TinyOS Simulator: TOSSIM

EE 693

Programming for Mobile Devices
Simulation for Sensor Networks

Sensor networks are: new, event-driven, large, deployed in remote locations

Simulation provides:
- Controlled, Reproducible testing environment
- Cost-effective alternative
- Means to explore and improve design space
Existing Simulation Tools

ns-2

- Packet Level network abstraction
- Does not model application behavior (not complete and inappropriate for sensor networks)
TOSSIM

- TinyOS mote simulator
- Scales to thousands of nodes
- Compiles directly from TinyOS source
- Simulates network at bit level
- Replaces hardware with software components
- Hardware interrupts are modeled as simulator events.
Figure 3: TOSSIM Architecture: Frames, Events, Models, Components, and Services
Goals of TOSSIM

- **Scalability**: able to handle large networks
- **Completeness**: capture complete system behavior
- **Fidelity**: Capture behavior at a fine-grain
- **Bridging**: the gap between algorithms and implementation
Bridging

- ‘make sim’ instead of ‘make mica’
- Instrumented nesC compiler
- Compile application code to TOSSIM or hardware platform as needed
- No change to application required
- Tested code can be deployed right away
Scalability

- Individual mote resources very small
- Static component memory model simplifies state management.
Fidelity

- Emulating hardware at component level
- Bit-level simulation: capturing network at high fidelity
- TOSSIM helped in debugging some TinyOS network stack problems
- These problems were unnoticed during testbed deployment
Simulation Time

- Time at mote instruction cycle frequency
- $4 \text{ MHz} = 4 \times 10^6 \text{ ticks/second}$
- e.g. 400 ticks between 10Kb radio interrupts
- Chosen as minimum value that allows accurate radio and system clock modeling
Radio Models

- Radio models external to TOSSIM
- Models network as a directed graph of bit error probabilities.
- Built-in models: “Simple”, “Static”, “Space”
Communication Services

- Allows user to drive, monitor, debug simulation
- Command/event interface
- Eg. TinyViz (only for TinyOS 1.x)
Network Simulation

- Most complex and fine grained component of TOSSIM
- Almost perfect simulation of TinyOS networking stack at bit level
- Combination of bit sampling and bit rate changes used for capturing the entire stack.
Evaluation

- Fidelity
  - radio noise
  - packet interactions
  - subtle race conditions

- Completeness
  - Surge
    http://firebug.sourceforge.net/surgepics.html
  - TinyDB
    http://www.tinyos.net/tinyos-1.x/doc/tutorial/tinydb.html
Figure 10: Send queue length over time for two motes in the Surge network: send queues often overflow.

Figure 11: Send queue length for two motes in the Surge network over time, with cycle and duplicate suppression fixes: send queues do not overflow.
Figure 12: Scalability of TOSSIM on three applications as a function of the number of motes. Each simulation ran for ten virtual seconds.
Possible Enhancements

- CPU modeling: run-instantly model
- Energy modeling
- Supporting thread based execution models
- Supporting heterogeneous platforms
- Allowing Different mote applications to run at once
Summary

- TOSSIM simulates TinyOS applications for sensor networks
- The same code can be used both for simulation and testbed deployment
- It is scalable and extensible
- Does not address energy profiling
- Applicable only in TinyOS platform
Features

- Simulates a MicaZ mote
  - ATmega128L (128KB ROM, 4KB RAM)
  - CC2420
- Uses CPM to model the radio noise
- Supports two programming interfaces:
  - Python
  - C++
## Anatomy

### TOSSIM

- tos/lib/tossim
- tos/chips/atm128/sim
- tos/chips/atm128/pins/sim
- tos/chips/atm128/timer/sim
- tos/chips/atm128/spi/sim
- tos/platforms/mica/sim
- tos/platforms/micaz/sim
- tos/platforms/micaz/chips/cc2420/sim

### Application

- Makefile
- *.nc
- *.h

### Simulation Driver

- *.py | *.cc
Quick Review

Application

NesC

Simulation

Glue

Python

C++
Building Process (Python)

$ make micaz sim

1. Generate an XML schema
2. Compile the application
3. Compile the Python support
4. Build a share object
5. Copying the Python support

$ ./sim.py