A distributed Wireless Sensor Network testbed with energy consumption estimation

Jang-Ping Sheu*

Department of Computer Science,
National Tsing Hua University,
Hsinchu, Taiwan 30013
E-mail: sheujp@cs.nthu.edu.tw
*Corresponding author

Chia-Chi Chang and Wei-Sheng Yang

Department of Computer Science and Information Engineering,
National Central University,
Chung-Li, Taiwan 32001
Fax: +886-3-572-3694
E-mail: chiajen@axp1.csie.ncu.edu.tw
E-mail: rick@axp1.csie.ncu.edu.tw

Abstract: In this paper, we design and implement a WSN testbed with a distributed architecture. It allows researchers to submit their experiments and to retrieve the results from the central server. In the testbed, we also propose a software-based hardware-supported scheme to measure the energy consumption of running sensor nodes. And we compare our energy-consumption estimation with oscilloscope measurements. The results show that the energy consumption is very close to oscilloscope measurements.

Keywords: embedded system; power consumption estimation; WSNs; wireless sensor networks.

Reference to this paper should be made as follows: Sheu, J-P., Chang, C-C. and Yang, W-S. (2010) 'A distributed Wireless Sensor Network testbed with energy consumption estimation', *Int. J. Ad Hoc and Ubiquitous Computing*, Vol. 6, No. 2, pp.63–74.

Biographical notes: Jang-Ping Sheu received his BS Degree in Computer Science from Tamkang University, Taiwan, Republic of China, in 1981, and his MS and PhD Degrees in Computer Science from National Tsing Hua University, Taiwan, Republic of China, in 1983 and 1987, respectively. He is currently a Chair Professor of the Department of Computer Science, National Tsing Hua University. His current research interests include wireless sensor networks, vehicular ad-hoc networks, and mobile computing. He was an associate editor of the IEEE Transactions on Parallel and Distributed Systems, *International Journal of Ad Hoc and Ubiquitous Computing*, and *International Journal of Sensor Networks*. He is an IEEE Fellow, a member of the ACM and Phi Tau Phi Society.

Chia-Chi Chang received his BS Degree in Mechanical Engineering from Chung Yuan Christian University in 1999, and his MS Degree in Mechanical Engineering from National Yunlin University of Science & Technology in 2001. He is pursuing his PhD in the Department of Computer Science and Information Engineering of National Central University. His current research interests include wireless sensor networks and embedded system design.

Wei-Sheng Yang received his BS Degree in Computer Science and Information Engineering from Tamkang University, in 2006, and his MS Degree in Computer Science and Information Engineering from the National Central University, Taiwan, in 2008. His current research interests include wireless sensor networks and mobile computing.

1 Introduction

With the advance of micro-manufacturing technology, small and inexpensive embedded devices have been widely used in various applications. A WSN consists of a large number of sensor nodes, dozens of sinks (or gateways), and some central servers. The sensor nodes are small embedded devices equipped with some sensing hardware, used to sense environment-related information like air temperature, humidity, luminosity, and even image. The sensor nodes are also able to compute and launch electromagnetic waves for communications. The sensed information is transformed

into digital data, then transferred to a sink by Radio Frequency (RF), and stored in the central server. Finally, the data collected from the network are analysed by researchers. Some common applications centre on wildlife monitoring, dangerous-environment monitoring, traffic monitoring, building monitoring, and human-health monitoring.

There are many important research subjects in the WSN field such as media-access control protocols, multi-hop routing protocols, coverage problems, and reprogramming techniques. Traditionally, simulators like NS-2 and GloMoSim (http://pcl.cs.ucla.edu/projects/glomosim/) have widely served to evaluate the performance of network protocols. Although the simulators are convenient for performance evaluation, the simulation may not present the exact effects of realistic environments for wireless communications. Many factors will influence the wireless communications of RF such as signal fading, anisotropic propagation, and path loss. Recently, there has been a shift in attention from a focus on the simulators to an emphasis on real sensor networks. Unfortunately, performance evaluation regarding real sensor networks is more complicated than simulations.

In general, the implementation of a real sensor network includes the following steps. First, the researchers implement their reprogramming protocols for the specific sensor node platform. After installation, these sensor nodes are deployed into a large-scale network. Because a sensor network may contain hundreds of sensor nodes, the installation and the sensor deployment will likely require a great deal of time. Finally, the researchers need to identify and analyse the experiment results. If some bugs are detected or some erroneous operations are conducted during the experiment, the operations in question must be repeated. Therefore, researchers require a well-suited tool that can handle those burdensome tasks. In this regard, the WSN testbed was introduced as a way to help researchers conduct performance evaluation of their algorithms. Once researchers submit the configuration of the experiment to the testbed, it automatically conducts the experiment and presents the final results. Since researchers can design their algorithms and applications on the testbed, they need not worry about how to reprogram nodes, collect the UART (Universal Asynchronous Receiver/Transmitter) log, or deploy nodes. It is worth noting here that energy consumption is recognised as one of the key parameters in WSNs. The sensor nodes usually use two AA batteries to supply power energy. If the designed protocol prodigally consumes the energy, the limited energy must be quickly exhausted and the life-time of the sensor network may be shortened (Olivares et al., 2006; Ritter et al., 2005). Thus, it is important that a WSN testbed can support the energy-consumption estimation of the running nodes.

Several WSN testbeds have been proposed in recent years. Perhaps the most famous is MoteLab (Allen et al., 2005) constructed by Harvard. MoteLab contains 190 Tmote-Sky deployed in the Maxwell Dworkin Building, and provides a web interface for users. Ohio State University constructed Kansei (Bapat et al., 2007; Ertin et al., 2006), which consists of 260 sensor nodes and 5 mobile nodes, robots. The Trio testbed (Dutta et al., 2006b) constructed by UC Berkeley deployed hundreds of sensor nodes constituting an outdoor network. No back-channel cable was connected to the sensor nodes. The reprogramming function was supported by a wireless reprogramming tool, known as Deluge (Dutta et al., 2006a); and a solar cell functioned as the power source. The TWIST (Handziski et al., 2006) testbed rests on the architecture of a heterogeneous WSN testbed. SignetLab (Crepaldi et al., 2006, 2007) and SenseNeT (Dimitriou et al., 2007) also constructed simple WSN testbeds. Mirage (Chun et al., 2005) treated the resource-allocation mechanism in WSN testbeds; SWARMS (Gruenwald et al., 2007) dealt with a design for the architecture of WSN's remote management systems. The authors in Furrer et al. (2006), Huo et al. (2006), Chao et al. (2007), Pham et al. (2006) and Sheu et al. (2008) also investigated many WSN testbeds; however, very few of these testbeds have provided the capability of energy-consumption estimation. In the current study, we integrate an 'energy-consumption estimation' mechanism into our testbed, named the University System of Taiwan TestBed (USTB).

Many researchers focus on methods to estimate the energy consumption of a single device. Regarding low levels, Wang et al. (2006) provides a study on energy consumption from the electronics viewpoint. Three conventional methods regarding high levels are summarised below. The first method (Margi et al., 2006; Allen et al., 2005) requires a connection between the multi-meter in series and the relevant devices to measure the current flow through the meter. A multiplication of the voltage would yield energy-consumption data. This method, in fact, can yield the most accurate energy-consumption data, but the cost of the hardware is prohibitive and inflexible. The second method (Jiang et al., 2007) involves using some self-designed hardware equipped with nodes to measure the energy consumption. The third method (Dunkels et al., 2007; Landsiedel et al., 2005; Pham et al., 2006; Shnayder et al., 2004) rests on software-based estimation in which the energy consumption of some component in some operating mode is multiplied by the corresponding operating time. The total energy consumption of the sensor device is the sum of all running components.

In this paper, we design and implement a WSN testbed with estimation of energy consumption. Like other testbeds, our USTB is able to schedule the submitted experiments, upload programs to nodes, and collect data logging. The USTB features a web interface with which users can submit their experiments and view the results. Our designed testbed has two contributions. The first contribution is able to link many sensor networks through the Internet and gateways, even if the networks are located at different locations. We have deployed the USTB with two WSNs in two universities. The second contribution is that we use a simple strategy (software-based and hardware-supported) to identify the accurate energy consumption of each sensor node. On the basis of oscilloscope measurements our

testbed can estimate the energy consumption of sensor nodes accurately. Thus, the users can identify the energy consumption of each sensor node by browsing the web page. If the energy profiling of some protocol is dissatisfactory, users can try to improve their protocol to improve, in turn, the power efficiency.

The rest of this paper is organised as follows. Section 2 introduces the related work of testbeds and power-consumption estimation for sensor nodes. Section 3 describes the design goal, the architecture, and the software implementation of the USTB, and the estimation for the power consumption of sensor nodes. Section 4 compares the energy consumption of experiments in the USTB with oscilloscope measurements. Section 5 concludes this paper.

2 Related works

In this section, we review some related works on WSN testbeds and on the energy-consumption estimation of sensor devices. The MoteLab (Allen et al., 2005) testbed consists of a set of permanently deployed sensor nodes connected to a central server that handles scheduling, reprogramming nodes, and data logging. In addition, each node in MoteLab is connected to a network-connected digital multi-meter, which helps continuously monitor the energy usage of the node. Trio (Dutta et al., 2006b) presented a large-scale, long-lived, 'outdoor sensor network' testbed. The solar system provided Trio with a renewable energy supply. All sensor nodes in Trio can be programmed according only to the Deluge reprogramming technique (Dutta et al., 2006a). Kansei (Bapat et al., 2007) is a heterogeneous testbed designed to facilitate research on networked sensing applications. Kansei's hardware infrastructure consists of three components: stationary array, portable array, and mobile array. The stationary array contains 210 sensor nodes deployed on 15×14 rectangular grids with 3 feet of spacing. Each portable array contains domain-specific sensors and generic software services for data storage, compression, management, and the like. The mobile array, which consists of five robotic nodes, operates on a transparent Plexiglas mobility plane. Kansei also provides a heterogeneous infrastructure to conduct experiments with 802.11b networking and Extreme Scale Motes (XSMs). SenseNeT (Dimitriou et al., 2007) presents a low-cost sensor-network testbed that depends only on a wireless channel to transfer data. It allows multiple users to interact with different portions of the testbed through a wireless channel. SignetLab (Crepaldi et al., 2006, 2007) is composed of 48 sensor nodes and deployed in a single laboratory. They provide a graphical interface for programming, sending messages, and data logging.

Developers are keenly interested in node and network lifetime, and not surprisingly, energy consumption is a crucial function for a testbed. Intuitively, we can put a multi-meter in series with the sensor nodes, and then the multi-meter would present the current. According to the power equation, the energy consumption of a node is equal to the current value multiplied by the supply voltage of the running node. MoteLab (Allen et al., 2005) has connected a node to a networked digital multi-meter, allowing the MoteLab center server to monitor the energy consumption of the node. However, the high cost of hardware makes the above-mentioned measurements unsuitable for large-scale sensor networks. The energy consumption can be measured by means of a specially designed piece of hardware attached to each sensor node. The SPOT (Jiang et al., 2007), which is special hardware attached to each sensor node, enables in situ measurement of node energy. The SPOT architecture consists of four stages: sensing, signal conditioning, digitisation, and output. In the sensing stage, a shunt resistor is put in series with a sensor node, converting current to voltage. The signal is amplified by using a differential amplifier in the signal-conditioning stage. In the digitisation stage, a Voltage-to-Frequency Converter (VFC) is used to transfer voltage signals into a periodic wave with a frequency proportional to the input voltage, which is also proportional to the power. Finally in the output stage, the output from the VFC is integrated into a counter so that accurate energy measurements can be obtained. The value of the counters can be read back by a sensor node via I2C interface.

In order to obtain energy consumption without losing flexibility, the current study proposes specific software-based estimation methods. Software-based estimation methods do not rely on complicated hardware design. The basic concept is that the total energy consumption is equal to the summation of the energy consumption of each hardware component. Because the energy consumption of a hardware component that works in some operating mode is fixed in a period of time, the total energy consumption of a component can be easily calculated as long as the operation time is obtained. For example, PowerTOSSIM (Shnayder et al., 2004) is an extension of TOSSIM (Levis et al., 2003), which is a simulator including prediction of energy consumption for TinyOS (http://www.tinyos.net/) executable codes. PowerTOSSIM first presented the hardware-power model for specific sensor nodes with microcontrollers (in active mode, idle mode, and sleep mode), radio transceivers (in transmit mode, receive mode, idle mode, and sleep mode), and LEDs' statuses. Multi-meters allow for measurement of these data in advance. Afterward, the addition of time measurements can modify TOSSIM, thereby providing accurate estimations of power consumption. AEON (Landsiedel et al., 2005) is implemented on top of the AVRORA (Titzer et al., 2005) emulator, and online energy estimation (Dunkels et al., 2007 is implemented in a Contiki operating system.

3 System design and implementation

The main design goals for our testbed are convenience, scheduling, scalability, dispersion, heterogeneity, stability, and efficiency. To achieve the convenience goal, we provide

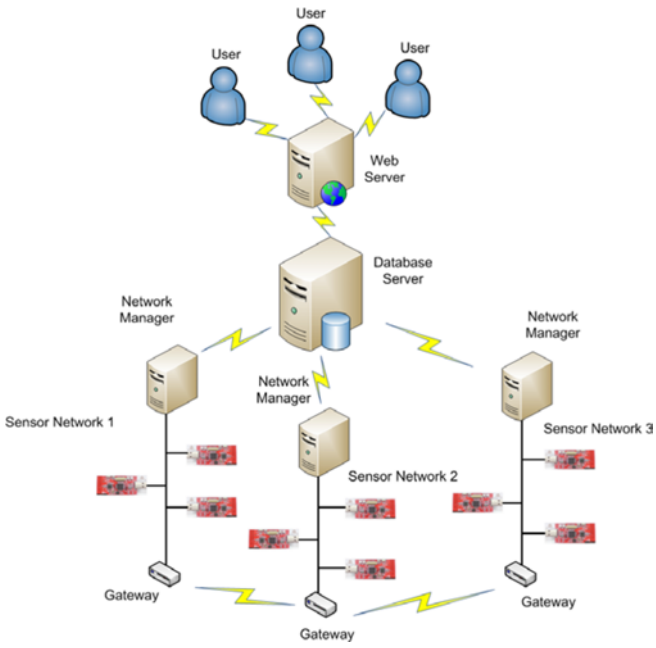
a friendly user interface for users to easily access our testbed. While experiments are submitted via user interface, the testbed should schedule these experiments according to policy prioritisation. Scalability means that the topology of the sensor network can be easily changed. We can extend or shrink a sensor network just by modifying the configuration of sensor nodes. The dispersion means that the testbed comprises several sensor networks, and these networks are located in different places. The heterogeneity allows the sensor network to contain different kinds of sensor nodes, as long as the nodes are compatible with the 802.15.4 radio standard. If some sensor nodes or a system crashes, the testbed can self-recover immediately. To facilitate the efficiency, we adopt in-line programming for our testbed.

3.1 The implementation of our testbed

Our USTB consists of many important components. Figure 1 shows the overall hardware architecture of the USTB. To accord with the above-mentioned design goals, there is no central management system managing the whole sensor network. There is only the central database to collect all data, including experiment results. The database works as an information-exchange centre. The function of each component in the USTB is individually described as follows. We built the USTB by using the Octopus II nodes (Sheu et al., 2008), which were developed in our laboratory. The Octopus II consists of a 4 MHz TI MSP430 microcontroller with 48 KB code memory and 10 KB on chip RAM, and an IEEE 802.15.4 Chipcon CC2420 radio operating at 2.4 GHz with a data rate of approximately 250 Kbps. It also provides an easy-to-use USB interface for programming and data collection, an on-board antenna, and an SMA-connector for an external antenna. The USTB testbed connects Octopus II nodes to a center server through the internet. Several WSNs can be connected through gateways. The gateway is an expandable single-board computer with a SC1100 processor running on the Linux operating system. The major purpose of gateways is to forward traffic between the WSNs and the 802.11 backbone network.

Our testbed can deploy different WSNs in different areas. A network manager functions to schedule user-submitted jobs, the reprogramming of nodes in parallel, data logging, and the storing of experiment results in databases. We implement the network manager in the C# programming language, and provide a graphic user-interface tool for managing sensor networks. The database built on the basis of the MySQL database is the centre holding all information about testbed status, about user-created experiments, and data logging generated by sensor nodes. A web interface is written in PHP and Javascript functions to carry out communications between a user and our USTB. We use the Apache2 web server to support our system. After logging on, users can submit and schedule their experiments through the web interface. Users are at liberty to retrieve logged data through the web interface or directly from the database.

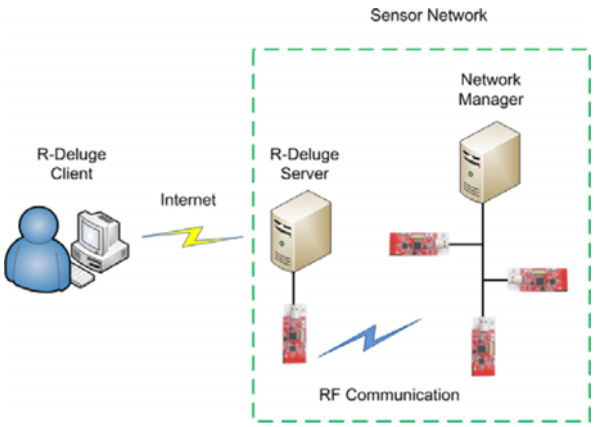
Figure 1 Overall hardware architecture (see online version for colours)



We have deployed two WSNs in two universities, respectively. One sensor network with 35 Octopus II nodes and a database server was built at National Central University, and the other sensor network with 30 nodes was built at National Tsing Hua University. Owing to the requirements that govern both the monitoring and the programming of the sensor nodes, each sensor node is connected to the backend server through the ethernet. Thus, we can directly retrieve all information pertaining to the running sensor nodes. However, the packets transmitted between sensor nodes are still over a wireless link and meet the 802.15.4 standard.

Deluge (Dutta et al., 2006a) is an important reprogramming technique without any wire connection. It provides inject-program functions, reboot functions, and reset functions for users. Users can inject a program through a radio from a source node to many destination nodes. Each sensor node can store 6 program images on its EEROM. Even though a sensor node is running, the user can run other stored programs by means of the reboot function. But the Deluge technique may not be applied in the testbed. We developed a pair of remote Deluge tools named R-Deluge, as shown in Figure 2. R-Deluge contains a server-daemon software and client software. The client part is a graphic interface software tool developed in the C# programming language. Users can log into the R-Deluge server during their own experiment time. Then, the users can use the R-Deluge tool intuitively, in line with the uses of the original Deluge tool, to inject code, to reboot, or to reset sensor nodes.

Figure 2 R-deluge architecture (see online version for colours)



The USTB also provides direct node access for users to communicate with sensor nodes directly. We developed a software tool in the C# programming language. Users can use the tool to create a TCP/IP connection for communication with a network manager. The network manager works as a communication bridge between sensor nodes and users. The tool first checks the username and password for authentication, and then checks whether or not the owner of the current running experiment matches the login user. Finally, the user can transmit data to sensor nodes and receive data from sensor nodes until completion of the experiment.

According to the architecture of the USTB, each sensor network is managed by an individual network manager, and every network manager communicates just through the database. There is no central server for management. Thus, the USTB achieves a distribution of property. The advantage is that the network topology of a sensor network can undergo changes without effects on the management system. A sensor network can be located anywhere as long as its network manager can contact the database through the ethernet. It is not necessary that the database actively contact network managers. Therefore, we can deploy a WSN by means of an existing ethernet structure in either a building or a laboratory. Each personal computer with a sensor node can execute the network-manager software. Thus, each personal computer can be viewed as a small WSN consisting of only one sensor node. Because each network manager needs only to manage one sensor node, the overhead of executing the management tool is light. We can easily deploy a number of WSNs through personal computers in a building or office. In terms of hardware cost, there is no extra requisite hardware except for sensor nodes. Consequently, there are savings relative to the cost of wires and management servers.

To enable re-use of programming code, we applied an object-oriented programming technique, polymorphism, to our network manager's design. There are many advantages in object-oriented design: for example, relative easiness characterises the writing, debugging, and maintaining of the codes. It is more important that object-oriented design should be able to improve the re-use of programming codes. If we plan to extend other kinds of sensor nodes or to change the database, the necessary work involves only overriding new classes that derive from their appropriate abstract classes and modifying the configurations without modifying other parts of the programming code.

3.2 Estimation model of energy consumption

In our testbed USTB, we adopt a software-based model while attaching extra hardware to each sensor node in order to estimate the energy consumption of each sensor node. The total energy consumption of a sensor node is equal to the summation of the energy consumption of each hardware component. The equation is

$$EC_{\text{Total}} = \sum_{\text{component } i} EC_i, \tag{1}$$

where EC_{Total} is the total energy consumption of a sensor node and EC_i is the energy consumption of the individual hardware component i. While a hardware component i works in a specific operation mode j, the component's own energy consumption is fixed during a unit of time. By multiplying the elapsed time with the unit-time energy consumption, we have the energy consumption of the hardware component. So, the energy consumption of a hardware component i can be obtained by

$$EC_i = \sum_{\text{operating mode } j} (UEC_{i,j} \times T_j), \tag{2}$$

where $UEC_{i,j}$ is the basic energy consumption of the hardware component i working in the operation mode j during a given unit time, and T_j is the time elapsed during operation mode j. We combine equations (1) and (2) in equation (3) as follows:

$$EC_{\text{Total}} = \sum_{\text{component } i} \sum_{\text{operating mode } j} (UEC_{i,j} \times T_j). \tag{3}$$

So the total energy consumption of a sensor node can be obtained by means of the relevant linear equations. We apply the above model to our USTB. The implementation details constitute two phases: a 'basic energy-consumption' measurement and a time measurement. In the 'basic energy-consumption' measurement, we measure and summarise the energy consumption of each hardware component working in each operation mode. In the time measurement, we modify the TinyOS operating-system library to measure the operation time of each operation mode of each hardware component. And then, we integrate the data collected from the above two measurement phases into each other and present the estimation results.

3.2.1 'Basic energy-consumption' measurement

Table 1 summarises the basic current draw of each hardware component in Octopus II. We divide the Octopus II into four types of hardware components: the microcontroller, LEDs, the RF chip, and the other small electronic components like resistors and transistors on the Printed Circuit Board (PCB). For measuring the current draw of each hardware component, we can program the Octopus II

to manipulate each hardware component into some specific operation mode, and then can put the Octopus II in series with the power supply and a multi-meter. Multiplying the measured current by the supplied voltage yields the sensor node's consumed energy. Therefore, the multi-meter can help yield the energy consumption attributable to the Octopus II, when operated in the specific operation mode cited.

Table 1 The basic power-consumption model for the Octopus II

Mode	*Current* (mA)	*Mode*	*Current* (mA)
CPU		*Radio*	
Active with PCB	2.75	RX	18.8
LPM1 with PCB	0.85	TX (–25 dmb, level 3)	8.5
LPM3 with PCB	0.70	TX (–15 dbm, level 7)	9.9
LEDs		TX (–10 dbm, level 11)	11.2
Red	4.76	TX (–7 dbm, level 15)	12.5
Green	4.13	TX (–5 dbm, level 19)	13.9
Blue	3.12	TX (–3 dbm, level 23)	15.2
Yellow	9.66	TX (–1 dbm, level 27)	16.5
		TX (0 dbm, level 31)	17.4

Since the TinyOS never turns off the microcontroller while the power is supplied, we aggregate the energy consumption of both the microcontroller and the other small electronic components on the PCB. The MSP430 microcontroller is able to work in six operating modes: active mode and five levels of low-power mode (LPM0, LPM1, LPM2, LPM3, and LPM4). However, the MSP430 microcontroller never shifts into the mode of LPM0, LPM2, or LPM4 during use of the TinyOS. So, we installed into the Octopus II a sensor application that disables LEDs and the RF chip and that then operates the microcontroller in active mode. The energy consumption of the PCB and the microcontroller that operates in active mode is 2.75 mA. Correspondingly, the energy consumption of LPM1 and LPM3 with the PCB are 0.85 mA and 0.70 mA, respectively. The Octopus II consists of four different colour LEDs. We programmed a sensor application that enables only one LED and that operates the microcontroller in LPM3 mode, and we installed this application into the Octopus II. According to the same measurement method as the above one, the measured value minus the energy consumption of both the microcontroller and the PCB is the energy consumption of the LED. The measurement results show that LEDs differed from one another regarding energy consumption. The yellow LED is the most extravagant one, consuming 9.66 mA. The blue LED is the most frugal one, as it consumes 3.12 mA.

In the RF chip, the CC2420 has a lot of operation modes like the power-down mode (or the sleep mode), the idle mode, the transmit mode, and the receive mode (Titzer et al., 2005). In the sleep mode, most of the CC2420 hardware components are switched off. In the idle mode, the voltage regulator and the oscillator are switched on, but the transceiver is inactive. The transmission component of the transceiver is active and the antenna radiates energy while the CC2420 is in the transmit (TX) mode, and while the CC2420 is in the receive (RX) mode, the transceiver is ready to receive packets. We consulted the CC2420 hardware specifications (Titzer et al., 2005) and have listed both the energy consumption of the transmit mode with eight transmission power levels and the energy consumption of the receive mode, as presented in the specifications. We do not present the energy-consumption data of the UART because the measurement of the sensor node's energy consumption usually takes place during the final step of the sensor-application development; thus, the UART transmission is useless because any bug should have been fixed before the sensor application takes effect online. Besides, we also use the same UART port to transmit the power profiling of a sensor node.

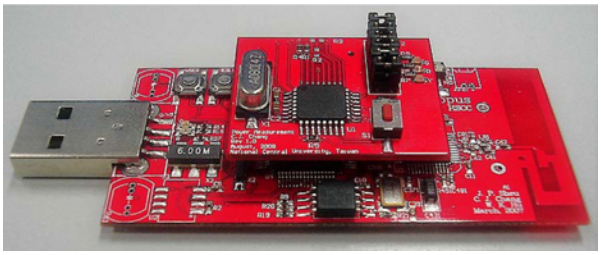
3.2.2 Time measurement

The MSP430 microcontroller contains many input and output pins, but some of them are never used by the TinyOS. We found 15 unused pins, which we have named 'Unused PINs' (UPINs). By modifying the TinyOS libraries, we can control and monitor the status (high or low) of every UPIN. We can set some of the UPINs to high or low during the operation mode of some hardware component and obtain the operation time by detecting the status of UPINs in the whole execution time. For detecting the status of UPINs, we use an extra ATmege168 microcontroller with 16 KB of code memory, named 'E-MCU', connected to the Octopus II as shown in Figure 3; in this way, we can monitor the status of the UPINs and report to the network manager through the UART port. Table 2 shows the 15 UPINs corresponding to the hardware components that we want to measure.

Table 2 The UPINs corresponding to hardware components

Pins	*Hardware components*	*Pins*	*Hardware components*
UPIN_1	MSP430 Active mode	UPIN_9	CC2420 TX (levels 24~31)
UPIN_2	MSP430 LPM1 mode	UPIN_10	CC2420 TX (levels 20~23)
UPIN_3	MSP430 LPM3 mode	UPIN_11	CC2420 TX (levels 16~19)
UPIN_4	Red LED	UPIN_12	CC2420 TX (levels 12~15)
UPIN_5	Yellow LED	UPIN_13	CC2420 TX (levels 8~11)
UPIN_6	Blue LED	UPIN_14	CC2420 TX (levels 4~7)
UPIN_7	Green LED	UPIN_15	CC2420 TX (level 3)
UPIN_8	CC2420 RX mode		

Figure 3 The E-MCU platform attached to an Octopus II node (see online version for colours)



In general, we can modify the TinyOS libraries to independently control the state (low level or high level) of each UPIN. While the microcontroller is running, UPIN_1 is set to the high level, and UPIN_2 and UPIN_3 are set to the low level. The E-MCU can measure the period from the high level to the low level on UPIN_1. According to equation (2), the energy consumption of the running microcontroller can be easily estimated. For LED displays, we use four UPINs (UPIN_4 to UPIN_7) to represent four LEDs respectively. In the RF mode, we use UPIN_8 to represent the receiving mode of the CC2420 radio chip. But we use seven UPINs (UPIN_9 to UPIN_15) to represent the seven power-transmission levels, as shown in Table 2.

For the MSP430 microcontroller, we have to record the elapsed time of the active, LPM1, and LPM3 operation modes. Therefore, we use the three UPINs from UPIN_1 to UPIN_3 to represent the occupancy time of each operation mode. When the TinyOS program is started, UPIN_1 is initialised to high, and UPIN_2 and UPIN_3 are initialised to low. We modify the TinyOS libraries to accomplish two jobs:

- to set UPIN_1 to low and to set either UPIN_2 or UPIN_3 to high before the microcontroller enters the low-power mode
- to set UPIN_1 to high and to set both UPIN_2 and UPIN_3 to low after the microcontroller wakes up.

To represent the four LEDs, we use the four UPINs from UPIN_4 to UPIN_7. Similarly in the RF transceiver, we use UPIN_8 to represent the receiving mode of the CC2420 radio chip. But we use the seven UPINs from UPIN_9 to UPIN_15 to represent the seven different power-transmission levels. Since the CC2420 has 29 power levels, from level 3 to level 31, several levels of CC2420 will be mapped to one of the seven levels, as shown in Table 2. According to the energy consumption of CC2420 as shown in Table 1, we combine the 29 power levels into seven levels appropriately, as shown in Table 2. For example, when the TinyOS uses a power level between 24 and 31 to transmit packets, UPIN_9 should be the pin of choice and the energy consumption is the average of energy consumed in level 31 and level 23. A packet transmission is enabled by issuing a 'STXON' or 'STXONCCA' command. During transmission of a packet, the SFD pin in the CC2420 chip is always set at high, and the SFD pin shifts to a low setting when the packet is completely transmitted. Therefore, we can detect the SFD pin to measure the transmission time of the CC2420 chip. Also, we can obtain the operation time of the receive mode by detecting the start time and the stop time of the CC2420 software components, except for the packet-transmission time.

4 Experiments

Basically, the energy consumption in a sensor node only includes the energy consumed by the hardware components of the microcontroller, the RF transceiver, and the LEDs of the node. The microcontroller provides various execution modes with different energy consumption such as the active mode, the LPM1 sleep mode, and the LPM3 deep-sleep mode. The LEDs' power-consumption levels differ from one another according to the LEDs' colour. In addition, the RF transceiver has four operational states: the transmit, receive, idle, and sleep states. Each state has its own power consumption. If we have the energy consumption of various modes of the microcontroller, the LED colours, and the RF transceiver's operational states, we can count the energy consumption of any program executed on a sensor node through a summary of the energy consumed by each component of the microcontroller, the LEDs, and the RF. Therefore, we use a multi-meter to measure the current draw of each state of each hardware component. The E-MCU functions to retrieve the execution time of each hardware component in a sensor node. The E-MCU can monitor the specific MSP430 I/O pins to count the execution time of each hardware component. The total energy consumption of a sensor-node program is the summation of the consumed current of each hardware module multiplied by the module's execution time. In our experiments, the energy consumption of each hardware component in different modes or states is validated by means of an oscilloscope.

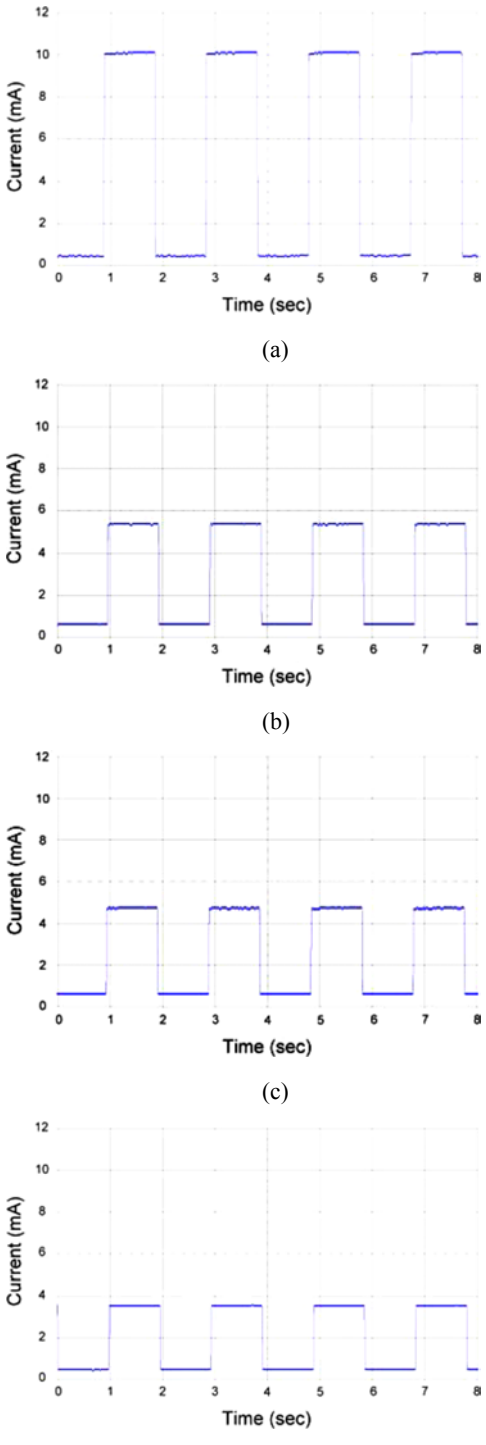
In this section, we compare the energy consumption of sensor nodes estimated by our testbed with the energy consumption measured by an Agilent DSO6032A oscilloscope. In order to measure the accurate energy consumption with an oscilloscope, we attached a shunt resistor (1 ohm) to the Octopus II sensor node. Since the resistor is very small, we can ignore its energy consumption on our energy measurement. Therefore, the real current draw can be proportionally converted to a voltage signal. We supply a 3 V power source to run each sensor application for 60 s and measure the energy consumption by means of both the oscilloscope and our testbed.

4.1 Experiment 1: Blink

The Blink is a simple application that uses a timer to blink LEDs periodically. The LEDs are flashed in a two-second temporal cycle. That is, the LED is on for one second and then off for another one second. The current draw of each LED measured by the oscilloscope is shown in Figure 4. In Figure 4(a), the Octopus Ⅱ consumes 10.36 mA while the yellow LED is turned on, and 0.7 mA while the yellow LED is turned off. This finding means that the yellow LED may consume 9.66 mA alone. Similarly, the red,

green, and blue LEDs consume 4.76 mA, 4.13 mA, and 3.12 mA, respectively. According to equation (2), the yellow Blink application consumes about 995 mJ (0.7 Ma × 3 V × 30 sec + 10.36 mA × 3 V × 30 sec). Similarly, the Blink applications with red, green, and blue LEDs consume 554 mJ, 498 mJ, and 407 mJ, respectively.

Figure 4 The current draw of Blink applications: (a) blink for yellow LED; (b) blink for red LED; (c) blink for green LED and (d) blink for blue LED (see online version for colours)



With the extra hardware of the E-MCU, we can monitor the status of UPINs and report results through the UART port. In the experiments, the MSP430 is 0.04% in the active mode and 99.96% in the LPM3 sleep mode. Lighting the LED is 48.5% in the active mode. These results are consistent with the results concerning the program behaviour of Blink applications. The total energy-consumption figures as estimated by our testbed are 969 mJ ((2.75 mA × 0.04% + 0.7 mA × 99.96% + 9.66 mA × 48.5%) × 3 V × 60 sec), 542 mJ, 487 mJ, and 399 mJ for the Blink applications with yellow, red, green, and blue LEDs, respectively. The maximum-difference percentage of the energy-consumption estimations regarding a comparison between our scheme and the oscilloscope is only 2.6%. The main energy-consumption difference between the oscilloscope and our testbed is that the active time of LEDs in the oscilloscope measurements is longer than our E-MCU measurements. This difference arises because the oscilloscope cannot show the exact duration of the square pulses, as shown in Figure 4. We can estimate their duration only by means of applications' semantics. And whereas our scheme can discover that the microcontroller has a very small percentage in the active mode, the oscilloscope cannot show this active mode owing to the oscilloscope's display limitations. However, the energy consumption measured by our scheme is still very close to that of the oscilloscope.

4.2 Experiment 2: CPU-load test

In this experiment, we disable the LEDs and focus on the energy consumption of the MSP430 microcontroller. Our experiment with the two CPULoadTest applications involves different CPU loads, and sets the timer period to one second. When the timer is triggered, the first application executes some instructions, as shown in Figure 5(a), and the second application executes more instructions than the first one, as shown in Figure 5(b). The MSP430 will return to the LPM3 mode after the instructions are executed. Our testbed yielded the following energy-consumption results. The light CPU-load application spends 6.68% in the active mode and 93.32% in the LPM3 sleeping mode, and the heavy CPU-load application spends 91.88% in the active mode and 8.12% in the LPM3 sleeping mode. Each of the two applications consumes 2.75 mA in the active mode and 0.7 mA in the LPM3 mode. So, the energy-consumption totals of these two applications are 150.65 mJ ((2.75 mA × 6.68% + 0.7 mA × 93.32%) × 3 V × 60 sec) and 465.04 mJ, respectively. Figure 6 shows the current draw of these two CPU-load test applications measured by the oscilloscope. The first application consumes 151.2 mJ (2.75 mA × 3 V × 6.83% × 60 s + 0.7 mA × 3 V × 93.17% × 60 sec) and the second one consumes 466.48 mJ (2.75 mA × 3 V × 92.27% × 60 s + 0.7 mA × 3 V × 7.73% × 60 s). The difference percentages of these two CPU-load applications, as measured by our testbed and the oscilloscope, are only 0.37% and 0.31%, respectively.

Figure 5 The pseudo code for different CPU loadings: (a) light loading procedure and (b) heavy loading procedure

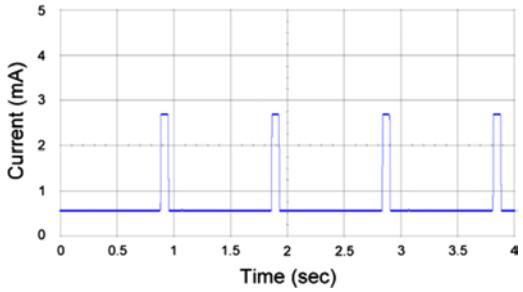
```
event void Timer.fired( )
{
   uint16_t i,j ;
    for ( i=0 ; i<5000 ; i++)
    {
        for ( j=0 ; j<10 ; j++)
        {
              // do nothing
        }
    }
}
```

(a)

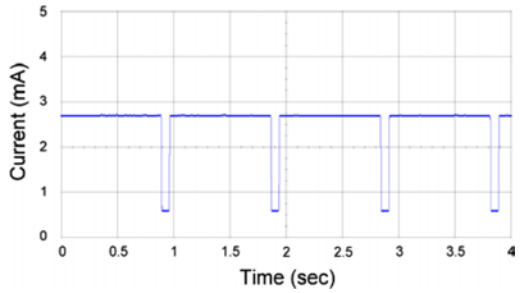
```
event void Timer.fired( )
{
   uint16_t i,j ;
    for ( i=0 ; i<5000 ; i++)
    {
        for ( j=0 ; j<150 ; j++)
        {
              // do nothing
        }
    }
}
```

(b)

Figure 6 The current draw of CPULoadTest applications: (a) with active time 50 ms and (b) with active time 900 ms (see online version for colours)



(a)

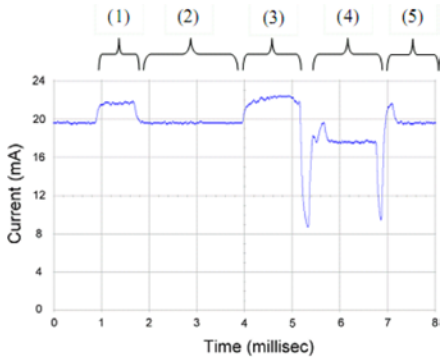


(b)

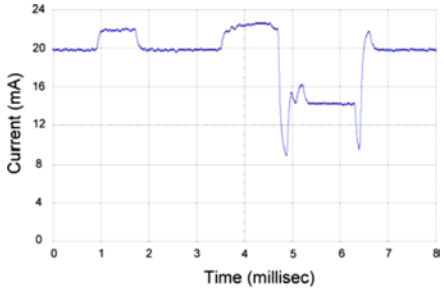
4.3 Experiment 3: CountSend

CountSend is a TinyOS application that transmits the value of a counter through an RF transceiver while the timer is triggered. We use three RF output power levels (level 31, level 19, and level 3) to send packets regularly every 200 milliseconds. Figure 7 presents the current draws of the applications, as measured by the oscilloscope, in relation to the three power levels. The packet transmission in the CC2420 can be divided into five phases, as shown in Figure 7(a). In phase (1), the timer is triggered and a transmission task is posted to the task queue. In phase (2), the packet is waiting for a random back-off. After the back-off time, the packet is written into the transmission buffer of the CC2420 in phase (3). In phase (4), the RF transceiver is trying to transmit the queued packet. In phase (5), the RF transceiver and the microcontroller will return to the RECEIVE mode and the LPM3 mode, respectively.

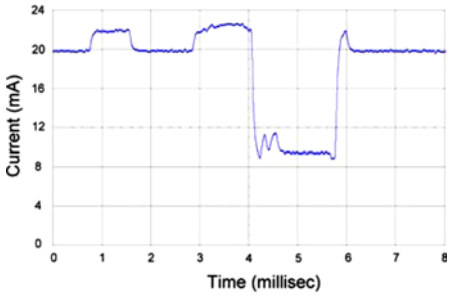
Figure 7 The current draws of CountSend applications without LEDs: (a) power level 31 (0 dbm); (b) power level 19 (–5 dbm) and (c) power level 3 (–25 dbm) (see online version for colours)



(a)



(b)



(c)

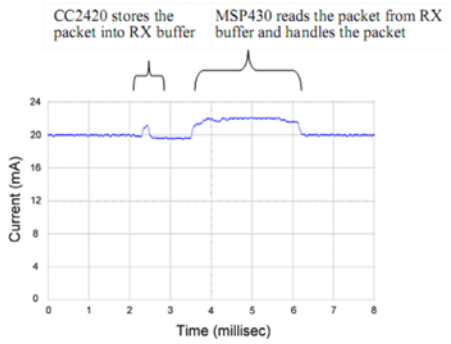
The length of the back-off time ranges from one to 16 time slots, and each time slot is 320 microseconds. In Figure 7(a)–(c), there are three different RF power levels: level 31 (17.6 mA), level 19 (14.27 mA), and level 3 (9.45 mA), respectively. Because the oscilloscope-measured current draw is unstable with time, we can only estimate

the current draw as it is in Figure 7(a). The current draw in phases 2 and 5 is about 19.57 mA, the current draw in phases 1 and 3 is about 21.7 mA, and the current draw in phase 4 is about 17.6 mA. Since the timer period is 200 milliseconds, there are 300 packet transmissions in one minute. Let us assume that the duration time of phases 1, 3, and 4 are 1 ms, 1 ms, and 2 ms, respectively. Therefore, the energy-consumption totals corresponding to these three power levels, as measured by the oscilloscope, are about 3524 mJ ((21.7 mA × 0.002 s + 17.6 mA × 0.0015 s + 19.57 mA × (60 s – 0.002 s – 0.0015 sec)) × 3 V × 300), 3519 mJ, and 3513 mJ, respectively. In our testbed, the microcontroller spends 1.44% of the time in the active mode, and 98.56% of the time in the LPM3 mode. The RF transceiver occupies 0.79% of the time in the transmit mode and 99.21% of the time in the receive mode. The total energy consumption of the application with the maximum power-transmission level (level 31) is 3513 mJ ((2.75 mA × 44% + 0.7 mA × 98.56% + 17.4 mA × 0.79% + 18.8 mA × 99.21) × 3 V × 60 s). Based on the same estimation, the energy-consumption totals corresponding to output power level 19 and level 3 are 3508 mJ and 3501 mJ, respectively. The difference percentage measured by our testbed and the oscilloscope is less than 0.34%. In this experiment, we can see that there is a very small difference regarding the energy consumed by the applications with maximum power level 31 and minimum power level 3.

4.4 Experiment 4: TOSBase

The TOSBase is a sensor application of the TinyOS wherein the sensor node works as a data sink. It shifts the radio chip into the receive mode and waits for radio packets. The radio chip's receive buffer stores the coming packet and generates an interruption to notify the microcontroller. We have an Octopus II that we installed in the TOSBase and that works as a receiver, and we have another Octopus II that is installed in the CountSend application and that works as a sender. The packet-dispatch timer in the CountSend application is set at one second. Each packet length is 29 bytes. Figure 8 presents the application's current draws. We can see that when a packet is received, the CC2420 stores it in the receive buffer, and that there is an interruption whereby the microcontroller can read the packet from the receive buffer to the main memory. The current draw in the receive mode is about 19.57 mA and rises to 21.7 mA for about 3 milliseconds after receipt of a packet. Therefore, the total energy consumed by the application receiving packets every one second is about 3524 mJ (21.7 mA × 3 V × 0.003 s × 60 + 19.57 mA × 3 V × (60 s – 0.003 s) × 60). In the results of our testbed, the microcontroller spends 0.27% in the active mode and the RF transceiver spends 99.73% in the receive mode. Thus, the total energy consumption is 3511 mJ ((2.75 mA × 0.27% + 0.7 mA × 99.73%) × 60 s × 3 V + (18.8 mA × 3 V × 60 s)). The difference percentage of the energy consumption measured by our testbed and the oscilloscope is about 0.4%.

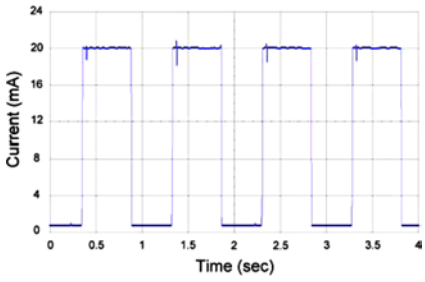
Figure 8 The current draw of TOSBase applications (see online version for colours)



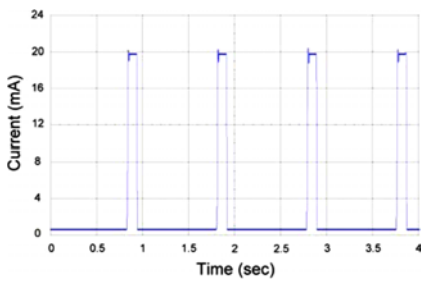
4.5 Experiment 5: RFPowerManagement test

In the RFPowerManagementTest application, the RF chip will be periodically turned off to extend the node and the network lifetime. In the first application, the RF is turned on (wake-up mode) for 500 milliseconds for receiving or transmitting packets, and then is turned off (sleepmode) for 500 milliseconds in repeating cycles. In the second application, the RF is turned on for 100 milliseconds and then is turned off for 900 milliseconds in cycles. After the RF is turned on, both of the applications transmit a 29-byte length packet through the RF chip with the maximum output power level. The consumed current of the two applications, as measured by the oscilloscope, are shown in Figure 9. The energy consumption of the first application and the second application, as measured by the oscilloscope, are about 1969 mJ ((19.57 mA × 54.25% + 0.7 mA × 45.75%) × 3 V × 60 s) and 483 mJ ((19.57 mA × 10.52% + 0.7 mA × 89.48%) × 3 V × 60 s), respectively.

Figure 9 The current draw of RFPowerManagementTest applications: (a) the RF is turned on for 500 ms every second and (b) The RF is turned on for 100 ms every second (see online version for colours)



(a)



(b)

According to our testbed measurements for the first application, the MSP430 spends 0.96% in the active mode and 99.04% in the LPM3 sleeping mode, and the RF chip spends 0.15% in the transmit mode and 53.08% in the receive mode. However, in the second application the MSP430 spends 0.97% in the active mode and 99.03% in the LPM3 sleeping mode, and the RF chip spends 0.15% in the transmit mode and 10.69% in the receive mode. Thus, the total energy consumption of the first application is (2.75 mA × 0.96% + 0.7 mA × 99.04%) × 3 V × 60 s + (17. 4 mA × 0.15% + 18.8 mA × 53.08%) × 3 V × 60 s = 1930 m J. Similarly, we can calculate that the energy consumption of the second application is 496 mJ. The difference percentages of the energy consumption, as measured by the oscilloscope and our testbed for the first application and second application, are 1.9% and 2.6%, respectively.

All results of energy consumption are summarised in Table 3. The maximum difference percentages of the experiments, as measured by the oscilloscope and our testbed, are less than 2.6%. The differences result mainly from the difficulty in accurately obtaining the energy consumption with the oscilloscope, and the following two reasons help explain this difficulty. One is the instability of the current draw. The other reason is that the duration of each current value is difficult to calculate from the screen of the oscilloscope. We may subjectively overestimate or underestimate the value on the basis of the oscilloscope's display. Therefore, our testbed provides a good tool that can estimate the energy consumption accurately in various applications of sensor networks. The testbed has been used in wireless sensor networks for two years. Our testbed can test and complete several exercises and projects such as the design of power-control routing protocols, data-aggregation protocols, and time-synchronisation protocols. We received considerable feedback from students and improved our testbed accordingly. Thus, we believe that our testbed is a useful tool that can help users develop their applications in WSNs.

Table 3 The comparisons of energy consumption measured by our scheme and the oscilloscope

	Measured by		
Experiments	*Oscilloscope* (mJ)	*Our testbed* (mJ)	*Difference* (%)
Blink (Yellow)	995	969	2.6
Blink (Red)	554	542	2.2
Blink (Green)	498	487	2.2
Blink (Blue)	407	399	2.0
CountLeds	2059	2078	0.93
CPULoadTest (5%)	151.20	150.65	0.37
CPULoadTest (90%)	466.48	465.04	0.31
CountSend (power level 31)	3524	3513	0.29
CountSend (power level 19)	3519	3508	0.31
CountSend (power level 3)	3513	3501	0.34
RFPowerManagementTest (50%)	1969	1930	1.9
RFPowerManagementTest (10%)	483	496	2.6
TOSBase (a packet per second)	3524	3511	0.37

5 Conclusions

It is important to estimate the energy consumption of running sensor nodes in WSNs. In this paper, we present a WSN testbed called the USTB (http://wsn.tw/), which estimates the energy consumption of each sensor node. The contributions of the USTB are twofold in comparison to the systems outlined in previous studies. The first contribution is the scalability of the USTB, which can combine several small WSNs into one large WSN even though those small WSNs are distributed on different locations. Our second contribution is to have developed an on-line 'energy-consumption measurement' device called the E-MCU. While the sensor nodes are equipped with the E-MCU, users can easily retrieve the energy consumption of the selected nodes. Our E-MCU is independent of the operations systems. Some state-of-the-art simulators like the TOSSIM and the PowerTOSSIM cannot fulfil all the requirements for real WSN applications. The E-MCU can measure the energy consumption of real hardware modules accurately and is validated by the oscilloscope measurements. In our experiments, the maximum difference between the E-MCU and the oscilloscope is less than 2.6%.

References

Allen, G.W., Swieskowski, P. and Welsh, M. (2005) 'MoteLab: a wireless sensor network testbed', *Proceedings of the 4th International Conference on Information Processing in Sensor Networks*, April, California, USA, pp.483–488.

Bapat, S., Leal, W., Kwon, T., Wei, P. and Arora, A. (2007) 'Chowkidar: a health monitor for wireless sensor network testbeds', *Proceedings of the 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, May, FL, USA, pp.1–10.

Chao, D., Datt, K., Rekhi, J. and Learmonth, G.P. (2007) 'Human-in-the-loop simulation testbed for wireless sensor networking', *Proceedings of Systems and Information Engineering Design Symposium*, April, Charlottesville, VA, USA, pp.1–6.

Chun, B.N., Buonadonna, P., AuYoungz, A., Ng, C., Parkes, D.C., Shneidman, J., Snoeren, A.C. and Vahdat, A. (2005) 'Mirage: a microeconomic resource allocation system for sensornet testbeds', *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors*, May, Sydney, Australia, pp.19–28.

Crepaldi, R., Friso, S., Harris III, A., Mastrogiovanni, M., Petrioli, C., Rossi, M., Zanella, A. and Zorzi, M. (2007) 'The design, deployment, and analysis of signetlab: a sensor network testbed and interactive management tool', *Proceedings of the 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, May, FL, USA, pp.93–94.

Crepaldi, R., Harris, A., Scarpa, A., Zanella, A. and Zorzi, M. (2006) 'SignetLab: deployable sensor network testbed and management tool', *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, Colorado, USA, pp.375–376.

Dimitriou, T., Kolokouris, J. and Zarokostas, N. (2007) 'SenseNeT: a wireless sensor network testbed', *Proceedings of the 10th ACM Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, Crete Island, Greece, pp.143–150.

Dunkels, A., Ɵsterlind, F., Tsiftes, N. and He, Z. (2007) 'Software-based on-line energy estimation for sensor nodes', *Proceedings of the 4th IEEE Workshop on Embedded Networked Sensors*, Cork, Ireland, pp.28–32.

Dutta, P.K., Hui, J.W., Chu, D.C. and Culler, D.E. (2006a) 'Securing the Deluge Network Programming System', *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, April, Nashville, TN, USA, pp.326–333.

Dutta, P., Hui, J., Jeong, J., Kim, S., Sharp, C., Taneja, J., Tolle, G., Whitehouse, K. and Culler, D. (2006b) 'Trio: enabling sustainable and scalable outdoor wireless sensor network deployments', *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, Nashville, TN, USA, pp.407–415.

Ertin, E., Arora, A., Ramnath, R., Nesterenko, M., Naik, V., Bapat, S., Kulathumani, V., Sridharan, M., Zhang, H. and Cao, H. (2006) 'Kansei: a testbed for sensing at scale', *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, July, Nashville, TN, USA, pp.399–406.

Furrer, S., Schott, W., Truong, H.L. and Weiss, B. (2006) 'The IBM wireless sensor networking testbed', *Proceedings of the 2nd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, July, Barcelona, Spain, pp.41–46.

Gruenwald, C., Hustvedt, A., Beach, A. and Han, R. (2007) 'SWARMS: a sensornet wide area remote management system', *Proceedings of the 3rd international conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, May, FL, USA, pp.1–10.

Handziski, V., Kopke, A., Willig, A. and Wolisz, A. (2006) 'TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks', *Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality*, Florence, Italy, pp.63–70.

Huo, H., Zhang, H., Niu, Y., Gao, S., Li, Z. and Zhang, S. (2006) 'MSRLab6: an IPv6 wireless sensor networks testbed', *Proceedings of the 8th International Conference on Signal Processing*, Beijing, China, Vol. 4, November.

Jiang, X., Dutta, P., Culler, D. and Stoica, I. (2007) 'Micro power meter for energy monitoring of wireless sensor networks at scale', *Proceedings of the 6th International Conference on Information Processing in Sensor Networks*, April, Cambridge, MA, England, pp.186–195.

Landsiedel, O., Wehrle, K. and Gotz, S. (2005) 'Accurate prediction of power consumption in sensor networks', *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensor*, May, Sydney, Australia, pp.37–44.

Levis, P., Lee, N., Welsh, M. and Culler, D. (2003) 'TOSSIM: accurate and scalable simulation of entire tinyos applications', *Proceedings of the First International Conference on Embedded Networked Sensor Systems*, California, USA, pp.126–137.

Margi, C.B., Petkov, V., Obraczka, K. and Manduchi, R. (2006) 'Characterizing energy consumption in a visual sensor network testbed', *Proceedings of the 2nd international conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, July, Barcelona, Spain, pp.331–339.

Olivares, T., Tirado, P.J., Orozco-Barbosa, L., López, V. and Pedrón, P. (2006) 'Simulation of power-aware wireless sensor network architectures', *Proceedings of the ACM International Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks*, Terromolinos, Spain, pp.32–39.

Pham, N.N., Youn, J. and Won, C. (2006) 'A comparison of wireless sensor network routing protocols on an experimental testbed', *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, Taichung*, June, Taiwan, pp.276–281.

Ritter, H., Schiller, J., Voigt, T. and Dunkels, A. (2005) 'Experimental evaluation of lifetime bounds for wireless sensor networks', *Proceedings of the 2nd European Workshop on Wireless Sensor Networks*, California, USA, February, pp.25–32.

Sheu, J.P., Chang, C.J. and Sun, C.Y. (2008) 'WSNTB: a testbed for heterogeneous wireless sensor networks', *Proceedings of the First IEEE International Conference on Ubi-media Computing and Workshops*, July, Lanzhou, China, pp.338–343.

Shnayder, V., Hempstead, M., Chen, B., Allen, G.W. and Welsh, M. (2004) 'Simulating the power consumption of largescale network applications', *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, Baltimore, MD, USA, pp.188–200.

Titzer, B.L., Lee, D.K. and Palsberg, J. (2005) 'Avrora: scalable sensor network simulation with precise timing', *Procedings of the 4th International Symposium on Information Processing in Sensor Networks*, April, California, USA, pp.477–482.

Wang, Q., Hempstead, M. and Yang, W. (2006) 'A realistic power consumption model for wireless sensor network devices', *Proceedings of the 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, Reston, September VA, USA, pp.286–295.

Websites

Global Mobile Information Systems Simulation Library. http://pcl.cs.ucla.edu/projects/glomosim/

TinyOS: an open-source operating system designed for wireless embedded sensor networks. http://www.tinyos.net/