CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Energy Consumption Awareness for Resource-Constrained Devices

Edgar M. Silva
Pedro Maló
Michele Albano*

*CISTER Research Centre

CISTER-TR-160405

2016/06/27

Energy Consumption Awareness for Resource-Constrained Devices

Edgar M. Silva, Pedro Maló, Michele Albano*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: mialb@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Energy Consumption Awareness for Resource-Constrained Devices

Edgar M. Silva* and Pedro Maló†
Universidade Nova de Lisboa (UNL),
Faculdade de Ciências e Tecnologia (FCT).
UNINOVA, Centro de Tecnologia e Sistemas (CTS), Portugal
Email: {*ems, †pmm}@uninova.pt

Michele Albano
CISTER Research Unit,
ISEP/INESC-TEC,
Polytechnic Institute of Porto, Portugal
Email: mialb@isep.ipp.pt

***Abstract*—The devices running embedded applications tend to be battery-powered, and the energy efficiency of their operations is an important enabler for the wide adoption of the Internet-of-Things. Optimization of energy usage depends on modelling power consumption. A model-based simulation must consider parameters that depend on the device used, the operating system, and the distributed application under study. A realistic simulation thus depends on knowledge regarding how and when devices consume energy. Direct measurement in wireless sensors is a common approach to evaluate the power consumed by the embedded devices in their different execution states. This paper presents an approach to direct measurement of consumed energy. We present the architecture and the measurement process that were implemented. Details are given regarding the setup of the experimental tests, and a discussion of the results hints at which architecture is the best for each application under study. The presented methodology can be easily extended to new architectures and applications, to streamline the process of building realistic models of power consumption.**

I. Introduction

The Internet-of-Things (IoT) is an active field of research, since it is on the verge of the maturity needed to provide added value to both industrial processes and people's everyday life. The Cyber Physical Systems (CPSs) that make up the IoT address different application domains and services to target different IoT scenarios, which span from Smart Cities to Domotics (Smart Buildings), to Intelligent Transportation Systems, to eHealth, etc. A common aspect of these scenarios is that the involved devices tend to be embedded and resource constrained (low power, small processing power, limited storage capabilities, etc.). In particular, the energy available to the devices is limited, since in most scenarios they are powered by batteries. Energy saving is thus one of the most important research topics in this area.

Research efforts, like other human activities, proceed by trends. Past trends on energy saving were considering mainly wireless sensor networks, and focused on minimizing the energy spent for the communication activities of the devices, in particular by studying network protocols and MAC layers. Within network protocols, the goal was to minimize the number of packets sent to perform a given activity in the sensor network as a whole; within MAC layers, the objective was to organize transmission and reception activities to maximize the time that the each device's wireless interface spent in the sleep state. More recent activities have generalized this vision in many ways. The focus has moved from wireless sensor networks to CPSs, which are devices not limited to data collection activities; the architecture of distributed systems and their Operating Systems (OSs) have been object of analysis, to maximize their energy efficiency; computations performed by devices has become part of the game, and for example it has been an important parameter in deciding which cryptographic algorithms can be used by constrained devices.

An accurate analysis of power consumption is a fundamental support for other R&D activities, since it is instrumental to predict expected devices lifetime and to allow developers to optimize energy consumption in distributed IoT applications. Two primary approaches have been followed up to now. Direct measurement is performed by engineering the devices to measure energy consumption while they are executing their distributed applications; this approach is accurate but it is very expensive since it involves engineering every single device involved, and it is not practical in large distributed applications. The second approach is related to simulation models, which are more practical and scale better, but which depend on direct measurement over smaller scenarios to collect the parameters used to enhance the realism of the model.

This paper follows the direct measurement approach. An architecture is presented, and implemented over a testbed. An approach to the set up of experiments is given, and the results of basic experimental tests are presented, to showcase how this methodology can be applied to investigate energy efficiency.

II. Background Information

A. Devices: Hardware and Operating Systems

Hardware technology for sensor nodes manufacturing is changing due to the advances in Micro-Electro Mechanical System (MEMS), and wireless communications and digital electronics have led to smaller and cheaper sensor nodes [1]. A wireless sensor node is composed by a micro-controller, memory, timer, transceiver, battery, sensing unit and Analogical-Digital Converter (ADC) [2]. Figure **??** shows a simplified block diagram of a sensor node typical architecture.

To manage efficiently sensor nodes' constrained hardware resources (memory, processor, communication interfaces and energy) and to allow access to system resources by concurrent applications, several Operating Systems (OS) have been proposed. According to the authors in [2][3] the most popular OS for sensors are TinyOS, Contiki, MANTIS, Nano-RK and

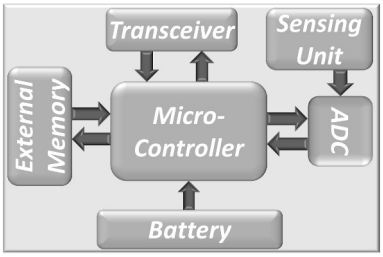

Fig. 1. Typical Sensor Node Architecture.

LiteOS. The study conducted in [4], based on some known scientific and engineering online databases including IEEE Xplore, ACM Digital Library and Science Direct, states that TinyOS (81%) and Contiki (9%) together account with 90% of the global references.

B. TinyOS Operating System

TinyOS [5], developed at the University of California, Berkeley, is a multi-platform, component-based and open-source OS. It presents a footprint of about 400 bytes, it falls under the monolithic architecture class and its execution model is event-based. TinyOS is intended to support concurrent applications with low memory requirements. The applications are designed as interaction between components, the latter being independent computational entities that expose one or more interfaces. The components are written and *wired* together using nesC (Network Embedded System C) [6], a component-based programming language based on C.

The earlier versions of TinyOS did not provide multithreading and the programming was done following a pure event driven model. Since version 2.1 TinyOS simulates concurrency using TOS threads [7], which are lightweight (context switches and system calls introduce an overhead of 0.92% or less) and use either a non-preemptive First-In-First-Out (FIFO) scheduling or the Earliest Deadline First (EDF) scheduling [5]. These two algorithms present known disadvantages (FIFO's waiting time depends on task execution time and EDF does not produce a feasible schedule when task concur for resources), and it can be concluded that TinyOS does not provide a solid real-time scheduling algorithm. Other features supported by TinyOS are efficient memory safety [8], support for communication through a number of routing protocols, among them 6lowpan [9], and MAC layers, among them IEEE 802.15.4, virtualization mechanisms to provide independent instances of resources shared between components, and a single level file system, the latter justified by the assumption that at, any given point in time, only a single application is accessing it. The TinyOS provides further features, such as database support (TinyDB [10]), security for communications (TinySec [11]) and simulation of TimyOS applications (TOSSIM [12], described in Subsection **??**). TinyOS widely adoption is also due to its extensive documentation, which can be found on the TinyOS home page (http://www.tinyos.net).

C. Simulators

Usage of simulators allows to set up a measurement pipeline by means of programming, i.e.: without working on the hardware of devices. The approach scales well because devices can be instantiated programmatically, while direct measurement needs preparation work on each and any physical device involved in the scenario. This subsection presents the most widely used simulators tailored to wireless sensor networks, to investigate how to maximize the impact of our direct measurement approach. We are here disregarding general-purpose simulators such as network simulator 2 and 3, since the few that are complete enough to provide energy information, are very complex simulation platforms, possess a steep learning curve, and can provide energy information regarding communication activities only.

1) TOSSIM: TOSSIM is a C/C++ library included in the TinyOS framework, and it works as a simulation tool for TinyOS applications. TOSSIM replaces low-level hardware components with simulated implementations, and it includes models for CPUs, ADCs, clocks, timers, flash memories and radio components. To perform a simulation, it is necessary to write a program that configures the simulation and runs it. The code used for the components simulated in TOSSIM can be the same nesC code that is deployed onto nodes. The abstract hardware model used by TOSSIM makes it impossible to capture low-level details of timings and interrupts, which are important for accurate power analysis. Moreover, TOSSIM is a platform-specific (MicaZ) and OS-specific (TinyOS) tool.

PowerTOSSIM [13] is an extension to TOSSIM that enables the estimation of node power consumption. It can be customized for different platforms, and it is shipped with a detailed model for hardware energy consumption of the Mica2 sensor node platform, built by extensive application-level benchmarking. Simulation results for the energy consumption of each node are achieved by applying the node activity trace to the detailed hardware model of the nodes. PowerTOSSIM authors ensure that it is able to achieve results within 0.4513% of the power consumption of real hardware nodes.

2) Avrora: Like most simulators, Avrora [14] is event-driven and thus based on discrete time. It is open-source and widely used, but it is able to simulate the AVRMCU core only. The simulator is a cycle-accurate instruction level simulator, and scales to networks of up to 10,000 nodes. It is language and operating system independent, presents support for sensor platforms such as Mica2 and MicaZ, and run AVR elf-binary or assembly codes. Avrora is written in Java, and each hardware component is represented as an object-oriented class. Avrora enables the use of monitors to retrieve useful information from the application simulation, both at runtime (e.g.: current LEDs state) and as a summary at the end of the simulation (e.g.: total energy consumption). Avrora is not actively maintained and it does not provide extensions for CPU architectures different than the AVRMCU cores, making it a platform-specific simulator.

3) Cooja: Cooja is devoted to simulating the Contiki OS, on either TI-MSP430 or AtmelAVR microcontrollers. It enables simultaneous simulations at the network, operating system and machine code instruction set level [15], to verify application before being uploaded to sensor nodes. Cooja is designed to be flexible, and each node can differ not only in on-board software, but also in the simulated hardware. Contiki programs can be executed either by running compiled

TABLE I. SIMULATORS COMPARATIVE ANALYSIS.

Simulator:	TOSSIM:	Avrora:	Cooja:
Simulation Level	Operating System	Instruction Level	Network, OS and machine code
Hardware Representation	Abstract Hardware Model	Object Classes (Java)	Not Available
Simulation Interface	C++/Python	Java	Java
Energy Consumption	PowerTOSSIM	Yes	No
Hardware Platform	MicaZ	AVRMCU cores (Mica2 and MicaZ)	TI MSP430 cores, Atmel AVR cores

code directly on the host CPU, or by emulating the compiled program code in an instruction-level TI-MSP430 emulator.

4) Comparative Analysis: In the following, the three simulators of choice (TOSSIM, Avrora and the Cooja) are compared to highlight their advantages and disadvantages. Results are summarized in Table **??**.

The simulators operate at different levels. With TOSSIM, the applications are written in nesC and converted into the TOSSIM simulation code. Avrora is capable of simulating machine code, AVR elf-binary or assembly code. Cooja is able to simulate Contiki applications as well as machine code. Thus, Avrora and Cooja are considered to be language and OS independent, while TOSSIM is dependent on the programming language and OS [14].

Hardware is simulated with different approaches. TOSSIM uses abstract hardware models to represent the device components, making it hard to capture low-level details important to the energy analysis. On the other hand, powerTOSSIM introduces a detailed model for the hardware energy consumption built from real-life tests of the Mica2 platform, and it can be extended to other platforms. Avrora represents the hardware through Java classes, but supports the AVRMCU core only, and does not provide extensions for other CPU architectures. Avrora is capable of performing energy consumption simulations, but is known to have some issues regarding the simulation of some components, and it is not actively maintained. Concerning Cooja, no information on this aspect was found, and it is believed to lack any kind of power consumption simulation tool.

The presented discussion led us to focus our measuring efforts, described in the rest of the paper, on applications running over TinyOS, since this OS has got the largest market share and its simulator powerTOSSIM is the only one compatible with a deep analysis of energy consumption.

III. ENERGY MEASUREMENT PROCESS

The proposed process, represented in Figure **??**, is based on the basic formula that says that $P = VI$ (consumed power is equal to the tension multiplied by the current), and it involves direct measurement of both tension and current on a System Under Test (SUT), which is a device that is executing a specified application and that is instrumented to allow the direct measurement process. As shown in the figure, the measurement process relies on two main blocks, called Circuits and Micro-Controller, which are used together to attain a reasonable resolution of consumed energy.

The Circuits block takes care of transforming the tension and current physical values into analog signals. An analysis of the operating values for the devices points out that the sensor nodes, powered by two batteries having maximum voltage between 3V and 3.3V, can work as long as batteries can provide at least 2.1 V. Thus, the Circuit block was developed to measure voltages between 1.65V to 3.3V. For the current signal, no values can be excluded, and the signal is obtained using a common sensing resistor assembly (a low resistor value so it can not interfere relevantly in the SUT energy consumption). For both elements, each measuring unit splits the signal in the middle into two part, and the following unit is tuned on the top or lower part of the signal, to gain an extra bit of definition for each measuring unit in the Circuit block.

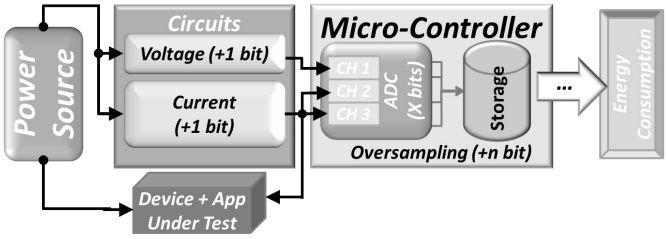

Fig. 2. Energy Measurement Process.

The Micro-Controller produces the digital values corresponding to the current and tension analog signal. The block has got at least three Analog-to-Digital Converter (ADC) inputs, a good ADC transformer, a reasonable micro-controller clock speed and some storage capabilities. The block can easily transform the incoming analog signal into bits. It is also possible to add bits using software, using the oversampling technique. The theory says that collecting 4^n additional samples leads to getting n more bits in the measured values, but the implementation of this approach has the drawback of a direct impact on the sampling rate, and the application of the technique represents a trade-off to be considered at design time. Currently, the Micro-Controller block is implemented in a Arduino DUE (Table **??** summarizes its specifications) coupled with a Micro-SD card shield for storage.

TABLE II. ARDUINO DUE - SPECIFICATIONS (RESUMED).

Microcontroller:	AT91SAM3X8E	SRAM:	96 KB
Operating Voltage:	3.3 V	**Clock Speed:**	84 MHz
Digital I/O Pins:	54	**Flash Memory:**	512 KB
Analog Inputs:	16	**ADC Resolution:**	12-bit 1Msps

The implemented SUT aim at a final precision of 14-bit for both voltage and current signals, obtained one bit by hardware, plus 12-bits given by the ADC and another one using the oversampling method. The sampling rate for the measurements ended up being 34 KHz (one sample every $\sim 29.5\mu s$).

IV. TESTS

The tests were designed based on an analysis on the market share of IoT devices, and on the study on OS reported in

Section II. Four different devices were selected, having similar features and compatible with both TinyOS and Contiki OS. Figure **??** presents the list of devices used in the experimental tests.

	Platform			Microcontroller		Memory		Radio
	Design	OS's		Ref.	Prog. Flash	Data RAM	Ext. Flash	RF Chip
XM1000	TelosB	TinyOS	Contiki	MSP430 @16MHz	116KB	8KB	1MB	TI - CC2420
CM5000				MSP430 @8MHz	48KB	10KB	1MB	
CM3300				MSP430 @8MHz	48KB	10KB	1MB	
XM2110CA	IRIS			ATmega1281 @8MHz	128KB	8KB	-	Atmel-RF230

Fig. 3. Devices Selected to be Tested.

The XM1000, CM5000 and CM3300 are TelosB platforms, using a MSP430 microcontroller and a Texas Instrument CC2420 wireless interface. The XM2110 is an Iris sensor, using an ATmega 1281 microcontroller and an Atmel-RF 230 wireless interface. The CM3300 device has got an amplifier to provide more power to the antenna. One more difference between the XM1000 and the other two TelosB devices, is that the XM1000 has got a faster CPU (its MSP430 is set at 16 MHz vs the 8 MHz of the other two TelosBs), a larger programmable flash chip (116 KB vs 48 KB of the other devices) but a smaller RAM (10 KB vs 8 KB of the other two devices). The Iris sensor's processor is set at 8 MHz, it has got a larger flash memory (128 KB) but just 8 KB of RAM. Since it is the market leader, all applications were executed over tinyOS (version 2.1.2).

Four simple applications were selected for this preliminary study. The simplest application (**Empty**) is a void application, and it helped studying how the devices take care of sleep states, and provide a baseline for the power consumption in the idle state. Since the most important job for sensor nodes is to sense the environment, the second application (**Timer 1000 ms**) studied how the timers are managed by the OS and the devices, by setting up and firing a timer with a fixed period of 1 second. Finally, three applications (**Blink Led 0, Blink Led 1, Blink Led 2**) were switching periodically on / off one of the three LEDs of the device, and they were used to study how these operations are scheduled, and how much power the LEDs consume. In the experiments, the LEDs were turned on and off with the *toggle* function of TinyOS. In the case of **Blink Led 0**, the LED stayed on for 500 ms, then off for 500 ms, and so on for CM3300, XM2110 and CM5000; the led stayed in the on and off states for 1s in the case of XM1000. In the **Blink Led 1** and **Blink Led 2** applications, all LEDs stayed on and off for $1s$ at a time.

Table **??** shows the memory footprint of programming each application on the respective device.

TABLE III. PROGRAMMED APPLICATIONS SIZE (BYTES).

	Empty (ROM/RAM):	**Timer 1000ms (ROM/RAM):**	**Blink Led 0, 1 , 2 (ROM/RAM):**
CM3300:	1320/6	2250/36	2420/56
XM1000:	1244/6	2174/36	2344/56
XM2110:	754/4	2088/33	2182/51
CM5000:	1320/6	2250/36	2420/56

V. RESULTS

The data collected from the tests is represented in the Figures **??**, **??**, **??**, **??**, **??**. The results made it clear that the devices, which are sharing the same OS and applications, have got different energy consumption.

From the results regarding the Empty application, it appears a first difference between MSP430 controller and ATmega1281 (Iris). ATmega1281 is put to sleep completely, while the MSP430 needs to wake every $1.85s$ to verify if it has received any interrupt to process. This result is confirmed on the other graphs too, since the MSP430-based sensors have got more consumption spikes. The behaviour is related to the lack of a wake up interrupt of the TeloSB microcontroller, causing it to sleep for a fixed period, and then waking up to verify if it has received data to process.

Another result from the Empty application is that the amplifier used in CM3300 devices has an extra energy consumption of 5 mA (as mentioned in its datasheet), even when the radio is not part of the picture. Thus, the CM3300 sensor has got no way to switch off the amplifier, at least with the current TinyOS libraries.

Finally, results from the blink leds applications shows that significant differences exist between devices, which are directly related to the led color used. For example, and excluding the CM3300 since it consumes always the most energy for its amplifier, the Iris mote consume more energy than the others when it is blinking its Led 0. In the case of Led 1, the XM1000 is the mote that consumes the most energy.

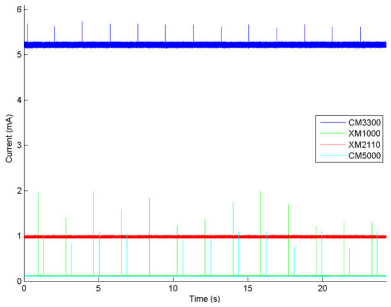

Fig. 4. Current Results for the Empty Application.

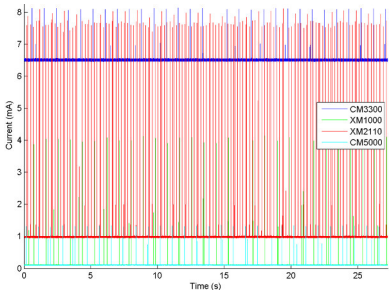

Fig. 5. Current Results for Firing a Timer each 1000ms.

Based on the results, it is possible for an application designer to select the mote that has got the longer lifetime. For example, an application that can sleep for a long time and get back to work only when receiving an interrupt, could be deployed on an Iris. Another conclusion that can be drawn

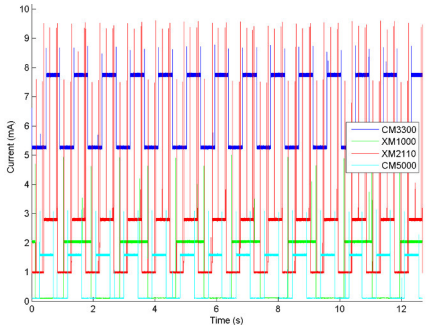

Fig. 6. Current Results for Blink Application (Led 0).

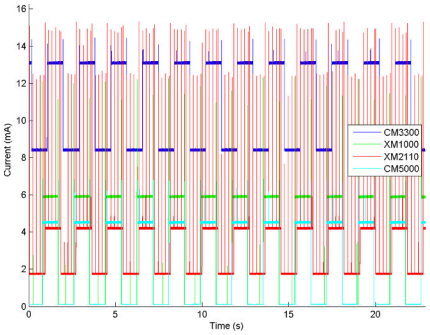

Fig. 7. Current Results for Blink Application (Led 1).

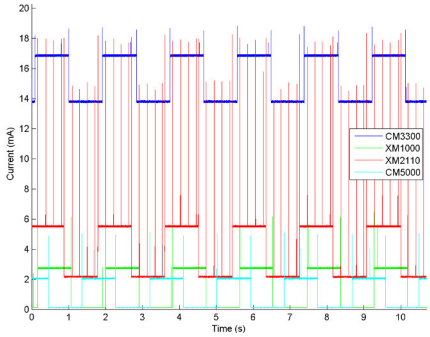

Fig. 8. Current Results for Blink Application (Led 2).

is that it would be better not to use motes equipped with the amplifier for most applications, since it is currently impossible to switch it off and thus any application on that platform would always consume a lot of energy.

VI. Conclusions

A number of conclusions can be drawn from the results. It is hard for the programmer to have complete knowledge of the energy consumption of his applications, since many low level details are hidden by the OS, and the details of the operations executed by the OS are device-dependant. What started as a work to tune up the energy model for energy simulations with data taken from real hardware, ended up being a critical analysis of how the same application code is executed on different platforms. For example, the results highlighted that some capabilities of the OS are not implemented yet on some hardware - such as switching off the amplifier of the radio on the CM3300 sensor. Thus, it appears of the utmost importance to have access to data measured directly on the hardware devices with the correct configurations and software, to be able to predict energy consumption of complex applications in a realistic manner.

Future work involves the definition of a complete architecture, in terms of hardware and software, to facilitate the direct measurement of energy consumption, to be distilled into information to be used in model-based simulations of energy consumption. The approach will streamline how data is included into the model-based simulators, to allow the application designer to receive realistic and accurate analysis of the consumed energy by simulation only.

Acknowledgment

We would like to thank to João Rodrigues for his contribution in testing the devices. This research work was partially supported by national funds via the FCT Fundacao para a Ciencia e a Tecnologia and funds provided by the European Commission in the scope of ECSEL/H2020-662189 MANTIS and ARTEMIS/FCT-332987 Arrowhead RTD projects.

References

[1] J. Yicka, B. Mukherjeea, and D. Ghosal, "Wireless sensor network survey", Comput. Networks, vol. 58, pp. 2292-2330, 2008.

[2] M. O. Farooq and T. Kunz, "Operating systems for wireless sensor networks: a survey", Sensors, vol. 11, pp. 5900-5930, 2011.

[3] A. Moschitta and I. Neri, "Power consumption Assessment in Wireless Sensor Networks", in ICT - Energy - Concepts Towards Zero - Power Information and Communication Technology, D. G. Fagas, Ed. 2014.

[4] R. Lajara, J. Pelegr-Sebasti and J. J. Perez Solano, "Power consumption analysis of operating systems for wireless sensor networks", Sensors, vol. 10, pp. 5809-5826, Jan. 2010.

[5] P. Levis, et al., TinyOS: An operating system for sensor networks, in Ambient Intelligence, pp. 115-148, 2005.

[6] 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 PLDI, vol. 38, pp. 1-11, 2003.

[7] K. Klues, et al., TOSThreads: thread-safe and non-invasive preemption in TinyOS, Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems. pp. 127-140, 2009.

[8] N. Cooprider, W. Archer, E. Eide, D. Gay, and . Regehr, Efficient memory safety for TinyOS, in Proceedings of the 5th international conference on Embedded networked sensor systems, pp. 205-218, 2007.

[9] G. Montenegro, N. Kushalnagar, . Hui, and D. Culler, Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC, vol. RFC4944. pp. 1-30, 2007.

[10] S. R. Madden, M. J. Franklin, . M. Hellerstein, and W. Hong, TinyDB: an acquisitional query processing system for sensor networks, ACM Transactions on Database Systems, vol. 30. pp. 122-173, 2005.

[11] C. Karlof, N. Sastry, and D. Wagner, TinySec: a link layer security architecture for wireless sensor networks. p. 162, 2004.

[12] P. Levis, N. Lee, M. Welsh, and D. Culler, TOSSIM: accurate and scalable simulation of entire TinyOS applications, in Proceedings of the 1st international conference on Embedded networked sensor systems, pp. 126-137, 2003.

[13] E. Perla, Art, R. S. Carbajo, M. Huggard, and C. M. Goldrick, PowerTOSSIM z: realistic energy modelling for wireless sensor network environments, in Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks, pp. 35-42, 2008.

[14] B. L. Titzer, D. K. Lee, and . Palsberg, Avrora: scalable sensor network simulation with precise timing, IPSN 2005. Fourth Int. Symp. Inf. Process. Sens. Networks, 2005.

[15] F. Osterlind, A. Dunkels, . Eriksson, N. Finne, and T. Voigt, Cross-Level Sensor Network Simulation with COOJA, Proceedings. 2006 31st IEEE Conf. Local Comput. Networks, 2006.