Accurate Emulation of Wireless Sensor Networks

Hejun Wu[+] Qiong Luo[+] Pei Zheng[*] Bingsheng He[+] Lionel M. Ni[+]

[+] Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon
Hong Kong, China
{whjnn, luo, saven, ni}@cs.ust.hk

[*] Department of Computer Science
Arcadia University
450 South Easton Road
Glenside, PA 19038, USA
zheng@arcadia.edu

Abstract. Wireless sensor networks (WSNs) have a wide range of useful, data-centric applications, and major techniques involved in these applications include in-network query processing and query-informed routing. Both techniques require realistic environments and detailed system feedback for development and evaluation. Unfortunately, neither real sensor networks nor existing simulators/emulators are suitable for this requirement. In this design paper, we propose a distributed sensor network emulator, a Virtual Mote Network (VMNet), to meet this requirement. We describe the system architecture, the synchronization of the nodes and the virtual time emulation with a focus on mechanisms that are effective for accurate emulation.

1. Introduction

Wireless sensor networks (WSNs) enable applications to obtain up-to-date information about the physical world. This information is especially valuable for environments in which it is inefficient, difficult or dangerous for people to collect data on site by themselves. However, such environments also make it hard to study techniques for data-centric WSN applications in real sensor networks. Furthermore, major techniques in data-centric WSNs, such as in-network query processing [8][11] and query-informed routing [3], need a realistic development and evaluation environment with system feedback at a suitable level of detail. In this design paper, we propose to develop an accurate sensor network emulator in order to facilitate studies of techniques for data-centric WSN applications.

Traditionally, simulators and emulators are useful tools for networking research in that they simulate or emulate real networking protocols and provide a controllable environment for studies. This usefulness is even greater for WSN applications, because real WSNs are in frequent upgrades and their deployment is tightly embedded

in the physical environment. As evidence, several sensor network simulators and emulators [9] have been developed for large-scale WSN studies.

If we focus on developing and debugging WSN applictions in a realistic environment, existing tools such as TOSSIM [7] and EmStar [4] are excellent choices. However, with the advance of data-centric WSN applications, such as environmental monitoring and assisted living, more requirements for simulation and emulation are posed for developing and evaluating techniques in these applications. For instance, in-network query processing and query-informed routing, two major cross-layer techniques for data-centric WSN applications, require the WSN to return information about sensor node power consumption and response time in order to make decisions for network routing and query processing. Evaluation of alternatives of each technique also requires this information for performance comparison. Unfortunately, current WSN simulators/emulators are insufficient to address this need. Specifically, an accurate emulation of timing and power consumption for node execution and communication is missing in current WSN simulators and emulators.

Aiming at accurate emulation of a WSN for data-centric applications, we propose a WSN emulator called a Virtual Mote Network (VMNet). A VMNet consists of virtual sensor nodes connected through a virtual channel. Each virtual sensor node in turn consists of an emulated CPU as well as emulated hardware peripherals (e.g., sensing units and radio frequency units). The emulated CPU executes software that can run on real sensor nodes and reports execution time at the granularity of the emulated CPU cycle. The emulated hardware peripherals generate interrupts with realistic delays. The virtual channel is emulated through UDP (User Datagram Protocol) on networked PCs with emulated bit errors, delays, and packet collision. Putting all these units together, the timing information of the software under study (e.g., an in-network query processor or a query-informed router) can be accurately emulated and be fed back for the execution and evaluation of the software.

The remainder of this paper is structured as follows. Section 2 introduces the background of our work. Section 3 presents the design of VMNet, including the architecture and components. Section 4 discusses related work briefly and Section 5 concludes.

2. Background

2. 1. Terminology

The following terms are used throughout the paper:

Node and **Mote**: both refer to a sensor node consisting of computation, sensing, and communication units. The two terms are used interchangeably.

Real (Target) vs. **Virtual (Emulated):** A real or target component is one in a real WSN and its counterpart in VMNet is virtual or emulated. For instance, a real CPU is in a real sensor node and a virtual CPU in a virtual mote. Similarly, we refer to the execution time of real software being emulated as the virtual time (not the time of executing the emulation itself).

2.2. Overview of a Target WSN

Fig. 1 shows a typical WSN. The sensor mote in the WSN is MICA2 by crossbow [2]. We choose MICA2 as the target because it is most commonly used.

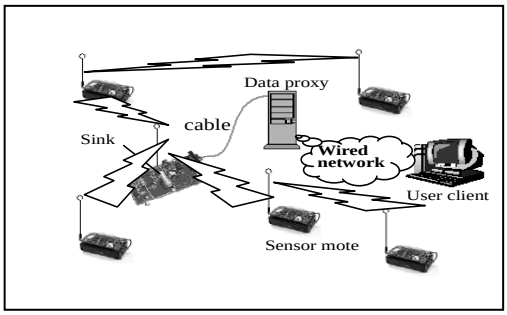

Fig. 1. A Typical WSN

The WSN in Fig. 1 consists of several components, each of which performs different functions. Table 1 lists these components and their composition. Each sensor node in the WSN runs application software (e.g., a query processor) developed using TinyOS [3]. The sink node acts as the root of the WSN and communicates with the data proxy. The data proxy in turn communicates with the user client. Note that the data proxy and the user client can be on the same PC.

Table 1. Components of a WSN

Components	Composition
Sensor mote (MICA2)	A main board (MPR410) with the Atmega 128, 8 bit, 7.3827MHz CPU and the Chipcon CC1000, 38kb/s, CSMA radio circuit, and a sensor board (MTS300)
Sink node	A main board (MPR410) and a PC interface card with a serial port
Data proxy	A PC that communicates with the sink node via a serial port
User client	A PC that runs the user interface program

The operation of the WSN is as follows: Use the user client to post commands and queries. These queries are parsed by the data proxy and are disseminated via the sink node to the network. If a mote acquires data that satisfy a query, it sends the sensory data tuples to the data proxy through the sink node. The data proxy forwards the result to the user client.

3. VMNet Design

Our VMNet is designed via a divide-and-conquer approach. First, we analyze the target WSN, and divide the WSN into components. Second, we design the architecture of the emulator based on the architecture of the real WSN. Third, we design each emulated component based on its counterpart in the real WSN.

3.1. VMNet Architecture

The architecture of VMNet (Fig. 2) resembles that of a real WSN. It consists of the virtual sink node (VM 0) and other virtual motes connected through virtual channels. Real application software runs on the virtual motes for sensing, processing, and routing. Emulated radio signals travel on the virtual channels. Additionally, the Application User Interface (AUI) and the Network Manager (NM) reside on VM0 for application management (corresponds to the data proxy and the user client in the real WSN) and network emulation management respectively.

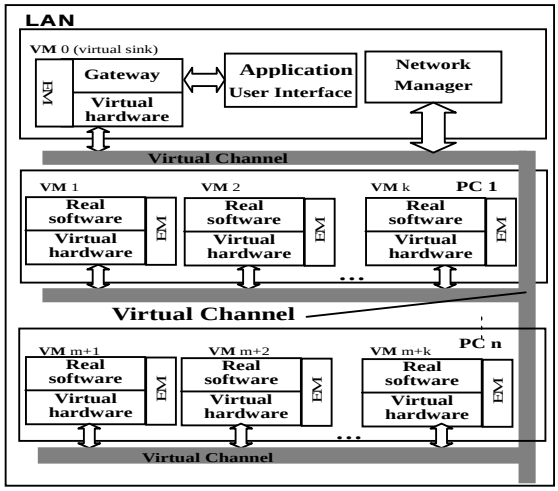

Fig. 1. VMNet architecture

VMNet is designed to be a distributed system in order to achieve fast emulation, high accuracy and scalability. It employs a wired Local Area Network (LAN) to emulate the wireless network. Each real mote in the target WSN is emulated by a program running on the LAN. From our past experience [13], this parallel architecture that VMNet adopts has shown high fidelity and scalability in emulating general wireless networks.

In brief, the architecture of VMNet abstracts the common features of WSNs. Although VMNet can only emulate one type of WSNs at one time, the generality of its architecture makes it easy to switch to other WSNs.

3.2. Virtual Mote

A virtual mote (VM) has three components: the virtual hardware, the real software and the Emulation Manager (EM). Fig. 3 shows the structure of a VM.

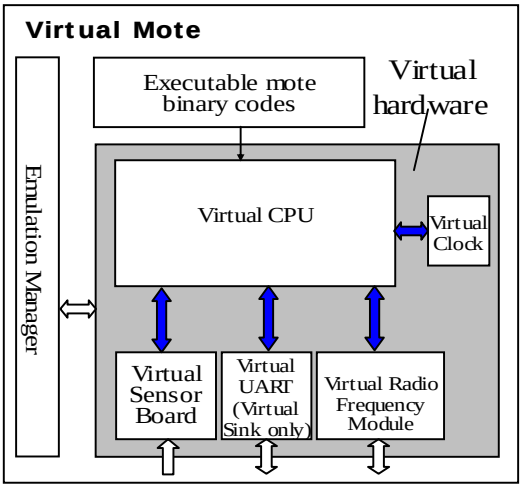

Fig. 2. Structure of a VM

The virtual hardware is the emulation of a mote's hardware (MICA2 in this paper). It is composed of the following units: a virtual CPU with a virtual clock, a Virtual Radio Frequency Module (VRFM) and a virtual sensor board. In the virtual sink node, there is a virtual UART, which emulates the serial port of the sink node. The virtual hardware units are the same type as their corresponding real hardware. For instance, the virtual CPU emulates the Atmega 128 CPU of MICA2 mote.

The virtual CPU and the virtual clock in a VM are critical for the accuracy of emulation, because they control execution and timing. The virtual CPU parses the executable binary codes of a mote and executes them. It also interacts with other virtual hardware units via the virtual I/O ports. The virtual clock is incremented by the virtual CPU per mote CPU clock cycle and records the virtual time in a VM.

The emulation manager manages mote emulation and logs the emulated actions, the execution time of various modes and runtime status of these units.

VM 0 (the virtual sink) is different from other VMs in that it has the virtual UART but no virtual sensor board. The difference is emulated by the gateway software, which operates on the virtual UART and disregards the virtual sensor board.

All three components – the virtual hardware, the real application software and the emulation manager, are separate from one another in a VM. There is a clear interface between the three components. This design ensures the reusability of our emulator when the target application or hardware changes. It also provides a reasonable solution to the conflict between the generality and the accuracy (specificity) of emulation.

3.3. Virtual Channel

The virtual channel generates network effects using three software modules: the bit error module, the delay module and the collision module (shown in Fig. 4). Let us first describe the transmission process of data on a virtual channel connected with a VM. When outgoing bits are sent from the Virtual Radio Frequency Module (VRFM) of the VM to the virtual channel, they pass through the three modules and stay in a

buffer (in the lower right corner of Fig. 4) for wrapping. When all bits of a packet arrive in the buffer, the virtual channel wraps them into a packet and sends out the packet via UDP. When an incoming UDP packet arrives at the virtual channel, it is put into a queue (lower left of Fig. 4) and is decomposed into bits to be sent to the VRFM of the VM via another buffer (on the left of Fig. 4).

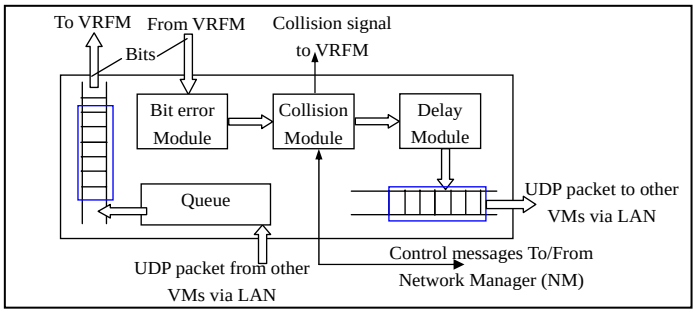

Fig. 3. Virtual Channel

The bit error module uses an experiential radio signal error data model to generate the error rate. The bit error rate model is a table with two attributes: distance and bit error rate, which is defined as: (number of error bits received by the receiver) / (number of total bits sent by the sender). The module randomly generates the bit error at a rate that the table specifies.

The transmission delay module adds a delay to the virtual time of the outgoing packet. The collision module emulates radio signal collision by performing two operations: *carrier sense* and *collision*. Both operations need information about the virtual time and the data transmission status of all VMs. This information is kept in the Network Manager.

In the *carrier sense* operation, the collision module asks the network manager whether if a sending VM can hear any VMs that are transmitting data. If so, the sending VM will wait a random time defined by the network protocols. In the *collision* operation, the collision module destroys the current bit to be sent on one of the two conditions: (1) another VM is transmitting and the sender of the bit can hear that transmitting VM, or (2) another VM is sending to the same destination as this sender.

3.4. Virtual Time

The major criterion for the accuracy of our emulation is the emulated time, or the virtual time. Mathematic models are one way to estimate time, but it is hard to achieve a high accuracy with simple models. In VMNet, we follow the approach of real execution. That is, the emulator executes the real software and measures the virtual time of the execution.

We have described the timing mechanism in the virtual CPU with a virtual clock in Section 3.2. Moreover, the working time of hardware peripherals such as sensing time and transmitting time are also emulated. Let us take the virtual sensor board as an example. When the virtual sensor board receives a command from the virtual CPU, it checks the virtual clock and after a delay (the length of delay is based on measure-

ments in real systems) sends an interruption signal to the interruption port connected to the virtual CPU. When the virtual CPU receives the interruption signal, it executes the sensor data interruption service program. Therefore, the virtual time together with the interruption is accurately emulated. The timing and the interruption of RFM and other hardware peripherals are emulated similarly.

Since the sleep mode of real motes is important for power efficiency, accurate emulation of WSN should consider the sleep mode. After the virtual CPU executes the "sleep" instruction, it should sleep until there is a timer interruption The VM advances its virtual clock by the sleep time, reports its status to the network manager and waits for synchronization.

Up to this point, we have discussed time emulation for individual VMs. Because VMs run simultaneously, synchronization is needed to ensure that the messages and the operations of VMs are in the same order with that of the target WSN.

The synchronization procedure is as follows: At the startup time, the network manager initializes its table of network status information including the total number of VMs *n* and the value of the virtual clock of each VM: vt_0, vt_1… vt_{n-1}. Whenever the VMs run for a predefined interval T, which is called the synchronization interval, they pause and report to the network manager. After every VM has reported to the network manager that its virtual clock has advanced by T, the network manager sends out a broadcast message to inform the VMs to resume running.

It is possible that when the fastest VMs (with a virtual time *vt*) are waiting, other VMs may exceed the fastest VMs. This does not matter as the message order is not affected and the exceeded time will be synchronized in next interval. In order to ensure correct ordering of messages, the virtual channel queues received UDP packets. The packets are sorted by their virtual time in the ascending order. This queue method avoids the semantic error: When a message is processed, it finds there is another message in the buffer that should be processed earlier.

In summary, the virtual time is carefully emulated in VMNet for accuracy. VMNet adopts a virtual CPU with a virtual clock for each VM to manage the timing. The virtual hardware peripherals of a VM generate interruptions with realistic delays. The sleep mode of a VM is considered and is gracefully handled. Synchronization of multiple VMs is performed periodically to ensure the correctness of emulation.

4. Related Work

Previous work, including Glomosim [12], Maisie[10] and SWiMNET [1], has shown that parallel and distributed architectures can speed up simulations. In this direction, our effort on VMNet is an outgrowth of our previous work on a distributed wire-line and wireless network emulation framework EMPOWER [13]. Our previous work EMWIN [14] gives the experiences in emulating wireless networks.

In the area of sensor network simulation and emulation, UC Berkeley's TOSSIM [7] simulates the network at the bit level. It is useful for debugging applications but it has not provided detailed timing information of the target. UCLA's EmStar [4][6] is another simulator of WSNs. It has not focused on detailed performance evaluation of the target WSN yet.

5. Conclusions and Future Work

We have presented our design of VMNet, a distributed emulator for WSNs with accurate system feedback. We are currently implementing VMNet, with a focus on the accurate estimation of application execution time. We plan to add power consumption emulation as well as mobility emulation to VMNet in the near future.

Acknowledgement

This work is part of the BLOSSOMS project [5]. Funding for this work is provided by the Hong Kong Research Grant Council through Grants HKUST6158/03E and HKUST6161/03E.

Reference:

[1] Azzedine Boukerche, Alessandro Fabbri. Partitioning Parallel Simulation of Wireless Networks. The 2000 Winter Simulation Conference (WSC), 2000.
[2] Crossbow Technology, Inc. http://www.xbow.com/.
[3] Robert Castañeda, Samir R. Das. Query Localization Techniques for On-Demand Routing Protocols in Ad Hoc Networks. Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, 1999.
[4] EmStar. http://cvs.cens.ucla.edu/emstar/.
[5] Wen Gao, Lionel M. Ni, Zhiwei Xu. BLOSSOMS: A CAS/HKUST Joint Project to Build Lightweight Optimized Sensor Systems on a Massive Scale. The IFIP NPC'04 Workshop on Building Intelligent Sensor Networks (BISON'04), 2004.
[6] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, Deborah Estrin. EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks. USENIX, 2004.
[7] Philip Levis, Nelson Lee, Matt Welsh, David Culler. TOSSIM: accurate and scalable simulation of entire TinyOS applications. The first international conference on Embedded networked sensor systems, 2003.
[8] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong. The design of an acquisitional query processor for sensor networks. SIGMOD, 2003.
[9] NS-2: The Network Simulator: http://www.isi.edu/nsnam/ns/.
[10] Joel Short, Rajive Bagrodia, Leonard Kleinrock. Mobile wireless network system simulation, Wireless Networks 1, 1995.
[11] Yong Yao, Johannes Gehrke. Query Processing in Sensor Networks. CIDR 2003.
[12] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. The 12th Workshop on Parallel and Distributed Simulations (PADS), 1998.
[13] Pei Zheng, Lionel M. Ni. EMPOWER: A Network Emulator for Wireless and Wired Networks. INFOCOM, 2003.
[14] Pei Zheng and Lionel M. Ni. EMWIN: Emulating a Mobile Wireles Network using a Wired Network. The International Workshop on Wireless Mobile Multimedia, 2002.