VMNet: Realistic Emulation of Wireless Sensor Networks

Hejun Wu, *Student Member, IEEE*, Qiong Luo,
Pei Zheng, *Member, IEEE*, and Lionel M. Ni, *Fellow, IEEE*

Abstract—Many research activities on Wireless Sensor Networks (WSNs) need detailed performance statistics about protocols, systems, and applications; however, current simulation tools and testbeds lack mechanisms to report these statistics realistically and conveniently. To address this need, we have developed a WSN emulator, VMNet. VMNet emulates networked sensor nodes at the level of CPU clock cycles and executes the binary code of real applications directly. It emulates the radio channel with loss and noise as well as emulates the peripherals in sufficient detail. Moreover, VMNet takes parameter values from the real world and logs detailed runtime information of emulated nodes. Consequently, the application performance, both in response time and in power consumption, is reported realistically in VMNet, as demonstrated by our comparison studies with real sensor networks.

Index Terms—Emulation, wireless sensor networks, performance evaluation.

1 INTRODUCTION

IN-DEPTH studies on wireless sensor networks (WSNs) require realistic simulation and emulation environments. Even though field studies on real WSNs are of great impact [8], [17] and recent real WSN testbeds such as MoteLab [31] are also useful for controllable experiments, the deployment, maintenance, and data collection tasks of these real WSNs are labor-intensive and error-prone. Moreover, studying the performance of WSN protocols and applications needs results in various settings, on various aspects of the system, and at various levels of details. Therefore, it is desirable that the experiments are conducted in a realistic but conveniently controllable environment. As evidence, recent work (e.g., EmStar [11], TOSSIM [15], and PowerTOSSIM [24]) has provided such software environments for the initial development and deployment of WSN applications. In this paper, we follow up with VMNet [30], a WSN emulator focusing on enabling realistic performance evaluation of WSNs.

We choose emulation over simulation in VMNet for fidelity and user convenience. As the performance of WSNs involves various factors, we start from detailed emulation and gradually raise the level of abstraction rather than abstracting a few performance factors in the very beginning. Furthermore, emulation is convenient to users since the binary code of a target system can be executed directly in an emulator. This feature also eliminates possible interference from implementation and compilation. Additionally, emulation of an entire system allows the evaluation of cross-layer techniques for OSs, protocols, and applications of WSNs.

In VMNet, a target WSN is emulated as a VMN (Virtual Mote Network). Each sensor node (mote) is emulated as a VM (Virtual Mote). The CPU of a mote is emulated at the clock cycle level, and the sensing units and other hardware peripherals are emulated in sufficient detail. The radio signal transmission is emulated by the communication between VMs with the effects of path loss and noise. Finally, the binary code of a target WSN application can be run directly on the VMN for debugging, testing, and, most importantly, performance evaluation.

We select two performance metrics for evaluation: One is response time and the other power consumption. Response time is critical to some WSN applications, such as chemical detection, whereas power consumption is the major concern for applications in which replacing batteries is inconvenient or infeasible. In order to evaluate the response time and power consumption, VMNet logs detailed running status of the application code under evaluation. The running status is categorized by the operations of the CPU, the flash memory, the sensor board, and the radio module. The virtual running time (i.e., the time in the emulated world) of each operation is also logged. Using these logged statistics from emulation and the parameter values (e.g., electric current) from real-world measurement, VMNet is able to calculate the two performance metrics.

We have evaluated VMNet in comparison with real WSNs, TOSSIM, and PowerTOSSIM. Our results demonstrate that, with the basic measurement statistics of real WSNs and the running status logged in VMNet, the estimated running time and power consumption in VMNet are close to those in the target WSNs. In addition, VMNet facilitates detailed studies on data delivery schemes and network protocols. Due to its high fidelity, VMNet has an emulation speed of 1/10 of that of TOSSIM on average.

The remainder of this paper is organized as follows: Section 2 describes the VMNet architecture and the VM organization. Section 3 presents the design and implementation of VMNet in detail. We describe our experimental results in Section 4, and discuss related work in Section 5.

• H. Wu, Q. Luo, and L.M. Ni are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.
E-mail: {whjnn, luo, ni}@cse.ust.hk.

• P. Zheng is with Microsoft, One Microsoft Way, Redmond, WA 98052.
E-mail: zhengpei@gmail.com.

Manuscript received 3 Apr. 2005; revised 27 Oct. 2005; accepted 16 Dec. 2005; published online 27 Dec. 2006.

Recommended for acceptance by C. Raghavendra.

For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0222-0405.

Fig. 1. A target WSN.

Finally, we draw conclusions and outline future research directions in Section 6.

2 OVERVIEW OF VMNET

In this section, we first present the system architecture of VMNet in comparison with a target WSN and then describe the components of a VM briefly.

2.1 System Architecture

To illustrate VMNet, we first introduce a target WSN as shown in Fig. 1. This WSN consists of a server and a number of wireless sensor nodes. The server posts data acquisitional queries to the sensor network and receives query results from the network via the sink. Similar to a target WSN, a VMN consists of a number of VMs, including a virtual sink *VM0* connected to the server, as shown in Fig. 2. The server programs are the same as those in the target WSN except for the connection between the server and the sink (e.g., the RS232 connection), which is emulated in VMNet.

In each VM, the mote hardware is emulated as virtual hardware. The binary code that runs on real motes is run on top of the virtual hardware directly. The EM (Emulation Manager) in each VM controls the VM operations and logs the running status. These running status statistics enable debugging and performance evaluation.

With this emulation architecture, a VMN works in the same way as its target WSN. Users submit queries at the server, and the queries are parsed and disseminated through the server to the virtual motes. The application code is executed at the virtual motes, and the sensory data are transmitted from the virtual motes through the virtual sink to the server.

2.2 Components in a VM

As shown in Fig. 3, a VM contains both virtual hardware and real software components. The real software component is

Fig. 3. A virtual mote.

the binary code that is compiled for the real motes, which handles hardware interrupts, MAC (medium access control) layer communication, and query processing. The virtual hardware components include a Virtual CPU (VCPU), virtual flash memory, a virtual clock, a virtual sensor board, and a Virtual Radio Frequency Module (VRFM). The virtual RAM is included in the VCPU and is omitted in the figure. If the VM is a virtual sink node, it has a virtual UART to emulate the communication interface between the sink and a server. Finally, all virtual hardware components are contained in the virtual main board, which provides a number of virtual sockets to connect the virtual hardware components.

With these components, a VM performs tasks of a real mote. It acquires sensory data, receives and transmits bits via radio channels, executes CPU instructions, and handles interrupts.

3 DESIGN AND IMPLEMENTATION

In this section, we discuss several key issues in the design and implementation of VMNet, including the hardware emulation, the radio channel effects, the sensor node running status, and the statistics logging and performance reporting mechanisms.

3.1 Hardware Emulation in a VM

To emulate the hardware components, we have designed and implemented extensible libraries for the VCPU, the VRFM, the virtual clock, the virtual sensor board, and the virtual UART.

For realistic performance evaluation in VMNet, we currently adopt the hardware configuration of the widely used Crossbow [6] MICA2 motes. This configuration includes the Atmel [2] Atmega128 CPU and the Chipcon [5] CC1000 radio frequency circuit in the RFM. For this configuration, we modified the CPU emulator in Atemu 4.0 [1] to be the VCPU in a VM and implemented our own virtual clock, virtual sensor board, VRFM, and virtual UART.

The virtual clock emulates an oscillator that outputs clock cycles. Different from a real oscillator, it does not output the logic "0"s and "1"s. Instead, it outputs the number of cycles to the virtual hardware components when it ticks (i.e., it increments its clock cycle counter). The output of the virtual clock ensures that the virtual components work at the same timing as their counterparts do in a real mote.

Fig. 2. VMNet architecture.

The virtual sensor board is another core component of a VM. As the major operations of a real sensor board are sensing (sampling) the external environment and Analog/Digital Conversion (ADC), we simulate these two operations, sampling and converting, in the virtual sensor board. For each unit of sensory data, the virtual sensor board reads from files during the sampling period and waits until the converting period ends. After these two periods have passed, the virtual sensor board sends the sampled data to the VCPU via the virtual socket. Because the lengths of these time periods are obtained from real sensor board operations, the virtual sensor board works at the same pace as its target real sensor board.

The VRFM performs the coding, decoding (e.g., Manchester and Non-Return-to-Zero), radio signal transmitting, and receiving functions as the real RFM does. When a VM has data to send, its VCPU sends the bits to its VRFM. The VRFM encodes the bits into virtual radio signals and sends the virtual signals to the VRFBs of other VMs via the virtual radio channel. In a receiving VM, the VRFM decodes the virtual signals into bits and sends these bits to the VCPU. The effects of signal power loss, collision, and errors in signal transmission are emulated.

The virtual UART is used as the interface between the virtual sink and the server. It is implemented on top of TCP, and has several levels of communication granularity, including one bit, one byte, two bytes, and four bytes, for the emulation of various PC-sink interfaces in practice.

Finally, because its system architecture is highly modularized, VMNet can be changed to emulate other WSN hardware configurations (e.g., Crossbow MICA2 motes) easily. For the change to a new hardware configuration, the developers need to provide the emulation modules of the new hardware and the virtual sockets to connect these new virtual hardware components to the virtual main board. After the new emulation modules are plugged into VMNet and are connected to the virtual main board via their corresponding virtual sockets, VMNet is ready to emulate the new hardware configuration.

3.2 Radio Channel Effects

There have been a number of models to predict radio signal propagation and estimate the ambient noise and path loss in various environments [3], [14], [25], [26]. To simulate the radio channel effects, we have implemented several path loss and noise models [3], [12], [26], for VMNet. For a target environment, users can choose a path loss model and a noise model that we implemented or load their own models into VMNet.

One path loss model in VMNet is Bullington's general signal propagation model [3], shown in (1). We implemented it in VMNet because it considers the effects of antenna height and our experience with MICA2 motes shows that the antenna height affects the received power considerably.

$$\frac{P_R}{P_T} = G_R G_T \frac{(h_t^2 + h_0^2)(h_r^2 + h_0^2)}{d^4}. \quad (1)$$

A VM uses this model to calculate the received power P_R , given the transmitted power P_T , the receiver antenna gain G_R , the transmitter antenna gain G_T , the minimal

effective height h_0 of antennas, the actual antenna heights of the transmitter and receiver nodes, h_t and h_r , and the distance d between the antennas of the transmitter and the receiver. Among these variables, h_0 has different experimental values for different radio frequencies and radio channels [3]. The values of other variables can be obtained from the configuration of the target sensor mote.

Another model that we implemented in VMNet is Seidel and Rapport's path loss model [26] for 914 MHz radio in buildings. It is shown in (2). This model is applicable to MICA2 motes, since MICA2 motes communicate at 915 MHz in our environment. In (2), $PL(d)$ is the path loss in decibels from a transmitter to the receiver that is d meters away, d_0 the unit distance with a default value of one meter, n the environment exponent, and X_σ the zero-mean log-normally distributed random variable with a standard deviation σ . Both n and σ are experiential values [26].

$$PL(d) = PL(d_0) + 10n \log_{10} \left(\frac{d}{d_0} \right) + X_\sigma. \quad (2)$$

Equations (1) and (2) model signal fading, but not noise. To simulate noise, we adopt a Gaussian model as the default noise model in VMNet to compute the BER (Bit Error Rate). Equation (3) models the BER of a noncoherent FSK receiver [12], where SNR is the Signal-to-Noise Ratio. Lacking an accurate device to measure the strength of the noise, which is weak and unstable, we use an experiential SNR value [12] to compute the BER.

$$BER = 0.5 \exp(-SNR/2). \quad (3)$$

Note that, at one point in time, VMNet computes the *received powers* of both signals "1" and "0" and regards the one that has a larger *power* as the received signal. Therefore, the shadowing phenomenon [33], which is common in real-world wireless communications, is emulated. When the *power* of the received signal is lower than a predefined threshold, a VM receives no virtual radio signal, which emulates the radio circuit sensitivity [5].

3.3 Node Running States

With the hardware and the radio channel emulated, VMNet can run WSN programs for debugging and testing. However, in order to enable performance evaluation, VMNet needs to obtain the running status of a VMN.

VMNet uses a set of running states to model the dynamics of WSNs. A *running state* or a phase is a commonly used term to describe a set of runtime behaviors at a specific time in WSNs [4], [19]. Specifically, VMNet uses the operations of four major hardware components (the CPU, the flash memory, the sensor board, and the RFM) in a mote to define a running state. As the UART is used only in a sink node, which usually has external power, it is excluded from the consideration on running states. LEDs are omitted because they are optional components. The RAM and the oscillator are also omitted since their operations are included in the CPU operations. Finally, the set of running states can be adjusted when there are major components added or removed.

First, we define the operations of the components in a sensor mote. The CPU of a sensor mote has three

TABLE 1
Running States of a VM

CPU	Flash Memory	Sensor Board	Radio Frequency Module
<i>I</i>	<i>I</i>	<i>A/I</i>	<i>I/R</i>
<i>C/I</i>	<i>I</i>	<i>I</i>	<i>T</i>
<i>C</i>	<i>I/R/W</i>	<i>A/I</i>	<i>I/R/C</i>
<i>H</i>	<i>P</i>	<i>P</i>	<i>P</i>

operations: *Compute* (denoted as *C*), *Idle* (*I*), and *Hibernate* (*H*). The term *Hibernate* is used instead of *Sleep* for an operation is to distinguish from the CPU instruction *sleep*. A sensor board also has three operations: *Acquire* (*A*), *Idle* (*I*), and *Power off* (*P*). The flash memory has four operations: *Read* (*R*), *Write* (*W*), *Idle* (*I*), and *Power off* (*P*). Finally, the RFM of a sensor mote has five operations: *Transmit* (*T*), *Receive* (*R*), *Control* (*C*), *Idle* (*I*), and *Power off* (*P*). We denote a running state by four letters, each of which represents the operation of the CPU, the flash memory, the sensor board, and the RFM, in order. For instance, the running state “*IIII*” means that the CPU, the flash memory, the RFM, and the sensor board are all in the operation *I*.

Next, we give more details about these operations. First, in the CPU, the operation *C* is to execute nonsleep instructions. The operation *I* is to execute the *sleep* instruction, but the other hardware components keep running and may generate interrupts to the CPU. The operation *H* is that the CPU turns off all other hardware components and halts. Usually, when the program on a node sets the control registers to enable the power-off mode and then executes a *sleep* instruction, the CPU will halt. The CPU will wake up from hibernation after a time interval specified in the program. Second, in the flash memory, the operations *R*, *W*, and *I* denote that the flash memory is being read, being written, and idle, respectively. Third, in the sensor board, the operation *A* acquires sensory data and the operation *I* receives commands or sends results. Fourth, in the RFM, the operation *T* sends a bit to the air, *R* receives a bit from the air, *C* sets the RFM configuration registers, and *I* does nothing but senses radio signal. Finally, the operation *P* of the flash memory, the sensor board, and the RFM powers off the corresponding component.

Given these operations of the four components, not all combinations ($3^4 \times 3^5 = 180$) of them may occur in a target WSN (currently we use MICA2 motes). First, the CPU operation *H* and the *P*s of the flash memory, RFM, and sensor board must either occur in one state or none of them occur. Therefore, we are left with $2^3 \times 2^4 = 48$ possible states plus one state (*HPPP*). Among these 48 states, certain operations of the other components are possible only when the CPU is active (as shown in the third row of Table 1). Moreover, the CPU does not read or write the flash when it is controlling the RFM to transmit (as shown in the second row of Table 1). In addition, the sensor board does not sense when the RFM is sending a radio signal. Hence, the final number of all running states is 25, as shown in Table 1.

3.4 Logging and Reporting

For general battery-powered WSNs, the most frequently used performance metrics are power consumption and

TABLE 2
A Fraction of the Emulation Log in a VM

Running state	Time (In terms of CPU clock cycles)
<i>IIII</i>	1926031985
<i>IIIR</i>	1926032167
<i>CRIR</i>	1926034874

execution time of a task or a query [4], [19]. VMNet logs virtual clock cycles and node running states to support these two performance metrics.

Table 2 shows a fraction of the log in a VM, consisting of three running states and their start times (in CPU clock cycles). At the clock cycle 1926031985, the VM enters the state *IIII*. At the clock cycle 1926032167, the VRFM receives some virtual radio signals while the VCPU is still *Idle*. At the clock cycle 1926034874, the VCPU starts to process the message and the VRFM is still receiving radio signals. From the log, we can also learn how the running states are transited from one to another.

Given the running states and the timing information, VMNet calculates the response time t , one of the performance metrics, in (4), where M is the total number of occurrences of the running states and vt_m is the total virtual execution time of the running state m .

$$t = \sum_{m=1}^M vt_m. \quad (4)$$

For the other performance metric, power consumption, VMNet calculates it using (5).

$$W = \sum_{m=1}^M U^* I_m^* vt_m. \quad (5)$$

In (5), W is the energy consumption in Joules, I_m the electric current in running state m , and U the voltage. In Section 4, we give more detail about I_m .

4 EVALUATION

For evaluation, we first measured electric currents in small-scale target WSNs using electric equipment. We then compared the performance of TinyOS [7], [29], applications evaluated in TOSSIM, PowerTOSSIM, VMNet, and real WSNs to study VMNet’s accuracy. We skipped EmStar [9] in the comparison because it does not simulate the MAC behavior and interrupts and, thus, is unable to evaluate the performance of TinyOS applications. Next, to show the usefulness of VMNet, we investigated the breakdown of power consumption by running states and compared two data delivery algorithms as well as two routing protocols in sensor network query processing. Finally, we evaluated the scalability of VMNet in comparison with TOSSIM and PowerTOSSIM using networks of up to 500 simulated or emulated nodes.

4.1 Experimental Setup

Table 3 lists the equipment used in the experiments. The software used in these experiments was TinyOS [7], [29],

TABLE 3
Equipment Used

Name	Model	Quantity
Oscilloscope	HP 4155A	1
PC	Pentium® 3.2GHz, 1GB RAM, 40GB hard disk, OS: Red Hat Linux 9.0	1
Sink mote	MIB510 interface board MPR 410 processor board	1
Sensor mote	MPR 410 processor board MTS300 sensor board	19

with TinyDB [28] (V1.1.0), TOSSIM, and PowerTOSSIM. We ran VMNet, TOSSIM, or PowerTOSSIM each on a PC. We used the oscilloscope in an electronic lab to measure the electric current and the voltage of a sensor mote. The connection for measuring the electric current of a mote is shown in Fig. 4.

With the electric current and voltage of a sensor mote measured, its power consumption can be computed using (6), which is similar to (5) in Section 3.4 except the intervals are between two measurements. U_i and I_i are the voltage and the electric current in a measurement, N the total number of measurements, and t_i the interval between two measurements.

$$W = \sum_{i=0}^N U_i I_i t_i. \quad (6)$$

To choose a suitable path loss model for simulating the radio channel in the electronic lab, we measured the path loss in the lab using the MICA2 sensor motes and compared with those computed from the two path loss models that we implemented in VMNet.

To measure the path loss in the lab, we ran two TinyOS application programs, each on one sensor mote. One mote kept sending packets and the other kept receiving. Since the transmitted power P_T was 0.1 mW [5] under the default configuration in TinyOS, we only measured the received power P_R via the Radio Signal Strength Indicator (RSSI) [5] of the receiving mote. Then, we computed the path loss as P_R/P_T .

To estimate the path loss in the lab using the models, we set the parameter values for both models. The parameter

Fig. 5. Path loss versus distance.

values we used in Bullington's model [3] were as follows: The antenna height of a MICA2 sensor mote was 0.12 meters, the antenna gains G_R and G_T were both 1, and the minimal effective height h_0 of an antenna was 0 for the 915 MHz radio in a building [3], [5]. In Seidel and Rapport's model [26], we set the parameters n as 4.01 and σ as 4.3 according to our lab environment.

The path loss with respect to distance in our environment is shown in Fig. 5. As the results from Bullington's model were closer to the measurements than those from Seidel and Rapport's were, we used Bullington's model in VMNet for all experiments in our lab. Note that, even though Bullington's model was chosen for simulating the environment of our lab, there are environments in which Seidel's model is more suitable [16], [35].

Throughout the experiments, the transmitted power P_T of the RFM on a node was set to 0.1 mW [5] by default and did not change dynamically. With this configuration, we measured the average transmission range to be seven meters in the real WSNs in our lab. Note that VMNet can emulate and log the operations when the P_T of a VM is dynamically configured. However, the oscilloscope may not be able to capture the change because its minimum sampling period, four milliseconds, is too large to capture the microsecond-level operations such as setting configuration registers of the RFM.

Since VMNet emulates radio transmission at the radio signal level, for a fairer comparison, we configured TOSSIM and PowerTOSSIM to simulate at the bit level even though they can simulate either at the bit level or at the packet level. Other configurations such as BER and transmission range in TOSSIM and PowerTOSSIM were the same as in VMNet.

To evaluate the performance of query processing application programs, we used four queries, Queries 1-4, to run on TinyDB in the experiments. The only difference between Queries 1-3 is the length of the sampling period, which is useful to reveal the performance effects of various factors such as running state transitions and data volume. Query 4 was used to evaluate data aggregation schemes since it involves the computation of the average function.

Fig. 4. Connection to the oscilloscope.

Fig. 6. Electric currents of a sensor mote running in “IIIR” in different WSNs.

- Query 1: Sample the temperature every 2,048 milliseconds.
- Query 2: Sample the temperature every 10 seconds.
- Query 3: Sample the temperature every minute.
- Query 4: Return the average sound level every 2,048 milliseconds.

4.2 Electric Currents in Target WSNs

To measure the electric current in a running state, we designed a TinyOS application program that made the components of a sensor mote work continuously in one operation. For instance, to measure the electric current in the running state *IIIR*, the program made the CPU, the sensor board, and the flash memory all *idle*, and the RFM keep *receiving*. For this state, the program disabled the interrupts from the RFM so that the CPU and the flash memory remained *idle* even when the RFM was receiving data from another node. We ran this program on a MICA2 sensor mote while other motes all kept sending radio signals and used the oscilloscope to measure the electric current of this mote. We found that the electric current in a running state of a sensor mote was nearly constant, regardless of the configuration of the WSN, as shown in Fig. 6.

Table 4 lists the measured electric currents of various running states of a sensor mote except the flash-writing states since our experiments did not involve writing the flash memory. The battery voltage was 3 Volts. The measurement results were consistent with both the data sheet of the MICA2 sensor mote [6] and that of Chipcon CC1000 radio circuit [5]. Note that the data sheets only specify electric currents of individual hardware components. For example, under the configuration of TinyOS, the electric current of the sensor board is 5 mA and that of the CC1000 radio circuit is 8 mA when it is receiving radio signals. Therefore, it is necessary to measure the electric currents of a whole sensor mote in various running states instead of individual parts.

It can be seen from Table 4 that the CPU operations *Idle* and *Compute* made no difference on power consumption in our experiments. Two reasons were involved. First, the power saving bit (*SE*) in the register *MCUCR* of the Atmega CPU [2] was not set before the sleep instruction was executed in the *Idle* operation. Second, the analog comparator in the Atmega CPU was enabled in the *Idle* operation. These settings were, by default, in TinyOS applications. In our experiment, we also found that a sensor board consumed almost the same amount of power for

TABLE 4
Electric Currents of Running States

Running state	Electric current (mA)	Variance
<i>CIAI / IIAI / CIII / IIIR</i>	17.2	0.005
<i>CRAI / CRII</i>	23.9	7.514×10^{-9}
<i>CIAR / IIR / CIIR / IIIR</i>	25.3	1.626×10^{-11}
<i>CRAR / CRIR</i>	31.0	0.054
<i>CIIC / CIAC</i>	21.0	0.007
<i>CRIC / CRAC</i>	26.7	3.074×10^{-9}
<i>CIIT / IIIT</i>	28.5	1.895×10^{-9}
<i>HPPP</i>	0.016	3.043×10^{-5}

Acquire and *Idle* and we suspect the reason was still to do with the hardware configuration. Note that, even though the electric current of a running state in Table 4 may change with the hardware settings, this change only affects the parameter values in the power consumption computation and has no effect on the emulation or performance evaluation mechanism in VMNet.

4.3 Validation of VMNet Readings

To validate VMNet, we compare the performance evaluation of one WSN application using VMNet, TOSSIM, PowerTOSSIM, and the oscilloscope. The performance in PowerTOSSIM, VMNet, and target WSNs included both response time and power consumption and that in TOSSIM included response time only as TOSSIM did not report node power consumption.

We deployed four WSNs with 2, 5, 10, and 20 nodes each. In the WSNs, every node was placed within 2 meters from the sink; as a result, the WSNs were all one-hop networks. The configurations of the VMNs in VMNet, TOSSIM, and PowerTOSSIM were the same as those of the target WSNs we measured. The application that was used sampled the temperature every 2,048 milliseconds. Since the oscilloscope could only capture the electric currents for 10 seconds with the minimal granularity of 4 milliseconds for one capture, we measured the time and power consumption for a node to return two query results and repeated this measurement for 30 times. To obtain stable performance results, we set a two-second warm-up period for each 10-second capture period and discarded abnormal results.

The average response time and power consumption per node are shown in Table 5. As shown in Table 5, when the number of nodes in the network increases from 2 to 20, the query response time in the real WSNs increases from 3.8 seconds to 4.7 seconds. This increase indicates that the size of network affects the query response time. One possible cause is signal collision. With a higher density of nodes in the given area, more nodes are within each other's communication range and the collision of radio signals is more severe, especially around the sink node. Consequently, when one node is sending a query result to the sink, signals from other nodes to the sink will collide with this node's. Subsequently, these node have to wait, which results in a longer response time.

TABLE 5
Power Consumption Measured from VMNet, TOSSIM,
PowerTOSSIM, and Target WSNs

Metrics Network	Node Power Consumption (Joules)		
# Nodes	WSN	VMNet	PowerTOSSIM
2	0.229	0.214	0.174
5	0.268	0.227	0.173
10	0.303	0.262	0.173
20	0.335	0.289	0.173

In comparison with the real WSN response time, the query response time estimated in VMNet increases from 4.0 seconds to 4.8 seconds, whereas that estimated in TOSSIM or PowerTOSSIM remains at around 3.7 seconds, when the number of node increases from 2 to 20. Similarly, both the real WSNs and VMNet report increasing power consumption when the number of nodes increases, but PowerTOSSIM does not. We also examined the other sensor notes in these deployed networks. The results were similar to those in Table 5 and Table 6.

Even though the VMNet estimation on time and power consumption follows the WSN measurement, there are noticeable differences between the estimation and the measurement, especially on power consumption. There are a few possible reasons for these differences. One possible reason is that VMNet does not consider the effects of the capacitors or inductances in a sensor mote, which causes the electric current to change more slowly in real WSNs than the sudden transitions in VMNet. Another possible reason is that the oscilloscope cannot capture the changes of the electric current in a time period that is smaller than 4 milliseconds. Thus, the real measurement results using the oscilloscope are inaccurate themselves. In addition, the radio effect models also differ from the effects on real-world radio channels.

4.4 Running States of VMs

In addition to the performance evaluation in comparison with real WSNs, VMNet is useful for studying detailed information of the running states of each VM. From the analysis of the statistics on the running status, we can identify the major running states that cost the most time and power. This analysis helps to make a faithful abstraction of the dynamics in individual nodes as well as in the entire network.

TABLE 6
Response Time Measured from VMNet, TOSSIM,
PowerTOSSIM, and Target WSNs

Metrics Network	Node Response Time (Milliseconds)		
# Nodes	WSN	VMNet	TOSSIM / PowerTOSSIM
2	3781.8	4043.3	3741.9
5	4214.2	4186.2	3729.6
10	4426.5	4585.2	3725.1
20	4737.3	4825.1	3727.6

Fig. 7. Deployment of three VMNs.

We set up three VMNs with 10, 30, and 50 nodes each to study the effect of network size on the running states of a single node. The setup is shown in Fig. 7. All nodes deployed below the dashed diagonal line {1} formed the 10-node VMN. Similarly, all nodes below the dashed diagonal lines {2} and {3} formed the 30-node and 50-node VMNs, respectively. The node at (0,0) was the sink node for all three VMNs. In this experiment, we examined the running status of the node at the location (3,3) (node #a). We compared the power consumption of different running states in node #a during a fixed time interval in the three VMNs. The deployment isolated the effect of network size on running states as the distance between this node and the sink node was fixed and the density of the three VMNs was about the same.

We ran *Query 1* and logged the running states for virtual time 20 seconds in the three VMNs. Fig. 8 shows the breakdown of the power consumed by node #a in 10, 30, and 50-node VMNs, respectively. The legends “***R” and “***T” denote the running states in which the VRFM was receiving or sending signals, respectively.

Across the three VMNs, most of the power (60 percent to 70 percent) was consumed in the *IIII* state. The second largest power consumer (20 percent to 30 percent) was the radio receiving states and the third the radio transmission

Fig. 8. Breakdown of node power consumption.

Fig. 9. The deployment of the VMN.

states. These results confirm that, even though the sampling period of a query was as short as two seconds, 1) most of the energy was still consumed when all components of a node were idle and 2) radio communication was the major power consumer in comparison with CPU computation and sensor sensing.

Among the three VMNs, the percentage of the power consumed in the radio receiving states ($^{***}R$) in the 30-node VMN was larger than that in the 10-node VMN and that in the 50-node VMN. A possible cause for this phenomenon is the interplay between signal overhearing [10] and waiting time for the transmission channel. When the number of nodes in the network grows from 10 to 30, the effect of overhearing increases more significantly than the waiting time for the transmission channel and, thus, the power consumption in $^{***}R$ increases. In comparison, when the number of nodes increases from 30 to 50, waiting for the transmission channel dominates overhearing since many nodes are acquiring sensory data. As a result, most nodes must be in IIII for a long time to wait for the radio channel to become clear. This long waiting time causes the IIII state to consume more power when the number of nodes increases from 30 to 50.

4.5 Effects of Data Aggregation Schemes

Studying the power consumption of running states is one example of the usefulness of VMNet and another example is to study the performance impact of data aggregation in delivery schemes.

Researchers have noted the importance and benefits of database query processing, especially data aggregation, in WSN applications [19], [34]. As a case study, we show how VMNet can be used to compare two different algorithms of a query processor under the same configuration.

We compared two common algorithms of aggregation query processing, one is *direct-delivery* and the other *partial-aggregation* [34]. The algorithm *direct-delivery* simply sends the sensory data from each sensor node to the sink node via multihop routing. The sink node then forwards the data to

TABLE 7
Node Power Consumption of Two Algorithms (Joules)

VM ID	Running for 300 seconds		Running for 60 epochs	
	Direct-delivery	Partial-aggregation	Direct-delivery	Partial-aggregation
1	15.254	15.204	18.284	16.416
2	14.642	14.579	17.536	15.714
3	15.405	15.074	18.472	16.266
4	14.422	14.230	17.357	15.353
5	14.547	14.240	17.380	15.378
6	15.425	15.254	18.488	16.441
7	15.252	15.097	18.285	16.273
8	15.268	15.058	18.299	16.399
9	15.607	15.227	18.767	16.431

the server, which in turn computes the aggregation (e.g., `count()`, `sum()`, and `average()`) of the reported data in every epoch. In this algorithm, the internal nodes (nonleaf nodes) do nothing but simply forward data toward the sink. In comparison, the algorithm *partial-aggregation* requires an internal node to calculate the aggregation (typically, `count()` and `sum()`) of the data that the node receives from its child nodes. The internal node then forwards the partially aggregated data, instead of the original data from its child nodes, toward the sink. As *partial-aggregation* transmits less data than *direct-delivery*, it is expected to be more power efficient. We have implemented both algorithms in TinyDB and used VMNet to analyze their performance quantitatively.

In this experiment, we used *Query 4* to query the average sound level sensed by a 10-node VMN every two seconds. The deployment and the initial routing tree of the 10-node VMN is shown in Fig. 9. There are several multihop routes in the VMN, which are common in real WSNs and are necessary for showing the effects of the data aggregation algorithms. To make the experiments controllable, we need a fixed topology for both algorithms. However, the topology of a WSN may change if there are packet losses or some nodes transit into the state *HPPP*. Note that TinyOS does not make nodes transit to *HPPP*, but TinyDB does when the sample interval of the query being processed is longer than 4 seconds [28]. To fix the network topology, we set the sampling period of the query to two seconds so that VMs did not enter *HPPP* in such a short period. In addition, we set the BER to be 0 in VMNet to ensure that there was no packet loss during query processing.

After the multihop topology was set up and became stable, VMNet started to log the running status. For each algorithm, VMNet logged 300 seconds virtual time. Because a sink mote is usually on external power, we do not study its power consumption. The power consumption of other VMs for these 300 seconds is listed in Table 7. On average, *partial-aggregation* consumed 0.207 Joules less power than *direct-delivery* in this deployment. In other words, given the same battery, *partial-aggregation* could run four seconds longer than *direct-delivery* for every 300 seconds in a WSN that has the same configuration as this 10-node VMN.

If we consider the power consumption in acquiring a fixed number of sensory data results, *partial-aggregation* is

Fig. 10. The 3 * 3 grid deployment.

even better than direct-delivery. We logged the performance of these two algorithms obtaining 60 epochs of query results. The partial-aggregation algorithm took 323.7 seconds, whereas the direct-delivery algorithm took 359.3 seconds. On average, a VM with partial-aggregation saved 2.022 Joules of power than with direct-delivery for a total of 60 epochs. These results quantify the time efficiency as well as the power efficiency of in-network aggregation, which has been studied through simulation [18], [34], but not emulation.

4.6 Performance Comparison of Routing Protocols

In addition to data aggregation schemes, we tested the performance of two routing protocols in VMNet. One routing protocol is from TinyDB and the other TinyAODV [27]. The routing protocol in TinyDB sets up a routing tree toward the destination (sink) after initialization. In contrast, TinyAODV does not construct a routing tree. Instead, a source node broadcasts its request for a route to the destination when it has data to send. After the route is set up, the source node sends its data, which will be forwarded to the destination along this route [22], [27].

We set up two VMNs (VMN_1 and VMN_2) of identical configuration. We replaced the protocol in the original TinyDB with TinyAODV and then built a new version of TinyDB, denoted as $TinyDB_{aodv}$. We loaded the original TinyDB binary code to VMN_1 and that of $TinyDB_{aodv}$ to VMN_2 . The network topology of the two VMNs was a 3 * 3 network with the virtual sink node in the center. The setup is shown in Fig. 10. As the transmission range was seven meters, there were two hops in the network topology.

We ran each of *Query 1* and *Query 2* in the two VMNs for 120 seconds of virtual time and logged the running states in VMNet. The average power consumption of the two VMNs is shown in Table 8. It shows that the routing protocol in TinyDB consumes less power than TinyAODV for both queries and this difference is more significant for *Query 2*. One reason is that the routing protocol in the original TinyDB sets up a routing tree, which saves the routing overhead and, therefore, saves more energy. Additionally,

TABLE 8
Average Power Consumption (Joules) of a Node
in the Two VMNs

Query ID	VMN_1 (original TinyDB)	VMN_2 ($TinyDB_{aodv}$)
<i>Query 1</i>	8.774	8.791
<i>Query 2</i>	3.785	5.410

the sensor nodes do not transit to *HPPP* in *Query 1*, but they do in *Query 2*.

In addition to the difference between protocols, Table 8 shows that each VMN consumes much more power in running *Query 1* than running *Query 2*. There are two reasons for this difference. First, a node running *Query 1* produces more sensory results (one result every two seconds) than it does in running *Query 2* (one result every 10 seconds). The more results produced, the more energy is consumed to transmit the results. Second, a node running *Query 2* runs into *HPPP* for several seconds in each sampling interval (10 seconds), which saves power, whereas a node running *Query 1* does not.

4.7 Scalability of VMNet

Given the high fidelity of VMNet, the emulation speed, or the scalability of VMNet is a concern to address. Therefore, we studied the scalability of VMNet in comparison with that of TOSSIM and PowerTOSSIM, which is known to be highly scalable, on a single PC. We chose TinyDB, as opposed to other simple applications, for the scalability test because we believe that query processors for sensory data acquisition are one of the major WSN applications.

In our experiments, we fixed the density of the emulated network to be one node per nine square meters and varied the total number of emulated nodes from 10 to 500. We ran *Query 1* as well as *Query 3* and recorded the emulation time (the running time) of emulating 10 virtual seconds in VMNet. We also measured the simulation time of TOSSIM and PowerTOSSIM under the same configuration as that of VMNet.

Figs. 11a and 11b show the time needed by VMNet, TOSSIM, and PowerTOSSIM to complete the emulation or simulation of the 10 virtual seconds of query processing. The speed of VMNet was about 1/10 of that of TOSSIM on average because VMNet emulates sensor nodes at the CPU clock cycle level and logs detailed running state statistics. This is the trade-off between speed and fidelity.

Figs. 11a and 11b also show that the speed of running *Query 1* is slower than running *Query 3*. The reason is that *Query 1* requires more communication and computation than *Query 3* due to the short sample interval of *Query 1*. Furthermore, when a sensor network runs *Query 3*, nodes sometimes enter the state *HPPP*, in which the emulation or simulation runs much faster than in other states as the virtual hardware components do nothing in *HPPP*.

To find the bottleneck that hindered the emulation speed of VMNet, we investigated the usage of system resources. Since it was single-PC simulation or emulation, we exclude network resources from consideration. Table 9 gives the usage statistics of the system resources with which VMNet

(a)

(b)

Fig. 11. Scalability of VMNet, TOSSIM, and PowerTOSSIM.

ran *Query 1* on the 500-node VMN. It can be seen that memory was not a limiting factor and neither was disk I/O. In contrast, the CPU usage was intensive and was the bottleneck for running VMNet on a single PC.

In order to address the CPU bottleneck on single-PC emulation, we plan to extend VMNet to run on multiple networked PCs.

5 RELATED WORK

There have been a number of simulators for network systems. ns-2 [21] is a commonly used discrete simulator that models a network as multiple layers. J-sim [13] is also a widely used simulator that employs layered network models. Unlike ns-2, J-sim has a Java interface. However, these simulators with layered network models are unsuitable for performance evaluation of WSNs because the performance of WSNs is affected by the hardware, the OS, the networking protocols, the application code, as well as cross-layer optimization techniques.

There has also been work on detailed CPU emulation. For instance, Embra [32] was developed to emulate the MIPS R3000/R4000 processor. Studies on Embra demonstrate that this kind of CPU emulation is crucial for high fidelity emulation, safe abstraction, and modeling. Another example is the instruction-level power analysis in Myrmigki by Stanley-Marbell and Hsiao, which shows that an instruction-level power consumption model can be accurate [20]. Both systems show that the level of detail in modeling or emulating the CPU has a significant impact on performance analysis. Along this line, we used a modified VCPU module for Crossbow MICA2 motes in VMNet. This VCPU module is modified from Atemu [1], [23], a detailed CPU emulator for Atmel CPUs [2]. It emulates the registers and executes the CPU instructions. We modified the Atemu CPU emulator to emulate the CPU halt and the UART communication and to

TABLE 9
Resource Usage

Item	Usage
CPU	100%
Memory	210MB
Disk I/O (for logging)	20KB/second (on average)

log the operations of the CPU. Moreover, we added our UART and radio channel emulation modules as well as running status logging for performance evaluation.

TOSSIM [15] and EmStar [11] are two WSN simulation tools that are most related to our work. TOSSIM is a TinyOS simulator with a network model, whereas EmStar is a software environment for developing and deploying micro-server grade sensor nodes. Both of them provide researchers powerful tools to test and to debug the application programs. However, neither of them performs CPU emulation nor focuses on realistic performance evaluation as VMNet does. The TinyOS and application code is compiled for PC in both TOSSIM and EmStar, not compiled for motes directly as in VMNet. Finally, both TOSSIM and EmStar allow trade-offs between fidelity and simulation speed by simulating at different levels (e.g., bit or packet level). We currently trade emulation speed for fidelity in VMNet and are investigating the alternative of utilizing networked PCs to improve the emulation speed.

Adding the feature of power estimation to TOSSIM, PowerTOSSIM [24] estimates power consumption of target nodes through logging the operations both in TOSSIM modules and in the application code. In comparison, VMNet does not instrument the application code. In PowerTOSSIM, the electric current of each type of hardware peripheral, such as the CPU, radio, sensor, EEPROM, and LEDs, is measured separately. Subsequently, PowerTOSSIM uses the measured electric currents as its experiential power model. In contrast, VMNet categorizes the running states of a mote with respect to the operations of its major components and measures the electric currents of the running states for parameter values.

Most recently, Motelab [31], a WSN testbed consisting of about 30 MICA2 nodes [6], has been developed and deployed in the Maxwell Dworkin Laboratory at Harvard. It allows public access for developing and testing TinyOS applications. Users can program the nodes, run TinyOS application programs, and get the logs of the WSN packets. While this testbed is valuable for small-scale experiments on real WSNs in a lab environment, simulators and emulators, including our VMNet, are still irreplaceable for larger experiments in various settings on various performance aspects.

Additionally, VMNet can be seen as an outgrowth of our previous work on a distributed wired and wireless network emulation framework EMPOWER [36]. Our previous work on wireless network evaluation [37] taught us some lessons in evaluating wireless networks.

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented VMNet, a WSN emulator for realistic performance evaluation. It emulates and logs the operations of each component in a sensor mote at the level of CPU clock cycles. It also emulates the radio transmission channel with signal loss and noise. We categorize the running states of sensor motes and study their power consumption characteristics in the real-world. Subsequently, we use these real-world statistics for the performance estimation of VMNs.

We have conducted extensive experiments to evaluate VMNet in comparison with real WSNs, TOSSIM and PowerTOSSIM. We find that VMNet estimates the response time and power consumption of VMNs reasonably well. Moreover, VMNet enables quantitative studies on VMNs such as the power consumption of the running states and that of different data aggregation schemes. Finally, we have evaluated the emulation speed of VMNet in comparison with the simulation speed of TOSSIM and PowerTOSSIM and have investigated the performance bottleneck of single-PC VMNet.

We conclude that detailed emulation in VMNet is useful for researchers in many areas such as architecture, OS, and sensor network query processing. It also provides insights for parameter value setting and validation of WSN simulation and modeling. The documentation and the code base of VMNet are online at <http://www.cs.ust.hk/vmnet> for public use.

The immediate focus of our ongoing work is to improve the emulation speed for large VMNs. We are investigating multiple abstraction levels to trade fidelity for speed and extending VMNet to run on networked PCs.

ACKNOWLEDGMENTS

This work was supported by the Hong Kong Research Grants Council (RGC) through HKUST6263/04E. This research was also supported in part by Hong Kong RGC Grants HKUST6161/03E, HKUST6183/05E, and AoE/E-01/99; and by the National Basic Research Program of China (973 Program) under Grant No. 2006CB303000.

REFERENCES

- [1] Atemu, <http://www.cshcn.umd.edu/research/atemu/>, 2003.
- [2] Atmel, <http://www.atmel.com/products/avr/>, 2005.
- [3] K. Bullington, "Radio Propagation Fundamentals," *The Bell System Technical J.*, vol. 36, no. 3, pp. 593-625, 1957.
- [4] C.-F. Chiasserini and M. Garetto, "Modeling the Performance of Wireless Sensor Networks," *Proc. IEEE INFOCOM*, 2004.
- [5] Chipcon, <http://www.chipcon.com/>, 2005.
- [6] Crossbow, <http://www.xbow.com/>, 2005.
- [7] D.E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo, "A Network-Centric Approach to Embedded Software for Tiny Devices," *Proc. ACM Conf. Embedded Systems Software (EMSOFT)*, 2001.
- [8] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler, "Design of a Wireless Sensor Network Platform for Detecting Rare, Random, and Ephemeral Events," *Proc. Conf. Information Processing in Sensor Networks (IPSN)*, 2005.
- [9] EmStar, <http://cvs.cens.ucla.edu/emstar/>, 2005.
- [10] L.M. Feeney and M. Nilsson, "Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment," *Proc. IEEE INFOCOM*, 2001.
- [11] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, "EmStar: A Software Environment for Developing and Deploying Wireless Sensor Networks," *Proc. USENIX Ann. Technical Conf.*, 2004.
- [12] D. Goodman and N. Mandayam, "Power Control for Wireless Data," *IEEE Personal Comm.*, vol. 7, no. 2, pp. 48-54, Apr. 2000.
- [13] J-sim, <http://www.j-sim.org/>, 2005.
- [14] J.-F. LaFortune and M. Lecours, "Measurement and Modeling of Propagation Losses in a Building at 900 MHz," *IEEE Trans. Vehicular Technology*, vol. 39, no. 2, pp. 101-108, May 1990.
- [15] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," *Proc. First Int'l Conf. Embedded Networked Sensor Systems*, 2003.
- [16] D. Lymberopoulos, Q. Lindsey, and A. Savvides, "An Empirical Analysis of Radio Signal Strength Variability in IEEE 802.15.4 Networks Using Monopole Antennas," Technical Report ENA-LAB-050501, Yale Univ., 2005.
- [17] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," *Proc. ACM Int'l Workshop Wireless Sensor Networks and Applications*, 2002.
- [18] S. Madden, M.J. Franklin, J.M. Heller-Stein, and W. Hong, "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks," *Proc. Symp. Operating Systems Design and Implementation (OSDI)*, 2002.
- [19] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "The Design of an Acquisitional Query Processor for Sensor Networks," *Proc. IEEE SIGMOD*, 2003.
- [20] P. Stanley-Marbell and M. Hsiao, "Fast, Flexible, Cycle-Accurate Energy Estimation," *Proc. 2001 Int'l Symp. Low Power Electronics and Design*, 2001.
- [21] ns-2, <http://www.isi.edu/nsnam/ns/>, 2005.
- [22] C.E. Perkins, E.M. Belding-Royer, and S. Das, "Ad Hoc On-Demand Distance Vector (AODV) Routing," *IETF RFC 3561*, 2003.
- [23] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J.S. Baras, "ATEMU: A Fine-Grained Sensor Network Simulator," *Proc. First IEEE Int'l Conf. Sensor and Ad Hoc Comm. Networks*, 2004.
- [24] V. Shnayder, M. Hempstead, B. Chen, G.W. Allen, and M. Welsh, "Simulating the Power Consumption of Large-Scale Sensor Network Applications," *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*, 2004.
- [25] V. Rodoplu and T.H. Meng, "Minimum Energy Mobile Wireless Networks," *IEEE J. Selected Areas in Comm.*, vol. 17, no. 8, pp. 1333-1344, Aug. 1999.
- [26] S.Y. Seidel and T.S. Rapport, "914 MHz Path Loss Prediction Model for Indoor Wireless Communications in Multifloored Buildings," *IEEE Trans. Antennas and Propagation*, vol. 40, no. 2, pp. 207-217, Feb. 1992.
- [27] TinyAODV, <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/tinyos/tinyos-1.x/contrib/hsn/>, 2003.
- [28] TinyDB, <http://telegraph.cs.berkeley.edu/tinydb/>, 2003.
- [29] TinyOS, <http://www.tinyos.net/>, 2003.
- [30] VMNet, <http://www.cs.ust.hk/vmnet>, 2004.
- [31] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A Wireless Sensor Network Testbed," *Proc. IEEE/ACM, IPSN Track on Sensor Platform, Tools and Design Methods for Networked Embedded Systems Conf. (SPOTS)*, 2005.
- [32] E. Witchel and M. Rosenblum, "Embra: Fast and Flexible Machine Simulation," *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, 1996.
- [33] A. Woo, K. Whitehouse, F. Jiang, J. Polastre, and D. Culler, "The Shadowing Phenomenon: Implications of Receiving during a Collision," Technical Report CSD-04-1313, Univ. of California, Berkeley, 2004.
- [34] Y. Yao and J. Gehrke, "Query Processing in Sensor Networks," *Proc. Conf. Innovative Data Systems Research (CIDR)*, 2003.
- [35] J. Zhao and R. Govindan, "Understanding Packet Delivery Performance in Dense Wireless Sensor Networks," *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys)*, 2003.
- [36] P. Zheng and L.M. Ni, "EMPOWER: A Network Emulator for Wireless and Wired Networks," *Proc. IEEE INFOCOM*, 2003.
- [37] P. Zheng and L.M. Ni, "Test and Evaluation of Wide Area Networks Using Emulator Cluster," *Proc. Int'l Comm. Conf. (ICC)*, 2003.

Hejun Wu received the master's degree in computer science from Nankai University in 2003. He is a PhD student of computer science at the Hong Kong University of Science and Technology (HKUST). His research interests are in wireless sensor networks, distributed systems, and embedded systems. He is a student member of the IEEE.

Pei Zheng received the PhD degree in computer science from Michigan State University in 2003. He joined Microsoft in 2005. He was a member of the technical staff in Bell Laboratories, Lucent Technologies. His research interests include distributed systems, network simulation and emulation, and mobile computing. He is a member of the IEEE.

Qiong Luo received the PhD degree in computer science from the University of Wisconsin-Madison. She is an assistant professor of computer science at the Hong Kong University of Science and Technology. Her research interests are in database systems, distributed systems, pervasive computing, and sensor networks.

Lionel M. Ni received the PhD degree in electrical and computer engineering from Purdue University, West Lafayette, Indiana, in 1980. He is a chair professor at the Hong Kong University of Science and Technology and heads the Computer Science Department. He is also director of the HKUST China Ministry of Education/Microsoft Research Asia IT Key Lab. A fellow of the IEEE, Dr. Ni has chaired many professional conferences and has received a number of awards for authoring outstanding papers. He is a coauthor of the book *Interconnection Networks: An Engineering Approach* (with Jose Duato and Sudhakar Yalamanchili, Morgan Kaufmann, 2002) and the book *Smart Phone and Next Generation Mobile Computing* (with Pei Zheng, Morgan Kaufmann, 2006).

► **For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.**