



# CS649

## Sensor Networks

### Lectures 30: WSN Simulation

Andreas Terzis

<http://hinrg.cs.jhu.edu/wsn06/>

# Overview

- The need for simulation
- Different simulation axes
  - High fidelity simulation of TinyOS Apps (TOSSIM)
  - Power consumption simulation
  - Simulation of hierarchical applications and emulation (EmTOS)



# **TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications**

P. Levis, N. Lee, M. Welsh, and D. Culler  
UC Berkeley  
Presented at SenSys 2003

# Outline

- The need for simulation
- TOSSIM Components
- Execution Model
- Hardware Emulation
- Radio Model
- Evaluation
- Discussion

# The need for WSN simulators

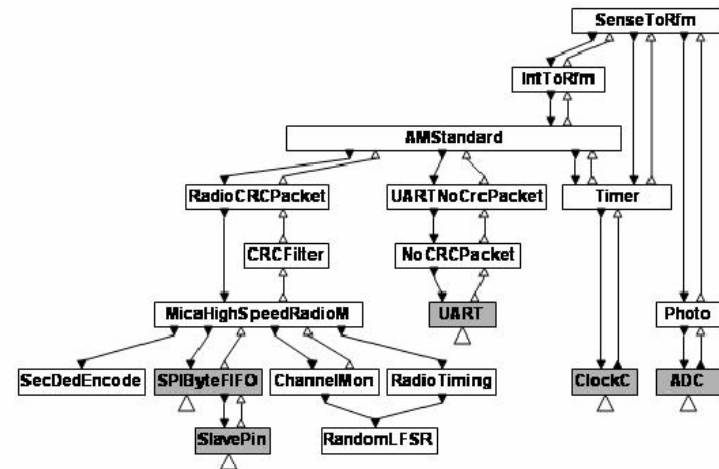
- WSNs are inherently large scale distributed applications
- Large cost of deploying a network for testing
  - Potentially impossible
- Lowers the barrier for experimentation and research
- Parallel to experience from other disciplines
  - ns-2 experience in networking

# Simulator Requirements

- Different levels of simulation
  - Algorithms vs. hardware-level simulation
  - What is the right model?
- TOSSIM requirements
  - Scalability
  - Completeness
  - Fidelity
  - Bridging (same code for real-life app and simulation)

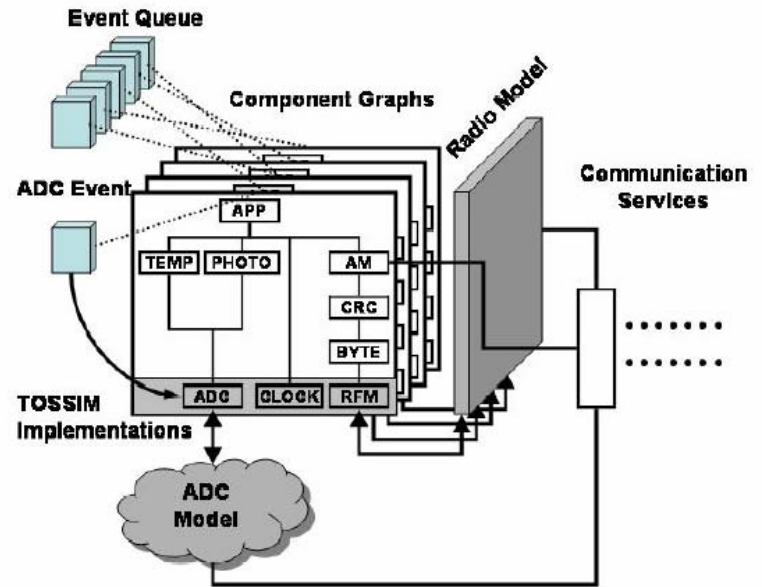
# TinyOS Refresher

- TinyOS characteristics
  - Component-based system
  - nesC
  - Component characteristics
    - Private storage
    - Commands and events
    - Tasks
  - Scheduling
  - Integration with hardware
- Challenges of writing TOS apps
  - Each component simple -> many components for complex apps->large number of interactions



# TOSSIM Overview

- TOSSIM Components
  - Compilation to sim code
  - Discrete event queue
  - Hardware abstraction components
  - Radio Model
  - External communications



# Compiler Support

- nesC compiler can compile TOS apps to simulation code
- How?
  - Each node runs the same code
    - Code path is the same, need multiple copies of state variables
    - Components variables are replaced with an array, one copy for each simulated mote.
  - Plus additional plumbing for discrete event queue, network model and re-implementation of HW components

nesC TinyOS Code	Mote C Code	TOSSIM C Code
<pre>result_t StdControl.init(){   state = 0;   return SUCCESS; }</pre>	<pre>result_t Counter\$StdControl\$init(void){   Counter\$state = 0;   return SUCCESS; }</pre>	<pre>result_t Counter\$StdControl\$init(void){   Counter\$state[tos_state.current_node] = 0;   return SUCCESS; }</pre>

# Execution Model

- Core simulator loops de-queues and executes next event from the event queue
  - HW Interrupts are modeled by simulator events
- Event is delivered to component that simulated HW component
- HW component posts events, commands to other components on mote it's running
- All tasks from node's task queue are executed until completion
  - Interrupt handlers cannot preempt tasks
  - Events, commands and tasks run instantly
  - Implications?

# Hardware Emulation

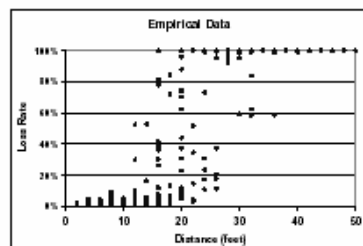
- Simulated HW components
  - ADC
  - Clock
  - Flash
  - Radio

# Radio Model (I)

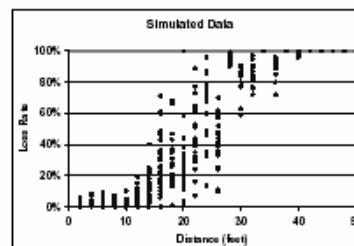
- Model is a directed graph with bit error probabilities
  - Edge  $(u,v,p)$ : Prob[bit sent by  $u$  received at  $v$ ] =  $p$
  - Bit errors are independent, asymmetric links are allowed
- Simulator delivers bit to all nodes within communication range of the sender
- Bit level simulation can provide a wide range of real-life scenarios (hidden-terminal, loss of start symbol, loss of ACK)

# Radio Model (II)

- Simulating Radio Noise
  - Empirical model of packet loss as a function of distance
  - Derive Gaussian packet loss pdf for each distance
  - Per packet loss rate for node pair is determined by sampling the Gaussian distribution
  - Translate packet loss rate to bit loss rate



(a) Empirical



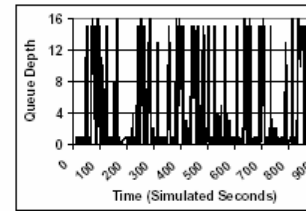
(b) Simulated

# Communication Services

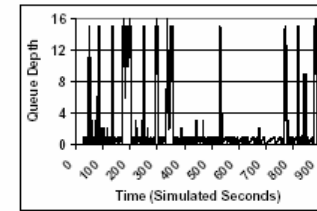
- Simulator can interact with external apps through socket interface
  - Sensing environment model
  - Visualization (TinyViz)

# Evaluation

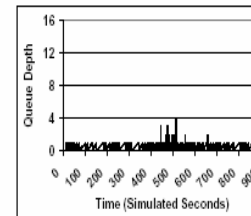
- Simulator was used to detect bugs in *Surge* application
- Noted the large number of packets were not delivered to the root
  - Monitoring showed that node send queue was overflowing
  - First bug: message received but ack lost. Surge would retx
  - Second bug: cycles (not including the origin)



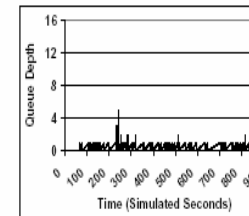
(a) 14 feet from base



(b) 28 feet from base



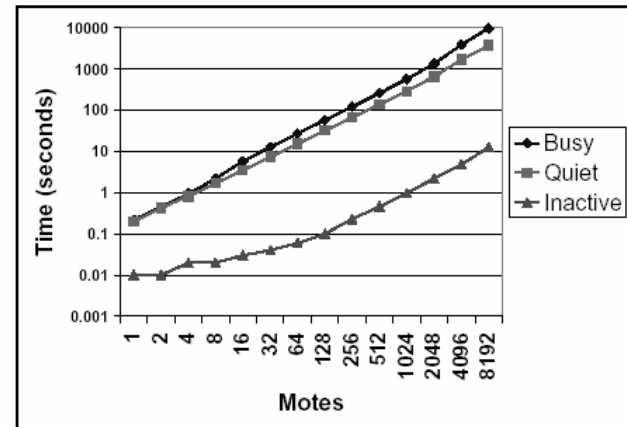
(a) 14 feet from base



(b) 28 feet from base

# Scalability

- Simulated three application types with varying level of network activity
- Performance Metrics
  - Run for ten simulation seconds
  - Memory footprint
    - 10,000 nodes → 4MB
  - Execution Time (10k nodes)
    - Network inactive app: 12.5 sec
    - Network-intensive app: 2.75 hours



# Discussion

- Good Points
  - Single code image for simulation, deployment
  - Scalable
- Bad Points
  - Bit-level radio simulation (can also be good)
  - Single application
  - Difficult to change/extend hardware model
  - Instant execution of tasks
  - It's still nesC 😊



# Simulating the Power Consumption of Large-Scale Sensor Network Applications

V. Shnayder, M. Hempstead, B.Chen, G.W.  
Allen, and M. Welsh

Harvard University

Presented at SenSys 2004

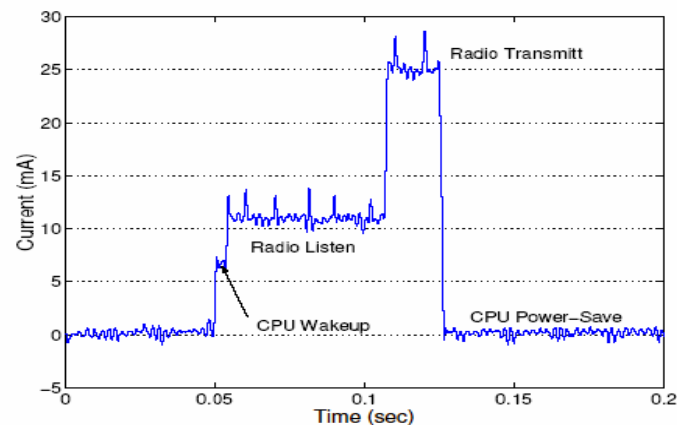
# Motivation

- Power consumption is major concern in WSNs
- Difficult to measure consumption in deployed networks
- Existing simulators
  - Do not simulate power consumption
  - Detailed simulation → Non scalable
- Goal: Extend TOSSIM to simulate power consumption

# Hardware Power Model

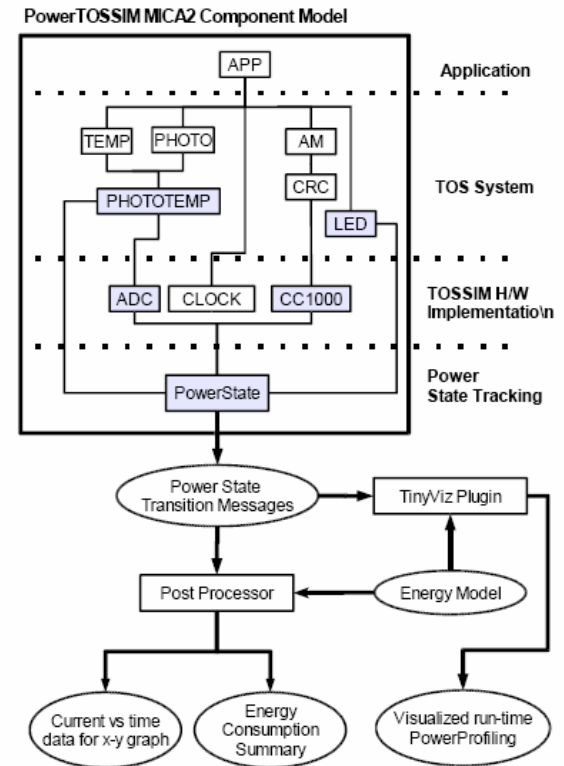
- To simulate power consumption need to create power consumption model (Mica2)
- Approach: Use benchmarks to measure power consumption of actual mote

Mode	Current	Mode	Current
<b>CPU</b>		<b>Radio</b>	
Active	8.0 mA	Rx	7.0 mA
Idle	3.2 mA	Tx (-20 dBm)	3.7 mA
ADC Noise Reduce	1.0 mA	Tx (-19 dBm)	5.2 mA
Power-down	103 $\mu$ A	Tx (-15 dBm)	5.4 mA
Power-save	110 $\mu$ A	Tx (-8 dBm)	6.5 mA
Standby	216 $\mu$ A	Tx (-5 dBm)	7.1 mA
Extended Standby	223 $\mu$ A	Tx (0 dBm)	8.5 mA
Internal Oscillator	0.93 mA	Tx (+4 dBm)	11.6 mA
LEDs	2.2 mA	Tx (+6 dBm)	13.8 mA
Sensor board	0.7 mA	Tx (+8 dBm)	17.4 mA
EEPROM access		Tx (+10 dBm)	21.5 mA
Read	6.2 mA		
Read Time	565 $\mu$ s		
Write	18.4 mA		
Write Time	12.9 ms		



# PowerTOSSIM Architecture

- Power Transition Messages are used to track the state of each HW component
- New PowerState component
- CPU consumption measured differently
- Derived energy model is used to calculate power consumption
- Processing can be done either off-line or through TinyViz



# CPU Profiling

- Calculating PCU power consumption is challenging
  - TOSSIM compiles code into native PC binary
  - All tasks run instantaneously
- General idea
  - Measure the amount of time CPU spends in each power state

$$E_{CPU} = \sum_i P_{s_i} t_{s_i}$$

- Instrument code to obtain execution count of each *basic block*
- Map each basic block to assembly instructions in AVR binary
- Determine number of CPU cycles for each basic block
- Combine execution counts with corresponding cycle counts to obtain time CPU is active

# Profiling example

```
App Code  
if(x>0) {  
    t = x+42;  
    v = t / pi;  
} else {  
    v = -1;  
}
```

CIL  
→

```
Transformed Code  
bb[mote][1]++;  
if(x>0) {  
    bb[mote][2]++;  
    t = x+42;  
    v = t / pi;  
} else {  
    bb[mote][3]++;  
    v = -1;  
}
```

↓ Compile

```
Mote Binary  
if(x>0) { → 2 cycles  
    t = x+42; → 21 cycles  
    v = t / pi;  
} else {  
    v = -1; → 1 cycle  
}
```

Disassemble  
and analyze  
→

Basic Block	Cycles
1	2
2	21
3	1

# Method Accuracy

- Compared calculated CPU cycles to instruction-level simulator
  - Micro-benchmarks
- Compared calculated power consumption against actual implementation
  - TinyOS applications

Task	PowerTOSSIM	atemu	error (%)
hash(char[27])	3002	2956	1.5
hash(char[404])	45415	40279	11.3
qsort(int[100])	69581	92598	-33.0
encrypt(char[100])	103756	106096	-2.3
decrypt(char[100])	105075	107890	-2.7

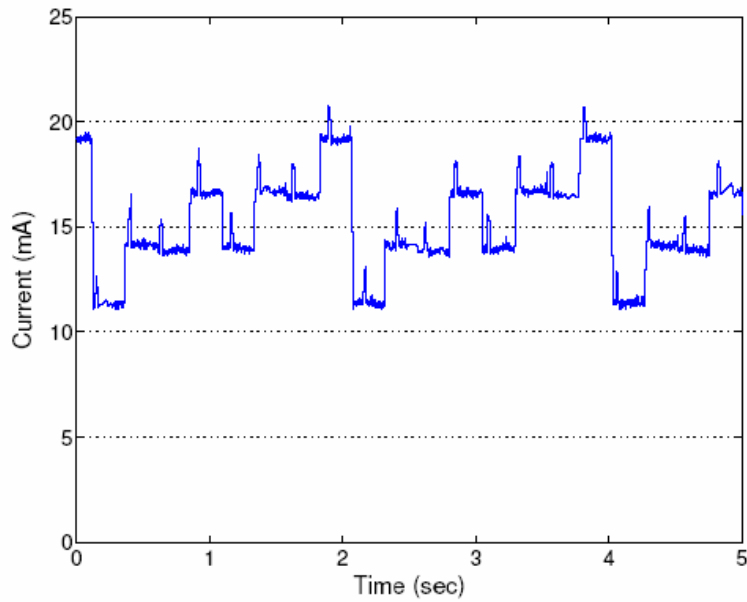
Benchmark	Simulated	Measured	Error (%)
Beacon	92.93	106.73	-12.9
Blink	940.26	931.72	0.85
BlinkTask	940.28	917.90	2.5
CntToLeds	1336.49	1330.00	0.45
CntToLedsAndRfm	2620.37	2562.00	2.3
CntToRfm	2028.09	1985.00	2.1
Oscilloscope	867.94	801.60	8.3
OscilloscopeRF	2136.45	2021.90	5.7
Sense	865.59	900.72	-3.8
SenseLightToLog	2133.89	2005.26	6.4
SenseTask	865.62	944.74	-8.3
SenseToLeds	868.70	977.73	-11.1
SenseToRfm	2152.27	2059.16	4.5
<b>Average</b>			<b>4.7</b>
TinyDB (idle)	2001.31	2275.55	-12.1
TinyDB (select light)	2144.86	2465.30	-13.0
Surge	2089.09	2028.40	3.0
<b>Average</b>			<b>9.5</b>

# Evaluation

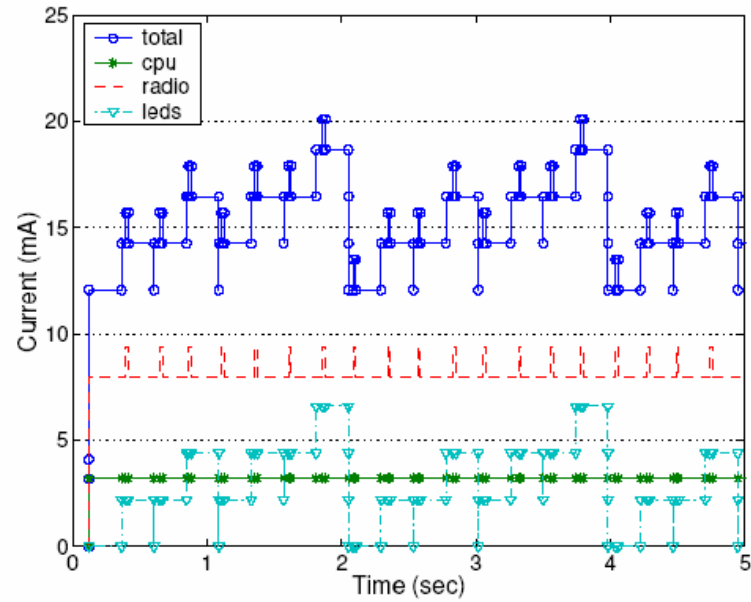
- Simulated a number of TinyOS applications
- Findings
  - Idle CPU consumes most energy
  - LEDs are expensive

Application	CPU idle	CPU active	Radio	Leds	Sensor Board	EEPROM	Total
Beacon	35.86	0.58	47.68	8.81	0.00	0.00	92.93
Blink	742.50	0.25	0.00	197.52	0.00	0.00	940.26
BlinkTask	742.50	0.27	0.00	197.52	0.00	0.00	940.28
CntToLeds	743.72	0.57	0.00	592.20	0.00	0.00	1336.49
CntToLedsAndRfm	741.90	1.61	1284.65	592.20	0.00	0.00	2620.37
CntToRfm	741.90	1.54	1284.65	0.00	0.00	0.00	2028.09
Oscilloscope	742.65	1.46	0.00	0.00	123.82	0.00	867.94
OscilloscopeRF	741.90	1.85	1268.76	0.00	123.95	0.00	2136.45
Sense	742.21	0.38	0.00	0.00	123.00	0.00	865.59
SenseLightToLog	741.90	0.81	1262.95	0.00	123.95	4.28	2133.89
SenseTask	742.21	0.42	0.00	0.00	123.00	0.00	865.62
SenseToLeds	743.72	0.73	0.00	0.00	124.25	0.00	868.70
SenseToRfm	741.90	1.77	1284.65	0.00	123.95	0.00	2152.27
TinyDB (idle)	693.29	10.41	1181.78	0.00	115.83	0.00	2001.31
TinyDBApp (select)	742.85	11.20	1266.70	0.00	124.11	0.00	2144.86
Surge	727.28	1.50	1239.02	0.00	121.30	0.00	2089.09

# Current Traces



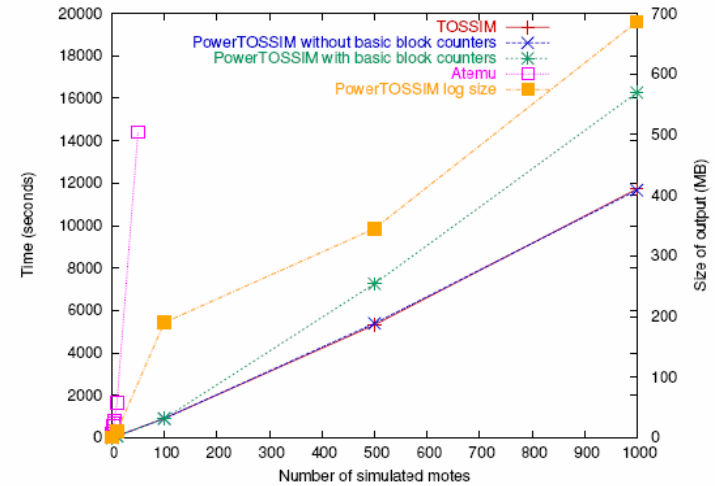
(a) Measured



(b) Simulated

# Scalability

- Compare PowerTOSSIM, TOSSIM and instruction-level simulator



# Case Study: ESS Development

- Development process
  - Use of simulation, emulation, and real modes
  - Real mode used to validate vs. Emulation mode
- Analysis and performance measurement
  - Real mode packet traces with precise timing
  - Offline analysis of collisions and retx
- In deployment
  - EmTOS integrates Multihop tree-building and transport protocol with microserver software
  - EmStar provides visibility and robustness for microserver code
  - Live topology data returned via transport layer piped to visualization tools

