Accurate Timing in Sensor Network Simulation

Muhammad Hamad Alizai, Olaf Landsiedel, Klaus Wehrle
Distributed Systems Group
RWTH Aachen
{hamad.alizai, olaf.landsiedel, klaus.wehrle}@rwth-aachen.de

ABSTRACT

Accuracy, speed and scalability are the basic requirements of sensor network simulation. To comprehend the accurate behavior of resource constrained embedded systems such as sensor nodes it is important in simulations to model the time-dependent behavior of the system. In this paper we present our extensions of TOSSIM [2] – a widely used event-driven simulation environment for sensor networks – to enable its simulation models to capture the time-accurate behavior of sensor networks by exhibiting timing and interrupt properties of the platform dependent source-code. By mapping the device specific code with the simulation model, we can derive the timing of functional code blocks. As result of such a mapping it is possible to determine the time when a certain code block gets executed and the time the execution takes, eliminating the need of expensive cycle-accurate instruction level simulators with limited speed and restricted scalability.

1. INTRODUCTION

Simulation indisputably remains one of the most important tools for analyzing, evaluating and validating system design. The importance of simulation is further aggravated for systems having an embedded nature, high deployment costs, or possessing unobservable fast interactions yet important to validate the system design. Sensor Networks with their distributed behavior, strenuous deployment requirements, constrained resources, and invisible and unpredictable interaction between the sensor nodes poses additional demands on their simulation.

In the past few years a great deal of effort has been invested in the design and development of simulators for sensor networks to embrace the special requirements imposed by the highly distributed and dynamic nature of sensor networks. Unfortunately all these efforts have made compromises over different attributes of the simulation, for example, accuracy has been compromised over scalability and vice versa. SWAN [4], SensorSim [5], and SENS [6] are examples of sensor network simulators which compromise scalability over accuracy by using nonfigurative models of the sensor nodes. Such simulation models only contribute to quantify network delays, throughputs, packet collisions, power usage and the effect of several power management schemes [3]. However, these models do not reveal the timing and interrupt properties of applications, the operating systems, and hardware components.

ATEMU [7] and Avrora [3] on the other hand are cycle-accurate instruction level simulators for sensor networks with the most expressive simulation models. Nevertheless, they compromise the scalability and performance/speed. ATEMU is 30 times slower than TOSSIM [3], and its poor performance limits its scalability to 120 nodes. Avrora shows better performance measures than ATEMU with reasonably good speed for small number of sensor network nodes but it is still 50% slower than TOSSIM. The performance measures of Avrora have been calculated on a 16 processor machine, not easily accessible to normal end-users and developers. Avrora exhibits typical performance bottlenecks of instruction level simulators when run on customary end-user machines, especially, when several avrora-monitors are enabled for detailed analysis of the sensor network behavior.

Our goal is to provide time accurate simulation for sensor networks at the basic-block granularity (i.e. sequence of instructions with a single entry point, single exit point, and no internal branches) of the source code without compromising the speed and scalability, and hence eliminating the need to use expensive instruction level simulators. We extend TOSSIM to exhibit the timing and interrupt properties of sensor network code without destroying its performance and scalability advantages.

2. TOSSIM

TOSSIM is an extremely fast sensor network simulator scalable to thousands of sensor network nodes. It compiles directly from the TinyOS source code into the simulation environment by adding an alternative compilation target. The fact that it compiles directly from the platform dependent source-code makes it more expressive than SensorSim, SWAN, and SENS. TOSSIM only requires to model the low level components responsible for hardware interaction such as low level access to timers, communication channels, sensors, and the radio. These low level components expose the real hardware and are placed at the Hardware Presentation Layer (HPL) of the TinyOS-2.0's platform abstraction model [8]. TOSSIM also benefits from the event-based, component oriented programming model of TinyOS by translating the asynchronous-events and hardware interrupts into discrete simulator events which drive the simulation.

TOSSIM's level of detail was sufficient to measure packet losses, packet CRC failure rates, and the length of the send queue for up to 8,192 nodes [3]. However, TOSSIM's compilation steps lose the fine-grained timing and interrupt properties of the code that are extremely important for a time-accurate simulation [3].

We address these problems by exploiting the fact that TinyOS runs the same code (except the small platform dependent HPL layer) in simulation and on the sensor network hardware. This feature of TOSSIM enables to create a mapping between the platform dependent binary and the simulation code. We use Mica-2 as our target platform. Our method is to (1) analyze the platform dependent assembly program and compute the cycle count corresponding to each basic-block; (2) assign a priority number to every simulator event to enable TOSSIM to model the interrupt

and preemption behavior of the real hardware; (3) extend the C-source code generated by TOSSIM to (a) increment the simulation clock at the start of every source-code line by the cycle count information obtained in the first step, hence, enabling the TOSSIM to exhibit the timing properties of the code. (b) Re-schedule the TOSSIM event queue at the start of every basic-block on the basis of new timing information obtained, and also on basis of the assigned interrupt priority of each event in the simulation queue to model the masking and preemption properties of the hardware interrupts in the simulation infrastructure.

Our approach is different from CPU-profiling approach in PowerTOSSIM[1] – an extension of TOSSIM for simulating the power consumption of sensor networks, which does offline processing to obtain the cycle counts for CPU power profiling. We, on the other hand embed TOSSIM with the information obtained from the assembly of Mica-2 motes to perform online adjustments in the simulation clock and event queue.

3. TIME ACCURATE SIMULATION

This section describes the details of the time accuracy related problems in TOSSIM and our approach to address these problems.

3.1 Timing Discrepancy

TOSSIM captures the TinyOS event-driven concurrency model at interrupt and task granularity [9], and it has a single queue both for the tasks and the events. The simulation is triggered by the events and the tasks in the TOSSIM event-queue which is sorted in the increasing time order. TOSSIM adjusts its simulation clock at the start of the execution of every event by assigning the time stamp of the recently popped event from the queue to the simulation clock. Events and tasks take zero execution time in TOSSIM as the simulation clock remains unadjusted during the course of execution; hence, TOSSIM loses the fine-grained time accuracy of the code. This imperfection of TOSSIM introduces even more problems, for example, TOSSIM is unable to differentiate between a task requiring a large number clock cycles to transmit several bytes over the radio from a task requiring few clock cycles just to blink an LED attached to the microcontroller pin or to report a timer fire.

The execution time of an event or task may also affect the timing of next events or tasks in the queue as shown in Figure-1. For example, if TOSSIM is currently executing an event associated with high priority interrupt and there is an immediately scheduled task or event representing a low priority interrupt, then its execution time should be delayed – timestamp should be readjusted, at least until the execution of current event is finished. TOSSIM, because of its imperfection to track the system time during execution of an event, is unable to capture this priority based interrupt behavior of the hardware which masks the less priority interrupt or delays the execution of tasks while handling a high priority interrupt. Similarly, in TOSSIM the simulator events run atomically one after another, therefore, unlike on real hardware, interrupts cannot preempt one another [9]. On the other hand, long tasks – tasks requiring several clock cycles to execute, delay the execution of other tasks and can be preempted by events, but TOSSIM is unable to model such behaviors as show in Figure-2.

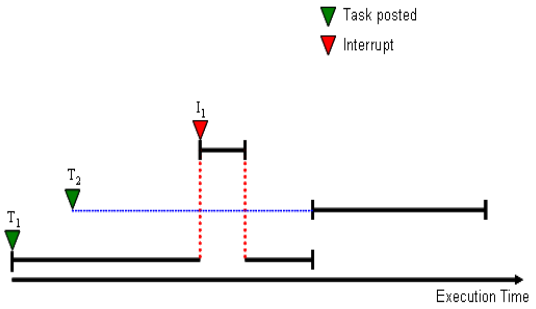

Figure 1. TinyOS event handling and execution flow

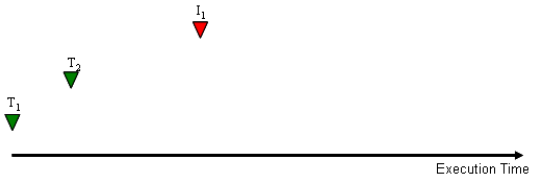

Figure 2. TOSSIM execution flow

3.2 Our Solution

Our approach to solve this timing discrepancy involves three steps.

3.2.1 Basic-block Mapping

We address the timing discrepancy of TOSSIM by enabling it to exhibit the timing properties of the code at the basic-block granularity. We achieve this by creating a mapping between the TOSSIM's C-source code and the assembly of platform dependent code (Mica-2 in our case). Our mapping technique is similar to PowerTOSSIM.

TinyOS uses the NesC compiler to compile the TinyOS component graph to a single C-source file, which in effect is then compiled into the binary for the specified target platform through appropriate C-compiler (i.e. gcc for TOSSIM and avr-gcc for Mica-2). We use the avr-objdump utility with appropriate options to obtain the assembly of Mica-2 platform which also contains a mapping of the assembly instructions to the original nesC source-code. We parse this assembly file to obtain the cycle counts corresponding to the basic-blocks of the source-code. On the other hand, we use the C-source file generated by the nesC compiler for the TOSSIM platform. The C-source file of TOSSIM also provides the mapping between C-source code and the original nesC source code, thus, enabling the mapping between the platform dependent assembly and the TOSSIM's C-source file.

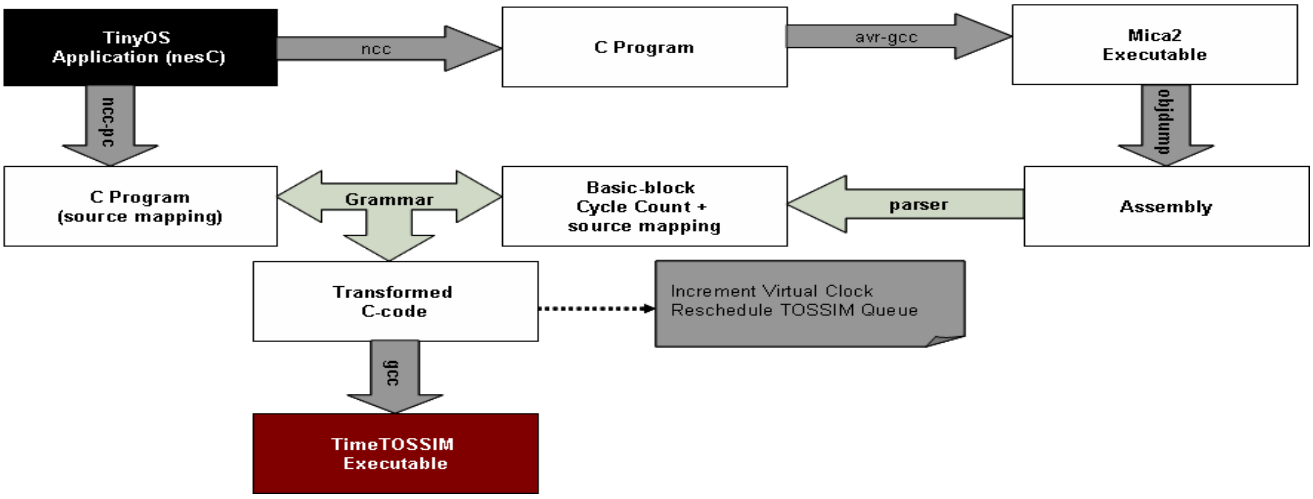

Figure 3. Block Diagram: Extending TOSSIM to capture time-accurate behavior of the system

We parse the C-source file of TOSSIM using ANTLR's [10] GNU-C grammar to perform source-to-source transformation. Our transformation includes (1) extending the C-source file by adding functions that increment the simulation clock and perform online adjustments in the TOSSIM Queue; (2) adding a call to these functions at the start of every basic-block. These transformations enable TOSSIM to exhibit the timing properties of application at the basic-block granularity. The whole process of extending the TOSSIM is shown in Figure-3.

3.2.2 Rescheduling the TOSSIM Event Queue

By extending TOSSIM to incorporate the timing properties of the system at basic-block granularity also enables us to reschedule the TOSSIM queue and intensify TOSSIM even further to exhibit the interrupt properties of the hardware. We do this by rescheduling every event and task in the TOSSIM queue (hereinafter referred to as target event) whose time-stamp is less than the simulation clock time. Additionally, we assign interrupt priority numbers to every event in the TOSSIM Queue. Tasks are assigned zero interrupt priority. Rescheduling the event queue introduces two possibilities; (1) either the target event in the event-queue has an interrupt priority less than or equal to the current event or task being executed. In this case we increment the time-stamp of the target event by the amount of time needed to execute the current basic-block; (2) or the target event represents a high priority interrupt. In this case we interleave the execution of the current event or task (i.e. at the start of the basic-block) and start the execution of the target event in the queue with high priority.

3.2.3 Hardware Component Profiling

The NesC compiler, when compiling for TOSSIM, replaces the components at the HPL of platform abstraction architecture with their corresponding reimplementation for TOSSIM. Our transformations work very well when the TOSSIM is executing the platform independent part of the application code (i.e .common for TOSSIM and Mica-2 platform), and we achieve 100% basic-block mapping. But this basic-block mapping fails and we loose our granularity once the TOSSIM enters the execution of its own reimplementation of hardware related components.

We address this problem by profiling the hardware related components. We observed that the behavior of these low level components, that expose the hardware, is static. For example, it always takes the same amount of cycles to turn an LED On or Off. It is also possible to do some manual mapping between the components that share the same algorithmic properties and execution flow but their execution time is not static. For example, TOSSIM has its own scheduler but its execution flow is analogous to the TinyOS Scheduler, nonetheless, execution time of the scheduler is not static because it performs some context switching as well as processes long queues of tasks. We do manual mapping between the TinyOS scheduler and the TOSSIM Scheduler to maintain the same basic-block granularity and timing resolutions that we desire to achieve.

4. CONCLUSION AND FUTURE WORK

In this paper we discussed the importance of timing properties of the source code in simulations. We showcased a distinct technique and demonstrated how time-accurate simulation can be achieved using this approach as described in section-3.2. It enables to model the time-accurate behavior of the system at the basic-block granularity without using the non scalable and low performance instruction level simulators.

We are still in the active development phase of our work. Intense evaluation is yet to be performed, though the initial results are very promising. We achieve a beyond 99% time accuracy with basic prototype applications like Blink and TestScheduler. We plan to rectify TOSSIM's hardware models including timers and radio to model the original hardware accurately. TOSSIM is also unable to model the behavior of atomic statements – block of statement that run uninterrupted. Access to the application code at the basic-block level can also help in accurately modeling the atomic statement blocks in the code.

5. REFERENCES

[1] Victor Schnayder, Mark Hampstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. *Simulating the power consumption of large-scale sensor network applications*. In Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys) 2003, Nov. 2003.

[2] P. Levis, N. Lee, M. Welsh, and D. Culler. *TOSSIM:Accurate and scalable simulation of entire TinyOS applications*. In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys) 2003, Nov. 2003.

[3] Ben Titzer, Daniel Lee, and Jens Palsberg. *Avrora: Scalable Sensor Network Simulation with Precise Timing*. In Proceedings of IPSN'05, Fourth International Conference on Information Processing in Sensor Networks, Los Angeles, 2005.

[4] J. Liu, D. Nicol, F. Perrone, M. Liljenstam, C. Elliot, and D. Pearson. *Simulation modeling of large-scale ad-hoc sensor networks*. In Proc. European Interoperability Workshop 2001, London, England, June 2001.

[5] S. Park, A. Savvides, and M. B. Srivastava. *SensorSim: A simulation framework for sensor networks*. In Proc. MSWIM 2000, Boston, MA, August 2000.

[6] S. Sundresh, W.-Y. Kim, and G. Agha. *SENS: A sensor, environment and network simulator*. In Proc. 37th Annual Simulation Symposium (ANSS '04), 2004.

[7] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, and M. Karir. *ATEMU: A fine-grained sensor network simulator*. In Proceedings of SECON'04, First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004.

[8] Vlado Handziski, Joseph Polastre, Jan-Hinrich Hauer, Cory Sharp, Adam Wolisz, David Culler, David Gay. *TinyOS 2.0 Enhancement Proposal (TEP – 2)*. http://www.tinyos.net/tinyos-2.x/doc/html/tep2.html

[9] Philip Levis and Nelson Lee. *TOSSIM: A Simulator for TinyOS Networks* http://www.cs.berkeley.edu/~pal/research/../pubs/nido.pdf.

[10] Terence Parr. *ANTLR Parser Generator*. http://www.antlr.org/.