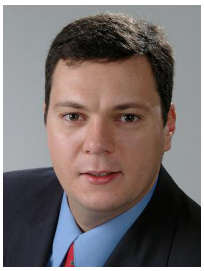
Muhammad Hamad Alizai, Olaf Landsiedel and Klaus Wehrle, Distributed Systems Group, RWTH Aachen University, Germany

Modeling Execution Time and Energy Consumption in Sensor Node Simulation

Muhammad Hamad Alizai is a PhD student at the Distributed Systems Group, RWTH Aachen University. He received his Masters Degree from RWTH Aachen in 2007. His studies were funded by "DAAD-Siemens scholarship Asia 21st century". Hamad started his PhD in the March of 2008 under the PhD scholarship program of DAAD. His research interests include wireless sensor networks mainly focusing on modeling & simulation, protocol design, and link level optimizations. He is the lead developer of TimeTOSSIM – a time accurate extension of TOSSIM simulator.

Olaf Landsiedel is a PhD student at the Distributed Systems Group, RWTH Aachen University. He received his Masters Degree in Computer Science from the University of Kansas, where he graduated in 2003 with honors. His studies were funded by a Fulbright and a Direct Exchange Scholarship. Olaf received his undergraduate degree from the University of Kiel. His research interests include protocol and systems engineering as well as network modeling and simulation with a focus on embedded systems.



Klaus Wehrle is the head of the Distributed Systems Group at RWTH Aachen University. Klaus received his Diploma and PhD from the University of Karlsruhe in 1999 and 2002, respectively, both with honors. From 2002 to 2003, Klaus was Postdoctoral Fellow at ICSI Berkeley. His research activities are focused on protocol and systems engineering, modeling and network simulation, Peer-to-Peer systems, sensor networks, and operating system issues of networking. Klaus actively participates in the IETF; recently RFC 3662 and RFC 3754 were published.

ABSTRACT

Constrained energy and computational resources of sensor nodes are the two most critical limitations of sensor networks. Moreover, sensor nodes are envisioned to be deployed at inaccessible, remote and harsh physical environments where exchanging node's power supply is not feasible. Therefore, it is essential for sensor network developers to thoroughly evaluate and fine-tune their applications from energy and timing perspectives. As incorrect energy estimates could result in underperformance or possible breakdown of real sensor deployments before fulfilling the desired operation.

Modeling energy-states of each hardware device and the time duration it spends in each state is the basic requirement for accurate energy[1] prediction in discrete event based simulations. In this article we present the architecture of our power related extensions to TimeTOSSIM – a TinyOS based simulation environment for sensor network evaluation. It employs fine grained, automated instrumentation of simulation models with cycle counts derived from application binaries to enable time accurate simulations. By instrumenting the simulation models with timing information we can capture the duty-cycle and energy-state of each hardware component. As a result, the energy consumption of each component of a sensor node can be computed. The presented approach achieves highly precise time accuracy when compared to emulation.

I. INTRODUCTION

The desired small physical size of a sensor node and the absence of permanent network infrastructure results in lack of constant supply of power for sensor nodes. Therefore, the operation of sensor nodes rely on limited energy reserves, typically in the form of batteries and solar cells. As a result, energy still dominates as a primary concern both in soft- and hardware development for sensor networks. On the hardware side, it results in using low-power technologies for communication, processing and sensing hardware. As a consequence, it strictly limits the capabilities of sensor nodes from these perspectives. Recognizing severe resource restrictions at the software side, consuming minimal energy and computational resources is the key design objective for algorithms. From physical up to the application layer, research focuses on developing energy aware encoding, medium access, and routing schemes [2][3].

Thorough evaluation of such energy aware applications before deployment is essential as real deployments are costly. Hence, developing energy aware applications and protocols requires a new set of evaluation tools to assist developers. Accurate prediction of energy consumption and execution time of algorithms are among the key characteristics of such tools.

We identify three main requirements in discrete event based simulation of sensor nodes to accurately predict the power consumption of applications. First, it shall model the energy-state(s) of each hardware device. For example, in the case of radio chip, it includes transmitting, receiving, power-down, and idle as energy-states. Second, it shall determine the time a device spends in each of its energy-states during simulation. Higher time resolutions would definitely result in more accurate energy predictions. Third,

[1] We use the term *power* and *energy* interchangeably throughout this article.

it shall provide accurate models of energy consumed per time by a device in each of its states.

In this article we introduce TimeTOSSIM [10] and our architecture for its energy related extensions. TimeTOSSIM enables time accurate simulations of sensor network applications. It is an extension of TOSSIM [6], a discrete event based simulation tool for sensor networks. The principle technique to achieve time accuracy in TimeTOSSIM is to map the platform dependent binary with simulation source-code. Such a mapping determines the number of clock-cycles consumed by each source-code line. As a result, we can instrument the simulation sources to increment the simulation clock accordingly. TimeTOSSIM achieves a granularity of source-code line level. This high level of timing detail attained by TimeTOSSIM simulation enables us to capture the energy-states and transitions of each hardware device. Similarly, the time, i.e. the number of clock cycles, consumed by a device in each of its energy-states is determined by tracking the simulation-clock. By combining accurate energy models with the detailed timing and state information revealed by TimeTOSSIM, we can accurately predict the power consumption of sensor nodes.

The rest of this article is organized as follows. Section II presents related work and background. In section III we discuss TOSSIM and its energy related extensions. Sections IV and V present our approach for modeling execution time and energy consumption respectively. Section VI presents detailed evaluation of accuracy and performance of TimeTOSSIM. Section VII concludes the article and presents future work directions.

II. Related Work

In recent years, the complexity of deployed sensor network applications has heavily increased [11][12]. Therefore, high degree of realism is desired to thoroughly evaluate applications before deployment. Similarly, it is also of strong interest that the evaluation tool be fast and scalable to a very large number of sensor nodes. Complying with the demands of sensor network evaluation, many discrete event simulation [7][8] and cycle accurate emulation [1][9] based tools have been developed. However, we believe that none of these tools has been able to fully integrate the aforementioned evaluation requirements of sensor networks.

Discrete event based simulation, due to its high performance and scalability, is used to evaluate algorithmic functionality of sensor network applications. However, its high level of abstraction and therefore lack of detailed timing prohibits the use of simulation[2] for in-depth analysis of applications. SensorSim [7] and SENS [8] are examples of discrete event based simulators for sensor networks which compromise accuracy over scalability by using abstract simulation models of sensor nodes.

Emulation, as it is accurate down to the clock-cycle granularity, offers detailed evaluation of applications and operating systems. AEON [5] is an emulation based energy estimation tool for sensor networks. As it is an extension of Avrora emulator [9][10], therefore it is inherently cycle accurate and models the energy consumption at clock-cycle granularity. Emulation based evaluation tools also provide deep insight into the much important timing and interrupt properties of applications. However, emulation typically suffers from low speed, limited scalability and platform dependence. Furthermore, emulation environments have reached a complexity which is an order of magnitude higher than the system to evaluate, i.e. the sensor node. As a result, cycle accurate emulators are hard to maintain, extend and debug. Concluding, emulation is not a recommend choice for a system like a sensor network, which offers variety of sensor platforms and can scale to thousands of sensor nodes.

Our main contribution in this article is that we bridge the gap between scalable but abstract simulation and cycle accurate emulation. We provide near cycle accurate timing combined with the scalability, flexibility and portability of simulation. Hence, TimeTOSSIM is capable of combining all the essential evaluation properties like time accuracy, energy prediction, speed and scalability on a single evaluation platform.

III. TOSSIM

TOSSIM [6] is a TinyOS based simulator for sensor networks scalable to thousands of network nodes. Although its simulation core is discrete event based, it significantly differs from the traditional simulators in two respects. First, it compiles directly from the platform dependent source code into the simulation infrastructure by adding an alternative compilation target. Hence, TOSSIM does not require the algorithms to be implemented separately for simulation and hardware platform. Second, unlike traditional simulation, TOSSIM only replaces low level device drivers with simulation wrappers while rest of the code including the operating system (OS) is executed in simulation. Traditional simulation abstracts from OS level details by providing abstract simulation models usually implemented in high level languages like Java or C++.

The fact that TOSSIM compiles directly from the platform dependent source-code makes it more expressive and realistic than SensorSim and SENS. TOSSIM only needs 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 [13]. TOSSIM also benefits from the event-based, component oriented programming model of TinyOS by translating the hardware interrupts into discrete simulator events which drive the simulation.

One of the extensions of TOSSIM is PowerTOSSIM [4], which evaluates the energy consumption to predict the life time of a sensor node. It extends TOSSIM by adding a new *PowerState* module that records energy-state transitions of each hardware component. For the CPU, a mapping technique is used at basic-block granularity to determine the number of clock-cycles for which the CPU remained active.

[2] Throughout this article the term "simulation" is used to represent discrete event based simulation and the term "emulation" is used to represent cycle accurate instruction set simulation.

PowerTOSSIM relies on an abstract simulation clock which is adjusted at the start of every new simulator-event to determine the time for which a hardware device remained in each of its power-states. Therefore, the simulation results show deviations of up to 13% from real power measurements. Moreover, rigorous testing of the CPU profiling technique has revealed an error of 5700% in calculating CPU active time [5]. PowerTOSSIM uses *Postmortem analysis* (i.e. offline analysis after simulation) to calculate the energy consumed by sensor network during simulation. Such offline analysis can only contribute to determine the power consumption. It is unable to assist the developers in observing the behavior of applications in response to the unreliability caused by decaying and expiring nodes during the simulation.

The approach presented in this article is partly similar to PowerTOSSIM's CPU profiling technique. However, we generalize it to perform online clock advancement and dynamic event queue adaptation compared to offline modeling in PowerTOSSIM. By doing so, we also provide a deep insight into the much important timing and interrupt properties of applications, operating systems and hardware components. Furthermore, we provide a more fine grained instrumentation level than PowerTOSSIM and features - such as energy models - can be easily derived from the detailed timing model presented in this article. Similarly, TimeTOSSIM outperforms Avrora in terms of speed and scalability while maintaining highly precise time accuracy.

IV. Modeling Execution Time

Classic simulation models the behavior of a system at event granularity. It translates all events, e.g. interrupts and tasks in TinyOS into discrete simulator events. Events are executed one after another. Thus, time in simulation is handled discretely; at the beginning of an event the simulation time is set to the execution time of the event and remains unadjusted throughout the event execution. Therefore, events in simulation take zero simulation time. However, in real life events have an execution time and may interrupt, interfere or delay each other, resulting in different execution and completion order compared to simulation. Under peak loads, this may even load to event misses on interrupts and tasks. Summarizing, simulation only contributes to testify the algorithmic functionality of an application. However, due to the lack of time accuracy in modeling a system, false-positives about the performance of applications are inevitable in simulation.

A. Simulation Clock Incrementation

We resolve timing discrepancy of sensor network simulation by enabling TOSSIM's simulation to track the system time during event execution. Our proposed solution determines the execution time (clock-cycles) of each source-code line being executed inside a simulator-event and then increments the simulation time accordingly. The underlying technique is to automate the mapping between simulation source-code and the platform specific executable. This is only possible when nearly identical application and operating system code is executed in simulation and on the hardware platform, which is typically the case in sensor network operating systems. Such a mapping enables us to

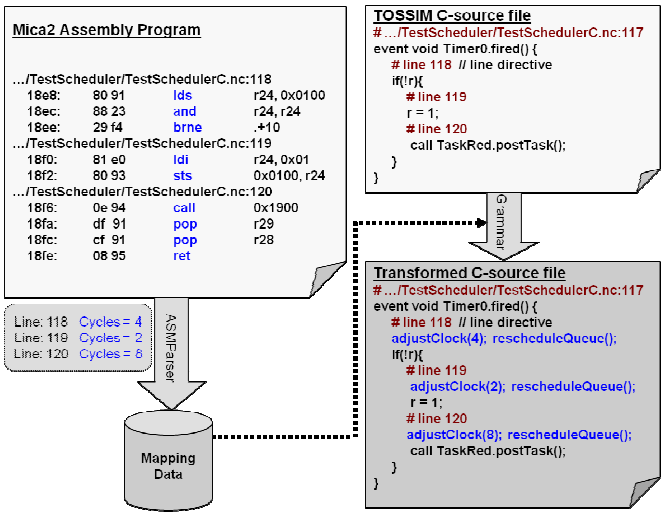


Figure 1: Source-code mapping and instrumentation

identify the processor instructions corresponding to a source-code line. From the respective processor data-sheet we next retrieve the number of cycles consumed by each instruction and therefore can compute the time to execute each source-code line on the sensor node platform. Figure 1 illustrates this process.

The code mapping technique is particularly suited for embedded CPUs (such as in sensor nodes) employing sequential instruction execution without any pipelining and caching strategies. For such platforms, the execution time of a binary instruction is static and can be modeled without interpreting each individual instruction.

B. Event Queue Adaptation

Tracking system time during event execution may result in overlapping events and only helps in determining the execution time of each event separately. However, the overall timing and interrupt behavior of an application still remains undetermined. For example, in TinyOS tasks are executed sequentially and therefore can delay each other's execution. However, interrupts are executed immediately and delay the execution of any currently active task. Under peak loads, interrupts and tasks are even dropped when their corresponding queues overflow. By extending the simulation queue with priorities representing tasks and the various interrupt levels, we can easily model such a behavior.

We assign execution priorities to different events. As events in the TOSSIM event-queue represent hardware interrupts or TinyOS tasks, it is possible to determine the type of an event and its execution priority from the processor data-sheets. Correct ordering of events can be achieved by visiting the event queue at the start of every source line after incrementing the simulation clock. The idea is to reschedule events with lower priority, execute events with higher priority immediately, and thereby delay or interrupt the execution of currently active events. Overall, these timing and rescheduling extensions to simulation models give a detailed insight into the performance of a system without the need for complex emulators or test-beds.

C. Static and Manual Code Mapping

For simulation, TOSSIM replaces low-level device drivers on the hardware presentation layer[3] (HPL) of TinyOS with simulation wrappers. Therefore, simulation and platform specific code differ on the hardware presentation layer and the presented code automated instrumentation techniques does not apply for low-level device drivers. However, these layers are commonly quite slim. In this sub-section we present the implementation of two techniques to enable accurate timing even in these code sections: (1) static code mapping and (2) manual code mapping.

We apply static code mapping in simple device drivers that do not contain any conditional statements and therefore execute in a constant number of cycles. For example, we applied this approach to model the time required to enable or disable pins of the microcontroller, timers and radio into TimeTOSSIM. Although this process does not introduce inaccuracies in terms of cycles, it is not as fine granular as the commonly used source line granularity. Thus, interrupts may get delayed by a number of cycles. However, HPL code sections are usually 10 to 100 cycles and therefore executed in a couple of micro seconds.

Likewise, to model code sections that were extended for simulation in TOSSIM and to address that some code in the HPL layer may have a higher complexity, we use manual mapping. Based on the fact that the simulation model needs to reassemble the functionality of the device specific code, we manually map sections with equal functionality and instrument the simulation code with the corresponding number of cycles. We applied this approach to the TOSSIM scheduler. Its implementation strongly differs from the device specific one, but it reassembles the same functionality and therefore can be easily instrumented manually.

It is important to highlight that this automated mapping and simulation instrumentation process is performed offline i.e. during compilation of the simulation-code to enable time accurate simulation execution. Please note that this process is neither bound to a certain hardware platform nor to TinyOS. Therefore, it can easily be applied to any other sensor node architecture and operating system. We use TOSSIM for our prototype implementation because it compiles directly from the platform dependent source-code into the simulation infrastructure. Moreover, our choice is influenced by the fact that TOSSIM simulates TinyOS, which is the de-facto standard operating system for sensor networks.

V. From Timing To Energy

Timing is the preliminary requirement for modeling energy consumption of a device. Once we have the knowledge about different states and the duration a device has been in each state, we can easily determine the energy drainage of that device using accurate energy models. In this section we discuss the key requirements for predicting energy consumption of sensor networks. We also show that TimeTOSSIM is an ideal platform that fulfills all these requirements and can easily be extended to provide fine-grained energy estimates.

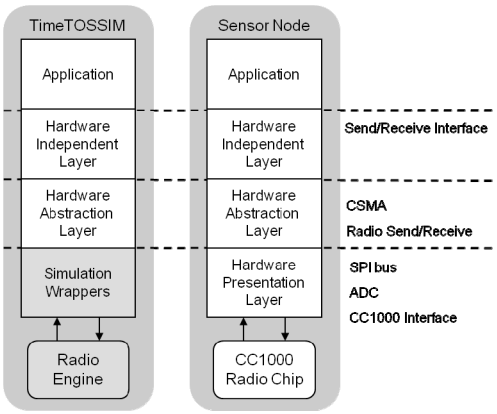


Figure 2: Implementation of CC1000 radio chip for TimeTOSSIM

A. Modeling Energy-states

To model the energy-states of hardware we benefit from the TOSSIM's simulation infrastructure. It only replaces low level device drivers at the HPL layer with simulation wrappers to enable simulation. Rest of the code remains same for simulation and real hardware. These simulation wrappers capture all the events triggered by higher level code to change device's energy-state. For example, when sensors are activated to read data and deactivated to save energy or LEDs are turned on and off. The only exception where we are unable to capture the energy-states and transitions is the radio chip. The reason is that in the case of radio TOSSIM abstracts from the original platform dependant code and provides a complete simulation implementation for modeling radio propagation.

To overcome this limitation we extended TimeTOSSIM to provide support for the simulation of CC1000 radio chip[4]. Our implementation is based on the original TOSSIM approach i.e. to utilize maximum platform dependant code and abstract from the original implementation at the lowest possible level (i.e. HPL layer). We use the original TinyOS code for CC1000 radio chip and provide our own simulation wrappers at the HPL layer, as shown in Figure 2. In doing so, we regain our granularity by enabling automated mapping and instrumentation of communication related source-code and capture each and every state transition of the radio chip. Hence, it enables us to accurately predict the energy consumption of the radio chip. Our prototype implementation of radio chip also facilitates us in evaluating TimeTOSSIM, as CC1000 is supported by publically available emulation platforms such as Avrora.

[3] TinyOS has a platform abstraction architecture consisting three layers; Hardware Independent Layer (HIL), Hardware Abstraction Layer (HAL), and Hardware Presentation Layer (HPL). The code at HIL and HAL level is platform independent while only a small portion of code lies at HPL level that provides access to the original hardware and is platform dependent.

[4] Currently, TOSSIM provides an abstract implementation of packet level CC2420 radio chip present in MicaZ and TelosB sensor node platforms.

B. Determining Time per Energy-state

Our instrumentation approach increments the simulation clock before the execution of each source-code line. Therefore, it allows us to determine the duration a device spends each of its energy-states at source-code line granularity. Hence, TimeTOSSIM is capable of predicting the energy consumption at much higher granularity than PowerTOSSIM, which relies on the abstract simulation clock of TOSSIM. On the other hand, TimeTOSSIM it is inherently much more sclable and faster than AEON as shown by our evaluaiton results in section 5.

Application	Instrumentation level and accuracy in %			
	Source-line (No optimizations)	Source-line (Space related optimizations)	Basic block	Function
Blink	99.69	99.63	99.79	98.93
BlinkTask	99.73	99.55	99.73	98.84
CntToLeds	99.69	99.64	99.69	98.97
TestScheduler	87.7	81.44	87.7	NA

Table 3: Time accuracy of different standard TinyOS applications achieved in TimeTOSSIM (compared to Avrora) for different instrumentation granularities and compiler optimizations.

C. Energy models

Energy models can be created for each hardware component by consulting respective data-sheets, which provide current estimates at clock-cycle granularity. A simple application level benchmarking can be used to validate energy models derived from the data-sheets. In short, this process does not require any complex low level benchmarking. Due to our previous work on AEON's [5] energy models, we benefit from the availability of accurate energy models for Mica2 sensor node.

VI. Evaluation and Performance Comparison

In this section we thoroughly evaluate TimeTOSSIM both from performance and accuracy perspectives. We compare the accuracy of TimeTOSSIM with the cycle accurate emulator, Avrora, and the speed of TimeTOSSIM with the original TOSSIM implementation. The evaluation is based on three types of benchmarks: (1) micro benchmarks, (2) evaluation of static and manual instrumentation, and (3) macro benchmarks.

Component	Accuracy	Minimum Granularity
Leds	100%	47 clock-cycles
Timers	100%	Same as emulation

Table 1: Accuracy of different hardware components

Code-block	Clock drift in cycles
Statements	0
While loops	0
Do loops	0
For loops	+4
Nested while	-1
If-else clause	0
Switch clause	15

Table 2: Simulation clock drifts for different code blocks

In our micro-benchmarks we evaluated the time accuracy of different types of mapped code-blocks (loops, control-structures etc.) independently from each other to give a deep insight into the timing properties of source-code. Table 2 shows that we achieve 100% accuracy in the case of simple statements (variable initialization, assignment etc.), do and while loops, and if-else clauses. However, in case of for loops, switch clauses and nested-while loops, the simulation clock drifts by few clock cycles. The reason is that our mapping technique calculates the execution time of a source line by counting the number of cycles required by the corresponding assembly instructions without interpreting them. Thus, in the case of switch clause and for loop, the total number of assembly instructions being executed depends on the current value of the decision variable and the current iteration of the loop, respectively. For example, the initialization of loop variable happens only before the first iteration but our mapping technique, as its abstracts from instruction interpretation, counts the corresponding number of cycles for each iteration of the loop. Patching the corresponding compiler to add further code annotations can remove this inaccuracy, but here we deliberately compromise the inaccuracy over the complexity of our approach: Firstly, because the simulation clock gets re-synchronized at the start of every new simulator event (a timer fire or hardware interrupt). Secondly, as our macro-benchmarks later in this section show, the overall accuracy of TimeTOSSIM is only slightly influenced by these inaccuracies and we still achieve beyond 99% time accuracy for most applications.

After evaluating the accuracy of TimeTOSSIM regarding programming structures, we testify our static and manual mapping technique. We evaluate the time accuracy of different operations performed on the most frequently used on-chip hardware components: LEDs and timers. Currently, apart from instruction execution, Avrora only emulates these two on-chip hardware components correctly for TinyOS-2 based applications. LEDs are the simplest example of a hardware component attached to a micro-controller pin. Evaluating LED operations fully tests the functionality of our approach because any operation on LEDs involves automatic, static and manual code mapping and instrumentation, as all hardware components are accessed via HPL layer of TinyOS. Profiling of the low level LED component of TinyOS shows that the minimum granularity (maximum clock advancement) in LED operations is 47 clock cycles (6 microseconds). Similar to the access to microcontroller pins, we evaluated the accuracy of Timer components in TimeTOSSIM. Our results show that we achieve the same accuracy and granularity as emulation (see Table 1).

In our macro-benchmarks, we test the time accuracy and scalability of TimeTOSSIM at application level. Table 3 shows the accuracy level we achieve with different off-the-shelf applications. We compare the simulation traces of TimeTOSSIM with Avrora. Our measured results show beyond 99% time accuracy for most of the applications. Additionally, we use the *TestScheduler* application to stress-test the accuracy of TimeTOSSIM from the worst-case point of view. The *TestScheduler* application is a sanity check for TinyOS scheduler and has no hardware events that could re-synchronize the simulation-clock. Nonetheless, we still achieve 88% accuracy. This level of accuracy is independent from the compiler optimizations of the sensor

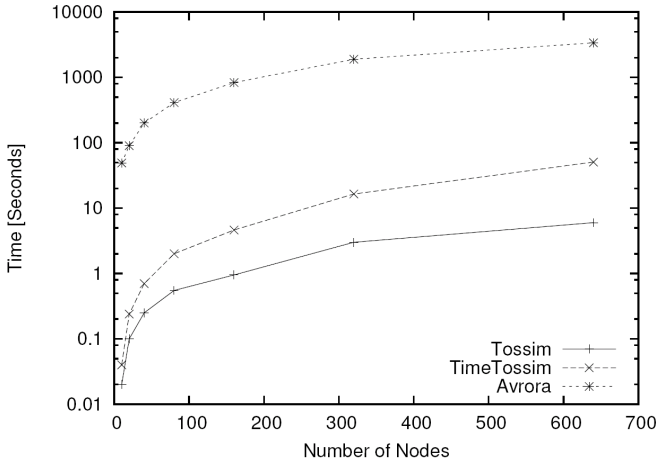


Figure 3: Scalability comparison of different sensor network simulators and emulators. Please note the logarithmic scale on y-axis

node application. Basic-block level instrumentation achieves similar timing results as source line instrumentation. However, it has a lower granularity and therefore may delay interrupts under high load. Function level instrumentation results in even less accurate modeling compared to basic-block and source line granularity. However, basic block and function level granularity result in less code instrumentation and therefore increase the simulation speed of TimeTOSSIM.

After evaluating the accuracy achieved with TimeTOSSIM, we evaluate the performance of TimeTOSSIM when compared to TOSSIM and Avrora. All experimental results discussed in this section were executed on a customary end-user machine, a Pentium IV with 3 GHz clock frequency and 1GB of RAM. Our evaluations show that TimeTOSSIM when using instrumentation on source line granularity is up to 10 times slower than TOSSIM while being more than 100 times faster than Avrora, especially when using large numbers of nodes (see Figure 3). For single node simulations the overhead of TimeTOSSIM is reduced to a factor of 1 to 6, as the number of adaptations of the event queue gets reduced drastically.

In comparison to PowerTOSSIM, TimeTOSSIM shows a similar performance overhead. Thus, PowerTOSSIM and TimeTOSSIM need about the same time for simulation. However, TimeTOSSIM provides much more functionality and features like fine gained energy modeling can be easily added to TimeTOSSIM based on the derived cycle counts.

Concluding the performance and accuracy evaluation, it can be said that TimeTOSSIM, though slower than TOSSIM, provides a very accurate simulation of sensor nodes. Although code instrumentation on source code line granularity introduces some inaccuracies, their overall impact seems to be small. Furthermore, the fact that instrumentation of source lines does not require any special compiler extensions ensures that TimeTOSSIM can easily be ported to various sensor node platforms and operating systems.

VII. Conclusion and Future Work

Energy estimation is a crucial characteristic of evaluating sensor networks and their applications. In this article we presented automated instrumentation of simulation models to enable time accurate simulation – a prerequisite for accurate energy estimation. Our evaluation results have shown that automated instrumentation on source-code line granularity provides beyond 99% accuracy for typical sensor network applications while offering much higher performance, scalability and easy portability compared to today's emulators. Finally, we illustrated that our approach leads to determine the energy-states and duty cycle of each hardware component of a sensor node. As a result, we can determine the energy consumption of each hardware component, and in due course, of the whole sensor network.

Although our approach promises highly accurate energy estimates, a true evaluation can only be performed after the integration of energy models to TimeTOSSIM. Currently, we are adding energy models of different sensor platoforms (e.g. Mica2 and MicaZ) and features like shutting down expired nodes in TimeTOSSIM simulation.

References

[1] Polley, J., Blazakis, D. McGee, J., Rusk, D. Baras, J.S. "ATEMU: A Fine-Grained Sensor Network Simulator" In Proceedings of IEEE SeCon, Santa Clara, USA, 2004.

[2] Shah R.C, Rabaey J.M, "Energy aware routing for low energy ad hoc sensor networks", In Proceedings IEEE WCNC, Orlando, USA, 2002

[3] Tijs van Dam, Koen Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks", In Proceedings Of ACM Sensys, Los Angeles, USA, 2003 .

[4] Shnayder, V., Hempstead, M., Chen, B., Allen, G. W., and Welsh, M. "Simulating the Power Consumption of Large-Scale Sensor Network Applications," In Proceedings of ACM SenSys, Nov. 2004.

[5] Landsiedel, O., Wehrle, K., and Gotz, S., "Accurate Prediction of Power Consumption in Sensor Networks," In Proceedings of IEEE EmNetS-II, Sydney, Australia, 2005.

[6] Levis, P., Lee, N., Welsh, M., and Culler, D. "TOSSIM: accurate and scalable simulation of entire tinyOS applications," In Proceedings of ACM SenSys, New York, USA, 2003.

[7] S. Park, A. Savvides, and M. B. Srivastava, "SensorSim: a simulation framework for sensor networks," In Proceedings of ACM MSWiM, New York, 2000.

[8] S. Sundresh, W. Kim, and G. Agha, "SENS: A Sensor, Environment and Network Simulator," In Proceedings of Annual Simulation Symposium, Washington, USA, 2004.

[9] Titzer, B.L. Lee, D.K. Palsberg, J. "Avrora: scalable sensor network simulation with precise timing" , In Proceedings of ACM/IEEE IPSN, Los Angeles, USA, 2005.

[10] Landsiedel, O., Alizai, H., and Wehrle, K. "When Timing Matters: Enabling Time Accurate and Scalable Simulation of Sensor Network Applications", In Proceedings of ACM/IEEE IPSN, Washington, USA, 2008.

[11] Chintalapudi, K., Fu, T., Paek, J., Kothari, N., Rangwala, S., Caffrey, J., Govindan, R., Johnson, E., and Masri, S., "Monitoring Civil Structures with a Wireless Sensor Network", In Proceedings of IEEE Internet Computing Journal, 2006.

[12] Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., and Welsh, M., "Fidelity and yield in a volcano monitoring sensor network", In Proceedings of the 7th Symposium on Operating Systems Design and Implementation , Berkley, USA, 2006.

[13] V.Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, D. Culler, " Flexible hardware abstraction for wireless sensor networks", In Proceeding of European Workshop on Wireless Sensor Networks, 2005