TYBSc CS 2020-2021 USCS607:Wireless Sensor Network

RollNo: 18043 Date: 24/03/2021

**Practical.No:03**

**Aim**:Understanding TOSSIM for – Mote-mote radio communication – Mote-PC serial communication.

**Introduction:**

TOSSIM is a discrete event simulator for TinyOS sensor networks. Instead of compiling a TinyOS application for a mote, users can compile it into the TOSSIM framework, which runs on a PC. This allows users to debug, test, and analyze algorithms in a controlled and repeatable environment. As TOSSIM runs on a PC, users can examine their TinyOS code using debuggers and other development tools. Brief summary of its characteristics:

* **Fidelity:**By default, TOSSIM captures TinyOS’ behavior at a very low level. It simulates the network at the bit level, simulates each individual ADC capture, and every interrupt in the system.
* **Time:**While TOSSIM precisely times interrupts (allowing things like bit-level radio simulation), it does not model execution time. From TOSSIM’s perspective, a piece of code runs instantaneously. Time is kept at a 4MHz granularity (the CPU clock rate of the rene and mica platforms).
* **Models:**TOSSIM itself does not model the real world. Instead, it provides abstractions of certain real-world phenomena (such as bit error). With tools outside the simulation itself, users can then manipulate these abstractions to implement whatever models they want to use.
* **Radio:**TOSSIM does not model radio propagation; instead, it provides a radio abstraction of directed independent bit errors between two nodes. An external program can provide a desired radio model and map it to these bit errors. Having directed bit error rates means that asymmetric links can be easily modeled.
* **Power/Energy:**TOSSIM does not mod–1 power draw or energy consumption. However, it is very simple to add annotations to components that consume power to provide information on when their power states change (e.g., turned on or off). After a simulation is run, a user can apply an energy or power model to these transitions, calculating overall energy consumption.
* **Building:**TOSSIM builds directly from TinyOS code. To simulate a protocol or system, you must write a TinyOS implementation of it.
* **Imperfections:**Although TOSSIM captures TinyOS behavior at a very low level, it makes several simplifying assumptions. This means that it is very possible that code which runs in a simulation might not run on a real mote. For example, in TOSSIM interrupts are non-preemptive (a result of being a discrete event simulator).
* **Networking:** Currently, TOSSIM simulates the 4OKbit RFM mica networking stack, including the MAC, encoding, timing, and synchronous acknowledgements.
* **Authority:** Initial experience from real-world deployments has shown that TinyOS networks have very complex and highly variable behavior. While TOSSIM is useful to get a sense of how algorithms perform in comparison to one another, TOSSIM results shouldn’t be considered authoritative.

**Compiling and Running a Simulation:**

TOSSIM is automatically built when you compile an application. Applications are compiled by entering an application directory (e.g. /apps/Blink) and typing make. Alternatively, when in an application directory, you can type make pc, which will only compile a simulation of the application.

**The TOSSIM executable is named main.exe, and resides in build/pc. It has the following usage:**

Usage: ./build/pc/main.exe [options] num\_nodes

[options] are:

-h, --help Display this message.

-gui pauses simulation waiting for GUI to connect

-a= specifies ADC model (generic is default) options: generic random

-b= motes boot over first seconds (default: 10)

-ef= use for eeprom; otherwise anonymous file is used

-l= run sim at times real time (fp constant)

-r= specifies a radio model (simple is default) options: simple static lossy

-rf= specifies file input for lossy model (lossy.nss is default)

-s= only boot of nodes

-t= run simulation for virtual seconds num\_nodes number of nodes to simulate

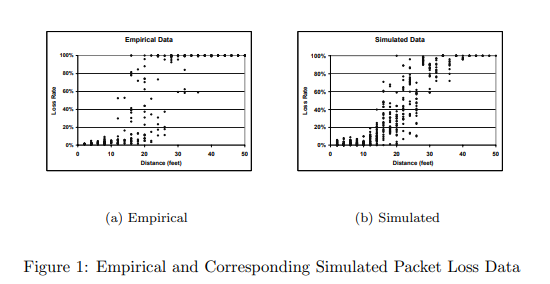
* **dbg**

TOSSIM provides configuration of debugging output at run-time. Much of the TinyOS source contains debugging statements. Each debugging statement is accompanied by one or more modal flags. When the simulator starts, it reads in the DBG environment variable to determine which modes should be enabled. Modes are stored and processed as entries in a bit-mask, so a single output can be enabled for multiple modes, and a user can specify multiple modes to be displayed.

* **Network Monitoring and Packet Injection**

To interact with a simulated network, you must use SerialForwarder, the standard TinyOS interface tool. To work with TOSSIM, SerialForwarder’s input source must be set appropriately. TOSSIM provides two modes: Communication through a serial port to mote 0 and network snooping.

The serial port mode (“tossim-serial” in the “Mote Communications” field or “-comm tossim-serial” at the command line) interacts with mote 0 over its serial port. Programs connecting to SerialForwarder can read messages mote 0 sends to its serial port, and send messages to mote 0 over its serial port.



**Radio Models:**

TOSSIM simulates the TinyOS network at the bit level, using TinyOS component implementations almost identical to the mica 4OKbit REM-based stack. TOSSIM provides two radio models: simple and lossy. The mica2 CC1000-based stack does not currently have a simulation implementation. Using LossyBuilder.

LossyBuilder assumes each mote has a transmission radius of 50 feet. Combined with the bit error rate, this means each mote transmits its signal in a disc of radius 50 feet, with the bit error rate increasing with distance from the center.

* **Bit Errors and Packet Errors**

The formula for calculating packet error rates (Ep) from bit error rates (Eb) for the mica RFM 40Kb stack with SecDed encoding is:

* Ep = 1- (Ss • (Se)d)
* Ss = (1 – Eb)9
* Se = (1 – Eb)8 + (8 • Eb – (1 Eb)12)
* Ep = 1 – ((1 – Eb)9 • ((1 – Eb)8 + (8 – Eb • (1 Eb)12))d)

Where Ss is the start symbol success probability, Se the probability a packet byte is uncorrupted (zero or one bit errors), and d is the number of bytes in the packet.

* **Lossy Model Actuation**

Specifying a loss topology with a file defines a static topology over an entire simulation. There are simulation situations, however, in which changing topologies are needed. TOSSIIVI therefore allows users to modify the loss topology at run-time.

LossyBuilder can read in or generate physical topologies ((x,y) coordinates), and generate loss topologies from physical topologies by sampling from the model of the empirical distribution in Figure 1(a). Its usage is:

usage: java net.tinyos.sim.LossyBuilder [options]

options: -t grid: Topology (grid only and default)

-d : Grid size (m by n) (default: 10 x 10)

-s : Spacing factor (default: 5.0)

-o : Output file

-i : Input file of positions

-p: Generate positions, not error rates

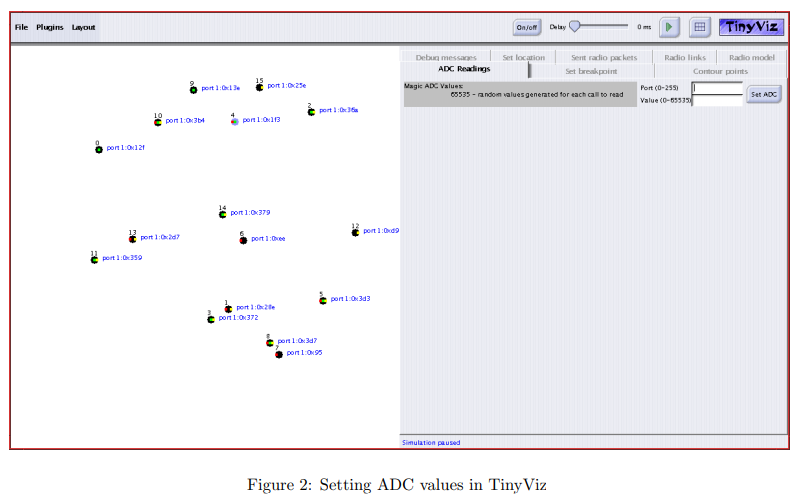
**ADC Models:**

TOSSIM provides two ADC models: random and generic. The model chosen specifies how readings taken from the ADC are generated. Whenever any channel in the ADC is sampled in the random model, it returns a 10-bit random value (the rene and mica ADCs are 10 bits).

**EEPROM:**

TOSSIM models the EEPROM at the line (16-byte block) level. TOSSIM models it with a large, memory mapped file. By default, this file is anonymous, and disappears when a simulation ends.

However, with the -ef option, a user can specify a named file to use; this allows the EEPROM data to persist across multiple simulation invocations.



**TinyViz:**

TinyViz is a Java visualization and actuation environment for TOSSIM. The main TinyViz class is a jar file, tools/java/net/tinyos/sim/tinyviz.jar. TinyViz can be attached to a running simulation. Also, TOSSIM can be made to wait for TinyViz to connect before it starts up, with the GUI flag. This allows users to be sure that TinyViz captures all the events in a given simulation.

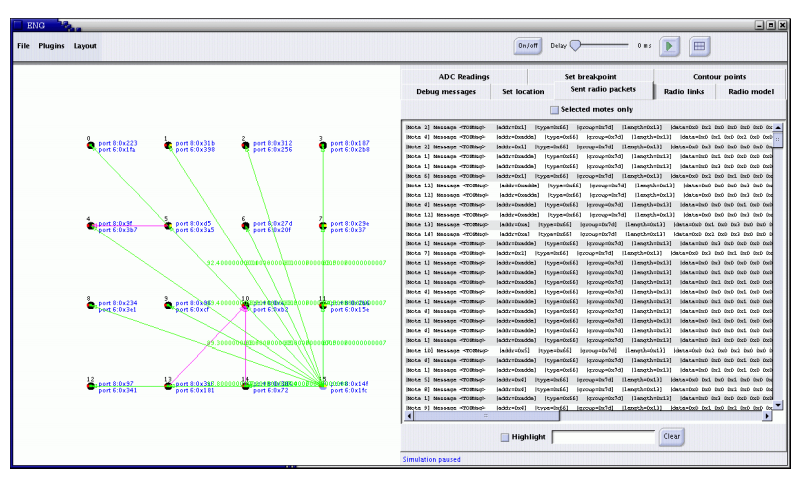
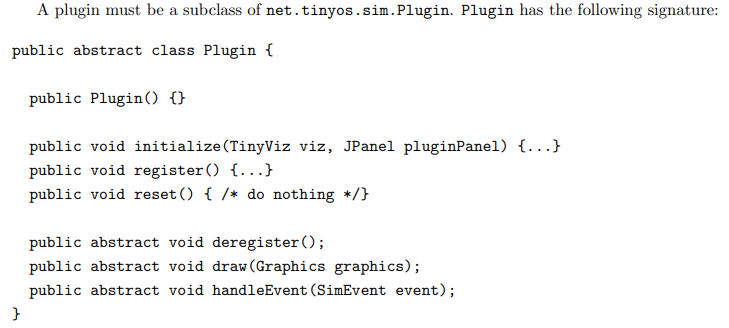


Figure 3: TinyViz connected to TOSSIM running an object tracking application. The right panel shows sent radio packets, the left panel exhibits radio connectivity for mote 15 and network traffic. The green arrows and corresponding labels represent link probabilities for mote 15, and the magenta arrows indicate packet transmission.

**TinyViz Plugins:**

Users can write new plugins, which TinyViz can dynamically load. A simple event bus sits in the center of TinyViz; simulator messages sent to TinyViz appear as events, which any plugin can respond to. For example, when a mote transmits a packet in TOSSIM, the simulator sends a packet send message to TinyViz, which generates a packet send event and broadcasts it on the event bus.

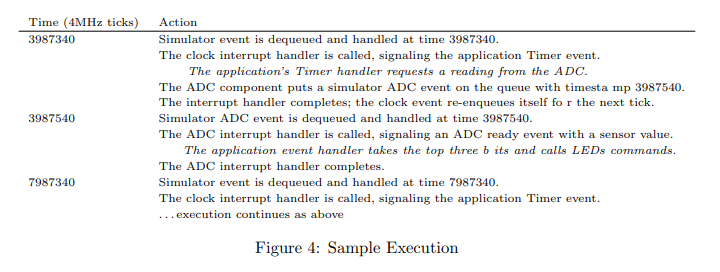


**Using gdb:**

The binary executable produced by typing make PC in any app directory can be debugged using GDB. Developers of TinyOS code can step through programs they have written, debugging deterministic, logical aspects of their code before loading programs onto motes.

**Concurrency Model:**

TOSSIM captures the TinyOS event-driven concurrency model at interrupt and task granularity. For a description of the TinyOS concurrency model, refer to the TinyOS documentation. This document does not describe the model, merely how TOSSIM implements it.

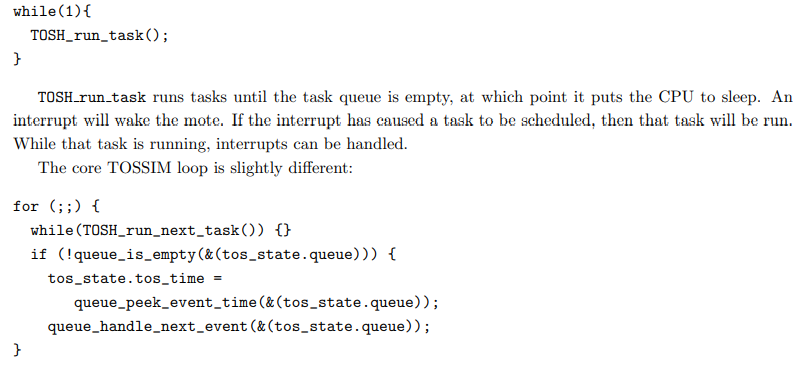


**Internals:**

Currently, TOSSIM compiles by using alternative implementations of some core system components (e.g. Main.nc, HPLUART.nc) and incorporating a few extra files (e.g. event queue.c). The basic TOSSIM structure definitions exist in platform/PC/nido.h. These include the global simulation state (event queue, time, etc.) and individual node state.

**TOSSIM Architecture:**

TOSSIM replaces some TinyOS components, the components that handle interrupts and the Main component.



**TOSSIM Implementations:**

To load a component, the nesC compiler uses a directory search path. The default search path is:

* The application directory
* tos/platform/xxx (the selected platform)
* tos/sensor boards/xxx (the selected sensor board(s))
* tos/system 15
* tos/lib

**Mote-PC serial communication and SerialForwarder (TOS 2.1.1 and later) Packet sources and TestSerial:**

The first step is to check that you are able to get your PC to communicate with a mote. Most motes have a serial port or similar interface. For example, the mica family can directly control a serial port: programming boards basically connect the mote’s serial port pins to the actual serial port on the board. Telos motes also have a serial interface, but it talks to their USB hardware, which is similar in functionality but very different in terms of cables and connectors.

**MOTECOM:**

if you do not pass a -comm parameter, then tools will check the MOTECOM environment variable for a packet source, and if there is no MOTECOM, they default to a SerialForwarder. This means that if you’re always communicating with a mote over your serial port, you can just set MOTECOM and no longer have to specify the -comm parameter.

**BaseStation and net.tinyos.tools. :**

BaseStation is a basic TinyOS utility application. It acts as a bridge between the serial port and radio network. When it receives a packet from the serial port, it transmits it on the radio; when it receives a packet over the radio, it transmits it to the serial port.

**MIG: generating packet objects:**

i.e. Listen program is the most basic way of communicating with the mote; it just prints binary packets to the screen. Obviously, it is not easy to visualize the sensor data using this program.

**SerialForwarder and other packet sources:**

One problem with directly using the serial port is that only one PC program can interact with the mote. Additionally, it requires you to run the application on the PC which is physically connected to the mote.

**Packet Sources:**

In addition to serial ports and Serial Forwarders, the TinyOS messaging library supports a third packet source, motes which are connected to an ethernet port through a Crossbow **MIB**600 ethernet board. This is the full set of packet sources:

|  |  |
| --- | --- |
| **Syntax** | **Source** |
| serial@PORT:SPEED | Serial ports |
| sf@HOST:PORT | SerialForwarder, TMote Connect |
| network@HOST:PORT | MIB 600 |

In the network packet source, the default MIB 600 port is 10002. The Moteiv TMote Connect appliance is a SerialForwarder packet source.

**Sending a packet to the serial port in TinyOS:**

Sending an AM packet to the serial port in TinyOS is very much like sending  it to the radio. A component uses the AMSend interface, calls AMSend.send, and handles AMSend.sendDone. The serial stack will send it over the serial port regardless of the AM address specified.

**TOSThreads Example:**

For TOSThreads applications, the TOSThreads library provides both nesC and C APIs for serial communication.

nesC APIs are in tos/lib/tosthreads/system/: BlockingSerialActiveMessageC component provides the BlockingStdControl interface to turn ON/OFF serial communication.