A Unified Network and Node Level Simulation Framework for Wireless Sensor Networks

Heemin Park, Weiping Liao, King Ho Tam, Mani B. Srivastava and Lei He

University of California, Los Angeles
Electrical Engineering Department
420 Westwood Plaza
Los Angeles, CA 90095
{hmpark, wliao, ktam, mbs, lhe}@ee.ucla.edu

ABSTRACT

Low power consumption is an essential requirement for wireless sensor networks. In many situations un-tethered wireless sensor nodes are expected to operate on non-renewable batteries for extended periods of time. To survive in such regimes, the embedded software running on the sensor nodes should strive to optimize power consumption at the node and network levels. In this paper, we present a novel simulation framework for quantifying power consumption in a unified way that also reflects the node level performance to network-wide power estimation. At the node level, our framework combines power simulator for the StrongARM processor with a radio and a sensing component. This is then integrated into SensorSim, an enhanced version of the ns-2 network simulator that also incorporates the sensing aspects required to simulate realistic sensor network scenarios. We illustrate the capabilities of our simulation framework to explore different power management schemes and interactions across the node level network layer and sensor field events using a representative example scenario.

1. INTRODUCTION

Embedded systems have networking capability to communicate wirelessly and in order to monitor physical phenomenon it is common to equip various sensors such as seismic, light, temperature sensors or accelerometers. Such networked and embedded systems with sensing capability are called *Wireless Sensor Networks*. Sensor networks can be used widely for monitoring physical phenomenon, like fire detection, target detection and tracking, traffic monitoring, wildlife habitat monitoring, etc. A sensor network comprises of a large number of sensor nodes, which are equipped with CPU for computing, radio for communicating wirelessly, and various sensors for capturing physical phenomenon which interests. In Figure 1, an example of wireless sensor networks and sensor node are illustrated. In this example, the role of the sensor network is to monitor and track the movement of the truck. As shown in Figure 1, the wireless sensor networks can be viewed in two levels. One is network level of which interests are connectivity, routing, channel characteristics, protocols, etc. The other is node level which consists of hardware and software components, radio, CPU, sensors, embedded software and limited energy.

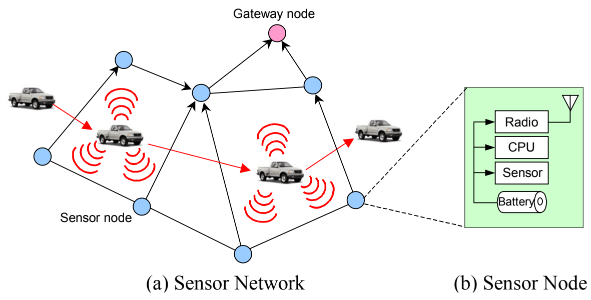


Figure 1: An Example of Sensor Network (a) and Sensor Node (b)

Since the sensor nodes are likely to be deployed in ad-hoc fashion (i.e. air-dropped over an area), most sensor nodes can only afford a battery as their energy source. Sensor networks are often designed to monitor physical phenomenon over long period time, e.g. from six months to one year. Thus, it is essential for the sensor network to have long lifetime. Therefore, the low power consumption is emerging as the most important factor in designing the wireless sensor networks. If one of the sensor nodes dies, then the entire sensor network might not perform the function perfectly. The lifetime of a sensor network can be defined as the lifetime of the node which dies first after depleting its battery completely. The main sources of energy consumptions are CPU, radio and sensor. To optimize overall energy consumption, it is desirable to optimize the energy consumption in these three components. It is known that transmitting or receiving a bit is much expensive than processing a bit in local CPU [14]. The energy by radio communication can be greatly reduced by in-network data processing. This is done by compressing data to be transmitted by processing them in local CPU rather than transmit the raw data to the destination node. Thus, the role of embedded software in a node has become more important not only for computing and forwarding the data, but also for reducing overall energy consumption in a network. This leads to the need to calculate the software power precisely at the node level. On the other hand, to precisely calculate the energy consumption for embedded software and radio communication at network level, it is desirable to make the embedded software fully functional at network level as well as at node level. Most current network level simulation tools do not have details of the node level architecture and power profiling. Therefore, with current available network simulators and node level power simulators, it is hard to optimize the overall energy consumption for the whole network because there has been no available framework which has the capability to explore the interactions between network and node level performance and sensor signal events. In this paper, we present a novel simulation framework to address this problem.

Our work is motivated by two observations. First, although a number of power simulators have been developed [3, 11, 17, 19], they can only simulate software power merely inside processors. Embedded systems often have special hardware requirements such as sensors and radio. Therefore, in order to calculate power consumption in wireless networks accurately, the power simulator should take into account the power consumption in radio and sensor by making them fully functional in the simulator. Moreover, communication is known to be much more expensive than computation [14]. The number of communications between nodes can be calculated using network simulators [5, 8]. To calculate the energy consumption with this information, the cost of using radio receiver and transmitter need to be prepared beforehand. Likewise, the techniques to reduce energy consumption in wireless communication and embedded systems are implemented as a form of embedded software in individual node. Dynamic voltage scaling, dynamic modulation scaling [13, 16], dynamic shutdown and wakeup method, energy-aware MAC protocol and topology control can be the example of the techniques. Pure network simulators which neglect all these node-level information are less useful in optimization of the embedded software. Although there are some network simulators has the special requirements of sensor networks [9, 10, 15, 18], none of them can provide a detailed perspective on the energy consumption of embedded software and peripherals like radios and sensors. Bakshi [1, 2] proposed to use node level energy profiling results in network level simulation. However, with [1, 2], the simulation would be done separately, which means the network level simulation would be able to be performed only after running node level simulation. Hence, with [1, 2] network and node level perspectives cannot be obtained at the same time. Among prior work, the most similar one with ours is [4, 7] by Drago et al. They integrated two simulation environments, SystemC and ns-2 by using a share memory queue to provide an extension to detail simulation of hardware for network simulator, and vice versa. Although SystemC is a design environment of hardware and software, it is hard to be used in profiling realistic power consumption for specific platforms such as StrongARM which is popularly used in wireless embedded systems. That is because the hardware is described in behavioral level or RT-level in SystemC. It results in very slow simulation of complex embedded software. Therefore, [4, 7] is not suitable to be used in optimizing power consumption of embedded software of wireless sensor networks.

We have developed a novel simulation framework for calculating energy consumption for wireless networks, which has the capability to simulate network level and node level at the same time. With our framework, the interaction between network level issues and node level issues can be observed. The network protocols stacks such as routing protocol, MAC protocol and noise, error and channel model would be network level issues. Likewise, the software implementation of those network protocols and operating system point of view like tasking and scheduling are node level issues. Users can explore the various alternatives for network stack and the embedded software in unified way. Hence, users can optimize the embedded software as well as network stacks.

Moreover, since our unified simulation framework has the capability of simulating the node level embedded software, users can deploy the embedded software used in unified simulation directly to the real sensor node. To facilitate this direct deployment and ease of use, we provide a set of API.

The remainder of this paper is organized as follows. In the next section, we present overall idea and structure of our unified simulation framework. Section 3 presents the details of our node level simulator, Embedded Systems Power Simulator. In section 4, we demonstrate the capabilities of our simulation framework and section 5 concludes this paper.

2. Unified Simulation Framework

Among currently available node level simulators [3, 11, 12, 17, 19] The EMSIM [12] is an energy simulation framework that simulates a simple embedded system featuring StrongARM microprocessor [6] and Linux OS. It basically includes a complete Linux system with support to necessary peripherals such as timers, serial UARTs, etc. It is able to provide detail power statistics for each hardware component as well as power profiling for each running task. The EMSIM, however, does not provide the important modules for embedded system researches, such as sensors and radio. In some sense, it is more like a general purpose simulator running on top of StrongARM processors and Linux OS. In order to have node level simulation feature for wireless sensor networks, we have extended such simulator to Embedded Systems Power Simulator (ESyPS) by integrating sensor and radio modules into EMSIM.

For the network level simulation capability, we use SensorSim [9]. SensorSim is an enhanced version of the ns-2 network simulator [8]. Because it incorporates the sensing aspects required to simulate realistic sensor network scenarios SensorSim is a suitable network level simulation tool for wireless sensor networks. It has the notion of sensor channel through which the sensing signal propagates, and the notion of target node which models the objects that are being tracked or monitored.

We integrate these two simulators, SensorSim and our node level simulator (ESyPS) for StrongARM processor, into one unified framework. The architecture of two types of sensor nodes in our simulation framework is shown in Figure 2. In our simulation framework, the sensor node is either one of the two types, SensorSim node or ESyPS node. The specific node which we are interested in detailed energy profiling for CPU, radio and sensor would be the ESyPS node. The SensorSim node has sensor stack as well as network stack. In SensorSim, there is sensor channel in which sensor signals, such as seismic signal, can propagate [9]. For the ESyPS node, the detailed node level power simulator will be executed for the embedded software. The basic idea of using ESyPS in conjunction with SensorSim is to connect ESyPS radio and sensor module with the radio and sensor physical layer through wireless and sensor channel. In integrating network level simulator (SensorSim) and our node level simulator (ESyPS), there are two main issues. Those are 1) time synchronization between two simulators and 2) Connecting radio and sensor module in ESyPS and the physical layer objects in SensorSim.

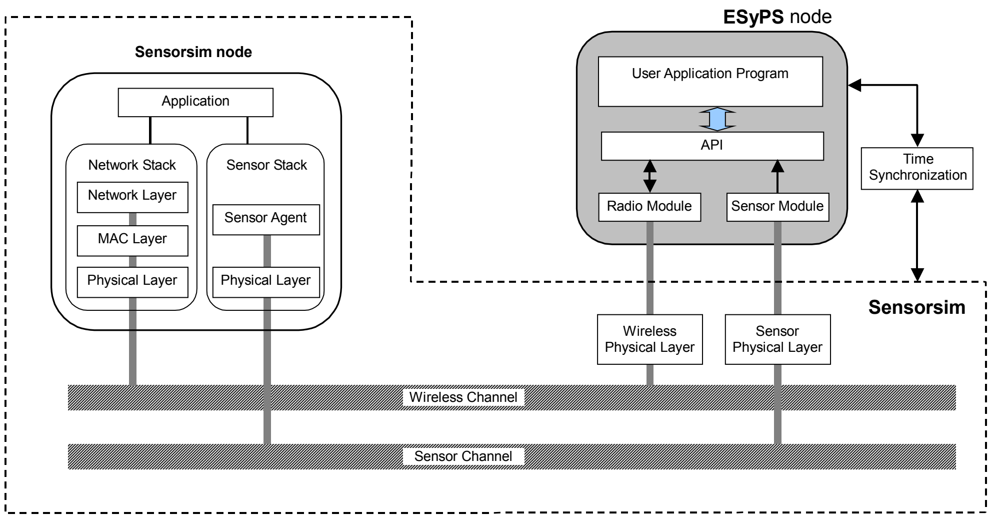


Figure 2: Node Architecture in Unified Simulation Framework.

In order for two simulators to operate consistently, the time in them should be synchronized to certain level of precision. We introduced a time synchronization module in between two simulators. The role of this module is to control the advance of ESyPS simulation steps with respect to the real time in SensorSim. ESyPS is an extension of ARMulator (the ARM processor simulator developed initially by ARM), which has only a notion of the number of clocks inside the simulator instead of real time. The time in ESyPS is obtained by multiplying the number of clocks elapsed in ESyPS by time per clock period. The time per clock can be obtained from the operating frequency of the processor (i.e. 206MHz for StrongARM). The pseudo code of the time synchronization algorithm is described in Figure 3.

```
timeSync_module begin
    if this is the first entry begin
        Execute ESyPS until Linux is booted
    end
    t_SensorSim is obtained from scheduler of SensorSim
    t_ESyPS = CLOCK_PERIOD * number of cycles after bootup
    while t_SensorSim > t_ESyPS begin
        Execute ESyPS for one cycle
    end
    Reschedule timeSync_module after SYNC_RESOLUTION
end
```

Figure 3: The Pseudo Code of the Time Synchronization Algorithm.

What the time synchronization module(*timeSync_module*) do are as follows. SensorSim starts going ahead of ESyPS by SYNC_RESOLUTION which the maximum allowed time difference between two simulators. Then, the ESyPS simulator catches Sensorsim until the times of two simulators become same. This process is periodically repeated at every SYNC_RESOLUTION. When *timeSync_module* is executed for the first time, the Linux OS bootup sequence is executed. The value of SYNC_RESOLUTION can be adjusted according to the requirement of application. For the demonstration purpose, we set this value to 0.0001 second in our example. However, time synchronization can be as precise as one clock period of the processor.

The other issue is to transfer sensor signal to ESyPS and to do wireless communication with ESyPS through wireless channel provided by SensorSim. In SensorSim node, if the wireless or sensor physical layer receives packets and signals through channel, then it sends the packets and signals to upper layer stack, such as MAC or sensor agent. For ESyPS node, we send up the packets and signals to ESyPS simulator after converting the packets from ns-2 format to ESyPS packet format. Then, in ESyPS simulator, the packets transferred to user application program through the Application Program Interface (API). The process for the user application program to transmit packets to SensorSim node can be done in a reverse way.

3. ESyPS:Embedded Systems Power Simulator

In this section, we present our Embedded System Power Simulator (ESyPS). The ESyPS is developed by integrating sensor and radio modules into EMSIM [12]. ESyPS has a fully functional sensor node in embedded systems with detection, processing and wireless communication capability. The ESyPS part consists of three parts: processing core, peripherals and sensing devices as shown in Figure 4. The processor core is a full model for ARM processor with functional units, caches, memory management unit (MMU) and read/write buffers. The peripherals include the timer, serial UARTs, interrupt controller and the main memory. The sensing devices are sensors and radio modules specially designed for embedded system sensor nodes.

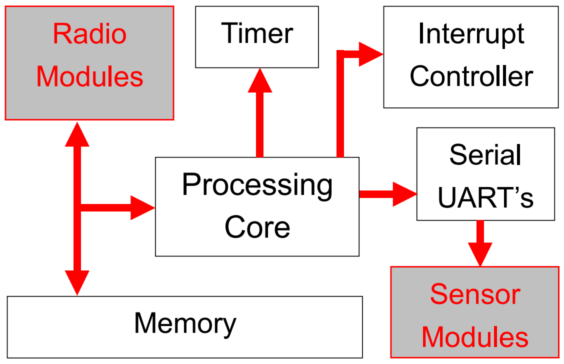


Figure 4: The Structure of Embedded System Power Simulator(ESyPS).

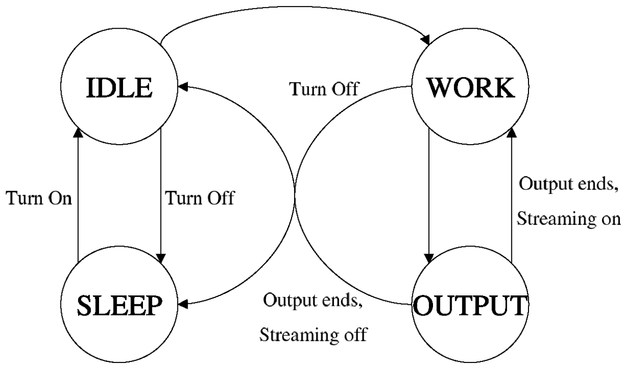


Figure 5: The State Machine of Sensors.

3.1 Sensor Modules

The sensor model consists of state machines and a collection of controlling functions. It models multiple sensors attached to a sensor manager, which in turn connects to the processor through the UART. Owing to the fact that the emulator has got only one UART port available (the other port is used for terminal service, e.g. printf on screen), the sensor manager, which is the only piece of hardware having a port, has to deal with all sensor requests delivery from the processor and data collection from sensors. The sensor model is constructed as one of the hardware components in the ARMulator which interacts with the system only through the serial port. User application program can control the sensor model by opening and writing to the serial port using the normal functions like 'open' and 'write'. The cross compiler will then translate this into writing to special memory addresses which correspond to ports 0x2f8. The virtual memory simulator, when receiving such address access requests, will call the UART handler. The original UART handler is used for modem emulated by network sockets. We have modified it so that it calls the simulation library functions of the sensor model.

Messages from the user program is packetized and sent serially to UART. Each message has a fixed length and a fixed format. User programs can control the sensors with the following parameters: 1) Turn On and Off, which corresponds to idle/reset and sleep modes. 2) Sense once and report after a delay. 3) Keep sensing and reporting periodically, with period specified. 4) Read sensor readings once the sensor manager gets one of these commands, it start parsing the packet and then dispatch a work request to change the state of the specified sensor. Sensor can be set either in WORK, IDLE, SLEEP or OUTPUT modes. A sensor state machine is shown in Figure 5.

On the configuration side, users can set the number of sensors attached to the system, their intrinsic minimum response time and their power values at their various states in the header file of the ARMulator sensor module. To estimate energy consumption, power values and time have to be known before the simulation is run. Power is set as described above. Time is computed in the simulator. The global system time notion is based on the number of cycles elapsed on the processor. With the processor frequency known as a priori (and in our system it is 206Mhz), the program can gives the energy and power values. SLEEP, IDLE and WORK states all have their power values independently configurable. At state changes, the total consumed energy value is updated according to the sensor state just before the state change and the time which the sensor has spent in such state. In order to facilitate users to program for using sensor modules, we provide APIs so that users can just call library functions to control sensors. Since multiple sensors are sharing one UART buffer, these library functions have to be run atomically to avoid multiple accesses to the UART buffer. All the functional calls are shown below, where id is the sensor id (as defined in the ARMulator sensor header file), t is the delay and d is the period. With API, users are freed from the details of having to make up packetized commands and un-packetize respond messages for sensor values.

- `void SetUpUART()`
- `void CloseSensor()`
- `void TurnOnSensor(int id)`
- `void TurnO_Sensor(int id)`
- `void Sense(int id, unsigned int t)`
- `void Streaming(int id, unsigned int d)`
- `int ReadSensor(int id)`

3.2 Radio Modules

We connect radio module to EMSIM using memory-mapped mechanism. Our embedded systems power simulator can afford several number of radio modules simultaneously to support various requirements of sensor nodes. Figure 6 shows the control flow for radio modules.

Each radio module has a 4KB buffer, which can be read or written by the processor directly via memory-mapped mechanism. In our system, we choose the main memory address between 0xdfff0000 and 0xdfff0000 for mapping. The memory-mapped mechanism is done by intercepting the memory accesses from the processor. We check all memory references before the virtual address is translated. If the memory addresses fall with the range for radio module, we redirect the memory accesses to radio module. No further memory translation is necessary. Each radio module has three power states: active, idle and sleep. The active power is dissipated when the radio module is transmitting or receiving data. The idle power is the power when the radio is turned on but doing nothing. The sleep power is dissipated when

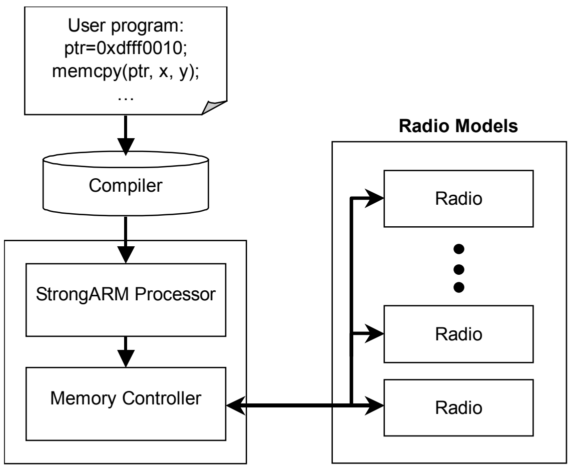


Figure 6: Radio Modules.

the radio is turned off. Currently we set the value for all three power states with the relation of $P_{active} : P_{idle} : P_{sleep} = 100 : 10 : 1$. These parameters can be adjusted according to user specific radio component. Another feature for the radio module is its modulation level. Instead of setting the absolute value for modulation level, we choose the relative modulation levels with the base modulation level equal to the capability to transmit 1 bit for every 100000 cycles. The base modulation level is equivalent to 1kbps at 100MHz processor. Other relative modulation levels are multiple of the base modulation level in the power of 2. For each modulation level M, the cycle number to transmit or receive N bit data is:

$$\text{Total Cycles} = \frac{100000}{M} \times N$$

With this modulation capability of our radio modules, the dynamic modulation scaling method [16] to reduce communication power can be easily implemented and verified. Similar to the sensor module, we develop APIs for radio. This includes turning on/off radio, set/get modulation level, transmit/receive radio, get current status of radio. The APIs for radio module are as follows.

- `int TurnOnRadio(int id)`
- `int TurnOffRadio(int id)`
- `int SetModLevel(int id, int mod)`
- `SendRadio(int id, int size, char *buffer)`
- `int GetRadio(int id, int size, char *buffer)`
- `int GetStatus(int id)`
- `int GetModLevel(int id)`

4. DEMONSTRATIONS

In this section, we demonstrate the capabilities of our unified simulation framework. We use a simple tracking scenario example, in which sensor nodes are randomly deployed and a target (i.e. truck) movements are injected along a path. Each sensor node will detect a seismic signal from the target using a seismic sensor. After collecting a predefined number of samples, the sensor node determines if the sensed signal is true or false by calculating the signal-to-noise ratio (SNR). The noise model is provided by network level simulator, SensorSim. Once processing the sensed signals, it sends the data to gateway node which has long-range radio or internet access to send the results to remote place. Every sensor node acts as router as well as collector of seismic signal. The example network is as follows. A 100 sensor nodes are deployed randomly onto 1000x1000 plane. The network topology is shown in Figure 7. The solid dots represent sensor nodes. The circled node is the gateway node and the boxed node is the node equipped with the node-level simulator (ESyPS). We select this node because this is located at about center of the network and the pathway to gateway node from the bottom side of the network. We will examine the node level energy consumption for the boxed node in detail. The line in the middle of the network is the trajectory of the target movements.

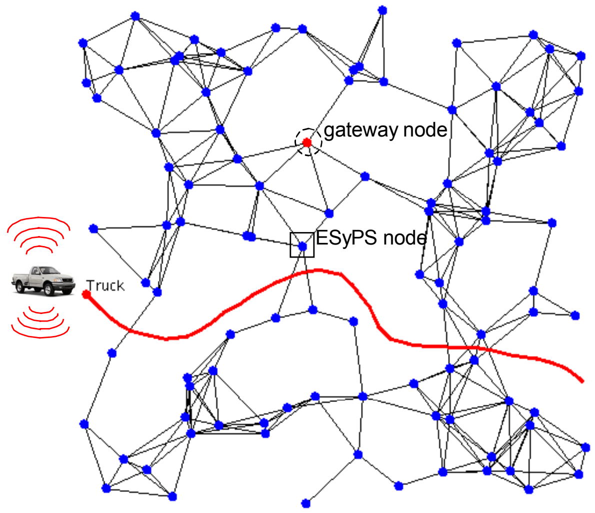


Figure 7: Test Topology with 100 nodes

The target will pass along the line at the speed of 40 per second. The link between nodes means those two nodes are within radio range. The radio range is 150 and sensing range is 100. The seismic sensor detects some signals if the target is within 100 from the sensor node. In this scenario, the seismic sensor run at a sample rate of 100Hz using timer interrupt. whenever each node collects 100 samples of seismic signal, it calculates the SNR. When the SNR is greater than a certain threshold, then the sensor node regards the the sensed signal being true and sends a signature size of which is 16 bits to gateway node. Otherwise, because the sensed signal needs further complex signal processing to determine its certainty, the node sends all the data collected (i.e. 100 samples) to gateway node.

We have implemented a lightweight protocol stack for SensorSim node and ESyPS node. TDMA based medium access control (MAC) protocol is used. A unique TDMA slot for each node is pre-calculated by our scenario generation tool and assigned to each node. The routing protocol used is distance vector routing. The scenario generation tool we are using also discovers the shortest hop distance paths from each node to gateway node and then assigns the next-hop-to-gateway node to each sensor node. When a node receives a packet, it will forward the packet to the next-hop-to-gateway node according to the distance vector assignments. We perform a simulation for this example scenario for 30 seconds. With our simulation framework, the overall

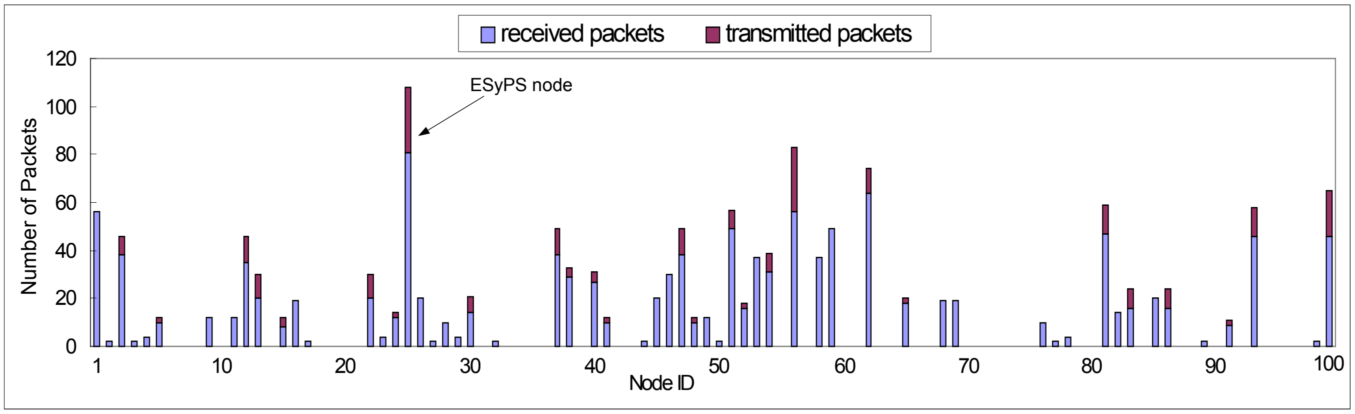
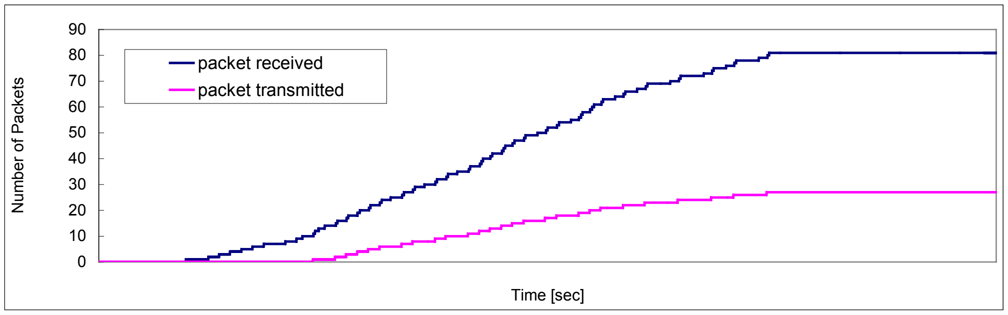
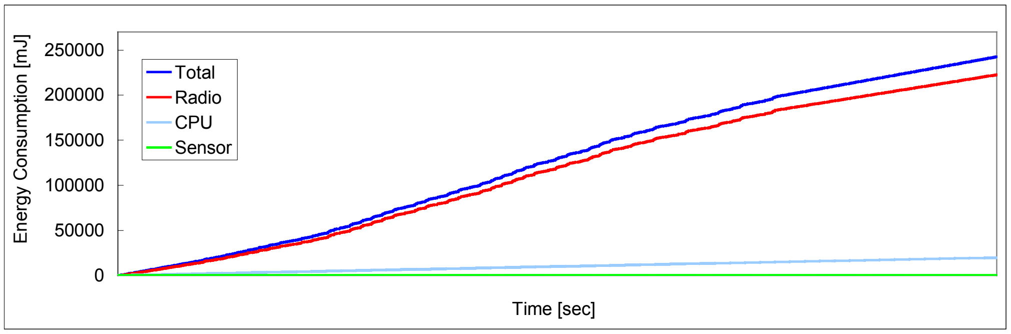


Figure 8: Network Traffic Statistics.



(a) Number of Packets Received and Transmitted.



(b) Cumulative Energy Consumption for Each Component

Figure 9: Node Level View of Simulation Results.

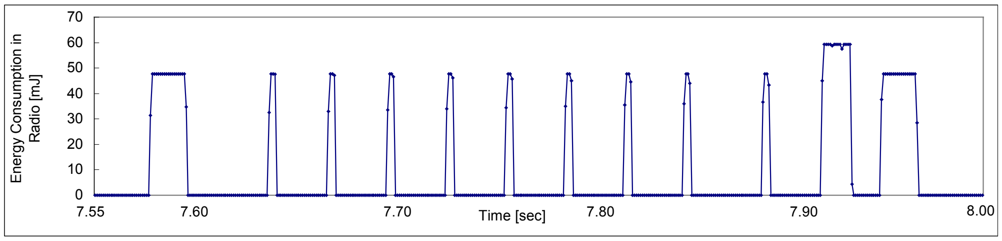
network behavior and the detail power profiling for the node which we are interested in can be observed.

Figure 8 shows the number of packets received and transmitted for each node. Since the node 1 is the gateway node, it does not have transmitted packets in this example. As Figure 8 verifies, the sensor node which is modeled in detail using ESyPS receives and transmits a large number of packets as it is on the pathway to the gateway node from the bottom part of the network.

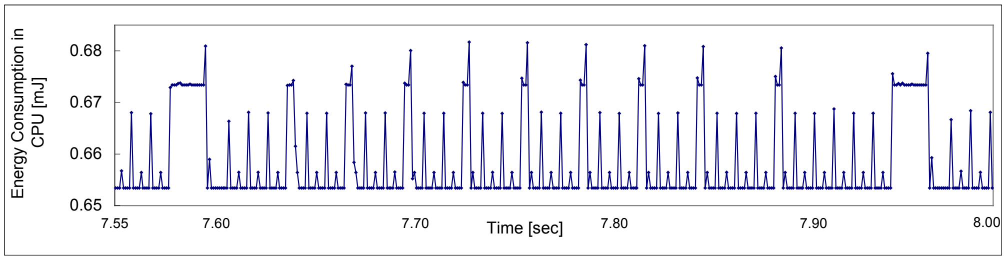
To examine the detail packet statistics and energy profiling for the node we are interest in, the received and transmitted packets for ESyPS node and the cumulative energy consumption for each components are demonstrated in Figure 9(a) and (b), respectively. As shown in Figure 9(a) and (b), energy is consumed according to the packets receiving and transmitting for each components, radio, CPU and sensor.

Let us zoom in the energy consumption to more detail. We have plotted graphs of the energy consumption of the ESyPS node from 7.55 second to 8.0 second. During this period, there are packet receptions in the ESyPS node at 7.57 and 7.94 second, and the ESyPS node forwards at 7.92 second the packet received at 7.57. The energy consumption of these detailed operation of radio, CPU as well as sensor are shown in Figure 10.

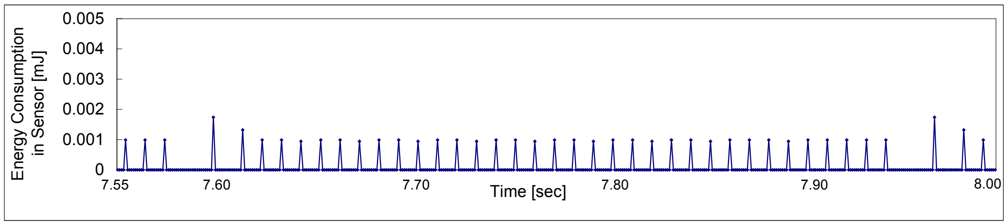
As shown in figure 10(c), since the sensor is always performing sensing at every 100Hz, it consumes similar energy whenever it senses over time. Meanwhile, the radio consumes lots of energy whenever it actively receives or transmits the packets.



(a) Energy Consumption in Radio



(b) Energy Consumption in CPU



(c) Energy Consumption in Sensor

Figure 10: Detail Energy Consumption Results.

We can determine that there are three communications as we can see in Figure 10(a). Energy consumption at other time is much less than reception or transmission time. This is because the node checks at every start time of TDMA slots if there is a packet by probing channel only for a short period of time. If there is no packet in wireless channel, the radio is going back to IDLE state.

The energy consumption in CPU shows a little bit different behavior. Whenever the node generates an interrupt signal and executes interrupt service route for sensing the seismic signal and checks if there is a packet in channel, it consumes energy for a short time. If there is a packet, then the node needs to process the packet to determine if destination of the packet is itself or if the packet requires forwarding. If it requires forwarding, the node needs to put the packet in outgoing queue. This is shown in Figure 10(b) at time around 7.57 and 7.94 second. However, the packet transmission to neighbors does not affect much in energy consumption of CPU. The transmission of packets has to be done in unique assigned slot. Thus, while transmitting packets, CPU does not need to participate. This only increases energy consumption in CPU slightly. This case happens at 7.92 second in Figure 10(a) and (b). As demonstrated in this section, with our unified simulation framework, the interactions between network and node level performance can be observed at the same time. This capability is essential in optimization of embedded software and overall energy consumption of the wireless networks.

5. CONCLUSIONS

In this paper, we present a unified simulation framework for network and node level which is suitable for energy profiling of wireless sensor networks. This framework is composed of two main parts. One is the Embedded Systems Power Simulator (ESyPS) which has been built on EMSIM. ESyPS has two important extensions, the sensor and the radio models. These models are designed to be highly configurable so that users can simply modify the hardware characteristics, and application program interface(API) is provided to facilitate for users to use the models. By integrating this ESyPS and SensorSim which has the capability of incorporating sensing aspect into network simulator, we have build a unified simulation framework which has multi-level perspective of network and node level at the same time. We demonstrate the capabilities of our simulation framework using a tracking scenario. In the demonstration, we show that the overall network issues and detailed node level energy profiling are available. With this unified simulation framework, it is possible to explore the interactions between network level and node level performance and interactions with sensor in-field events.

6. REFERENCES

[1] A. Agrawal, A. Bakshi, J. Davis, B. Eames, A. Ledeczi, S. Mohanty, V. Mathur, S. Neema, G. Nordstrom, V. Prasanna, C. Raghavendra,

and M. Singh, *MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems*, Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES 2001), Snowbird, Utah, June 2001.

[2] Amol Bakshi, Jingzhao Ou, and Viktor K. Prasanna, *Towards Automatic Synthesis of a Class of Application-Specific Sensor Networks*, International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2002), October 2002.

[3] David Brooks , Vivek Tiwari and Margaret Martonosi, *Wattch: a framework for architectural-level power analysis and optimizations*, Proceedings of the 27th annual international symposium on Computer architecture, pp. 83-94, June 2000, Vancouver, British Columbia, Canada.

[4] N. Drago, F. Fummi and M. Poncino, *Modeling network embedded systems with NS-2 and SystemC*, Circuits and Systems for Communications, 2002, Proceedings. ICCSC '02. 1st IEEE International Conference on , 26-28 June 2002, pp. 240 –245.

[5] *GlomoSim 2.0*, http://pcl.cs.ucla.edu/projects/glomosim, 2001.

[6] Intel StrongARM Processors, http://developer.intel.com/ design/pca/applicationsprocessors/1110_brf.htm.

[7] M. Martignano, N. Drago, F. Fummi and S. Martini, *A combined approach to validate the design of embedded network devices*, Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on , Volume: 3 , 26-29 May 2002, pp. III-169 -III-172 vol.3.

[8] *Ns-2 simulator*, http://www.isi.edu/nsnam/ns, 2001.

[9] S. Park, A. Savvides and M. B. Srivastava, *SensorSim: A Simulation Framework for Sensor Networks*, in the Proceedings of MSWiM 2000, Boston, MA, August 11, 2000.

[10] S. Park, A. Savvides and M. B. Srivastava, *Simulating networks of wireless sensors*. Winter Simulation Conference, pp. 1330-1338, 2001.

[11] *Powerimpact*, in http://ee.ucla.edu /PowerImpact, 2002.

[12] T. K. Tan, A. Raghunathan and N. K. Jha, *EMSIM: an energy simulation framework for an embedded operating system*, IEEE International Symposium on Circuits and Systems, pp. 464-467, volume 2, 2002.

[13] V. Raghunathan, S. Ganeriwal, C. Schurgers, M. B. Srivastava, *E2WFQ: An Energy Efficient Fair Scheduling Policy for Wireless Systems*, International Symposium on Low Power Electronics and Design (ISLPED'02), Monterey, CA, August 12-14, 2002.

[14] V. Raghunathan, C. Schurgers, S. Park, and M.B. Srivastava, *Energy-Aware Wireless Microsensor Networks*, IEEE Signal Processing Magazine, vol.19, no.2, March 2002, pp.40-50.

[15] A. Savvides, S. Park, and M. B. Srivastava, *On Modeling Networks of Wireless Micro Sensors,* poster session at SIGMETRICS 2001, Boston, MA, June 2001.

[16] C. Schurgers, O. Aberthorne, M. B. Srivastava, *Modulation Scaling for Energy Aware Communication Systems*, International Symposium on Low Power Electronics and Design (ISLPED'01), Huntington Beach, CA, pp. 96-99, August 6-7, 2001.

[17] A. Sinha and A. P. Chandrakasan, *JouleTrack – A Web based tool for software energy profiling*, Proceedings of the 38th Design Automation Conference, 2001, pp. 220-225.

[18] C. Ulmer, *Sensor Network Simulator*, http://users.ece.gatech.edu/~grimace/research/sensorsimii, 2001.

[19] W. Ye , N. Vijaykrishnan , M. Kandemir and M. J. Irwin, *The design and use of simplepower: a cycle-accurate energy estimation tool*, Proceedings of the 37th Design Automation Conference, pp. 340-345, June 05-09, 2000, Los Angeles, California, United States