SENS: A Sensor, Environment and Network Simulator

Sameer Sundresh, Wooyoung Kim, and Gul Agha *
Department of Computer Science
University of Illinois at Urbana-Champaign
201 N. Goodwin, Urbana, IL 61801
e-mail: {sundresh|wooyoung|agha}@cs.uiuc.edu
http://osl.cs.uiuc.edu

Abstract

Recent advances in micro electro-mechanical systems and VLSI lithography have enabled the miniaturization of sensors and controllers. Such minitiarization facilitates the deployment of large-scale wireless sensor networks *(WSNs). However, the considerable cost of deploying and maintaining large-scale WSNs for experimental purposes makes simulation useful in developing dependable and portable WSN applications. SENS is a customizable sensor network simulator for WSN applications, consisting of interchangeable and extensible components for applications, network communication, and the physical environment. Multiple component implementations in SENS offer varying degrees of realism. Users can assemble application-specific environments; such environments are modeled in SENS by their different signal propagation characteristics. The same source code that is executed on simulated sensor nodes in SENS may also be deployed on actual sensor nodes; this enables application portability. Furthermore, SENS provides diagnostic facilities such as power utilization analysis for development of dependable applications. We validate and demonstrate usability of these capabilities through analyzing two simple WSN services.*

1. Introduction

The continuing miniaturization of components by advances in VLSI and MEMS technologies has allowed researchers to synthesize a full suite of computation, communication, sensing and actuation capabilities on a single silicon die [15]. Such advances are quickly bringing to life the idea of large-scale *ad hoc* wireless sensor networks (WSNs), where each embedded node in the network has *in situ* computation capability. A typical node in such networks includes one or more sensors, perhaps some actuators, and wireless communication circuitry, all orchestrated by an embedded microprocessor. Recent research in WSNs has suggested many interesting application scenarios, including monitoring [1, 10], precision agriculture, and structural health monitoring and maintenance of civil infrastructures (buildings, roads and bridges) [9].

Development of WSN applications with performance guarantees is a very challenging task. The small form factor of sensor nodes (*e.g.*, MICA-2 Motes [2]) imposes severe constraints on the availability of resources, such as power, memory, communication range, and sensing capability. Such constraints complicate the development of dependable WSN applications: for example, most WSN nodes limit their radio range to conserve power and reduce packet collision, but limited radio range also increases the effects of location and orientation on the communication error rate [4]. The difficulty of providing performance predictions makes it imperative for developers to test applications thoroughly, preferably by using different sets of parameters in a realistic environment at all phases of the application development cycle. Simulation is a cost-effective choice for prototyping and testing such WSN applications, as the cost, time, and complexity involved in deploying and constantly changing actual large-scale WSNs for experimental purposes are prohibitively high.

In this paper we describe SENS, a simulator for wireless sensor networks. SENS has a modular, layered architecture with customizable components which model an application, network communication, and the physical environment (Figure 1). By choosing appropriate component implementations, users may capture a variety of application-specific scenarios, with accuracy and efficiency tuned on a per-node basis. To enable realistic simulations, we use values from real sensors to represent the behavior of component implementations. Such behavior includes sound and radio signal strength characteristics and power usage. Fur-

*This research has been supported in part by DARPA (F33615-01-C-1097) and in part by Motorola Center for Communications.

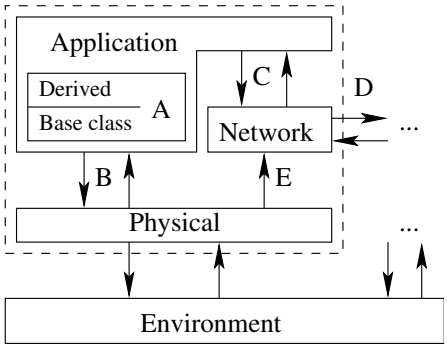

Figure 1. SENS is structured as a graph of components. Dashed box encloses one sensor node.

thermore, SENS is platform-independent: as new WSN platforms are introduced, their parameter profiles can be added to the simulator. The ability to develop portable applications is an important feature, considering that WSN platforms constantly evolve as new sensor node implementations emerge.

Another salient feature of SENS is its novel mechanism for modeling physical environments. WSN applications feature tight integration of computation, communication and interaction with the physical environment. When a node drives its actuator, it may affect the environment and alter network propagation characteristics. Thus, the validity and effectiveness of simulation results depends heavily on how accurately the environment is modeled. To provide users with the flexibility of modeling the environment and its interaction with applications at different levels of detail, SENS defines an environment as a grid of interchangeable tiles. Currently, tile implementations for concrete, grass and walls are available, each of which has different signal propagation characteristics; users may define other tiles to suit their needs.

2. Simulator Structure

SENS consists of several simulated sensor nodes interacting with an Environment component. Each node consists of three components, called Application, Network, and Physical (Figure 1). Each component has a virtual clock; messages can be sent with any delay past the sender's current virtual time. For example, a node's Network component may simulate the reception of two packets which collide and hence are not received, while at the same virtual time the node's Application processes some data. Thus components are isolated and interchangeable; a user may employ any of the implementations SENS provides, modify existing components, or write entirely new ones for custom

Command	Arguments	Serviced by
Send	message, delay, transmission_time	Network
Schedule	message, delay	Application
Sense	sensor_id	Phy, Env
Actuate	actuator_id, command	Phy, Env
Beep	duration, delay	Phy, Env
Sleep	duration, delay	Phy, Net

Table 1. Operations in the `Application` base class (A in Figure 1).

applications, network models, sensor capabilities, or environments.

The user may select various implementations of Application, Network, Physical and Environment components. For example, in early prototyping, one may want to ignore packet collisions; later on, more realism can be added to test for robustness. Similarly, nodes may be configured differently to simulate a heterogeneous sensor network. This can be useful when different nodes have varying capabilities (e.g., in a hierarchy), or to test the effect of introducing updated sensor nodes into an existing network.

2.1. Application Components

An Application component simulates the execution of software on a single sensor node. A node's Application component communicates with its Network component to send or receive packets and with its Physical component to read sensor values or control actuators. For convenience, we supply a C++ base class for Applications, providing the interface depicted in Table 1. Similarly, an Application may receive and act upon any message of the types in Table 2; results are sent back as a message at some later time. Blocking behavior can be simulated by ignoring other messages until the result is received.

Users have two ways to create Applications. First, they may derive a new class from the `Application` class to directly implement an application. However, such a program may not run directly on existing WSN platforms. We have developed a thin compatibility layer to enable direct portability between our simulator and real sensor nodes. When compiling source code intended to run on a real sensor node, we link it with a library which translates sensor node API calls to SENS Application APIs. This technique allows for a SENS target for TinyOS [6], similar to the approach used in TOSSIM [8] and TOSSF [14].

2.2. Network Components

A Network component simulates the packet send and receive functions of a wireless sensor node. All such com-

Message type
Scheduled_mesg(data)
Sensor_value(sensor,value)
Receive_packet(data)

Table 2. Messages sent to Application components (A, B, C in Figure 1).

Element	Description
Signal_strength	Packet signal strength at receiver
Transmission_time	Duration of packet transmission
Sender	Identity of sending node
Message_data	Packet contents

Table 3. Contents of messages sent between Network components (D in Figure 1).

ponents are derived from the `Network` base class, which specifies the basic Network interface. Each Network component is connected to a single Application and the Network components of neighboring nodes. The format of messages exchanged between neighbors is fixed (Table 3) to allow multiple implementations with different characteristics, including the following three.

SimpleNetwork simply forwards messages to neighbors and delivers all messages received to the Application.

ProbLossyNetwork delivers or drops packets based on some error probability. This probability is expressed as a traffic-independent component, p, and a traffic-dependent component based on the recent packet rate. The reasoning is that network congestion leads to collisions and packet loss. Thus, we monitor the number of packets n sent or received in the past Δt seconds. During low-traffic conditions, a packet is randomly assigned an error with probability p. When $n/\Delta t$ exceeds threshold r, packets are assigned transmission errors with probability $1-(1-p)\frac{r}{n/\Delta t}$. Similarly, an outgoing packet is sent to a particular neighbor with probability proportional to the simulated signal strength at the recipient, as computed by the Environment component.

CollisionLossyNetwork calculates collisions between packets at each receiving node. The component monitors each packet over the interval $[t_{\text{start}}, t_{\text{start}} + \text{duration}]$. If a collection of packets are found to have overlapping transmission intervals, only a subset will be delivered, and these may contain errors due to interference. For each packet p, we compute the amount of interference during interval $[a, b]$ as $\sum_{\text{q}} \int_a^b \text{ss}(q,t)\text{d}t$ where $\text{ss}(q,t)$ is the signal strength of interfering packet q at time t. Errors can be simulated by either dropping the packet or introducing random bit errors in the appropriate part of the packet. The granularity of simulated errors can be selected to trade off accuracy and performance.

Message type	Description
Sense(sensor)	Application requests sensor value.
Sensor_value(sensor,value)	Physical's reply.
Actuate(actuator_id,command)	Application requests actuation.

Table 4. Messages sent between Physical and Application components (B in Figure 1).

State	Current
radio send	8.1 mA
radio receive	7.0 mA
speaker on	3.4 mA
microphone on	2.91 mA
cpu active	2.9 mA
sleep	1.9 mA

Table 5. Current drawn by nodes in different states. Based on Table 1 from [13].

2.3. Physical Components

Each simulated node includes a Physical component which models sensors, actuators and power and interacts with the Environment. Initially, each Physical registers its node with the Environment. The Environment then replies with a list of the node's neighbors, along with radio and sound signal strength and delay for each neighbor. During simulation, a node's Physical accepts `Sense(`*sensor_id*`)` and `Actuate(`*actuator_id, command*`)` messages from the Application (Table 4). We currently provide microphone (sensor) and speaker (actuator) devices.

The Physical component also simulates a node's power usage. When the Application or Network enters a different power mode, they notify the Physical by `actuate` messages to turn on or off associated virtual hardware. The current levels we use to calculate power usage in different states are listed in Table 5. For example, when a radio message is transmitted, the Network component sends the duration of transmission to the Physical. The current is simply multiplied by a nominal 3V and power usage accumulated over time.

2.4. Environment Component

The role of the Environment component is to provide a useful model of a real environment with which nodes' sensors and radios might interact. By varying the Environment, developers can test a wide variety of settings at a fraction of the effort of setting up actual experiments. To allow modular, reconfigurable scenarios, an environment is simulated

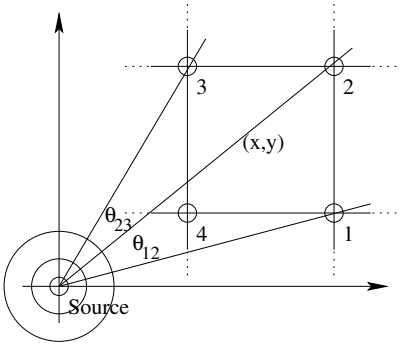

Figure 2. Circular wave propagation through tile in different row and column than source.

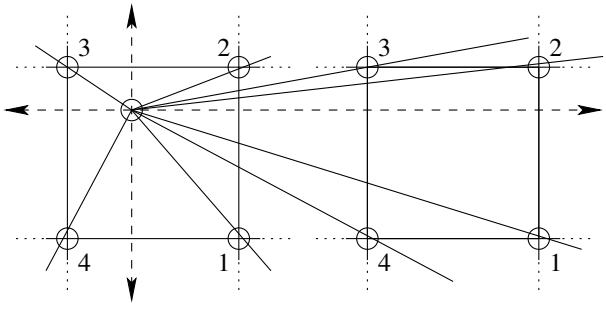

Figure 3. Circular wave propagation from the source, through a tile in the same row.

as a 2-dimensional grid of interchangeable square tiles. This generally models sensors on the ground outdoors. Tiles use experimentally-measured parameters for how radio and sound waves propagate. SENS provides tiles to simulate grass, concrete sidewalks, and walls. Concrete is considered a baseline, and other tiles with greater signal attenuation or delays are called *obstacles.*

The Environment component models circular wave propagation through the 2-D grid of tiles. Since a tile may be anywhere on the map, propagation rules must use only local information about a wave. This information is modified by tiles as the wave propagates, and passed on to neighbors which lie along the propagation paths. Tiles receive and propagate the following information describing (part of) a wave: *(1)* source location (x_s, y_s), *(2)* the amount of energy contained in that part of the wave, and *(3)* the delay profile along the edge through which the wave entered a tile.

Ideal 2-dimensional propagation. We perform 2-D circular wave propagation on a grid of tiles as in Figure 2. Angle $\theta_{13} = \theta_{12} + \theta_{23}$, the angle from the source spanning tile (x, y), determines the total energy that passes through the tile. Angle θ_{23} represents the fraction that passes through tile (x, y) and the tile above, $(x, y + 1)$, while θ_{12} the fraction through tiles (x, y) and $(x + 1, y)$. The situation is similar for tiles containing the source or tiles in the same row or column as the source (Figure 3).

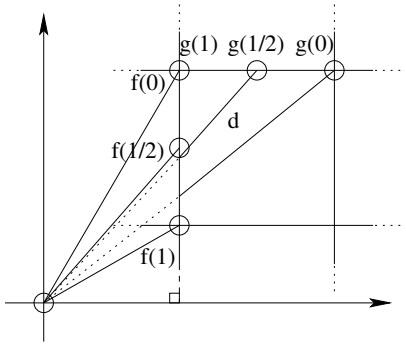

Figure 4. Time-squared propagated along a circular wave via 3 samples per edge.

Measurement. Signal strength (SS) measured by a sensor depends on energy density. This is obtained by dividing the amount of energy traveling through a tile by the arc length over which it is distributed. We approximate this arc by the longest line segment tangent to the circular wavefront, which must end at either corner 1 or 3 of the tile. The length of this segment can be computed as $\langle |\cos\theta|, |\sin\theta| \rangle \cdot \langle 1, 1 \rangle$, or simply $|\cos\theta| + |\sin\theta|$. Thus, when a sensor measures a signal with energy e, we obtain the perceived signal strength s using this *measurement function*:

$$s = M(e) = \frac{e}{|\sin\theta| + |\cos\theta|}$$

Lossy and 3-dimensional propagation. Signals sustain loss traveling through real media. SS at distance r from the source is often a function of $1/r$. Since SS varies with $1/(2\pi r)$ for 2-D circular waves, we can raise the SS to the n^{th} power to achieve order $1/r^n$ attenuation. In particular, we model 3-D spherical waves by propagating the square-root of the actual energy e and using a revised measurement function.

$$s = M(\sqrt{e}) = \pi \left(\frac{\sqrt{e}}{|\sin\theta| + |\cos\theta|} \right)^2 \qquad (1)$$

Obstacle tiles attenuate signals. This is achieved either by directly modifying propagated energy e, or by attenuating measured SS $M(e)$ and then applying M^{-1}.

Simulating delays. Slow-moving signals, such as audio, have a non-negligible propagation time which must be considered in simulation. Since waves may travel along indirect paths, the straight-line distance between source and sensor is not sufficient for simulating delays.

We represent the delay incurred by the part of a signal which crosses each point along a line segment parametrically as (x_u, y_u) for $0 \le u \le 1$, as in Figure 4. When a

Tile	Range	Delay	Attenuation/echo
concrete	28m	0s	1.0/0.0×
grass	14m	0s	0.95/0.0×
wall	8m	0.018s	0.7/0.3×
random error	68cm	0.002s	–

Table 6. Sound propagation parameters.

Maximum range	100m
Directional variation	16%
Attenuation through wall	0.80×
Packet rate	14/s

Table 7. Radio propagation parameters.

signal propagates from tile a to adjacent tile b, we pass the delays at three points along their common edge in addition to energy. Tile b, in turn, uses the delay values along its input edges together with the speed of propagation through its body and angle of propagation (from the source) to determine the delay samples along each of its output edges. When a signal encounters an obstacle, it may be delayed and attenuated; after this, it may either be re-emitted along its original course or a different direction. The first case is handled by using a slower propagation speed, while the second by generating a new wave with a non-zero initial delay. To maintain good performance, we only allow one generation of secondary waves.

Tiles and parameters. We have used MICA-2 wireless sensor nodes [2] placed $10\mathrm{cm}$ above the ground to measure parameters of the environment and the sensor node. Tables 6 and 7 summarize measurements and simulation parameters used to model them. A maximum communication range of r is implemented using an initial SS of $\sqrt{4\pi r^2}$. Thus SS measured using Equation 1 at distances beyond r are less than a detection noise floor of 1.0. To achieve the lower distances for grass and wall tiles, we introduce attenuation factors; each time part of a wave passes through such a tile, its energy is multiplied by the corresponding attenuation factor. For walls, initial waves also cause echo waves with a listed SS and a new source reflected across the wall. Since nodes may measure sound travel times, we added a uniform random error of up to 0.002s to sound event arrival times.

3. Simulation Examples

Now we present two example applications to illustrate the features of SENS. We created several random environments, ranging from 0% to 100% obstacle tiles (grass, walls), with the remaining tiles concrete.A random environment with 25% obstacles is depicted in Figure 8. All simulations were run using `CollisionLossyNetwork`.

3.1. Spanning Tree

Our first example is a service which generates a (partial) spanning tree via flooding. The service is initiated at time 0 by a root node in the middle of a $400\mathrm{m} \times 400\mathrm{m}$ region containing 1000 nodes. The root broadcasts a single spanning tree message containing its ID. When a node receives such a message, it reads the sender's ID, stores it as its parent, broadcasts a new spanning tree message containing its own ID, and goes to sleep.

Figure 5 shows how spanning tree coverage varies with obstacle density. The coverage peaks at 914 of 1000 nodes for 25% obstacles. It then drops quickly as we add more obstacles, as some nodes become entirely cut off from the root. Tree coverage also decreases with obstacle density. As it turns out, the problem under very low obstacle density is collisions (Figure 5); obstacles decrease radio range, and hence increase usable network capacity until the point that the network is partitioned. Note that ollisions greatly outnumbered (broadcast) messages sent because they were counted at each receiving node.

3.2. Simplified Localization

Many sensor network applications need location information to correlate measurements. We use the simulator to study a simple localization service based on acoustic ranging. Simulations were performed in a $50\mathrm{m} \times 50\mathrm{m}$ environment with 6 anchor nodes and 200 non-anchors (Figure 8). Anchors are nodes with known locations; all others are non-anchors with unknown locations. Anchors periodically broadcast their ID and location over the radio, immediately followed by a $0.1\mathrm{s}$ beep. When a non-anchor receives such a radio message, it measures the delay until it hears the beep to estimate the distance to the anchor. Furthermore, non-anchors take the median of the past 10 measurements from an anchor to filter out erroneous data. To simplify the example, we made non-anchors estimate their location by averaging anchors' locations, weighted by the inverse of the approximate distance to each anchor:

$$(x, y) = \frac{1}{r_1}(x_1, y_1) + ... + \frac{1}{r_n}(x_n, y_n) \qquad (2)$$

Figure 6 compares simulated to experimental sound propagation times. An anchor node is adjacent to a $1\mathrm{m}$-high wall. The wall creates in indirect sound path, resulting in longer propagation times to the left. Discrepancies are due to both random error and difference in temperature and humidity between simulator calibration and the experiment.

A node is considered *successfully localized* if it has computed its own location to within $10\mathrm{m}$ (the large tolerance compensates for the error in Equation 2). Figure 7 shows

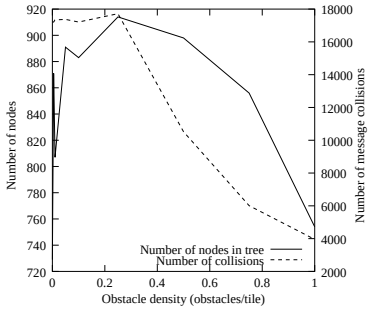

Figure 5. Spanning tree: numbers of nodes, collisions.

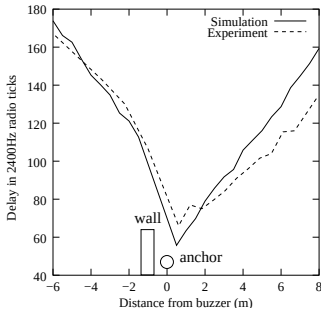

Figure 6. Sim. vs experiment: sound propagation delay.

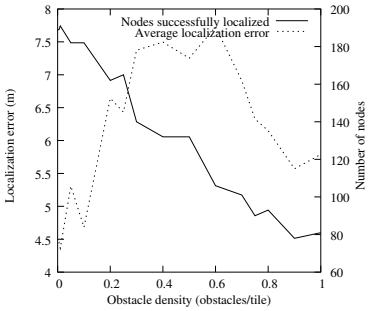

Figure 7. Nodes successfully localized, average error.

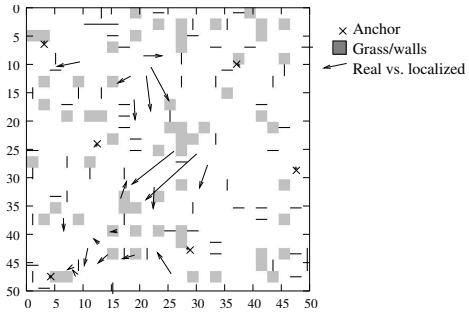

Figure 8. Example random map; arrows point from real to localized non-anchor positions.

that the number of successfully localized nodes varies inversely, roughly linearly with obstacle density. This linear decline indicates that anchor-based acoustic ranging has fairly predictable behavior with respect to obstacles, as opposed to the accelerating drop-off in spanning tree coverage.

A view of the environment, anchor nodes and a subset of 20 localized non-anchor nodes is presented in Figure 8. Nodes with several barriers between themselves and any anchor tend to have worse errors. Furthermore, localized positions are primarily skewed towards those anchors to which they have a more direct path. This is because direct paths appear shorter than indirect paths obstructed by walls and provide a stronger signal than those where sound signals are attenuated by grass. In general, the component structure of SENS allows users to quickly run and visualize their applications like in the figure because obstacles, network models, etc. can simply be plugged in rather than re-written.

The decrease in localization error for large obstacle densities in Figure 7 may seem surprising. However, based on the low number of nodes successfully localized, it is apparent that for high obstacle densities, nodes either have relatively direct sound paths to anchor nodes or cannot hear them at all.

3.3. Power Usage

We now consider the power usage of the previous example, using parameters from Table 5. Rather than per-node power usage, we look at per-node *effective* power usage, i.e., power divided by the number of successfully localized nodes. Figure 9 depicts how efficiently power was spent (anchors were omitted as their power usage was near minimum). Note that average power usage closely follows the maximum, while the minimum falls behind. This indicates that a small number of nodes are entirely cut off from the anchors for high obstacle density, while many nodes receive an anchor's radio messages but time out without detecting the tone. Thus when a non-anchor can receive radio messages but consistently does not hear sounds from a given anchor, it should ignore that anchor and sleep rather than listening. Figure 10 shows the improved power usage for this blacklisting policy: minimum, average, and maximum power usage are all decreased, and the average follows the minimum more closely than the maximum.

3.4. Simulator Performance

To be useful, a simulator should offer a strong performance advantage over setting up real sensor networks. Figure 11 shows that this is the case for our simulator. The figure depicts runs of the simplified localization with varying numbers of randomly positioned nodes. For n nodes, we used a $\sqrt{n} \times \sqrt{n}$ environment with $n/16$ anchors and 50% of the tiles assigned as obstacles. For a network of 8192 nodes, we simulated 1000 virtual seconds of the application in only 136 seconds of real time on a 2.5GHz Pentium 4 with 512MB RAM running Linux 2.4.20; of this, 124 seconds were spent on environment initialization, with a mere 12 seconds dedicated to actual execution. The performance ratio would improve for longer simulation runs as we amortize the cost of environment initialization, which can be quadratic in the number of nodes. Note that applica-

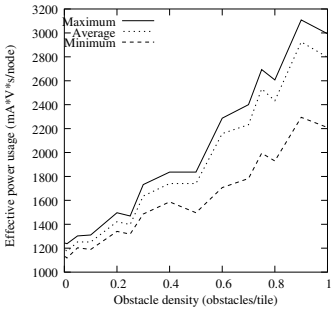

Figure 9. Effective power usage by non-anchors.

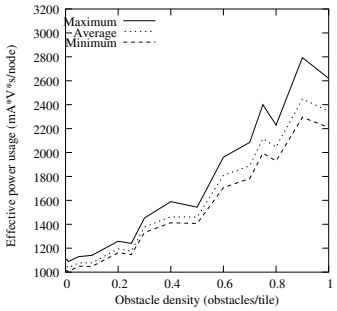

Figure 10. Effective power usage with blacklisting.

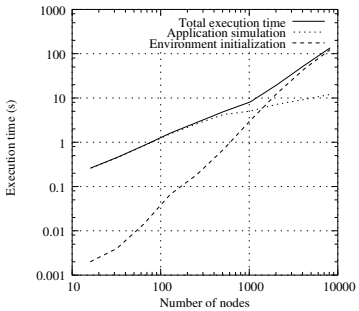

Figure 11. Real time for 1000 sim. seconds of localization.

tion simulation time increases more slowly past 512 nodes. This corresponds to a $22\mathrm{m} \times 22\mathrm{m}$ environment, which is close to the maximum sound range used, so node degree stops increasing.

4. Related Work

Arguably, the most straightforward way to develop a simulator for a WSN application is to build it from the ground up to exploit application-specific details. Although not efficient in terms of development effort, application-specific simulators abound in the WSN literature. We feel that scarcity of flexible simulation frameworks with appropriate architectural support contributed to this proliferation.

Apart from the "do-it-yourself" class of simulators, there are a few widely adopted network simulators, including OPNET [11] and *ns-2* [19]. Wireless capabilities have been addressed by extending existing simulators, as in the Monarch project [18, 17], or specifically building new ones, such as GloMoSim [5]. These simulators focus on protocols and algorithms for layers of the network stack, but do not directly support sensor networks.

Recently, several simulation frameworks have emerged to specifically address WSNs. These range from extensions of existing tools to application-oriented simulators. Although these frameworks have some shared objectives (e.g., a notion of environment), they visibly differ in design goals, architecture, and extent of abstraction provided for applications.

At one end of the spectrum are simulators that focus on wireless protocol stacks for sensor networks, including UCLA's SensorSim [12] and Georgia Tech's SensorSimII [20]. [1] UCLA SensorSim extends *ns-2*, providing power and communication protocol models and support for hybrid simulations which interact with real wireless sensors. Georgia Tech SensorSimII organizes a sensor node into three components, application, network, and link; sensors directly interact with applications. Aside from SensorSim's notion of power, neither takes into account tight resource constraints on sensor nodes, limiting the realism of results. Furthermore, many applications do not need detailed protocol stack simulation to ensure expected behavior; rather, validation of functional correctness and performance guarantees is necessary at the WSN level.

TOSSIM [8] is a discrete-event simulator for TinyOS [6, 16] applications on MICA Motes [2]. It aims to facilitate source-level application and OS debugging, so that programs can be directly targeted to Motes without modification. TOSSIM assumes all nodes share the exact same code image, simulates at bit granularity, and assumes static node connectivity known a priori (modulo a very simple form of node mobility). In these regards, TOSSIM is more of a TinyOS emulator than a general WSN simulator. TOSSF [14] can be viewed as a scalable version of TOSSIM. TOSSF is an adaption of SWAN, a simulator for wireless ad hoc networks. In particular, it allows a heterogeneous collection of sensor nodes and dynamic network topology. Since both of these simulators are tightly coupled with TinyOS and Motes, they may be inappropriate for early prototyping or developing portable WSN applications.

At the other end of the spectrum are application-oriented simulators, including SENS, Siesta [7] and EmStar [3]. Siesta has a modular, layered architecture, with the physical system model and application as pluggable components interacting via sensors and actuators. However, the latest release is geared towards application and middleware component simulation, and does not provide adequate flexibility to model an underlying wireless network. Communication capabilities are encoded point-to-point between neighbors, and network characteristics such as communication delay and error rate are not supported. EmStar is a framework for developing applications on WSNs, encompassing pure simulation, distributed deployment on iPAQs, and a hybrid simulation mode similar UCLA SensorSim. As in TOSSIM,

[1] Despite the similar names, these are actually independent projects.

simulation is at the source level so that code need not be modified for each mode.

5. Conclusion

We have presented SENS, a Sensor, Environment and Network Simulator. SENS features a modular architecture to permit simulation of a range of different WSN scenarios. In particular, we have implemented components to support sensor nodes communicating via wireless broadcast in an environment represented by tiles which modulate sound and radio propagation. We have demonstrated the utility of this approach in analyzing the behavior of WSN applications under a variety of different environmental scenarios.

Simulation of WSNs is an ongoing effort. As such, each of the components of the SENS architecture have avenues for enhancement. Applications would benefit from automatically-generated timing information. The network stack could be made more flexible by implementing higher-level protocol abstractions.The Physical component's power model could be improved to include a realistic battery model and its effects on component behavior, which will improve confidence in the long-term viability of a WSN. Simulations could be sped up by caching environment behavior rather than re-computing it for each run. Furthermore, environment simulation could be enhanced by characterizing a wider variety of objects, allowing 3-D maps, and supporting wave phenomena such as diffraction. Finally, application development using SENS will certainly reveal further opportunities for improvement. We expect that the SENS style of system simulation will help facilitate the development of robust WSN applications.

SENS is available on the WWW at:
`http://osl.cs.uiuc.edu/sens`

References

[1] A. Cerpa, J. Elson, D. Estrin, and L. Girod. Habitat Monitoring: Application Driver for Wireless Communications Technology. In *Proceedings of the ACM SIGCOMM Workshop on Data Communications in Latin America and the Carribean*, pages 20–41, 2001.

[2] Crossbow Technology, Inc. MICA: Wireless Measurement System, 2003. http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA.pdf.

[3] 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. Technical report, Center for Embedded Networked Sensing, University of California, Los Angeles, 2003. CENS Technical Report 009.

[4] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. An empirical study of epidemic algorithms in large scale multihop wireless networks. Technical report, Intel Research, 2002. IRB-TR-02-003. http://www.intel-research.net/Publications/Berkeley/120520021022_19.pdf.

[5] GloMoSim. http://pcl.cs.ucla.edu/projects/glomosim/.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System Architecture Directions for Network Sensors. In *Proceedings of ASPLOS 2000*, 2000.

[7] A. Ledeczi, M. Maroti, and I. Bartok. *Simple NEST Application Simulator (Siesta 02-05-02)*, October 2001. Draft. http://www.isis.vanderbilt.edu/projects/nest/downloads.asp (part of the latest Siesta distribution).

[8] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003. (to appear).

[9] J.P. Lynch, K.H. Law, A.S. Kiremidjian, E. Carryer, T.W. Kenny, A. Partridge, and A. Sundararajan. Validation of a wireless modular monitoring system for structures. In *SPIE 9th Annual International Symposium on Smart Structures and Materials*, March 2002.

[10] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the First ACM Workshop on Wireless Sensor Networks and Applications*, pages 88–97, 2002.

[11] OPNET. http://www.opnet.com.

[12] S. Park, A. Savvides, and M. B. Srivastava. SensorSim: A Simulation Framework for Sensor Networks. In *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000)*, pages 104–111, 2000.

[13] S. Park, A. Savvides, and M. B. Srivastava. Simulating Networks of Wireless Sensors. In *Proceedings of the 2001 Winter Simulation Conference*, 2001.

[14] L.F. Perrone and D.M. Nicol. A Scalable Simulator For TinyOS Applications. In E. Yücessan, C.-H. Chen, J.L. Snowdon, and J.M. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference*, 2002.

[15] Smart Dust Project. http://robotics.eecs.berkeley.edu/~pister/SmartDust/.

[16] The Berkeley Wireless Embedded Systems Project. http://webs.cs.berkeley.edu/, 2003.

[17] The CMU Monarch Project. The CMU Monarch Project's Wireless and Mobility Extensions to *ns*, August 1998. Release 1.1.0 Beta.

[18] The CMU Monarch Project, 2003. Moved to http://www.monarch.cs.rice.edu/.

[19] *ns-2*. http://www.isi.edu/nsnam/ns/.

[20] C. Ulmer. Wireless Sensor Probe Networks - SensorSimII, 2000. http://users.ece.gatech.edu/~grimace/research/sensorsimii/.