The Design, Deployment, and Analysis of SignetLab: A Sensor Network Testbed and Interactive Management Tool

Riccardo Crepaldi[†], Simone Friso[†], Albert Harris III[†], Michele Mastrogiovanni[§]
Chiara Petrioli[§], Michele Rossi[†], Andrea Zanella[†], and Michele Zorzi[†]
[†]Department of Information Engineering, University of Padova, Italy
[§]Department of Computer Science, University of Rome "La Sapienza," Italy
{riccardo.crepaldi,simone.friso,harris,rossi,zanella,zorzi}@dei.unipd.it, {mastrogiovanni,petrioli}@di.uniroma1.it

***Abstract*—The emergence of small, inexpensive, network-capable sensing devices led to a great deal of research on the design and implementation of sensor networks. A critical step in taking protocols from theory to actual deployment is comprehensive testing on physical sensor networks. Sensor network testbeds provide one way to facilitate such testing without requiring the deployment of a specialized sensor network for each protocol. However, for such testbeds to be useful, they must not overwhelm researchers with maintenance tasks and high learning curves.**

Previous work in testbed design has primarily focused on creating interfaces to maximize their usage by convenient scheduling of jobs and output access. In this work, we present two contributions to sensor network testbed design. The first is a unique management tool that allows users to program, interact with, and receive data from nodes in the network, filling a gap in current testbed management solutions. The second is the design, deployment, and analysis of the SignetLab testbed. The analysis of the testbed and its results provide quantitative measurements of the impact of physical deployment on signal propagation characteristics. Additionally, we present two case studies where researchers have used the testbed and discuss the user experiences and lessons learned.

I. Introduction

The ability to manufacture small, inexpensive computing devices with wireless networking capabilities has led to a large amount of research in the area of sensor network protocol design. The vision of large, self-forming networks of small devices, each equipped with sensing hardware to monitor an environment (*e.g.*, a battlefield or disaster zone), requires the design of communication protocols that are highly scalable (to thousands of nodes), loss-tolerant (the devices can be unreliable and prone to failure), and energy-efficient (sensor devices operate on batteries that are not easily replaceable). A key difficulty in designing protocols for this type of system is the lack of appropriate methods of testing. There are a number of simulation options available (*e.g.*, the ns2 network simulator [1]), but each of these necessarily hides real-world effects (fading properties, anisotropic propagation, etc.).

To address this limitation, a number of testbed solutions have been proposed very recently. The primary focus of these early testbeds has been either on sensor node design [2], [3], [4], [5] or on tools to allow users to timeshare the network [6], [7]. One feature that these works cite as a future goal is a tool that allows users to have fine-grained control of experiments during their timeslot as well as real-time feedback from the network [6]. The first contribution of our work is the development of such a tool, which provides a simple interface through which to program, interact with, and receive data from the sensor nodes that does not rely on any single technology or operating system, such as TinyOS [8].

Software tools are only a partial solution to the problem, however. First, appropriate hardware must be chosen that supports functionalities favorable to protocol testing (*e.g.*, supporting various sensing capabilities and data aggregation). Second, a physical space for the deployment of the network must be chosen to approximate realistic sensor deployments (*e.g.*, is the space large enough and how does it affect the results). Third, a backplane must be chosen to allow data collection and node monitoring for the sensor network without interfering with the wireless traffic.

Our second contribution is the design, deployment, and analysis of SignetLab, our sensor network testbed. SignetLab is composed of 48 EyesIFXv2 nodes [2], a USB data backplane, and is supported by a software tool that allows node selection, visualization, and network programming and debugging. It is deployed in the Signet research lab in the University of Padova, DEI building. SignetLab is actively used by researchers at the University of Padova as well as the University of Rome. The analysis of the testbed provides insight into the impact of physical deployment on its functionality. This analysis and its results will aid other researchers in designing and deploying their own testbeds.

We conclude this paper with the presentation of two case studies, where researchers have used our testbed to test deployments of their protocols and applications. We discuss their experiences in using the testbed and in expanding and using the software tool via the provided API. We follow the case studies with a discussion of user experiences and lessons learned.

This material is based upon work partially supported by the MIUR International FIRB RBIN047MH9.

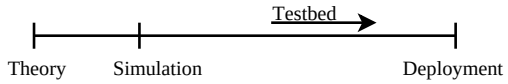

Fig. 1. From Theory to Deployment

II. Related Work

Research progresses from theory to deployable systems (see Figure 1). While simulations are an important step towards deployment due to their ability to provide repeatability of experimentation, simulation environments typically ignore the effects of some real world network properties (*e.g.*, anisotropic links and lack of time synchronization). However, building deployments to test protocols is prohibitive in terms of difficulty, cost, time commitment, and repeatability of experimentation. Therefore, a middle-ground is needed: the testbed. Testbeds aim to provide most of the real characteristics of a deployment while maintaining some experimental repeatability and providing rapid testing and prototyping capabilities.

There have been a number of attempts to create testbeds maximizing usability and realism. Some work focuses on mobile ad hoc networks [9], [10]. Typically, these solutions are targeted at dealing with issues specific to supporting mobility. These networks also tend to focus on IEEE 802.11 [11] wireless technology. Other testbeds focus on the analysis of protocols addressing specific problems (*e.g.*, power control [3], [5] or mobility [4]).

The two works that are most relevant to our testbed are MoteLab [6] and Mobile Emulab [7], [12]. Both are aimed at maximizing testbed utilization among different users. To this end, they provide a web interface through which users can schedule jobs.

MoteLab gives the users access to the nodes to do realtime data analysis via the TinyOS serialforwarder [8]. Data is stored in a mySql database [13] as well. Mobile Emulab focuses on providing users remote access to a testbed that supports mobility. Both MoteLab and Mobile Emulab strictly limit the users' ability to interact with the node reprogramming phase. A desire has been expressed for a tool that would give users fine-grained control during their timeslots [6]. Our software tool provides this capability and is presented in Section IV-B.

In addition to full testbed solutions, some tools have been developed that provide similar functionality to some of the plugins included with our tool. Marionette [14] provides a library of hooks to be embedded on a sensor device allowing users to remotely interact with the application executing on the node. Essentially applications are compiled with the embedded Marionette libraries and then, using remote procedure calls (RPC), clients can interact at runtime with the nodes. Marionette provides a method to execute instructions on nodes similar to the capability provided by TinyOS; however, it works over the wireless link, interfering with the network operations. Marionette could be integrated into our SignetLab tool, if desired.

TinyOS has a utility called the Message Center [8] that provides a graphical user interface to the serial forwarder to send and receive messages between a PC and nodes in the network. One of the plugins for our tool provides a similar interface, but our tool is not limited to TinyOS, only using it as an example.

In addition to testbed projects, there have also been a number of research sensor network deployments [15], [16], [17], [18]. Each one of these demonstrates problems that are encountered in terms of hardware failure, anisotropic signal propagation, and many other properties of physical networks. It is important for our testbed to replicate any of the real properties of these sensor networks to maintain a high level of realism. The next section presents the exact goals of our design, which are directly driven by observations made about deployed sensor networks and current testbed solutions.

III. Goals For a Testbed

The first step in designing a sensor network testbed is to identify specific goals the testbed should achieve. We have identified seven primary goals.

First goal: the testbed should improve research productivity. This is the fundamental goal of our design. If it is more time-efficient for researchers to build their own sensor network deployments for protocol experimentation, then the testbed would be a failure. This first goal leads immediately to the next two goals.

Second goal: the testbed should be easy to maintain. Normally, research groups do not want to hire a full-time network administrator; therefore, the researchers themselves must handle management duties. If these duties are too time consuming, research will end up being stifled rather than augmented.

Third goal: the testbed should provide a minimal learning curve to be useful for researchers. Therefore, the tool should have a convenient graphical user interface and be easily customizable.

Fourth goal: utilization of the testbed should be maximized across groups of researchers. It should be easy for them to use the network, whether physically present or not. An effective time-sharing tool should be provided to allow various researchers the ability to schedule experiments. We do not directly address this goal in this paper because there are previous solutions providing web-based, timesharing utilities [6], [19].

Fifth goal: the testbed should support a large variety of protocol experiments. There are a large number of sensor network scenarios for which protocols can be designed (dense networks, sparse networks, multihop networks, etc.). Experimentation for protocols on any number of these scenarios should be possible with minimal reconfiguration.

Sixth goal: the testbed should be deployable in a reasonable setting. One that requires 100,000 square meters would not be practical.

Seventh goal: the testbed should provide as close to a realistic environment as possible, while still allowing realtime monitoring of the protocols running on it. Clearly, if the environment is too sterile (*e.g.*, engineered to minimize external

interference) it will not provide the bridge between simulation and deployment for which it was meant.

It is clear that some of these goals can be conflicting. For example, it may be desirable to deploy the network in a lab; however, the conditions in the lab are almost certainly not similar to those in a large outdoor space. Therefore, some design decisions must be made to accommodate these trade-offs. The following section describes the design choices that we made in the implementation of SignetLab.

IV. SignetLab

SignetLab is a sensor network testbed deployed at the University of Padova. In its design, we followed a two pronged approach: design of the physical deployment and design of the software tool. We elected to make the tool as independent from the physical deployment as possible. This will allow the testbed to grow and change without the need to re-implement the software. Also, this decision allows other labs to easily make use of our tool without the need to replicate the hardware used in SignetLab. The following subsections describe each part of SignetLab in detail.

A. Testbed Hardware

The choice of hardware for the sensor network testbed needs to support a number of goals. First, the radio should provide sufficient range and power settings to allow the testing of a variety of protocols. Second, the nodes must provide a means to alter their sensing capability in order to provide support for a variety of applications. Third, the processor on the nodes should provide sufficient computational resources to allow the execution of interesting protocols and applications while still being realistic for a sensor node. Finally, there should be a reasonable way to get realtime status and debugging information from the testbed without interfering with the execution of the main application.

1) Deployment Space: SignetLab is deployed in a 10m x 11m lab due to space limitations at the University of Padova. Our deployment is on a grid suspended 60cm from the ceiling and 2.4m above the floor. In this way, the lab is not overtaken by the sensor network deployment. The network is made up of 48 EyesIFXv2 nodes [2], separated by 160cm in one direction and 120cm in the other direction. These distances were chosen to provide a uniform distribution in the lab.

2) Sensor Nodes: The EyesIFXv2 nodes were developed during a three year European research project on self-organizing energy-efficient sensor networks [2]. The nodes use an ultra-low power MSP430 processor with 10 KB on chip RAM, 48 KB flash/ROM, and an additional 512 KB serial EPROM.

The radio chip is a low power FSK/ASK transceiver, providing half-duplex, low data rate communication in the 868 MHz ISM band. It operates using FSK modulation, with a sensitivity of < -109 dBm, enabling up to 64 Kbps, half-duplex, wireless connectivity.

The platform is also equipped with an on-board stripline antenna and an SMA-connector for an external antenna. The external antenna is the default. The onboard antenna can be selected by soldering a resistor into the correct location on the chipboard. However, using either of the available antennae created a radio range, even at the lowest transmit power setting, which reduced the testbed to a one-hop network. One option was to use a low-gain setting at the receiver; however, this does not decrease the interference range of the transmitters. Therefore, this is not an acceptable option. SignetLab uses home-grown, low-gain antennae inserted into the external antenna plug to provide transmission ranges that require the use of multiple hops. Section V presents the performance of the testbed with various transmit power settings.

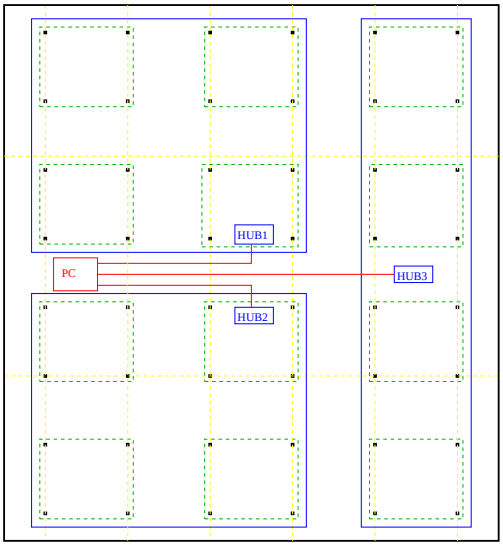

Fig. 2. SignetLab Node Map

The transceiver can accepts a supply voltage of 5.5 V. The typical current is $I_s = 9$ mA in receive mode, and $I_s = 12$ mA in transmit mode. The transmit power can be modulated by means of a digital potentiometer with 255 settings (although only 180 to 255 produce useful transmit power level variations).

The nodes come with onboard temperature and light sensors as well as an SPI expansion port that can be used for additional sensing capabilities. The SPI bus is shared between the expansion port, the radio, and the processor. Therefore, there is a hard restriction on the amount of resources used at a time.

The nodes can be powered either by batteries with a capacity of 1000 mAh or through a power supply connected via an external polarized connector or a USB connection.

3) Backplane Connections: So that debugging and data gathering do not interfere with the operation of the testbed, we provide a backplane using USB connections. These same USB connections are used to supply power to the nodes; therefore, only a single cable is required to connect each node. Figure 2 depicts the backplane architecture, which is composed of two tiers of hubs. Each of the hubs (15 in all) has its own power supply. The dashed squares represent the second tier hubs,

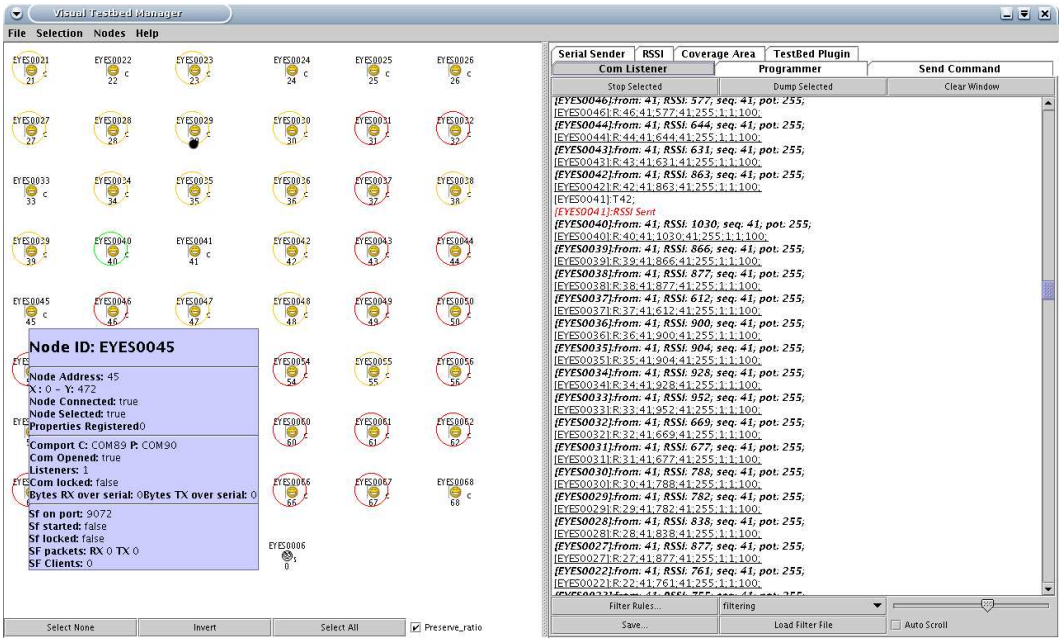

Fig. 3. Application Main View

each of which connects four sensor nodes. The solid rectangles represent the three first tier hubs, each connecting four second tier hubs. The first tier hubs are connected directly to the controlling PC.

One of the driving factors in this layout was the fact that USB cable lengths could not be greater than 5 m, in order to keep transmission error rates sufficiently low. This is due to insufficient power at the hubs to transmit signals that can be accurately decoded over long distances.

B. Software Tool

The software tool of SignetLab was designed to support a number of goals. It should provide a single programming interface to all users that is intuitive to use (*i.e.*, small learning curve). It should be supported on multiple operating systems to allow users to easily integrate it into their own work environment. The tool should also support multiple physical sensor network testbeds (*i.e.*, different node technologies, different node layouts, etc.). Programming nodes (either all or some subset), including compiling and uploading code, should be simple and automated, giving the user as much control as possible during their use of the testbed. Finally, it should be easy for users to add functionality to the tool. We point out that our software tool does not have a component installed on each node and does not rely on TinyOS. We present a number of plugins for TinyOS as examples to demonstrate the tool's use without loss of generality. The software tool is freely available under the LGPL license on the Signet group website (`http://www.dei.unipd.it/research/signet/`).

1) Main Application Window: The SignetLab software tool is a Java application and a set of configuration files that set up the environment. When the application starts up, the main window is split into two sections, the GUI node selection pane on the left and the plugin pane on the right (see Figure 3).

The GUI node selection pane reproduces the topology of the network as specified in the topology configuration file. The user is able to select the entire set of nodes or any subset of nodes by either clicking on the nodes, dragging a bounding box around them, or using the selection menu. Once nodes are selected, various plugins can be used to program the nodes and begin code execution.

The plugin pane contains various plugins and their interfaces (described in Section IV-B3). Users can easily expand the capabilities of the application by using a simple API to write their own plugins.

2) Configuration Interface: The SignetLab software tool is easily customizable by each user through the use of configuration files. The main configuration file defines paths for various utilities (*e.g.*, Perl) that are needed by the application.

The topology configuration file is used to input the physical topology of the network. Through the use of this file, it is trivial to connect the application to different testbeds, or use subsets of nodes for different testbed configurations.

Finally, the plugin configuration file allows users to choose which plugins are displayed in the plugin pane (see Figure 3).

3) Plugins: The application has four main plugins to allow interactions with the testbed: the `ComListener` plugin, the `Programmer` plugin, the `SendCommand` plugin, and the `SerialSender` plugin.

The `ComListener` plugin: The `ComListener` plugin provides the backplane functionality that allows realtime debug and trace information to be collected from the nodes without interfering with the wireless traffic. Essentially, the `ComListener` listens on the USB backplane and provides analysis and viewing functionality. The plugin is attached to

one or more nodes and begins collecting data, which is passed to the *analyzer thread*. This thread performs data filtering, writes the data out to log files, and prints the data to the screen. In order to avoid causing the plugin to consume 100% of the CPU, the analyzer thread is run at low priority and buffers up to 10,000 lines of output for realtime debugging (configured with a slide bar).

The `analyzer thread` allows filters to be defined to make the debugging information be presented on screen in a more useful manner. Filters can be written as regular expressions and include output formatting and highlighting and background coloring (see Figure 3). This allows users of the tool to format the output in a way that facilitates rapid debugging of the experiments running on the sensor network.

The `Programmer` Plugin: The `Programmer` plugin implements the node programming interface. It currently supports nesC [20] applications but is easily expandable to support other languages. The user selects a *Makefile* and then uses the compile button to build the application. A pane is provided to show compile-time output so the user can debug if necessary. Finally, when the application has been built, it can be installed on any subset of nodes by highlighting them in the GUI pane and clicking install. Any installation errors are displayed in a dialog window. This plugin can also be used to reset or erase selected nodes, using the appropriate command buttons.

The `SendCommand` Plugin: The `SendCommand` plugin implements the ability to send commands to the nodes using TinyOS's *Active Message* header format [8]. We designed a structured message format to support the control of nodes. Different commands are described by the entries in a configuration file according to the following simple format: `COMMAND/EDITABLE` *`cmd_name cmd_num p1 p2 p3 p4 p5`*. The first keyword specifies if the user is allowed to modify the parameters of the command before sending it. This is followed by the name of the command and a number identifying it. Finally, there is a parameter list.

The `SerialSender` Plugin: The `SerialSender` plugin also implements the ability to send commands to the nodes; however, it merely provides a byte stream for the communication. The interface is extremely simple, allowing the user to select a node and enter any ASCII string. When the enter key is pressed, the message is sent. Alternatively, the user can send messages to groups of nodes.

V. Analysis of SignetLab

Analysis of the testbed in terms of the environment it provides for protocol experimentation yields insight into the best practices for testbed design and deployment. The fundamental tunable parameter that alters the sensor network environment is the potentiometer setting that adjusts the transmit power of the nodes. This setting determines the distance each node can reach, subject to additional propagation and environmental phenomena (*e.g.*, multipath fading). Analyzing the effects of different potentiometer settings shows the range of environments that the testbed can provide.

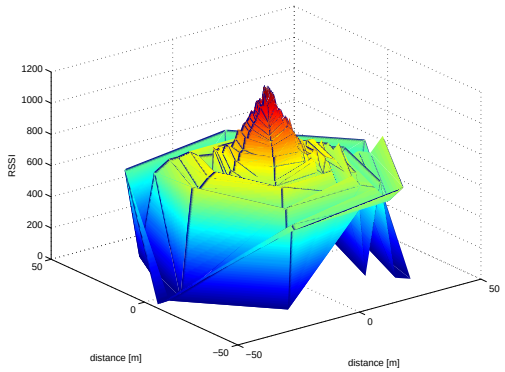

Fig. 4. Outdoor Transmission Pattern

We define two metrics to analyze the testbed. Consider the signal propagation from a single sensor node for a given transmit power level. Theoretically, in the absence of any interference or reflections, the area where the signal is received at greater than some strength, x, would define a circle. However, in real physical environments, there are a number of factors that alter this perfect circle. Figure 4 depicts the received signal strength versus relative node position. The sender and receivers in this case were placed outdoors, in an environment with no trees, buildings, or other close obstructions. A horizontal slice of this graph at a given signal strength, x, does not describe a perfect circle, although it is close. For an indoor environment, the contour resulting from such a slice would, in general, be very different. Our two metrics are defined by inscribing and circumscribing circles for each of these signal strength slices in the graph. We define the *greatest continuous distance reached* as the radius of the inscribed circle, which is the distance inside which the average received signal strength is guaranteed to be greater than x. We define the *farthest distance reached* as the radius of the circumscribed circle, which is the distance outside which the average received signal strength is guaranteed to be less than x. Instead of using received signal strength as our metric to slice the graph, we use percentage of packets received, which is essentially the same, as a received signal strength can always be translated to a probability of packet error and vice versa.

To demonstrate the impact on distance reached of the choice of the percentage of packets received that defines x, Figure 5 shows the greatest continuous distance reached and the farthest distance reached with 95%, 90%, 80%, and 30% packet reception. This shows that the performance of the network with respect to the farthest distance is not very sensitive to the definition of reachability in terms of percentage of packets received. The continuous distance is more sensitive because a single node in a region of poor signal quality will reduce the radius of the inscribed circle defining this value.

Using these metrics, we analyzed the characteristics of SignetLab in terms of number of hops required to traverse the network (demonstrating its ability to support a variety of sensor network scenarios, including significant multihop behavior) and in terms of the propagation characteristics of

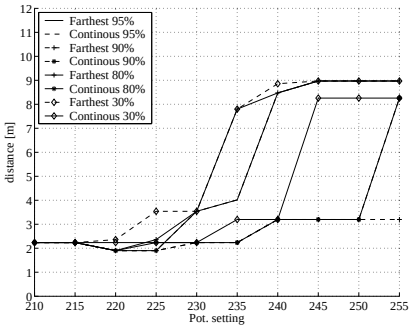

Fig. 5. Impact of Reachability Definition

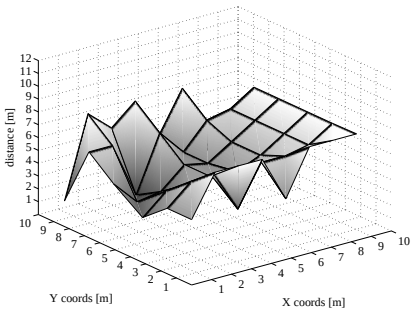

Fig. 6. Greatest Continuous Distance Reached, High Potentiometer

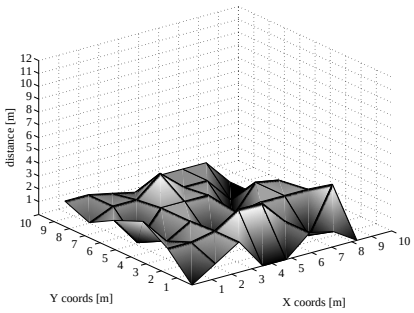

Fig. 7. Greatest Continuous Distance Reached, Low Potentiometer

different nodes in the network. This analysis gives insight into the effects of node placement on the properties of the network.

Figures 6 through 9 represent a map of the network, with each node being depicted according to the x and y coordinates. Figure 6 presents the greatest continuous distance reached with the potentiometer set at the highest level (255). This graph represents the case where a node is considered reachable if 80% of the transmissions arrive reliably. We also did mappings with other choices of the percentage of packets received; however, their general shape is the same. Nodes in the center reach shorter distances only because they reach the edges of the network, which are about five meters away. Nodes at the corners of the network can reach about seven meters, meaning that even with the highest potentiometer setting, they cannot reach the entire network with a single hop.

Similarly, Figure 7 presents the greatest continuous distance reached with the potentiometer set at a lower level (230). With this setting, the distance reached is dramatically decreased, around two meters on average. One interesting thing to notice is that nodes at the borders of the network on average have shorter continuous distances reached than other nodes. This is due to their close proximity to the walls, which causes the signal reflections to be stronger.

Figure 8 maps the farthest distances reached for each node in the network. From here it can be seen that at the highest transmit power setting, the edges of the network can reach each other with high probability (though the corners cannot). However, at the lower potentiometer setting (230), the distance is reduced by about 50% (see Figure 9).

To further analyze the effects of the indoor environment on signal propagation patterns we present three additional sets of data. Figure 10 shows the greatest continuous distance reached for each of a line of nodes against the wall at the top of Figure 2 as a function of the potentiometer settings. As desired, the distance increases with higher potentiometer settings; however, the nodes do not reach uniform distances, mostly due to their location in the network. Figure 11 presents the farthest distance reachable as a function of the potentiometer setting. Again, this distance is greater than in the case of the greatest continuous distance and increases with increasing potentiometer settings. The nodes with the lowest distance increases are the ones near the corners of the network. In fact, our results show that nodes at the edges of the network (*i.e.*, the nodes closest to the walls) consistently had the lowest reachable distance. This implies that our node placement should be farther away from the walls if a more uniform environment is desired.

Finally, the same anisotropic behavior can be seen when considering signal propagation in different directions from a single node. Consider the circle around a node to be divided into 45° wedges, Figure 12 and Figure 13 depict the greatest continuous distance and the farthest distance reachable, respectively, by a single node located in the middle of the network in these eight wedges (given as directions) as a function of the potentiometer setting. This quantitatively demonstrates that not only do physically distinct nodes have different propagation patterns, but also the same node has different propagation patterns in different directions. It can be seen that the magnitude of the difference between directions is quite large. This variance depends on certain physical characteristics of the walls (*e.g*, a network wiring cabinet sticking out of one of the walls). This effect in the lab environment is not easy to avoid, though we would expect deployments in rooms of more regular shape (*e.g.*, hallways) to show less variance.

VI. Case Study 1: The ROCRSSI Localization Protocol

The SignetLab testbed has been used in order to study the performance of a localization algorithm, ROCRSSI [21], ROCRSSI+, and a refinement of each [22], [23]. These experiments were run locally in the Signet laboratory using the software management tool. The management tool was extended using the provided API to allow specialized visualization of the results of the localization algorithm.

ROCRSSI uses Received Signal Strength Indicator (RSSI) values instead of distances to compute the position of the nodes in a distributed fashion, where every node computes its own position based on information from beacon nodes (*i.e.*, nodes that know their exact position).

The protocol can be summarized as follows: each beacon collects RSSI values for transmissions from other beacons and stores them in a table. The beacons then broadcast the tables to all nodes in the network. Each localizing node compares the signal strength of broadcast messages received with the values

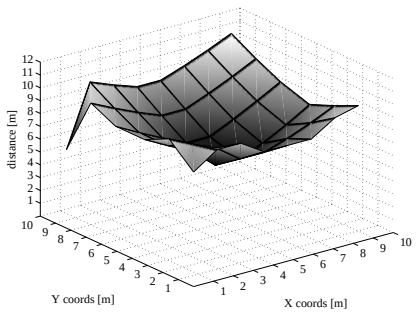

Fig. 8. Farthest Distance Reached, High Potentiometer

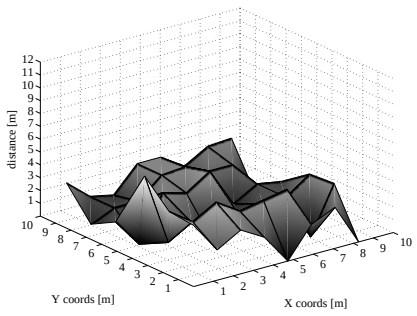

Fig. 9. Farthest Distance Reached, Low Potentiometer

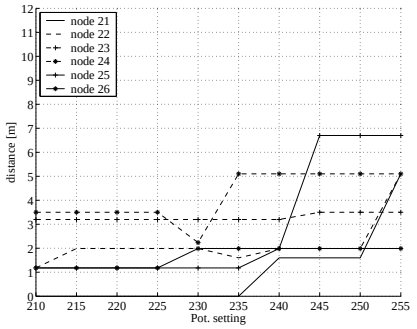

Fig. 10. Greatest Continuous Distance Reached, Line of Nodes

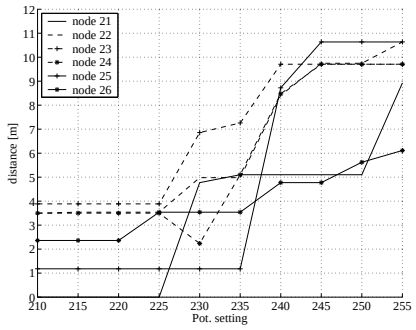

Fig. 11. Farthest Distance Reached, Line of Nodes

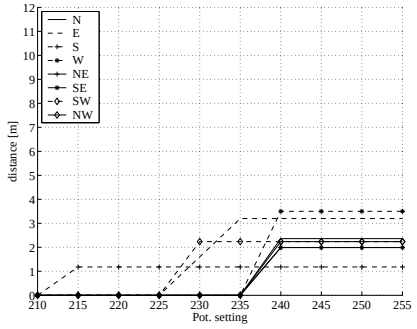

Fig. 12. Greatest Continuous Distance Reached in 8 Directions, Single Node

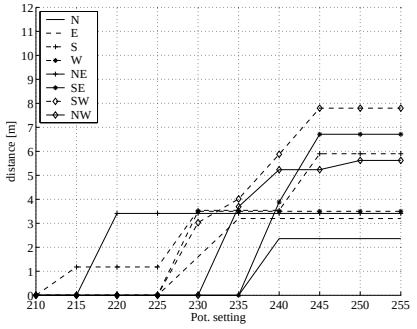

Fig. 13. Farthest Distance Reached in 8 Directions, Single Node

contained in the tables in the messages. Using this information, it estimates its distance from the source beacon. This distance is used to describe a ring around the source beacon in which the node must lie. This operation is repeated for every beacon message received. When all the beacon messages are processed the node assumes its location is in the centroid of the intersection of all rings. ROCRSSI+ is a derivative algorithm that yields more accurate localization [22], [23].

These algorithms suffer from the fact that they use RSSI values to attempt to determine distance, which are strongly affected by shadowing and anisotropic propagation and fading. While some attempts to solve this problem have been worked into the algorithms [23], only through the use of real hardware can the extent of these effects be quantified.

A. Test Setup

The performance tests have been conducted in both an indoor and an outdoor environment (for the outdoor experiments the entire testbed structure was replicated out of the lab). The software management tool was extended with a new plugin using the API provided. This plugin uses the software tool features to catch the debug messages coming over the serial line of each node. These messages have different prefixes depending on their type (localization started, beacons table filled, localization done). The data is displayed in a table in the plugin frame to allow the user to quickly compare the real position to the result of localization before and after the refinement. After the entire process is completed two buttons allow the user to display on the topology frame a colored line for each node, again using the main software API. These lines start on the real positions of the nodes and end on the computed ones (see Figure 14). This allows the user to immediately visualize the magnitude of the localization error and see whether or not systematic errors appear.

B. Results

Figures 15 and 16 show the results of the described tests for the indoor and outdoor testbeds respectively. Many runs were conducted, for both the ROCRSSI and the ROCRSSI+ algorithms, varying the number of beacons present in the network, but preserving the topology. After each run the localization error was computed. The figures depict the mean of all errors. From the figures, it can be seen that having more beacons allows the nodes to perform more precise localization, while in every case the ROCSRSSI+ is more accurate than ROCSRSSI.

VII. Case Study 2: The IRIS Protocol

The IRIS protocol suite was implemented on the testbed to validate previously published simulation and theoretical results [24], [25]. The testbed was used remotely from Rome. IRIS is a cross-layer solution including nodes awake-asleep schedules, MAC, and routing. It supports both interest dissemination and data convergecasting to the sink. IRIS can be summarized as follows. Nodes alternate between awake and asleep states according to a duty cycle d. Each node does

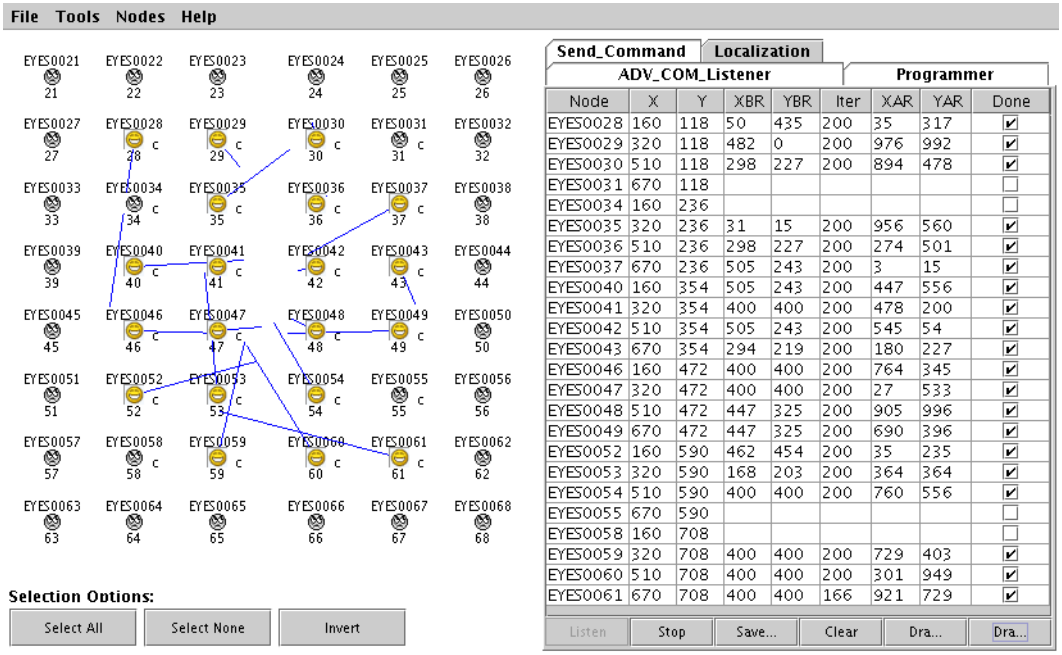

Fig. 14. Localization Plugin

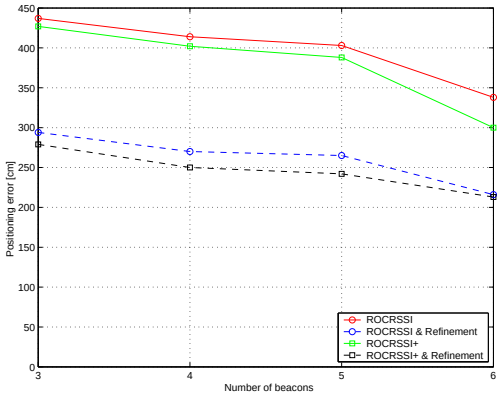

Fig. 15. Indoor localization results

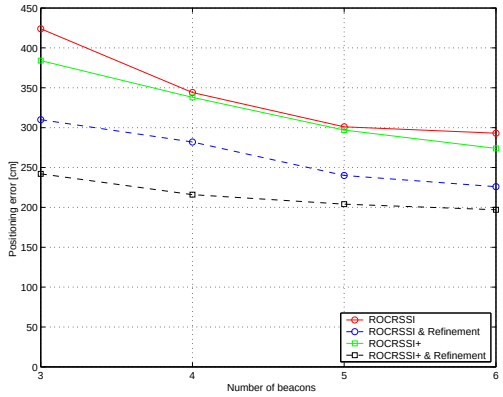

Fig. 16. Outdoor localization results

not know its neighbors and their awake-asleep schedules. The interest dissemination is performed according to the Fireworks algorithm [26], where each node receiving the interest takes one of two possible actions. With probability p it re-transmits the interest to all of its neighbors, or with probability $1-p$ it randomly picks c of its neighbors and sends the interest to them. Implementing such a simple scheme in a scenario in which nodes do not know their neighbors and their duty cycles is quite challenging. For this purpose the interest dissemination is integrated with a neighbor estimation procedure. The interest broadcast by a node triggers ACKs from the receiving neighbors, allowing the node to get a sample of the awake neighbors. This information is in turn used by the node to refine its estimate of the number of neighbors. This estimate is exploited to decide whether the interest has to be further broadcast by the node or not (the node stops sending the interest as soon as it believes it has reached all the intended destinations). The interest dissemination also allows nodes to discover their distance (in hops) from the sink. When the interest is received by source nodes, the convergecasting procedure is started. Nodes choose optimal next hops according to the SARA algorithm. For complete details of the algorithms, see [24], [25].

A. Test Setup

In order to perform the experiments researchers had to implement their own plugin to be used with the software management tool. The plugin, using the library of the testbed manager, configures a selected node as sink, instructs the sink to perform a given number of interest disseminations and collects results in a single file. IRIS code was then remotely uploaded on the SIGNET nodes.

In the first experiment, a node in one corner of the network was programmed to act as the sink, in the second the sink was instead located at the center of the deployment area. The two experiments were performed on the indoor SIGNET laboratory. In both of the experiments the transmission power of the nodes was set to the maximum allowed. The duty cycle d was varied between 0.05 and 0.9.

Metrics we investigated include the average time needed to complete the interest dissemination, the coverage (defined as the percentage of nodes reached by the interest dissemination), the overhead (number of interest messages and ACKs sent per node, on average, during each interest dissemination), and the average number of estimation rounds needed before a node decides it has reached all its intended destinations. Results reported are averaged over 20 different experiments.

B. Results

In all the experiments IRIS had a 100% coverage. Results on the interest dissemination duration are reported in Figure 17. As expected, the higher the duty cycle, the smaller the number of times the interest has to be rebroadcast by each node and the faster the interest dissemination.

Figure 18 shows the average number of REQUEST and RESPONSE messages transmitted by each node during a single interest dissemination. REQUEST messages contain the interest and hop count information and trigger ACKs (named RESPONSE messages). When the duty cycle is low, an inquiring node (*i.e.*, a node sending a REQUEST) is likely to find no awake neighbors. This forces the node to repeat the inquiry procedure several times leading to a higher overhead in terms of REQUEST messages. The duty cycle has instead only a limited impact on the number of RESPONSE messages which mostly depend on the number of neighbors and the p and c parameters.

Finally, Figure 19 displays the number of estimation rounds per node. The lower the duty cycle, the more challenging it is to achieve an accurate neighbor estimation and therefore the higher the number of rounds needed by each node before the neighbor estimation process is completed. However, the number of estimation rounds is quite low for all the different d values, ranging from 4 ($d = 0.9$) to 16 ($d = 0.05$).

VIII. Lessons Learned and User Experience

The first case studied involved the extension of the software tool using the provided API to allow the rapid visualization of the results of a localization technique. The researchers reported that the creation of the localization data collection and visualization pane was simple and facilitated the rapid testing of their protocol. The testbed allowed the examination of the extent to which real propagation effects distort the location predictions based on RSSI and whether or not their methods to mitigate this error were effective.

While the researchers were able to produce the needed results using the testbed, they expressed a desire for a more easily deployable testbed as a number of their tests needed to be performed in outdoor environments. To this end, we are developing such a deployable testbed (for a preliminary description see [27], presented as a demonstration at Sensys).

The second case studied involved using the testbed remotely. The researchers from Rome reported that the software tool made the process of compiling, executing and testing code on the testbed very simple, even remotely. Furthermore, it aided the ability to generate meaningful results quickly without interfering with the execution of the sensor network.

Using a real testbed demonstrated difficulties in implementing solutions that were not shown through simulation. The main such difficulty was in timing issues. The simulations performed allowed "perfect" knowledge of event times; however, in the real network, often times events occurred simultaneously or close enough in time that the granularity of the timers could not distinguish them. This led to synchronization issues that needed to be solved. To this end, the researchers had to implement more complicated task handlers to augment the protocols to work in real deployments.

Two needs were identified by the remote tests. First, while the hop count provided by the SignetLab testbed is highly variable, for routing protocols it may be desirable to have even larger hop counts than currently provided. Our deployable version of the testbed will also solve this problem, allowing wider distances and greater numbers of nodes to be deployed [27].

The second need was a time management portion of the software tool. Since the testbed is remotely accessible it is critical to incorporate a method to make sure only one researcher is using the testbed at a time. To this end, we plan to include a time-sharing solution in the tool. As mentioned earlier, there are many such solutions to choose from (*e.g.*, Motelab [6]), some of which will be included in the next version of the tool.

IX. Conclusions and Future Directions

This paper has presented the SignetLab sensor network testbed. It is composed of 48 EyesIFXv2 sensor nodes deployed in a single lab at the University of Padova, and of a software tool for managing and running experiments on the network. The software tool provides a graphical interface that allows researchers to select nodes to program, to send messages to nodes, and to gather realtime output and debugging information from nodes. The testbed provides a backplane so that such realtime information can be transmitted without interfering with the operation of the sensor network applications and protocols being tested. The software tool is written in Java and is therefore platform independent and can support any sensor network topology.

This paper has also presented an analysis of the propagation properties of the physical network. Key to the design of the testbed was the ability to fit the nodes within the space of the Signet lab while still providing an adequate multihop environment to test routing protocols. To solve this problem we had to construct our own antennae and then analyze the propagation capabilities of the network at various transmit power settings. The deployment provides a large variance in node coverage areas, providing the capability to test a wide

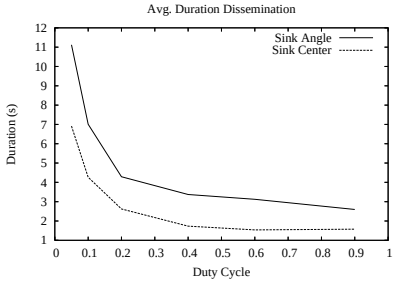

Fig. 17. Average time needed to be reached by Interest

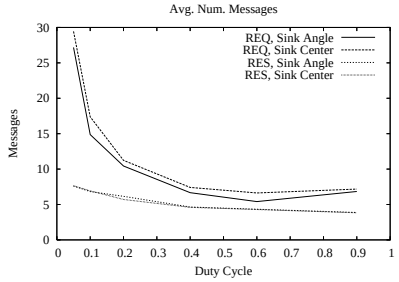

Fig. 18. Average number of messages

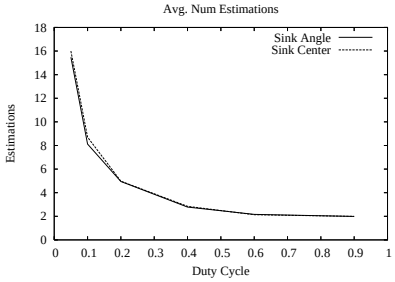

Fig. 19. Average number of estimation requests

variety of protocols. However, we also found that the testbed deployed shows a large amount of anisotropic behavior due to the physical characteristics of the area in which it is deployed.

Finally, the paper presented lessons learned from two case studies of researchers using the testbed and extending the software tool using the provided API. These experiences with the testbed led us to developing a deployable version of our testbed [27].

Integrating scheduling management software would round out the system and the next version of the tool will incorporate one of the available solutions. We have enough nodes to double the size of the testbed and are looking for meaningful ways to do so. It would also be interesting to incorporate nodes with different wireless technologies (*e.g.*, ZigBee [28], MicaMotes [29], etc.) to allow testing on a hybrid network. The software tool is prepared to handle this case and all that needs to be performed is the installation and profiling of the new nodes.

REFERENCES

[1] ns2 Network Simulator, http://www.isi.edu/nsnam/ns/.

[2] Infineon, Ltd., "EyesIFXv2 version 2.0," http://www.infineon.com.

[3] 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 *ACM Sensys*, November 2004.

[4] S. Jadhav, T. Brown, S. Doshi, D. Henkel, and R. Thekkekunnel, "Lessons learned constructing a wireless ad hoc network test bed," in *First Workshop on Wireless Network Measurements*, April 2005.

[5] 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)*, Bangkok, Thailand, May 2003.

[6] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: A wireless sensor network testbed," in *IEEE/ACM IPSN/SPOTS*, April 2005.

[7] D. Johnson, T. Stack, R. Fish, D. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile Emulab: A Robotic Wireless and Sensor Network Testbed," in *IEEE INFOCOM*, April 2006.

[8] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for network sensors," in *ASPLOS*, November 2000.

[9] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Lui, and M. Singh, "Overview fo the orbit radio grid testbed for evaluation for next-generation wireless network protocols," in *IEEE Wireless Communications and Networking Conference*, March 2005.

[10] P. De, A. Raniwala, S. Sharma, and T. Chiueh, "Mint: A miniaturized network for mobile wireless research," in *IEEE INFOCOM*, March 2005.

[11] IEEE 802 LAN/MAN Standards Committee, "Wireless LAN medium access control MAC and physical layer (PHY) specifications," IEEE Standard 802.11, 1999.

[12] M. Hiber, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, "Feedback-directed virtualization techniques for scalable network experimentation," Technical Note FTN-2004-02. University of Utah, 2004.

[13] mySql, http://www.mysql.com/.

[14] K. Whitehouse, G. Tolle, J. Taneja, C. Sharp, S. Kim, J. Jeong, J. Hui, P. Dutta, and D. Culler, "Marionette: Using RPC for interactive development and debugging of wireless embedded networks," in *IEEE/ACM IPSN/SPOTS*, 2006.

[15] K. Langendoen, A. Baggio, and O. Visser, "Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture," in *Workshop on Parallel and Distributed Real-Time Systems*, April 2006.

[16] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *ACM WSNA*, September 2002.

[17] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a sensor network expedition," in *EWSN*, January 2004.

[18] M. Yarvis, W. Conner, L. Krishnamurthy, J. Chhabra, B. Elliott, and A. Mainwaring, "Lessons from a sensor network expedition," in *EWSN*, January 2004.

[19] B. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. Snoeren, and A. Vahdat, "Mirage: A microeconomic resource allocation system for sensornet testbeds," in *IEEE Workshop on Embedded Networked Sensors*, May 2005.

[20] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems," in *Proceedings of Programming Language Design and Implementation*, June 2003.

[21] T. H. Chong Liu, Kui Wu, "Sensor localization with ring overlapping based on comparison of received signal strength indicator," in *Proceddings of the 1st IEEE international Conference on Mobile ad-hoc and sensor system (MASS)*, October 2004.

[22] S. Blom, M. Andretto, A. Zanella, M. Zorzi, S. Friso, and R. Crepaldi, "Poster abstract: Experimental localization results in an indoor wireless sensor network testbed," in *Third European Workshop on Wireless Sensor Networks (EWSN)*, Feb. 2006.

[23] R. Crepaldi, P. Casari, A. Zanella, and M. Zorzi, "Testbed implementation and refinement of a range-based localization algorithm for wireless sensor networks," in *Proc. of IEE Mobility Conference*, Oct. 2006.

[24] M. Rossi, R. R. Rao, and M. Zorzi, "Cost Efficient Routing Strategies over Virtual Topologies for Wireless Sensor Networks," in *IEEE Globecom*, 2005.

[25] M. Mastrogiovanni, C. Petrioli, A. Vitaletti, M. Rossi, and M. Zorzi, "Integrated Data Delivery and Interest Dissemination Techniques for Wireless Sensor Networks," in *IEEE Globecom*, 2006.

[26] L. Orecchia, A. Panconesi, C. Petrioli, and A. Vitaletti, "Localized techniques for broadcasting in wireless sensor networks," in *ACM DIAL M-POMC*, 2004.

[27] R. Crepaldi, A. Harris, A. Scarpa, A. Zanella, and M. Zorzi, "Demo: Signetlab: A deployable sensor network testbed and management tool," in *ACM Sensys*, 2006.

[28] ZigBee Alliance, "Zigbee specification," http://www.zigbee.org/.

[29] XBOW, "2nd generation micamote," http://www.xbow.com.