SEMU: A Framework of Simulation Environment for Wireless Sensor Networks with Co-simulation Model

Shih-Hsiang Lo, Jiun-Hung Ding, Sheng-Je Hung, Jin-Wei Tang, Wei-Lun Tsai and Yeh-Ching Chung
Department of Computer Science, National Tsing Hua University, Taiwan
{albert, adjunhon, claboy, garnet, welentsai}@sslab.cs.nthu.edu.tw, ychung@cs.nthu.edu.tw

Abstract. This paper presents a framework of simulation environment (SEMU) which allows developers to understand the behavior of applications or protocols for a wireless sensor network (WSN) before deploying real nodes in a physical environment. For eliminating the gap between simulation and real deployment, SEMU has supported fast real code emulation by dynamic binary translation technique. SEMU also models the controlled environment as virtual operating system (Virtual OS) to coordinate the interactions of large number of nodes. In addition, we have proposed a co-simulation model to enhance the accuracy of pure software simulation. Then a further synchronization problem will be addressed and resolved by the co-simulation model. The evaluation results show SEMU is really a fast scalable WSN simulator with real code emulation.

Index Terms: Simulator, wireless sensor networks, dynamic binary translation, hardware and software co-simulation

1 Introduction

A wireless sensor network (WSN) is a network composed of a large number of sensor nodes, which are deployed in the environment. Recently, with the rapid development of WSNs, providing development tools such as simulation environment before deploying real nodes in physical environments is getting more important. A well simulation environment can help developers build their prototype models to know the interactions and the behavior of each node. In addition, most of WSN applications will deploy a large number of nodes in a simulation environment. However, the simulation speed depends on the simulation fidelity and scale. Therefore how to build up a fast scalable WSN simulation environment with the fine-grained information is the main research problem in this paper.

In this paper, a framework of simulation environment (SEMU) is presented. SEMU has a first version implementation and an extension model for hardware and software co-simulation. In order to extract the real behavior of each node, SEMU supports real applications to run on the virtual nodes. And, the new development trend shows us the powerful nodes [2], [4], [9] are also applied to WSNs.

Consequently, our first implementation version of SEMU supports the virtual nodes to directly run the real Linux applications on the Linux platform.

For the extension of SEMU, co-simulation model, this paper has addressed two problems when SEMU uses pure software simulation models. One problem is that software sensing channel models are difficult to be built the same as real sensing channels, and furthermore they fail to interact with physical environments. The other problem shows that the pure software simulation needs a considerable amount of effort to model the behavior of real communication devices and real communication protocol. By the co-simulation model, SEMU can further satisfy the requirement of WSN development, and collect more realistic profiling information and physical environment conditions during simulation.

This rest of paper is organized as follows. In the next Section, we state the related work to compare different approaches of real code emulation and co-simulation model. Section 3 clearly illustrates the overall design of the simulation framework. Then Section 4 introduces the hardware and software co-simulation model based on SEMU. In Section 5 provides our evaluation results. Finally, Section 6 concludes this paper.

2 Related Works

In the literature, several WSN simulators have been proposed to support real applications targeted on different platforms. These approaches can be divided into two categories. One is static translation which maps the real code into the simulation platform before run time. The other is dynamic translation which interprets the real code during simulation. Besides, related works about co-simulation in WSN simulators are also described here.

TOSSIM [7] is a notable example to represent the static translation technique. TOSSIM is a discrete event simulator which can directly run a TinyOS [6] application through compilation support. This method is an excellent way to reduce the runtime overhead for code translation with advanced compiler supports. Nevertheless, the supported languages highly depend on the modified compiler.

There are some simulators using the dynamic translation technique [1], [3], [5]. Atemu [3] is a fine grained sensor node emulator for AVR processor based systems. Although the low-level emulation of the sensor node hardware can acquire the high-fidelity results, the run time interpretation overhead makes the emulation speed much slower than other approaches.

EmStar [5] is an environment for developing wireless embedded systems software in Linux-based software framework. EmStar provides a pure simulation, a true distributed deployment, and two hybrid modes. The software stack of Emstar is composed of several components, and each of them presents a Linux process with its own address space. However, it can not emulate the real binary codes which run on the real platform.

Embra runs as part of the SimOS simulation system [1]. To achieve high simulation speed, Embra uses dynamic binary translation (DBT) technique to generate code sequences which simulate the workload.

For co-simulation, SensorSim [10] use a gateway node and a simulated protocol stack to connect the real nodes. Since the real nodes and simulated nodes are not time synchronized, real node can not interact with simulated node on the correct time. EmStar [5] has proposed three kinds of hybrid modes for WSN simulation including data replay, ceiling array and portable array. Our co-simulation model is similar to the three hybrid modes of EmStar. However our co-simulation model can provide run time profiling to help SEMU to resolve synchronization problem. Hence our co-simulation model can improve SEMU for developing WSN applications with more accurate model.

3 The Framework of SEMU

When we design SEMU, we take following design issues into account: fast real code emulation for Linux application, a simulation engine for harmony and environment model. According to these design issues, we have proposed a framework for SEMU as Figure 1. It consists of five layers, VM layer, Communication layer, Virtual OS layer, Module layer, and Native OS layer. The framework has become as a backbone for our future extensions.

The top layer, Virtual Machine (VM) layer, achieves fast real code emulation on Linux platforms. In VM layer, a virtual node represents as an emulation of a real node. A virtual node can consist of several virtual machines. Through communication links between VM and Communication layer, virtual nodes can interact with the simulation environment. We use a modified QEMU as our VM layer [10].

Communication layer is partitioned into three parts including Communicator, Gateway and Connector. Communicator enables VM layer to communicate with Virtual OS layer. Gateway is used for hybrid simulation, as hardware and software co-simulation. As to Connector, it lets the Distributed GUI and the simulation engine can be run on different machines and work together.

The Virtual Operating System (Virtual OS) layer stands for the control center to harmonize the simulation. As general operating system, SEMU provides BootLoader to initialize the whole system and to boot Virtual OS. The BLR Shell provides a command interface of BootLoader for simulation users. SIM Kernel is also a service provider which helps virtual nodes to forward their service requests to Module layer and Native OS layer. Through OS Shell, run time simulation system can be operated by users.

In the Module layer, all of the components enhance the functionalities of the simulation framework. In the current work, the Module layer has supplied seven components. SIMState maintains all simulation status and references of the simulation objects in a centralized way. SIMState is configured with initial states by Configer before starting the SIM Kernel. In order to arrange the chaotic messages, Logger will analyze log information with classification. For simulating parallel execution of multiple nodes in a sequential computer, an adequate Scheduler needs to be applied. Time Manager is to resolve synchronization problems. Node Model supplies device configurations for developers to form a node element. Users can

integrate their protocol algorithms into the Protocol Stack. Environment will provide several models to reflect real conditions in physical environment.

The bottom layer of SEMU is Native Operating System (Native OS) layer. For system emulation, SEMU takes native operating system as the foundation of framework.

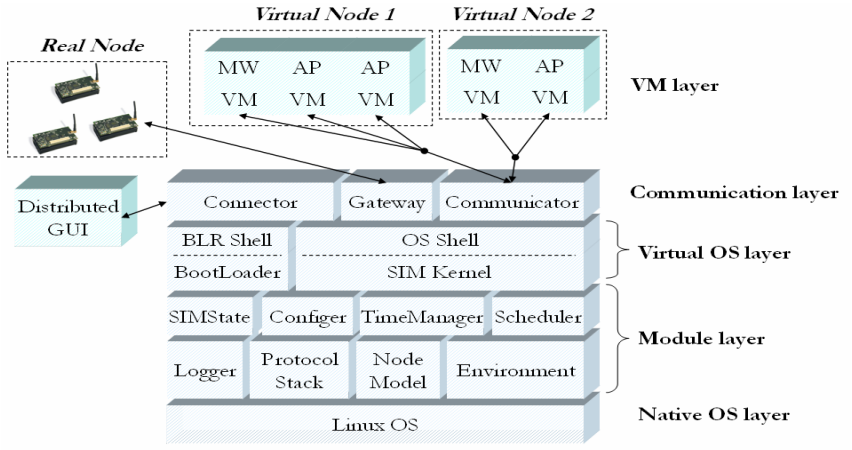

Figure 1: The architecture of SEMU.

4 Co-simulation Model

In this section, we discuss the extension model of SEMU, co-simulation model, and address the synchronization problem when hardware and software run together.

The co-simulation model supports the real communication and sensing channel by cooperating with real node devices. The co-simulation model consists of three components including Communication Agent, Sensing Agent and Environment Recorder. Communication Agent helps SEMU make use of real communication device and real communication protocol to gather more realistic communication latency and to support more accurate model than pure software simulation. Then Sensing Agent can provide real sensing channels for collecting raw data from physical environment. By Environment Recorder, packets and sensing information with timestamp can be recorded in the buffer, which is similar to Digital Video Recorder. SEMU can use Environment Recorder to track the real time events and make the events re-present.

We propose a WSN co-simulation model to extend the SEMU design. In our co-simulation model, SIM Kernel serves all virtual nodes to request actions, such as sending, receiving and sensing. SIM Kernel will choose an agent which can represent the realistic behavior of the node. When an agent is chosen for the request,

it will do the corresponding action and provide the run time profiling information. After finishing the action, it will notify the SIM Kernel of results including the execution time and the status for the action. Then SIM Kernel will request Time Manager whether the action can be completed. If Time Manager grants the request, the SIM Kernel will return the real data such as the packet or sensing data from environment recorder to the virtual node. Otherwise, SIM Kernel will block the virtual node until the request is granted by Time Manager.

From our co-simulation model, we must make sure that virtual nodes and the agents will cooperate with each other, virtual nodes will take the execution time of the agents into account, and virtual nodes will access the data according to their virtual time. Hence, in our co-simulation model, it needs to achieve the interaction synchronization, the time synchronization and the data synchronization.

For the interaction synchronization, when a virtual node requests an action to an agent, the action will be done by the agent and the status and the real execution time of the agent will be return from the agent. For the time synchronization, we need to profile the time of executing an action in an agent. The profiling result represents as number of clock cycles of executing the action in real device. The elapsed clock cycles will be translated into a virtual time according to cycles per instruction (CPI). Then the generated virtual time can help Time Manager to decide whether the action is granted or not. Therefore, we ensure that a virtual node and an agent can work in time synchronization. For the data synchronization, we use an environment recorder to collect information from physical environment such as sensing data and packets in wall time. The environment recorder will convert the wall time of the information to the virtual time. All of the information will be stored in the storage. The frequency of sensing or receiving depends on the requirement of a WSN application.

5 Evaluation Results

In our experiment, virtual nodes are deployed as grid and will cooperate to broadcast an event to whole network. The event will be sent by a specific virtual node deployed on the corner. Then this event will be relayed by a flooding protocol until all of the virtual nodes receiving it. We want to know how fast SEMU can complete the simulation as the number of virtual nodes increasing. We performed the experiment on a Celeron 3.0ghz machine with 1.5 GB of RAM running Linux 2.6.11. SEMU can run fast below 1250 virtual nodes. When the number of nodes is over 1250, the execution time of simulation increases quickly under the machine with 1.5 GB of physical RAM. Because if we create more virtual nodes, the simulation will use swap space, in which the simulation runs slowly. Hence, the scalability of SEMU highly relates to the resource management of Linux OS.

6 Conclusion

In this paper, we have presented a framework of SEMU to develop a WSN simulator. The framework allows developers to understand the behaviors of a WSN applications

or protocols before real deployment. Due to the trend of complex software platforms used in the WSN, such as Linux, the implementation of SEMU supports a real Linux application to run directly on the SEMU by fast real code emulation. We also have proposed a co-simulation model to enhance the accuracy of pure software simulation. The synchronization problem between virtual nodes and real nodes is addressed and resolved by the co-simulation model. Finally, the evaluation results show that SEMU can support fast real code emulation. Consequently, the framework of WSN simulation environment, SEMU, really assists developers in the development of WSN applications.

Acknowledgments. The work of this paper is partially supported by National Science Council and Ministry of Economic Affairs under NSC 95-2221-E-007-018 and MOEA 95-EC-17-A-04-S1-044.

References

1. E. Witchel , and M. Rosenblum, “Embra: fast and flexible machine simulation”, Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, p.68-79, May 23-26, 1996, Philadelphia, Pennsylvania, United States.
2. I. Downes, Leili B. Rad*, and H. Aghajan, “Development of a Mote for Wireless Image Sensor Networks” In Proc. of Cognitive Systems and Interactive Sensors (COGIS), March 2006.
3. J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. Karir, “ATEMU: A fine-grained sensor network simulator,” in Proceedings of SECON’04, First IEEE Communications Society Conference on Sensor and Ad Hoc Comunications and Networks, 2004.
4. L. Nachman, R. Kling, R. Adler, J. Huang, and V. Hummel, “The intel mote platform: a bluetooth-based sensor network for industrial monitoring.” in IPSN 2005, pp. 437–442, Apr. 2005.
5. L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin, “Emstar: a software environment for developing and deploying wireless sensor networks,” in Proceedings of the USENIX Technical Conference, 2004.
6. P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, “TinyOS: An operating system for wireless sensor networks” In Ambient Intelligence. Springer-Verlag, 2004.
7. P. Levis, N. Lee, M. Welsh, and D. Culler, “TOSSIM: accurate and scalable simulation of entire tinyOS applications”, Proceedings of the 1st international conference on Embedded networked sensor systems, November 05-07, 2003, Los Angeles, California, USA.
8. QEMU project. http://fabrice.bellard.free.fr/qemu/.
9. Stargate: a platform X project. http://platformx.sourceforge.net/.
10. Sung Park, Andreas Savvides, and Mani B. Srivastava, "SensorSim: a simulation framework for sensor networks", Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems, p.104-111, August 20-20, 2000, Boston, Massachusetts, United States.