WSNTB: A Testbed for Heterogeneous Wireless Sensor Networks

Jang-Ping Sheu[1], Chia-Jen Chang[2], Chung-Yueh Sun[2], and Wei-Kai Hu[2]
[1]Department of Computer Science, National Tsing Hua University
[2]Department of Computer Science and Information Engineering, National Central University
sheujp@cs.nthu.edu.tw

***Abstract*—In this paper, we design and implement a testbed to realize various experiments in heterogeneous wireless sensor networks. Our implementation includes hardware infrastructure and software framework. The hardware infrastructure consists of servers, gateways with converters, and sensor nodes in three-tier. Our testbed can support different sensor nodes with USB or RS232 interface. Users can experiment with real hardware resources and interactive with our testbed in real-time. Here, we deploy two kinds of self-designed sensor nodes, Octopus I and Octopus II, in our testbed. The Octopus sensor nodes are compatible with IEEE 802.15.4/ZigBee standard for experiments. The software framework composes with three main layers: services interface layer, testbed core layer, and resource access layer. Our testbed allows users to customize their applications for specific sensor nodes and experiments locally with remote hardware resource. Users can freely choose the number of nodes and assign a period of processing time through our testbed website. By using our testbed, users can save lots time from creating an experiment environment, reduce the hardware expense, enhance the devices utilization rate, and fasten on the verification of experiment results.**

Keywords: Testbed, wireless sensor networks, Zigbee

I. INTRODUCTION

Wireless sensor networks (WSNs) consist of many sensing devices which can run individual applications and communicate with each other. Usually, we verify our network protocols by using simulations or experiments in real WSNs. In simulations, we cannot control precise packet timing, hardware interrupts, and real PHY/MAC layer events. Not all simulation results are equal to the real experiments. In real experiments, we have complex environment settings and resource sharing problems. Not every user has enough hardware devices to experiment on large-scale WSNs. In this paper, we propose a reconfigurable heterogeneous WSNs testbed called *WSNTB*. We divided the testbed architecture into three layers. The first layer is servers. The second layer is gateways with converters. The third layer is sensor nodes.

In hardware infrastructure, our testbed provides connectors and gateways for sensor nodes. We use the standard interfaces to avoid the hardware binding problems. The converters help sensor nodes to convert serial data into TCP/IP packets. The gateways, a bridge between Internet and WSNs, have their own physical Internet Protocol (IP) address to communicate with sensor nodes. The gateways and converters are controlled by servers. Thus the *WSNTB* with gateways can support multiple WSNs over the Internet. In the sensor nodes, we use two kinds of self-designed sensor nodes in our testbed. The first one is called Octopus I based on the Atmel AVR architecture[1], and the other one is called Octopus II based on the Texas Instruments MSP430 architecture[15]. We deploy 34 sensor nodes (Octopus I and Octopus II) in *WSNTB*. The nodes are distributed over two laboratory rooms of our department building and the passage. In the *WSNTB*, the average cost of each node is about 50 US dollars. Since the cost is cheap, expending the testbed scale is easily. In software framework, our testbed supports the famous sensor operating system TinyOS [4] and our implemented operating system LOS [14]. We develop a middleware to help users to process their individual experiments within different sensor nodes and heterogeneous WSNs. Readers can find more information about *WSNTB* on the web at http://testbed.wsn.tw.

The remainder of this paper is organized as follows. Section 2 describes the related works of WSN testbeds. Section 3 involves the design and implementation details of our testbed. The system operation and evaluation results are shown in Section 4. Section 5 concludes this paper.

II. RELATED WORKS

The WSN testbeds are useful and flexible to realize experiments for researchers. Users can use the testbeds to verify and prove their protocols such as power control, time synchronization, object tracking, routing, and security etc. Over the past few years, users usually create and setup the WSN environment by themselves. Then many complex steps and various environment factors will confuse users. In order to make users focus on their research, a number of testbeds are proposed by researchers.

Emulab [9] is a network emulation testbed developed by Utah University. Users can develop, debug, and evaluate their own systems. They can do a wide range experiments such as pure emulation, 802.11 wireless experiments, sensor networks experiments, mobile wireless experiments, and NS-2 Simulation[11]. The testbed consists of 25 MICA2 motes [4] connected with a serial port. Users can full control and debug these nodes.

EmStar [6] is developed by University of California. This is a software platform for WSN experiments. They deployed a CENS heterogeneous testbed to prove and test their platform. The Linux-based framework consists of 39 nodes. There are 26 Mica2 motes with MIB510 [2] programming board, and 13 Mica2 motes with the Stargate[2]. The server is connected to the MIB510 with serial port and the Stargate with Ethernet. They not only provide an integrated TinyOS environment but also a service protocol to run experiments easily.

This work was supported by the National Science Council under grant NSC 96-2218-E-007-016.

978 -1-4244-1866-4/08/$25.00 © 2008IEEE

MoteLab [16] is developed by Harvard University. They use the database to log the experiment data and provide a web-based interface for users. The authorized users can upload binary images to run their experiments. They use a job queue system for experiments scheduling. Users can submit jobs and run experiments. After the experiments, users can view the visualized results. The hardware platform consists of 30 MICAz nodes with Ethernet gateway. Until now, there are 190 Motes for experiments. Each two Motes are connected to a Tmote Connector.

The TWIST [8] testbed is hold by Technische Universität Berlin. They help users load programs and run experiments such as time synchronization and power control. The system is divided into two major parts. The first part is the server to serve the demands of users and control all of nodes. The second part includes two types of sensor nodes, eyesIFX v2 [7] and Telos motes [12] which are plugged onto the switch. The architecture is extended form the UC Berkeley's Omega testbed and Motescope testbed [13].

In Ohio State University, the Kansei sensor testbed [5] is a large-scale testbed including both 210 Extreme Scale Motes (XSM) and Extreme Scale Stargates (XSS). The devices are special design for Kansei testbed. The topology is using both Ethernet and 802.11b wireless LAN to control the testbed. Kansei also provide a web interface for users to upload programs, scheduled jobs, and retrieve results with EmStar software framework. The nodes deployed on 35 4' x 6' tables by grid pattern. They could be configured as 2 ~ 3 feet with 210 nodes, 3 ~ 4 feet with 140 nodes, 4 ~ 6 feet with 70 nodes, 6 ~ infinity feet with 35 nodes. Besides, users can choose the grid region for their experiments.

We follow IEEE 802.15.4 standard protocol and provide the standard interface with USB and RS232 for any sensor node. The web-based interfaces [10] support cross-platform services. In addition, we add a new experiment method *local mode* for users who can run their programs locally on our testbed. When users start experiments by schedule, they will receive notification and have permission to add remote serial port. Moreover, users can transmit and receive data via serial port directly. In other words, users have full right to control the actual hardware resource in local.

All of the related testbed projects are connected with Ethernet for management. The Emulab and EmStar are running with MICA Motes and communicate via serial ports. The MoteLab and TWIST are running with Tmote Sky via USB interface. The Kansei testbed is running with Extreme Scale Motes as sensor nodes. From the software point of view, all of the related testbed projects are based on TinyOS. The Kansei testbed use EmStar as their software framework. All of the above testbeds have web interface for experiment except for the TWIST testbed. But all of them have no local mode function as we proposed.

III. Design and Implementation of WSNTB

Our *WSNTB* is designed for heterogeneous WSN experiments. *WSNTB* provides users with real hardware resources in WSNs. Users can use both the web-based interface and the special function, called local mode, to run their applications on testbed. It involves two WSNs and three gateways. Each WSN has 17 sensor nodes. According to users' requirements, users can choose the single one or both of WSNs, with or without the gateways to experiment.

A. Hardware Infrastructure

In this section, we will describe the hardware infrastructure of our testbed. The hardware infrastructure, being a chain from the server tier to node tier, consists of servers, gateways with converters, and the sensor nodes. All of these nodes are connected to the network. Recently, most testbeds are constituted of several specific nodes. But we want to avoid sensor nodes bound with specific ones; we use converters and gateways as standard connector to make up the communication chain. The converters and gateways with RS232/USB connector can easily connect to our self-implemented nodes. The benefits are that users can rapidly change or upgrade the nodes on demand. Otherwise, users can control various instruments through Ethernet.

A 3-tier hardware infrastructure was designed to construct the *WSNTB* as shown in Fig. 1. The web server, emulating server, and database server are running with Intel's Pentium 4 CPU. The database server requires more disk space to store all experiment results, but the web server needs more RAM space to handle all requests from the whole world. Finally, the emulating server just needs a stable operating environment for long work. All of them have high speed Ethernet interface to connect to the network backbone.

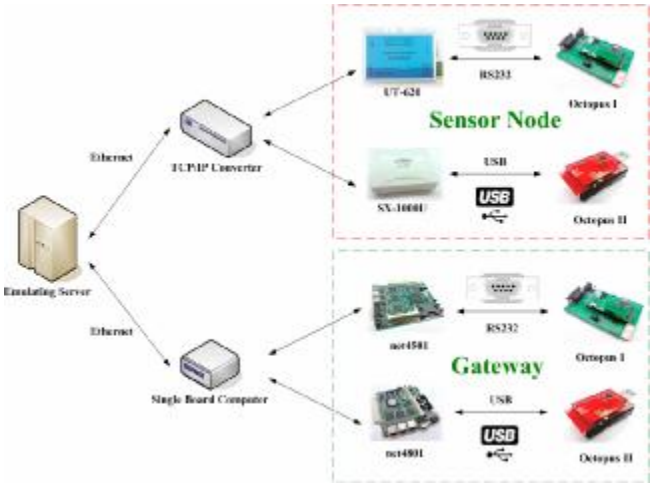

Figure 1. *WSNTB* architecture

There are two converters, RS232 to Ethernet (UT-620) and USB to Ethernet (SX-1000U). All of our sensor nodes are connected with those converters. The first generation node, called Octopus I, consists of an 8-bit MCU (AVR Amega128L[1]) with 128Kbytes ROM and a 2.4GHz IEEE 802.15.4 compliant RF transceiver (Texas Instrument CC2420). The dock of Octopus I provides not only a serial programming interface but also extended power prediction. The second generation node, called Octopus II, is a reliable low-power wireless sensor node platform. Octopus II is equipped with a 16-bit MCU (Texas Instruments MSP430) and a CC2420 transceiver.

Gateways play the role of communication between WSNs and Internet access. For example, we can deploy a single board computer with public IP address as gateway in a WSN. Then the node on the single board computer can communicate via RF with neighbor nodes and access

Internet with IP address. Several testbeds use the customized hardware as the gateway, such as the Crossbow Stargate [2]. But the Stargate cannot run the usual PC program. If users want to develop their own applications on Stargate, they need to be training again. In order to solve the problem, we adopt the Soekris Engineering's products with the standard x86 PC CPU and port PC operating system directly. We use the two versions of PC compatible single board computers: Soekris Engineering single board computers net4501 and net4801 as our gateways. Thus, it is flexible to use net4501 and net4801 to provide a whole range of different functions and run communication applications.

We install the operating system either by preloading the Compact Flash storage, or by booting over the network using the PXE boot code in the standard BIOS. While we have a new experiment, we reinstall them via the Ethernet to avoid any security risks. They are not expensive but support standard protocol, common interfaces, and popular operating systems. That using the gateway will help users to develop their programs fast and easily. Finally, we use the ZyXEL ES-108A Ethernet switch as LAN switch to build our testbed Ethernet backbone. The ZyXEL Ethernet Switch offers advanced VIP ports to provide high priority bandwidth. So we can let sensor nodes use normal port and the backbone lines use the VIP ports. The network for expansion becomes very rapidly and handily.

B. Software Implementation

In this section, we want to describe the software framework of *WSNTB*. The software framework of *WSNTB* includes services interface layer, testbed core layer, and resource access layer as shown in Fig.2.

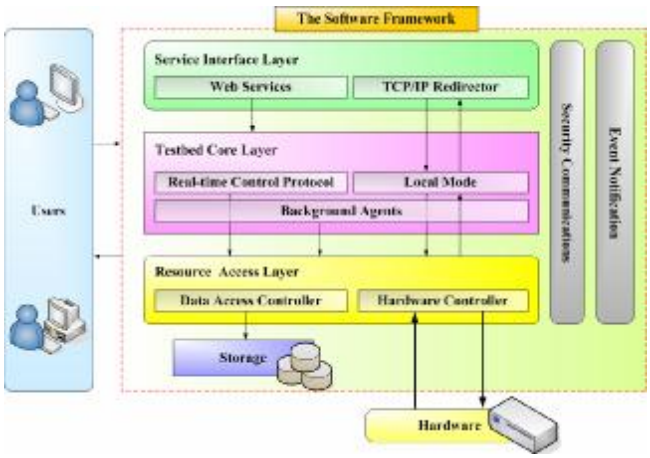

Figure 2. The software framework of WSNTB

The service interface layer let users access our *web services* from the website. We allow users to control and interactive with the testbed via web services. The service interface layer consists of web services and TCP/IP redirector. Web services will dispatch the jobs in testbed core layer. TCP/IP redirector is the kernel of local mode. In the testbed core layer, there are *real-time control protocol*, local mode, and *background agents*. The real-time control protocol and local mode are very important components in *WSNTB* we developed. Real-time control protocol is a new process in our testbed. In the past, users can only load program into sensor nodes by schedule and waiting the experiment results. But we allow users to assign a period of time for their experiments. During the experiment period, users still receive packets form individual notice and have the domination of controlling our testbed, including sending commands, getting response, or uploading files into sensor nodes in real-time. Background agents are dealing with users' requirements in running experiments. Users can interactive with sensor nodes under TinyOS and collect experiment results just like that users have the whole hardware infrastructure to themselves. The function of local mode can help users to access hardware resources directly. Users can develop their programs and even collect results at their original TinyOS environment. In the resource access layer, we provide the *data access controller* to save files and database records. In security, we adopt the private IP address in communication with database and storage. Hardware controller can assist the testbed core layer to detect the living hardware, its status, and reset it. The software framework in *WSNTB* also has event notification. We can detect existing events and response events correctly. When users do their experiments, they can get real-time messages and interactive with the devices.

Mapping to the real WSN environment, the implementation detail is shown in Fig.3. We divide the *WSNTB* into 3-tier architecture. The first tier of *WSNTB* consists of the web server, emulating server, and database server. The three servers are connected with the Ethernet backbone. But only the web server has its public IP address for public access, the others own their private IP address for security. Internet users can only connect to web server to avoid any illegal access.

We have installed Apache with PHP as web and MySQL as database in all three servers. They are open source software and can execute in cross operating systems. These servers are following HTTP protocol. Then, we deploy different software packages into individual server. In web server, we use WYSIWYG Editor as the basic text editor and phpMyAdmin as database management interface for MySQL. We developed the following modules: management system and user interface. In the management system module, we provided the online experiment, environment status, feedback functions, and the map. In the user interface, we provided the testbed documents, data downloads, and the user register functions. *WSNTB* is public for operation and users can register their accounts via web server.

The emulating server plays an important role in online experiment, especially in local mode and real-time control protocol. We install Microsoft VB and Java runtime library in the emulating server. The Java runtime library supports TinyOS and Cygwin environment and the VB library supports windows GUI applications. The Virtual COM is software for users to create virtual serial port in local computer. By using the software, users can use local mode to access sensor nodes directly in *WSNTB*. AVR tools are used for writing the image of AVR series and reprogramming the boot loader program for our sensor nodes. We also deploy two modules, system control and hardware control, in the emulating server. In system control modules, we provide the local mode, job queue, result

viewer, and scheduler. In hardware control modules, we provide the real-time control protocol, TCP/IP tools, hardware device reseter, and hardware device checker. The most important applications of emulating server are local mode, real-time control protocol, and hardware device reset.

We provided local mode for users to experiment on their own computer with the actual hardware resources. The local mode does not appear in other studies of testbeds. In the past, users can only experiment on the testbed by using the existing interfaces and functions. It is traditional to display the results with packet format in text in the above-mentioned testbeds. If users want to show a curve or

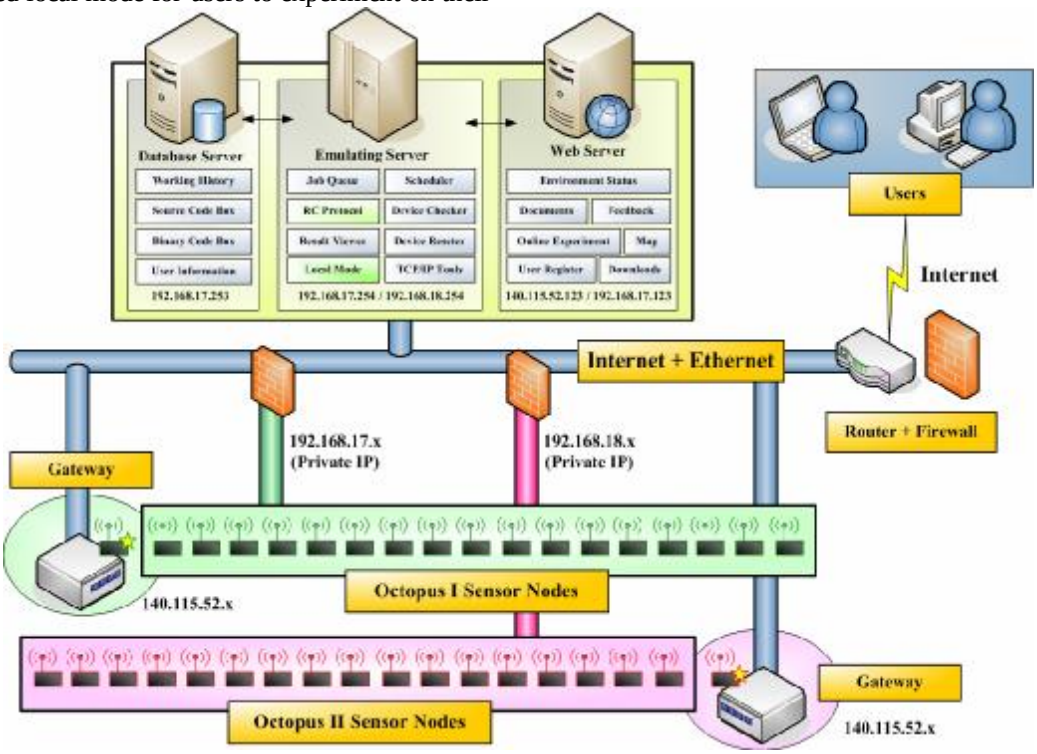

Figure 3. The implementation details of WSNTB

a table, they need to get the experiment results back and parse the results to the chart. As implied by the name, local mode let users develop and run their programs in users' computers. When users get a period of time for their experiments, they will receive a note to ask them to get permission and to use local mode. Users need to login *WSNTB* and configure the serial port redirector. According to the nodes chosen by users, we will open the corresponding TCP ports for them to add the ports as local serial ports. For example, assume a user has 10 nodes to perform his experiment. The emulating server will open 10 TCP ports with individual public IP address. Then user can use the TCP/IP software to add these TCP ports as local serial ports. After setting the TCP ports, the user can communicate with individual sensor node and experiment directly in his local computer.

We design a real-time control protocol for users to make instruction submission, data exchange, and TCP/IP communications in real-time. The communication flow is between emulating server and converters. It is necessary to submit experiment and wait experiment results. For example, users compile their programs with TinyOS and start their experiments. When all the experiments were done, users could analyze experiment results. But users need to interactive with sensor nodes in experiments. The real-time control protocol is different from traditional method. During the processing experiment, users can send command or compile programs to the sensor nodes via this protocol. The emulating server will help users call system shell to execute programs and catch the response messages for users.

If the system ran into a halt or application error, the system administrator needs to reset the sensor nodes manually for Octopus I. To reduce the workload of system administrator, we design a hardware reset device attached to each sensor node. When a sensor node receives the reset command, the sensor node will reboot automatically.

In emulating server, we provide the job queue and scheduler, result viewer, TCP/IP tools, device checker, and device reseter. Job queue and scheduler can allow users to assign a period of time for experiments. Users can check any device via the web but only the administrator has the right to reboot a specific device.

We deploy user management modules and job management modules in the database server. The working history function will record experiment information and environment status. The user information includes personal information, feedback opinion, announcement for administrator etc. The source code box is to collect the experiment's source code, and compress them into a zip file. Users can load their programs form the source code box, and edit it again. The binary code box saves the experiment's binary image. If users want to run the same experiment again, they can easily choose the existed binary image in binary code box.

The second tier of *WSNTB* means servers to nodes. They

are the Internet and Ethernet backbone, Ethernet switches, and the gateways with converters. The third tier of *WSNTB* means the terminal nodes. In security issue, we build three firewalls on the Internet and Ethernet backbone. The first firewall built in front of the Internet router to block the illegal incoming connections from remote users. Then the two firewalls built on each WSN backbone to protect network environment.

The gateways are the bridges over our network environment. In our testbed, we deploy the gateway with public IP address and allow users to access the gateway via the direct connections. We use the single board computer as our gateway. The single board computer can replace the specific Ethernet gateway hardware such as Stargate. Gateways can communicate with WSNs and Internet. We installed Linux as our operating system and porting TinyOS on the gateway. We assign private IP address for each converter. Through the Ethernet connections, the server can control individual converter directly and avoid the Internet illegal access. This solution can make sure receiving the experiment data correctly. The nodes execution environment is composed of self-made hardware, firmware, and software. We developed a boot loader to receive programs form the gateways or converters. The library of sensor operating system will pack with applications.

In this section, we present the main system architecture, hardware infrastructure and software framework, and the implementation details. The detail system operations will describe in next section.

IV. SYSTEM OPERATIONS

A. *System Menu*

We design and implement several menu items on *WSNTB*. Users can use these menu functions from our website as shown in Fig.4. We display the users' menu according to their permission. For example, the guest users just can see the basic menu items. The authorized users not only see the basic menus but also their own menu. In addition, the administrator can see the administrator menu. Our testbed has the top level menus and submenus. The top level menus consist of *system architecture*, *environment*, *information*, *services*, *my workspace*, *experiments*, *configuration*, *device maintain*, *information maintain*, and *system maintain*. After users login, they will get their own menus. Each normal user has basically four top level menus: system architecture, environment, information, and services. The authorized user has additional *my workspace* menu for experiments. The administrator has the extra five menus in top level for administrative purpose.

In system architecture, we introduce *WSNTB* and show the hardware infrastructure and software framework. In environment menu, we give the environment status of *WSNTB*. Users can view the location of testbed map and status of hardware devices. And we will show the detail of hardware devices. Users can check anytime as they want to use them. The hardware devices include nodes, converters, gateways, and the servers. In information menu, users can obtain the latest news of *WSNTB*. There are references to *WSNTB* and publications. Users can contact with our testbed project director and staffs.

In service menu, there is a service center for users. If users have any question about *WSNTB*, users can use the feedback item to tell us their problems. We will help users to solve the problems. Also users can subscribe our e-letter to get the news immediately.

My workspace menu is only for authorized users. When user's register is legal, user can do his/her experiment. In my workspaces menu, it is the core applications of *WSNTB*. Especially for the experiments menu, there is an experiment flow for users. First, users need to create a new experiment task or choose the existed experiment one from the code box, and choose which nodes and gateways they want to use. Users can use the code box to create and maintain their program. If users want to use the program again in the new experiments, they can load the programs form the code box. Finally users have to decide when to run the experiments.

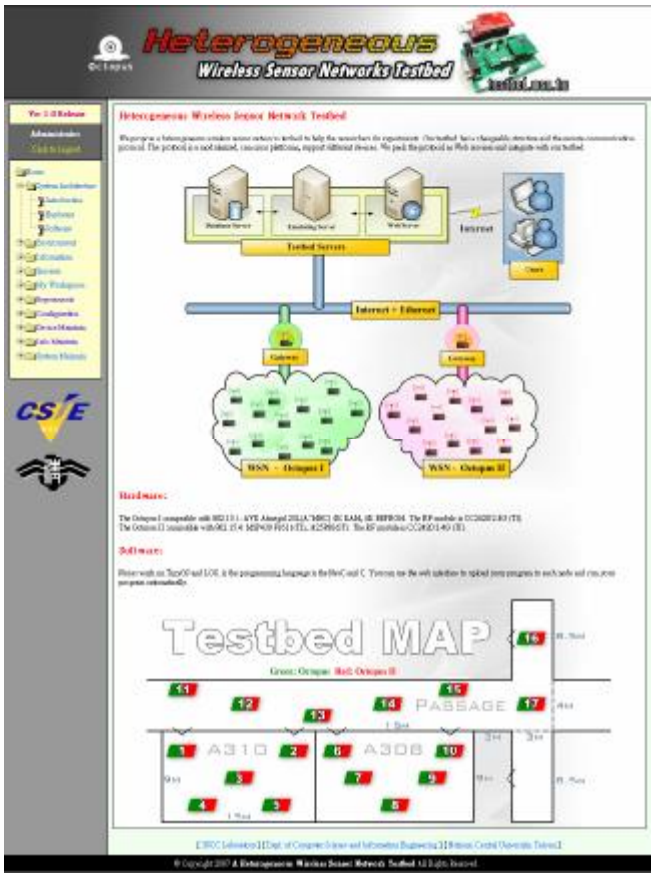

Figure IV. The screenshot of *WSNTB*

As experiments finished, users can retrieve and download results. If users use the local mode, we allow users to configure the TCP port setting and access testbed's hardware resource directly. There is a space for java code saving. We usually use the TOSBase to listen packets. The listen program is a java program in TinyOS. So we provide the java code in the java box. Then users can modify and compile the code. When a new experiment starts, users can choose the code to run from java box. Besides, users can interactive with sensor nodes in real-time by using the real-time control protocol.

A *sysadmin* can retrieve the experiment files, control the job schedule, and modify the testbed's structure. There is a

map editor for changing the location map. The listening of node, converter, and gateway allow an administrator to add or remove hardware devices. And the LAN switch port settings are to mark the location of Ethernet topology. In device maintenance menu, the *sysadmin* can adjust the devices and check their status. If a node is not work, the *sysadmin* may take action to reset the device. The information maintenance menu let the *sysadmin* to maintain these submenus: announcement, references, papers, Q&A, feedback, and about us. We provide add action, modify action, delete action, and view action for maintenance. In the system maintenance menu, the *sysadmin* can maintain system configurations such as the account, permission, contact email, and so on.

B. Experiments with WSNTB

There are two test programs running in our testbed. The first test is running an application with distributed adaptive transmission power-control algorithm, and the other one is with secure reprogramming protocols.

- **Distributed adaptive transmission power control algorithm:** In this algorithm, we proposed a distributed adaptive transmission power control algorithm. In the algorithm, each node utilizes the RSSI (Received Signal Strength Indicator) and LQI (Link Quality Indicator) of the radio to determine the appropriate transmission power for its neighbors. The algorithm can dynamically adjust the transmission power from the surrounding change. All of experiments are implemented on our testbed. The experimental results show that the adaptive transmission power-control algorithm can save 20% ~ 30% energy consumption. Besides, the algorithm can achieve at least 99% average packet delivery ratio between two nodes.
- **Secure Reprogramming Protocols**: We consider the remote reprogramming protocols in wireless sensor networks. As a remote code is distributed to the networks, users make sure of each sensor node that can receive the remote code securely through wireless medium. We implement three recently proposed secure reprogramming protocols on our testbed [3]. The three secure reprogramming protocols are based on Deluge reprogramming protocol. Finally, we also present the performance of the three reprogramming protocols.

V. CONCLUSION

WSNTB is a heterogeneous, stable, scalable, reconfigurable, and expandable architecture testbed. We allow any devices with an USB and RS232 connector as sensor device. The standard interface is easy to upgrade and expand at scale. In our department building, we constructed a testbed which consists of 34 sensor nodes, three gateways, and three servers. To test and demonstrate the presented concept we have presented a software framework to perform *WSNTB*. The web-based interface is followed by the HTTP protocol. In *WSNTB*, we can experiment cross WSNs and choose their own nodes and gateways. Users can experiment anytime and anywhere and avoid spending time in establishing the WSNs environment. The local mode, real-time control protocol, and hardware reset devices on *WSNTB* are the core value. Users can run any customized applications by themselves and work with any node over Internet. Especially, the local mode function allows users to experiment on their own computer but use our hardware resources in WSNTB. We propose a software framework for users to run their programs and collect experiment results. Using the testbed website, users can book a period of time and chooses nodes to experiment. Then users can upload their own programs and reprogramming them. We provide many functions such as compiler, online editor, data logger, working status monitor, job queue system, and so on. By using our testbed, researchers can save lots of time on building a huge experiment environment and reduce the hardware cost, but enhance the devices utility rate, and fasten to verify the experiment results.

REFERENCES

[1] Atmel AVR 8-Bit RISC processor. at http://www.atmel.com

[2] Crossbow Technology Inc. at http://www.xbow.com.

[3] P.-K. Dutta, J. -W. Hui, D. -C. Chu, and D. -E. Culler, "Securing the Deluge network programming system," *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, TN, USA, April 2006.

[4] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems", *Proceedings of Programming Language Design and Implementation* , pp. 1-11, San Diego, CA, USA, June 2003.

[5] E. Ertin, A. Arora, R. Ramnath, M. Nesterenko, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, H. Cao, "Kansei: A Testbed for Sensing at Scale", *Proceedings of International Symposium on Mobile Ad Hoc Networking and Computing*, pp 399-406, 2006.

[6] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, D. Estrin, "EmStar: A Software Environment for Developing and Deploying Wireless Sensor Networks", *Proceedings of USENIX Annual Technical Conference*, June 2004.

[7] V. Handziski, J. Polastre, J.-H. Hauer, and C. Sharp, "Flexible Hardware Abstraction of The TI MSP430 Microcontroller in TinyOS", *roceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pp 277-278, Nov. 2004.

[8] V. Handziski, A. K¨opke, A. Willig, A. Wolisz, "TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Networks", *Proceedings of International Symposium on Mobile Ad Hoc Networking and Computing*, pp 63-70, 2006.

[9] D. Johnson, D. Flickinger, T. Stack, R. Ricci, L. Stoller, R. Fish, K. Webb, M. Minor, J. Lepreau, "Emulab's Wireless Sensor Net Testbed: True Mobility, Location Precision, and Remote Access," *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, 2005.

[10] B.-H. Khan, "Web-Based Instruction", Educational Technology Publications, Feb. 1997.

[11] S. McCanne, S. Floyd., ns-2 Network Simulator. at http://www.isi.edu/nsnam/ns/.

[12] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-low Power Wireless Research", *Proceedings of Information Processing in Sensor Networks*, 2005.

[13] Omega and Motescope testbed at UC Berkeley, 802.15.4 wireless sensor network, http://www.millennium.berkeley.edu/sensornets/.

[14] J.-P. Sheu, B.-K. Hsu, P.-C. Lin, and C.-J. Chang, "Design and Implementation of A Lightweight Operating System for Wireless Sensor Networks," *Proceeding of International Computer Symposium*, Taipei, Taiwan, Dec. 2006.

[15] Texas Instruments ultra-low-power MSP430 MCUs. at http://www.ti.com

[16] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A Wireless Sensor Network Testbed", *Proceedings of Information Processing in Sensor Networks*, pp 483-488, 2005.