Evaluation of Wireless Sensor Network Simulators

Miloš Jevtić, Nikola Zogović, *Member, IEEE* and Goran Dimić, *Member, IEEE*

***Abstract* — This paper is inspired by the lack of detailed wireless sensor network (WSN) simulator surveys. We examine four WSN simulators: *ns-2*, *Castalia* (*OMNeT++* based), *TOSSIM,* and *COOJA/MSPSim*, and define a set of criteria to evaluate and compare the simulators. We provide short descriptions of simulators and tabular comparison based on the criteria. Since none of the simulators under survey is a universal solution, rough guidelines on which simulator to use in particular situation are given.**

***Keywords* — Wireless Sensor Networks, Simulation**

I. Introduction

WIRELESS sensor network (WSN) is composed of spatially distributed autonomous devices using sensors to cooperatively monitor some phenomenon. These devices, called sensor nodes or motes, are small-sized, low-cost, low-power, embedded systems built around a low-power microcontroller and equipped with one or more sensors, radio transceiver and a power source. WSN have a wide variety of applications such as battlefield surveillance, target tracking, industrial process monitoring, environment and habitat monitoring, precision agriculture, and disaster area monitoring.

Use of simulators is necessary when developing or researching in the field of WSN. Reasons for this are numerous. Manufacturers have not achieved expected low costs for sensor nodes yet, so experiments on a real-world WSN (which could consist of hundreds or thousands of nodes) are expensive [1]. It is more cost-effective to use simulation, except for final stages of development, when real-world tests are needed. Distributed nature and large number of sensor nodes make debugging a WSN very complicated task. Again, simulation can be used to detect and correct many bugs and issues before testing on a real system. Some applications of WSN require operation in very specific environments (e.g. volcano activity monitoring). Experimenting in those environments could be expensive or dangerous, which is another reason for using simulation [2]. Finally, simulation enables experiments under controlled conditions. For example, it is possible to create specific scenarios that are hard to carry out in reality or repeat the same scenario multiple times with different parameters of system under test.

Milos Jevtić, Nikola Zogović, Goran Dimić, Institute Mihajlo Pupin, Volgina 15, 11050 Belgrade, Serbia; (e-mail: milos.jevtic@institutpupin.rs, nikola.zogovic@institutpupin.rs, goran.dimic@institutepupin.com)

There are many WSN simulators currently available, and choosing the right one for a given application is very important. To make this choice easier, we conduct a survey of several simulators that we find significant and interesting. In this paper, we present the results of our survey. In the second section, we present our methodology and selected criteria. In the third section, we describe and compare simulators according to the methodology and criteria. In the fourth section, we further discuss our findings, and in the fifth section, we conclude the paper.

II. Methodology and Criteria

A survey similar to ours was conducted in [1], but some of the presented information is outdated. Survey presented in [2] encompassed large number of WSN simulators, without entering into details. In contrast to that, we make a more detailed survey of four simulators.

We define a set of criteria to evaluate and compare the selected simulators and entitle these criteria *evaluation criteria*. The criteria are:

1. *Level of details* – There are three types of simulators. The first type is a *generic simulator*, which focuses on high-level aspects of WSN, such as networking, sensing, and data processing while operating system (OS) and hardware architecture of sensor nodes are not considered. This type of simulators is useful for evaluation of high-level protocols and algorithms. The second type is a *code level simulator* that uses the same code in simulation as in real sensor node. Application code and OS code (device drivers have to be altered, because there are no real hardware devices) are compiled for the machine that is running the simulator [3] while hardware architecture of sensor nodes is not taken into account. These simulators can be used to find bugs that are not related to timing or hardware architecture [3]. The third type is a *firmware level simulator*, which uses hardware emulation to execute deployable application and OS code compiled for the target platform. Using this kind of simulators, most types of bugs can be found, and timing-sensitive software can be tested [3].

2. *Timing* – In *discrete event* simulators, events that affect state of the system are chronologically ordered into event queue, and event scheduler executes them one by one. *Continuous* simulators are concerned with modeling a set of equations that represent system over time [4].

3. *Software license* – A simulator can be proprietary, or have one of the free software/open source licenses.

4. *Popularity* – Number of hits on Google with

"Wireless Sensor Network <simulator name>" query measures popularity.

5. *Simulator platform* – OS that simulator runs on.

6. *WSN platforms* – Sensor nodes/platforms that can be simulated.

7. *Graphical User Interface (GUI) support* – Is it available and how useful it is?

8. *Available models and protocols* – They are examined separately for each important layer or segment of WSN: radio propagation, physical (PHY) layer, medium access control (MAC) layer, network layer, transport layer and sensing.

9. *Energy consumption model* – Is it available and what level of detail it has?

Sources of information for this survey are scientific papers, vendor web sites and available documentation.

III. SIMULATORS

Simulators for this survey were selected using two criteria. The first is availability of simulator free of charge for academic use. The second is whether a vendor actively develops and supports a simulator or not. Based on these criteria, we have selected four simulators: *ns-2*, *OMNeT++* based *Castalia*, *TOSSIM* and *COOJA/MSPSim*. The logic behind this choice is: *ns-2* is the most popular network simulator; *OMNeT++* has growing popularity and modular structure, which gives it potential to grow, define more details, and optimize; *TOSSIM* is part of TinyOS, the most widely used OS for WSN; *COOJA/MSPSim*, besides having some very interesting features, is part of Contiki OS, with growing popularity.

We compare selected simulators based on evaluation criteria defined in previous section. Results of the comparison are presented in Table 1. We first give short descriptions of simulators in the following subsections.

A. *ns-2*

ns-2 [5] is the most widely used WSN simulator [1]. It began as a general network simulator, and support for mobile ad-hoc wireless networks was added later [2]. It is a generic, discrete event simulator.

ns-2 is an object-oriented (OO) simulator, written in C++, with an OTcl interpreter as a front-end [5]. Simulation kernel, models, protocols and other components are implemented in C++, but are also accessible from OTcl. OTcl scripts are used for simulator configuration, setting up network topology, specifying scenarios, recording simulation results etc. Typical ns-2 OTcl script for wireless simulation begins with configuration command, which is used to specify PHY, MAC and routing protocol, radio propagation and antenna model, topology etc. The next step is creation of mobile nodes. Node movement and network traffic patterns are usually defined in separate files [5]. Tools for generating these files are provided.

Very simple energy consumption model is used. Each node starts with initial amount of energy. Amounts of energy spent for packet transmission and reception are also defined. After receiving or transmitting packet, node's energy is decremented for a corresponding amount. When node's energy reaches zero, node cannot send or receive packets any more [5].

Tool called *nam* enables graphical visualization of simulation flow. During simulation, *ns-2* generates special *nam* trace file. OTcl script is used to select what information should be recorded in this file. *nam* uses the data stored in trace file to visualize network topology and animate packet flow. A tape recorder style user interface is used to control the simulation replay.

Using *ns-2* as a WSN simulator has some drawbacks. First, sensing model does not exist. Second, packet formats and MAC protocols are different from those found on typical WSN platforms [1]. Third, energy model is too simple.

However, *ns-2* is extensible and several third party add-ons that address some of the mentioned drawbacks have been created. *Mannasim* [6] for example adds a sensing model, several application models, LEACH routing protocol, Mica2 PHY model, etc. A GUI tool that automatically creates OTcl scripts is also provided.

B. *OMNeT++ and Castalia*

Another popular discrete event simulator is *OMNeT++* [7]. It is not a WSN simulator, nor even a network simulator, but a rich simulation platform on which various independent groups can build their own simulators [7].

An *OMNeT++* based simulator is built from elements called *modules*. *Simple module* is a basic unit of execution and is written in C++. *Compound module* consists of other modules (simple or compound) that are linked by *connections*. Top-level compound module is called *network module*. Modules communicate via *messages* that are sent via connections or directly from module to module. Topology (i.e. structure of compound modules and network module) is defined using declarative language called *NED*. Scenarios and various simulation parameters are defined in INI files, and thus are separated from models and topology [7].

OMNeT++ includes an integrated development environment (IDE) that enables C++ programming and debugging of simple modules, as well as graphical and textual editing of *NED* files. *Tkenv* is a GUI tool for monitoring simulation flow, featuring animation of message flow on network charts, visualizing node state changes, displaying debug output of modules or module groups, viewing and manually changing state of simulation objects etc. Tools for visualizing dynamic interactions among modules and for results analysis and visualization are also provided [7].

Example of a WSN simulator built on top of *OMNeT++* is *Castalia* [8]. It is a generic simulator intended for the first order validation of high-level algorithms before moving to a specific sensor platform.

In *Castalia*, sensor nodes are implemented as compound modules, consisting of sub-modules that represent, for instance, network stack layers, application, and sensor. Node modules are connected to wireless channel and physical process modules [8].

TABLE 1: SIMULATOR COMPARISON

Simulator		***ns-2***	***Castalia (based on OMNeT++)***	***TOSSIM***	***COOJA/MSPSim***
Level of details		generic	generic	code level	all levels
Timing		discrete event	discrete event	discrete event	discrete event
Software License		GNU GPL	Academic Public License	BSD	BSD
Popularity		780000	11900	9810	3010
Simulator platform		FreeBSD, Linux, SunOS, Solaris, Windows (Cygwin)	Linux, Unix, Windows (Cygwin)	Linux, Windows (Cygwin)	Linux
WSN platforms		n/a	n/a	MicaZ	Tmote Sky, ESB/2
GUI support		monitoring of simulation flow	monitoring of simulation flow, C++ development, topology definition, result analysis and visualization	none	yes
Available models and protocols	wireless channel	free space, two-ray ground reflection, shadowing	lognormal shadowing, experimentally measured path loss map, packet reception rates map, temporal variation, unit disk	lognormal shadowing	multi-path ray-tracing with support for attenuating obstacles, unit disk
	PHY	Lucent WaveLan DSSS	CC1100, CC2420	CC2420	no data
	MAC	802.11 (several implementations), preamble based TDMA (still at a preliminary stage)	TMAC, SMAC, Tunable MAC (can approximate BMAC, LPL, etc)	standard TinyOS 2.0 CC2420 stack	X-MAC, LPP, NULLMAC
	network	DSDV, DSR, TORA, AODV	Simple Tree, Multi-path Rings	no data	no data
	transport	UDP, TCP	none	no data	no data
	sensing	random process with Mannasim add-on	generic moving time-varying physical process	no data	no data
Energy consumption model		yes	yes	with PowerTOSSIM z add-on	yes

PHY module models a generic low power radio. It supports multiple states with different power consumptions, multiple levels of transmission power, carrier sensing and modulations. PSK and FSK modulations are supported, while custom modulation can be modeled by defining SNR-BER curve [8].

There are two MAC modules available: first implements TMAC and SMAC, while second can approximate several protocols but supports only broadcast communication. PHY and MAC modules can be controlled from application module [8].

Sensed phenomenon is modeled with a generic physical process that can move and change its value while effects of diffusion are also taken into consideration. Sensor noise and bias are modeled, too. Energy model is very simple, comparable to one used in *ns-2*.

C. TOSSIM

A typical example of a code level WSN simulator is *TOSSIM* [9], a part of the standard TinyOS [10] distribution. *TOSSIM* enables simulation of entire TinyOS applications by replacing few low-level components with simulation implementations. It is a discrete event simulator, where simulation events represent hardware interrupts, high-level system events and posted tasks [9].

TinyOS application and *TOSSIM* specific software modules are compiled and linked into a software library. Python interpreter can be used with this library to define topology, configure and run simulation etc. Alternatively, C++ application linked to the library can be used instead of Python. Python approach is easier enabling dynamic interaction with simulation and inspection of variables in a running TinyOS program. C++ does not allow variable inspection, but is faster and thus better suited for high performance simulations [9].

Wireless channel model is based on defining propagation loss for each pair of nodes, in both directions. Loss values can be obtained from real-world measurements or by applying a theoretical model. A tool that calculates loss values for given topology using lognormal shadowing model is provided. *TOSSIM* does not provide a specific PHY model, but provides several low-level primitives that can express a wide range of radios and their behavior. By default, CC2420 PHY is simulated. RF noise and interference from other nodes and outside sources are also simulated. Closest Pattern Matching (CPM) algorithm is used to analyze noise trace and create a statistical model from it. Then, this model is used for noise and interference simulation [9].

TOSSIM has three shortcomings. First, all simulated nodes run the same application code. Second, *TOSSIM* does not model energy consumption, though there is an add-on *PowerTOSSIM z* [11] that corrects this problem. Third, there is a lack of decent documentation.

D. COOJA/MSPSim

COOJA [12] and *MSPSim* [13] are WSN simulators included in Contiki [14] distribution. *MSPSim* can be integrated into *COOJA*, forming *COOJA/MSPSim*.

MSPSim is a firmware level simulator for WSN platforms based on Texas Instruments MSP430 microcontroller. It combines cycle accurate interpretation of CPU instructions with discrete event simulation of all other components. Some components (such as A/D converter) require cycle accurate timing, while other components (such as radio transceiver) do not. Therefore, two event queues are utilized, one for events based on clock cycles, and the other one for events based on simulation time.

MSPSim has debugging capabilities such as break points, watches, logging, and single stepping. Statistics, e.g. how much time a component spent in different operating modes, are also provided, which can be useful when investigating power consumption.

COOJA is primarily a code level simulator for networks consisting of nodes running Contiki OS. Nodes with different simulated hardware and different on-board software may co-exist in the same simulation. Code level simulation is achieved by compiling Contiki core, user processes and special simulation glue drivers into object code native to the simulator platform, and then executing this object code from *COOJA*. Since *COOJA* is a Java application, all interaction with compiled Contiki code is done through Java Native Interface (JNI). Firmware level simulation can be achieved by compiling Contiki core and user processes into target platform object code that can be executed in *MSPSim*. COOJA is also able to simulate non-Contiki nodes, whose functionality is implemented in Java, and act as a generic WSN simulator.

COOJA/MSPSim can simulate sensor nodes at all three levels of details. In addition, nodes simulated at different levels of details can co-exist and interact in the same simulation. This feature is called cross-level simulation and is one of the highlights of *COOJA/MSPSim*.

IV. Discussion

Data presented in previous section enable us to compare simulators and give rough guidelines for their usage.

ns-2 and *Castalia* are generic WSN simulators and should be used for evaluation of high-level algorithms, protocols and applications before moving to a specific platform.

ns-2 is the most widely used WSN simulator. However, it relies on third party add-ons to provide WSN specific features such as MAC protocols and sensing models. *Mannasim* is the only such add-on that is currently available, and it is not clear whether it is still developed.

Castalia is built on top of *OMNeT++* from which it inherits hierarchical architecture, strong GUI and IDE support, clear separation of simulation kernel, models, topology and scenarios. *Castalia* also provides realistic models of wireless channel and PHY, and above-average sensing model. *Castalia* is in the state of active development, but it has not made larger impact so far.

TOSSIM is suitable for simulating nodes that run TinyOS.

COOJA/MSPSim is the best choice for development of Contiki based WSN. Theoretically, *COOJA/MSPSim* could be used in multiple development phases. In first phase, high-level concepts would be validated using only Java nodes. Then, Contiki code would be written and partly tested by using code level simulation. Finally, firmware level simulation would be used for thorough testing.

V. Conclusion

We examine four WSN simulators: *ns-2*, *Castalia* (*OMNeT++* based), *TOSSIM,* and *COOJA/MSPSim,* and define a set of criteria to evaluate and compare the simulators. Simulators are compared based on the criteria, and comparison results are presented in tabular form. In addition to that, short descriptions of simulators are provided. Since none of the simulators under survey is a universal solution, we give rough guidelines on which simulator to use in particular situation.

References

[1] D. Curren, "A survey of simulation in sensor networks", University of Binghamton project report for subject CS580.

[2] M. Mekni, B. Moulin, "A survey on sensor webs simulation tools", *Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications*, 2008, pp. 574-579.

[3] J. Eriksson, "Detailed simulation of heterogeneous wireless sensor networks," Licentiate thesis, Department of Information Technology, Uppsala University, May 2009.

[4] R. McHaney, *Computer simulation: a practical perspective*. San Diego, CA, USA: Academic Press Professional, 1991

[5] K. Fall and K. Varadhan, "The ns manual", User's manual, UC Berkeley, LBL, USC/ISI, and Xerox PARC, January 2009.

[6] Mannasim Home Page, http://www.mannasim.dcc.ufmg.br (accessed on June 2009)

[7] A. Varga, R. Hornig, "An overview of the OMNeT++ simulation environment", *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Marseille, France, 2008.

[8] A. Boulis, "Castalia, a simulator for wireless sensor networks and body area networks, version 2.2", User's manual, NICTA, August 2009.

[9] TOSSIM, From TinyOS Documentation Wiki, http://docs.tinyos.net/index.php/TOSSIM (accessed on Sept. 2009)

[10] P. Levis et al, "TinyOS: an operating system for sensor networks", in *Ambient Intelligence*, W. Weber, J. M. Rabaey and E. Aarts, Ed. Springer Berlin Heidelberg, 2005, pp. 115-148.

[11] E. Perla et al, "PowerTOSSIM z: realistic energy modelling for wireless sensor network environments," *Proceedings of the 3rd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, Vancouver, British Columbia, Canada, 2008, pp. 35-42.

[12] F. Österlind et al, "Cross-level sensor network simulation with COOJA," *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications*, Tampa, Florida, USA, 2006.

[13] J. Eriksson et al, "MSPSim - an extensible simulator for MSP430-equipped sensor boards," *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, 2007.

[14] A. Dunkels, B. Gronvall, T. Voigt, "Contiki - A lightweight and flexible operating system for tiny networked sensors," *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, 2004, pp. 455-462.