Wireless Sensor networks

Lab-1: First step on TinyOS

Initial Configuration

1. In a terminal, type:

```
$ tos1
$ sudo updatedb
$ locate comm.jar
```

2. Add the third path to the CLASSPATH variable, e.g. :

```
$ export CLASSPATH=$CLASSPATH:/opt/tinyos-2.x-
   contrib/rincon/support/sdk/java/com/rincon/comports/comm.jar
```

3. Compile tinyos tool:

```
$ cd /opt/tinyos-1.x/tools/java
$ make
```

4. Test the tinyviz tool.

```
$ tinyviz
```

Exercise 1 : *TinyOS discovery.*

This application simply causes the red LED on the mote to turn on and off at 1Hz. Blink application is composed of two components: a module, called "BlinkM.nc", and a configuration, called "Blink.nc". Remember that all applications require a top-level configuration file, which is typically named after the application itself. In this case Blink.nc is the configuration for the Blink application and the source file that the nesC compiler uses to generate an executable file. BlinkM.nc, on the other hand, actually provides the implementation of the Blink application. As you might guess, Blink.nc is used to wire the BlinkM.nc module to other components that the Blink application requires.

In a terminal :

1. Customize the version tinyos-1.x
   `$ tos1`

2. List the files of the **`Blink`** application:
   `$ cd /opt/tinyos-1.x/apps/Blink`

3. Compile than ran the application **`Blink`** with **`TOSSIM`** simulator.

```
$ make pc
$ build/pc/main.exe 1      /* run w. 1 node */
```

4. Run with graphical interface **tinyviz**.
   ```
   $ tinyviz –run build/pc/main.exe 1
   ```

Exercise 2: Sensor network with a ring topology.

An application context requires the deployment of a sensor network with a ring topology. The network consists of ten nodes numbered from 0 to 9. Each node performs a read and then sends it to the next node. The purpose of such a deployment is to avoid any central equipment on the one hand. On the other hand, we can know if there is an anomaly by querying any node in the network.

1. Download the source code on the following link :
   www.irit.fr/~Rahim.Kacimi/supports/AppRing.zip
2. Complete the **'AppRing'** application skeleton.
3. Activate the **'Radio Links'** and **'Debug messages'** Plugins to observe the messages exchanged between the sensors.

> *Before running, activate the USR1 debug profile*
>
> ```
> $ export DBG=usr1
> ```

Exercice 3 : *Energy consumption.*

TOSSIM offers the possibility to follow the energy consumption of the nodes. This goes through the use of the **-p** option for execution.

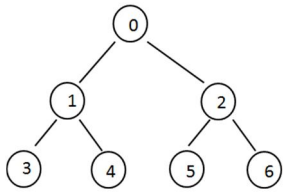
```
$ tinyviz –run 'build/pc/main.exe –p' 10
```

Energy profiling is visible with the tinyviz tool. Simply activate the **"Power Profiling"** tab to observe the consumption (in mW) of each component and all the nodes of the network.

> *Here, activate power and USR1 debug profiles :*
>
> ```
> $ export DBG=usr1,power
> ```

Exercise 4: ***Reduced version of LEACH protocol.***

Here we develop a TinyOS application to deploy a network with an arbitrary number of nodes in a static binary tree topology. Here is an example of a network with eight sensor-nodes numbered from 0 to 7. The network have the following topology:



There are three types of nodes:

- Each leaf node generates a random value (in the interval [0..7]) every three seconds, displays it on its LEDs and sends it to its parent node.
- Each parent node keeps track of the values for its two childs, and displays the maximum using the LEDs. Whenever the maximum changes, it sends this new value to its parent.
- The base station (node 0) also displays the maximum value of its two wires, but transmits no messages.

1. Complete the skeleton of the **`Tree`** application (Tree.zip) to implement the topology.
2. Compare the energy consumption of both **`AppRing`** and **`Tree`** applications.

ANNEXE

Manipulation des LED :

```
call Leds.init();    // initialization des Leds
call Leds.greenOn(); // allumer le Led vert
call Leds.yellowOff();  // éteindre le Led jeaune
call Leds.redToggle(); // faire clignoter le Led rouge
```

Manipulation du Timer :

```
call Timer.start(OPTION, FREQUENCE);
```

ex :

```
call Timer.start(TIMER_REPEAT, 1000); // démarrer un timer générant
                                      // des tops chaque 1000 millisecondes
call Timer.stop() ; // arrêter le timer
```

Adressage :

L'adresse destination peut un être un simple numéro désignant le nœud destinataire comme elle peut être sous une des formes suivantes :

```
TOS_LOCAL_ADDRESS+1  // adresse du nœud ayant l'id suivant
TOS_BCAST_ADDR       // adresse de diffusion
```

L'adresse du nœud courant ou l'adresse locale est écrite comme suit :

```
TOS_LOCAL_ADDRESS    // adresse locale (du nœud courant)
```

Envoi de Message :

```
call SendMsg.send(TOS_BCAST_ADDR, sizeof(uint16_t), &msg);
                                                  // diffusion du message msg

call SendMsg.send(3, sizeof(uint16_t), &msg);
                                  // Diffusion du message msg au noeud 3
```

Détection et mesure :

```
call ADC.getData();   // lancer la détection par le composant ADC
```