Instrucciones para compilar y ejecutar aplicaciones sobre TOSSIM, el simulador de TinyOS

Francisco Javier Rincón Vallejos
francisco.rincon@epfl.ch

Marzo 2007

Cómo compilar una aplicación para TOSSIM

Una vez instalada la distribución de TinyOS que usamos si ejecutamos el siguiente comando:

cd $TOSROOT/apps/

entramos en el directorio de aplicaciones. Si hacemos “ls” podemos ver todas las aplicaciones que tenemos disponibles.
Vamos a compilar la aplicación Blink para ejecutarla con TOSSIM, el simulador de TinyOS, para ello entramos en el directorio Blink:

cd Blink

y para compilar ejecutamos el comando:

make pc

Cuando termina la compilación se crea un directorio llamado “build”, dentro del cual hay otro llamado “pc”. Si entramos en el directorio “pc” podemos ver que se han creado varios ficheros, uno de los cuales es el ejecutable que tendremos que lanzar (“main.exe”).

Antes de ejecutar

Antes de ejecutar cualquier aplicación debemos modificar la variable de entorno DBG. Esta variable tiene que ver con la depuración, dependiendo del valor que tenga se nos mostrarán ciertos mensajes de información cuando simulamos una aplicación. Los valores que puede tomar DBG son los siguientes:

all: Enable all available messages
boot: Simulation boot and StdControl
clock: The hardware clock
task: Task enqueueing/dequeueing/running
sched: The TinyOS scheduler
sensor: Sensor readings
led: Mote leds
crypto: Cryptographic operations (e.g., TinySec)
route: Routing systems
am: Active messages transmission/reception
crc: CRC checks on active messages
packet: Packet-level transmission/reception
encode: Packet encoding/decoding
radio: Low-level radio operations: bits and bytes
logger: Non-volatile storage
adc: The ADC
i2c: The I2C bus
uart: The UART (serial port)
prog: Network reprogramming
sounder: The sounder on the mica sensor board
time: Timers
sim: TOSSIM internals
queue: TOSSIM event queue
simradio: TOSSIM radio models

hardware: TOSSIM hardware abstractions
simmem: TOSSIM memory allocation/deallocation (for finding leaks)
usr1: User output mode 1
usr2: User output mode 2
usr3: User output mode 3
temp: For temporary use

Por defecto el valor de DBG es “all”, si no la modificamos se nos van a mostrar todos los mensajes posibles en la simulación, con lo cual la misma va a tardar muchísimo tiempo. Para modificar el valor de DBG ejecutamos, por ejemplo, el siguiente comando:

export DBG=boot,led

Al ejecutar este comando estamos diciéndole al simulador que durante la ejecución de la aplicación queremos que nos muestre por pantalla sólo los mensajes que tienen que ver con el arranque del nodo y con los leds.

NOTA: cuando programamos una aplicación podemos poner en cualquier sitio de nuestro código sentencias del tipo:

dbg(DBG_USR1, “Entro por aquí”);

Estas sentencias son las que se imprimen por pantalla cuando ejecutamos nuestra aplicación. Los tipos usr1, usr2 y usr3 se reservan a mensajes que el usuario quiera agregar en su programa para luego depurarlo, es decir, si añadimos sentencias en nuestro código como la anterior y luego le damos a DBG el valor “usr1”, vamos a ver sólo los mensajes que hemos puesto nosotros con el tipo usr1.

Cómo ejecutar una aplicación ya compilada

Vamos a continuar con el ejemplo de la aplicación Blink. Supongamos que ya la hemos compilado y que nos encontramos en el directorio /opt/tinyos-1.x/apps/Blink. Para lanzar la aplicación ejecutamos el siguiente comando:

./build/pc/main.exe 1

Con el número que ponemos al final estamos especificando el número de nodos que queremos para nuestra simulación, en el caso de la aplicación Blink solo ponemos uno, pues esta aplicación lo único que hace es encender y apagar el led del nodo, un nodo sería suficiente para ver el comportamiendo de la misma.

Si antes de lanzar la aplicación ponemos “led” como valor de la variable de entorno DBG (export DBG=led), los mensajes que se van a mostrar durante la simulación van a ser los siguientes:

0: LEDS: Red on.
0: LEDS: Red off.
0: LEDS: Red on.
0: LEDS: Red off.
0: LEDS: Red on.
0: LEDS: Red off.

0: LEDS: Red on.
0: LEDS: Red off.
0: LEDS: Red on.
0: LEDS: Red off.
…

NOTA: el número que aparece al principio de cada línea nos indica el número del nodo al que pertenece el mensaje. En este caso, como solo tenemos un nodo, siempre es 0.

Para parar la simulación pulsamos “Ctrl + C”

Opciones de ejecución

La aplicación Blink cambia el estado del led cada segundo. Tiene un temporizador que se dispara cada segundo y cambia el led de encendido a apagado o viceversa. Si hemos ejecutado la aplicación como acabamos de explicar vemos que la simulación es muy rápida. Hay una opción de ejecución (opción “-l”) que nos permite escalar la simulación con respecto al tiempo real, por tanto si ejecutamos el siguiente comando:

./build/pc/main.exe –l=1 1

vamos a lanzar la misma simulación que en el caso anterior pero no va a ir tan rápido, porque le estamos diciendo que 1 segundo en nuestra simulación es 1 segundo real y vamos a ver como ahora sí se produce una transición en el led cada segundo.

Aquí podemos ver listadas todas las opciones que podemos pasar para la ejecución:

```
Usage: ./build/pc/main.exe [options] num_nodes
[options] are:
-h, --help Display this message.
-gui pauses simulation waiting for GUI to connect
-a=<model> specifies ADC model (generic is default)
options: generic random
-b=<sec> motes boot over first <sec> seconds (default: 10)
-ef=<file> use <file> for eeprom; otherwise anonymous file is used
-l=<scale> run sim at <scale> times real time (fp constant)
-r=<model> specifies a radio model (simple is default)
options: simple static lossy
-rf=<file> specifies file input for lossy model (lossy.nss is default)
-s=<num> only boot <num> of nodes
-t=<sec> run simulation for <sec> virtual seconds
num_nodes number of nodes to simulate
```

Otras aplicaciones

Para compilar otras aplicaciones se procede del mismo modo que hemos hecho para Blink, vamos a ver un segundo ejemplo con Surge.

1- Entramos en el directorio de la aplicación Surge y la compilamos:

```
cd $TOSROOT/apps/Surge
make pc
```

2- Damos el valor que nos interese a la variable de entorno DBG, en este caso nos interesa ver los mensajes de arranque y los mensajes de envio/recepción de paquetes, por tanto vamos a darle el valor “boot,am”:

```
export DBG=boot,am
```

3- Ejecutamos la aplicación, en este caso lo vamos a hacer para 5 nodos:

```
./build/pc/main.exe –l=1 5
```

En los directorios de las aplicaciones hay un archivo README en el que se explica qué hace cada una de ellas. Surge manda paquetes entre los nodos de la red, si echamos un vistazo a los mensajes que nos salen podemos ver cómo los nodos envían y reciben paquetes. La red simulada por Surge es además una red MultiHop, es decir, no todos los nodos están conectados con todos los demás, por lo tanto hay veces que para enviar un mensaje de un determinado nodo a otro, si estos no están conectados directamente, el mensaje tiene que dar varios saltos por nodos intermedios hasta llegar al nodo destino.

Algo curioso de esta aplicación es que podemos usar una herramienta de Java también llamada Surge, que consiste en una interfaz gráfica con la que vemos la topología de la red que estamos simulando. Vamos a ver los pasos que tenemos que seguir para hacer esto.

1- Lanzamos la aplicación Surge para, por ejemplo, 5 nodos:

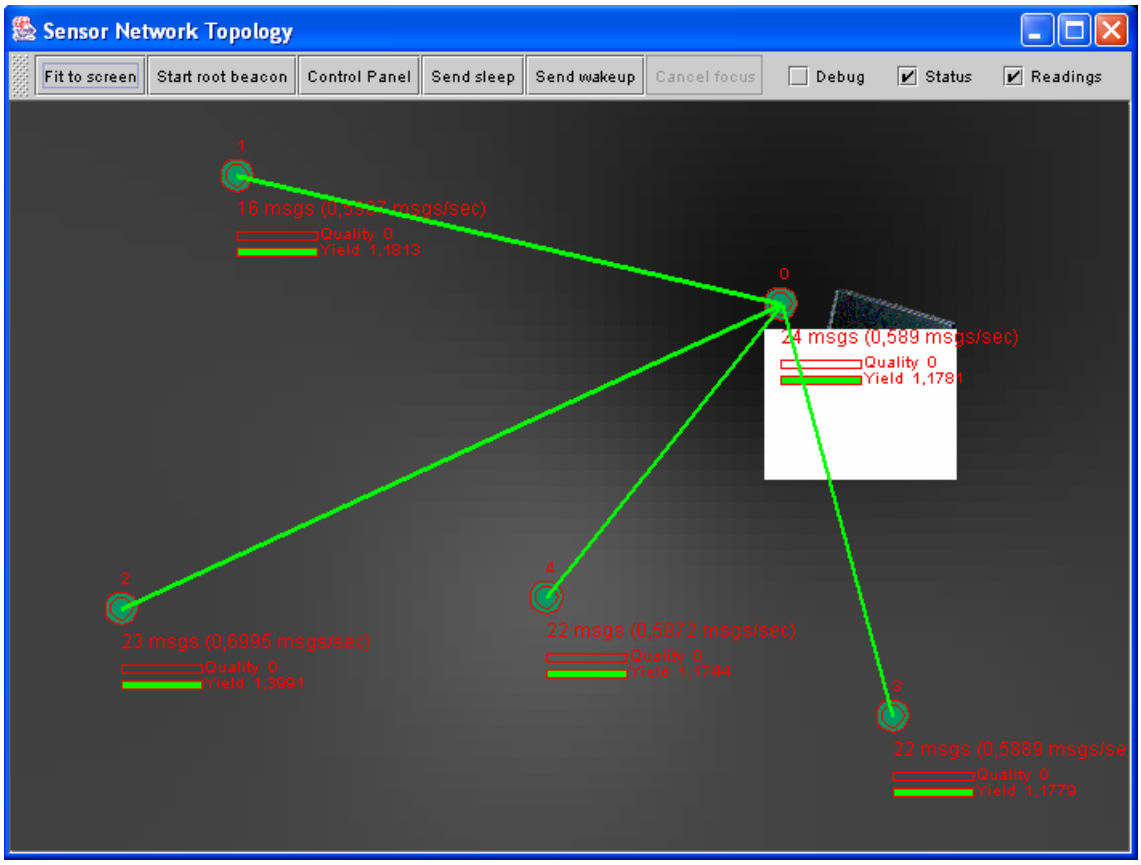
```
./build/pc/main.exe –l=1 5
```

2- Ahora lanzamos la interfaz gráfica, para esto abrimos mejor otra ventana de Cygwin y ejecutamos los siguientes comandos:

```
cd /opt/tinyos-1.x/tools/java/net/tinyos/surge/
make clean
SURGE_PLATFORM=pc make
java net.tinyos.sf.SerialForwarder -comm tossim-serial &
export MOTECOM=sf@localhost:9001
java net.tinyos.surge.MainClass 0x7d
```

Una vez ejecutados esos comandos tiene que aparecer una ventana como la siguiente, en la que podemos ver la topología de la red. En esta imagen podemos ver que el nodo 0 es la estación base y tenemos una topología de estrella, en la que la estación base es el centro y todos los demás nodos están conectados con ella.

NOTA: la interfaz gráfica tarda un poco hasta que detecta la topología de la red y la muestra, paciencia…


Sensor Network Topology
Fit to screen
Start root beacon
Control Panel
Send sleep
Send wakeup
Cancel focus
Debug
Status
Readings
1
16 msgs (0,5867 msgs/sec)
Quality 0
Yield 1,1813
0
24 msgs (0,589 msgs/sec)
Quality 0
Yield 1,1781
2
23 msgs (0,6995 msgs/sec)
Quality 0
Yield 1,2991
4
22 msgs (0,5872 msgs/sec)
Quality 0
Yield 1,1744
3
22 msgs (0,5889 msgs/se
Quality 0
Yield 1,1778