Emuli: Emulated Stimuli for Wireless Sensor Network Experimentation

Thomas Clouser, Richie Thomas and Mikhail Nesterenko
Department of Computer Science
Kent State University
Kent, Ohio 44242-0001 USA
{tclouser,rthomas,mikhail}@cs.kent.edu

Technical report: TR-KSU-CS-2007-04, Kent State University

April 23, 2007

Abstract

We describe Emuli — a method of effectively substituting sensor data by synthetic data on physical wireless nodes (motes). The sensor application is oblivious to this substitution. Emuli generates data on demand from the application. The replies are based on the sensor model which is driven by the data pre-loaded to the mote before the experiment. Since the preloaded data is an approximation of the sensor behavior rather than raw sensor readings, it is rather compact. This allows Emuli to drive sophisticated experiments. The emulated stimuli can be synchronized and coordinated across multiple motes which allows to experiment with distributed events. Emuli abstracts the sensing component of a complete application and allows the experimenter to focus only on processing and transport function of a wireless sensor networks. We demonstrate statistical and deterministic sensor models. We showcase the use of Emuli in a light measurement and a target tracking experiment.

1 Introduction

Wireless sensor networking is rapidly moving from a promising engineering novelty to a technology of choice for a variety of applications. Sensor networks is an attractive platform as it enables the applications outside the realm of traditional computing. An individual sensor node is inexpensive due to its limited computing, communication and power resources. This permits the deployment of hundreds and possibly tens of thousands of them. Such massive networks allow gathering of information about the environment with unprecedented resolution. The sensor networks are used for environmental monitoring [4, 24], habitat monitoring [12, 14] and military surveillance [1,3,7]. Berkeley prototype sensor nodes (motes) [9] proved to be the most widely accepted experimentation and development platform due to their ease of operation and flexibility. Motes run TinyOS operating system [8] which is specifically designed for operation on such resource constrained devices.

Wireless sensor networks proved to be rather difficult to experiment with. A field experiment is an ultimate test of operation of a sensor network. However, experiments in the field are not well-suited for software development and debugging. The limited resources of each individual node and the distributed nature of sensing applications make examining and controlling the processes in the network challenging. For example, due to changing weather conditions it is difficult to repeat or even ascertain the inputs of the environment monitoring network. In addition, the logistics of deploying a large number of sensors render field testing a supplemental rather than the main means of experimentation. Hence, a sensor network simulator [5, 6, 11, 17–23, 26] emerged as a major research tool. Simulators range widely in the degree of fidelity of representation of the sensor architecture. On

the one end of the this range there are high-level simulators like MATLAB-based Prowler [20] which focuses on abstract distributed algorithms for sensor networks. On the other — instruction level emulators like Avrora [23] and ATEMU [19]. However, fidelity of a simulation cannot always match the physical devices. To increase simulation fidelity, *hardware-in-the-loop* or *hybrid* simulators [6, 22, 26] execute some of the functions on real sensor nodes. A popular approach to achieve high-fidelity in experiments is to use a sensor *testbed* [2, 25, 27, 28]. A stationary array of sensor nodes is placed in an environment where their behavior can be controlled and studied. Besides saving the deployment effort, a testbed, due to stationary sensors and controlled setting, allows for a certain predictability of the experimental environment.

Even though testbeds are quite useful to study the data transport and processing functions of the network, the sensing function is more difficult to reproduce. For example, to provide authentic sensor stimuli for a tracking application, the testbed has to be outfitted with targets, the mechanism to determine the exact track and velocity of the target movement as well as means of controlling and repeating such movement. Besides increasing the costs of a testbed, such approach limits the experimentation to one particular application. Yet, as the sensing function of a wireless sensor network is inextricably linked to the other functions, the utility of a testbed diminishes. Hence the need to consider effective sensing emulation.

Related literature. Several prior projects focused on sensing emulation. Park and Chou [16] consider replacing a sensor with a device that supplies the sensor node with readings that the experimenter programs. While this method of sensor emulation has the potential to be quite accurate, the cost and possible lack of flexibility may limit its use. Jia, Krogh and Wong [10] describe TOSHILT: a tool that allows the application to replay previously recorded or synthetic sensor readings. The readings are loaded in advance and stored in sensor nodes' EEPROM memory. Luo et al [13] further enhance the concept of this direct playback by providing a markup language and a compiler that allows the programmer to specify which particular data points need to be recorded and then replayed. This approach of direct sensor emulation has two limitations. The length of the experiment is limited by the EEPROM memory size. TOSHILT allows to repeat the playback when the end of the recording is reached. However, this kind of extension may not be flexible enough for some applications. Another limitation lies in the necessity to adjust the timing of the requested sensor reading to the timing of the recorded one. Specifically, direct recorded sensor readings state the value of the measured parameter at the time of the recording — a *spot measurement*. Assume that that this parameter is quickly changing with time. If the timing of the sensor request deviates from the recording time, due for example to jitter or interrupt processing interference, the played back value will diverge from the expected one.

In this paper we describe *Emuli*: a method of emulating sensor stimuli of sensors. Emuli implements a model of a sensor behavior. In contrast to the earlier presented approaches, does not record and play back spot measurements. Instead, Emuli stores the model parameters. This results in a rather compact data memory footprint and a convenient and flexible sensor model. Emuli is designed to increase the capability of sensor testbeds and other deployments to experiment with environment sensing and monitoring.

2 Emuli Description

2.1 Architecture

Emuli is designed to operate on motes [9] running TinyOS [8]. In TinyOS the application is merged with the operating system into a single hierarchy of components. Each component provides functionality for the upper components and utilizes the services of lower components. The hardware blocks such as a timer or an analog-to-digital converter (ADC) are represented as the lowest components. The component interaction is carried out through a well defined interface of commands and events. An event is signaled by the lower component and consumed by the upper one. A command is called by the upper component to the lower one. Ultimately, the events are hardware interrupts initiated by hardware. The synchronous work is thus done by TinyOS components servicing an interrupt.

In a TinyOS application, periodic data sensing is typically done as follows. TinyOS configures timer interrupts to occur at desired sampling rate. When timer event oc-

curs, the application components request the sensor component to do the measurement. The sensor sample is digitized by the ADC asynchronously. When the ADC reading is ready, the ADC component posts an interrupt which is interpreted as an event by the upper components. Emuli replaces the sensor component by its own component that simulates its operation. The objective is to model the output of the sensor with sufficient accuracy.

mote id	readings	
	actual	Emuli
1	900.68 ± 0.92	900.69 ± 0.21
2	882.76 ± 1.04	882.76 ± 0.24
3	859.53 ± 1.47	859.55 ± 0.34
4	916.88 ± 0.61	916.87 ± 0.14
5	868.56 ± 1.02	868.57 ± 0.24
6	952.73 ± 0.56	952.73 ± 0.13
7	957.78 ± 0.5	957.78 ± 0.12
8	943.59 ± 0.52	943.61 ± 0.12
9	940.42 ± 0.61	940.44 ± 0.14
10	952.97 ± 0.54	952.96 ± 0.13
11	915.81 ± 0.63	915.77 ± 0.15
12	951.42 ± 0.47	951.36 ± 0.11
13	927.75 ± 0.8	927.73 ± 0.19
14	957.78 ± 0.46	957.78 ± 0.11

Table 1: Average actual and Emuli-generated light sensor readings with 95% confidence interval.

2.2 Light Sensor Emulation

The simulation of a light sensor demonstrates how a statistical model can be effectively used to represent environmental sensing. We implemented a simple light-sensor data collection application. To instantiate our model we collected light sensor data from 14 motes mounted on benches in a lab and equipped with MTS300 sensor board. For each node we collected 60 light measurements over an hour (one sample per minute) with the overhead fluorescent lights on. Surprisingly, the values and their distribution differed significantly between the motes. However, there was no discernable pattern in the readings. Left column of Table 1 shows the average readings for all motes. Left column of Figure 2 shows histograms of the readings of representative nodes. We ran a similar experiment with lights off but the readings were not conducive to statistical model instantiation: most motes constantly. reported a single value.

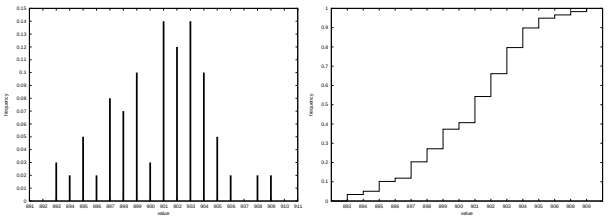

Figure 1: Light measurement frequency histogram and corresponding cumulative distribution function for mote 1.

On the basis of the experimental data readings we created a model or *personality* for each mote. The probability that Emuli reports a certain value x was made proportional to the number of times x was reported in the actual experiment. The experimental histogram and the corresponding cumulative distribution function for mote 1 is shown in Figure 1. This distribution function was stored in tabular form in the flash memory to be loaded at boot-time. We used RandomMLCG component from `contrib` directory of TinyOS distribution for random number generation. The initialization seed was left default.

For each individual mote we replaced the light-sensor component in our application with an Emuli component configured with a unique personality and run the experiments with this model for 1000 samples. The results are shown in Table 1 and Figure 2. The results indicate that Emuli data closely matches the experimental data in both average, standard deviation and distribution.

2.3 Range Sensor Emulation and Target Tracking

Range Sensor Emulation. We used Emuli to simulate a range sensor reading of target following a pre-defined track. A range sensor determines the distance from the mote to the target within its range. For this experiment the sensor range was set to 32 meters. The target moved in a zigzag pattern with speed of 3 meters per second across a

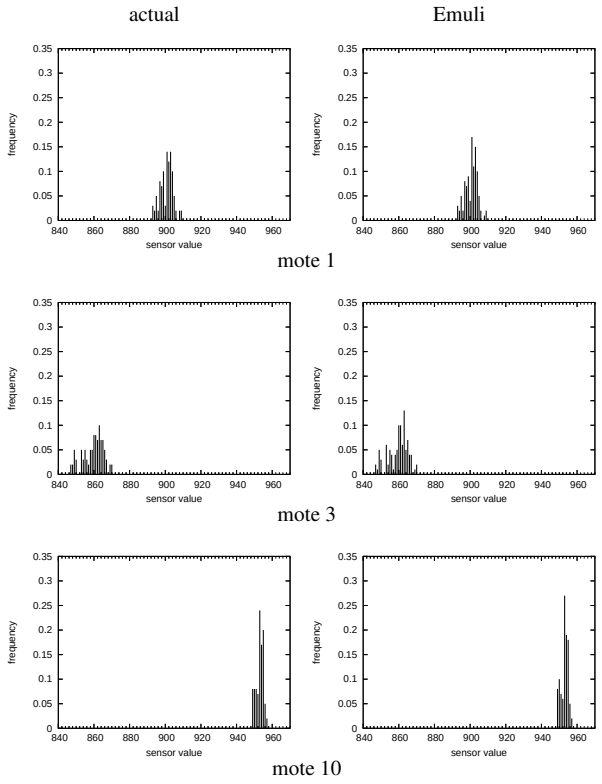

Figure 2: Sample measurement frequency histograms for light sensor readings.

plane surface. We used 5 motes in the experiment. The modeled track of the target and mote positions are shown in Figure 3.

To report the position of the target, each mote stored the information about the segment of the target track that was within the range of its sensor. Refer to Figure 4 for illustration. To optimize the calculations and storage, the segment is always represented by two points: the closest to the mote and the furthest sensed (i.e. the intersection of the track and the sensor range). Note that the track of the target through the sensing field of individual mote may have to be represented by multiple segments. This is especially so if the target changes direction within the sensing rage. To synchronize the target readings between motes, Emuli runs the FTSP time synchronization protocol [15].

For the track segment $|A, B_0|$, the mote stored the following data: times t_a and t_{b0} when the target was at the endpoints of the segment, target speed s, and distances $a^2 = |C, B|^2$ and b_0. Note that if $t_a < t_{b0}$ then the target moved from A to B_0. Assume that this is so. Times were stored as integers, distance and speed – as floating point numbers. To computer distance c of the target at time t, the Emuli component first determined whether t is within the time interval of this segment. If it was, Emuli computed the distance $b = (t - t_a)/s$. The actual distance c to the target was computed as $c = \sqrt{a^2 + b^2}$. Emuli also handles special cases for the target passing exactly over the mote or just touching the sensing range.

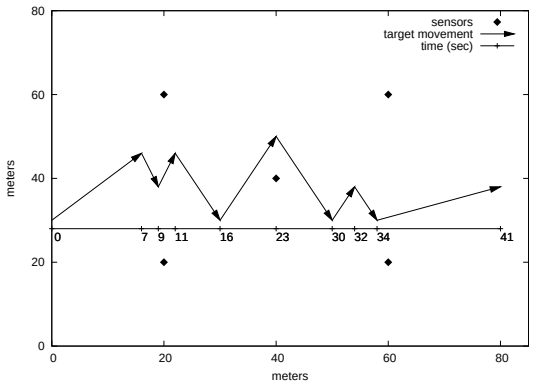

Figure 3: Simulated track presented by Emuli to the tracking application.

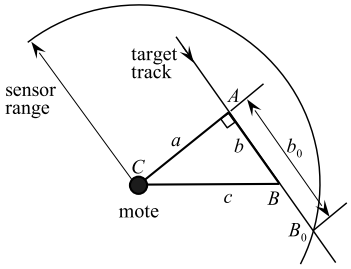

Figure 4: Target distance computation with Emuli

Target tracking and evaluation. To demonstrate the operation of range sensing simulation with Emuli, we implemented a simple trilateration application. The trilateration requires target distance measurements from three

motes (see Figure 5). The measurements are assumed to be simultaneous. We computed intersection points of the circles whose radius are these measurements. We selected two arbitrary pairs of intersection points and draw lines through them. The target location was at the intersection of these two lines. In our triangulation application, motes reported their timestamped target distance measurement to the base station. Offline, for each distance measurements, we selected two other closest in time measurements and computed the target location.

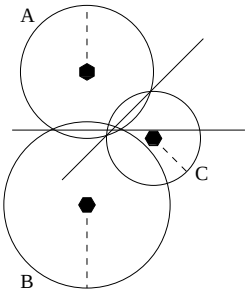

Figure 5: Trilateration with Emuli

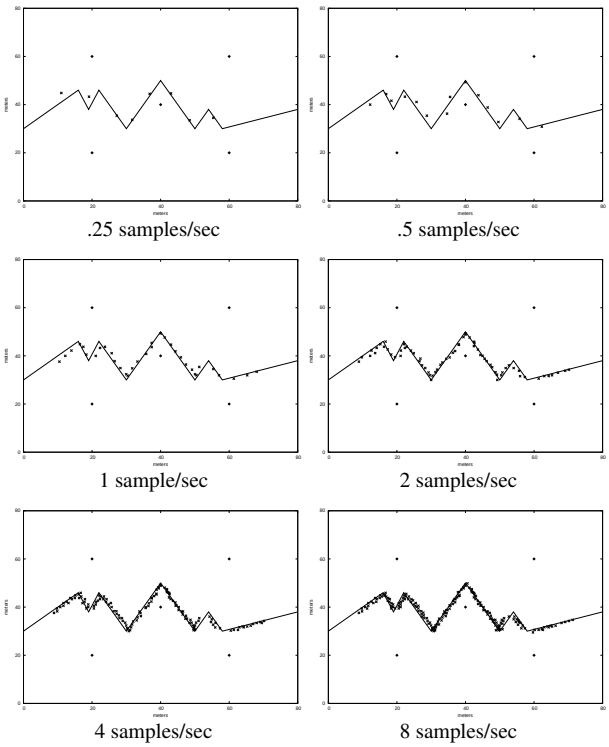

Figure 6: Tracking of simulated target at various sampling rates. Crosses mark the target location and black diamonds are motes.

To showcase Emuli, we varied the sampling rate of our application. The results are shown in Figure 6.

The precise track simulated by Emuli was known. This allowed us to compare the results computed by tracking application to this simulated "ground truth". In essence this comparison evaluates the performance of the Emuli-modeled range sensor. The analysis is shown in Figure 7. Figure 7 (a) shows the difference in distance returned by range sensor of Emuli and the actual distance of the simulated target from the mote at the time the measurement is taken. This difference stays roughly the same across sampling rates. The error can be attributed to time synchronization error and floating point rounding errors. Figure 7(b) shows the time differences between the three distance measurements used for trilateration. Recall that our trilateration algorithm selects three measurements at separate motes that are the closest in time. As the sampling rate increases, the time between measurements at separate motes decreases.

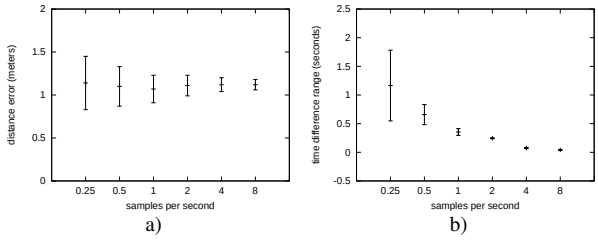

Figure 7: Analysis of tracking measurements: a) distance difference between reported by Emuli and the simulated distance, b) time difference between measurements used for trilateration.

3 Towards Comprehensive Sensor Emulation

In this article we described how sensors can be modeled with Emuli. We illustrated the method using light sensor and range sensor modeling. Even though both sensor models are rather basic, these models can be readily extended to more realistic ones. For example, a range sensor may return values according to distance and statistical distribution. A alight sensor may emulate "digital sunset": an environmental quantity that follows a statistical distribution and gradually changes over time.

There are several directions to further expand the realism of emulated sensors. Note that the data required to instantiate the Emuli sensor model easily fit into flash program memory of a mote. A more sophisticated model may not fit there. Hence, the data for such model needs to be read and possibly buffered from EEPROM. Note also that Emuli does not simulate the time delay incurred by sensor sampling and ADC operation. This can be incorporated in future versions of Emuli. Certainly the greatest asset for Emuli would be a library of emulated sensors that wireless sensor network designers can experiment with.

References

[1] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A line in the sand: a wireless sensor networking for target detection, classification, and tracking. *Computer Networks, Special Issue on Future Advances in Military Communication and Technology*, 46(5):605–634, December 2004.

[2] A. Arora, E. Ertin, R. Ramnath, M. Nesterenko, and W. Leal. Kansei: A high-fidelity sensing testbed. *IEEE Internet Computing*, 10(2):35–47, 2006.

[3] A. Arora, R. Ramnath, P. Sinha, E. Ertin, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, M. Sridharan, S. Kumar, H. Cao, N. Seddon, C. Anderson, T. Herman, C. Zhang, N. Trivedi, M.G. Gouda, Y.-R. Choi, M. Nesterenko, R. Shah, S.S. Kulkarni, M. Aramugam, L. Wang, D.E. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. Project exscal (short abstract). In *Proceedings of Distributed Computing in Sensor Systems, First IEEE International Conference, (DCOSS)*, volume 3560 of *Lecture Notes in Computer Science*, pages 393–394, Marina del Rey, CA, USA, June 2005. Springer.

[4] M.A. Batalin, M.H. Rahimi, Y. Yu, D. Liu, A. Kansal, G.S. Sukhatme, W.J. Kaiser, M. Hansen, G.J. Pottie, M.B. Srivastava, and D. Estrin. Call and response: Experiments in sampling the environment. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, (SenSys)*, pages 25–38, Baltimore, MD, USA, November 2004. ACM.

[5] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. EmStar: a software environment for developing and deploying wireless sensor networks. In *Proceedings of USENIX 04*, pages 283–296, 2004.

[6] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 201–213, Baltimore, MD, USA, November 2004. ACM.

[7] T. He, S. Krishnamurthy, J.A. Stankovic, T.F. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B.H. Krogh. Energy-efficient surveillance system using wireless sensor networks. In *Proceedings of the International Convference on Mobile Systems, Appications and Services (MobiSys)*. USENIX, June 2004.

[8] J. Hill, R. Szewczyk, A. Woo, D. Culler, S. Hollar, and K. Pister. System architecture directions for networked sensors. *ACM SIGPLAN Notices*, 35(11):93–104, November 2000.

[9] J.L. Hill and D.E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November/December 2002.

[10] D. Jia, B.H. Krogh, and C. Wong. TOSHILT: Middleware for hardware-in-the-loop testing of wireless sensor networks, 2005.

[11] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the first international conference on embedded networked sensor systems*, pages 126–137. ACM Press, 2003.

[12] T. Liu, C.M. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrained mobile sensors: Experiences with Impala and ZebraNet. In *Proceeding of the 2nd international conference on mobile systems, applications and services (MobiSys)*, pages 256–269, New York, NY, USA, 2004. USENIX.

[13] L. Luo, T. He, G. Zhou, L. Gu, T.F. Abdelzaher, and J.A. Stakovic. Achieving repeatability of asynchronous events in wireless sensor networks with envirolog. In *25th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. IEEE, 2006.

[14] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 88–97, New York, NY, USA, September 2002. ACM Press.

[15] M. Maróti, B. Kusy, G.Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, (SenSys)*, pages 39–49, Baltimore, MD, USA, November 2004.

[16] C. Park and P.H. Chou. Empro: An environment/energy emulation and profiling platform for wireless sensor networks. In *3rd Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, volume 1, pages 158–167. IEEE, June 2006.

[17] S. Park, A. Savvides, and M.B. Srivastava. SensorSim: A simulation framework for sensor networks. In *Proceedings of MSWiM*, August 2000.

[18] L.F. Perrone and D.M. Nicol. A scalable simulator for TinyOS applications. In *Winter Simulation Conference*, 2002.

[19] J. Polley, D. Blazakis, J. McGee, D. Rusk, J.S. Baras, and M. Karir. ATEMU: A fine-grained sensor network simulator. In *Proceedings of the First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, Santa Clara, CA, USA, October 2004.

[20] G. Simon, P. Völgyesi, M. Maróti, and A. Lédeczi. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In *IEEE Aerospace Conference*, 2003.

[21] A. Sobeih, W.-P. Chen, J.C. Hou, L.-C. Kung, N. Li, H. Lim, H.-Y. Tyan, and H. Zhang. J-sim: A simulation environment for wireless sensor networks. In *Annual Simulation Symposium*, pages 175–187. IEEE Computer Society, 2005.

[22] S. Sundresh, W. Kim, and G. Agha. SENS: A sensor, environment, and network simulator. In *The 37th Annual Symposium on Simulation (ANSS)*, Washington, DC, USA, April 2004.

[23] B. Titzer, D.K. Lee, and J. Palsberg. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 477–482. IEEE, 2005.

[24] G. Tolle, J. Polastre, R. Szewczyk, D.E. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, (SenSys)*, pages 51–63, Sand Diego, California, USA, November 2005. ACM.

[25] Tutornet: A tiered wireless sensor network testbed. `http://enl.usc.edu/projects/tutornet/`.

[26] D. Watson and M. Nesterenko. Mule: A hybrid simulator for testing and debugging wireless sensor networks. In *Second International Workshop on Sensor and Actor Network Protocols and Applications*, pages 67–71, August 2004.

[27] E. Welsh, W. Fish, and P. Frantz. GNOMES: a testbed for low power heterogeneous wireless sensor networks. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 836–389, Bangkok, Thailand, May 2003.

[28] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. In *Proceedings of the Fourth Intrnational Conference on Information Processing in Sensor Networks (IPSN)*, Los Angeles, CA, USA, April 2005.