A Software-Hardware Emulator for Sensor Networks

Jingyao Zhang, Yi Tang, Sachin Hirve, Srikrishna Iyer, Patrick Schaumont, and Yaling Yang

Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24060
Email: {jingyao, yt22, hsachin, skr, schaum, yyang8}@vt.edu

Abstract—Simulators are important tools for analyzing and evaluating different design options for wireless sensor networks (sensornets) and hence, have been intensively studied in the past decades. However, existing simulators only support evaluations of protocols and software aspects of sensornet design. They cannot accurately capture the significant impacts of various hardware designs on sensornet performance. As a result, the performance/energy benefits of customized hardware designs are difficult to be evaluated in sensornet research. To fill in this technical void, in this paper, we describe the design and implementation of SUNSHINE, a scalable hardware-software emulator for sensornet applications. SUNSHINE is the first sensornet simulator that effectively supports joint evaluation and design of sensor hardware and software performance in a networked context. SUNSHINE captures the performance of network protocols, software and hardware up to cycle-level accuracy through its seamless integration of three existing sensornet simulators: a network simulator TOSSIM [1], an instruction-set simulator SimulAVR [2] and a hardware simulator GEZEL [3]. SUNSHINE solves several sensornet simulation challenges, including data exchanges and time synchronizations across different simulation domains and simulation accuracy levels. SUNSHINE also provides hardware specification scheme for simulating flexible and customized hardware designs. Several experiments are given to illustrate SUNSHINE’s simulation capability. Evaluation results are provided to demonstrate that SUNSHINE is an efficient tool for software-hardware co-design in sensornet research.

I. INTRODUCTION

Over the past few years, we have witnessed an impressive growth of sensornet applications, ranging from environmental monitoring, to health care and home entertainment. A remaining roadblock to the success of sensornets is the constrained processing-power and energy-budget of existing sensor platforms. This prevents many interesting candidate applications, whose software implementations are prohibitively slow and energy-wise impractical over these platforms. On the other hand, in the hardware community, it is well-known that the specialized hardware implementation of demanding sensor tasks can outperform equivalent software implementations by orders of magnitude. In addition, recent advances in low-power programmable hardware chips (Field-Programmable Gate Arrays) have made flexible and efficient hardware implementations achievable for sensor node architectures [4]. Hence, the joint software-hardware design of a sensornet application is a very appealing approach to support sensornets.

Unfortunately, joint software-hardware designs of sensornet applications remain largely unexplored since there is no effective simulation tool for these designs. Due to the distributed nature of sensornets, simulators are necessary tools to help sensornet researchers develop and analyze new designs. Developing hardware-software co-designed sensornet applications would have been an extremely difficult job without the help of a good simulation and analysis instrument. While a great effort has been invested in developing sensornet simulators, these existing sensornet simulators, such as TOSSIM [1], ATEMU [5], and Avrora [6] focus on evaluating the designs of communication protocols and application software. They all assume a fixed hardware platform and their inflexible models of hardware cannot accurately capture the impact of alternative hardware designs on the performance of network applications. As a result, sensornet researchers cannot easily configure and evaluate various joint software-hardware designs and are forced to fit into the constraints of existing fixed sensor hardware platforms. This lack of simulator support also makes it difficult for the sensornet research community to develop a clear direction on improving the sensor hardware platforms. The performance/energy benefits that are available to the hardware community therefore remain hard to reach.

To address this critical problem, we developed a new sensornet simulator, named SUNSHINE\(^1\) (Sensor Unified aNalyzer for Software and Hardware in Networked Environments), to support hardware-software co-design in sensornets. By the integration of a network simulator TOSSIM, an instruction-set simulator SimulAVR, and a hardware simulator GEZEL, SUNSHINE can simulate the impact of various hardware designs on sensornets at cycle-level accuracy. The performance of software network protocols and applications under realistic hardware constraints and network settings can be captured by SUNSHINE.

The rest of the paper is organized as follows. Section II provides a description of SUNSHINE’s architecture. Section III discusses cross-domain techniques used in SUNSHINE. Section IV describes SUNSHINE’s hardware simulation support. Section V provides experiment results and evaluation of SUNSHINE. Section VI introduces some related network simulators and makes comparisons between SUNSHINE and other

\(^1\)SUNSHINE is an open source software, the code is keeping updated and can be checked at http://sunshine-sim.sourceforge.net.
sensornet simulators. Section VII discusses future work of SUNSHINE. Finally, Section VIII provides some conclusions.

II. SYSTEM DESCRIPTION

SUNSHINE combines three existing simulators: network domain simulator TOSSIM [1], software domain simulator SimulAVR [2], and hardware domain simulator GEZEL [3]. In the following, we first briefly introduce these three simulators. Then, we introduce SUNSHINE’s system architecture and its simulation process.

A. System Components

1) TOSSIM: TOSSIM [1] is an event-based simulator for TinyOS-based wireless sensor networks. TinyOS is a sensor network operating system that runs on sensor motes. TOSSIM is able to simulate a complete TinyOS-based sensor network as well as capture the network behaviors and interactions. TOSSIM provides functional-level abstract implementations of both software and hardware modules for several existing sensor node architectures, such as the MICAz mote. In TOSSIM, an event-based network simulator, sensor nodes' behaviors are regarded as functional-level events, which are kept in TOSSIM’s event queue in sequence according to the events’ timestamps. These events are processed in ascending order of their timestamps. When the simulation time arrives at one event’s timestamp, that event is executed by the simulator.

Even though TOSSIM is able to capture the sensor motes behaviors and interactions, such as packet transmission, reception and packet losses at a high fidelity, it does not consider the sensor motes processors’ execution time. Therefore, TOSSIM cannot capture the fine-grained timing and interrupt properties of software code.

2) SimulAVR: SimulAVR [2] is an instruction-set simulator that supports software domain simulation for the Atmel AVR family of microcontrollers which are popular choices for processors in sensor node designs. SimulAVR provides accurate timing of software execution and can simulate multiple AVR microcontrollers in one simulation. SimulAVR is also integrated into the hardware domain simulator in SUNSHINE, and through this integration, the detailed interactions between sensor hardware and software can be evaluated. Currently, SimulAVR does not support simulation of sleep mode or wakeup mode of sensor nodes. We have added sleep and wakeup schemes to provide simulation support for energy saving mode of sensor networks.

3) GEZEL: GEZEL [3] is a hardware domain simulator that includes a simulation kernel and a hardware description language. In GEZEL, a platform is defined as the combination of a microprocessor connected with one or more other hardware modules. For example, a platform may include a microprocessor, a hardware coprocessor, and a radio chip module. To simulate the operations of such a platform, one has to combine software simulation domain, which captures software executions over the microprocessor, and hardware simulation domain, which captures the behaviors of hardware modules and their interaction with the microprocessor. GEZEL is able to provide a hardware-software co-design environment that seamlessly integrates the hardware and software simulation domains at cycle-level. GEZEL has been used for hardware-software co-design of crypto-processors [7], cryptographic hashing modules [8], and formal verification of security properties of hardware modules [9], etc. GEZEL models can be automatically translated into a hardware implementation that enables a user to create his/her hardware, to determine the functional correctness of the custom hardware within actual system context and to monitor cycle-accurate performance metrics for the design.

GEZEL is the key technology to enable a user to optimize the partition between hardware and software, and to optimize the sensor node’s architecture. With the support of GEZEL, the simulator can capture the software-hardware interactions and performance at cycle-level in a networked context.

B. System Architecture

SUNSHINE integrates TOSSIM, SimulAVR and GEZEL to simulate sensornet in network, software, and hardware domains. A user of SUNSHINE can select a subset of sensor nodes to be emulated in hardware and software domains. These nodes are called cycle-level hardware-software co-simulated (co-sim) nodes and their cycle-level behaviors are accurately captured by SimulAVR and GEZEL. Other nodes are simulated in network domain by TOSSIM and only the high-level functional behaviors are captured. These nodes are named TOSSIM nodes. SUNSHINE is able to run multiple co-sim nodes with TOSSIM nodes in one simulation. The network topology in the right part of Figure 1 illustrates the basic idea of SUNSHINE. The white nodes are TOSSIM nodes, which are simulated in network domain, while the shaded nodes are co-sim nodes, which are emulated in software and hardware domains. When running simulation, these TOSSIM nodes and co-sim nodes interact with each other according to the network configuration and sensornet applications. Cycle-level co-sim nodes can show details of sensor nodes’ behaviors, such as hardware behavior, but are relatively slower to simulate. TOSSIM nodes do not simulate many details of the sensor nodes but are simulated much faster. The mix of cycle-level simulation with event-based simulation ensures that SUNSHINE can leverage the fidelity of cycle-accurate simulation, while still benefiting from the scalability of event-driven simulation.
The simulation process in SUNSHINE is illustrated by Figure 1. Firstly, for co-sim nodes that emulate real sensor motes, executable binaries are compiled from TinyOS applications using nesC compiler (ncc) and executed directly over these co-sim nodes. This is because co-sim nodes emulate hardware platform at cycle level. Therefore, TinyOS executable binaries can be interpreted by SimulAVR, the AVR simulation component of SUNSHINE, instruction-by-instruction. At the same time, GEZEL interprets the sensor node’s hardware architecture description, and simulates the AVR microcontroller’s interactions with other hardware modules at every clock cycle. One of the hardware modules that GEZEL simulates is the radio chip module. This radio chip module provides an interface to TOSSIM, which models the wireless communication channels. Through these wireless channels, co-sim nodes interact with other sensor nodes, which are simulated either as co-sim nodes by GEZEL and SimulAVR, or as functional-level nodes by TOSSIM. To maintain the correct causal relationship, the interactions between TOSSIM nodes and co-sim nodes are based on the timing synchronization and cross-domain data exchange techniques, which will be introduced in the following sections.

III. CROSS-DOMAIN INTERFACE

In this section, we will discuss how we interface the three components of SUNSHINE, each working in a different domain of simulation.

A. Integrate SimulAVR with GEZEL

GEZEL is able to provide co-simulation interfaces with ARM cores, 8051 microcontrollers, and PicoBlaze processor cores, to form a hardware-software emulator.

In SUNSHINE, in order to let the simulated AVR microcontroller exchange data with the simulated hardware modules, GEZEL creates cycle accurate hardware-software co-simulation interfaces according to the AVR microcontroller’s datasheet [10]. Once these interfaces are established, data can be changed between GEZEL-simulated hardware entities and SimulAVR-simulated microcontroller. The details of the co-simulation interfaces can be found in GEZEL’s documentation [11]. With the support of GEZEL’s co-simulation interfaces, SUNSHINE is able to form an emulator to capture the sensor nodes’ hardware-software interactions and performance. In the following, we call this resulting emulator as P-sim, which combines the software domain simulator SimulAVR and the hardware domain simulator GEZEL.

B. Timing Synchronization

SUNSHINE integrates network simulator TOSSIM and hardware-software emulator P-sim for the purpose of scalability. However, simulations in these three domains run at different step sizes. Without proper synchronization, we can easily get mismatches in simulation time between event-driven simulation and cycle-level simulation as shown by Figure 2. The wall clock time is the time required by the simulator to complete a simulation, i.e., the simulator’s run time. The simulation time is the simulator’s prediction of the execution time of a sensornet application based on the simulation of the sensornet. As shown in Figure 2, P-sim runs at cycle-level steps, where each simulation step captures the behaviors of an AVR microcontroller or a hardware component at one clock cycle. Therefore, the simulation time is gradually increasing. However, in TOSSIM, a discrete event simulator, each simulation step captures the occurrence and handling of a network event. As the time durations between events are irregular, the simulation time in TOSSIM also increases at irregular steps. This difference in simulation time may cause potential violations in causal relationship among different sensor nodes in simulation.

To solve this issue, SUNSHINE includes a time synchronization scheme as depicted in Figure 4. In the design, TOSSIM uses the Event Scheduler to handle all the network events while P-sim uses the Cycle-level Simulation Engine to control the simulation of hardware modules and the AVR microcontroller every clock cycle. All network events are in the Event Queue and are sorted according to their timestamps that record their occurrence time. The Event Scheduler processes the head-of-line (HOL) event in the Event Queue only when the Cycle-level Simulation Engine has progressed to the event’s timestamp. By selecting either an event or a cycle-level simulation to be simulated next, SUNSHINE will maintain the correct causality between different simulation schemes in the whole network.
nodes of the Active Node List. As a result, a node’s sleep or wake up state does not need to be checked every clock cycle. Given the fact that in sensornets, a sensor node spends most of its time in sleep mode, this design will greatly accelerate SUNSHINE’s simulation speed.

Based on the synchronization scheme, the desired behavior of a synchronized simulation can be achieved as shown in Figure 3. Events in the network domain are processed with the correct causal order compared to the cycle-level simulation, and the SUNSHINE simulator correctly interleaves cycle-level processing with event-driven processing.

C. Cross-Domain Data Exchange

Since SUNSHINE integrates simulation engines working in three different domains, it is necessary to implement interfaces for cross-domain data exchange between these simulators. The data exchange between SimulAVR and GEZEL is explained in Section III-A. In this section, we focus on discussing how data exchanges between hardware-software emulator P-sim with event-based simulator TOSSIM.

1) Noise Models: A wireless network simulator needs to build radio and noise models to simulate wireless packet delivery. Since SUNSHINE integrates P-sim with network simulator TOSSIM, it is convenient to adopt TOSSIM’s radio model to simulate wireless packet transmission and reception. TOSSIM also uses the closest-fit pattern matching (CPM) noise model to simulate whether the packets can be successfully received from the channel.

Since TOSSIM simulates high functional level network behavior, if there is a collision of the packets in the channel (i.e., two nodes send packets to the third node at the same time), TOSSIM simply assumes that the packets are corrupted and drops the packets. This is different from the real radio chip. In reality, the radio chip performs Frame Check Sequence (FCS) scheme to check whether the packet is received correctly and marks its CRC bit accordingly.

To simulate the radio chip’s real performance in SUNSHINE, the CPM model is modified by adding a receive FIFO (RXFIFO) to the radio chip module to store the received packets. In the simulation, when the CPM model determines a node successfully receives a packet, the received packet is stored in the RXFIFO with CRC bit set to 1 to demonstrate that the packet is received successfully without error. However, if the CPM model determines a node receives a corrupted packet, the RXFIFO stores the received data with CRC bit set to 0 to mention that the data is not received correctly. This process is in accordance with the real radio chip’s behavior.

2) Event Converter: Sensor nodes in network domain simulated by TOSSIM need to exchange messages with nodes in software-hardware domain simulated by P-sim through the TOSSIM simulated channel. However, network domain simulation and hardware-software domain emulation have different simulation abstractions. For TOSSIM, it abstracts the functions and interactions among network components as high-level abstracted events. For example, as shown in Figure 5, the transmission or reception of an entire packet is regarded as a single event in TOSSIM. In hardware-software domain emulation, the packet transmission and reception related functions and interactions among hardware modules are simulated as series of behaviors in many cycles. For example, to simulate the reception of a packet, the bits received and read from the radio chip module should be simulated at each clock cycle. Therefore, a time converter is needed to bridge this gap in time granularity.

Another issue is the message format defined in TOSSIM is different from the message format in the real mote according to the radio chip’s datasheet. Therefore, a package converter is built to facilitate the conversion of packets between TOSSIM and P-sim.

Figure 6 illustrates the event conversion process. If a co-sim node transmits a data packet, it should follow several steps in simulation. The simulated AVR microcontroller first sends the packet to the radio chip module at cycle level. The radio chip module stores the packet in a transmit FIFO (TXFIFO). As soon as the radio chip module receives a send command from the simulated microcontroller, the time converter transforms P-sim’s simulation time to TOSSIM’s simulation time, while the packet converter changes the real mote’s packet format to TOSSIM’s packet format, and sends the packet to the TOSSIM simulated channel. Based on this scheme, both TOSSIM nodes and co-sim nodes in the receiver side are able to receive the packets from the sender.
for the simulated AVR microcontroller to read data from the RXFIFO according to the datasheet [14].

Using the event converter, SUNSHINE is able to convert coarse packet communication events to the cycle-level packet reception and transmission behaviors and vice versa. Based on this mechanism, SUNSHINE satisfies both P-sim’s cycle-level and TOSSIM’s event-level requirements.

IV. HARDWARE SIMULATION SUPPORT

As SUNSHINE is able to simulate hardware behavior, in this section, we discuss SUNSHINE’s support for hardware simulation.

A. Hardware Specification Scheme

One of the primary contributions of SUNSHINE is to support hardware flexibility and extensibility. SUNSHINE describes sensor motes architecture at simulation’s configuration level using GEZEL-based hardware specification files. Users of SUNSHINE can make various modifications to sensor motes architecture, such as using different microcontrollers, adopting multiple microcontrollers, adding hardware coprocessors, connecting with new peripherals and performing other customizations on the platform. The syntax of a valid hardware specification file based on GEZEL is relatively simple. Users are able to write their own specification files according to GEZEL semantics [15].

To demonstrate this point, Figure 7 shows specific details of how hardware architecture of a MICAz mote is described in SUNSHINE. We listed a snippet of the hardware specification file in Figure 7. The file is divided into three pieces, each of which is dedicated to a relevant hardware part. From the code snippet, we would see that users could pick hardware components using “iptype” statements to configure a sensor node’s hardware platform. In this specific example, microcontroller Atmega128L and radio chip CC2420 are chosen to form the MICAz hardware platform. The components’ corresponding ports are interconnected through virtual wires that are also described in the specification file. For example, “Atm128sSinkpin” wires the input pin B3 of the AVR microcontroller’s core to the output pin MISO of the CC2420 chip, while “Atm128Sourcepin” wires the output pin B0 of the AVR microcontroller’s core to the SS input pin of the CC2420 chip. While our example shows the MICAz platform, a user can also pick other components to form a different hardware platform in their sensornet simulation. For example, one can use ARM or 8051 microcontroller instead of Atmega128L by modifying the hardware specification file. Based on this mechanism, SUNSHINE can easily combine different hardware components to form different hardware platforms for sensornet simulation. In other words, SUNSHINE supports running network simulation over flexible hardware platforms that are created based on either commercial off-the-shelf sensor boards or the user’s customized platform designs.

The example in Figure 7 also shows how SUNSHINE enables different co-sim nodes to run different software applications through the use of “ipparm” statements. The “ipparm” can also be used to set parameters for hardware components. In Figure 7, the statement “ipparm “exec = app”” means the simulated AVR microcontroller would interpret the executable binary named “app”, which is compiled from a software application using ncc compiler. Users can also configure the simulated AVR microcontroller to execute other binaries in a co-sim node through ipparm statements. By configuring different co-sim nodes to execute different software binaries, SUNSHINE can simulate a sensornet that has multiple different applications. This is a significant improvement over TOSSIM, which can only run one application in a whole network. Essentially, SUNSHINE’s simulation configuration steps are as follows. First, the executable binaries of applications are compiled from their source codes. Then, as shown in Figure 7, a Hardware Specification file is created to describe how hardware components form the hardware platforms in the sensornet. The Hardware Specification file also links the generated executable binaries to the corresponding hardware platforms. After the configuration, SUNSHINE simulation can start.

From the above description, one would see that SUNSHINE can be used to simulate various hardware platform designs to find the most suitable hardware module for a given network environment and a given set of application requirements. Therefore, SUNSHINE is an efficient tool to help hardware designers develop better sensor motes. In addition, researchers in the field of software can also use SUNSHINE to easily configure novel hardware architectures and then evaluate their sensornet applications and protocols over these customized architectures. Because SUNSHINE can change hardware components easily at simulation’s configuration level, even software researchers with little hardware knowledge can configure sensornet hardware platforms themselves.

B. Hardware Behavior

Unlike other sensornet simulators, SUNSHINE is able to accurately capture sensor nodes’ hardware behaviors. Users are able to know whether the microcontroller is in sleep mode...
or active mode as well as identify the radio chip’s current radio control state. In addition, through interpreting GEZEL code, a hardware description language, SUNSHINE is able to display cycle-level behavior of hardware components when applications are running on co-sim nodes. This would help hardware designers know how hardware module behaves in sensornet applications.

![Fig. 8. Traces for TinyOS Reception application](image)

For example, users can track hardware pins’ activities when running a sensornet application on a co-sim node in SUNSHINE by doing the following. The signal tracing mechanism of SUNSHINE records stimuli files when the simulation is set in debug mode. These stimuli files, named Value-Change Dump (VCD) files, can be read by digital waveform viewing tools, such as GTKWave, to produce graphic illustrations of hardware pins’ values. An experiment is provided to show SUNSHINE’s capability of capturing the sensor nodes’ hardware performance. In the experiment, a TinyOS TxThroughput application runs on one co-sim node and the Reception application runs on the other co-sim node. In the TxThroughput application, the sensor node keeps sending packets to the radio channel using the largest message payload size. In the Reception application, the node listens to the channel and receives packets from the channel. Figure 8 shows the hardware pins’ detailed activities of the receiving node. Through these traces, users are able to detect how the AVR microcontroller interacts with the CC2420 radio chip.

V. EVALUATION OF SUNSHINE

We performed the experiments on a Dell laptop that has Intel (R) Core (TM) 2 Duo CPU T5750 @ 2.00GHz, 3G RAM and runs Linux 2.6.32-23-generics. SUNSHINE integrates TinyOS version 2.1.1, SimulAVR and GEZEL version 2.5. We used the latest version of the simulators available at the time of performing the experiments. The hardware platform configured in these simulations is MICAz.

A. Scability

In the following, we simulated several applications to analyze SUNSHINE’s scability. In the first application, we varied the number of nodes that are randomly distributed in a fixed area from 2 to 128. Nodes are paired to communicate with each other. We wrote an application to let the paired nodes send packets between each other. The simulation ends when all of the nodes receive one message from its neighbor. We considered four cases: the first case is pure co-sim nodes network, the second one is pure TOSSIM nodes network, the third is the combination of 50% co-sim nodes with 50% TOSSIM nodes network, and the fourth is 25% co-sim nodes with 75% TOSSIM nodes network.

Figure 9 shows SUNSHINE’s wall clock time which represents the time required by SUNSHINE to complete the simulation. As expected, pure TOSSIM simulation outperforms SUNSHINE in terms of simulation speed by abstracting away the detailed behaviors of sensor nodes, such as hardware clock cycles and microprocessor’s instructions. On the other hand, SUNSHINE’s low execution speed comes from its fine-grained simulation accuracy. However, Figure 9 shows that SUNSHINE has the ability of simulating hybrid network consists of co-sim nodes and TOSSIM nodes. When simulating the mixed network, SUNSHINE’s execution speed is accelerated and hence can be suitable for even large networks.

Figure 10 shows the memory utilization of the simulation. The simulation with 100% co-sim nodes utilizes large CPU memory because cycle-level simulation needs to cache a lot of co-sim nodes’ data and states from GEZEL, simulAVR and TOSSIM. These data and states can take a large amount of memory space when simulating a large network. To reduce the memory consumption, SUNSHINE can combine TOSSIM nodes with co-sim nodes to decrease the memory utilization.

Given this information, combining co-sim nodes with TOSSIM nodes becomes an advantage of both speeding up the simulator’s run time and decreasing memory usage. Also, this combination is acceptable since in most network scenarios, only important nodes need to be simulated at cycle-level fine granularity (i.e. simulated as co-sim nodes) to evaluate their hardware and software performance. Other nodes, whose detailed behaviors are not important, can be simulated in TOSSIM. Several specific examples are given as follows.

1) Ring Network: We simulated a packet relaying application based on a 320 nodes’ ring network. In the packet relaying application, the first node sends a packet with two bytes payload length to the next hop. As soon as the second node receives the packet from the previous one, it forwards the same packet to the next node. The application ends when the first node receives the two bytes packet from its previous node. In this case, most of the sensor nodes have the same behaviors (e.g. receiving and forwarding the data to another node). Since co-sim nodes are used to analyze sensor nodes’ cycle level software-hardware performance, only simulating a few co-sim nodes are sufficient to analyze the network behavior. In this experiment, we used 5% co-sim nodes and 95% TOSSIM nodes to consist the network. We randomly chose co-sim nodes’ positions in order to show the interconnection between TOSSIM and co-sim nodes. We simulated the application ten times with different co-sim nodes’ positions and calculated the average of the simulator’s run time. In the experiment, simulating 320 nodes only takes 217.35s. Using ring network avoids packets collisions in the channel. Dense networks are deployed to illustrate SUNSHINE’s performance in the following experiments.
2) **Star Network:** A nine nodes’ star network is simulated in SUNSHINE. The network topology is shown in Figure 12(a), which includes one base station placed at the center, that receives data from other nodes, and eight normal sensors, that take turns to send one packet to the base station. The simulation ends when the base station receives all the leaf nodes’ packets. In this application, to analyze fined-grained network behavior, we only need to simulate the base station and one leaf node as co-sim nodes, while other leaf nodes can be set to TOSSIM nodes. SUNSHINE finishes simulation in 3.71s using two co-sim nodes, compared to 19.75s run time using all (nine in this case) co-sim nodes.

3) **Tree Network:** A three-layered tree network is considered as shown in Figure 12(b). Nodes 1 to 12 send packets to their parent nodes, 13 to 15, respectively. After receiving the packets from all their children nodes, nodes 13 to node 15 first perform several computational tasks (e.g. compressing the data received from its children nodes) and then send the packets to the root node 16. As soon as node 16 receives the packets from nodes 13 to 15, simulation ends. Since in a real sensor network, the bottleneck node is highly likely to be node 16, to investigate the bottleneck node’s behavior under heavy load, it is reasonable to simulate the root node 16 as co-sim node. In addition, several nodes that perform computational tasks and can become overloaded, such as nodes 13 to 15, can also be considered as co-sim nodes. In this experiment, simulating four co-sim nodes (nodes 13 to 16) with 12 TOSSIM nodes (nodes 1 to 12) takes 159.00s. However, using the root node 16 as co-sim node while others (nodes 1 to 15) are TOSSIM nodes only takes 24.64s.

According to the above experiments, we can draw a conclusion that SUNSHINE is able to capture sensor nodes’ cycle accurate hardware-software performance while keep the simulator’s execution speed fast by mixing co-sim nodes with TOSSIM nodes in the network simulation. Therefore, users should choose important nodes as co-sim nodes running at cycle level, while other nodes as TOSSIM nodes to ensure SUNSHINE’s simulation can scale to large networks.

### B. Simulation Fidelity

In this section, we conducted two real-mote experiments on Crossbow MICAz OEM reference boards to show the simulation fidelity of SUNSHINE. Each result is the average value of ten experiment runs.

In the first experiment as shown in Figure 13(a), we deployed a five node sensor network to analyze SUNSHINE’s channel performance. Since SUNSHINE utilizes the TOSSIM’s radio and noise models which have been validated in [1], [12], in this experiment, it is sufficient to consider a simple ring network topology with a focus on packet relaying applications (that are introduced in Section V-A1).

As measured in real motes, the average network throughput is 1568.6 b/s, which is close to that of using SUNSHINE, i.e., 1699.3 b/s (both values are measured without ack). As can be inferred from the results, SUNSHINE is able to provide fairly reliable results as reference for the sensor network applications.

In the second experiment, we evaluated SUNSHINE’s capability of executing computational tasks. On the testbed as shown in Figure 13(b), we ran the TinyOS TxThroughput application (mentioned in Section IV-B). The sensor node executes a dummy computational task of multiple empty loops before sending packets to other nodes, and we varied the number of empty loops to represent various levels of
computation intensity. We compared SUNSHINE, TOSSIM and the real mote in terms of the task execution time in simulation/experiment, and the results are shown in Figure 11.

From the results, we are able to observe that (1) TOSSIM runs fastest as expected, and its predicted task execution time is much less than the real task execution time; and (2) SUNSHINE is able to provide a simulated task execution time that coincides with that of the real mote experiment. TOSSIM’s fast simulation speed is attributed to its inability of capturing the task execution time on the microcontroller, which will apparently limit its applicability time-sensitive applications/protocols. Many security protocols, such as the distance-bounding protocol [16], require precise time-out behavior to thwart physical man-in-the-middle attacks. When testing and verifying these protocols, SUNSHINE will out-compete TOSSIM since SUNSHINE is able to correctly capture the impact of computation intensity on sensornet performance.

VI. COMPARISONS OF SUNSHINE WITH EXISTING SIMULATORS

Due to the difficulties in setting up sensor network testbeds, sensornet researchers prefer to simulate and validate their applications and protocols before experimenting in real networks. This makes sensornet simulators an important tool in sensornet research. A number of wireless network simulators have been proposed including TOSSIM [1], ATEMU [5], Avrora [6], and NS-2 [17] to name a few. In this section, we made several comparisons between SUNSHINE and other existing network simulators.

A. Existing event-based network simulators

NS-2 [17] is the classical network simulation framework that is used in the context of wired and wireless networks. NS-2 is a discrete event-based simulator that simulates networks at packet level. It is widely used in wireless network area to evaluate lower layer communication algorithms. Even though NS-2 is a useful network simulation framework, it is not suitable for wireless sensor networks for several reasons.

Firstly, NS-2 lacks an appropriate radio module that fits for sensor networks. In addition, NS-2 is focused on evaluating network protocols using general models, such as routing model, mobility model and MAC layer model, etc. It fails to model application behaviors which can have a greater impact on sensor’s performance and life estimation.

Similar to NS-2, TOSSIM is also an event-based network simulator. TOSSIM is a widely used simulator in sensornet research community due to its higher scalability and more accurate representation of sensornet compared to NS-2 [1]. However, TOSSIM simulates wireless nodes at functional level and does not provide enough details at cycle-level. Therefore, TOSSIM cannot capture and compare the performance of various hardware designs and the software implementations of sensornet applications.

B. Existing cycle-level sensornet simulators

ATEMU [5] and Avrora [6] are the existing sensornet simulators that venture out of the event-based simulations in network domain. They provide cycle-accurate software domain simulation to evaluate the fine-grained behaviors of software over AVR microcontrollers of MICA2 sensor nodes.

Though ATEMU and Avrora are cycle-level sensornet simulators, they can only simulate MICA2 sensor motes. They cannot accurately capture the impact of alternative hardware designs on the performance of sensornet applications. In other words, they do not support flexibility and extensibility in hardware beyond very simple parameter settings.

C. SUNSHINE

SUNSHINE provides hardware flexibility where a user can make changes in hardware design of sensor platforms and verify his/her sensornet application’s feasibility. SUNSHINE is able to simulate different potential hardware architectures. For example, SUNSHINE can simulate a sensor board with an FPGA to handle heavy computational intensive tasks, such as advanced data packets encryption/decryption and data packets compression. This provides a new direction to sensornet design and enables network researchers to evaluate their designs under different hardware platforms. SUNSHINE provides a valuable instrument to both sensornet community and hardware development community.

Further, each existing simulator can only work in one domain. For example, NS2 and TOSSIM only work in event-based network simulation domain while ATEMU and Avrora can only execute cycle-accurate simulations. Different from these existing simulators, SUNSHINE offers its user flexible middle ground between cycle-accurate and event-based simulations. It can combine a variety of nodes that simulated at coarse event-level and nodes that are simulated at fine cycle-level.

Table I summarizes the differences between TOSSIM, Avrora, ATEMU and SUNSHINE. As shown in Table I, hardware flexibility is one of the most significant advantages of SUNSHINE. Also, SUNSHINE’s ability of capturing hardware behavior is another improvement for sensornet simulators.

VII. FUTURE WORK

In this paper, we focus on simulating the network, software and hardware behavior of MICAz mote. Based on the
characteristics of GEZEL, different instruction-set simulators are able to be connected with GEZEL to emulate the microcontrollers’ hardware and software performance. We have already interfaced GEZEL with ARM, 8051 microcontroller, etc. GEZEL is also able to connect to other instruction-set simulators, for example, MSPsim [18], an simulator for MSP430, to emulate MSP430 microcontroller’s hardware and software performance. In other words, SUNSHINE would be able to capture other existing sensor motes’ (for example, TelosB) hardware and software behaviors in a networked context.

Currently, the CC2420 radio chip module is built for SUNSHINE’s simulation. Constructing different radio chip modules, for example, CC1000, is also practicable. Then, MICA2 mote (Atmega128L microcontroller with CC1000 radio chip) is able to be emulated using SUNSHINE.

The radio channel and noise model used in SUNSHINE follows TOSSIM’s scheme. In other words, receiving or dropping packets in the radio channel is according to TOSSIM’s decision. Improving TOSSIM radio and noise models are feasible. Several works are provided as references [12], [19], [20].

Though SUNSHINE’s slow run time is reasonable, because it needs to simulate sensor node’s hardware and software behavior at every clock cycle, it is still possible to increase SUNSHINE’s execution speed. One way is to change the simulator design from single process to multi-process to leverage multi-core capability of modern computers.

We are considering transiting SUNSHINE simulation models into real sensor prototypes. To do so, we will create a design flow to map simulations of software and hardware designs to real software and hardware implementations. SUNSHINE supports simulation reusability since it can map simulations of software and hardware designs to real software and hardware implementations on sensor prototypes. This can reduce the development time for sensor networks. We consider developing a reconfigurable hardware platform based on low-power FPGAs to demonstrate this design flow. This hardware platform can also serve directly as a novel flexible platform that enables the sensor network community to quickly prototype various sensor architectures.

VIII. CONCLUSION

In this article, we have presented SUNSHINE, a novel simulator for the design, development and implementation of wireless sensor network applications. SUNSHINE is realized by the integration of a network-oriented simulation engine, an instruction-set simulator and a hardware domain simulation engine. By the seamless integration of the simulators in different domains, the performance of network protocols and software applications under realistic hardware constraints and network settings can be captured by SUNSHINE with network-event, instruction-level, and cycle-level accuracy. SUNSHINE outperforms other existing sensornet simulators because it can support user-defined sensor platform architecture, which is a significant improvement for sensornet simulators. SUNSHINE can also capture hardware behavior which is the unique feature of sensornet simulators. SUNSHINE serves as an efficient tool for both software and hardware researchers to design sensor platform architectures as well as develop sensornet applications.

ACKNOWLEDGMENT

This work is supported by National Science Foundation award CCF-0916763. We thank the anonymous reviewers for their suggestions on revising the paper.

REFERENCES