


TinyOS and TOSSIM

Mihai Fonoage
Dept. of Computer Science and Engineering
Florida Atlantic University


mfonoage@fau.edu mmarta@fau.edu



Outline

- TinyOS Introduction
- Network Communication
- Sensor Data Acquisition
- TOSSIM
- Example
- Summary
- References


2



TinyOS Introduction

- What is TinyOS
 - A component based Operating System
 - A platform targeting Wireless Sensor Network
 - Open Source development environment
 - Set of cooperating tasks and processes
 - Written in nesC programming language


3



TinyOS Introduction (cont.)

- What is nesC?
 - Programming Language for structured component-based applications
 - Intended for embedded systems
 - Has a C-like syntax
 - Used for building components employed in concurrent systems


4



TinyOS Introduction (cont.)

- Application Model
 - Build out of components
 - Components are specified by an interface
 - Provides means of wiring components together
 - Components are statically wired together by means of their interfaces
 - Compiler optimization

5



TinyOS Introduction (cont.)

- Components Specification
 - Provides and uses interfaces, commands and events
 - Interfaces -> Point of access to the component
 - Commands -> Set of functions declared by an interface
 - Implemented by the interface provider
 - Events -> Set of functions declared by an interface
 - Implemented by the interface user

6

TinyOS Introduction (cont.)

- Components Hierarchy
 - **Commands**
 - Flow downwards
 - Control returns to caller
 - **Events**
 - Flow upwards
 - Control returns to signaller
 - Events can call Commands, but commands should not signal events (prevents loops)
 - Post a task instead

7

TinyOS Introduction (cont.)

- Components Implementation
 - Two type of components in nesC
 - **Modules** – contain behaviour
 - Provide application code
 - Implement one or more interfaces
 - **Configurations** – contain wires
 - Used to assemble components together
 - Top-level configuration view

8

TinyOS Introduction (cont.)

- Concurrency Model
 - One program is executed
 - Two threads of execution
 - **Tasks**
 - Time flexible; Longer background processing jobs; Atomic with respect to other tasks (single threaded); Preempted by events.
 - **Events**
 - Time critical; Shorter duration (can use tasks here); Interrupts task; LIFO semantics (no priority between events).
 - Hardware event handlers are executed in response to a hardware interrupt and also runs to completion

9

TinyOS Introduction (cont.)

- Anything executed as a direct result of a hardware interrupt must be declared **async**
 - E.g., "**async command** result_t cmdName(...)"
- Variables shared across sync and async boundaries should be protected by **atomic{...}**
 - Can suppress warning if you put **norace** in front of variable declaration
 - Use with extreme caution

10

Network Communication

- Inter-node communication
 - **Sender:**
 - Fill message buffer with data
 - Specify Recipients
 - Pass buffer to OS
 - Determine when message buffer can be reused
 - **Receiver:**
 - OS Buffers incoming message in a free buffer
 - Signal application with new message
 - OS obtains free buffer to store next message

11

Network Communication (cont.)

- Group Ids and Addresses
 - Group Ids create a virtual network
 - Group ID is an 8 bit value specified in <tos>/apps/Makerules
 - The address is a 16-bit value specified by the make command
 - make mica install.<id>

12

Network Communication (cont.)

Group Ids and Addresses

- Reserved addresses:
 - 0x007E – UART (TOS_UART_ADDR)
 - Send message to serial port
 - 0xFFFF – Broadcast (TOS_BCAST_ADDR)
 - Broadcast message to all nodes
- Local address:
 - TOS_LOCAL_ADDRESS
 - Used to specify usually the local source address

13

Network Communication (cont.)

TOS Active Messages

- TinyOS follows the Active Message (AM) model => <tos>/types/AM.h
- Message is "active" because it contains the handler ID
- TOSH_DATA_LENGTH = 29 bytes
 - Can change via MSG_SIZE = n in Makefile (Max 36)

```
typedef struct TOS_Msg {
    // the following are transmitted
    uint16_t addr;
    uint8_t type;
    int8_t group;
    uint8_t length;
    int8_t data[TOSH_DATA_LENGTH];
    uint16_t crc;
    // the following are not transmitted
    uint16_t strength;
    uint8_t ack;
    uint16_t time;
    uint8_t sendSecurityMode;
    uint8_t receiveSecurityMode;
} TOS_Msg;
```

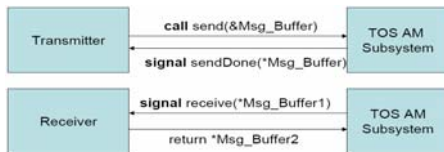
Header (5) Payload (29) CRC

14

Network Communication (cont.)

Message Buffer

- Follows an alternative ownership protocol
 - Avoid expensive memory management



15

Sensor Data Acquisition

Obtaining Sensor Data

- Each sensor has a component that provides one or more **ADC** interfaces
 - MTS300CA:
 - Components in <tos>/sensorboards
 - Include in Makefile: SENSORBOARD=micasb

- I. ADC.dataReady() event causes the sensor reading to be stored in a circular buffer.
- II. When the appropriate number of samples have been collected the Sensing.done() event is signalled

```
includes ADC;
includes sensorboard; // this defines the user names for the ports

interface ADC {
    async command result_t getData();
    async command result_t getContinuousData();
    async event result_t dataReady(uint16_t data);
}
```

16

Sensor Data Acquisition (cont.)

Sensor Components

- Sensor components usually provide:
 - StdControl
 - Initialize before taking measurements
 - GenericComm
 - Initializing it turns on the power
 - LedsC

```
module SenseLightToLogM {
    provides interface StdControl;
    uses {
        interface StdControl as PhotoControl;
    }
}

Implementation { ...
    command result_t StdControl.init() {
        return rcombine(call PhotoControl.init(),
            call Leds.init());
    }
    command result_t StdControl.start() {
        return call PhotoControl.start();
    }
    command result_t StdControl.stop() {
        return call PhotoControl.stop();
    }
    ...
}
```

17

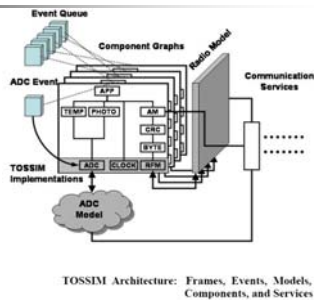
TOSSIM

What is TOSSIM?

- Event Simulator for TinyOS sensor networks
- Runs on a PC
- Can simulate thousands of nodes
- Compiles directly from TinyOS source
- Simulates network at a bit level
- Replaces hardware with software components
- Hardware interrupts <-> simulator events 18

TOSSIM (cont.)

- Generates discrete-event simulations directly from TinyOS component graphs
- Runs the same code that runs on sensor network hardware
- Translates hardware interrupts into discrete simulator events
- The simulator event queue delivers the interrupts that drive the execution of a TinyOS application



19

TOSSIM (cont.)

Goals of TOSSIM

- Scalability**
 - Individual mote resources are very small
 - Static component memory model simplifies state management
- Completeness**
 - Capture complete system behaviour
- Fidelity**
 - Emulating hardware at component level
 - Bit-level simulation : capturing network at high fidelity
- Bridging**
 - Compile application code to TOSSIM or hardware platform as needed
 - TOSSIM is automatically built when we compile an application
 - No change to application required

20

TOSSIM (cont.)

Network Monitoring and Packet Injection

- SerialForwarder -> Interface tool
- 2 modes of communication:
 - Serial port to mote 0
 - Network snooping
 - Snooping mode outputs every message *sent*, so it does not consider loss
 - Programs connecting will hear packets that might not arrive successfully at any mote.
- MIG -> Message Interface Generator
 - Parses C structures for TinyOS packets and builds a Java class with getters for each field

21

TOSSIM (cont.)

Radio Models

- Provides two radio models:
 - Simple
 - Places all nodes in a single cell
 - Useful for testing: Single-hop algorithms & TinyOS components for correctness
 - Lossy
 - Places the node in a directed graph
 - LossyBuilder – Java tool that generates loss rates

22

TOSSIM (cont.)

ADC Models

- Two ADC models:
 - Random* and *Generic* -> both specify how readings taken from ADC are generated
- EEPROM
 - Modeled at the line (16-byte block) level
 - Modeled with a large, memory-mapped file
- Debugging
 - Use *gdb*

23

TOSSIM (cont.)

TinyViz

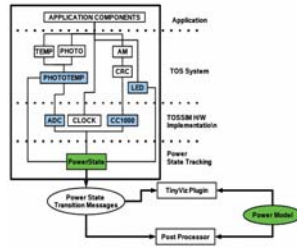
- Java visualization and actuation environment for TOSSIM
- Framework in which plugins can provide functionality
 - Publishes TOSSIM events to loaded plugins
 - i.e. A Network Plugin can visualize network traffic as motes receive messages

24

TOSSIM (cont.)

• PowerTOSSIM

- Extensions to TOSSIM to include energy consumption
- Add a module to keep track of power state
- Modifications to other modules to report transitions
- CPU energy usage -> estimate number of cycles in AVR (Advanced Virtual RISC)
- Generate traces that will be processed later



TOSSIM (cont.)

• Mica2 Power Model

Mode	Current	Mode	Current
CPU		Radio	
Active	8.0 mA	Rx	7.03 mA
Idle	3.2 mA	Tx (power = 00)	3.72 mA
ADC Noise Reduction	1.0 mA	Tx (power = 01)	5.21 mA
Power-down	103 µA	Tx (power = 03)	5.37 mA
Power-save	110 µA	Tx (power = 06)	6.47 mA
Standby	216 µA	Tx (power = 09)	7.05 mA
Extended Standby	223 µA	Tx (power = 0F)	8.47 mA
Internal Oscillator	0.93 mA	Tx (power = 60)	11.57 mA
Leds		Tx (power = 80)	13.77 mA
Mica2 sensorboard	0.7 mA	Tx (power = C0)	17.37 mA
EEPROM		Tx (power = FF)	21.48 mA
Read	6.2 mA		
Read time	565 µs		
Write	18.4 mA		
Write time	12.9 ms		

TOSSIM (cont.)

• Other Sensor Network Simulators

- JiST / SWANS
 - High-performance discrete event simulation engine (JiST) and a scalable wireless network simulator (SWANS) built atop the JiST platform.
 - Small memory footprint and fast event execution
 - Simulate larger networks: 1,000,000 nodes simulated with a 933 MB memory footprint
- ns-2
 - Simulates networks at the packet level
 - Does not model application behaviour (not complete and inappropriate for sensor networks)
- Other: GloMoSim, SensorSim, EmStar, TOSSF, Proteus, J-Sim, Shawn

27

Example - Blink

```

module Blink {
    provide {
        interface StdControl;
    }

    uses {
        interface Timer;
        interface Leds;
    }

    implementation {
        components Main, BlinkM, SingleTimer, LedsC;
        Main.StdControl -> SingleTimer.StdControl;
        Main.StdControl -> BlinkM.StdControl;
        BlinkM.Timer -> SingleTimer.Timer;
        BlinkM.Leds -> LedsC;
    }

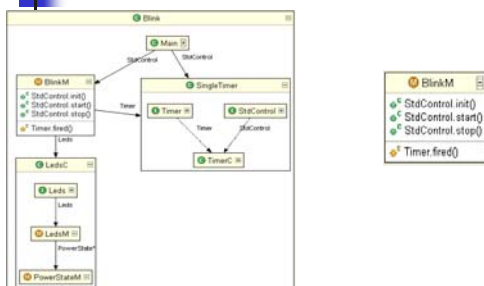
    command result_t StdControl.init() {
        call Leds.init();
        return SUCCESS;
    }

    command result_t StdControl.start() {
        // Start a repeating timer that fires every 1000ms
        @log(DBG_WARN, "StdControl started the timer!!!\n");
        return call Timer.start(TIMER_REPEAT, 1000);
    }

    command result_t StdControl.stop() {
        return call Timer.stop();
    }

    event result_t Timer.fired() {
        @log(DBG_WARN, "Timer fired!!!\n");
        call Leds.toggle();
        return SUCCESS;
    }
}
    
```

Example - Blink



29

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

30



References

- TinyOS -- <http://www.tinyos.net/tinyos-1.x/doc/tutorial/index.html>
- *"TinyOS Tutorial"*, Chien-Liang Fok
- *"nesC"*, by Prof. Chenyang Lu
- *"TOSSIM A Simulator for TinyOS"*, by Bhavana, presented at SenSys 2003
- *"TOSSIM: A Simulator for TinyOS Networks"*, Philip Levis and Nelson Lee
- *"TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications"*, Philip Levis, Nelson Lee, Matt Welsh, and David Culler

31



References

- PowerTOSSIM -- <http://www.eecs.harvard.edu/~shnayder/ptossim/>
- JIST / SWANS -- <http://jist.ece.cornell.edu/>
- *"Simulating the Power Consumption of Large-Scale Sensor Network Applications"*, Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh
- *"Energy Consumption Issues in Sensor Networks"*, Cintia B. Margi