IT Licentiate theses
2009-001

Detailed Simulation of Heterogeneous Wireless Sensor Networks

JOAKIM ERIKSSON

UPPSALA UNIVERSITY
Department of Information Technology

UPPSALA
UNIVERSITET

Detailed Simulation of Heterogeneous Wireless Sensor Networks

BY

JOAKIM ERIKSSON

April 2009

DIVISION OF COMPUTER SYSTEMS
DEPARTMENT OF INFORMATION TECHNOLOGY
UPPSALA UNIVERSITY
UPPSALA
SWEDEN

Dissertation for the degree of Licentiate of Philosophy in Computer Science
at Uppsala University 2009

Detailed Simulation of Heterogeneous Wireless Sensor Networks

Joakim Eriksson
joakime@sics.se

Division of Computer Systems
Department of Information Technology
Uppsala University
Box 337
SE-751 05 Uppsala
Sweden

http://www.it.uu.se/

ISSN 1404-5117
Printed by the Department of Information Technology, Uppsala University, Sweden

Abstract

Wireless sensor networks consist of many small nodes. Each node has a microprocessor, a radio chip, some sensors, and is usually battery powered which limits network lifetime. Applications of wireless sensor networks range from environmental monitoring and health-care to industrial automation and military surveillance.

Since the nodes are battery powered and communication consumes more power than computation much of the research focuses on power efficient communication. One of the problems is however to measure the power consumption and communication quality.

Simulation of sensor networks can greatly increase development speed and also be used for evaluating power consumption as well as communication quality. For application and system development, simulators can be used to test code functionality and find software bugs. For research experiments, simulators can be used to get easier access to fine grained results than corresponding real world experiments. One problem with simulators is that it is hard to show that a simulation experiment corresponds well with a similar real world experiment.

This thesis studies how detailed simulation of wireless sensor networks can be validated for power profiling accuracy and shows that detailed, emulation based simulation is a useful tool for white-box interoperability testing. Both power profiling and interoperability testing represent important topics in today's wireless sensor network research and development.

The results and main contributions of the thesis are the simulation platform COOJA/MSPSim and that we show that three low-power MAC-protocol experiments performed in our simulator COOJA/MSPSim correspond well with experiments performed in our testbed. We also show that using COOJA/MSPSim any software running in the simulation can be power profiled, and that COOJA/MSPSim can be used for white-box interoperability testing.

To Oliver, Viktor and Agneta

Acknowledgements

I would like to thank my advisor and colleague Thiemo Voigt for all inspiration and encouragement during the work with the thesis. I would also like to thank my co-advisor Mats Björkman for good advices, and discussions. I am also very grateful to Per Gunningberg, co-advisor, inspiring teacher in many university courses that I took, and the one who helped me get the opportunity to work at SICS.

Many thanks also go to my colleagues at SICS, including Adam Dunkels, Niclas Finne, Fredrik Österlind - main developer of COOJA, Nicolas Tsiftes, Olle Olsson, Sverker Janson, Zhitao He, Daniel Gillblad, and many many more.

I am grateful to the SAVE-IT industrial Ph.D. program at MdH in which I have been an industrial Ph.D. student during the work with this thesis. Many thanks to ABB Corporate Research in Västerås that have given me the opportunity to participate in industrial research and development projects. Many thanks to Tomas Lennvall, Lennart Balgård and Jonas Neander at ABB Corporate Research in Västerås for interesting discussions. I am also grateful to the former ABB CRC employees Stefan Svensson, now at Acreo, and Martin Strand, now at ABB Power Systems. Thanks also go to Jan-Erik Frej and Mikael Gidlund, both ABB CRC, for interesting discussions about research directions, projects, and relations between Swedish industry and research organisations.

Finally, many thanks to my wife Agneta and my two sons Oliver and Viktor for making life more fun and lively.

The work in this thesis is in part supported by VINNOVA, FMV, The CONET FP7 Network of Excellence, SAVE-IT/KKS, and the Uppsala VINN Excellence Center for Wireless Sensor Networks (WISENET). The Swedish Institute of Computer Science, SICS AB, is sponsored by TeliaSonera, Ericsson, Saab Systems, FMV, Green Cargo, ABB and Bombardier Transportation.

Included Papers

This thesis is composed of the following papers. In the thesis the papers will be referred to as papers A to C.

A Niclas Finne, Joakim Eriksson, Adam Dunkels and Thiemo Voigt. Experiences from two sensor network deployments: self-monitoring and self-configuration keys to success. *Proceedings of Wired/Wireless Internet Communications: 6th International Conference, WWIC 2008.* 28-30 May 2008, Tampere, Finland.

B Joakim Eriksson, Fredrik Österlind, Niclas Finne, Adam Dunkels, Thiemo Voigt and Nicolas Tsiftes. Accurate, network-scale power profiling for sensor network simulators. *Proceedings of EWSN 2009, the 6th European Conference on Wireless Sensor Networks.* 11-13 February 2009, Cork, Ireland.

C Joakim Eriksson, Fredrik Österlind, Thiemo Voigt, Adam Dunkels, Niclas Finne, Nicolas Tsiftes, Robert Sauter and Pedro José Marrón. COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks. *Proceedings of SIMUTools 2009, the 2nd International Conference on Simulation Tools and Techniques.* 2-6 March 2009, Rome, Italy.

Selected Papers not Included in the Thesis

1 Fredrik Österlind, Adam Dunkels, Thiemo Voigt, Nicolas Tsiftes, Joakim Eriksson and Niclas Finne. Sensornet checkpointing: enabling repeatability in testbeds and realism in simulations. *Proceedings of EWSN 2009, the 6th European Conference on Wireless Sensor Networks.* 11-13 February 2009, Cork, Ireland.

2 Joakim Eriksson, Fredrik Österlind, Niclas Finne, Adam Dunkels, Thiemo Voigt. Accurate power profiling for sensor network simulators. *8th Scandinavian Workshop on Wireless Ad-hoc and Sensor Networks.* 7-8 May 2008, Stockholm, Sweden.

3 Joakim Eriksson, Adam Dunkels, Niclas Finne, Fredrik Österlind, Thiemo Voigt and Nicolas Tsiftes. Demo abstract: MSPsim - an extensible simulator for MSP430-equipped sensor boards. *EWSN 2008, 5th European Conference on Wireless Sensor Networks.* 30 Jan - 1 Feb 2008, Bologna, Italy.

4 Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne and Thiemo Voigt. Cross-level simulation in COOJA. *European Conference on Wireless Sensor Networks, EWSN 07.* January 2007, Delft, The Netherlands.

5 Fredrik Österlind, Erik Pramsten, Dainel Roberthson, Joakim Eriksson, Niclas Finne and Thiemo Voigt. Integrating Building Automation Systems and Wireless Sensor Networks. *12th IEEE Conference on Emerging Technologies and Factory Automation.* 25-28 September 2007, Patras, Greece.

6 Thiemo Voigt, Fredik Österlind, Niclas Finne, Nicolas Tsiftes, Zhitao He, Joakim Eriksson, Adam Dunkels, Ulf Båmstedt, Jochen Schiller and Klas Hjort. Sensor Networking in Aquatic Environments - Experiences and New Challenges. *Second IEEE International Workshop on Practical Issues in Building Sensor Network Applications.* 15-18 Oct 2007, Dublin, Ireland.

7 Adam Dunkels, Niclas Finne, Joakim Eriksson and Thiemo Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems, SenSys 2006.* November 2006, Boulder, Colorado, USA.

Contents

Part I

Thesis

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Wireless sensor networks consist of small autonomous nodes. Each node has a small microprocessor, a radio chip, some sensors, and is usually battery powered which limits network lifetime. Applications of wireless sensor networks range from environmental monitoring and health-care to industrial automation and military surveillance.

To make sensor networks a technology that can be used in a large number of application areas it is important that the network nodes are low-cost. Therefore, the network nodes are resource constrained and a typical microprocessor is limited to a few MHz in processing speed, has a few kbyte of RAM and some tens of kbyte storage for programs. The radio chip typically has a communication range of up to a few hundred meters outdoors and less indoors, and a few hundred kbit/s of communication speed.

Since the nodes are resource constrained, common operating systems, communication stacks and development tools cannot be used. This has lead to the development of small operating systems specifically designed for resource constrained networked embedded systems. The most well known operating systems are Contiki OS [6], TinyOS [14], Mantis OS [1] and SOS [11]. They all have tools such as simulators, communication stacks, and are ported to several hardware platforms.

The combination of resource constraints and lack of mature development tools make programming wireless sensor network a challenge. Sensor networks are typically expected to last several years, and since the sensor nodes are battery powered, another challenge is to develop power efficient communication protocols and applications.

The focus of this thesis is to develop and study detailed wireless sensor network simulators and to investigate how they can be used in some application areas such as power profiling and interoperability testing. The thesis consists of three published papers; one motivates the work and the other

two describe and evaluate the simulator.

1.2 Wireless Sensor Network Simulators

A simulator is a software tool that imitates selected parts of the behavior of the real world and is normally used as a tool for research and development. Simulators exist for a wide variety of fields including physics, biology, economics, and computer systems. Depending on the intended usage of the simulator, different parts of the real-world system are modeled and imitated. The parts that are modeled can also be of varying abstraction level. A wireless sensor network simulator imitates the wireless network media - the air - and the nodes in the network. Some sensor network simulators have a detailed model of the wireless media including effects of obstacles between nodes, while other simulators have a more abstract model. In this thesis the focus is mainly on simulators that have detailed node models.

During development of applications, systems and protocols for sensor networks, a large part of the time is spent compiling, testing, debugging and evaluating. Either a network of real sensor nodes or a wireless sensor network simulator is used during the testing, debugging and evaluation. In the research group I am working in, the Networked Embedded Systems group, we develop the sensor network operating system Contiki OS [6] including IPv6 communication stacks [8] and tools such as COOJA/MSPSim. By using COOJA/MSPSim we have found both alignment problems and compiler bugs during development and porting of communication stacks and other software. COOJA/MSPSim is also used for our automatic nightly tests of the Contiki OS software library.

When using simulators for research experiments, the evaluation of the experiment can be much less time consuming and information about nodes and their communication can be measured at a high level of detail. It is also possible to repeat the exact same experiment several times, something more or less impossible when evaluating an experiment on real sensor networks. In simulation it is also possible to control most aspects of the environment such as number of nodes, mobility, packet loss ratio, etc.

Sensor network simulators can be used for a wide range of tasks. Some of these are briefly introduced below.

Application and System Development

When developing applications or systems, simulators can be used as a tool for testing the complete behavior of the system. By executing the application or system in a simulator with support for debugging it is possible to find software bugs before deploying the application on real nodes. Installing, executing and debugging using a simulator can save a substantial amount of time compared to using real nodes since it takes much less time to install and

execute in a simulator. It is also easier to get detailed information about internal states and other debugging related information of the simulated nodes than it is on real nodes.

Evaluation of new Communication Protocols

When developing new types of communication protocols for wireless sensor networks it is necessary to evaluate some aspects of the protocol such as energy consumption, throughput, reliability, etc. under varying conditions. Simulators provide detailed evaluations as well as control and variation of the conditions in the simulated environment. Evaluating in a simulator is typically both easier and faster than on a real world deployment or testbed.

Power Profiling of Applications

Many sensor network applications have high lifetime requirements. Using simulators it is possible to get an expectation of how long the batteries in the nodes will last. It is, however, important that the simulator has a fine-grained model of the nodes and that it is accurate in its power consumption predictions.

1.2.1 Types of Sensor Network Simulators

Simulations can be performed at several different abstraction levels, from generic simulation where only the most important aspects are simulated to high detail simulations where many details are simulated. In this thesis I classify the available simulators into the three different categories: generic network simulators, code level simulators, and firmware level simulators as shortly described below.

Generic Network Simulators

Generic network simulators simulate systems with a focus on networking aspects. The user of the simulator typically writes the simulation application in a high level language different from the one used for the real sensor network. Since the focus of the simulation is on networking the simulator typically provides detailed simulation of the radio medium, but less detailed simulation of the nodes.

The application or protocol code is usually written in the same programming language as the simulator itself. Most network simulators provide implementations of network stacks, MAC protocols, radio medium simulation, etc. Generic network simulators are useful for evaluating new types of communication protocols, but less useful for interoperability testing or finding software bugs in deployable code, since the code executed is not the same as on real nodes.

Code Level Simulators

Code level simulators use the same code in simulation as in real sensor network nodes. The code is compiled for the machine that is running the simulator, typically a PC workstation that is magnitudes faster than the sensor node. Typically code level simulators are operating system specific since they need to replace driver code for the sensors and radio chips available on the node with driver code that instead have hooks into the simulator.

Code level simulators provide implementation of the network stacks that are available for the specific operating system since the code is the same as on real nodes. The simulators also provide simulation of the radio medium and in some cases simulation of sensors, etc. Code level simulators can be used for finding some types of bugs in deployable code, logical error, buffer overrun errors, etc. Bugs that relate to timing, CPU architecture or low-level drivers are usually hard to find using code level simulators since they do not simulate the hardware in detail. Code level simulators can be used for interoperability testing, but since they are operating system specific the tests will be limited to communication stacks within the same operating system.

Firmware Level Simulators

These simulators are based on emulation of the sensor nodes and the software that runs in the simulator is the actual firmware that can be deployed in the real sensor network. This approach gives the highest level of detail in the simulation and enables accurate execution statistics. This type of simulation provides emulation of microprocessor, radio chip and other peripherals and simulation of radio medium. Due to the high level of detail provided by firmware level simulators, they are usually slower than code level or generic network simulators.

Firmware level of simulation is useful when timing-sensitive software such as MAC protocols or low level device drivers are tested, debugged and evaluated. Most types of bugs can be found since the target CPU is emulated with its specific properties such as word alignment and memory limitations.

While simulators are useful in many cases it is important to understand the limitations of the simulator and the quality of evaluation results that are the output of the simulations. Some aspects of the real world such as weather effects or execution speed of nodes might not be simulated, and if these aspects affect the simulated sensor network the results might not be realistic and cannot be trusted.

1.3 The COOJA/MSPSim Simulator

Most of the work in this thesis is focused on COOJA/MSPSim, our sensor network simulator and sensor node emulator. COOJA/MSPSim is a combination of two separate tools into one cross-level wireless sensor network simulator. These tools are introduced in the following sections.

1.3.1 The MSPSim Emulator

MSPSim [9] is a Java-based instruction level emulator of the MSP430 microprocessor series. It emulates complete sensor networking platforms such as the Tmote Sky [24] and ESB/2 [26]. MSPSim provides detailed simulation with accurate timing and strong debugging support.

MSPSim combines cycle accurate interpretation of CPU instructions with a discrete-event based simulation of all other components, both internal and external. MSPSim uses an event-based execution kernel that enables accurate timing while keeping the host processor utilization as low as possible.

The emulator provides a programming interface for integration with network simulators such as COOJA. In addition, the emulator can be extended with new mote types through a mote interface and I/O interfaces that correspond to the MSP430 I/O ports and serial communication ports.

MSPSim provides both debugging capabilities such as break points, watches, logging, and single stepping as well as statistics about the operating modes of the emulated components, statistics such as how much time the CPU has consumed in the different low-power modes. All features and information can be accessed either via a command line interface, or via the integration programming interfaces.

1.3.2 The COOJA Network Simulator

COOJA [22] is a flexible Java-based simulator initially designed for simulating networks of sensor nodes running the Contiki operating system. COOJA simulates networks of sensor nodes where each node can be of a different type; differing not only in on-board software, but also in the simulated hardware. COOJA is flexible in that many parts of the simulator can be easily replaced or extended with additional functionality.

COOJA can execute Contiki programs either by running the program code compiled for the PC workstation CPU, or by running code compiled for the sensor node in MSPSim. COOJA can also run nodes programmed in Java. All different approaches have advantages as well as disadvantages. Java-based nodes enable much faster simulations but do not run deployable code. Emulating nodes allows control and retrieval of more fine-grained execution details compared to Java-based nodes or nodes running PC host

code. Combining the different levels in the same simulation gives both a fast simulation as well as fine-grained execution details on selected nodes.

We have extended COOJA with support for other operating systems such as TinyOS and for emulating nodes based on the Atmel AVR microcontroller.

Chapter 2

Challenges and Research Questions

There are many challenges and open research questions in the area of wireless sensor network simulation. Some of the important challenges are introduced in this section together with a discussion on how this thesis addresses the challenges.

2.1 Power Profiling in Simulators

Simulators are useful tools for studying many aspects of sensor networking but without validating the simulator the results are of uncertain quality. Two important aspects to validate are timing and the properties of the radio medium. Timing validation and improvements of timing accuracy have been made for some simulators such as TOSSIM but only for single nodes and not for networked nodes. In this thesis the focus is on validating the timing and power profiling on a network scale and study the accuracy when nodes interact. This is done by comparing testbed results with results from simulation in COOJA/MSPSim. This work is presented in paper B.

Validation of the radio medium and the radio chip is also important and there is some work in this field [3]. Validation of the radio medium is orthogonal to this thesis work on validation of timing accuracy.

2.2 Level of Detail in Simulators

An important topic is how much level of detail should be used in different simulations. Less detail usually give a faster simulation, while more details potentially give more accurate results. Heidemann et al. suggest that simulation of wireless sensor networks needs more detail than for example simulation of wired networks [13]. This is partly due to the characteristics of the wireless medium, but also due to the focus of energy efficiency

which leads to very timing-sensitive energy optimizations. In Paper B we suggest using emulation of sensor nodes to achieve a high level of detail in the simulation, making detailed power profiling and fine-grained white-box interoperability testing possible.

2.3 Interoperability Testing and Simulation of Heterogeneous Networks

Many standards for sensor networking are currently emerging, and interoperability testing is needed to ensure interoperability between different implementations. Typically interoperability tests are performed by either running the communication stack together with stacks that are correctly implemented or with test-scripts that generate packets and study the responses. This will result in a low level of detail when answering if the stack is compliant or not. In the worst case it will only give the answer "not compliant" and no more information. When porting Contiki's 6LoWPAN/IPv6 stack with HC01 compression to the MSP430 microprocessor we used COOJA/MSPSim for interoperability testing. During the tests we found several alignment bugs that caused interoperability problems. These bugs were found by MSPSim's alignment checker.

Since interoperability testing tests different implementations of communication stacks it is important that the simulator supports simulation of heterogeneous sensor networks. COOJA/MSPSim can simulate heterogeneous networks and supports both several operating systems and several CPU architectures.

Paper C studies the usage of emulator based simulators such as COOJA/MSPSim for interoperability testing. By using emulators it is easy to get detailed execution traces and statistics which makes testing and debugging of the communication stack a much easier task.

2.4 Self-Configuration

Many sensor network applications require some form of self-configuration to perform optimally and adapt in a changing environment. There are several proposals for self-configuration architectures for sensor networks such as Generic Role Assignment [10] and Hood [32]. There are also some adaptive and self-configuring communication protocols [2], but the self-configuration mechanism is often specific for the given protocol. A challenge is to develop generic methods that can find self-configuration rules or behavior for a wide range of applications. Some of the candidates for these methods are based on machine learning, and usually require lots of statistical data for the learning process. Instead of or in addition to running these methods on a live sensor network it is possible to use simulators. Using simulators

will both speed-up the learning and make it easy to vary the environment in a systematic way. These methods require a fast and accurate simulation for getting good results. Paper A motivates the need for self-configuration and paper B validates the accuracy of COOJA/MSPSim which is important when learning and evaluating self-configuration policies.

2.5 Contributions and Results

The main contributions and results of this thesis are:

1) The design and implementation of COOJA/MSPSim and evaluation of it as a detailed simulator for sensor network and its accuracy with respect to power profiling of interacting nodes. Earlier work only evaluated simulation accuracy on a per node basis.

2) Evaluation of COOJA/MSPSim as a tool for white-box interoperability testing. Using an emulator-based simulator it is possible to perform interoperability testing and get detailed information from the tests instead of only getting yes/no answers. COOJA/MSPSim allows testing of heterogeneous sensor networks running different node types and different operating systems.

Both MSPSim stand-alone and COOJA/MSPSim have been used by a large number of people during development and evaluation. Some of the uses include: debugging of the mspgcc compiler, automatic nightly tests and algorithm profiling. The simulator is used both in academy and in commercial companies.

Chapter 3

Related Work

The discussion of related work is divided into the sections generic sensor network simulators, code level sensor network simulators and sensor node emulators.

3.1 Generic Sensor Network Simulators

There are many generic sensor network simulators available. Some of them are network simulators with extensions for simulation of sensor networking; others are designed for sensor network simulation. NS-2 is one of the most wide spread network simulators [21]. NS-2 is an object-oriented discrete event based simulator with support for TCP/IP simulation over both wired and wireless links. NS-2 is extensible and new protocols can be implemented in C++ and the simulator can be controlled via OTcl code. There are some extensions related to ad-hoc networks and wireless sensor networks [5]. SensorSim [23] provides additional features for NS-2 such as sensor networking protocols, power consumption models for sensor nodes and battery models.

OMNeT++ [31] is a discrete event simulator with focus on simulation of communication networks. OMNeT++ is more efficient than NS-2 when simulating wireless networks and has several extensions for sensor network simulations including models of energy consumption of communication on IEEE 802.15.4 networks [4]. MiXiM [16] extends OMNeT++ by combining several sensor network related OMNeT++ extensions into one more complete package for simulation of wireless sensor networks.

GloMoSim [33] is another object-oriented discrete event simulator based on the parsec library for parallel execution of the simulation. The focus is on wireless networks and it has support for mobility.

J-Sim [30] is a component-based, real-time process-driven simulator. Real-time process-driven simulation differs from discrete-event simulators in that events consume time corresponding to the time they would consume in the real word. J-Sim is Java-based and thus platform independent.

The generic network simulators can be used for some of the simulation tasks that COOJA/MSPSim can be used for, but none of them are useful for interoperability testing since that requires the same code in simulation as on real nodes. Power profiling can typically not be done either, at least not with the same accuracy as in emulation based simulators [17].

3.2 Code Level Sensor Network Simulators

There are a few simulators that use the same code as is used on real nodes. These simulators are specific for one single operating system since this type of simulator requires low-level drivers for hardware components, etc. to be replaced by the corresponding code in the simulator.

TOSSIM [19] is a code level simulator for detailed simulation of TinyOS based nodes, including simulation of interrupts and bit-based radio communication. TOSSIM provides simulation of radio stack and radio mediums, etc.

POWERTOSSIM [27] is an extension of TOSSIM that adds an energy model. This enables energy profiling of the simulated applications, but since the node model is not executing the same compiled code as real sensor nodes would do, the accuracy is not as good as if the nodes are emulated.

TimeTOSSIM [18] is another extension of TOSSIM that focus on accurate timing without emulation. They improve accuracy by adding instrumentation to the source code. Their evaluation shows that they get 99% timing accuracy compared to the Avrora emulator without emulation. TimeTOSSIM is claimed to be faster than emulation but several times slower than TOSSIM without the instrumentation.

XMOS is Mantis OS's [1] simulator which basically is a port of the Mantis OS to the X86-platform so that multiple nodes can be run on x86 machines.

The code level simulators are more useful than generic network simulators for tasks such as interoperability tests since the code executed in simulation is the same as on real nodes. Code level simulators, however, typically support only one operating system which limits the availability of communication stacks to test. Another problem is that compiling for another CPU architecture might cause a different behavior than on the target sensor node (typically a MSP430 or AVR microcontroller). Using COOJA/MSPSim it is possible to either perform tests on a code level or on emulated nodes executing the same object code that is executed on a real node. COOJA/MSPSim also has support for multiple operating systems so that several different communication stacks can be tested. Code level simulators do not have the same accuracy level as emulated node when it comes to timing and power profiling.

3.3 Sensor Node Emulators

Sensor node emulators provide a very detailed model of the sensor nodes by emulating node hardware such as microprocessor and radio transeiver. The COOJA/MSPSim simulator is a sensor node emulator and has many features in common with other simulators in this category.

Avrora [29] is a cycle-accurate emulator of Atmel AVR-based nodes while MSPSim emulates the Texas Instruments MSP430 architecture. Avrora has a built-in radio medium model and can simulate many nodes. When executing more than one node, Avrora creates a Java thread for each emulated node. Avrora uses selectable synchronization policies for keeping the threads running at the same speed. COOJA/MSPSim handles scheduling of each node explicitly instead of creating threads. While Avrora offers a cycle-accurate simulation of AVR-based nodes, it did not initially have a power profiler. For this purpose, Landsiedel et al. created the power analyzer AEON [17] on top of Avrora. AvroraZ [3] is another extension of Avrora that adds CC2420 and MicaZ emulation to Avrora which makes it possible to execute applications running 802.15.4 based communication protocols (Zigbee [34], 6LoWPAN [20], WirelessHART [12], ISA100 [15]). The same radio chip CC2420 is emulated in COOJA/MSPSim and is used for the Tmote Sky sensor node emulation.

ATEMU [25] is another emulator for Atmel AVR-based sensor nodes. ATEMU is a tick based emulator unlike MSPSim and Avrora that are both event based. Tick based emulators are typically slower since they call emulated components each cycle or tick while event based emulators only call the components when they request to be called. ATEMU supports the Mica2 sensor node which means that it does not have any 802.15.4 support that both AvroraZ and COOJA/MSPSim have.

There are also more limited emulators that can be used for code and algorithm testing. For example the debugger, GDB, usually comes with emulators. The main differences between these emulators and COOJA/MSPSim are that they do not emulate external peripherals and only emulate a single microprocessor.

Chapter 4

Summary of the Papers

The thesis consists of the following papers.

4.1 Paper A: Experiences from Two Sensor Network Deployments — Self-Monitoring and Self-Configuration Keys to Success

Niclas Finne, Joakim Eriksson, Adam Dunkels, Thiemo Voigt. In *Proceedings of WWIC 2008, the 6th International Conference on Wired/Wireless Internet Communications.* Tampere, Finland, May 2008.

Summary. This paper describes two deployments of surveillance applications made together with the Swedish defense unit Markstridsskolan, and two mechanisms for self-monitoring and self-management in sensor network applications. During the deployments we discovered some problems caused by failing hardware of the sensor nodes. Based on this experience we designed two mechanisms for self-monitoring of hardware and software. These mechanisms are presented and evaluated in the paper. The first mechanism probes the hardware for errors by activating different hardware components and measuring all sensors. If any sensor consistently reports unexpected input while activating a component, a hardware error is assumed to be present. The other mechanism monitors the software by using the built-in energy estimator in Contiki OS and compares the results with an application specific power profile. When the estimated energy consumption deviates from the profile a software error is likely. The first mechanism captures the problems detected during the deployments, and the second mechanism captures software errors such as forgetting to turn of the radio after sending/receiving packets.

Contribution. The main contribution of this paper is to highlight the importance of and provide mechanisms for self-monitoring and self-management in wireless sensor networks. The paper serves as background

and motivates the work on detailed and accurate simulation for evaluating power profiling mechanisms as tools for bug detection.

My contribution. I performed both experiments together with Niclas Finne, SICS and Mikael Axelsson, Swedish Defence. I am also co-author of the paper and worked on design and development of the self-monitoring system and experiments.

4.2 Paper B: Accurate Network-Scale Power Profiling for Sensor Network Simulators

Joakim Eriksson, Fredrik Österlind, Niclas Finne, Adam Dunkels, Nicolas Tsiftes, Thiemo Voigt. In *Proceedings of EWSN 2009, the 6th European Conference on Wireless Sensor Networking.* Cork, Ireland, February 2009.

Summary. In this paper we evaluate the accuracy of the combined sensor network simulation tool COOJA/MSPSim that consists of COOJA, a sensor network simulator, and MSPSim, a sensor node emulator. The evaluation is made using Contiki's power profiler as base-line [7]. The power profiler measures time spent in different modes for each chip on a node and calculates power consumption by multiplying time with pre-measured current draw and battery voltage. We compare experimental results measured on real sensor nodes with simulation results for three different MAC protocols. The MAC protocols are of varying types, one is TDMA based (CoReDac) and one is low power probing (LPP), and the final one is based on low power listening (X-MAC). The results of the evaluation indicate that COOJA/MSPSim enables accurate network-scale simulation of the power consumption of sensor networks.

Contribution. The main contribution of this paper is that we evaluate the accuracy of power profiling in simulation by comparing the results from simulation with results from execution on real sensor nodes. We did this evaluation on a network scale which differs from previous efforts that only evaluate single nodes without any communication aspects. Another important contribution is the simulation tool, COOJA/MSPSim that supports accurate power profiling.

My Contribution. I am the main developer of MSPSim and I improved it for better support of power profiling, improved CC2420 radio chip emulation and extended the integration with COOJA. I also made some of the experiments and wrote parts of the paper.

4.3 Paper C: COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks

Joakim Eriksson, Fredrik Österlind, Niclas Finne, Nicolas Tsiftes, Adam Dunkels, Thiemo Voigt, Swedish Institute of Computer Science, SICS.
Robert Sauter, Pedro José Marrón, University of Bonn and Fraunhofer IAIS.
In *Proceedings of SIMUTools 2009, the Second International Conference on Simulation Tools and Techniques.* Rome, Italy, March 2009.

Summary. In this paper we show that COOJA/MSPSim can be used for interoperability tests between different protocol stack implementations in different sensor network operating systems. We also show that the built-in power profiling in MSPSim is as accurate as the Contiki's power profiler and that it can be used for power profiling any application without any power profiling support from the operating system in the node. We evaluate COOJA/MSPSim for use in interoperability tests by adding support for TinyOS and performing basic experiments where nodes based on TinyOS communicate with nodes based on Contiki OS.

Contribution. The main contributions of this paper are that we show that COOJA/MSPSim can be used as an interoperability testing tool and that it accurately evaluates power consumption of the simulated nodes. Interoperability testing in COOJA/MSPSim gives the tester much more detailed information than performing the same test on real nodes.

My Contribution. I improved MSPSim for better support of the radio chip CC2420, specifically to meet the needs of TinyOS such as support for SFD capture interrupt. I also performed experiments and wrote parts of the paper.

Chapter 5

Conclusions and Future Work

This licentiate thesis presents important steps towards simulating sensor networks with high level of detail and with support for accurate power profiling and timing of interacting nodes. To achieve even more realistic simulations a combination of detailed node models with accurate timing and improved radio medium simulation such as AvroraZ [3] is needed. The thesis also shows that this type of tools can be used for white-box interoperability tests with high level of control and detailed information during the test runs. White-box interoperability testing using COOJA/MSPSim can save much time and effort when developing or porting implementations of standard protocols such as 6LoWPAN/IPv6 and WirelessHART.

Another area where I will use COOJA/MSPSim is to study methods that optimize performance and enable a higher degree of self-configuration in sensor networks. I will study methods for learning self-configuration policies for sensor network applications and protocols. One promising method is reinforcement learning [28]. By formulating a utility function for the sensor network application it is possible to use reinforcement learning to learn configuration policies that optimize the utility. A typical utility function for a sensor network application includes parameters such as power consumption, reliability, response time, etc. The challenges for using reinforcement learning for configuration in sensor networks are first to develop methods that work in multi-agent settings and then to make the learned policies small enough to fit in the very resource constrained sensor nodes.

I will also improve COOJA/MSPSim to better support fast and accurate simulations that combine Java level simulation with firmware level simulation to get high level of detail while simulating at a high speed.

Bibliography

[1] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. *CM/Kluwer Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks*, 10(4), August 2005.

[2] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320, Boulder, Colorado, USA, 2006.

[3] Rodolfo de Paz Alberola and Dirk Pesch. Avroraz: Extending avrora with an ieee 802.15.4 compliant radio chip mode. In *3rd ACM International Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks*, Vancouver, Canada, October 2008.

[4] Isabel Dietrich, Feng Chen, Reinhard German, and Falko Dressler. Modeling energy consumption of wireless communications in OMNeT++. GI/ITG KuVS Fachgespräch Systemsoftware und Energiebewusste Systeme, October 2007.

[5] Ian Downard. Simulating sensor networks in ns-2. NRL Formal Report 5522, April 2004.

[6] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, November 2004.

[7] Adam Dunkels, Fredrik Österlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, pages 28–32, 2007.

[8] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and

A. Dunkels. Making Sensor Networks IPv6 Ready. In *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008)*, Raleigh, North Carolina, USA, November 2008.

[9] J. Eriksson, A. Dunkels, N. Finne, F. Österlind, and T. Voigt. Mspsim – an extensible simulator for msp430-equipped sensor boards. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, January 2007.

[10] Christian Frank and Kay Römer. Algorithms for generic role assignment in wireless sensor networks. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 230–242, New York, NY, USA, 2005. ACM Press.

[11] C. Han, R. K. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava. SOS: A dynamic operating system for sensor networks. In *MobiSys '05*, 2005.

[12] Wirelesshart. Web page. http://www.hartcomm2.org/.

[13] John Heidemann, Nirupama Bulusu, Jeremy Elson, Chalermek Intanagonwiwat, Kun chan Lan, Ya Xu, Wei Ye, Deborah Estrin, and Ramesh Govindan. Effects of detail in wireless network simulation. In *In Proceedings of the SCS Multiconference on Distributed Simulation*, pages 3–11, 2001.

[14] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[15] Isa100. Web page. http://www.isa.org/isa100/.

[16] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Klein Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating wireless and mobile networks in omnet++ the mixim vision. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–8, ICST, Brussels, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[17] O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *Proceedings of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, Sydney, Australia, May 2005.

[18] Olaf Landsiedel, Hamad Alizai, and Klaus Wehrle. When timing matters: Enabling time accurate and scalable simulation of sensor network applications. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*, pages 344–355, Washington, DC, USA, 2008. IEEE Computer Society.

[19] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 126–137, 2003.

[20] G. Mulligan, N. Kushalnagar, and G. Montenegro. IPv6 over IEEE 802.15.4 BOF (6lowplan). Web page. Visited 2005-02-21. `http://www.ietf.org/ietf/04nov/6lowpan.txt`

[21] The Network Simulator NS-2. http://www.isi.edu/nsnam/ns/.

[22] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, Tampa, Florida, USA, November 2006.

[23] Sung Park, Andreas Savvides, and Mani B. Srivastava. Sensorsim: a simulation framework for sensor networks. In *MSWIM '00: Proceedings of the 3rd ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, pages 104–111, New York, NY, USA, 2000. ACM.

[24] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of The Fourth International Conference on Information Processing in Sensor Networks. IPSN/SPOTS'05*, Los Angeles, CA, USA, April 2005.

[25] Jonathan Polley, Dionysys Blazakis, Jonathan Mcgee, Dan Rusk, and John S. Baras. Atemu: A fine-grained sensor network simulator. In *IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004.

[26] J. Schiller, H. Ritter, A. Liers, and T. Voigt. Scatterweb - low power nodes and energy aware routing. In *Proceedings of Hawaii International Conference on System Sciences*, Hawaii, USA, 2005.

[27] V. Shnayder, M. Hempstead, Chen B., G.W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *2nd International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2004.

[28] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998. ISBN: 0262193981

[29] B.L. Titzer, D.K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *Proceedings of the 4th international symposium on Information processing in sensor networks (IPSN)*, April 2005.

[30] Hung-Ying Tyan. *Design, Realization and Evaluation of a Component-Based Compositional Software Architecture for Network Simulation*. PhD thesis, 2002.

[31] Andras Varga. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference*, pages 319–324, Prague, Czech Republic, June 2001. SCS – European Publishing House.

[32] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. In *Proc. ACM MobiSys'04*, Boston, MA, USA, June 2004.

[33] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. *SIGSIM Simul. Dig.*, 28(1):154–161, 1998. ISSN: 0163-6103

[34] Zigbee. Web page. 2007-11-21. http://www.zigbee.org.

Part II

Included papers

Chapter 6

Experiences from Two Sensor Network Deployments — Self-Monitoring and Self-Configuration Keys to Success

Niclas Finne, Joakim Eriksson, Adam Dunkels, Thiemo Voigt.
Swedish Institute of Computer Science, SICS.
{nfi,joakime,adam,thiemo}@sics.se

6.1 Abstract

Despite sensor network protocols being self-configuring, sensor network deployments continue to fail. We report our experience from two recently deployed IP-based multi-hop sensor networks: one in-door surveillance network in a factory complex and a combined out-door and in-door surveillance network. Our experiences highlight that adaptive protocols alone are not sufficient, but that an approach to self-monitoring and self-configuration that covers more aspects than protocol adaptation is needed. Based on our experiences, we design and implement an architecture for self-monitoring of sensor nodes. We show that the self-monitoring architecture detects and prevents the problems with false alarms encountered in our deployments. The architecture also detects software bugs by monitoring actual and expected duty-cycle of key components of the sensor node. We show that the energy-monitoring architecture detects bugs that cause the radio chip to be active longer than expected.

6.2 Introduction

Surveillance is one of the most prominent application domains for wireless sensor networks. Wireless sensor networks enable rapidly deployed surveillance applications in urban terrain. While most wireless sensor network mechanisms are self-configuring and designed to operate in changing conditions [12, 16], the characteristics of the deployment environment often cause additional and unexpected problems [8, 9, 11]. In particular, Langendoen et al. [8] point out the difficulties posed by, e.g., hardware not working as expected.

To contribute to the understanding of the problems encountered in real-world sensor network deployments, we report on our experience from recent deployments of two surveillance applications: one in-door surveillance application in a factory complex, and one combined out-door and in-door surveillance network. Both applications covered a large area and therefore required multi-hop networking.

Our experiences highlight that adaptive protocols alone are not sufficient, but that an approach to self-monitoring and self-configuration that covers more aspects than protocol adaptation is needed. An example where we have experienced the need for self-monitoring of sensor nodes is when the components used in low-cost sensor nodes behave differently on different nodes. In many of our experiments, radio transmissions triggered the motion detector on a subset of our nodes while other nodes did not experience this problem.

Motivated by the observation that self-configuration and adaptation is not sufficient to circumvent unexpected hardware and software problems, we design and implement a self-monitoring architecture for detecting hardware and software problems. Our architecture consists of pairs of probes and activators where the activators start up an activity that is suspected to trigger problems and the probes measure if sensor components react to the activator's activity. Callback functions enable a node to self-configure its handling of a detected problem. We experimentally demonstrate that our approach solves the observed problem of packet transmissions triggering the motion detector.

To find software problems, we integrate Contiki's software-based on-line energy estimator [5] into the self-monitoring architecture. This allows us to detect problems such as the CPU not going into the correct low power mode, a problem previously encountered by Langendoen et al. [8]. With two examples we demonstrate the effectiveness of the self-monitoring architecture. Based on our deployment experiences, we believe this tool to be very valuable for both application developers and system developers.

The rest of the paper is structured as follows. The setup and measurements for the two deployments are described in Section 6.3. In Section 6.4 we present our experiences from the deployments, including unexpected be-

havior. Section 6.5 describes our architecture for self-monitoring while the following section evaluates it. Finally, we describe related work in Section 6.7 and our conclusions in Section 6.8.

6.3 Deployments

We have deployed two sensor network surveillance applications in two different environments. The first network was deployed indoors in a large factory complex setting with concrete floors and walls, and the second in a combined outdoor and indoor setting in an urban environment.

In both experiments, we used ESB sensor nodes [14] consisting of a MSP430 microprocessor with 2kB RAM, 60kB flash, a TR1001 868 MHz radio and several sensors. During the deployments, we used the ESB's motion detector (PIR) and vibration sensor.

We implemented the applications on top of the Contiki operating system [4] that features the uIP stack, the smallest RFC-compliant TCP/IP stack [3]. All communication uses UDP broadcast and header compression that reduces the UDP/IP header down to only six bytes: the full source IP address and UDP port, as well as a flag field that indicates whether or not the header is compressed.

We used three different types of messages: *Measurement messages* to send sensor data to the sink, *Path messages* to report forwarding paths to the sink, and *Alarm messages* that send alarms about detected activity.

We used two different protocols during the deployment. In the first experiment, we used a single-hop protocol where all nodes broadcast messages to the sink. In the second experiment, we used a multi-hop protocol where each node calculates the number of hops to the sink and transmits messages with a limit on hops to the sink. A node only forwards messages for nodes it has accepted to be relay node for. A message can take several paths to the sink and arrive multiple times. During the first deployment only a few nodes were configured to forward messages, but in the second deployment any node could configure itself to act as relay node.

After a sensor has triggered an alarm, an alarm message is sent towards the sink. Alarm messages are retransmitted up to three times unless the node hears an explicit acknowledgment message or overhears that another node forwards the message further. Only the latest alarm from each node is forwarded.

6.3.1 First Deployment: Factory Complex

The first deployment of the surveillance sensor network was performed in a factory complex. The main building was about 250 meters times 25 meters in size and three floors high. Both floors and most walls were made of concrete but there were sections with office-like rooms that were separated

by wooden walls. Between the bottom floor and first floor there was a smaller half-height floor. The largest distance between the sink and the most distant nodes was slightly less than 100 meters.

The sensor network we deployed consisted of 25 ESB nodes running a surveillance application. All nodes were either forwarding messages to the sink or monitored their environment using the PIR sensor and the vibration detector. We made several experiments ranging from a single hop network for measuring communication quality to a multi-hop surveillance network.

Single-Hop Network Experiment

We made the first experiment to understand the limitations of communication range and quality in the building. All nodes communicated directly with the sink and sent measurement packets at regular intervals.

Node	Distance (meter)	Walls	Received	Sent (expected)	Sent (actual)	Reception ratio (percent)	Signal strength (avg,max)	
2	65	1 C	92	621	639	15%	1829	2104
3	21	1 W	329	587	588	56%	1940	2314
4	55	1 C	72	501	517	14%	1774	1979
5	33	2 W	114	611	613	19%	1758	1969
6	18	1 W	212	580	590	37%	1866	2230
7	26	2 W	347	587	588	59%	2102	2568
8	15	1 W	419	584	585	71%	2131	2643
9	25	1 W	194	575	599	34%	1868	2218
10	23	2 W	219	597	599	37%	1815	2106
11	17	1 W	331	591	593	56%	2102	2582
50	27	2 W	230	587	594	39%	1945	2334

Table 6.1: Communication related measurements for the first experiment.

Table 6.1 shows the results of the measurements. The columns from left are node id, distance from the sink in meters, number of concrete and wooden walls between node and the sink, number of messages received at the sink from the node, number of messages sent by the node (calculated on sequence number), actual number of messages sent (read from a log stored in each node), percentage of successfully delivered messages, and signal strength measured at the sink. Table 6.1 shows that as expected the ratio of received messages decreases with increasing distance from the sink. As full sensor coverage of the factory was not possible using a single-hop network, we performed the other experiments with multi-hop networks.

Multi-Hop Network Experiments

After performing some experiments to understand the performance of a multi-hop sensor network with respect to communication and surveillance coverage, we performed the final experiment in the first deployment during a MOUT (Military Operation on Urban Terrain) exercise with 15-20 soldiers moving up and down the stairs and running in the office complex at the top level of the building.

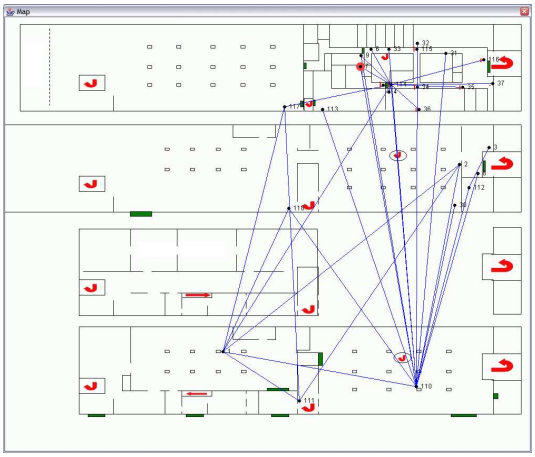

Figure 6.1: Screenshot from the final experiment in the factory deployment illustrating placement of the sensor nodes during the surveillance and the paths used to transfer messages. All levels of the factory are shown with the top level at the top of the screenshot and the basement at the bottom. The office complex is on the right side at the top level in the figure.

Node 110 (see Figure 6.1) was the most heavily loaded forwarding node in the network. It had a direct connection to the sink and forwarded 10270 messages from other nodes during the three hour long experiment. During the experiment the sink received 604 alarms generated by node 110. 27 percent of these alarms were received several times due to retransmissions. Node 8 had seven paths to the sink, one direct connection with the sink, and six paths via different forwarding nodes. The sink received 1270 unique alarms from node 8 and 957 duplicates. Most of the other nodes' messages multi-hopped over a few alternative paths to the sink, with similar or smaller delays than those from node 110 and 8. This indicates that the network was reliable and that most of the alarms got to the server; in many cases via several paths. With the 25 sensor nodes we achieved coverage of the most important passages of the factory complex, namely doors, stairs, and corridors.

6.3.2 Second Deployment: Combined in-door and out-door urban terrain

The second deployment was made in an artificial town built for MOUT exercises. It consisted of a main street and a crossing with several wooden buildings on both sides of the streets. At the end of the main street there were some concrete buildings. The distance between the sink and the nodes at the edge of the network was about 200 meters, making the network more

than twice as long as in the first deployment.

The surveillance system was improved in two important ways. First, the network was more self-configuring in that there was no need for manually configuring which role each node should have (relay node or sensor node). Each node configured itself for relaying if the connectivity to sink was above a threshold. Second, alarm messages also included path information so that information of the current configuration of the network was constantly updated as messages arrived to the sink. Even with the added path information in the alarm messages, the response times for alarms in the network were similar to the response times in the first deployment despite that the distant nodes were three or four hops away from the sink rather than two or three. Using 25 nodes we achieved fairly good sensor coverage of the most important areas.

6.4 Deployment Experiences

When we deployed the sensor network application we did not know what to expect in terms of deployment speed, communication quality, applicability of sensors, etc. Both deployments were made in locations that were new to us. This section reports on the various experiences we made during the deployments.

Network Configuration

During the first deployment the configuration needed to make a node act as a relay node was done manually. This made it very important to plan the network carefully and make measurements on connectivity at different locations in order to get an adequate number of forwarding nodes. This was one of the largest problems with the first deployment. During the second deployment the network's self-configuration capabilities made deployment a faster and easier task.

The importance of self-configuration of the network routing turned out to be higher than we expected since we suddenly needed to move together with the sink to a safer location during the second deployment, where we did not risk being fired at. This happened while the network was deployed and active.

Unforeseen Hardware Problems

During radio transmissions a few of the sensor nodes triggered sensor readings which cause unwanted false alarms. Since we detected and understood this during the first deployment, we rewrote the application to turn off sensing on the nodes that had this behavior. Our long term solution is described in the next section.

Parameter Configuration

In the implementation of the communication protocols and surveillance application there are a number of parameters with static values set during early testing with small networks. Many of these parameters need to be optimized for better application performance. Due to differences in the environment, this optimization can only partly be done before deployment. Examples of such parameters are retransmission timers, alarm triggering delays, radio transmission power level, and time before refreshing a communication link.

Ground Truth

It is important for understanding the performance of a sensor network deployment to compare the sensed data to ground truth. In our deployments, we did not have an explicit installation of a parallel monitoring system to obtain ground truth, but in both deployments we received a limited amount of parallel feedback.

During the first deployment, the sensor networks alerted us of movements in various parts of the factory but since we did not have any information about the soldiers' current locations it was difficult to estimate the time between detection by the sensor nodes and the alarm at the sink. Sometimes the soldiers threw grenades powerful enough to trigger the vibration sensors on the nodes. This way, we could estimate the time between the grenade explosions and the arrival of the vibration alarm at the sink. During the second deployment we received a real time feed from a wireless camera and used it to compare the soldiers' path with the alarms from the sensor network.

Radio Transmission

During the first deployment we placed forwarding nodes in places where we expected good radio signal strength (less walls, and floors). We expected the most used path to the sink via forwarding nodes in the stairwells. However, most messages took a path straight through two concrete floors via a node placed at the ground floor below the office rooms where the sensors were deployed.

Instant Feedback

One important feature of the application during deployment was that when started, a node visualized its connection. When it connected, the node beeped and flashed all its leds before being silent. Without this feature we would have been calling the person at the sink all the time just to see if the node had connected to the network. This way, we could also estimate a node's link quality. The longer time for the node to connect to the network,

the worse it was connected. We usually moved nodes that required more than 5 - 10 seconds to connect to a position with better connectivity.

6.5 A Self-Monitoring Architecture for Detecting Hardware and Software Problems

To ensure automatic detection of the nodes that have hardware problems, we design a self-monitoring architecture that probes potential hardware problems. Our experiences show that the nodes can be categorized into two types: those with a hardware problem and those without. The self-probing mechanism could therefore possibly be run at start-up, during deployment, or even prior to deployment.

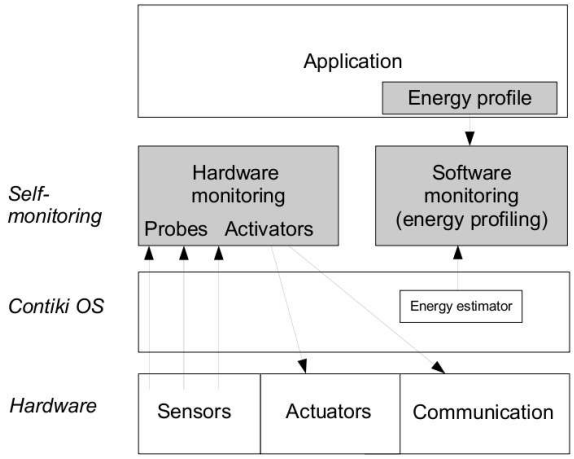

Figure 6.2: Architecture for performing self-monitoring of both hardware and software. Self-tests of hardware are performed at startup or while running the sensor network application. Monitoring of the running software is done continuously using the built-in energy estimator in Contiki.

6.5.1 Hardware Self-Test

Detection of hardware problems is done using a self-test at node start-up. The goal of the self-test is to make it possible to detect if a node has any hardware problems.

The self-test architecture consists of pairs of probes and activators. The activators start up an activity that is suspected to trigger problems and the probes measure if sensors or hardware components react to the activator's activity. An example of a probe/activator pair is measuring PIR interrupts when sending radio traffic. The API for the callbacks from the self tester

for probing for problems, executing activators and handling the results are shown in Figure 6.3.

```
int probe();
void execute_activator();
void report(int activator, int probe, int percentage);
```

Figure 6.3: The hardware self-test API.

With a few defined probes and activators, it is possible to call a self-test function that will run all probe/activator pairs. If a probe returns anything else than zero, this is an indication that a sensor or hardware component has reacted to the activity caused by the activator. The code in Figure 6.4 shows the basic algorithm for the self-test. The code assumes that the activator takes the time it needs for triggering potential problems, and the probes just read the data from the activators. This causes the self-test to monopolize the CPU, so the application can only call it when there is time for a self-test.

```
/* Do a self-test for each activator */
for(i = 0; i < activator_count; i++) {
  /* Clear the probes before running the activator */
  for(p = 0; p < probe_count; p++) {
    probe[p]->probe();
    probe_data[p] = 0;
  }
  for(t = 0; t < TEST_COUNT; t++) {
    /* run the activator and probe all the probes */
    activator[i]->execute_activator();
    for(p = 0; p < probe_count; p++)
      probe_data[p] += probe[p]->probe() ? 1 : 0;
  }
  /* send a report on the results for this activator-probe pair */
  for(p = 0; p < probe_count; p++)
    report(i, p, (100 * probe_data[p]) / TEST_COUNT);
}
```

Figure 6.4: Basic self-test algorithm expressed in C-code.

The self-test mechanism can either be built-in into the OS or a part of the application code. For the experiments we implement a self-test component in Contiki on the ESB platform.

A component on a node can break during the network's execution (we experienced complete breakdown of a node due to severe physical damage). In such case the initial self-test will not automatically detect the failure. Most sensor network applications have moments of low action, and in these cases it is possible to re-run the tests or parts of the tests to ensure that no new hardware errors have occurred.

6.5.2 Software Self-Monitoring

Monitoring the hardware for failure is taking care of some of the potential problem in a sensor network node. Some bugs in the software can also cause unexpected problems, such as the inability to put the CPU into low power mode [8]. This can be monitored using Contiki's energy estimator [5] combined with energy profiles described by the application developer.

```
ENERGY_PROFILE(60 * CLOCK_SECOND,      /* Check profile every 60 seconds */
               energy_profile_warning, /* Call this function if mismatch */
               EP(CPU, 0, 20),         /* CPU 0%-20% duty cycle */
               EP(TRANSMIT, 0, 20),    /* Transmit 0%-20% duty cycle */
               EP(LISTEN, 0, 10));     /* Listen 0%-10% duty cycle */
```

Figure 6.5: An energy profile for an application with a maximum CPU duty cycle of 20 percent and a listen duty cycle between 0 and 10 percent. The profile is checked every 60 seconds. Each time the system deviates from the profile, a call to the function *energy_profile_warning* is made.

6.5.3 Self-Configuration

Based on the information collected from the hardware and software monitoring the application and the operating system can re-configure to adapt to problems. In the case of the surveillance application described above the application can turn off the PIR sensor during radio transmissions if a PIR hardware problem is detected.

6.6 Evaluation

We evaluate the self-monitoring architecture by performing controlled experiments with nodes that have hardware defects and nodes without defects. We also introduce artificial bugs into our software that demonstrate the effectiveness of our software self-monitoring approach.

6.6.1 Detection of Hardware Problems

For the evaluation of the hardware self-testing we use one probe measuring PIR interrupts, and activators for sending data over radio, sending over RS232, blinking leds and beeping the beeper. The probes and activators are used to run the tests on ten ESB nodes of which two are having the hardware problems.

A complete but simplified set of probes, activators and report functions is shown in Figure 6.6. In this case, the results are only printed instead of used for deciding how the specific node should be configured.

```
/* A basic PIR sensor probe */
static int probe_pir(void) {
  static unsigned int lastpir;
  unsigned int value = lastpir;
  lastpir = (unsigned int) pir_sensor.value(0);
  return lastpir - value;
}

/* A basic activator for sending data over radio */
static void activator_send(void) {
  /* send packet */
  rimebuf_copyfrom(PACKET_DATA, sizeof(PACKET_DATA));
  abc_send(&abc);
}

/* Print out the report */
static void report(int activator, int probe, int trigged_percent) {
  printf("Activator %u Probe %u: %u%%\n", activator, probe, trigged_percent);
}
```

Figure 6.6: A complete set of callback functions for a self test of radio triggered PIR sensor.

As experienced in our two deployments, the PIR sensor triggers when transmitting data over the radio during the tests on a problem node. On other nodes the PIR sensors remain untriggered. Designing efficient activators and probes is important for the self-monitoring system. Figure 6.7 illustrates the variations in detection ratio when varying the number of transmitted packets and packet sizes during execution of the activator. Based on these results a good activator for the radio seems to be sending three 50 bytes packets.

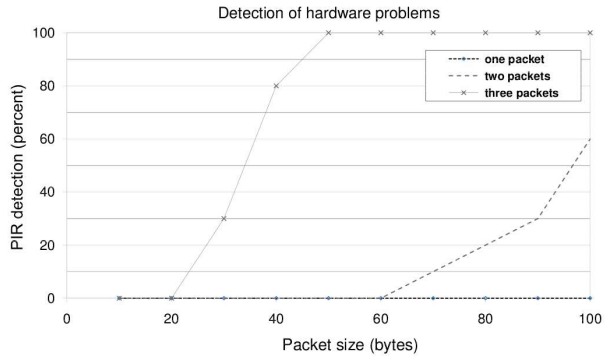

Figure 6.7: Results of varying the activator for sending radio packets on a problem node. Sending only one packet does not trigger the PIR probe, while sending three packets with a packet size larger than 50 bytes always triggers the problem. On a good node no PIR triggerings occur.

6.6.2 Detection of Software and Configuration Problems

Some of the problems encountered during development and deployment of sensor network software are related to minor software bugs and misconfigurations that decrease the lifetime of the network [8]. Bugs such as missing to power down a sensor or the radio chip when going into sleep mode, or a missed frequency divisor and therefore higher sample rate in an interrupt driven A/D based sensor can decrease the network's expected lifetime. We explicitly create two software problems causing this type of behavior.

The first problem consists of failing to turn off the radio in some situations. Figure 6.8 shows the duty cycle of the radio for an application that periodically sends data. XMAC [1] is used as MAC protocol to save energy. XMAC periodically turns on the radio to listen for transmissions and when sending it sends several packet preambles to wake up listeners. Using the profile from Figure 6.5 a warning is issued when the radio listen duty cycle drastically increases due to the software problem being triggered.

In the second problem the sound sensor is misconfigured causing the A/D converter to run at twice the desired sample rate. The node then consumes almost 50% more CPU time than normal. This misbehavior is also detected by the software self-monitoring component.

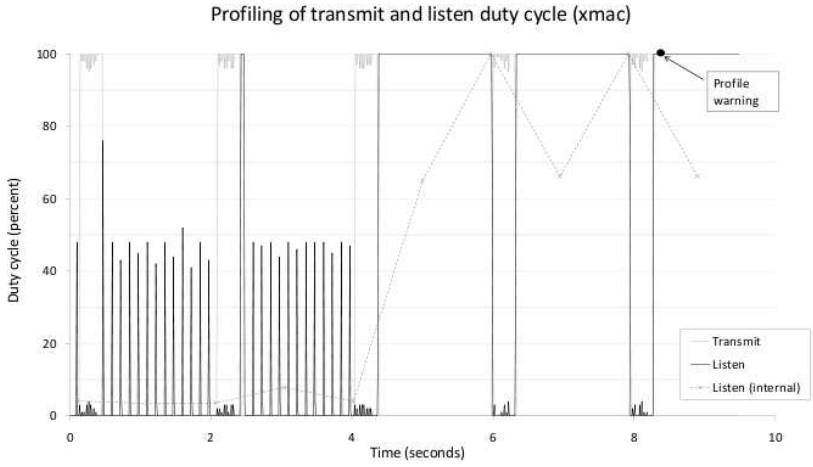

Figure 6.8: Duty-cycle for listen and transmit. The left part shows the application's normal behavior with a low duty cycle on both listen and transmit. The right part shows the behavior after triggering a bug that causes the radio chip to remain active. The dashed line shows the listen duty cycle estimated using the same mechanism as the energy profiler but sampled more often. The next time the energy profile is checked the deviation is detected and a warning issued.

6.7 Related Work

During recent years several wireless sensor networks have been deployed the most prominent being probably the one on Great Duck Island [9]. Other efforts include glacier [11] and water quality monitoring [2]. For an overview on wireless sensor network deployments, see Römer and Mattern [13].

Despite efforts to increase adaptiveness and self-configurationin wireless sensor networks [10, 12, 16], sensor network deployments still encounter severe problems: Werner-Allen et al. have deployed a sensor network to monitor a volcano in South America [15]. They encountered several bugs in TinyOS after the deployment. For example, one bug caused a three day outage of the entire network, other bugs made nodes lose time synchronization. We have already mentioned the project by Langendoen that encountered severe problems [8]. Further examples include a surveillance application called "A line in the sand" [6] where some nodes would detect false events exhausting their batteries early and Lakshman et al.'s network that included nodes with a hardware problem that caused highly spatially correlated failures [7]. Our work has revealed additional insights such as a subset of nodes having a specific hardware problem where packet transmissions triggered the motion detector.

6.8 Conclusions

In this paper we have reported results and experiences from two sensor network surveillance deployments. Based on our experiences we have designed, implemented and evaluated an architecture for detecting both hardware and software problems. The evaluation demonstrates that our architecture detects and handles hardware problems we experienced and software problems experienced during deployments of other researchers.

Acknowledgments

This work was funded by the Swedish Defence Materiel Administration and the Swedish Agency for Innovation Systems, VINNOVA.

Bibliography

[1] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-mac: a short preamble mac protocol for duty-cycled wireless sensor networks. In *ACM SenSys*, November 2006.

[2] T.L. Dinh, W. Hu, P. Sikka, P. Corke, L. Overs, and S. Brosnan. Design and Deployment of a Remote Robust Sensor Network: Experiences from an Outdoor Water Quality Monitoring Network. *IEEE Congf. on Local Computer Networks*, pages 799–806, 2007.

[3] A. Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of The First International Conference on Mobile Systems, Applications, and Services (MOBISYS '03)*, May 2003.

[4] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, November 2004.

[5] Adam Dunkels, Fredrik Österlind, Nicolas Tsiftes, and Zhitao He. Software-based on-line energy estimation for sensor nodes. In *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*, pages 28–32, 2007.

[6] Anish Arora et al. A line in the sand: a wireless sensor network for target detection, classification, and tracking. *Computer Networks*, 46(5):605–634, 2004.

[7] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *ACM SenSys*, pages 64–75, 2005.

[8] K. Langendoen, A. Baggio, and O. Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Rhodes Island, Greece, April 2006. IEEE.

[9] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *First ACM Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, Atlanta, GA, USA, September 2002.

[10] PJ Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, and K. Rothermel. TinyCubus: a flexible and adaptive framework sensor networks. *EWSN 2005*.

[11] P. Padhy, K. K. Martinez, A. Riddoch, H. Ong, and J. Hart. Glacial environment monitoring using sensor networks. In *Proc. of the Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, Stockholm, Sweden, June 2005.

[12] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-MAC: a hybrid MAC for wireless sensor networks. *ACM SenSys*, pages 90–101, 2005.

[13] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[14] J. Schiller, H. Ritter, A. Liers, and T. Voigt. Scatterweb - low power nodes and energy aware routing. In *Proceedings of Hawaii International Conference on System Sciences*, Hawaii, USA, 2005.

[15] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, USA, 2006.

[16] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. *ACM SenSys*, pages 14–27, 2003.

Chapter 7

Accurate Network-Scale Power Profiling for Sensor Network Simulators

Joakim Eriksson, Fredrik sterlind, Niclas Finne, Adam Dunkels
, Nicolas Tsiftes, Thiemo Voigt. Swedish Institute of Computer Science, SICS.
{joakime, fros, nfi, adam, nvt, thiemo}@sics.se

Abstract

Power consumption is the most important metric in wireless sensor network research, but existing simulation tools for measuring or estimating power consumption are either impractical or have unclear accuracy. We present COOJA/MSPSim, a practical simulation-based tool for network-scale power estimation based on Contiki's built-in power profiling mechanism, the COOJA sensor network simulator and the MSPSim sensor node emulator. We compare experimental results measured on real sensor nodes with simulation results for three different MAC protocols. The accuracy of our results indicates that COOJA/MSPSim enables accurate network-scale simulation of the power consumption of sensor networks.

7.1 Introduction

Power consumption is the most important metric in wireless sensor networks because reduced power consumption leads to increased network lifetime. Many different mechanisms for reducing the power consumption for sensor networks have been proposed. Energy has been reduced with more efficient topology management [4], routing [20] and medium access [1, 24]. Most power-saving mechanisms focus on reducing radio on-time because radio communication and idle listening are the most power-consuming task in wireless sensor networks [8, 18]. To evaluate the efficiency of power-saving mechanisms, researchers must be able to quantify the energy consumption at the network scale.

Software-based power profiling has enabled non-intrusive and scalable power profiling in real sensor networks [7]. The technique is based on measuring the time that each component is active and multiplying that time by the component's power consumption. This method of measuring energy is accurate, but by the nature of testbeds, it is typically limited in scale and mobility. Testbed experiments require setup, instrumentation and infrastructure. Furthermore, the arrangement of the nodes is usually fixed and difficult to change. Simulations, on the other hand, scale well and handle mobility and repeatability with ease. There exist a number of simulators for sensor networks. Some of them are able to estimate power consumption but their accuracy has only been demonstrated in node-local experiments.

In this paper we present COOJA/MSPSim, a power profiling tool that enables accurate network-scale energy measurements in a simulated environment. Our tool combines the sensor network simulator COOJA [16] with the MSPSim hardware emulator [9] and Contiki's software-based power profiler [7]. By using the detailed instruction level emulation of the MSP430 processor, we can obtain accurate power profiles of simulated networks in COOJA/MSPSim.

The contributions of this paper are the presentation and the evaluation of COOJA/MSPSim. Our results demonstrate that COOJA/MSPSim enables accurate network-scale power profiling for sensor networks. Our evaluation consists of three case studies. In each case study we use a different MAC protocol to explore power profiling nuances as far down as possible in the network stack. The MAC protocols are Low Power Probing [15], X-MAC [1], and a TDMA-based data collection protocol called CoReDac [25]. Our case studies demonstrate that the power measurements for both transmission and listen power in testbed and simulation match very well. CoReDac's transmission power is simulated with an accuracy of around $0.6\mu W$. Using LPP, we simulate the power consumption in high packet transmission rate scenarios that matches the testbed results with a difference of less than 0.8%. For the more complicated X-MAC protocol, the difference between the experimental and simulated results is typically below 2%.

The remainder of this paper is outlined as follows. We explore related work in Section 7.2. Then we describe our tool for network-scale power profiling in Section 7.3. In Section 7.4 we experimentally evaluate the accuracy of COOJA/MSPSim through a set of case studies with different MAC protocols. Section 7.5 concludes our work.

7.2 Related Work

There are many sensor network simulators with energy estimation abilities [12, 13, 21], but their accuracy has only been demonstrated in node-local experiments. Avrora [22] is a machine code level simulator similar to MSPSim. While it offers a cycle-accurate simulation of AVR-based nodes, it does not have a power profiler. For this purpose, Landsiedel et al. have created the power analyzer AEON [12] on top of Avrora. AEON is limited to TinyOS, however. Furthermore, the authors do not compare simulation with testbed results for multi-hop applications. In contrast, our work is aimed at power profiling at a network scale.

PowerTOSSIM [21] is an extension of TOSSIM [13] designed to estimate the power consumption of Mica2 sensor nodes. Since TOSSIM operates at the operating system level, its granularity with respect to timing and interrupt properties is not sufficient when nodes interact [22]. Our measurements of the radio power consumption show that a very detailed model of the node is required to obtain accurate results. Trathnigg et al. [23] improve the accuracy of PowerTOSSIM, but for a single node only. Colesanti et al. [5] evaluated the accuracy of a multi-node simulation using metrics such as packets sent and received. They got inaccurate results and concluded that a more sophisticated node model is required. By using emulated nodes in the simulation, COOJA/MSPSim uses a very detailed node model that considerably improves the power measurement accuracy.

Haq and Kunz compare emulated testbed results with simulation results for mobile ad-hoc network routing protocols [10]. While their results match in low traffic scenarios, the results differ in scenarios with higher traffic rates. In contrast, COOJA/MSPSim maintains the accuracy in high traffic. Cavin et al. compare simulation results obtained with several simulators for mobile ad-hoc networks [3]. They discovered large divergences for packet delivery rates, latency and message overhead when simulating an intentionally simple flooding algorithm in different scenarios.

Ivanov et al. show that after careful simulation parameter adjustment, NS-2 accurately models packet delivery ratios in a wireless mesh testbed [11]. The parameter adjustment did not improve the accuracy regarding connectivity and packet latencies, however. COOJA/MSPSim allows accurate power profiling of arbitrary nodes in the network, and is orthogonal to that of an accurate radio model.

7.3 Simulation-based Network-scale Power Profiling

We develop COOJA/MSPSim for network-scale power estimation by combining three existing tools: Contiki's power profiling [7], the COOJA sensor network simulator [16], and the MSPSim sensor node emulator [9].

7.3.1 Contiki Power Profiler

The built-in power profiler in Contiki estimates the power consumption of sensor nodes in a real network. Thereby it enables scalable measurements of the power consumption. The power profiling mechanism measures the time that hardware components spend in different operating modes. This data is then combined with detailed pre-measured data of power consumption for these components into power consumption estimations. This mechanism can be implemented on most microprocessors with very small overhead.

7.3.2 COOJA

The other important component of our power profiling software is the COOJA simulator, a Java-based sensor network simulator. COOJA has the ability to mix simulations of sensor devices at multiple abstraction levels. These levels are application level, OS level, and hardware level. In the application level the simulated nodes run the application logic reimplemented in Java - the native language of COOJA. In the OS level the nodes use the same code as real nodes, but compiled for the host machine running COOJA. Finally in the hardware level the nodes run the same compiled code that can be used in real nodes, e.g. the same system image. The hardware level is provided by MSPSim that emulates systems based on the MSP430 processor family. By using MSPSim underneath, COOJA allows simulated nodes to execute the same system image as the one used on the real nodes. The nodes at different abstraction levels communicate with each other using one of the three radio propagation models available in COOJA.

7.3.3 MSPSim

MSPSim is an instruction level emulator of MSP430-based sensor network nodes. MSPSim targets cycle accurate emulation of both the MSP430 CPU core and built-in peripherals such as timers, serial communication and analog to digital converters. Furthermore, MSPSim emulates external components such as the radio chip CC2420, sensors, and flash memories. MSPSim also provides emulation of complete sensor devices such as the Tmote Sky [17] and Scatterweb ESB [19].

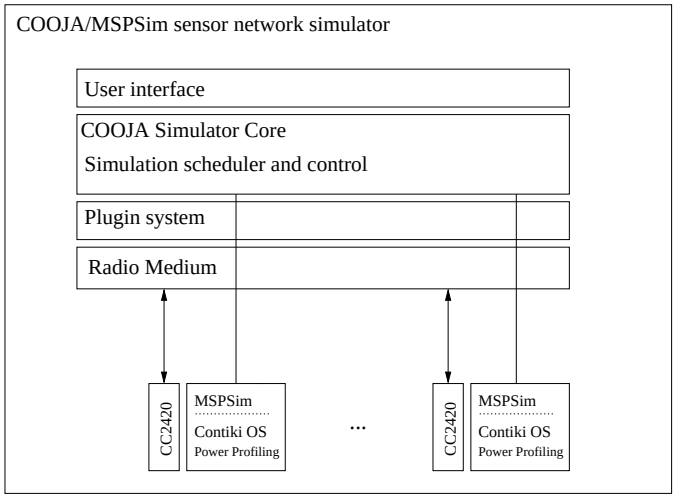

Figure 7.1: The architecture of the COOJA/MSPSim simulator. MSPSim is integrated into the COOJA simulator and Contiki's built-in power profiler provides estimation of power consumption.

7.3.4 A Network-Scale Power Profiler

We combine the three tools presented above into an accurate network-scale power profiler. Figure 7.1 shows the integrated COOJA/MSPSim architecture with COOJA controlling MSPSim and the power profiler in Contiki that provides COOJA with the estimation of energy consumption. It also shows the sensor nodes' radio communication via the emulated CC2420 and COOJA's radio medium simulation. Several improvements of the involved tools are necessary to achieve an accurate power profiling. We ensure that the components emulated in MSPSim have the same timing as in real nodes. In addition, we integrate MSPSim more tightly with COOJA including a more fine-grained connection between the emulated nodes' radio chips. In order to increase the power profiling accuracy we extend the timer emulation, and improve the timing precision of the SPI bus and the CC2420 radio transmissions.

We estimate the power consumption on a network scale in COOJA using Contiki's built-in power profiling mechanism, and we run the Contiki application on emulated hardware using MSPSim. In this paper our simulations make use of hardware-emulation for all nodes since we need accurate power profiling for the complete network. Hence, we benefit both from COOJA's ability to simulate network behaviour and the on-line energy estimation in Contiki. Furthermore, MSPSim's sophisticated and detailed node model provides fine-grained timing and interrupt modeling which is important for the accuracy when estimating power consumption. Figure 7.2 shows

a screenshot of our COOJA/MSPSim simulator during one of the evaluation experiments.

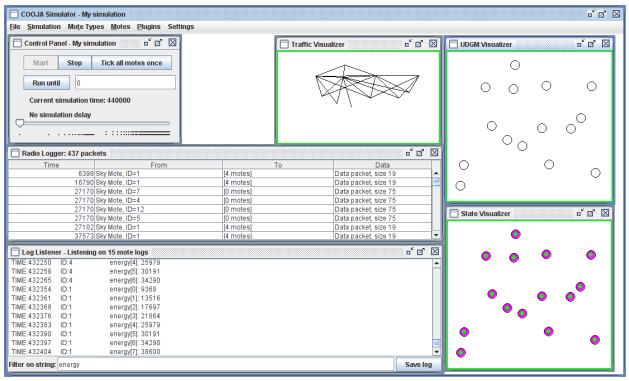

Figure 7.2: Screenshot of power profiling in the COOJA/MSPSim simulator.

7.4 Evaluation

To evaluate the accuracy of our simulation-based approach, we compare the results of the energy estimation obtained through simulation with results obtained through testbed experiments. For the testbed experiments, we implement all software in the Contiki operating system [6] and run it on Tmote Sky nodes. The nodes use Contiki's software-based method to measure power consumption as described in Section 7.3.1. We perform the first experiments with a tree-based data collection protocol CoReDac. Then we experiment with the MAC protocols Low Power Probing [15] and X-MAC [1] to evaluate our power profiling accuracy on the lowest levels of the network stack. To compute the power consumption, we assume a voltage of 3V and a current draw of 20mA for listening and 17.7mA for radio transmissions, as measured on Tmote Sky nodes by Dunkels et al. [7].

7.4.1 Case Study: Data Collection with CoReDac

Protocol Overview For this case study we use CoReDac, a TDMA-based convergecast protocol [25]. In contrast to other convergecast protocols such as Dozer [2], CoReDac builds a collection tree that guarantees collision-free radio traffic.

To achieve low delay, CoReDac borrows the idea of staggered communication from D-MAC [14] as shown in Figure 7.3. In D-MAC packets from nodes on the same level can cause collisions, whereas CoReDac parent nodes avoid collisions among packets from their children by assigning time slots for transmission to their children. The information about the assignment is

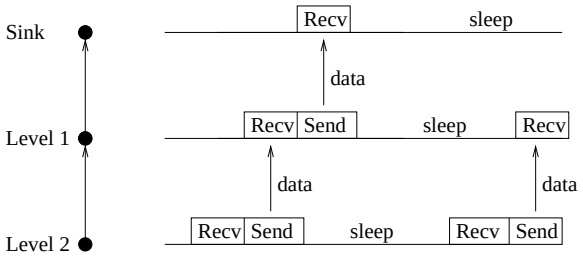

Figure 7.3: Staggered communication in CoReDac

contained in the acknowledgements. Acknowledgements play a pivotal role since they are also used for synchronization and on-demand slot assignment.

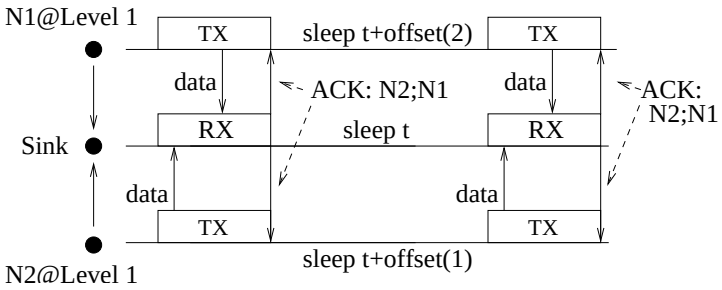

Figure 7.4: On-demand slot assignment to avoid collisions.

Figure 7.4 shows how CoReDac assigns transmit slots. The figure shows that the sink announces that $N2$ receives the transmit slot before $N1$. The sink's acknowledgement also signals when the sink's next receive slot starts, namely in $sleep_t$ seconds. This way, the acknowledgements contain all information to achieve a collision-free communication schedule between a parent node and its children. This scheme is recursively applied towards the whole tree. In order to avoid collisions between nodes on different levels, we set a maximum number of children per node. Based on this maximum number, its position in the tree and the receive slot of its parent, a node can compute its unique receive slot.

Setup and Results We measure CoReDac's energy-efficiency both on real hardware with Contiki's built-in power profiling mechanism and with COOJA/MSPSim as described in Section 7.3. In these experiments, the maximum number of children is set to three and $sleep_t$ is set to 30 seconds. We compare networks of different sizes namely with 4 and 15 nodes. We also simulate a network of 40 nodes. Due to the limited size of our testbed, we can simulate a network consisting of only 15 nodes. We are able to simulate more

than 40 nodes, but then we need to increase $sleep_t$ to guarantee collision-free trees. The length of the receive slot is not dependent of the number of nodes in a network. In our CoReDac implementation, there is a small difference between the length of the slots of the children of the same parent that depends on the order of the children. Therefore, we expect that the power consumption is independent of the size of the network but not exactly constant.

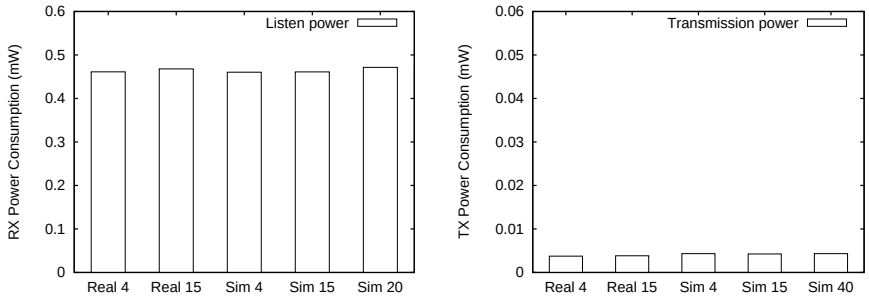

Figure 7.5: The results from a testbed measurement of the power consumption of CoReDac (Real) and the simulation runs (Sim) agree with each other. The left graph shows the power consumption of the radio in listen mode and the right graph the power consumption of the radio in transmission mode. Note that the scales are different.

The left graph in Figure 7.5 shows CoReDac's average power consumption per node of the radio in listen mode that we call RX power consumption. Real n denotes results from a testbed measurement with n nodes, whereas Sim n denotes simulation results with n nodes. The figure shows that the measured power consumption on real nodes matches very well with the power consumption estimated with COOJA/MSPSim. In particular, the difference between the results does not increase with the size of the network.

The right graph in Figure 7.5 presents the average power consumption per node for transmissions.The figure shows that the power consumption for transmitting packets in CoReDac is less than 1% of the power consumption for listening and receiving packets which confirms the measurements by Dunkels et al. [7]. As for power consumption of the radio in listen mode, the results obtained by simulation and experiments with real hardware match well. The difference of the power consumption for transmitting packets is less than $0.6\mu W$. Further, the difference between the results does not increase with the size of the simulated networks. These results show that COOJA/MSPSim accurately power profiles networks of nodes running TDMA-based MAC protocols.

7.4.2 Case Study: Low Power Probing

Protocol Overview Low power probing (LPP) is a power-saving MAC protocol [15]. As shown in Figure 7.6, LPP receivers periodically send small packets, so called probes, to announce that they are awake and ready to receive a data packet. After sending a probe, the receiver keeps its radio on for a short time to listen for data packets. A sender that has a packet to be sent turns on its radio waiting for a probe from a neighbour it wants to send to. On the reception of a probe from a potential receiver, the node sends an acknowledgement before the data packet.

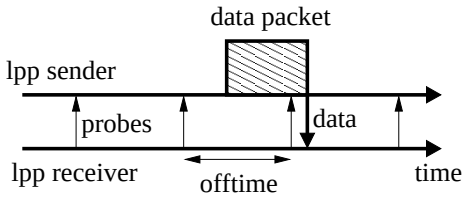

Figure 7.6: Low Power Probing

The LPP implementation in Contiki contains two important parameters. *Ontime* determines how long a receiver keeps the radio on after the transmission of a probe. *Offtime* is the time between probes. Nodes modify *offtime* with random offsets in order to avoid synchronization effects. The random offsets are distributed uniformly between $\frac{3}{4} \times offtime$ and $\frac{5}{4} \times offtime$.

Results: Sender-Receiver Scenario In the first scenario we have one LPP receiver and one LPP sender. We vary the packet rate at which the sender hands data packets to the MAC layer. In the experiments, we set *ontime* to $\frac{1}{128}$ seconds and *offtime* to 0.25 seconds.

The results of this experiment are shown in Figure 7.7 and Figure 7.8. The top graph of Figure 7.7 depicts the RX power consumption for the receiver when the transmission rate of the sender increases from zero packets to four packets per second. The figure shows that the basic power consumption of an LPP receiver is about 1.75 mW, namely 1.78 mW in the simulator and 1.725 mW on real nodes. Note that this is very close to the theoretical value that is $\frac{1}{33} \times 20mA \times 3V = 1.818mW$ with the assumptions above. The power is consumed for keeping the radio on after the transmission of the probes. The power consumption increases when more packets need to be received since the packet reception requires the radio to be turned on longer than *ontime*. The figure also shows that the estimated power consumption in the simulator matches the power consumption measured with real hardware.

With a packet rate of four packets/s, the sending rate is higher than

the probing rate and hence packets need to be dropped. Nevertheless, both the energy consumption and the packet reception are accurately simulated. In the simulation the packet rate is 87.9% on real nodes while it is 89.9% in simulation. The energy consumption with 4 packet/s is very accurate with a difference of less less than 4% for the receiver and less than 0.8% for the sender. The results clearly demonstrate that we do not encounter the problems that Haq and Kunz observed when comparing simulation and emulation of mobile ad hoc routing protocols, namely a large quantitative and qualitative difference under high traffic load [10]. The difference for lower packet rates is much smaller: for a packet rate of 0.25 packets/s less than 0.3% for the receiver.

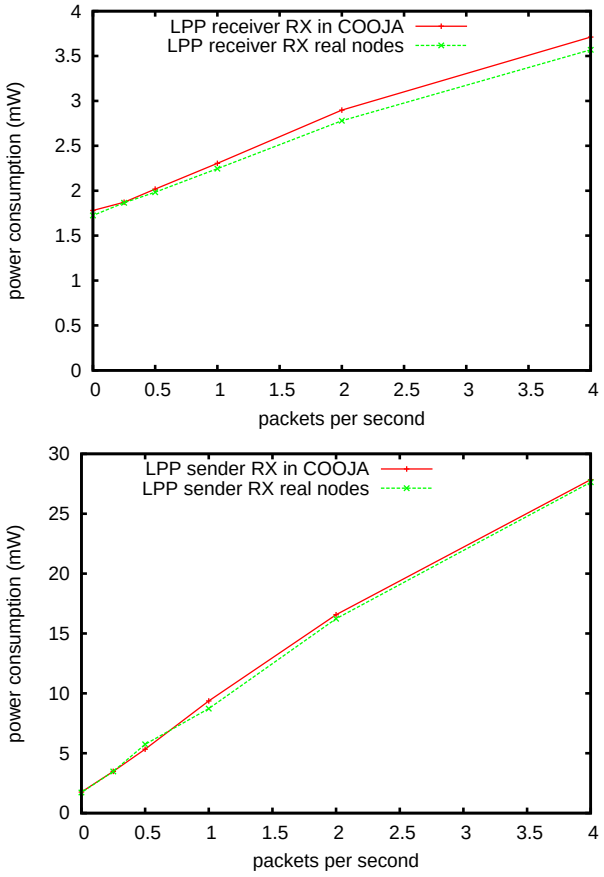

Figure 7.7: With LPP as the underlying MAC protocol the results of the RX power consumption of the receiver (top graph) and the sender (bottom graph) is accurately simulated.

The bottom graph of Figure 7.7 depicts the RX power consumption of the sender. Again, the results obtained by simulation match the results obtained

with real hardware very well. The figure also shows that as expected the power consumption of the sender increases with a higher packet sending rate. With a higher sending rate, the overall time a sender has its radio on waiting for probes increases which causes higher RX power consumption.

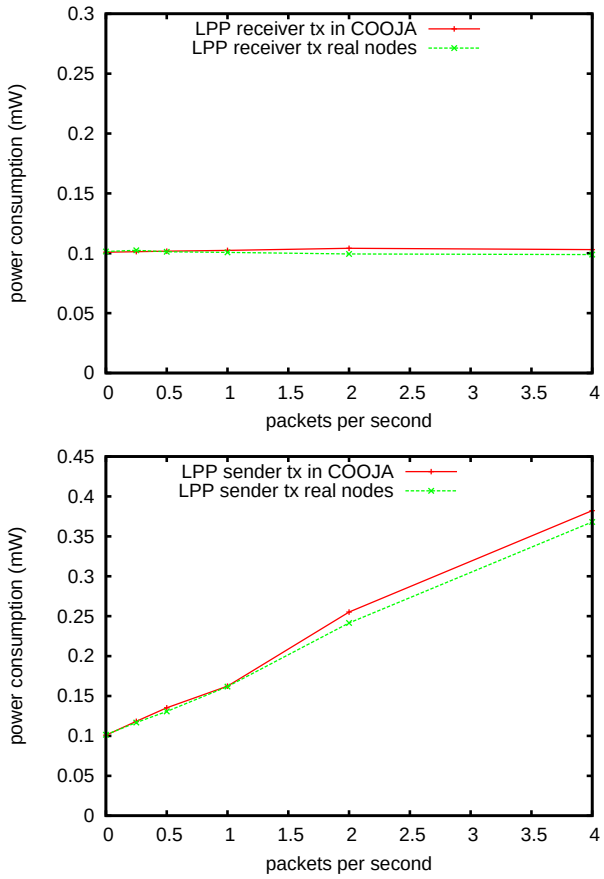

Figure 7.8: With LPP as the underlying MAC protocol also the TX power consumption of the receiver (top graph) and the sender (bottom graph) are accurately simulated.

Figure 7.8 shows that also the TX power consumption of both sender and receiver are accurately simulated despite that the TX power consumption is very low and hence only small timing differences could cause large relative discrepancies.

Results: Multi-hop scenario In the next experiment, we place a sender, a forwarder and the sink in radio range of each other. All nodes run LPP with the same configuration parameters as above. The sending rate of the sender is set two packets/s. During our experiments, we do not experience

any packet drops, i.e. all packets arrive at the sink.

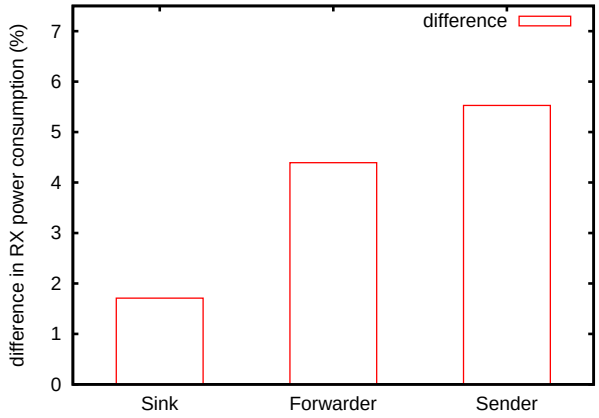

Figure 7.9: The difference between simulated and measured RX power consumption for a two-hop network is small.

Figure 7.9 shows the difference of the RX power consumption between the simulation and the experiments on real nodes. For the sink, the difference is below 2%. For the other two nodes, the difference is higher. When transmitting a packet, these nodes need to keep the radio on until they receive a probe. Probes are not sent at constant intervals to avoid synchronization effects, which is one possible reason for the larger difference between simulation and results with real nodes for nodes that transmit packets.

7.4.3 Case Study: X-MAC

Protocol Overview X-MAC is a power-saving MAC protocol [1] in which senders use a sequence of short preambles (strobes) to wake up receivers. Nodes turn off the radio for most of the time to reduce idle radio listening. They wake up shortly at regular intervals to listen for strobes. When a receiving node wakes up and receives a strobe with its receiver address, the receiver replies with an acknowledgement indicating that it is awake. On the reception of the acknowledgement, the sender transmits the full packet.

The X-MAC implementation in the Contiki operating system contains two important parameters, namely *ontime* that determines how long a receiver keeps the radio on when listening for strobes, and *offtime*, the time between two listening times.

During the tests *ontime* is set to $\frac{1}{100}s$ and *offtime* is set to $\frac{1}{4}s$. During the evaluation experiments the sending application sends data at a fixed packet rate but with small variations in order to avoid synchronization between the sender and receiver that would lead to a constant number of strobes before the receiver wakes up. By doing this the average number of strobes required

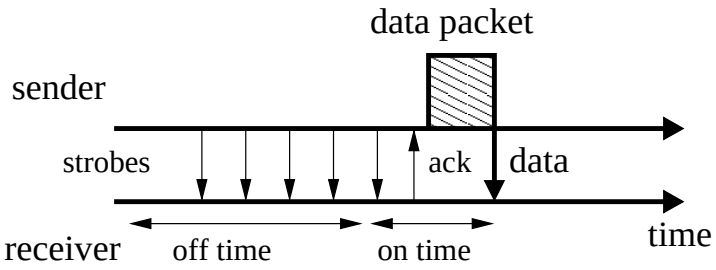

Figure 7.10: X-MAC: the sender strobes until the receiver is awake and can receive a packet.

before wake-up of the receiver is half the maximum strobes, which gives an average wake-up time of *offtime*/2.

Results: X-MAC Sender and Receiver The set-up of our first experiment with X-MAC consists of two nodes, one sender and one receiver. We use several different packet rates ranging from one packet per two seconds to four packets a second. The latter is also the maximum speed with the used X-MAC configuration. Figure 7.11 shows the result of the measurements for both simulated and real nodes. The top graph shows the power consumption of the radio while being in receive mode, whereas the bottom graph shows the transmission power. The difference between the average energy consumption in simulated nodes and real nodes is small, typically around two percent. The result for a packet rate of four packets/s differ more. Initial experiments indicate that the main problem is the speed of communication between the microprocessor and the radio chip for reading out received packets.

During the initial runs on real sensor nodes we observed a packet loss below one percent. This packet loss was not simulated causing a small difference in packet loss between simulation and real nodes. Since the packet loss is below one percent this difference does not affect the results significantly.

The results depicted in Figure 7.11 from both the simulations and real nodes show that the average energy consumption for a X-MAC sender corresponds well with our expectation of sending strobes for half the *offtime* before waking the receiver.

7.4.4 Power Profiling Accuracy

Our experiments suggest that the results obtained through simulation match the results obtained through testbed experiments and Contiki's power profiling mechanism for different MAC protocols including TDMA-based protocols, low power probing and X-MAC low power listening. The results

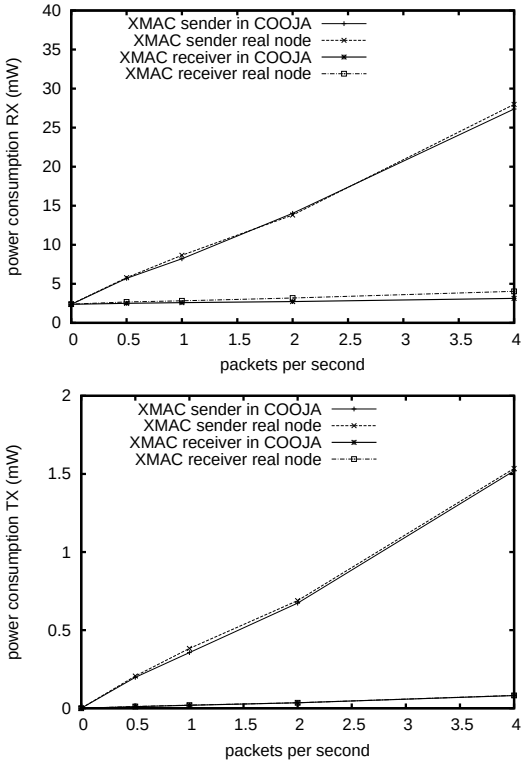

Figure 7.11: X-MAC power consumption of radio listen (top graph) and radio transmissions (bottom graph). Both real nodes and simulated nodes have very similar behaviour for the senders and receivers when varying the packet ratio. This shows that the simulation has a good accuracy when simulating X-MAC.

demonstrate that COOJA can accurately estimate the energy consumption of interacting nodes. For the TDMA-based CoReDac protocol we have shown that the simulation results also match results obtained in our testbed.

Our results also confirm the findings of Colesanti et al. that argued for better node models for improved accuracy [5]. Indeed, the results in Figure 7.8 initially differed with 50%, i.e. the power consumption was 0.1 mW on real nodes compared to 0.15 mW in simulation. Since transmitting a packet is inherently fast, even small inaccuracies can be responsible for the divergence of the results. By improving the accuracy of the timer system and the radio chip emulation in MSPSim, we were able to almost eliminate the discrepancy and achieve a difference of less than $0.6\mu W$ between simulation and testbed results.

As mentioned in Section 7.2, our goal is not to validate the accuracy of the radio models in COOJA but the radio power consumption caused by

the interaction of nodes. The TDMA-based CoReDac protocol is collision-free by design and hence there are no collisions between nodes after the initialization phase. Therefore, we were able to validate CoReDac's simulated power consumption also in the testbed. For LPP and X-MAC the actual power consumption depends very much on the delivery rate, the risk of packet collisions and other factors that are influenced by the environment. Therefore, we have constrained ourselves to demonstrate COOJA's ability to accurately simulate behaviour that relies on fine-grained timing and interrupts when nodes interact. Nevertheless, our results indicate that when appropriately modeling the surroundings, COOJA enables accurate simulation of the power consumption of large-scale sensor networks.

7.5 Conclusions

In this paper we have presented COOJA/MSPSim, a tool for simulation-based network-scale power profiling that combines Contiki's on-line power profiling mechanism, the COOJA sensor network simulator and the MSPSim sensor node emulator. We have shown that the results obtained with COOJA/MSPSim correspond well with the results obtained through measurements on real hardware. Our results demonstrate that COOJA/MSPSim enables accurate simulation of the power consumption of sensor networks.

Acknowledgements

This work was financed by VINNOVA, the Swedish Agency for Innovation Systems, and the Uppsala VINN Excellence Center for Wireless Sensor Networks WISENET, also partly funded by VINNOVA. This work has been partially supported by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053.

Bibliography

[1] M. Buettner, G. V. Yee, E. Anderson, and R. Han. X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 307–320, Boulder, Colorado, USA, 2006.

[2] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *IPSN '07*, 2007.

[3] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of MANET simulators. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, Toulouse, France, 2002.

[4] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. *IEEE TRANSACTIONS ON MOBILE COMPUTING*, pages 272–285, 2004.

[5] U.M. Colesanti, C. Crociani, and A. Vitaletti. On the accuracy of omnet++ in the wireless sensornetworks domain: simulation vs. testbed. In *Proceedings of the 4th ACM workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 25–31, Chania, Greece, October 2007.

[6] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, November 2004.

[7] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based online energy estimation for sensor nodes. In *Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV)*, Cork, Ireland, June 2007.

[8] P. Dutta, D. Culler, and S. Shenker. Procrastination might lead to a longer and more useful life. In *Proceedings of HotNets-VI*, Atlanta, GA, USA, November 2007.

[9] J. Eriksson, A. Dunkels, N. Finne, F. Österlind, and T. Voigt. Mspsim – an extensible simulator for msp430-equipped sensor boards. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, January 2007.

[10] F. Haq and T. Kunz. Simulation vs. emulation: Evaluating mobile ad hoc network routing protocols. In *Proceedings of the International Workshop on Wireless Ad-hoc Networks (IWWAN 2005)*, London, England, May 2005.

[11] S. Ivanov, A. Herms, and G. Lukas. Experimental validation of the ns-2 wireless model using simulation, emulation, and real network. In *Proceedings of the 4th Workshop on Mobile Ad-Hoc Networks (WMAN 2007)*, 2007.

[12] O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *Proceedings of The Second IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, Sydney, Australia, May 2005.

[13] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 126–137, 2003.

[14] G. Lu, B. Krishnamachari, and C. Raghavendra. An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[15] R. Musaloiu-E., C-J. M. Liang, and A. Terzis. Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks. In *IPSN '08*, 2008.

[16] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, Tampa, Florida, USA, November 2006.

[17] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of The Fourth International Conference on Information Processing in Sensor Networks. IPSN/SPOTS'05*, Los Angeles, CA, USA, April 2005.

[18] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava. Energy aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2):40–50, March 2002.

[19] J. Schiller, H. Ritter, A. Liers, and T. Voigt. Scatterweb - low power nodes and energy aware routing. In *Proceedings of Hawaii International Conference on System Sciences*, Hawaii, USA, 2005.

[20] R.C. Shah and J.M. Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, March 2002.

[21] V. Shnayder, M. Hempstead, Chen B., G.W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *2nd International Conference on Embedded Networked Sensor Systems (ACM SenSys)*, November 2004.

[22] B.L. Titzer, D.K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *Proceedings of the 4th international symposium on Information processing in sensor networks (IPSN)*, April 2005.

[23] T. Trathnigg, J. Moser, and R. Weisse. A low-cost energy measurement setup and improving the accuracy of energy simulators for wireless sensor networks. In *Proceedings of REALWSN 2008*, April 2008.

[24] T. van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, Los Angeles, California, USA, November 2003.

[25] T. Voigt and F. Österlind. CoReDac: Collision-free command-response data collection. In *13th IEEE Conference on Emerging Technologies and Factory Automation*, Hamburg, Germany, September 2008.

Chapter 8

COOJA/MSPSim: Interoperability Testing for Wireless Sensor Networks

Joakim Eriksson, Fredrik Österlind, Niclas Finne, Nicolas Tsiftes, Adam Dunkels, Thiemo Voigt
Swedish Institute of Computer Science
{joakime,fros,nfi,nvt,adam,thiemo}@sics.se

Robert Sauter, Pedro José Marrón
University of Bonn and Fraunhofer IAIS
{sauter,pjmarron}@cs.uni-bonn.de

abstract

Wireless sensor networks are moving towards emerging standards such as IP, ZigBee and WirelessHART which makes interoperability testing important. Interoperability testing is today performed through black-box testing with vendors physically meeting to test their equipment. Black-box testing can test interoperability but gives no detailed information of the internals in the nodes during the testing. Black-box testing is required because existing simulators cannot simultaneously simulate sensor nodes with different firmware. For standards such as IP and WirelessHART, a white-box interoperability testing approach is desired, since it gives details on both performance and clues about why tests succeeded or failed. To allow white-box testing, we propose a simulation-based approach to interoperability testing, where the firmware from different vendors is run in the same simulator.

We extend our MSPSim emulator and COOJA wireless sensor network simulator to support interoperable simulation of sensor nodes with firmware from different vendors. To demonstrate both cross-vendor interoperability

and the benefits of white-box interoperability testing, we run the state-of-the-art Contiki and TinyOS operating systems in a single simulation. Because of the white-box testing, we can do performance measurement and power profiling over both operating systems.

8.1 Introduction

Wireless sensor networks are distributed systems consisting of small, often battery-powered sensing devices that communicate using low-power wireless radios. Wireless sensor networks enable numerous applications ranging from water [27] and bridge monitoring [20] to predictive maintenance in industry [17].

The resource constraints of sensor devices have driven researchers to focus on performance rather than on modularity [23]. Consequentially, a number of highly efficient vertically integrated solutions such as Koala [21] and Dozer [2] have been developed. These low power data gathering applications operate on radio duty cycles less than 1%. The vertical integration, however, is by design non-interoperable and it is therefore difficult to exchange layers to achieve interoperability.

A vast body of research on sensor networks has yielded numerous different communication architectures and protocols. The rapid acceptance of sensor networks for industrial applications has, however, driven focus towards standardization and interoperability. Due to the proven industrial potential of wireless sensor networks, a number of standards have emerged during recent years. These include WirelessHART [15], ZigBee [28], and IPv6-based sensor networks [10].

The standardization of sensor networks makes interoperability a new requirement when implementing communication stacks and applications. Interoperability is difficult to verify, however, since sensor network simulators and prototyping tools lack the possibility to simulate heterogeneous sensor networks. For example, the widely used TOSSIM simulator [19] only simulates nodes running the TinyOS operating system [14]. These tools are thus unable to test the requirements that have arisen with the vision of an Internet of Things [9]. Instead, vendors are today required to physically meet to perform black-box testing of their equipment.

To meet the need for white-box testing of interoperability, we extend the Contiki simulator COOJA [22] and MSPSim, an instruction level emulator that is integrated in COOJA. Our simulator extensions enable simulations of heterogeneous sensor networks that consist of different operating systems and sensor devices. We also extend MSPSim with a power profiling mechanism similar to the software-based power profiler of the Contiki operating system [7]. Our experiments demonstrate that our combined simulator is not only a feasible tool for interoperability testing; it is also able to power

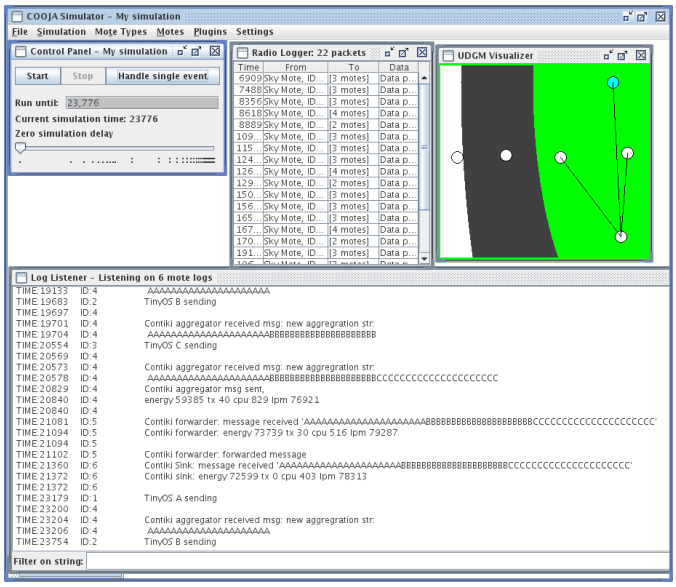

Figure 8.1: COOJA simulating a hierarchical, heterogeneous sensor network of TinyOS and Contiki nodes

profile nodes running different operating systems.

We evaluate our simulator with two state-of-the-art sensor node operating systems: Contiki and TinyOS. The Contiki operating system was the first operating system to support IP for sensor networks [4] and currently the only system that fulfills all the IPv6 Ready compliance requirements [10]. The TinyOS operating system is popular in academia and has been used to implement standard protocols [13].

The contribution of this paper is the design and evaluation of a heterogeneous simulator environment for interoperability testing, with which we demonstrate how white-box testing enables accurate and non-intrusive power profiling of different operating systems.

The rest of this paper is structured as follows. We describe the two simulators that we combine and the operating systems that we simulate in our experiments in Section 8.2. We outline the technical aspects of our implementation in Section 8.3. In Section 8.4, we evaluate the accuracy and ability of MSPsim and COOJA in heterogeneous sensor network simulations. Thereafter we discuss related work in Section 8.5, and conclude the paper in Section 8.6.

8.2 Background

Our white-box interoperability testing system builds on the COOJA and MSPSim simulators [22, 11]. COOJA is a cross-level sensor network simula-

```
while (running) {
  /* execute events */
  executeCycleEvents(cycles);
  executeTimeEvents(currentTime);

  /* fetch instruction to execute */
  op = memory[pc++] | (memory[pc++] << 8)
  instruction = decodeInstruction(op);
  switch(instruction) {
    case MOV:
      dst = readSrc(op);
      cycles += MOV_CYCLES;
      break;
    case ADD:
    ...
  }
}
```

Figure 8.2: The core emulation loop of MSPSim in pseudo-code. After handling queued events, the program counter (PC) is increased and the next instruction is decoded and executed.

tor. MSPSim can be used through COOJA to emulate sensor devices based on the popular MSP430 processor. We describe each of these two simulators as well as the two sensor network operating systems that we mainly target for testing interoperability.

8.2.1 The MSPSim Simulator

MSPSim [11] is a Java-based instruction level emulator of the MSP430 microprocessor series. In contrast with CPU-level emulators, it emulates complete sensor networking platforms such as the Tmote Sky [24] and ESB/2 [25]. MSPSim targets both realistic simulation with accurate timing for use as a research tool, and good debugging support for use as a development tool.

MSPSim combines cycle accurate interpretation of CPU instructions with a discrete-event based simulation of all other components, both internal and external. MSPSim uses an event-based execution kernel that enables accurate timing while keeping the host processor utilization as low as possible. We show an outline of the main execution loop in Figure 8.2.1. Before interpreting instructions, MSPSim executes all pending events in both event queues. Each queue handles events that are scheduled with a different perspective of time, with the first being based on CPU clock cycles, whereas the other is based on a high resolution clock. Most of the internal components

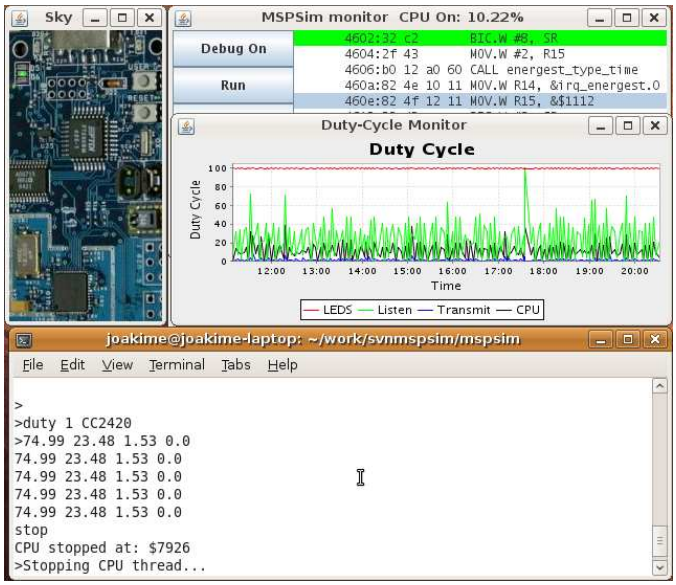

Figure 8.3: MSPSim emulating a Tmote Sky node running a program that periodically turns on the radio and sends data packets.

of the MSP430, such as the USART and the analog-to-digital converter use the event queue for clock cycles, while external components such as radio transceivers use the event queue for the the high resolution clock.

The emulator provides a programming interface for integration with simulation frameworks such as COOJA. In addition, the emulator can be extended with new mote types through a mote interface and I/O interfaces that correspond to the MSP430 I/O ports and serial communication ports.

MSPSim provides both debugging capabilities such as break points, watches, logging, and single stepping as well as statistics about the operating modes of the emulated components, statistics such as how much time the CPU has consumed in the different low-power modes. All features and information can be accessed either via a command line interface, or via the integration programming interfaces.

8.2.2 The COOJA Simulator

COOJA [22] is a flexible Java-based simulator initially designed for simulating networks of sensors running the Contiki operating system. COOJA simulates networks of sensor nodes where each node can be of a different type; differing not only in on-board software, but also in the simulated hardware. COOJA is flexible in that many parts of the simulator can be easily replaced or extended with additional functionality.

A simulated node in COOJA has three basic properties: its data memory, the node type, and its hardware peripherals. The node type may be shared

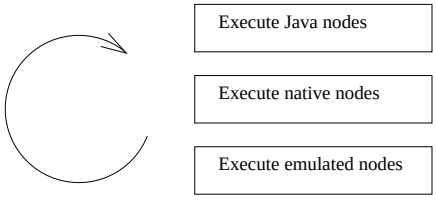

Figure 8.4: The main loop in COOJA executes nodes at different levels.

between several nodes and determines properties common to all these nodes. For example, nodes of the same type run the same program code on the same simulated hardware peripherals. Nodes of the same type are initialized with the same data memory, except for the node id. During execution, however, the data memories of the nodes will eventually differ after reacting to external stimuli.

COOJA can execute Contiki programs in two different ways. Either by running the program code as compiled native code directly on the host CPU, or by running compiled program code in MSPSim. COOJA is also able to simulate nodes developed in Java at the application level. All different approaches have advantages as well as disadvantages. Java-based nodes enable much faster simulations but do not run deployable code. Hence, they are useful for the development of e.g. distributed algorithms. Emulating nodes allows control and retrieval of more fine-grained execution details compared to Java-based nodes or nodes running native code. Finally, native code simulations are more efficient than node emulations and still simulate deployable code. Combining the different levels in the same simulation can give both an efficient simulation as well as fine-grained execution details on selected nodes.

8.2.3 Contiki

Contiki [5] is a sensor network operating system. Contiki supports three communication stacks: Rime [6], a light-weight layered communication stack that provides basic communication primitives on top of which more complex protocols are built, uIP [4] is a fully RFC compliant TCP/IPv4 stack for memory constrained systems, and uIPv6 [10], the world's smallest fully RFC compliant TCP/IPv6 stack.

Contiki has an on-line power profiling mechanism [8] which estimates the energy consumption by measuring the duration each component is in various modes such as low-power mode, transmitting.

8.2.4 TinyOS

TinyOS is a sensor network operating system popular in academia originally targeting hardware with just 512 Bytes of RAM [14]. The main difference to

other sensor network operating systems is the use of the specifically developed programming language nesC [12] that builds component abstractions on top of standard C. This programming language has to be used by application developers, since the application together with system components are used to generate a single binary image to be programmed to the sensor nodes. TinyOS is event-based, which is supported by special constructs in nesC, and requires also the application developer to follow this programming model.

The basic communication abstraction of TinyOS is a simple best-effort one-hop message transmission service. In addition to the frame format of the data link layer used by the radio chip, e.g., 802.15.4, TinyOS just adds one byte – the Active Message Type – to differentiate among up to 256 different software components as the intended destination on the receiver. Building on this abstraction, other protocols can be built. As an alternative to this, a partial IPv6 stack [13] has been added in recent TinyOS releases.

No mechanism for on-line estimation of power consumption is available. Instead, algorithms that require this information, e.g., for lifetime estimation and adaption of functionality, require an extensive prior evaluation by simulation to obtain estimates for inclusion in the deployment [18].

8.3 Implementation

By integrating MSPsim more closely into COOJA and adding some OS-specific support, we get a simulation tool that can simulate sensor networks consisting of both Contiki and TinyOS nodes.

8.3.1 Simulating TinyOS nodes

The main difference between simulating Contiki nodes and TinyOS nodes is that they use different node ID variables. Contiki uses *node_id* for its node ID while TinyOS uses the constant TOS_NODE_ID and the mutable Active Message address $TOS\_AM\_ADDRESS$. Initially, we added support in COOJA for setting just the node id, but initial experiments indicated that it is also practical to be able to set the Active Message address through COOJA.

In MSPSim, we made the emulation of the MSP430 and the CC2420 radio chip to be more accurate and complete. The initial emulation was limited to the subset of features used in only one of the operating systems. One important addition is the SFD capture interrupt that might be used for time stamping, etc.

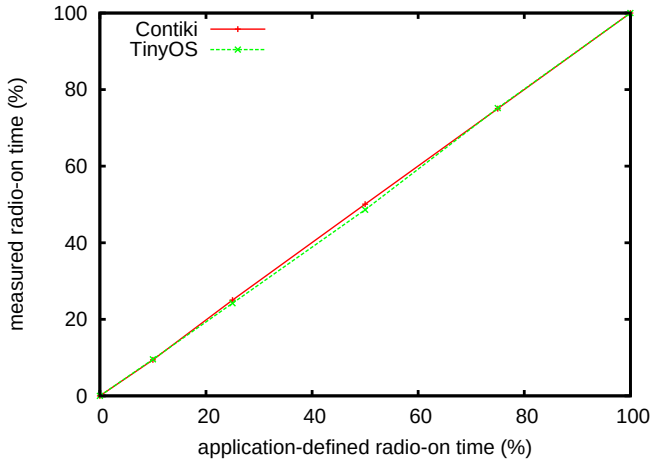

Figure 8.5: COOJA/MSPSim correctly measures the power consumption of both TinyOS and Contiki nodes

8.3.2 Power Profiling of all nodes

To get power profiling of non-Contiki firmwares in COOJA we also extended MSPSim with more detailed statistics for external components such as the CC2420 radio chip. The mechanism is similar to the built-in power profiling mechanism in Contiki. The information can be accessed per node from COOJA when power profiling is needed.

8.4 Evaluation

In this section we present results from experiments with the COOJA/MSP-Sim simulator.

8.4.1 Measuring Power Consumption

To evaluate whether the radio duty cycle measurement in MSPSim works as expected for both Contiki and TinyOS nodes, we have written TinyOS and Contiki applications that turn off the radio for 10%, 25%, 50%, 75% and 100% of the time. We show that the simulator is able to measure the radio duty cycle with the corresponding values.

Our results depicted in Figure 8.5 show that the differences between the expected and the measured values are very small for both operating systems, namely at maximum around 1%.

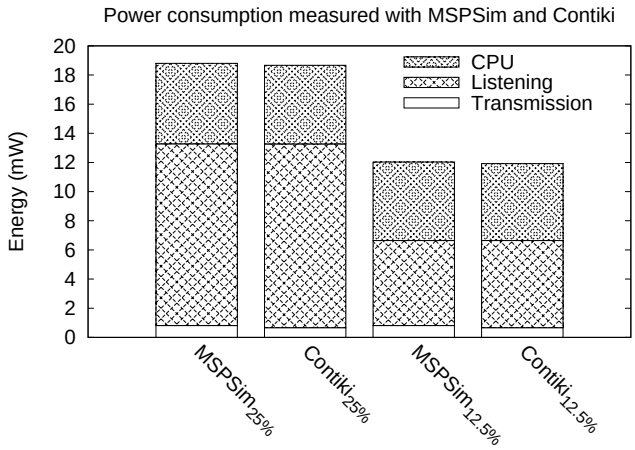

Figure 8.6: Energy estimation with MSPSim and Contiki for two duty cycle cases, 25% and 12.5%.

8.4.2 Measuring Power Consumption with MSPSim

To evaluate the accuracy of MSPSim's built-in power consumption measurements we compare with the energy consumption estimation provided by Contiki's power profiling.

The test application sends data packets at a regular interval and toggles the radio transceiver on and off with a given duty cycle.

We measure the energy consumption in MSPSim by printing the duty cycle of CPU active and the CC2420 radio listen and transmit modes as shown below:

```
>duty 1 "MSP430 Core.active" CC2420
10.39 74.99 23.48 1.53
10.39 74.99 23.48 1.53
10.39 74.99 23.48 1.53
...
```

The columns in the output are percentages of CPU activity and the modes of the CC2420 radio: power down, listen, and transmit. We calculate the energy consumption by multiplying the time spent in each mode with the respective power consumption in milliwatts.

The results in Figure 8.6 show that the energy estimations are very close to each other in both duty cycle cases. Maximum difference is around two percent on listen and CPU, but we observe a somewhat larger difference when estimating the power consumption of transmissions. The reason for this difference is that the built-in measurements in MSPSim have immediate knowledge when the radio switch over to transmission mode, while the Contiki counterpart must read status registers and therefore gets delayed

information about when transmission starts.

8.4.3 A Heterogeneous, Hierarchical Sensor Network

To demonstrate interoperability between Contiki and TinyOS in COOJA/MSPSim, we simulate a heterogeneous, hierarchical sensor network shown in Figure 8.3. The hierarchical sensornet consists of both TinyOS and Contiki nodes. The three TinyOS nodes placed in the right part of the top right plug-in in Figure 8.3 act as data sources that periodically send a data message with 20 bytes payload ("As", "Bs" or "Cs") to an aggregator node running Contiki. The aggregator node aggregates the data and sends it via a forwarder node in a multi-hop fashion to the sink (left-hand node in the plug-in). Both the forwarder and the sink node run Contiki.

Since the TinyOS frame format differs from the Contiki frame format, we have modified the Rime stack and the Chameleon module [6] to allow Contiki to understand packets from the TinyOS nodes.

In our experiments we send 200 packets from each TinyOS node. The aggregator node reduces the number of packets from 600 to 200. If this reduction of the number of packets leads to energy savings depends to a large extent on the MAC layer. In order to quantify the power savings, we have measured the power consumption with and without aggregation over two hops, i.e. from the aggregator node via the forwarder to the sink node. In these results, the aggregator either sends one large packets with 60 Bytes payload every three seconds or three shorter packets with 20 Bytes payload every second.

We use low power probing (LPP) as the underlying MAC layer [21]. LPP receivers periodically transmit probes. Essentially, probes are short packets that announce that the node in question is awake and ready to receive a data packet. After sending a probe, receivers keep their radio on for a short time to listen for data packets. A sender that has a packet to be sent turns on its radio waiting for a probe from a neighbor it wants to send to. On the reception of a probe from a potential receiver, the node sends a small hardware ACK that is followed by the actual data packet. We have used the default LPP configuration available in Contiki that sends on average four probes per second and that keeps the radio on for $\frac{1}{128}$ seconds after each probe.

Our results are shown in Figure 8.7 and Figure 8.8. Figure 8.7 shows that the reduction of the number of packets also leads to reduced power consumption for transmitting packets (TX power consumption) for the forwarder and the aggregator nodes. The TX power consumption for the sink nodes is not influenced which indicates that the reduced power consumption is not caused by a change of the number probing packets but by the reduced number of header bytes that needs to be transmitted.

As expected, the power consumption for the radio in listen mode (RX

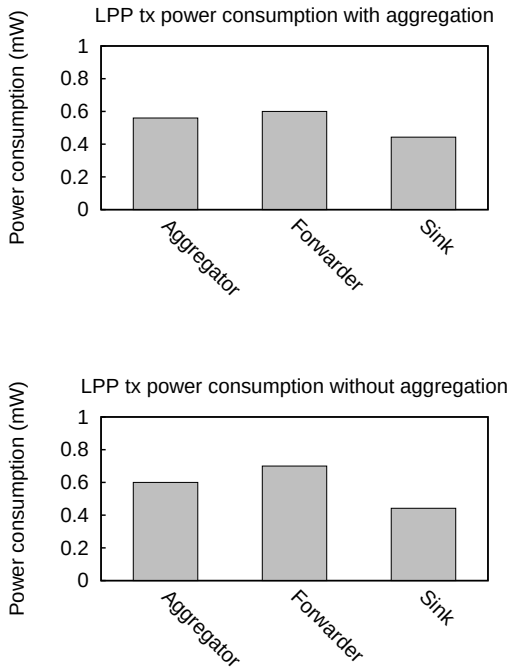

Figure 8.7: Aggregation reduces TX power consumption.

power consumption) is much lower than the TX power consumption. Using LPP, a node that wants to transmit a packet needs to turn the radio on and keep it in listening mode until it receives a probe from the receiver. Without aggregation, the number of packets a node transmits increases and hence the time a transmitter needs to keep the radio on. The results in Figure 8.8 show that the power consumption more than doubles without aggregation.

8.4.4 Interoperability Tests

To validate our simulation tool's capability for interoperability tests we perform an experiment where we develop the same networked application in in both TinyOS and Contiki.

We implement an application that broadcasts packets to its neighbors and when it receives packets it counts all neighbors and show the neighbor count on the leds. The application for TinyOS use the standard Active Message communication framework and the Contiki application is designed to replicate the TinyOS application. Since TinyOS use 802.15.4 we replace the default Contiki MAC protocol with 802.15.4 and add the TinyOS active message type as the first byte of the payload.

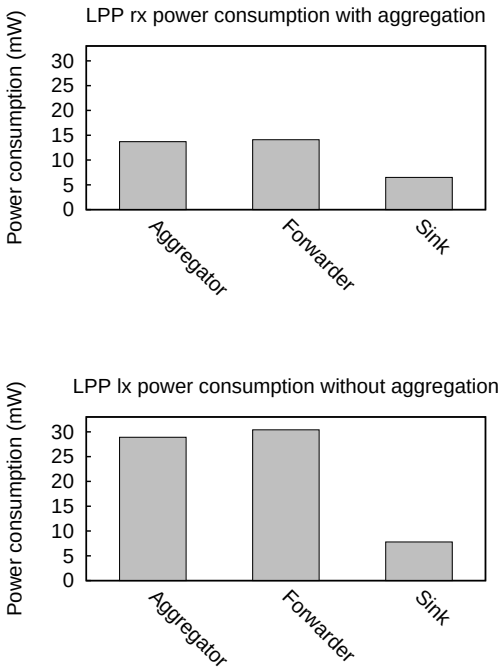

Figure 8.8: Aggregation reduces RX power consumption

Figure 8.9 shows the result of the test. All nodes communicated and counted neighbors as expected.

This interoperability test validates a very basic application protocol on top of 802.15.4 but still shows that it is possible to perform interoperability tests using COOJA/MSPSim.

8.5 Related Work

During recent years, a number of wireless sensor networking simulators have been developed. Most of these cannot be used for interoperability testing. Many simulators are developed for specific operating systems. An example is the TOSSIM simulator [19] that only simulates nodes running the TinyOS operating system [14]. These simulators usually run the same application code, but it is compiled for the simulator's host and not directly deployable without recompilation for the sensor device hardware. Other simulators such as Castalia [1] and MiXiM [16] do not simulate the operating system and the application code, but simulate at a higher level and with a focus on network related aspects.

Instruction level simulators such as MSPSim and Avrora [26] are able to

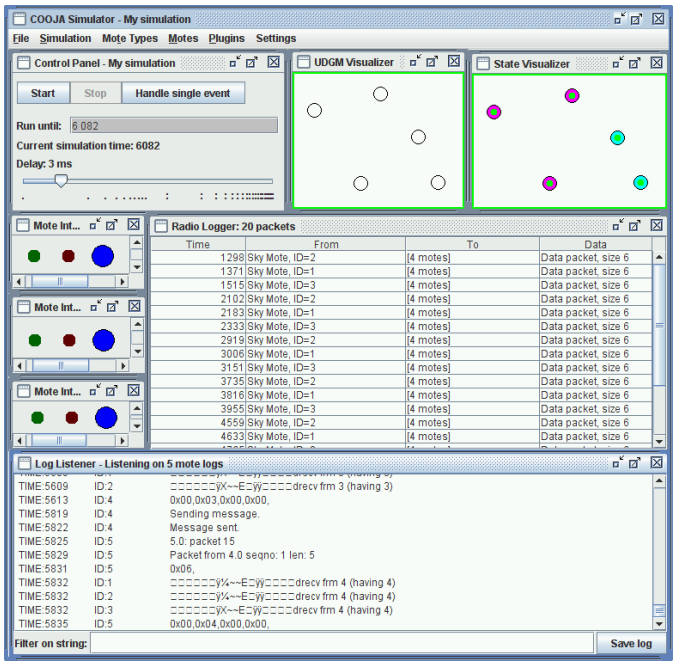

Figure 8.9: Basic interoperability test with three TinyOS nodes and two Contiki nodes communicating. Some of the nodes' led-panels are shown, and debug printouts are visible in the log window.

simulate nodes running different operating systems since they operate at the instruction set level. Avrora emulates Mica2 sensor nodes and can emulate several nodes simultaneously. It supports communication by emulating the Chipcon CC1000 radio chip. AvroraZ [3] is an extension of Avrora that provides a detailed emulation of the Texas Instruments Chipcon CC2420 radio chip including an indoor radio model. Avrora and AvroraZ run multiple nodes using Java threads while our simulation tool COOJA/MSPSim schedules the nodes explicitly. In contrast to these efforts, we have shown that COOJA/MSPSim can simulate nodes with different operating systems in the same simulation and perform power profiling.

8.6 Conclusions

We present a simulation-based approach for white-box interoperability testing in sensor networks. By combining simulations at network and hardware layer, we enable white-box testing in heterogeneous sensor network environments. Our approach makes interoperability testing more practical and transparent by allowing repeatable and fine-grained control of experiments. We demonstrate our simulation tool through experiments in which applications of two different operating systems exchange protocol data. Further-

more, we show that the new white-box testing environment allows accurate and non-intrusive power consumption measurements at the network scale.

More information about COOJA and MSPSim, including download links, can be found at:

http://www.sics.se/contiki/

and

http://www.sics.se/project/mspsim/

8.7 Acknowledgments

This work has been partially supported by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053. This work was also partly financed by VINNOVA, the Swedish Agency for Innovation Systems.

Bibliography

[1] A. Boulis. Castalia: revealing pitfalls in designing distributed algorithms in WSN. In *Poster proceedings of the 5th international conference on Embedded networked sensor systems*, pages 407–408, 2007.

[2] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *IPSN '07*, 2007.

[3] Rodolfo de Paz Alberola and Dirk Pesch. Avroraz: Extending avrora with an ieee 802.15.4 compliant radio chip mode. In *3rd ACM International Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks*, Vancouver, Canada, October 2008.

[4] A. Dunkels. Full TCP/IP for 8-bit architectures. In *Proceedings of The First International Conference on Mobile Systems, Applications, and Services (MOBISYS '03)*, May 2003.

[5] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, November 2004.

[6] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *Proceedings of the Fifth ACM Conference on Networked Embedded Sensor Systems (SenSys 2007)*, Sydney, Australia, November 2007.

[7] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based online energy estimation for sensor nodes. In *Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV)*, Cork, Ireland, June 2007.

[8] A. Dunkels, F. Österlind, N. Tsiftes, and Z. He. Software-based online energy estimation for sensor nodes. In *Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV)*, Cork, Ireland, June 2007.

[9] A. Dunkels and J-P. Vasseur. IP for Smart Objects, September 2008. IPSO Alliance White Paper #1.

[10] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels. Making Sensor Networks IPv6 Ready. In *Proceedings of the Sixth ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008)*, Raleigh, North Carolina, USA, November 2008.

[11] J. Eriksson, A. Dunkels, N. Finne, F. Österlind, and T. Voigt. Mspsim – an extensible simulator for msp430-equipped sensor boards. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, January 2007.

[12] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, 2003.

[13] Matus Harvan and Jürgen Schönwälder. A 6lowpan implementation for TinyOS 2.0. In *Proc. of the 6th GI/ITG KuVS Fachgespräch "Wireless Sensor Networks"*, Aachen, 2007.

[14] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, November 2000.

[15] F. Kim, A.and Hekland, S. Petersen, and P. Doyle. When hart goes wireless: Understanding and implementing the wirelesshart standard. In *13th IEEE Conference on Emerging Technologies and Factory Automation*, Hamburg, Germany, September 2008.

[16] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Klein Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating wireless and mobile networks in omnet++ the mixim vision. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–8, ICST, Brussels, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[17] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *Proceedings of the 3rd international conference on*

Embedded networked sensor systems (SenSyS), San Diego, CA, USA, November 2005.

[18] Andreas Lachenmann, Pedro José Marrón, Daniel Minder, and Kurt Rothermel. Meeting lifetime goals with energy levels. In *Proc. of the 5th ACM Conference on Embedded Networked Sensor Systems*, 2007.

[19] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 126–137, 2003.

[20] P. Marrón, R. Sauter, O. Saukh, M. Gauger, and K. Rothermel. Challenges of complex data processing in real world sensor network deployments. In *Proceedings of ACM Workshop on Real-World Wireless Sensor Networks (REALWSN'06)*, Uppsala, Sweden, June 2006.

[21] R. Musaloiu-E., C-J. M. Liang, and A. Terzis. Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks. In *IPSN '08*, 2008.

[22] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, Tampa, Florida, USA, November 2006.

[23] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys*, 2005.

[24] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of The Fourth International Conference on Information Processing in Sensor Networks. IPSN/SPOTS'05*, Los Angeles, CA, USA, April 2005.

[25] J. Schiller, H. Ritter, A. Liers, and T. Voigt. Scatterweb - low power nodes and energy aware routing. In *Proceedings of Hawaii International Conference on System Sciences*, Hawaii, USA, 2005.

[26] B.L. Titzer, D.K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *Proceedings of the 4th international symposium on Information processing in sensor networks (IPSN)*, April 2005.

[27] T. Voigt, F. Österlind, N. Finne, N. Tsiftes, Z. He, J. Eriksson, A. Dunkels, U. Båmsted, J. Schiller, and K. Hjort. Sensor networking in aquatic environments – experiences and new challenges. In *Workshop*

on Practical Issues in Building Sensor Network Applications (SenseApp 2007), Dublin, Ireland, October 2007.

[28] Zigbee. Web page. 2007-11-21. http://www.zigbee.org.

Recent licentiate theses from the Department of Information Technology

2008-003 Andreas Hellander: *Numerical Simulation of Well Stirred Biochemical Reaction Networks Governed by the Master Equation*

2008-002 Ioana Rodhe: *Query Authentication and Data Confidentiality in Wireless Sensor Networks*

2008-001 Mattias Wiggberg: *Unwinding Processes in Computer Science Student Projects*

2007-006 Björn Halvarsson: *Interaction Analysis and Control of Bioreactors for Nitrogen Removal*

2007-005 Mahen Jayawardena: *Parallel Algorithms and Implementations for Genetic Analysis of Quantitative Traits*

2007-004 Olof Rensfelt: *Tools and Methods for Evaluation of Overlay Networks*

2007-003 Thabotharan Kathiravelu: *Towards Content Distribution in Opportunistic Networks*

2007-002 Jonas Boustedt: *Students Working with a Large Software System: Experiences and Understandings*

2007-001 Manivasakan Sabesan: *Querying Mediated Web Services*

2006-012 Stefan Blomkvist: *User-Centred Design and Agile Development of IT Systems*

2006-011 Åsa Cajander: *Values and Perspectives Affecting IT Systems Development and Usability Work*

2006-010 Henrik Johansson: *Performance Characterization and Evaluation of Parallel PDE Solvers*

2006-009 Eddie Wadbro: *Topology Optimization for Acoustic Wave Propagation Problems*

UPPSALA
UNIVERSITET

Department of Information Technology, Uppsala University, Sweden