Unified Simulation Framework for Protocol Stacks in Sensor Operating Systems*

JINMAN JUNG1, BONGJAE KIM1 AND JIMAN HONG2,*
1School of Computer Science and Engineering
Seoul National University
Seoul, 151-744 Korea
E-mail: {jmjung; bjkim}@os.snu.ac.kr
2School of Computing
Soongsil University
Seoul, 156-743 Korea
E-mail: jiman@ssu.ac.kr

Network simulations are widely used in sensor network research to design and evaluate network protocols. With the help of network simulation, protocol developer can reduce the time and cost involved in setting up an entire test environment containing multiple sensors and sink nodes. One of the major requirements in sensor simulation is to achieve accurate results. However, most sensor network simulators still suffer from the lack of realistic and available models for an accurate simulation of real protocols. Unfortunately, this can lead to unexpected or inaccurate simulation results which can result in the inappropriate decisions.

In this paper, we propose a unified simulation framework to improve the accuracy and extensibility of the protocol stack in sensor operating systems. The aim of this study is to provide more realistic capabilities and extensive models. Our approach is based on the implementation of real sensor protocols, which are slightly modified at some system dependent points and complemented with real world data. The virtualization of the sensor protocol stack in our framework has an ability to provide a generalized protocol stack interface and share various protocols among multiple simulators. We demonstrate an implementation of the proposed framework adapted to existing well-known simulation packages. Experiments on the sensor protocol stack show that the proposed framework yields more extensive and accurate results.

Keywords: network simulator, simulation framework, wireless sensor network, sensor operating systems, virtual protocol stacks

1. INTRODUCTION

Wireless sensor networks (WSNs) are networks formed by a set of sensor nodes and sink node(s) connected through wireless links. These sensor nodes sense data in application domains (ranging from wildlife monitoring to vehicle tracking) and then transmit the sensed data to a sink node through wireless multi-hop routing. Sensor nodes have processing and communication capacities, whose main tasks include controlling sensors, processing sensed data, and transmitting collected data to the sink node. A typical sensor node has a low power CPU, tiny memory (RAM/ROM), R/F module, many kinds of

Received May 31, 2011; accepted March 31, 2012.
Communicated by Junyoung Heo and Tei-Wei Kuo.
* This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2012-C1090-1221-0010).
+ Corresponding author: jiman@ssu.ac.kr.
sensing units, and constrained battery power. For example, Berkeley’s MICA motes only have an 8-bit CPU, 4KB RAM, and only two AA alkaline batteries. The most energy-consuming component is the R/F module, which provides the wireless communications. The energy consumption when transmitting 1 bit of data on a wireless channel is similar to thousands of cycles of CPU instructions [1]. Thus, the energy efficiency of network protocols for wireless sensor networks largely affects the energy consumption and network lifetime of wireless sensor networks [2]. Recently, many network protocols have been made for WSNs including routing, clustering, and medium-access-controlling protocols. When developing such protocols, it is very important to measure their performance on the networks. However, the sensor networks generally consist of a huge number of sensor nodes, and this makes the developers use network simulators. Many network simulators have been made to test and evaluate the performance of the developed mechanisms in WSNs. However, network simulators still have some disadvantages including difficulties to program them.

Primarily, ns-2 [3], developed by LBNL, UC Berkeley, CMU, and many other research institutes, is a famous simulator not only for wired and wireless networks, but for previous works related to network protocols. Because of the use of OTcl (object-oriented tool command language) and C++, however, ns-2 is very hard to program and the internal structures are too complex to understand thoroughly. For this reason, many efforts have been made to implement easier simulation packages compared to ns-2.

GloMoSim [4] is a mobile network simulation library developed at UCLA. It was designed as a set of library modules using PARSEC, a C-based parallel simulation language. Though GloMoSim is extensible and suitable for wireless ad-hoc and sensor networks, protocol developers have a big burden to understand the entire structures of it and be familiar with PARSEC.

QualNet [5] extends GloMoSim for commercial use to the field of industry as well as academics with the advantage of verified built-in network protocols in the simulation package. Nevertheless, the process of adding new protocols in QualNet is rather complex and contains some unnecessary procedures.

OPNET [6] stands for OPTimized Network Engineering Tools. OPNET Modeler is based on a series of hierarchically-related editors. Among these editors, a process editor describes the behavior and functions of the modules. It uses a finite state machine to describe the protocols in detail. Each state of a process contains C/C++ code for specific control. For this reason, protocol developers need to understand the process model to implement their protocols in OPNET.

OMNeT++ [7] is a discrete event simulation environment. It provides the basic machinery and tools to conduct simulations, adopted a component-based architecture. In OMNeT++, the network simulation environments, called models, are assembled from several reusable modules based on a NED (NEtwork Description) language. Therefore, protocol developers have to be familiar with the concept of models, components, and the NED language while developing their protocols in OMNeT++.

TOSSIM [8], which was made by UC Berkeley’s TinyOS project team, is a well-known simulator for applications running on TinyOS [9]. The main goal of TOSSIM is to provide a debugging method for TinyOS and its applications. However, it can simulate only the sensor applications ported to the i386 architecture. It is not precise like ATEMU and AVRORA, and it cannot be used for other sensor operating systems and
RealSSim [10] is a wireless network simulator for scalable and fine-grained simulations in WSNs. It considers the structure of buildings and the quality of the construction materials for an accurate radio propagation analysis. However, developers need to understand their event handling system to implement some of the network protocols.

One of the major requirements in sensor network simulations is to achieve accurate results. However, most network simulators use abstract and simplified models that might not behave in the same way as real protocols. Unfortunately, this can lead to unexpected or inaccurate simulation results which can make the inappropriate decisions. Not only for that, but also a variety of implemented models at different levels of fidelity can produce significantly larger different and discrepant results between network simulators. In addition, they suffer from the lack of available simulation models. Since simulation models supported by one simulator environments are limited, developers may need to run the simulation with several simulators. For example, the GloMoSim [4] simulation environment is not suitable to analyze indoor sensor network systems due to the lack of the indoor radio propagation model. In this case, developers familiar with the GloMoSim development environment need to use another sensor simulator such as RealSSim [10] supporting the multi-wall model [11] and the ray-tracing model [12] which considers the properties of building materials for extensive research. Also, developing in a different simulator environment is a time-consuming and difficult procedure in which the developers are required to learn new host languages and libraries.

In this paper, we propose a unified simulation framework for the simulation of the sensor protocol stack. The goal is to provide more real-world capabilities and available models for higher fidelity simulation of sensor protocol stack. The approach in this study is based on the implementations of real protocols, which are slightly modified at some system dependent points and complemented with real world data. Our framework also uses a virtualization of sensor protocol stack that has an ability to provide a single generalized protocol stack interface and share various protocols among multiple simulators. By sharing realistic protocols over several simulators, developers can get more extensive simulation results. We implemented and adapted a unified simulation framework on the GloMoSim, RealSSim, and CoSim. Experiments over several simulators give more extensive results and higher fidelity simulations.

The remainder of this paper is organized as follows. In section 2, the simulation model for the sensor protocol stack is described. In section 3, a unified simulation framework for improving the accuracy and extensibility of the sensor protocol stack is proposed. In section 4, the performance of the proposed framework is evaluated. Finally, in section 5, the conclusions are presented.

2. SENSOR PROTOCOL STACK SIMULATION

In this section, a lightweight protocol stack for WSNs is described, and the main reasons for the differences between the simulation results and real world results are discussed.

The protocol stack developed for WSNs needs to be energy efficient due to resource constraints, and therefore, the main goal in a sensor protocol stack is to minimize power
consumption to prolong the lifetime of the sensor nodes. Fig. 1 shows a typical sensor protocol stack and simulation model. The lightweight protocol stack can be classified into four layers (from top to bottom): the application layer, the network layer, the medium access control layer (MAC layer), and the physical layer (PHY layer). In the application layer, sensor nodes are responsible to process sensed data and transmit the collected data to a sink node. The network layer is assigned to deliver collected and processed data to a sink node, or to disseminate control messages to other nodes through data dissemination. The role of the MAC layer is to control access to the shared medium among the sensor nodes. Finally, the PHY layer deals with the actual transmission of the bits over the wireless medium for the WSNs. Typically, the PHY layer and some part of the MAC layer are implemented in the radio transceiver hardware such as cc2420 [13]. The sensor protocol stack, which consists of these sub-layers, is used to support various sensor communications through the wireless channel.

Evaluation and analysis for the performance of the sensor protocol stack is an important aspect of sensor network research. Network simulators can be used to study the overall system performance or analyze the characteristics of these protocols under various environments. Measurements from simulation, however, are usually not as convincing as ones obtained from protocols on real sensor equipment. The followings are some significant reasons as to why simulations differ from real experiments.

First, simulated protocols are difficult to implement in the same way as existing real protocols to satisfy their design specifications. This is because simulators have a limited and different development environment, which includes host language, libraries, and system dependent platforms, compared with real protocols. Second, the difference in the execution and control of the model can cause discrepancies in the results. For example, most sensor simulators are based on a discrete event driven approach while the execution models of most sensor operation systems are classified into multi-thread based models.
and event driven models as shown Fig. 2. The control flow can be affected by the different execution models. An inverted ordering of the simulation events and a small latency due to this can affect significantly the simulation results. Third, simulators with a proper level of hardware abstraction can have different behaviors since it is hard to capture the lower-level characteristics of sensor hardware platforms. Finally, the simulation models cannot exactly represent the real world sensor environments such as radio propagation, wireless channel, and mobility.

Fig. 2. Sensor operating system and simulation execution model.

The accuracy of the simulation results for the sensor protocol stack depends on the fidelity or diversity of the input models supported in the simulation environment. Most existing network simulators provide a limited environment in which protocol developers suffer from the lack of available and realistic input models for the particular environments of the sensors. Simulation models developed in single simulation environment might not even be feasible for other environments due to a lack of interoperability and portability. In this paper, a united framework for sensor protocol stack simulation is presented in which the implementation of real world protocols is used in order to obtain results. Furthermore, by integrating several simulators including GloMoSim, and RealSSim, the proposed framework allows developers to share protocols with each other. It provides more extensive simulation environments in which developers can use more realistic and available models provided by various heterogeneous simulators.

3. UNIFIED SIMULATION FRAMEWORK FOR PROTOCOL STACK IN SENSOR OPERATING SYSTEM

In this section, we present a protocol abstraction for high fidelity simulations through the use of real protocols. A unified simulation framework is also proposed, which can be adapted to existing simulators. This framework allows protocol developers to share protocols. This methodology can help them achieve more extensive and accurate results because protocols can be simulated in multiple simulators with different models.
3.1 Protocol Abstraction for Simulation

For simulation of the sensor protocol stack, a protocol has to be modeled and the implementation details of a particular set of functionality have to be hidden. Protocol abstraction is required to make an intensive investigation of protocol characteristics in order to abstract some of its parts that are not relevant for the generalization. By observing many available protocol stacks, the real protocol can be abstracted and provide a common API to deal with them. For example, the characteristics of protocols in a network layer can be classified into several categories (Table 1). AODV [14] is a well-known reactive routing protocol, which executes single-path and almost fixed-path multi-hop routing. It has no aggregation function, and synchronization is not required on AODV. In case of LEACH [15], it executes multi-hop communication, single-level clustering, and time synchronization between participants and their cluster head. In the same way, the characteristics of every routing and clustering protocol can be extracted with the categories. For data transfer, the Flooding [16] method can also be used. Flooding is the simplest method for multi-hop communication without having to maintain routing caches in which all the nodes rebroadcast messages when a node receives a packet for the first time.

<table>
<thead>
<tr>
<th>Network Protocol</th>
<th>Characteristics to be categorized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>One-hop vs. Multi-hop Communication</td>
</tr>
<tr>
<td></td>
<td>Async. Vs. Synchronized Timing Mgmt.</td>
</tr>
<tr>
<td></td>
<td>Routing vs. non-Routing Cache</td>
</tr>
<tr>
<td>Routing</td>
<td>Proactive vs. Reactive Routing</td>
</tr>
<tr>
<td></td>
<td>Randomized vs. Fixed-path Routing</td>
</tr>
<tr>
<td></td>
<td>Single-path vs. Multi-path Routing</td>
</tr>
<tr>
<td>Clustering</td>
<td>Multi-level vs. Single-level Clustering</td>
</tr>
<tr>
<td></td>
<td>Randomized vs. Fixed Head Selection</td>
</tr>
<tr>
<td></td>
<td>Variable vs. Fixed Clustering Period</td>
</tr>
</tbody>
</table>

In this study, the approach for protocol abstraction is intended to provide an interface that is consistent enough to allow for the easy use of protocols (e.g., plug and play) by any simulator while being flexible enough to allow for the sophisticated use of specialized protocol features. Fig. 3 shows the comparison between the protocol abstraction and the real protocol, RENO. RENO (a variant of AODV) is implemented in the network layer of NanoQplus. The protocol abstraction for the simulation needs to be designed so to offer a single generalized interface as a layer of the protocol stack, and therefore, each protocol has to provide a unified interface to communicate with the protocols of both the upper and lower layer including the recv_from_up, recv_from_down, send_to_up, and send_to_down functions (Fig. 3 (a)). Each Protocol also has certain functions to provide a service suitable to the functional specification of that layer. Since the simulation packages usually use a discrete event scheduler, some functions for the protocol abstraction need to contain support for event mechanisms causing the protocol to trigger the appropriate event handler.
To maintain information on the state of the protocol, sensor nodes for the simulation keep two data structures: a local session to manage their state and to track the results of various operations done within its protocol service, and a global session needed by all the sensor nodes to share all the information for the simulation. For example, protocol specific information such as routing table and message sets are stored in the local session allocated to the node. To ensure platform-independent development of the sensor protocol stack, the input parameters such as the number of nodes and the topology size provided in the host simulator need to be translated into a generalized data format in either the local or global sessions.

### 3.2 High Fidelity Simulations with the Use of Real Protocols

Our framework uses a high fidelity methodology by reusing existing real network protocol stacks with minimal modifications. Since most parts of the protocol stack code may not have anything to do with the platforms or operating systems but rather just with the specific implementing algorithms, they can be simply linked to a simulator with some supplementary components necessary to integrate the real protocol stack code into the proposed unified framework.

On the other hand, system dependent parts, for example, thread specifics or device drivers, should be re-implemented with hardware models of abstraction to a variety of platform dependent notations while the functionality remains equivalent, as illustrated in Fig. 4. This hardware abstraction provides the functionality actually runs on constructed under our methodology while it making system specifics be hidden for the protocol developers. For example, a hardware model for a transceiver model has a variety of timing properties for each operation of that module. This information on the hardware models for the protocols can be supported by a set of callback functions.

Fig. 5 (a) shows an example of a function call graph for the send function of the AODV protocol on the real NanoQplus platform. `reno_route_table_lookup()` is a platform-independent function since it simply performs a selection algorithm and finds an ID of the immediate node for destination in the routing table without the need for developers to actually understand the system. This platform-independent function can be directly reused as a simulation function (Fig. 5 (b)). If there is no cached entry in the routing table,
the source node initiates a route discovery process by broadcasting a Route Request (RREQ) packet with the `reno_send_rreq` function. This function sends a RREQ by performing the `reno_send_to_mac` function. However, this function is platform-dependent since it depends on the transceiver device used (e.g., CC2420 [12]), and therefore, abstraction needs to be done with the support of information, which encapsulates the timing properties of that module. Additionally, `nos_thread_sleep_ms` function is used to wait for receiving a Route Reply (RREP) since NanoQplus is a multi-thread based sensor operat-
ing system. It has to be replaced by the use of the `settimer` function wrapped by the unified simulation framework while still able to maintain the `nos_thread_sleep_ms` semantics on the multi-thread runtime system. This is because the notion of control flow on an arbitrary platform can differ from simulation semantics.

All external protocol plug-in components exchange data with the nodes of the host simulator through events, packets, or sessions. The framework, therefore, has to make sure that the client input is validated before it is processed to prevent unnecessary access to the protocol. For better reliability, a wrapper is used, which verifies arguments and return values and accesses as shown Fig. 4. An invalid or unexpected input would trigger exceptions, which would be thrown out to indicate to the host simulator that the input was invalid. It can also check and securely update the protocol components.

This reuse of real protocols can lead to more accurate results than existing simulators in which abstraction has taken a lot away from the real implementation of protocols. Not only does this reduce the cost and time of development, but also makes the protocol on the framework behave closer to the real one for a sensor on a detailed level.

### 3.3 Unified Simulation Framework for an Extensive Simulation

One of the goals is to design and develop a unified framework interface for a sensor protocol stack. The unified interface provides simulator developers with the capability for abstraction of the sensor protocol stack and with an interface that is easier to program. The approach uses a runtime protocol plug-in facility and manages protocols as dynamically loadable components so that the simulator code does not change even if new custom protocols are added. Thereby, this supports dynamic switching between executing protocols without restarting the simulator. Sensor network protocols implemented with a simulation language along with libraries cannot be easily used in another environment due to the lack of portability between the different simulators and the interoperability between several simulation models. The use of unified simulation frameworks can be a mean for protocol developers to become less dependent of the specificities of the simulator engines and sensors platforms, and become less complex due to the high-level simulation functions, as well as the reuse of existing simulation models provided by the host simulator. Protocol developers do not need to understand whole structures of the simulator any more. Similar to middleware between applications and operating systems, the proposed unified simulation framework is a software layer that lies between the host simulator and protocols. The software layer allows protocols contained in one simulator to be accessed through another. To share these unified protocols, a simulator has to be built based on the proposed unified framework interface. Since each simulator has a different set of environment models, they show a subtle difference in meaning between simulation results. The protocol developer can delicately compare and analyze with more multiple simulation results. Fig. 6 shows the unified framework interface and the external components. In this approach, protocols are built out of general interface, sharable libraries and plug-in components (represented as unified protocols in Fig. 6), each of which implements a protocol communication or data sharing functionality. As a result, developers can share their network protocol modules among multiple simulators, and thereby, developers can get more extensive simulation results.
4. PERFORMANCE EVALUATION

In this section, the performance of the proposed unified simulation framework is presented from several aspects. For this, the unified simulation framework was implemented and adapted on GloMoSim, RealSSim, and CoSim. This study shows that the methodology yielded more extensive and accurate simulation results with realistic protocols shared between multiple simulators.

4.1 Simulation Setup

For the experiments in this study, the implementation of AODV and CSMA protocols, which were slightly modified at some system dependent points was considered. Since these protocols were also taken into consideration in the unified framework interface, the proposed framework has the ability to run CoSim, RealSSim [10], and the widely used GloMoSim [4] simulation environments due to the adaptation layer between the host simulator and the protocols. This means that the approach used in this study allows developers to get more extensive and accurate results with most of the available models and useful features provided from each simulator. GloMoSim, RealSSim, and CoSim have different features and environment models as described in Table 2, which leads to different degrees of accuracy.
Table 2. Some models and features of GloMoSim, RealSSim, and CoSim.

<table>
<thead>
<tr>
<th>Simulators</th>
<th>Environment models</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>GloMoSim</td>
<td>Energy: WaveLan</td>
<td>Parallel simulation capacity</td>
</tr>
<tr>
<td></td>
<td>Mobility: Random waypoint</td>
<td>Scalable up to tens of thousands of nodes</td>
</tr>
<tr>
<td></td>
<td>Radio: Free space/ Two ray</td>
<td>Fast simulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RealSSim</td>
<td>Energy: Sensor Specific</td>
<td>Dedicated scenario designer</td>
</tr>
<tr>
<td></td>
<td>Mobility: Random waypoint</td>
<td>Supporting realistic models and protocols</td>
</tr>
<tr>
<td></td>
<td>Radio: Three dimensional ray tracing</td>
<td>statistical graphing tool</td>
</tr>
<tr>
<td>CoSim</td>
<td>Energy: Sensor Specific</td>
<td>Instant playback of simulation results</td>
</tr>
<tr>
<td></td>
<td>Mobility: Random waypoint</td>
<td>Wizard-based development environments</td>
</tr>
<tr>
<td></td>
<td>Radio: Free space</td>
<td>Easy to develop and simulate for sensor</td>
</tr>
</tbody>
</table>

Here, a case study is presented to show the performance of the proposed unified framework, which yields more extensible and accurate results. In the simulation, sensor nodes were considered randomly distributed in a square area $500 \times 500$ m. Each node had the same transmit power of coverage, which was $120$ m. The traffic of the activated nodes was set to be the constant bit rate (CBR) with a packet size of $512$ bytes. The simulations ran for $150$ seconds.

Fig. 7 (a) shows a comparison of the average energy consumption per node obtained from three simulators. The simulation results of GloMoSim show some differences in the energy consumption compared with that of CoSim and RealSSim. This is mainly because GloMoSim uses the waveLan energy model [17]. It is not realistic enough, as it does not distinguish the sleep state as well as the low power mode for the sensor node. The in-
crease of the energy consumption in GloMoSim was less than the other simulators. In this case, the average energy consumption is depended on the energy consumed in the listen state. By sharing the AODV protocol implemented under the proposed framework interface, protocol developers can see the difference in the simulation results obtained between the multiple simulators without having to develop another protocol.

4.2 Extensive Simulation Results

Extensive simulation results also show that the proposed framework could significantly improve the accuracy of the simulation. Although the GloMoSim simulator has an unsuitable energy model for sensor networks, it is useful when developers want to obtain extensive results precisely between different radio propagation models because the accuracy of the radio propagation model has a strong impact on the simulation result. Figs. 8 (a) and (b) show the average energy consumption of the AODV protocol with the free space model [18] and Rayleigh model [19] according to the node’s density and velocity.

![Fig. 8. (a) Energy consumption with different radio models according to the density; (b) Energy consumption with different radio models according to the velocity.](image)

There are some similarities between the different radio models in the global behavior but they differ by the increase in the average energy consumption per node as shown in the Figs. 8 (a) and (b). This is because small movements of the receiver can have large effects on energy consumption due to multi-path propagation of the radio waves in the Rayleigh radio model. To obtain simulations that are more detailed, other extensive results are possible from the RealSSim simulator without any modifications of the protocols or models. RealSSim supports indoor scenario designer with the multiwall model and the ray-tracing model, which considers the properties of the building materials.

Fig. 9 shows the different behaviors of energy consumption for outdoor and indoor scenarios with the RealSSim simulator. The energy consumption in the indoor scenarios was higher than in the outdoor scenarios (Fig. 9). The difference between the outdoor and indoor scenarios can be associated with the presence of the structure for the building and the quality of the building materials such as the walls, windows, and doors [10].
Nevertheless, RealSSim suffers from the lack of available mobility models for mobile sensor networks. In this case, the simulation can be extended with both the CoSim and GloMoSim simulator for a more accurate simulation. For example, the RWMM [20] or RDMM [21] can be used under GloMoSim and CoSim simulation environments, respectively, as shown in Fig. 10.

Fig. 9. Comparison of the energy consumption for AODV and directed diffusion protocol in outdoor and indoor scenarios with the RealSSim simulator.

Fig. 10. Energy consumption with different mobility models.

5. CONCLUSION

This work presents a unified simulation framework for the sensor protocol stack. Our goal was to provide more real world capabilities and available models for higher fidelity simulations of the sensor protocol stack. The approach in this study was based on the implementation of real sensor protocols, which were slightly modified at some system dependent points and complemented with hardware models. This framework also used virtualization of the sensor protocol stack. This enables protocol developers to obtain multiple simulation results between several simulators without having to develop any other protocols or without any effort for the portability of the models. Thus, developers can get more extensive simulation results by only developing the sensor protocol stack once under our framework environments. For the sake of validity, we implemented the unified simulation framework and adapted it to GloMoSim, RealSSim, and CoSim. We show that our methodology yields more extensive and accurate simulation results with realistic protocols shared between multiple simulators.

REFERENCES

2. J. Lian, K. Naik, and G. B. Agnew, “Data capacity improvement of wireless sensor


Jinman Jung received his B.S. degree in Computer Science from Seoul National University, Seoul, Korea, in 2008. He is currently a Ph.D. student of School of Computer Science and Engineering, Seoul National University, Seoul, Korea. His current research interests include operating system, embedded systems, mobile communications, and sensor network systems.

Bongjae Kim received his B.S. degree in Computer Engineering from Kwangwoon University, Seoul, Korea, in 2008. He has been with School of Computer Science and Engineering, Seoul National University since 2009, where currently he is a Ph.D. candidate student. His research interests include operating systems, fault tolerance computing systems, embedded systems, and sensor networks.

Jiman Hong received the B.S. degree in Computer Science from Korea University, Seoul Korea in 1994 and the M.E. and Ph.D. degrees in Computer Engineering from Seoul National University, Seoul Korea, in 1997, and 2003, respectively. He was an assistant professor at Kwangwoon University from 2004 to 2006. He has been with School of Computing, Soongsil University, Seoul, Korea, since 2007, where currently he is a Professor. From 2000 to 2003, he served as a Chief of Technical Officer in the R&D center of GmanTech Incorporated Company, Seoul, Korea. His research interests include embedded operating systems, fault tolerance computing systems, distributed computing systems, and sensor network systems.