

UNIVERSITY OF CALIFORNIA  
Los Angeles

Simulation of SOS Systems

by

Balaji Vasu  
Computer Science Department

2005

## ABSTRACT

### Simulation of SOS Systems

By

Balaji Vasu

Master of Science in Computer Science  
University of California, Los Angeles, 2005

Sensor networks are in wide spread use and with deployments reaching larger numbers, the need for a scalable simulation framework is becoming apparent. Simulation of sensor networks offers the capability to test network protocols under different topologies, with repeatability in mind. While repeatability and scalability are important criteria, wireless channels are hard to model. The best-case scenario would be to evaluate the protocols under a testbed of real nodes. This too, has a severe drawback in that the evaluation is restricted by the size of the testbed. This project presents a simulation framework that can be used to simulate SOS applications and combine them with a hybrid simulation capability that allows for nodes in the simulation to be mapped to real nodes.

## 1. Introduction

Sensor networks are envisioned to be ubiquitous with the size of the networks reaching tens of thousands. The current state of the art sensor networks range from data collection to mobile platforms such as Ragobot [1]. These advances in sensor networks have induced a need for accurate simulations. While it is possible to test the code on a testbed, the testbeds generally tend to be a fraction of the size of the real deployments. Many important results, such as power consumption and hotspots, may not be realized in a small testbed. Accurate simulations allow testing of complex systems and evaluation of the systems' design with a high degree of confidence. The fallacy here is that the simulations need to be accurate. While the testbed offers the advantages of instant deployment and testing on real hardware, it lacks in that the size is not representative of the actual deployment. The simulations generally can be used to simulate the entire deployment, but the fidelity of the simulations dictates how useful the results are. Combining the two methods of testing to create a hybrid simulator offers the best of both methods.

The necessary features for a sensor network simulator:

### Scalability:

By keeping some of the nodes real, while most are simulated, it is possible to scale the network to the large numbers that are required.

### Fidelity:

It is possible to crosscheck the simulated nodes with the real nodes to see how accurate the simulated nodes are.

### Ease of deployment:

The code running inside the simulator is the same as the code running on the real nodes. There is no special code that needs to be written for simulation as opposed to deployment. This eases the gap between testing and deployment.

MicaZ motes [16] are the successor to the mica2 nodes. The MicaZ motes consist of an Atmel ATMEGA128L micro controller, a Chipcon CC2420 radio, an ADC, an UART,

timers, flash and leds. SOS is used in this class of sensor network devices in NESL and run on the XYZ node from Yale [17].

## 2. Related Work

For this project, the relevant types of simulators are those that simulate the networks and those that simulate AVR code. Network simulators like Qualnet [12], OpNet and NS-2 [14] can be used to evaluate network protocols for both conventional networks as well as wireless sensor networks. These simulators offer an accurate simulated propagation channel, as well as accurate physical models. These simulators, though, lack the capability to simulate code accurately.

There are also sensor network simulators, which specialize in simulating sensor networks. SensorSim [2] offers capabilities such as power modeling and other sensor network specific needs. sQualnet [15] focuses on the same features that SensorSim offers but uses Qualnet as a base for the simulations. A. Dixit's master project [7] focused on simulating TinyOS applications in Qualnet.

TOSSIM [3] is the base simulator for TinyOS [11]. It simulates any TinyOS application but lacks a radio model. It is very useful for debugging applications, as the simulator works by compiling the NesC code to the PC target instead of the AVR target.

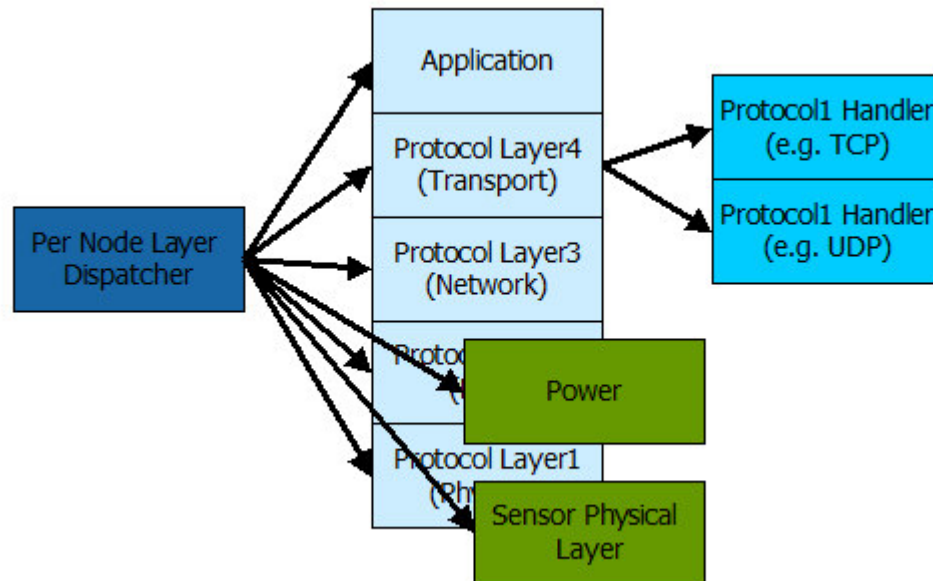
AVR simulators simulate AVR instructions and are cycle accurate. These lack an accurate radio model. Avrora [6] AVR instructions and therefore can simulate any application. ATEmu [4] is another AVR simulator. These types of simulators require powerful processors as they interpret and simulate each instruction.

EmTOS [4] is a work that is very close to this project; it simulates NesC code inside EmStar. EmStar can then be used for debugging and testing applications.

### 3. System Components

#### 3.1 Qualnet

Qualnet, a commercially available network simulator, combines the simulation kernel of PARSEC [8] with the network capabilities of Glomosim [10]. Qualnet scales well with the number of nodes, with a ten thousand-node network not out of its reach. Its strength lies in this scalability as well as its exceptional propagation models.



**Figure 1: Qualnet Layering**

Qualnet was chosen as the base simulator due to superior framework to NS-2 and other network simulators. Qualnet allows new models to be added to any layer with ease. The basic means of communication for intra or inter node communication is its message passing API. An application generates a message to be sent out of the radio using the same API as it does to set a timer for itself. This independence of the message passing API from the simulation leads to simpler programming.

The strength of qualnet is its scalability and exceptional propagation models. This is important due to two factors:

- a) Scalability: When simulating sensor networks, it is not uncommon to simulate tens of thousands of nodes; while a traditional network might only be simulated with 100s of nodes, a sensor network is inherently larger in size. A simulator that aims to be used a sensor network simulator must be able to simulate the larger number of nodes.
- b) Propagation models: When the user simulates a large sensor network, the topology used might be that of an actual deployment. It is, therefore, very important that the wireless channel is modeled accurately. The simulation can then be used to detect connectivity and other such problems.

### **3.2 SOS**

SOS is an operating system aimed at the mote-class sensor networks. Its primary advantage over TinyOS is that it allows modules to dynamically loaded off the air. It has a kernel that implements messaging, memory managing and module loading capabilities. The kernel also has required device drivers installed on to it. The application level software can either be compiled with the kernel or be loaded dynamically.

One of the primary reasons for allowing dynamic loading of modules is that the network can be reconfigured without any human intervention. A network that has been deployed can be modified as the administrator sees fit; modules can be updated and old modules can be removed. While this is possible in TinyOS and other operating systems, they are much more expensive in terms of energy costs.

The programming language for SOS is C, rather than NesC. This makes life simpler when attempting to morph the code to work under a different platform. Currently, SOS lacks a solid simulation platform, and this project aims to provide it that.

### **3.3 Hardware**

#### **3.3.1 Ceiling Array**

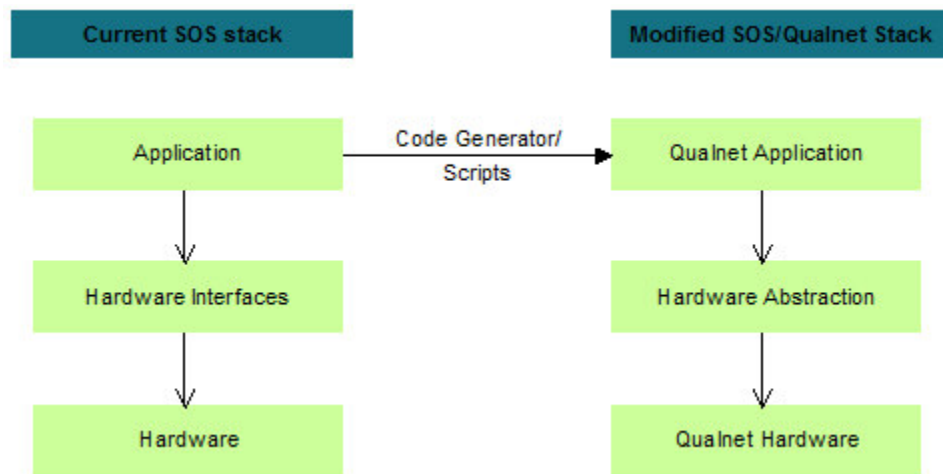
The Networked and Embedded Systems Laboratory (NESL) [18] has installed a ceiling array with 20 MicaZ motes on MIB600 programming boards. These programming boards can be accessed via the Internet. Each board has assigned to it an address; a command to this IP address programs the mote attached to the board. The UART can be accessed via port number 10002. This allows for a simple means of accessing the motes attached on the ceiling array. The ceiling array is powered by Power-over-Ethernet, rather than conventional adapters or batteries.

## 4. Simulation Capabilities

The primary simulation capabilities for a system running SOS are the following:

### 4.1 Unmodified code simulation

One of the primary focuses of this project is to allow SOS applications to be simulated inside qualnet. Forcing the programmer to make qualnet-specific modifications in the code decreases from the ease of use of the simulator. Therefore, all necessary code morphing is performed via a parser/code generator and script files. That the programming language is C allows for a much simpler code generator.



**Figure 2: Code Morphing process**

The following was done to allow for the code morphing:

- a) Code generator to generate C code: The C parser was obtained from Professor Palsberg. Then JTB was used to generate an AST. From here, the code generator was used to make context specific changes to the SOS code. These changes are required to allow SOS code to compile inside qualnet and do not change the semantics of the program.

- b) Script files: The script files preprocess the SOS code and generate a monolithic file that can be fed into the C parser.

## **4.2 Simulation Models**

Here, work from A.Dixit was used as a framework. The timers and radio are currently the two devices that can be simulated inside qualnet.

## **4.3 Module insertion**

SOS's primary claim is that modules can be loaded dynamically. Therefore this feature must be simulated as well. The simulation consists of a blank kernel running on the node, with each module triggered inside the simulation as module comes off the air.

## **4.4 Hybrid Simulation**

The final simulation highlight is that of hybrid simulation. There are three different modes that are possible:

### **4.4.1 Sensor Stack**

The sensor stack can be simulated on the real node, thus allowing for real data to be generated. This can be used for various purposes, including verification of a sensor model. One caveat is that an application that requires strict timing for its data would not be accurate with this mode of simulation. This mode is designed for applications that process sensor data with lax timing. In sensor networks, several of these types of applications exist. One such application is data gathering. Most applications generate data until their buffer is filled and only then do they transmit the data. This mode can also work if the application requires samples at a fairly low rate, such as 10 samples/sec.

#### **4.4.2 Channel emulation**

The MAC + radio layer resides on the real node, while the application code runs on the virtual node inside the simulation. This is very useful as this combines the best of simulations with the best of the real world testbed.

#### **4.4.3 Code Verification**

The code runs on the real node, while the MAC and the radio are simulated inside qualnet.

This mode is the opposite of the channel emulation mode. The code running on the real node ensures cycle accurate code simulation on the node, while the MAC and radio layer simulation allows for easier statistics tracking and debugging.

For example, consider the surge example.

Surge nodes periodically sample the light sensor and multi-hop the data back to the base-station. A spanning tree constructed by surge determines the route to the base-station.

The application code is on the real node, and therefore all the sensor data gathered are time-stamped properly. These time-stamps are accurate regardless of the sampling rate.

The radio is simulated inside the simulator and this allows for easier tracking of statistics, such as packets lost due to collision and other vital statistics for a routing protocol.

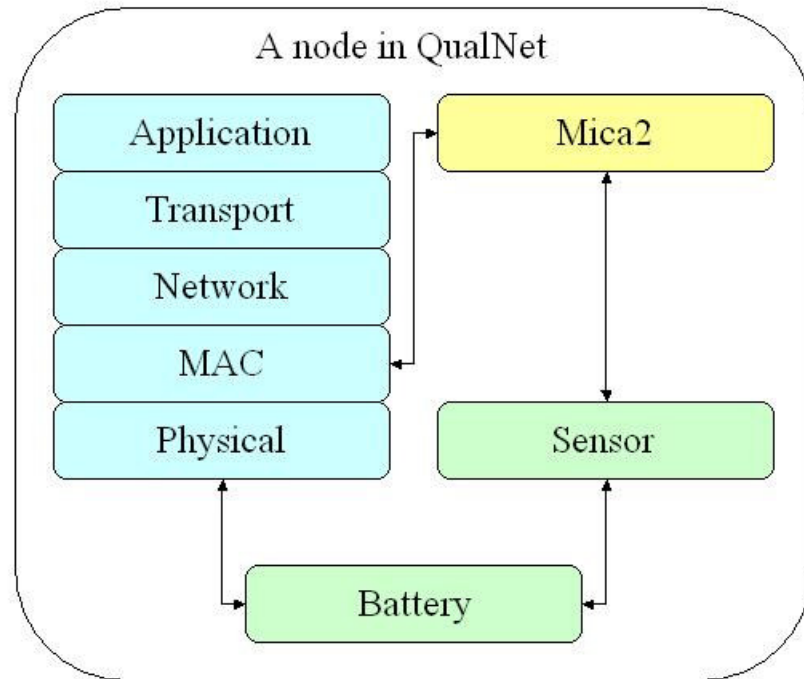
This method can also be used to debug code, as the MAC layer inside qualnet is completely configurable. To eliminate collisions, for example, the user can use TDMA at the MAC layer.

Finally, to test how well the routing protocol performs under adverse conditions, such as loss of control packets and network partitioning, the radio inside qualnet can be configured.

## 5. Implementation

### 5.1 Simulation Models

The framework for simulating TinyOS applications is as follows:



**Figure 3: Simulator Framework**

The same framework is used to simulate the SOS applications.

The mica2 layer resides on top of the MAC layer inside qualnet, providing the necessary API calls to the MAC layer. The important API calls are Queuing and Dequeing of packets.

Each SOS application is derived from a base class called SOSApplication, which has the necessary hardware calls and other signaling functions.

The necessary modification to the code scheduler is that the scheduler can no longer run in an infinite loop. Instead, at each event timestep, all the SOS events are scheduled and

taken care of; the scheduler then stops looping waiting for any more events. Instead it simply exists, allowing Qualnet to proceed the simulated time.

## 5.2 Module Insertion

Module insertion in a real SOS node works as below:

The module insertion message comes in off the air, and is sent to the main scheduler.

This message is then routed to the **ModuleD** daemon, which is responsible for parsing the message and registering the module. The **ModuleD** daemon follow a protocol to send and receive a module, but this project does not follow this protocol. It gives a framework for inserting modules, and allows the user to provide any protocol to send and receive modules. For example, the default protocol that comes with SOS is that when a node first wakes up, it broadcasts a message to receive any modules its neighbors have. While this is provided by default, a different user of the system might not want this protocol.

Therefore the protocol itself is left blank.

The simulation follows the same procedure. The module insertion message is sent to the **ModuleD** daemon, which then parses the message and registers it. The module insertion mechanism differs from that of a real implementation in that while in a real node, the newly received module is written to flash, the simulated version has all the modules precompiled. The code generation stage captures each module's three primary fields of information: Module Id, Module State Size and Module Handler and generates a table with this information.

The **ModuleD** daemon, when it parsers the information from the incoming message, looks up the table with the module id and registers the module handler. The network characteristics are intact as the incoming module insertion message has the module data as the message payload. The message does not contain just the id, state size and the handler address; this would defeat the purpose of using a network simulator.

### 5.3 Hybrid Simulation

The three modes of hybrid simulation use the same general method of communication with the real nodes. The basic necessity for a hybrid simulator is a channel to communicate with real nodes. This might be a serial port, but this requires all nodes to be connected to the simulating computer. Another method that has been used is to use a gateway node to which the network talks to, while the gateway node talks to the computer. This adds one more hop to the communication channel and complicates the timing synchronization. The method this project uses is to leverage the ceiling array testbed with Ethernet programming boards at NESL UCLA. Each node on the ceiling array can be mapped to a virtual node; while the communication channel is still the uart on the mote, the uart is mapped to an Ethernet port. It is to this Ethernet port that the simulator connects to and sends commands and receives data.

#### 5.3.1 External Interface

The interface between the external nodes and the simulator is the External Interface.

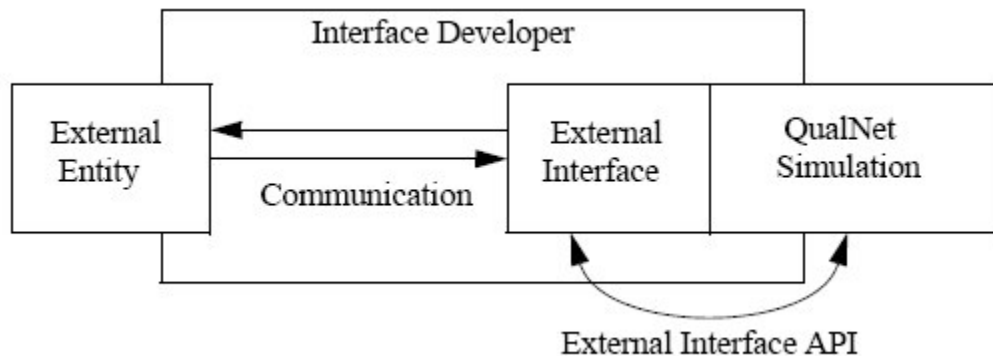


Figure 4: External Interface

The interface that has been implemented is called Mica2Interface. This interface performs the multiplexing and demultiplexing necessary for the virtual nodes to communicate with the real nodes. It also generates a socket connection for each mapping between a virtual node and a real node. The two primary functions that do this are **EXTERNAL\_ForwardData** and **EXTERNAL\_ReceiveData**. **EXTERNAL\_ForwardData** receives the packet from the virtual node, and maps the virtual node's id to the real node's id, and tunnels it to the real node via the corresponding socket

connection. **EXTERNAL\_ReceiveData** does the opposite; it reads all the sockets for any data and sends the data to the corresponding virtual node. The key data structure in this interface implementation is the mapping between virtual nodes and real nodes. For each map, the information stored is:

- 1) Virtual node's id
- 2) Socket connection to real node

In all, there are three distinct changes that were made to enable each type of hybrid simulation.

### 5.3.2 Sensor Stack

Here, the kernel drivers for the sensors were rewritten. **Ker\_sensor\_get\_data** is the system call from the application to gather data. It sends the rate and the sensor id from which to gather the data. This information is intercepted and formed into a SOS message and tunneled via the uart to the node on the real network. The real node has a parsing program that gathers this information and does a call to the real

**ker\_sensor\_get\_data** to gather data. The module driver in the SOS implementation allows the user to specify sensor number, rate, buffer size and a handler. This driver has the semantics that when the data gathered from the sensor number at the specified rate reaches the buffer size, the handler is called.

After the real node gathers the specified data, it tunnels back the information to the simulator via the uart. The interface then pushes this data back up to the receiving node. The virtual node keeps track of the time it sent the request and the time it received the information. This allows for crude time stamping of the received data.

### 5.3.3 Channel Emulation

Channel Emulation is simplified by a basic mode of SOS, NIC. When a node is in NIC mode, it sends any packet it received on the uart to the radio and vice-versa.

The radio drivers have been rewritten to use the uart in the virtual node as well. So when a virtual node sends a packet, the **radio\_msg\_alloc(...)** call results in a call to **EXTERNAL\_ForwardPacket** which tunnels the packet to the real node. The real node

then sends the packet out the radio to the receiving node. The receiving node then tunnels the radio packet it received via the uart to the simulator.

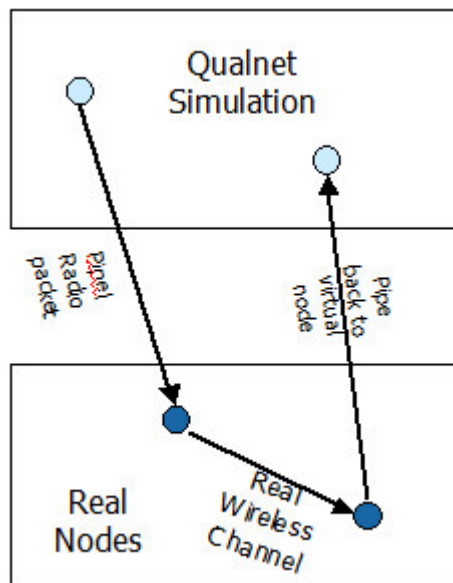
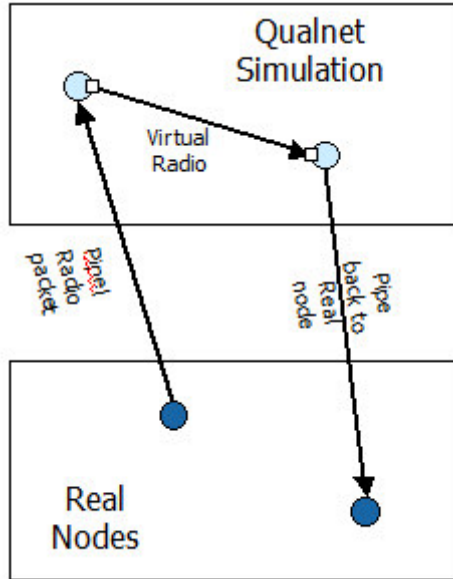


Figure 5: Channel Emulation Message order

The `radio_msg_alloc(...)` call in a virtual node also results in the packet being sent out the virtual radio, as there might be virtual nodes in its vicinity. All the nodes that are running in this mode will ignore any packets they receive on the virtual radio.

#### 5.3.4 Code Verification

This mode of hybrid simulation requires changes to the SOS kernel. The old kernel was configured to allow uart reception only if the node was defined as a `SOS_NIC`. In this case, a node running arbitrary code needs to be able to receive packets on the uart. The other change required is that the uart replaces the radio as the sending channel as well. This was already built into SOS, although it does require a new make target.



**Figure 6: Code Verification Message order**

With these changes, a real node sending a packet tunnels it through the uart to the simulator, which sends the packet via the virtual radio. Any nodes receiving this packet tunnel it through the uart, which is sent back to the corresponding application.

## **6. Future Work and Conclusion**

### **6.1 Future Work**

There are still various hardware simulations that need to be finished. The first of these will be the EEPROM, as this will allow for a more realistic module insertion simulation. Work is also being done to allow for simulation of actuation controlled sensor networks. The current implementation of the hybrid simulation assumes fairly lax timing constraints. Future implementations will allow more precise timing.

### **6.2 Conclusion**

A framework for simulating SOS applications in conjunction with real nodes was shown. Arbitrary SOS applications can be run inside Qualnet, with the module insertion and other key SOS features simulated. Important devices such as timers and radio were simulated; this allows for a deployment to be simulated accurately before the actual deployment. The hybrid simulation allows for even more accurate testing of the deployment system as it leverages the real wireless channels while allowing for repeatability of simulations.

## Bibliography

- [1] Ragobot – Real Action Gaming Robot, [www.ragobot.com](http://www.ragobot.com)
- [2] “SensorSim: A Simulation Framework for Sensor Networks”, S. Park, A. Savvides and M. B. Srivastava; Proceedings of 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM 2000
- [3] “TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications”, P. Levis, N. Lee, M. Welsh, and D. Culler; First ACM Conference on Embedded Networked Sensor Systems, SenSys '03, 2003
- [4] "A System for Simulation, Emulation, and Deployment of Heterogeneous Sensor Networks", L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer; Proceedings of SenSys 2004
- [5] ATEmu - Sensor Network Emulator / Simulator / Debugger, <http://www.cshcn.umd.edu/research/atemu/>
- [6] “Aurora: Scalable Sensor Network Simulation with Precise Timing”, B. Titzer, D. Lee, J. Palsberg; IPSN 2005
- [7] A.Dixit, “Simulation of TinyOS Applications in Qualnet”, Masters Project, UCLA 2004
- [8] "Parsec: A Parallel Simulation Environment for Complex Systems," R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, H. Song; IEEE Computer, Vol. 31(10), October 1998
- [9] “The nesC Language: A Holistic Approach to Networked Embedded Systems”, D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler; PLDI 2003, June 2003
- [10] “GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks”, Xiang Zeng, Rajive Bagrodia, Mario Gerla; Proceedings of the 12th Workshop on Parallel and Distributed Simulations, PADS 1998
- [11] Tinyos: a component based OS for the networked sensor regime. <http://webs.cs.berkeley.edu/tos/>.
- [12] QualNet – A scalable network simulator. <http://www.qualnet.com>

- [13] SOS, <http://nesl.ee.ucla.edu/projects/sos/>
- [14] The network simulator - ns-2, <http://www.isi.edu/nsnam/ns>
- [15] sQualnet, <http://nesl.ee.ucla.edu/projects/squalnet/>
- [16] MicaZ Mote, <http://www.xbow.com/Products/productsdetails.aspx?sid=101>
- [17] “XYZ: A Motion-Enabled, Power Aware Sensor Node Platform for Distributed Sensor Network Applications “, D. LyMBERopoulos, A. Savvides; IPSN 2005
- [18] Networked and Embedded Systems Laboratory, NESL, <http://nesl.ee.ucla.edu>