Flexible Energy-Aware Simulation of Heterogeneous Wireless Sensor Networks

Franco Fummi and Giovanni Perbellini and Davide Quaglia
Dipartimento di Informatica, Università di Verona
Strada le Grazie 15, I-37134, Verona, Italy
Email: [franco.fummi|giovanni.perbellini|davide.quaglia]@univr.it

Andrea Acquaviva
DAUIN - Politecnico di Torino
Corso Duca Degli Abruzzi 24, 10129 Torino, Italy
Email: andrea.acquaviva@polito.it

***Abstract*—This paper presents an accurate and scalable implementation of an energy-aware simulator for wireless sensor networks (WSN's). Scalability and accuracy have been achieved through an energy-aware instrumentation of the Instruction Set Simulator of node's microcontroller and a functional SystemC TLM model of the radio module implementing the IEEE 802.15.4 protocol. The framework allows to execute actual software and to evaluate accurately its effect on the network lifetime. We first validate energy estimation results against a working hardware prototype of a wireless sensor node. The methodology, compared against state-of-the-art simulators such as NS-2, represents a flexible and scalable solution for fast and accurate prototyping of WSN software.**

I. Introduction

Wireless sensor networks (WSN's) are complex systems that require scalable, flexible, and accurate simulation tools for their design [1]. The number of entities involved in a WSN may range from tens to thousands according to the application and the same simulation tool must reproduce their behavior in both cases without problems in speed and resource consumption. A good technique to speed up the simulation is decreasing the detail level at least for some nodes. This approach requires good flexibility, i.e., capability to simulate nodes described at different detail level or even with different functionality, e.g., sensing, routing, and bridging towards other networks. Finally WSN simulation must be accurate, i.e., statistics must capture the performance of the actual system with high fidelity. A particular aspect which must be accurately evaluated is power consumption that may affect network lifetime.

Figure 1 shows the different components to be modeled in a simple WSN of four nodes. First of all, not all nodes require the same level of detail. For example, Node 0 requires an accurate modeling of HW (e.g., I/O peripherals, timers, analog-to-digital converter, sensors, and radio module) and SW (e.g., application, operating system, drivers and interrupt service routines) components while in other nodes HW/SW partitioning has not been performed. Node 1 and node 2 are described as a set of smaller inter-connected blocks while Node 3 is described in a functional way. Furthermore, the interconnection of smaller blocks in Node 1 and Node 2 can be represented either at transactional level (TLM) (i.e., through function calls) or at RT level (RTL) (i.e., through signals and ports).

Research activity partially supported by the European project FP6-2005-IST-5-033506-STP

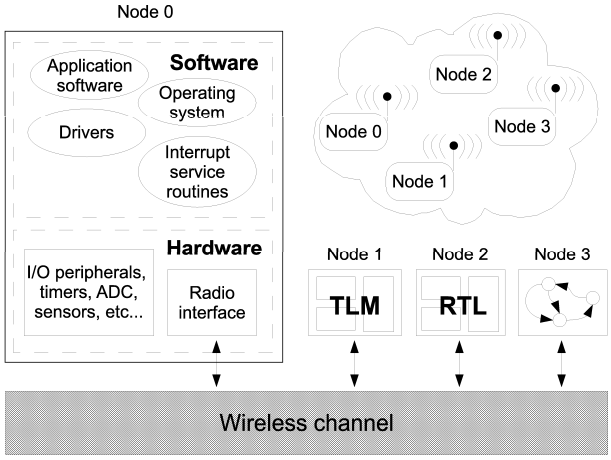

Fig. 1. Modeling aspects of a WSN.

Among the tools focusing on the software aspect, WSim [4], TOSSIM and its extension PowerTOSSIM [2] emulate the target instruction set on the host machine where the simulator runs. Regarding network simulation, WSNet [4] and NS-2 [3] are two of the traditional tools for WSN's. They represent nodes as traffic generators, focusing on the accurate simulation of communication protocols. Both approaches do not allow to describe HW components as in traditional HW description languages and, therefore, lack of integration with the HW design flow. Furthermore, they do not reproduce power-saving mechanisms based on a shared knowledge between network modules and the CPU.

A different solution consists in using an instruction set simulator (ISS) to execute the target code. This approach allows to simulate frequency scaling as well as shutdown policies as long as the ISS supports these features. However, the ISS cannot model efficiently the other HW components belonging to a wireless node and the communication channel and thus its combined use with other tools could improve accuracy. In literature SystemC [5] has been combined with ISS to create a *co-simulation* approach [6]. SystemC has the key feature of allowing the representation of system components at different detail levels, i.e., behavioral, structural

978-3-9810801-5-5/DATE09 © 2009 EDAA

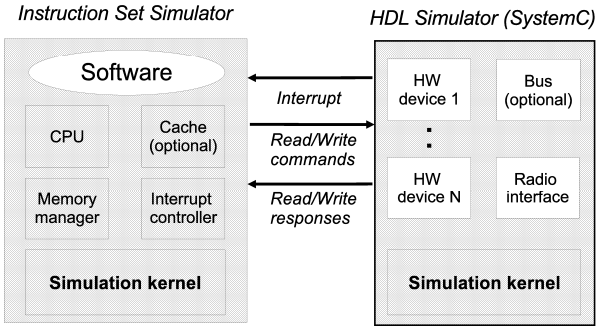

Fig. 2. Co-simulation schema.

and gate level. Furthermore, it can be used also to simulate the communication channel through appropriate libraries as explained below. These features increase the scalability of the WSN simulator by allowing the simulation of network nodes with variable accuracy. During design and optimization of node's software, programmers require a few nodes running the target software. Other nodes acting as routers or interfering nodes can be represented as traffic generators thus speeding up simulation time.

The paper is organized as follows. Section II gives an overview of the co-simulation approach as starting point of our work. Section III describes our contribution to provide a scalable power-aware co-simulation of WSN's. Section IV describes how this methodology can be applied to a reference platform. Section V reports and discusses the experimental results. Finally, conclusions are drawn in Section VI.

II. Background

A. Co-simulation approach

The energy-aware co-simulation methodology described in this paper is based on the ISS-centric co-simulation approach [6] depicted in Figure 2. This approach consists of an ISS running the application SW and system SW (i.e., operating system, drivers and interrupt service routines) interfaced to hardware models, specified by using SystemC at some levels of abstraction.

The ISS reproduces the effect of each instruction on the internal registers and memory locations. It keeps the notion of time by assigning a given number of clock cycles to each instruction and assuming a given clock frequency. In the case study of Section V the dynamic control of clock frequency will be the core of a power-saving policy. In general, the number of clock cycles needed to perform an instruction also depends on the presence of a cache which can be modeled in the ISS.

The memory is modeled directly in the ISS as a data array. An interrupt controller has been also implemented in the ISS; when an interrupt arrives, the value of program counter is changed accordingly to execute the corresponding service routine.

Hardware simulation is implemented by SystemC [5] which provides both a simulation kernel and a HW description language to describe devices at various level of detail (i.e., from transactional level to RT level). The SystemC kernel was modified to communicate with the ISS through inter-process communication primitives (e.g., a socket or shared memory). The information exchanged are: commands to read/write HW registers (from ISS to SystemC), interrupts (from SystemC to ISS). Each piece of information exchanged reports the simulation time at which it has been generated so that HW and SW simulations can proceed in a synchronized way.

B. Network Simulation

Figure 3 shows the architecture of a network scenario modeled with SCNSL. Three components are highlighted, i.e., the `Node`, the `NodeProxy`, and the `Network`.

Module `Node` models a network node and it can be used as a wrapper or base class for any SystemC module which must communicate over the channel. The node has two input ports and an output port to model network input, received signal energy and network output, respectively. The presence of a network port for each direction allows to model both wired and wireless interfaces; the input port reporting the received signal energy has been introduced since in some modern wireless interfaces it can be used both for carrier sense and localization techniques based on the evaluation of the received signal strength. Module `Node` has a set of properties which are used by the simulation framework to reproduce network behavior. Transmission *rate* represents the number of bits per unit of time which the interface can handle; it is used to compute the transmission delay and the network load. The *transmission power* is used to evaluate the transmission range and the signal-to-noise ratio.

Class `Network` is the core of the network simulator. It reproduces the behavior of the channel and manages the packet forwarding from the source node to destination nodes; transmission delay, path loss, collisions, and the state of destination nodes are taken into account. To accomplish this task, class `Network` maintains an object reference for each `NodeProxy` and keeps track of every on-going transmission.

Module `NodeProxy` is the interface between `Nodes` and `Network` and each instance of `Node` must be bound to a different instance of `NodeProxy`. Each `Node` interacts with its own `NodeProxy` by using SystemC ports only, while `NodeProxies` maintain an object reference of the `Network` and call its methods. Also the `Network` instance maintains an object reference for each `NodeProxy`. By using `NodeProxy`, nodes can be designed as pure SystemC modules without object references to other non-SystemC classes; this approach enables the use of traditional hardware verification and synthesis tools. `NodeProxy` manages two node properties: *node position* and *receiver sensitivity*. Node position is used to compute the path loss and to reproduce a mobile scenario. The receiver sensitivity is the minimum signal power below which the packet cannot be received.

III. Power-aware WSN co-simulation

Figure 4 shows the proposed modeling approach which is applied to an example of three interacting wireless nodes

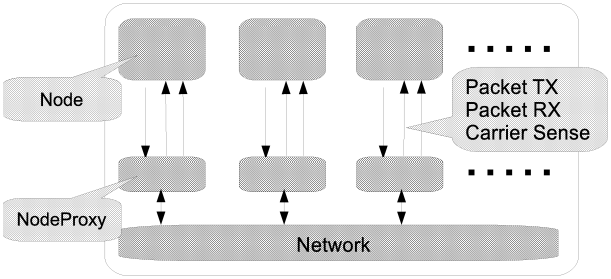

Fig. 3. Architecture of a network scenario modeled with the SystemC Network Simulation Library.

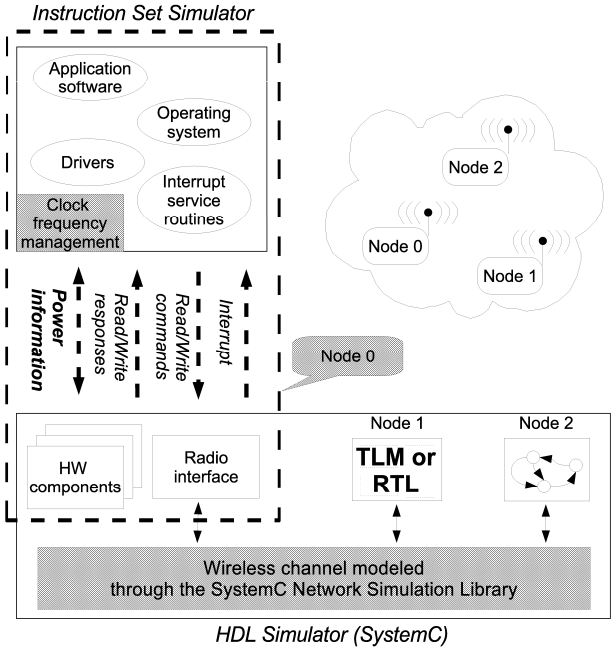

Fig. 4. Modeling approach with nodes at different detail level.

shown in the upper right part of the Figure. Normal arrows represent interactions inside the model, while dashed arrows represent interactions between simulation tools.

A. Support to Power-awareness

With respect to the basic co-simulation approach shown in Figure 2, we added some new features to support power-awareness:

- a new bi-directional channel between the ISS and the SystemC simulation kernel to exchange power information;
- a mechanism for the dynamic change of clock frequency in the ISS.

B. Hierarchical Modeling

In Figure 4 the nodes are described with different detail levels. Node 0 is described by fully exploiting the co-simulation framework described in Section II. The ISS executes application SW, the operating system, drivers, and the interrupt service routines. SystemC is used to model the radio interface and other HW peripherals such as timers, the serial interface, and physical sensors.

The other nodes of the example are modeled in pure SystemC with different abstraction levels trading off between design needs and simulation speed; the description can be either structural, i.e., reproducing the interaction of smaller modules, or functional, i.e., reproducing the behavior (e.g., through a finite state machine). Furthermore, the interaction of smaller modules can be described either at transactional level or RT level.

SystemC is also used to reproduce the behavior of the radio channel through the adoption of an open source library named SystemC Network Simulation Library (SCNSL) [8] whose features are summarized in Section II-B. Through this library, SystemC becomes a glue which gathers together all network nodes as a kind of *single system*.

The choice between co-simulation and pure SystemC for node modeling depends on the design phase and evaluation needs. Pure SystemC may be used to create raw functional descriptions to be progressively refined while co-simulation is used when HW/SW partition has already been performed, when the HW platform has been chosen, and when the designer wants to test the actual SW and to evaluate its performance and resource needs.

In general co-simulation is slower than pure SystemC simulation because of the need to exchange information between two different processes on the host machine. However SW and HW components can coupled at different levels of detail leading to different co-simulation overhead as follows.

- *Cycle-accurate interaction*: each read/write operation on HW registers and interrupt event is decomposed into bus operations and each operation generates messages between the ISS and SystemC. This kind of interaction can be used when the HW description is highly detailed and the bus has been introduced. However, the co-simulation overhead is very high and common CPU emulators do not generate bus actions; for these reasons we decided to avoid this approach.
- *Register-driven interaction*: each read/write operation on HW registers and interrupt event generates exactly one interaction between the ISS and SystemC. The advantage of this approach is that actual SW drivers can be used since they access HW components by writing/reading registers. In our case study, this approach has been used to interact with simple HW components (i.e., the UART and the accelerometer).
- *Function-driven interaction*: each interaction between the ISS and SystemC represents a high-level operation (e.g., packet transmission/reception on the network interface). This kind of interaction leads to the lowest co-simulation overhead but it requires the use of specific device drivers in the guest environment. In our case study, this approach has been used to interact with complex HW components such as the radio module.

IV. Reference Power-aware Modeling

This Section describes the modeling of a wireless sensor network based on the AquisGrain-2 node provided by

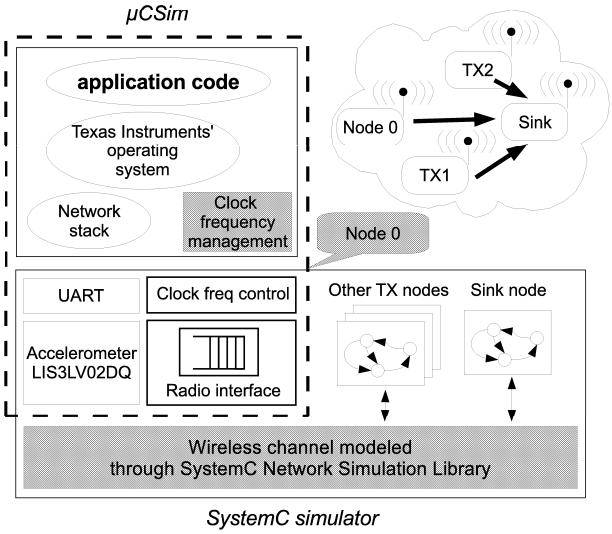

Fig. 5. Model of the AquisGrain-2 node and of the wireless network.

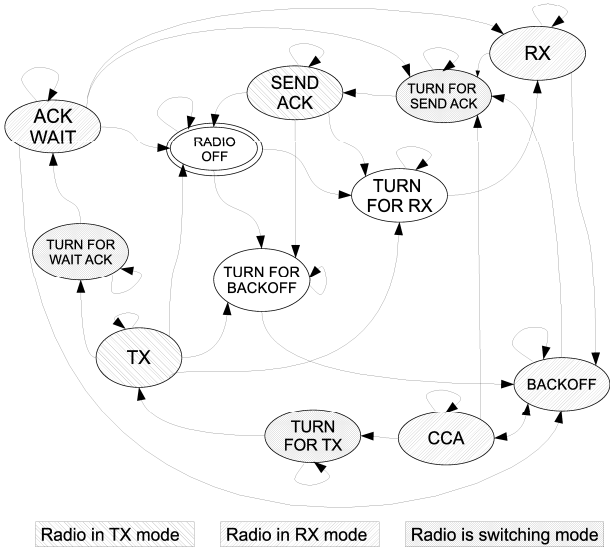

Fig. 6. Finite state machine of the IEEE 802.15.4 MAC protocol.

Philips [9] and designed for body-worn smart medical sensors. The main modules belonging to an AquisGrain2 node are: (I) Application SW and communication protocol stack, (II) Texas Instruments CC2430-F128, and (III) sensors/actuators. The CC2430-F128 is a system-on-chip containing an Intel 8051 MCU, 128 KB flash memory, an IEEE 802.15.4 radio module, a UART interface and several analog and digital ports. The Intel 8051 is a 8-bit data, 16-bit address microcontroller for embedded systems. The CC2430-F128 extends the 8051 addressing space to 128 KB by using a support register and a bank-oriented accessing mode. AquisGrain-2 node is equipped with a LIS3LV02DQ accelerometer by ST Microelectronics; this chip is connected to the MCU through the UART interface.

The test application consists in sampling acceleration values and sending them to a sink node. The application SW on Node 0 reads acceleration values from the UART module and puts them into the RF module for the transmission on the radio channel.

A. Model of the AquisGrain platform

Figure 5 shows how the different elements of the co-simulation framework are applied to reference platform. We used an ISS named μCSim [7] which emulates the behavior of the Intel 8051 microcontroller. This choice is related to the need of simulating a specific WSN platform; however the changes introduced in μCSim can be also applied to other ISS's. μCsim [7] recognizes the same HEX-format executable files downloaded on the actual board. The ISS has been modified to support co-simulation, CC2430 addressing mechanism, and clock frequency change. The accelerometer and the UART are memory-mapped devices modeled at RT level by using the SystemC language; co-simulation of these components follows the register-driven approach explained in Section III-B.

The radio interface contains the finite state machine of the peer-to-peer unslotted 802.15.4 MAC protocol. The states are shown in Figure 6; each of them not only describes a communication behavior but also it is characterized by a specific energy consumption thus enabling accurate power estimation during simulation. This module has been implemented both at RTL and at TLM-PVT level to analyze simulation speed in both cases.

TinyOS implements CC2420 radio stack (e.g., the Telos and MicaZ platforms both use the CC2420 radio stack). However, the TinyOS radio stack does not support the full IEEE 802.15.4 standard; in particular, it provided B-MAC primitives and sends 802.15.4-compliant packets, but does not implement the 802.15.4 MAC protocol.

B. Application and Frequency Scaling

Modern microcontrollers supports various operating frequencies that can be programmed by the applications through a dedicated API. However, depending on the system architecture, certain frequencies can be prohibited when the node is in a particular state. For instance, our MCU is integrated together with the RF transceiver. Since the transceiver requires the highest clock frequency to work, frequency scaling is disabled during transmission and reception. On the other side, when the radio interface is off, the frequency of the microcontroller can switch among a number of discrete frequencies. Typically these frequencies are obtained by dividing the maximum frequency by a factor of two. Indeed, frequency scaling is obtained by a pre-scaler hardware module acting on the clock signal entering the microcontroller core.

The acceleration sample rate depends on the speed of the microcontroller, as shown in Table I. There is a negligible delay for changing the speed at run time being this obtained by programming a pre-scaler sitting on the clock path to the core. In all the experiments the transmission buffer of the network interface is set to 32 packets. Each simulation ends after the transmission of a given amount of packets over the network.

Frequency	Output Rate	Power
8 MHz	2 KHz	8.25 mW
16 MHz	4 KHz	14.85 mW
32 MHz	8 KHz	31.35 mW

TABLE I
POWER CONSUMPTION AND PERFORMANCE OF THE MICROCONTROLLER AS A FUNCTION OF THE CLOCK FREQUENCY.

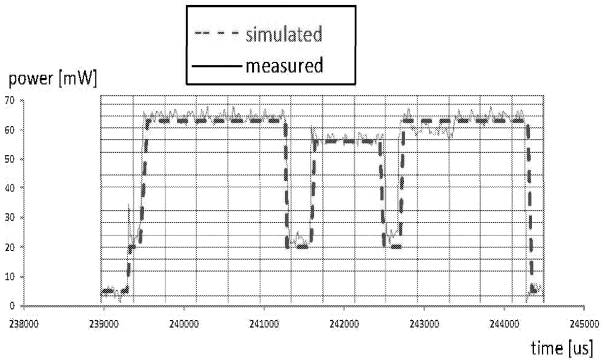

Fig. 7. Comparison between measured and simulated power consumption for a data request operation with acknowledge received.

V. EXPERIMENTS

A. Validation of Power Estimation

Characterization and validation of power consumption estimation provided by the network simulation has been performed against the Aquisgrane-2 wireless sensor node. The current absorbed by the radio interface has been monitored and sampled using a data acquisition board (DAQ) connected to a PC [10]. This allowed to extract power consumption levels used for power state characterization.

Since the 802.15.4 model captures the most relevant states of the transceiver both from a power and timing viewpoint, the resulting energy estimation is very accurate. This is confirmed by validation results where the energy consumed for one packet transmission has been quantitatively compared between hardware and simulator. Results indicate an error lower than 5%. In Figure 7 the simulator trace is reported together with the power trace obtained by hardware current measurements.

B. Scalability Assessment

Different scenarios have been simulated with SCNSL by using nodes at different abstraction levels: *1)* all nodes at TLM-PVT level, *2)* all nodes at RTL, and *3)* Node 0 at RTL and other nodes at TLM-PVT. The designer had written 172 code lines for the `sc_main()`, 688 code lines for the RTL node and 633 code lines for the TLM-PVT node.

Figure 8 shows the CPU time as a function of the number of nodes for the three scenarios and for a simulation provided by NS-2 representing the behavior of a pure network simulator. A logarithmic scale has been used to better show results. Simulations have been performed on the Intel Xeon 2.8 MHz with 8 GB of RAM and 2.6.23 Linux kernel; CPU time has been computed with the `time` command by summing up user and system time.

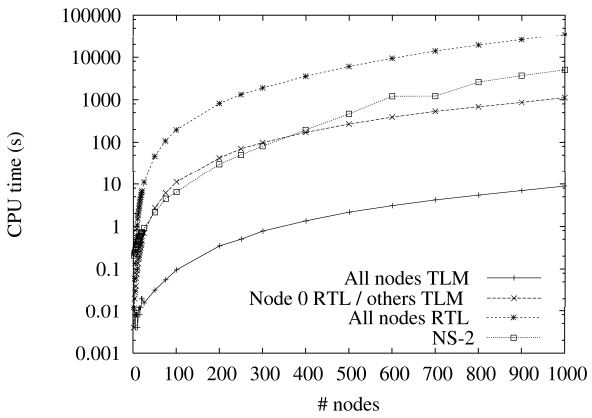

Fig. 8. CPU time as a function of the number of simulated nodes for the different tools and abstraction levels.

The speed of SCNSL simulations at TLM-PVT level is about two-order-magnitude higher than in case of NS-2 simulation showing the validity of SCNSL as a tool for efficient network simulation. Simulations at RT level are clearly slower because each node is implemented as a clocked finite state machine as commonly done to increase model accuracy in System design. However a good trade-off between simulation speed and accuracy can be achieved by mixing nodes at different abstraction levels; in this case, experimental results report about the same performance of NS-2 with the advantage that at least one node is described at RT level.

C. Design of energy management policies

In this section we describe how system designers can profit from the proposed framework to compare the effectiveness of power management policies for wireless sensor networks. The policies are implemented within the network interface driver, so that their cost in terms of performance and power is taken into account.

The clock scaling support can be exploited by an energy management policy to save power when the network is congested. In this case, the processing speed can be slowed-down and later sped-up when the congestion period ends. The transmission queue can be used as a monitor of the network congestion. In this experiments, congestion is emulated by imposing busy periods within the 802.15.4 model.

When a congestion period begins, the transmission queue starts filling-up at a speed depending on the producer rate. In order to save power on the microcontroller, a simple approach could be to switch to the lowest possible frequency (or even shut-off) when and if the transmission queue becomes full and restore the maximum frequency when the queue starts to be depleted again. A more aggressive approach would be to scale the processor speed before the queue is full to save more power. However, there are two side effects on the performance viewpoint. First, the occupancy level of the queue is lower on average, which implies that there could be less packets to be

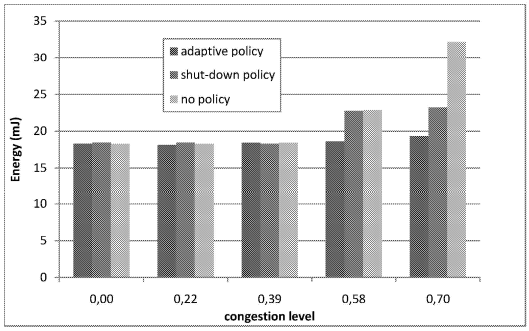

Fig. 9. Energy consumption of the microcontroller for three different cases: no policy, shut-down and congestion-adaptive.

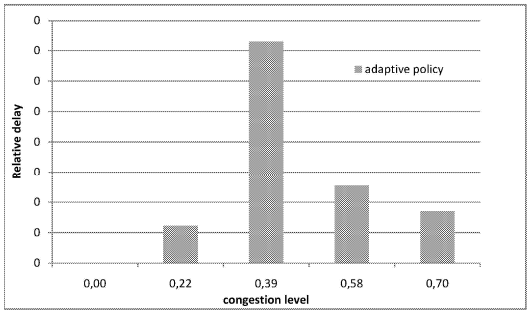

Fig. 10. Impact on performance of the congestion-adaptive policy.

transmitted once the congestion period finishes. Second, if the algorithm is not reactive enough, the processor speed at the end of the congestion period could be lower than the maximum, leading to a lower transmission rate on average. As such, there is a trade-off between power savings and performance that must be explored.

The trade-off depends on the characteristics of the control law by which the core frequency is regulated depending on the queue occupancy, which in turns depends on the network congestion. The simulator allows to tune the control algorithm to achieve the wanted trade-off. In this experiment, the adaptive policy is implemented using a linear proportional-integrative control law.

Energy comparison results between a simple shut-down policy and a congestion-adaptive approach are shown in Figure 9 as a function of the congestion fraction. This is the fraction of simulated time in which the network is busy because of congestion. Energy consumption values refer to one single node. Energy consumption when no power management policy is applied are also reported. Performance penalties related to the adaptive policy are also reported in Figure 10. They are normalized with respect to the simulation without congestion. Thanks to the energy estimation capabilities and frequency setting support it is possible to evaluate the effectiveness of power management policies and refine them to obtained the wanted performance vs power trade-off.

VI. Conclusions

This paper presented an energy-aware simulation tool for wireless sensor networks. We discussed its main features including the synchronization between ISS and SystemC simulation engines and the power estimation technique. We validated the power estimation capability with the CC2430 SoC by Texas Instruments and we compared the performance of the tool with respect to NS-2. Finally, we showed the use of the tool to design a power management policy based on network conditions.

References

[1] C.-Y. Chong and S. P. Kumar. Sensor networks: Evolution, opportunities, and challenges. *Proc. IEEE*, 91(8):1247–1256, Aug. 2003.

[2] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *Proc. 2nd Int. Conf. on Embedded networked sensor systems (SENSYS)*, 2004, pp. 188–200.

[3] S. McCanne and S. Floyd. NS Network Simulator – version 2. *http://www.isi.edu/nsnam/ns*.

[4] G. Chelius, A. Fraboulet, and E. Fleury, "Worldsens: development and prototyping tools for application specific wireless sensors networks," in *Int. ACM Conf. on Information Processing in Sensor Networks (IPSN)*, Apr. 2007.

[5] IEEE Std 1666 - 2005 IEEE Standard SystemC Language Reference Manual. *IEEE Std 1666-2005*, 2006.

[6] F. Fummi, G. Perbellini, M. Loghi, and M. Poncino. ISS-centric modular HW/SW co-simulation. In *GLSVLSI '06: Proceedings of the 16th ACM Great Lakes symposium on VLSI*, pages 31–36, New York, NY, USA, 2006. ACM Press.

[7] D. Drótos. μCSim: Software Simulator for Microcontrollers. *http://mazsola.iit.uni-miskolc.hu/˜drdani/embedded/s51/*.

[8] Systemc network simulation library – version 1.0, 2008. *http://sourceforge.net/projects/scnsl/*.

[9] J. Espina, T. Falck, and O. Mlhens. *Network Topologies, Communication Protocols, and Standards.* In: Yang, G.Z. (ed): Body Sensor Networks, pp. 145-182, Springer, London, England, 2006.

[10] B. Selvig, *AN053: Measuring power consumption with CC2430 and Z-Stack. SWRA144- Texas Instruments*