WSNED - A SUITE OF DATA COLLECTION AND DISSEMINATION APPLICATIONS FOR *W*IRELESS *S*ENSOR *N*ETWORK *ED*UCATION

Bo Sun, Lawrence Osborne
Department of Computer Science
Lamar University
Beaumont, TX 77713 U.S.A.
{bsun, ljosborne}@my.lamar.edu

T. Andrew Yang
Division of Computing and Mathematics
University of Houston – Clear Lake
Houston, TX 77058 U.S.A.
yang@uhcl.edu

ABSTRACT

Wireless Sensor Networks (WSNs) have become an ideal area of study to provide a wide range of applications for teaching Computer Science and Engineering courses, such as Computer Networks, Operating Systems, Software Engineering, and Embedded Systems. However, the complexities and extensive background knowledge requirement of WSNs make the design of suitable WSN courses a non-trivial task. The principle that learning must be grounded in experience necessitates a suite of suitable WSN applications for students to gain hands-on experience in order to deepen their understanding of WSNs. In this paper, focusing on using WSN data collection and dissemination as examples to provide technology-based education and training, we present a suite of practical WSN applications based on TinyOS 2.x (the dominant second generation operating system for low power WSNs) and MicaZ motes. Our presented suite ranges from the easy *Listen-BaseStation-BlinkToRadio* one-hop data collection application to the complex *Octopus* multi-hop data collection and dissemination application. For each of the applications, we illustrate its configuration details and the related background protocol principles. These applications can provide students with a deep understanding of WSNs and therefore intrigue their interests to achieve effective learning.

KEY WORDS

Wireless Sensor Networks, Applications, TinyOS, Data Collection and Dissemination.

1. Introduction

Wireless Sensor Networks (WSNs) have become ideal candidates to provide effective and economically viable solutions for a large variety of applications ranging from health monitoring, scientific data collection, environmental monitoring to military operations. Combining advanced technologies from Computer Networks, Operating Systems, Software Engineering, and Embedded Systems, WSNs have become a perfect example system for modern teaching and learning. Therefore, many universities offer related courses to introduce the basic principles of wireless sensor networks. Unfortunately, many of these courses only focus on the theoretical part of WSNs and lack a necessary component in education - *experimental learning* [1]. To achieve effective learning, students usually need vivid applications through which they can gain a deep understanding of WSNs by engaging hands-on projects. A suitable set of applications could also greatly benefit instructors to better introduce related concepts and protocols of wireless sensor networks.

Unfortunately, current WSN curricula lack a suitable suite of applications, mainly due to the complexity of WSNs and the difficulty of configuring WSN development environments. For example, even the most basic data collection application for one-hop WSNs requires considering the complex interactions among sensors, motes, and PCs. Furthermore, TinyOS [5], the dominant operating system for low power WSNs, is open source, which means no commercial support is yet available to cover mote programming, mote to mote communication and mote to PC communication. Currently, WSN application development is largely based on resources shared in the TinyOS community. However, the TinyOS community only provides very limited information. This is especially true for TinyOS 2.x [12] which is the second generation operating system for WSNs and is therefore relatively new. Many applications based on TinyOS 2.x do not have any tutorial at all. It is therefore easy to make all kinds of mistakes when presenting relevant applications. Some WSN applications require the inspection of source codes in order to understand their behavior.

To fill the aforementioned void in WSN education, we have obtained a feasible way to set up an easy-to-complex suite of WSN applications, named ***WSNED*** (Wireless Sensor Network EDucation), based on TinyOS 2.x [5]. Aiming at providing technology-based education, ***WSNED*** illustrates the most basic functions promised by WSN, data collection and dissemination, and enables technology-based learning. Specifically, the functions introduced in ***WSNED*** include: a. mote to mote communication; b. mote to PC communication; c. sensing; d. self-organizing properties of WSNs; e. data collection; f. data dissemination; g. remote data collection. Specifically, the following applications are presented:

• *Listen - BaseStation - BlinkToRadio*: This is the most

basic data collection application for one-hop WSNs. A text interface is used in the PC side to display the collected data;

• *Oscilloscope*: *Oscilloscope* is still a data collection application for one-hop WSNs. Compared with *Listen - BaseStation - BlinkToRadio*, *Oscilloscope* provides a GUI interface to facilitate data display in the PC side;

• *MultihopOscilloscope*: Based on the Collection Tree Protocol [8], *MultihopOscilloscope* is a data collection application for multi-hop WSNs. It also demonstrates the self-organizing capabilities of WSNs;

• *MViz*: *MViz* is also a data collection application for nultihop WSNs. Compared with *MultihopOscilloscope*, *MViz* provides a GUI interface that can generate the self-organized WSN network topology;

• *Octopus*: *Octopus* is the most complex application we present in the paper. Besides providing data collection functions, *Octopus* also implements data dissemination functions for multi-hop WSNs. Considering the fact that WSN network topology is typically dynamic and network membership is not static, *Octopus* also provides a more dynamic network topology than that of *MViz*.

Accompanying each of the applications is a brief introduction of the underlying network protocols and design principles, including how the protocols are designed and implemented, how the data are transmitted through protocol stacks and radio stacks, etc. By adopting the presented applications, an instructor provides students with the opportunity to develop and observe real WSNs, which are manageable in a classroom setting and can intrigue students' interests to further explore the various components of WSNs and how they work together to form an effective application. To facilitate easy adoption of the suite of WSN applications, we have made available in this paper all the relevant materials, including the URLs of the websites (see Appendix C) where the applications' source codes can be downloaded, configuration instructions for setting up the applications, etc. This paper also serves as a tutorial for deploying those applications. We intend to create additional tutorials and FAQs (Frequently Asked Questions) to cover common problems that may be encountered by the students when running those applications.

2. Related Work

2.1. TinyOS

TinyOS is an open source operating system designed specifically for low powered embedded wireless sensor nodes. Written in *NesC*, TinyOS is now the dominant operating system for sensornets. TinyOS aims at providing efficient yet flexible component-based applications. Little more than a non-preemptive scheduler, TinyOS applications are built from a large number of libraries of components [12].

The battery powered sensor nodes need to last unattended for a long time of periods, which makes energy a very valuable resource. To save energy and to increase the concurrency, operations in TinyOS are *split-phase*, which means the call to start a long *task* returns immediately and later receives a signal from the called component when the operation is done.

2.2. WSN Courses and Applications

Many WSN courses already exist in various universities and colleges. For example, WSN courses have been offered as special topics courses in most prestigious research universities. However, in most of those courses, only a reading list is presented and students are required to read papers and give research presentations. The final projects in these courses usually require writing a proposal illustrating research ideas. This kind of instruction formats may be suitable for research oriented institutes, because most students taking WSN courses aim at a Ph.D. degree. However, for small universities or teaching-oriented colleges, reading papers alone may become a daunting task, which makes it difficult for students to grasp the principles of wireless sensor networks.

There also exist open accessed sensor network labs. For example, MoteLab [13] from Harvard provides a public WSN testbed for development and testing of sensor network applications. However, this kind of labs can only be used by students after they have already developed some basic ideas of WSNs.

Numerous large-scale WSN applications have been deployed in field. For example, WSNs have been deployed to monitor seabird nesting environments on a small island off the coast of Maine [14]. WSNs have also been deployed to monitor a 70-meter tall redwood tree [15] to collect related air temperature, relative humidity, and photo synthetically active solar radiation. Unfortunately, it is impossible to reproduce these large-scale deployments of WSNs in classrooms.

It is therefore desirable to design a suite of WSN applications for educational uses. The applications will provide hands-on experience to complement students' learning of WSN theories. By developing the applications in the suite, students will gain deeper insights about the underlying protocols and design principles of WSN applications.

3. *WSNED* Suite of Applications

In this section, we present the details of the suite of WSN applications in ***WSNED***. Before presenting the applications in detail, we first discuss the various environments in which the applications may be deployed. For each of the applications, we present the details about how to set up the communications among motes, how to set up the communications between motes and PC, and

how to remotely read sensed values over the Internet.

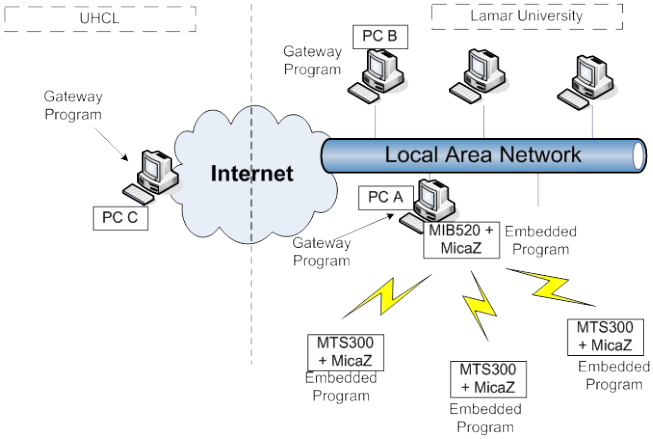
3.1. Environment Setup

For education purposes, we provide flexible host and network configurations for our presented WSN applications. Specifically, we present how to set up an application in various environments, including *stand-alone PC*, *Local Area Network* (LAN), and *Internet*. The *stand-alone PC* environment represents the most basic environment to run any WSN applications. It can demonstrate the necessary components in order for WSNs to work. *Local Area Network* and *Internet* environments make WSN applications more practical and interesting. To illustrate this point, just imagine the typical scenario: a field officer wants to remotely monitor temperature values of a field in ANY PC that connects to the Internet.

1) Stand-alone PC: Traditionally, it is not easy to set up a developmental environment for sensor networks. Although TinyOS community lists relatively detailed steps [2] to set up the environment under both *Windows XP* and *Linux*, many questions still come up. A quick look at the questions posted at TinyOS mailing lists [3] can confirm this observation.

In order to provide an easy-to-deploy WSN development environment, we create a development package based on the XubunTOS environment [4], which contains the TinyOS 1.x and 2.x [5] working environments based on a Xubuntu Linux [7] live CD. As a start, the user of that package may install the WSN development environment on a stand-alone PC. Figure 1(a) illustrates such a simple installation. In Figure 1(a), one MicaZ mote and one MIB520 gateway board from CrossBow [6] are coupled together to provide the gateway function; three additional MicaZ motes are used as sensor nodes, which perform data collection functions and transmit the collected data to the PC.

(a) Stand-alone PC Environment.



(b) Three Different Environments to Run WSN Applications.

Figure 1. How to Run WSN Applications.

Generally, any application under TinyOS can be categorized into either of two types:

• *Embedded Programs*: These are NesC programs that need to be installed on motes, for example, MicaZ motes [6]. Basic functions of these embedded programs include sensing, data receiving, data transmitting, and interactions with gateway boards to forward data to PCs. To compile and upload applications, one needs to go to the specific application directory. For example, the *BaseStation* application is located at $*TOSROOT/apps/BaseStation*.

• *Gateway Programs*: These are programs that run on PCs and interact with gateway boards to retrieve sensor data. *Java* language is widely used to implement gateway programs for mote to PC communications. Many important Java applications are located under $*TOSTOOR/support/sdk/java/net/tinyos/*. For example, tool *Listen*, which prints out the binary contents of heard packets on the PC, is located at $*TOSTOOR/support/sdk/java/net/tinyos/tools/Listen*.

2) Local Area Network and Internet: In this category, usually only the *gateway program* is required to run on remote PCs (probably used by a human operator to monitor the collected data). The stand-alone PC (working as a base station of the WSN) can open a *packet source*, a communication medium over which an application can receive and send packets [5]. Remote PCs can then connect to this packet source to retrieve sensor data. There are two popular types of packet sources:

• *serial@PORT:SPEED*: This means that the computer's serial ports are used as *packet source*. Suppose that the MIB520, a USB gateway, is used as the gateway hardware. PORT is then /dev/ttyUSB1. If MicaZ motes are used as wireless modules, SPEED is then 57600, which is the baud rate for MicaZ motes. The *MOTECOM* environmental variable can be set as *export MOTECOM=serial@/dev/ttyUSB1:micaz*.
This indicates that a PC serial port is used as the packet source.

• *sf@HOST:PORT*: This means that a packet source (e.g., *SerialForwarder*) runs on the PC *HOST* and port *PORT*. Therefore, remote computers can connect to *HOST* and *PORT* over TCP/IP.

Figure 1(b) summarizes the example environments for stand-alone PC, LAN, and the Internet. In Figure 1(b), PC *A* provides one example of the stand-alone PC configuration, PC *B* provides one example for LAN configuration, while PC *C* provides one example for the Internet configuration.

In the following, we provide step-by-step instructions to demonstrate our WSN applications. Throughout the rest of this paper, we use a series of products from CrossBow [6] as example hardware. Specifically, we use MicaZ motes as the wireless modules, MTS300 as the sensor boards, and MIB520 as the gateway board to connect motes to the PCs. The XBow website contains a reference for product details.

For the LAN environment, the rest of applications have been tested in a research lab at Lamar University. For the Internet environment, we set up the sensor network in the research lab at Lamar University. In the research lab at University of Houston at Clear Lake (UHCL), we run related applications to remotely read the sensor data collected at Lamar University over TCP/IP. Because the gateway programs at remote computers are the same under LAN and Internet environments, in the following, we only use the Internet as the example to illustrate the details.

3.2 Application One: Listen - BaseStation – BlinkToRadio

Listen - BaseStation - BlinkToRadio provides one of the most basic data collection examples in one-hop WSNs. The *BaseStation* application acts as a bridge between the PC serial port and the WSN. Whenever the *BaseStation* sends a packet to the sensor node through the serial port of the PC, LED 1 (Green LED) on the MicaZ mote is toggled. LED provides one of most basic approaches to debug WSN applications. To run this application, we need to upload the *BaseStation* binary image into the MicaZ mote that is connected to the PC through the MIB520 gateway (see Figure 2).

The purpose of the *BlinkToRadio* application is to transmit a data packet containing node ID whenever there is a firing of the timer event. In *BlinkToRadio*, the message is transmitted using the *AMSend.send()* to the broadcast address *AM BROADCAST ADDR*. We skip their introduction in this paper. For details, please refer to the TinyOS documentation [5].

On the PC side, whenever a packet is received, the *Listen* tool simply prints out the received message in a binary format. When the *Listen* tool is started, it can be configured to connect to different packet sources (See the implementation in *BuildSource.makePacketSource()* and *net/tinyos/toolsListen.java*). Here the packet source can be a local serial port or a remote port accessed through TCP/IP.

Following the environment introduced in Figure 1(b), Figure 2 presents the basic steps concerning how to set up and run the *Listen - BaseStation - BlinkToRadio* application. For the details about each step, please refer to Appendix A. As illustrated in Figure 2, we have four options when running this application:

• *Option 1*: The *Listen* tool directly reads data from the serial port at the local PC *A*;
• *Option 2*: We run a *SerialForwarder* in local PC *A*, which provides a packet source for received sensor data. We run the *Listen* tool to read data from this *SerialForwarder*;
• *Option 3*:
– We run a *SerialForwarder* in local PC *A*, which provides a packet source for received sensor data.
– We run the *Listen* tool in a remote PC *B* located at UHCL to read from the *SerialForwarder* running at PC *A*. In this way, the data collected at Lamar University can be remotely displayed over the Internet.

In the example illustrated in Figure 2, the *SerialForwarder* running at PC *A* provides a packet source of format *sf@140.158.130.239:9002*. Therefore, remote computers, like PC *B* in UHCL, can connect to it over TCP/IP.

• *Option 4*:
– We run a *SerialForwarder* in local PC *A*, which provides a packet source for received sensor data.
– We run a *SerialForwarder* in remote PC *B*, which provides a packet source based on the packet source provided in PC *A*.
– We run the *Listen* tool in remote PC *B* or some other remote PCs to read data from the *SerialForwarder* running at PC *B*. In this way, the data collected in Lamar University can be forwarded to a remote *SerialForwarder* over the Internet. This option demonstrates the great flexibility of running WSN applications.

Although *Listen - BaseStation - BlinkToRadio* only provides the most basic data collection application, it fully demonstrates the flexibility of network configuration to run a WSN application. Important concepts, including *SerialForwarder* and *packet source*, can be studied in detail through this application.

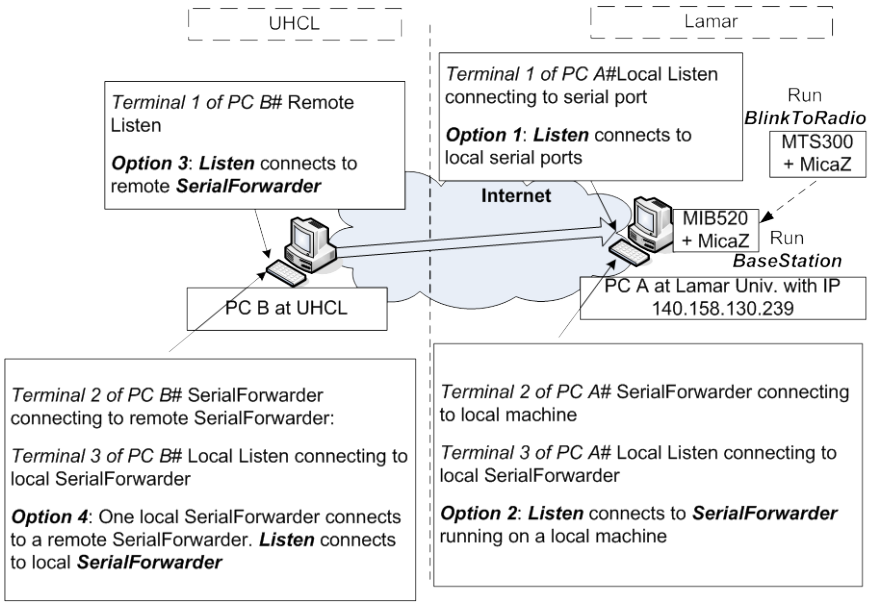


Figure 2. Listen - BaseStation - BlinkToRadio.

The rest of the applications can also be configured in similar manner as the *Listen – BaseStation - BlinkToRadio* application. To save space, we skip the network configuration of the following applications, while focusing on their unique functionalities.

3.3 Application Two: OscilloScope

Listen - BaseStation - BlinkToRadio application suffers from the following drawback: it only provides a text based interface to display sensor data in a binary format. Therefore, it is difficult to inspect and analyze the received data. To overcome this shortcoming, we introduce *Oscilloscope* application, which provides a GUI user interface for viewing the collected data.

On the mote side, *Oscilloscope* does not introduce much more complexity than *Listen -BaseStation - BlinkToRadio*. On the PC side, however, *Oscilloscope* provides a GUI interface to display the received data. We can, of course, still use the *Listen* tool to display the received data. The environment to run *Oscilloscope* is very similar to run *Listen - BaseStation - BlinkToRadio* application. The main difference is that we can invoke the *Oscilloscope/java/run* tool to invoke a GUI interface to display the received data. We skip the figure to run the *Oscilloscope* application to save space. Interested readers may download the figure from our course website at http://galaxy.cs.lamar.edu/~bsun/wsn/wsn.html.

3.4 Application Three: MultihopOscilloScope

Previous applications, i.e., *Listen - BaseStation - BlinkToRadio* and *Oscilloscope*, only form a single hop sensor network. In practice, each mote has limited transmission range. Therefore, a multi-hop sensor network is more practical, where each node can act as a router to relay messages from other nodes to the base station. Furthermore, self-organization is an essential requirement for WSNs, in which sensor nodes may spontaneously create an impromptu network.

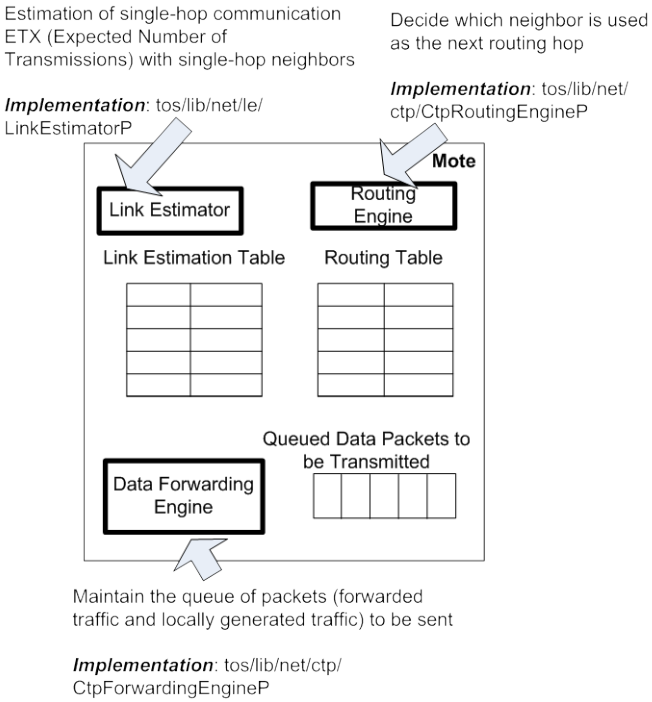


Figure 3. The Collection Tree Protocol.

MultihopOscilloscope is an application to demonstrate the above essential functionalities. In *MultihopOscilloscope*, nodes exchange *beacon* messages with their neighbors to self-organize the network into a data collection tree rooted at the root node with ID 0. *MultihopOscilloscope* is based on the Collection Tree Protocol (CTP) [8], in which all the nodes send data to the *root* of the tree. Three

components, *Link Estimation Engine*, *Routing Engine*, and *Data Forwarding Engine*, implement the core functionalities of CTP. Their relationships are illustrated in Figure 3.

• *Link Estimation Engine*: Each node runs a link estimation protocol to determine the choice of valid routes. In the current implementation of CTP, Expected Transmissions (ETX) is used as the routing gradients. The ETX of a node is the ETX of its parent plus the ETX of its link to its parent. Therefore, ETX provides an estimate of the number of transmissions it takes for one node to send a unicast packet whose acknowledgment is successfully received [8]. The *Link Estimation Engine* creates the *Link Estimation Table* for the node.

• *Routing Engine*: Based on the collected link quality, the routing engine can then decide the next neighbor to transmit data. Because WSN network membership is typically not static and link quality can vary significantly, the routing topology can also change dynamically. The *Routing Engine* generates the *Routing Table* for the node.

• *Data Forwarding Engine*: Based on the routing tables, the Data Forwarding Engine transmits the received packets. If a route is not available, the data forwarding engine maintains the queue of packets to be transmitted until the route is set up.

For similar reasons, the environment to run the *MultihopOscilloscope* application can be downloaded at http://galaxy.cs.lamar.edu/~bsun/wsn/wsn.html.

3.5 Application Four: MViz - Remote GUI Reader

Based on CTP, *MViz* is a further improvement over *MultihopOsciloscope*. First, the gateway program of the *MViz* application can display the self-organized network topology. This provides a great amount of information for users. Second, the gateway program provided by *MViz* is readily run to remotely retrieve data over the Internet through TCP/IP. As demonstrated in Figure 6, we collect the temperature and light data in one research lab in Lamar University and display the data remotely at UHCL. Appendix B lists extra necessary configuration in order to run the MViz application.

3.6 Application Five: Octopus

The WSN network topology is a very useful piece of information because it shows the dynamic connection among sensor nodes. One drawback of the displayed network topology in *MViz* application, however, is that it does not provide a timeout feature which can *dynamically* update the network topology. A more static topology is better for running MViz applications. Unfortunately, this is not often true as node connections in WSNs change over time.

As a further improvement over *MViz*, *Octopus* [10] provides a modular visualization and control tool for sensor networks. Compared to *MViz*, *Octopus* provides a *timeout* feature which can dynamically display the changed network topology. Furthermore, *Octopus* incorporates the *data dissemination* [9] utility to provide useful remote reprogramming functions. The data dissemination utility is implemented using TEP 118 [9], which is a proposal to establish eventual consistency on a shared variable. An example of a shared variable includes a new data sampling frequency which should be disseminated throughout the whole network.

TEP 118 is based on *Trickle* [11], which mainly focuses on addressing the issue of *when* shared small values should be exchanged among nodes in order to achieve value consistency. Because the communication demonstrates highly transient loss patterns and networking membership is not static in WSNs, *Trickle* is designed to achieve the tradeoff between rapid propagation and low maintenance. For details and performance analysis of *Trickle*, one can refer to [11].

4. Conclusions and Future Work

In this paper, we present a suite of practical applications - ***WSNED*** (Wireless Sensor Network EDucation). ***WSNED*** is built upon TinyOS 2.x and is flexible for the configuration of a standalone PC, Local Area Network and the Internet. ***WSNED*** focuses on the most basic functions promised by WSNs, data collection and dissemination, and provides easy-to-complex demonstrations for students. We also introduce related background protocol principles for ***WSNED***. These applications can provide students with a deep understanding of WSNs and inspires them to achieve effective learning.

We presented the applications of WSNED in a 2009 SIGCSE workshop [16]. Feedback from the participants in that workshop was very positive. Participants generally felt that the applications were well designed to enable effective learning; the learners were presented with applications of various complexities, from simple to intermediate to challenging. Some of the participants expressed their interests in adopting the WSNED applications in teaching wireless sensor networks in their own courses.

The presented applications in ***WSNED*** were used in teaching a Computer Science course in Wireless Sensor Networks at the Department of Computer Science at Lamar University in summer 2008 and 2009. Based on students' feedback, we observed that ***WSNED*** greatly facilitated students' interests in WSN study, which can foster both graduate and undergraduate research environment, encourage student collaboration, and enhance their confidence for future career. Faculty members adopting ***WSNED*** felt that it was easier to teach

relevant concepts about WSNs using the presented applications. The different environment we present in Section 3.1 makes it flexible for both students and faculty members to set up and to develop WSN applications. Considering the relatively low prices of sensor nodes, instructors at other universities may easily use different number of sensor nodes to set up applications at different scales by considering their financial situations.

As part of the future work, we plan to introduce ***WSNED*** into the Computer Science curricula at Lamar University and UHCL. Based on students' feedback, we plan to further elaborate on this set of applications and conduct a formal assessment of ***WSNED***. We also plan to introduce more applications and related lab exercises to assist in presenting additional important concepts of sensor networks.

Acknowledgements

This work was supported in part by grants from US NSF (DUE-0633445 and DUE-0633469) and the Institute of Space Systems Operations (ISSO).

References

[1] P. Denning, "Great Principles in Computing Curricula," *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, Norfolk, VA, 2004, pp. 336-341.
[2] http://www.tinyos.net/tinyos-2.x/doc/html/install-tinyos.html, accessed in May 2008.
[3] http://mail.millennium.berkeley.edu/pipermail/tinyos-help/, accessed in May 2008.
[4] http://toilers.mines.edu/Public/XubunTOS, accessed in May 2008.
[5] http://www.tinyos.net, accessed in May 2008.
[6] http://www.xbow.com, accessed in May 2008.
[7] http://www.xubuntu.org, accessed in May 2008.
[8] http://www.tinyos.net/tinyos-2.x/doc/html/tep123.html, *Collection Tree Protocol* (CTP), accessed in May 2008.
[9] http://www.tinyos.net/tinyos-2.x/doc/html/tep118.html, *Dissemination of Small Values*, accessed in May 2008.
[10] http://csserver.ucd.ie/~rjurdak/Octopus.htm, accessed in May 2008.
[11] P. Levis, N. Patel, D. Culler, and S. Shenker, "Trickle: a Self-Regulating Algorithm for code Propagation and Maintenance in Wireless Sensor Networks", *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation (ACM NSDI)*, San Francisco, CA, 2004.
[12] P. Levis, *et al.* "T2: A Second Generation OS for Embedded Sensor Networks," *Technical Report TKN-05-007*, Telecommunication Networks Group, Technische Universitat Berlin, Nov. 2005.
[13] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A Wireless Sensor Network Testbed," *Fourth International Symposium on Information Processing in Sensor Networks (IPSN)*, Los Angeles, CA, 2005, pp. 483-488.
[14] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring", *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, GA, 2002, pp. 88-97.
[15] G. Tolle, *et al.* "A Macroscope in the Redwoods," *Proceedings of the 3rd International conference on Embedded Networked Sensor Systems (ACM SenSys)*, San Diego, CA, 2005, pp. 51-63.
[16] B. Sun, and T. Andrew Yang, "Teaching Application Development of Wireless Sensor Networks using Motes and Sensors", *Workshop proposal in 40th ACM SIGCSE*, March 4-7, 2009, Chattanooga, TN U.S.A.

Appendix A - Detailed Commands Used in this Paper

• *Local Listen connecting to serial port*: java net.tinyos.tools.Listen -comm serial@/dev/ttyUSB1:micaz
• *Local Listen connecting to local SerialForwarder*: java net.tinyos.tools.Listen -comm sf@localhost:9002
• *Remote Listen*: java net.tinyos.tools.Listen -comm sf@140.158.130.239:9002
• *SerialForwarder connecting to local machine*: java net.tinyos.sf.SerialForwarder -comm serial@/dev/ttyUSB1:micaz
• *SerialForwarder connecting to remote SerialForwarder*: java net.tinyos.sf.SerialForwarder -comm sf@140.158.130.239:9002
• *Set up MOTECOM environment as local serial port*: export MOTECOM=serial@/dev/ttyUSB1:micaz
• *Set up MOTECOM environment as local SerialForwarder*: export MOTECOM=sf@localhost:9002
• *Set up MOTECOM environment as remote SerialForwarder*: export MOTECOM=sf@140.158.130.239:9002
• *Gateway program for Oscilloscope Application*: oscillloscope/java/run
• *Gateway program for MultihopOscilloscope Application*: MultihopOscilloscope/java/run
• *Local gateway program for Mviz Application*: tos-mviz -comm sf@localhost:9002 -dir /opt/tinyos-2.x/apps/MViz MVizMsg
• *Remote gateway program for Mviz Application*: tos-mviz -comm sf@140.158.130.239:9002 -dir /opt/tinyos-2.x/apps/Mviz MVizMsg

• *Gateway program for Octopus Application*: java OctopusGui

Appendix B - Appendix for Extra Configuration when Running MViz Application

• Add SENSORBOARD=mts300 in *Makefile* when compile
• Modify MVizSensorC.nc to add related sensors
• Based on MultihopOscilloscopeC.nc, the root id should be 0
– For *root* node: make micaz install,0 mib510,/dev/ttyUSB0
– For *non-root* node (e.g. node 1): make micaz install,1 mib510,/dev/ttyUSB0
• chmod 666 /dev/ttyUSB*
• How to configurate light, temperature sensors? Modify:
– $TOSROOT/tos/platforms/micaz/DemoSensorC.nc
– $TOSROOT/tos/sensorboards/mts300/DemoSensorC.nc
– $TOSROOT/apps/Mviz/MVizSensorC.nc

Appendix C - Appendix for URLs to Download Applications Illustrated in this Paper

TABLE I URLS FOR ILLUSTRATED APPLICATIONS

Application	URL
BaseStation	http://tinyos.cvs.sourceforge.net/tinyos/ tinyos-2.x/apps/BaseStation
BlinkToRadio	http://tinyos.cvs.sourceforge.net/tinyos/ tinyos-2.x/apps/tutorials/BlinkToRadio
Listen	http://tinyos.cvs.sourceforge.net/tinyos/ tinyos-2.x/support/sdk/java/net/tinyos/tools/Listen.java
OscilloScope	http://tinyos.cvs.sourceforge.net/tinyos/ tinyos-2.x/apps/Oscilloscope
MultihopOscilloScope	http://tinyos.cvs.sourceforge.net/tinyos/ tinyos-2.x/apps/MultihopOscilloscope
MViz	http://tinyos.cvs.sourceforge.net/tinyos/ tinyos-2.x/apps/MViz
Octopus	http://csserver.ucd.ie/~rjurdak/Octopus.htm