

An Analysis of Programming and Simulations in Sensor Networks

Tanveer Zia, Albert Zomaya
School of Information Technologies
University of Sydney

Madsen Building F09, Camperdown NSW 2006
Email: {tanzia, zomaya}@it.usyd.edu.au

ABSTRACT

Recent advancements in micro electro mechanical systems (MEMS) and wireless communications have made possible the developments of low cost sensor networks. Applications of sensor networks are broad from domestic applications to battlefield. Due to sensors nature of limited energy, limited memory and very small computing power there are significant challenges in its design, programming and simulations. This paper explores the existing programming and simulation environments in sensor networks and demonstrates Berkeley's TinyOS and TOSSIMS outcomes to address these challenges. We have emphasised that due to resource starved nature of sensor nodes it is vital to reduce the programming code, improve the response time and minimise the energy consumption.

1. INTRODUCTION

To develop an understanding of sensor network simulations, this paper investigates the programming and simulation challenges in tiny sensors.

Developments in micro electro mechanical systems (MEMS) and wireless networks are opening a new domain in networking history. Sensors; called "smart dust" are low cost small tiny devices with limited coverage, low power, smaller memory sizes and low bandwidth, will play a key role in collecting and disseminating data from the fields where ordinary networks are unreachable for various environmental and strategical reasons.

Research at UC Berkley and Intel's "Lablet" has produced thousands of smart sensor "motes" to observe everything in their deployed range. Lablet has developed a compact operating system "TinyOS" in just few Kilo Bytes that handles the encoded data packets to relay when the data is needed. Researchers in sensors at Berkeley are aiming to produce motes even smaller, one cubic millimetre, where they can be deployed in biometric applications.

To develop the understanding of mote programming and simulation this paper investigates the languages and simulators available in industry and then focuses on TinyOS and TOSSIM. We take the example of Berkeley's MICA and MICA2 motes

which are popular due to their tiny architecture, open source development and commercial availability. Table 1 shows the specifications of a typical MICA and MICA2 motes.

Table 1 – MICA and MICA2 specification

Mote type	MICA	MICA 2
Chip	ATmega103L	ATmega128L
Type	4 MHz, 8 bit	8Mhz, 8 bit
Program Memory	128KB	128KB
RAM	4KB	4KB
External Storage	512KB	512KB
Default Power	2xAA	2xAA
Typical capacity	2850mAh	2850mAh
Radio Frequency	868/916MHz	868/916MHz, 433, or 315 MHz
Modulation type	Amplitude Shift Key	Frequency Shift Key

Atmel ATmega103L is the main microcontroller unit (MCU) which does the regular processing. ATmega103L has integrated 512KB flash memory and 4KB of RAM.

The low powered radio from Chicom delivery up to 19.2Kbs application bandwidth on a single shared channel and with a range of up to around 100 meters, at full power a MICA2 mote can run for only around two weeks before losing its power.

These very tiny motes specifications have raised significant challenges in writing motes applications. We have raised the issues of challenges in sensor network programming and simulations. Section 2 summarise the related work. In section 3 we have emphasised on TinyOS and nesC the de-facto operating and programming environment for motes. Section 4 is on TOSSIM demonstrating its functionality in several scenarios. In section 5 we have described the programming challenges followed by Conclusion in Section 6.

2. RELATED WORK

2.1. Ns2

Ns2 is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks [9] on the packet level. The Ns project which was initiated by UC Berkeley is now part of the Virtual InterNetwork testbed (VINT) project that develops tools for network simulation. Ns-2 is an open source network simulator originally designed for wired, IP networks. However, extensions have been made to simulate wireless and mobile adhoc networks (MANETS). There are at least two efforts to extend Ns-2 to simulate sensor networks: SensorSim from UCLA and the NRL sensor network extension from the Navy Research Laboratory [6]. The main functionality of Ns-2 is implemented in C++ while the dynamics of the simulation is controlled by Tcl scripts. Basic components in Ns-2 are the layers in the protocol stack. Ns-2 architecture is composed of five parts. (1) event handler (2) network components (3) Tcl (4) OTcl library, and (5) Tcl 8.0 scripting language. The key advantage of Ns-2 is its rich libraries of protocols for nearly all network layers and for many routing mechanisms.

2.2. PARSEC

PARALLEL Simulation Environment for Complex Systems is a C based discrete-event simulation language developed by the Parallel Computing Laboratory at UCLA. One of the distinguishing features of PARSEC is to execute a discrete-event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. It is designed to separate the description of a simulation model from underlying simulation protocol, sequential or parallel, used to execute it. The latest version of PARSEC is available through the Parallel Computing Laboratory web site at <http://pcl.cs.ucla.edu/projects/parsec/> by filling out the registration form. [7]

2.3. GloMoSim

Global Mobile Information Systems Simulation Library is also UCLA product to build a scalable simulation environment for wireless and wired network systems [8]. It was designed using the parallel discrete-event simulation capability provided by Parsec. Glomosim architecture follows the OSI layering approach. Standard APIs will be used between the different simulations layers allowing the rapid integration of models developed at different layers by different people. To run Glomosim a familiarity with Parsec is needed, most of the code is in C with some Parsec functions for time management.

2.4. SensorSim

SensorSim is a simulation framework for modelling sensor networks [10]. SensorSim is build up on the ns-2 simulator and provides additional features for modelling sensor networks. The main features of SensorSim includes: sensing channel and sensor models, battery models, lightweight protocol stacks for wireless microsensors, scenario generation, and hybrid simulation that allows the interaction of real and simulated nodes. However, SensoSim remain unfinished and any further developments have been withdrawn.

2.5. MoteLab

Motelab is a web-based sensor network testbed, consisting of permanently deployed sensor network nodes connected to a central server which handles reprogramming and data logging while providing a web interface for creating and scheduling jobs on testbed [11]. There are four main components of MoteLab: (1) MySQL Database Backend - to store data collected during experiments. (2) Web Interface - PHP-generated pages present a user interface for job creation, scheduling, and data collection. (3) DBLogger - Java data logger to collect and parse data generated by jobs running on the lab, and (4) Job Daemon - A Perl script running as a cron job to setup and tear down jobs.

2.6. TinyGALS

TinyGALS is a structured globally asynchronous and locally asynchronous programming model for event driven embedded systems. Software components are composed locally and through synchronous method calls to form modules, and asynchronous message passing is used between modules to separate the flow control [13]. Message passing at global level might not be efficient if every update of data results a reaction, therefore a set of guarded synchronous variables TinyGUYS is provided at system level for thread-safe sharing by multiple modules without explicitly transferring the messages. TinyGALS supports all TinyOS components and have been implemented on Berkeley motes

3. NES C AND TINYOS

nesC is an extension to C language, a new event driven programming environment developed for networked embedded systems such as sensor networks. nesC supports a programming model that integrates reactivity to environment, concurrency and communication [2].

TinyOS is an operating system specifically designed for sensor networks. It is supported by nesC's component-based programming model. TinyOS applications are a collection of components, where each component has three computational abstractions: commands, events and tasks.

TinyOS defines a number of concepts that are expressed in nesC. First, nesC applications are

built out of components with well defined bidirectional interfaces. Second, nesC defines a concurrency model, based on tasks and hardware event handlers and detects data races at compile time [5]. There are two types of components in nesC *modules* and *configurations*. Modules are implemented by application code (written in a C like syntax). Configurations are implemented by connecting interfaces of existing components.

A nesC application consists of one or more linked together to form an executable. Components are classified as *provides* and *uses* interfaces components [6]. A *provide* interface is a set of methods calls the upper layers while *uses* interface is a set of methods calls the lower layer components. An interface declares a set of functions called *commands* that the interface provider must implement, and another set of functions called *events* that the interface user must implement.

4. TOSSIM: SIMULATOR FOR TINYOS

TOSSIM is a discrete event simulator for TinyOS wireless sensor networks which captures the behaviour and interactions of motes at bit level in contrast to the packet level in other simulators. TOSSIM architecture is composed of five parts:

- TinyOS Component Graphs – support the users in compiling TinyOS component graphs into the simulation. TinyOS component has five interrelated parts: command handlers, event handlers, an encapsulated private data frame, structure of private variables and a bundle of simple tasks. Tasks, commands and event handlers execute in the context of frame. Commands and events are mechanism of communication between components and task are internally used.
- Execution Model – a discrete event queue responsible of actuation of simulation. TOSSIM models interrupts through simulator events where each event is associated with a specific mote, after event execution scheduler executes tasks on mote's TinyOS tasks queue and executes all of them in FIFO scheduling order. Task queue of the associated mote is executed with the event until the whole queue is finished.
- Models – radio and analogue-to-digital converter (ADC) are used to model the characteristics of TinyOS motes. Radio models define the characteristics concerning the transmission from node to node. A drawback in TOSSIM radio model is that distance does not affect the signal strength making interference in TOSSIM generally worse than real world behaviour. There are two types of radio models [3]. The “simple” radio model and

the “lossy” radio model. The simple radio model places all nodes in a single cell where every bit transmitted is received without any error. Although no bits are corrupted due to error however, two motes transmitting at the same time lead to a problem of overlapping the signals, but the probability of two motes transmitting is very low due to TinyOS CSMA protocol.

The “lossy” radio model places the nodes in a directed graph. Each edge (x, y) in the graph means that the signal of node x can be heard by node y . Every edge has a value in the range representing the probability of a bit sent by node x will be corrupted (flipped) when node y hears it.

TOSSIM provides two ADC models: random and generic. The ADC has several channels that can be sampled. In this model, whenever a channel is sampled it returns a 10-bit random value. The generic model provides the random values; in addition to that it provides the possibility to be actuated by external applications by setting the value for any ADC port on any mote.

- Hardware abstraction components – TinyOS abstracts each hardware resource as a component. TOSSIM replaces only a small number of these components like ADC, the clock, the EEPROM etc. it emulates the behaviour of the underlying hardware. TOSSIM models these components in the hardware abstraction components part of the architecture.
- Communication services – to allow applications running on a PC to communicate with TOSSIM over TCP/IP.

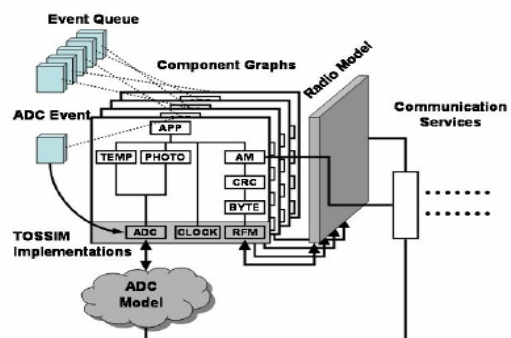


Figure 1: TOSSIM Architecture: Frames, Events, Models, Components and Services [1]

TOSSIM scales to thousands of nodes compiled directly from TinyOS source. TOSSIM simulates at bit level and replaces hardware with software components making possible scalable sensor network simulations. Here are four goals which TOSSIM aims to meet:

Scalability – able to handle large network of thousands of nodes in a wide range of configurations.

Completeness – accurately captures complete system behaviour

Fidelity – captures the network behaviour at fine grain by revealing unanticipated interactions. TOSSIM has helped debugging TinyOS network stack problems. It simulates the network at bit level.

Bridging – bridges the gap between algorithm and implementation by allowing developers to test and verify the code that will run on real hardware. Tested code can be deployed right away without any change to the application.

TinyViz is a TOSSIM’s visualisation package, which is a Java application that can connect to TOSSIM simulations. TinyViz also provides a mechanism to control a running simulation by modifying ADC readings, changing channel properties and injecting packets. TinyViz is designed as a communication service which interacts with the TOSSIM event queue. TinyViz provides a plugin interface allowing developers to implement their own application specific visualisation and control code within the TinyViz framework [4].

To test the basic features of TinyViz we have compiled a small application TestTinyViz of 20 motes which causes motes to periodically send a message to a random neighbour. Here we post the screen shots of a series of plugins while our application is running.

Figure 2 shows the graphical display of the sensor network in the left pane and the control panel on the right where we can interact with a series of plugins that control how TinyViz works. Here we activate Radio link plugin which graphically displays radio message activity. The blue circle denotes the message broadcast. When a mote sends a message to another mote, a directed arrow is drawn between the two motes

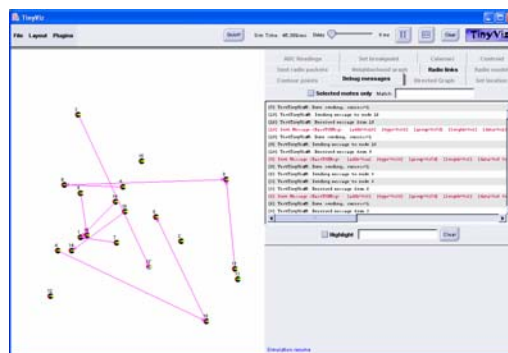


Figure 2: Debug Message plugin

In the figure above debug message plugin is displaying all the debug messages generated by the simulation.

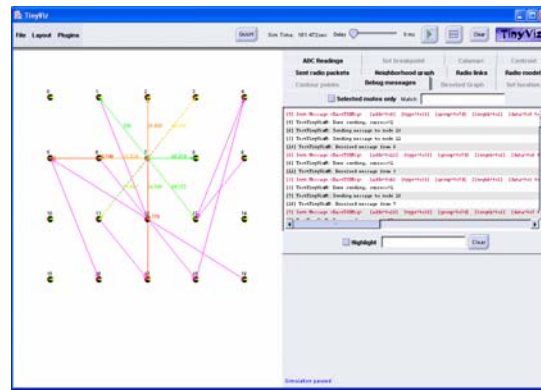


Figure 3: Radio Model

Figure 3 exhibits radio connectivity for 20 motes and network traffic. The green and yellow arrows represent link probabilities, and the magenta arrows indicate packet transmission.

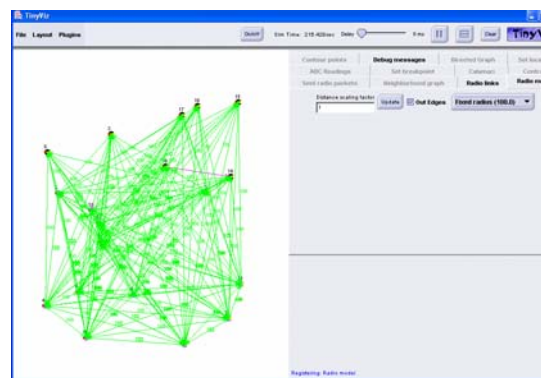


Figure 4: Fixed Radio model (RFM 100 radios)

Figure 4 illustrates a very densely-connected mesh, indicating that every mote has connectivity with all others in a fixed radius of RFM 100.

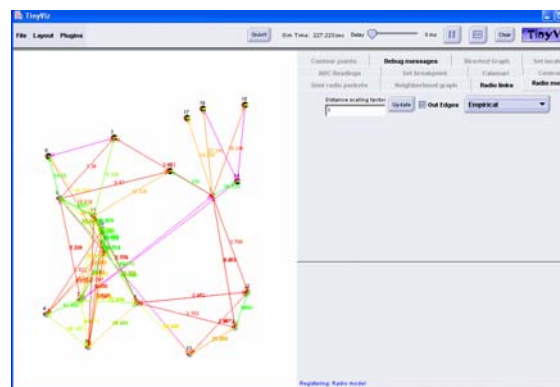


Figure5: Empirical Radio model

Radio model sets the bit error rate between motes according to their location and radio connectivity. Enabling this plugin allows us to use realistic connectivity models in the simulation. There are two built in models: Figure 5 shows the “Empirical model” based on out door trace of packet connectivity with the RFM1000 radios and in Figure 9 we display “Fixed radius” modelling motes within RFM100 radios having a perfect connectivity and no connectivity to the motes beyond this range.

5. PROGRAMMING CHALLENGES

The biggest challenge faced in sensor network programming is reducing the code size and improving the response time, and minimising the energy consumption in resource constrained nature of sensors with limited memory and very little computing power. Modularising TinyOS to deploy only the necessary parts of applications and real time scheduling of resources to more urgent tasks are the possible solutions. Event driven nature of programming in sensor networks allows the sensors to fall into low-power sleep mode when there is no activity going on. Programmers have to directly face the device drivers and scheduling algorithms, and optimise the code at assembly level. Though these techniques work well for stand-alone embedded systems, however, using them for sensor networks introduces further challenges: First, sensor networks are densely deployed large scale distributed systems where distributed algorithms itself are hard to implement on a large number of distributed nodes. Second, sensor networks are deployed in hostile situations where they have closer interaction with the surroundings; they should be able to respond to multiple concurrent stimuli at the speed of changes in the environment around them.

6. CONCLUSION

This paper has listed the programming and simulation methodologies in sensor networks. We have presented an overview of related work describing, NS2, Parsec, GlomoSim, SensorSim, MoteLab, and TinyGALS as node-level programming solutions. We have highlighted Berkeley's NesC, TinyOS and demonstrated the sensor network in several ways in TOSSIM. We have emphasised that programming tools for sensor networks are a new area which is more challenging due to resource starved nature of sensor nodes.

REFERENCES

- [1] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications", *Proceedings of the First ACM Conference on embedded Networkd Sensor Systems (SenSys 2003)* 2003.
- [2] D. Gay, P. Levis, R. V. Behren, M. Walsh, E. Brewer and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems", In *Proceedings of Programming Language Design and Implementation (PLDI) 2003*, June 2003
- [3] P. Levis and N. Lee, "TOSSIM: A Simulator for TinyOS Networks", UC Berkeley 2003
- [4] Simulating TinyOS Applications in TOSSIM. Available [online]<http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson5.html> [Retrieved on 27 June 2005]
- [5] TinyOS Getting Started Guide, available [online] http://www.xbow.com/Support/Support_pdf_files/TinyOS_Getting_Started_Guide_7430-0022-03_A.pdf [Retrieved on 27 June 2005]
- [6] F. Zhao and L. Guibas, "Wireless Sensor Networks: An information Processing Approach", Morgan Kaufmann Publishers, San Francisco CA, pp 248-255 2004.
- [7] UCLA Parallel Computing Lab: PARSEC. Available [online] <http://pcl.cs.ucla.edu/projects/parsec/> [Retrieved on 25 June 2005]
- [8] Glomosim. Available [online] <http://pcl.cs.ucla.edu/project/glomosim/> [Retrieved on 25 June 2005]
- [9] The Network Simulator Ns-2. Available [online]. <http://www.isi.edu/nsnam/ns/> [Retrieved on 26 June 2005]
- [10] S. Park, A. Savvides and M. B. Srivastava, "SensorSim: A Simulation Framework for Sensor Networks", *In the Proceedings of MSWiM 2000, Boston, MA*, August 11, 2000
- [11] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A wireless sensor network test bed", Harvard University 2005
- [12] J. Elson, S. Bien, N. Busek, V. Bychkovskiy, A. Cerpa, D. Ganesan, L. Girod, B. Greenstein, T. Schoellhammer, T. Stathopoulos, and D. Estrin, "EmStar: An environment for developing wireless embedded systems software", *19th ACM Symposium for Operating Systems Principles SOSP 2003*
- [13] E. Cheong, J. Liebman, J. Liu and F. Zhao, "TinyGALS: A Programming Model for Event-Driven Embedded Systems", Florida USA, SAC 2003