiers sont présents : **SimObjectGenerator.java** et **MoteSimObjectGenerator.java** s'ils existent, alors les supprimer de ce répertoire.
5- Editer le **makefile** qui est dans **C:\tinyos\cygwin\opt\tinyos-1.x\tools\java\net\tinyos\sim** et écrire cette instruction : **net/tinyos/message/avrmote/*.class**
(Voir ** makefile pour vérifier là où il faut insérer cette instruction)
6- Aller à **shell**et taper:

cd /opt/tinyos-1.x/tools/java/net/tinyos/sim
make clean
make
7- Accéder à l'application qui va être simulée. On prend par exemple, l'application Teen. Accéder au **shell** et faire:
cd opt/tinyos-1.x/apps/Teen
make pc
 # tinyviz
export PATH="$TOSROOT/tools/java/net/tinyos/sim: $PATH"
TinyViz -runbuild/pc/main.exe 20 ///Insérer le nombre de nœuds. Par exemple 20
**** makefile:**
Voici le makefile à éditer. Dans le niveau indiqué (la ligne avec un fond coloré), insérer l'instruction **net/tinyos/message/avrmote/*.class** si elle n'existe pas :

```
SUBDIRS = event plugins packet lossy script
ROOT = ../../..
PLUGINS_SRC = $(wildcard plugins/*.java)
PLUGINS = $(PLUGINS_SRC:.java=.class)
INITIAL_TARGETS = msgsjython ../sf/old/nido/NidoSerialDataSource.class
OTHER_CLEAN = msgs-clean plugins-list-clean jarclean
# Uncomment this line to make jarfile mandatory
FINAL_TARGETS = jarfile
include $(ROOT)/Makefile.include
../sf/nido/NidoSerialDataSource.class: ../sf/old/nido/NidoSerialDataSource.java
(cd ../sf/nido; $(MAKE))
```

```
msgs:
(cd msg; $(MAKE))
msgs-clean:
(cd msg; $(MAKE) clean)
# Make sure that jython gets built
jython: $(ROOT)/org/python/core/parser.class
$(ROOT)/org/python/core/parser.class:
(cd $(ROOT)/org/python && $(MAKE))
(cd $(ROOT)/org/apache && $(MAKE))
# Create a list of default plugins
plugins/plugins.list: $(PLUGINS)
echo $(PLUGINS) > plugins/plugins.list
plugins-list-clean:
rm -f plugins/plugins.list
# This is ugly. The only way to embed a jar file inside another is to
# unpack it and repack them together into a single flat file.
jarfile: plugins/plugins.list
@echo "Creating simdriver.jar..."
(cd $(ROOT); \
jar cmf net/tinyos/sim/simdriver.manifest \
net/tinyos/sim/simdriver-tmp.jar \
net/tinyos/sim/*.class \
net/tinyos/sim/event/*.class \
net/tinyos/sim/lossy/*.class \
net/tinyos/sim/msg/*.class \
net/tinyos/sim/packet/*.class \
net/tinyos/sim/plugins/*.class \
net/tinyos/sim/script/*.class \
net/tinyos/sim/script/reflect/*.class \
net/tinyos/sim/ui \
net/tinyos/sim/plugins/plugins.list \
net/tinyos/sf/*.class \
net/tinyos/util/*.class \
net/tinyos/packet/*.class \
```

```
net/tinyos/message/*.class \
net/tinyos/message/avrmote/*.class \
org/apache/oro/text/regex/*.class \
org/python/compiler/*.class \
org/python/core/*.class \
org/python/modules/*.class \
org/python/parser/*.class \
org/python/parser/ast/*.class \
org/python/rmi/*.class \
org/python/util/*.class)
rm -rfjarbuild-tmp
mkdirjarbuild-tmp
(cd jarbuild-tmp; jar xf ../simdriver-tmp.jar; jar xf ./$(ROOT)/jars/oalnf.jar; rm -rf
METAINF;
jar cmf ../simdriver.manifest../simdriver.jar.)
rm -rf simdriver-tmp.jar jarbuild-tmp
jarclean:
rm -f simdriver.jar
```

3. Lancer PowerTOSSIM

A- Pour récupérer l'énergie consommée par les nœuds du réseau, il faut passer par ces étapes :
1- Accéder à l'application à simuler et la compiler en tapant: **make pc**
2- Taper **export DBG=power**
3- Exécuter **main.exe** en choisissant le temps de simulation avec **–t** et le nombre de nœuds du réseau avec **–p**. Une trace de simulation est enregistré dans un fichier dont l'extension est trace. Pour ce faire, taper **: /build/pc/main.exe -t=60 -p 10 >Teen.trace**
(Le temps est égal à 60 secondes et le nombre de nœuds à 10)
4- Exécuter **postprocess.py** sur la trace de simulation en spécifiant les paramètres –sb et --em **/opt/tools/scripts/PowerTOSSIM/postprocess.py –sb=0 –em /opt/tools/scripts/Power TOSSIM/mica2_energy_model.txt Teen.trace**
Le paramètre –sb spécifie si les nœuds sont attachés à un autre nœud (i.e. embarqué). En outre, le paramètre –em spécifie le modèle d'énergie.
Pour plus de détail sur l'utilisation d'autres paramètres de PowerTOSSIM, exécuter **postprocess.py –help**

5-Le résultat enregistre l'énergie totale utilisée par chaque composant sur chaque nœud. Il est sous la forme suivante :

```
Mote 0, cpu total: 98.370312
Mote 0, radio total: 74.138897
Mote 0, adc total: 0.000000
Mote 0, leds total: 0.000000
Mote 0, sensor total: 0.000000
Mote 0, eeprom total: 0.000000
Mote 0, cpu_cycle total: 0.000000
Mote 0, Total energy: 172.509210
.
.
Mote 9, cpu total: 50.854043
Mote 9, radio total: 86.658596
Mote 9, adc total: 0.000000
Mote 9, leds total: 0.000000
Mote 9, sensor total: 0.000000
Mote 9, eeprom total: 0.000000
Mote 9, cpu_cycle total: 0.000000
Mote 9, Total energy: 137.512639
```

6- Pour ne pas perdre ce résultat, il est recommandé de le sauvegarder dans un fichier texte. Pour ce faire, Ajouter dans l'instruction de l'étape 4 :
/opt/tools/scripts/PowerTOSSIM/postprocess.py –sb=0 –em /opt/tools/scripts/Power TOSSIM/mica2_energy_model.txt Teen.trace> Result.txt
7- Pour avoir un résultat d'énergie plus détaillé, ajouter le paramètre **–detail** dans l'instruction de l'étape 4. Le résultat est enregistré automatiquement dans des fichiers textes dont le nombre est égal au nombre de nœuds simulés. Autrement dit, chaque fichier contient le détail de la consommation énergétique d'un seul nœud du réseau.

B- Pour récupérer l'état de l'horloge lors de la transmission et de la réception de paquets, il faut passer par ces étapes :

1- Accéder à l'application à simuler et la compiler en tapant: **make pc**

2- Taper **export DBG=clock**

Pour afficher des messages en parallèle avec l'horloge, taper **export DBG=clock,usr1**

3- Exécuter **main.exe** en tapant **: /build/pc/main.exe -t=60 -p 10 >Teen.trace**

4- Accéder au fichier **Teen.trace**

Il contient des lignes sous la forme suivante :

2: CLOCK: event handled for mote 2 at **0:0:36.47777400** (347634 ticks).
2: CLOCK: Setting clock interval to 218 @ 0:0:36.47777400
2: j'envoie le paquet de données à la destination 42 ///DBG usr1
42: j'ai reçu le paquet de données de la source 2 ///DBG usr1
.
.
42: CLOCK: event handled for mote 42 at **0:0:36.60979650** (902286 ticks).
42: CLOCK: Setting clock interval to 231 @ 0:0:36.60979650

Annexe 2 : Résultats TEEN

Mote 0, cpu total: 98.370312

Mote 0, radio total: 74.138897

Mote 0, adc total: 0.000000

Mote 0, leds total: 0.000000

Mote 0, sensor total: 0.000000

Mote 0, eeprom total: 0.000000

Mote 0, cpu_cycle total: 0.000000

Mote 0, Total energy: 172.509210

Mote 1, cpu total: 98.370312

Mote 1, radio total: 143.268575

Mote 1, adc total: 0.000000

Mote 1, leds total: 0.000000

Mote 1, sensor total: 0.000000

Mote 1, eeprom total: 0.000000

Mote 1, cpu_cycle total: 0.000000

Mote 1, Total energy: 241.638887

Mote 2, cpu total: 98.370312

Mote 2, radio total: 140.172109

Mote 2, adc total: 0.000000

Mote 2, leds total: 0.000000

Mote 2, sensor total: 0.000000

Mote 2, eeprom total: 0.000000

Mote 2, cpu_cycle total: 0.000000

Mote 2, Total energy: 238.542421

Mote 3, cpu total: 98.370312

Mote 3, radio total: 71.899834

Mote 3, adc total: 0.000000

Mote 3, leds total: 0.000000

Mote 3, sensor total: 0.000000

Mote 3, eeprom total: 0.000000

Mote 3, cpu_cycle total: 0.000000

Mote 3, Total energy: 170.270146

Mote 4, cpu total: 98.370312

Mote 4, radio total: 75.612640

Mote 4, adc total: 0.000000

Mote 4, leds total: 0.000000

Mote 4, sensor total: 0.000000

Mote 4, eeprom total: 0.000000

Mote 4, cpu_cycle total: 0.000000

Mote 4, Total energy: 173.982952

Mote 5, cpu total: 98.370312

Mote 5, radio total: 82.315301

Mote 5, adc total: 0.000000

Mote 5, leds total: 0.000000

Mote 5, sensor total: 0.000000

Mote 5, eeprom total: 0.000000

Mote 5, cpu_cycle total: 0.000000

Mote 5, Total energy: 180.685613

Mote 6, cpu total: 98.370312

Mote 6, radio total: 167.539801

Mote 6, adc total: 0.000000

Mote 6, leds total: 0.000000

Mote 6, sensor total: 0.000000

Mote 6, eeprom total: 0.000000

Mote 6, cpu_cycle total: 0.000000

Mote 6, Total energy: 265.910113

Mote 7, cpu total: 95.396216

Mote 7, radio total: 0.000000

Mote 7, adc total: 0.000000

Mote 7, leds total: 0.000000

Mote 7, sensor total: 0.000000

Mote 7, eeprom total: 0.000000

Mote 7, cpu_cycle total: 0.000000

Mote 7, Total energy: 95.396216

Mote 8, cpu total: 95.396216

Mote 8, radio total: 162.477357

Mote 8, adc total: 0.000000

Mote 8, leds total: 0.000000

Mote 8, sensor total: 0.000000

Mote 8, eeprom total: 0.000000

Mote 8, cpu_cycle total: 0.000000

Mote 8, Total energy: 257.873574

Mote 9, cpu total: 50.854043

Mote 9, radio total: 86.658596

Mote 9, adc total: 0.000000

Mote 9, leds total: 0.000000

Mote 9, sensor total: 0.000000

Mote 9, eeprom total: 0.000000

Mote 9, cpu_cycle total: 0.000000

Mote 9, Total energy: 137.512639