Simulation Issues in Wireless Sensor Networks: A Survey

Abdelrahman Abuarqoub, Fayez Al-Fayez, Tariq Alsboui, Mohammad Hammoudeh, Andrew Nisbet
School of Computing, Mathematics and Digital Technology
Manchester Metropolitan University
Manchester, UK
f.a.alfayez@gmail.com{A.Abuarqoub, M.Hammoudeh, T.Alsboui, A.Nisbet}@mmu.ac.uk

***Abstract*—This paper presents a survey of simulation tools and systems for wireless sensor networks. Wireless sensor network modelling and simulation methodologies are presented for each system alongside judgments concerning their relative ease of use and accuracy. Finally, we propose a mixed-mode simulation methodology that integrates a simulated environment with real wireless sensor network testbed hardware in order to improve both the accuracy and scalability of results when evaluating different prototype designs and systems.**

Keywords-Wireless Sensor Networks; Simulation tools; Survey; Testbeds; Mix-mode simulation.

I. INTRODUCTION

A successful large-scale Wireless Sensor Network (WSN) deployment necessitates that the design concepts are checked before they are optimised for a specific hardware platform. Developing, testing, and evaluating network protocols and supporting architectures and services for WSNs can be undertaken through test-beds or simulation. Whilst test-beds are extremely valuable, implementing such test-beds is not always viable because it is difficult to adapt a large number of nodes in order to study the different factors of concern. The substantial cost of deploying and maintaining large-scale WSNs and the time needed for setting up the network for experimental goals makes simulation invaluable in developing reliable and portable WSNs applications.

In WSNs, simulation provides a cost effective method of assessing the appropriateness of systems before deployment. It can, for example, help assess the scalability of algorithms free of the constraints of a hardware platform. Furthermore, simulators can be used to simplify the software development process for a particular WSN application. For instance, TOSSIM [1] utilises the component based architecture of TinyOS [2] and provides a hardware resource abstraction layer that enables the simulation of TinyOS applications which can then be ported directly to a hardware platform without further modifications.

Simulation is hence the research tool of choice for the majority of the mobile ad hoc network community. An examination of research papers published in SENSORCOMM 2011 [3] reveals a significant increase in using real testbeds compared to the study published by Kurkowski et al. [4]. Yet, 53% of the authors used simulation in their research. Apart from the self-developed simulators, there are a few widely used network simulators including NS-2 [5] , OPNET [6], MATLAB [7], IFAS [8], and OMNet++ [9]. Figure 1 shows the simulator usage following a survey of simulation based papers in SENSORCOMM 2011 conference. Simulation of ad hoc wireless capabilities for WSNs have been addressed by extending existing simulators, or specifically building new ones, such as NS-3 [10]. The latter class of simulators mostly focus on protocols and algorithms for layers of the network stack, but they do not directly support WSNs.

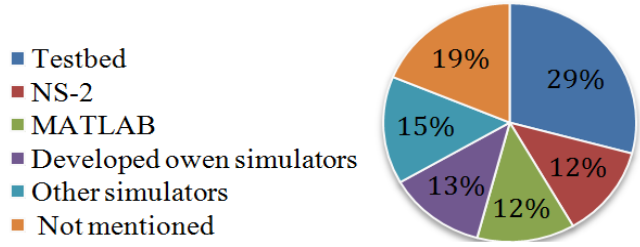

Figure 1. Simulator usage results from a survey of simulation based papers in SENSORCOMM 2011.

Recently, several simulation tools have appeared to specifically address WSNs, varying from extensions of existing tools to application specific simulators. Although these tools have some collective objectives, they obviously differ in design goals, architecture, and applications abstraction level. In the next section, we review some of the important WSNs simulation tools and explore their characteristics.

The rest of the paper is organised as follows: In Section II, the most popular WSNs simulators are outlined and their strengths and weaknesses are discussed. Section III, presents our views about the future of WSNs testing and evaluation methods. Section IV concludes the paper.

II. WSNS NETWORK SIMULATION TOOLS

A. *SensorSim*

SensorSim [11] builds on the NS-2 simulator providing additional capabilities for modelling WSNs. The main features of this platform are: power and communication protocol models; sensing channel and sensor models; scenario generation; and support for hybrid simulations.

 ISBN: 978-1-61208-207-3

The public release of the SensorSim suite of tools was withdrawn due to its unfinished nature and the inability of authors to provide the needed level of support.

Georgia Tech SensorSimII [12] is written in a modular style, where sensor nodes are organised into three components: application, network, and link. The work in SensorSimII may be divided into two areas: the simulator core and the visualisation tools. The simulator core essentially manages an array of independent sensor nodes throughout time. The visualisation tools provide views of both individual node state and communication traffic between nodes.

Both SensorSim projects are open source and free to use. However, the simulators are limited in their realism because (apart from SensorSim's power modules) neither simulator considers the limited resources of sensor nodes such as memory, and real-time computational capability. Moreover, it is not always required by the WSN to validate the functional correctness and/or, to provide performance guarantees. SensorSim simulates the complete WSN protocol stack, although this can be regarded as overkill and adding unnecessary complexity as this is not required in order to simulate the expected behaviour. This makes the SensorSim platform complex and difficult to use.

B. TOSSIM

There are platforms specifically designed to simulate WSNs, such as TOSSIM [1] which is a part of the TinyOS development efforts [2]. TOSSIM is a discrete-event simulator for TinyOS applications [13]. It aims to assist TinyOS application development and debugging by compiling applications into the TOSSIM framework, which runs on a PC instead of compiling them for a mote. Using the TOSSIM framework, programs can be directly targeted to motes without modification. This gives users a bigger margin to debug, test, and analyse algorithms in a controlled and repeatable environment. In TOSSIM, all nodes share the exact same code image, simulated at bit granularity, and assuming static node connectivity is known in advance. Therefore, TOSSIM is more of a TinyOS emulator than a general WSN simulator. It focuses on simulating TinyOS rather than simulating the real world. This has the advantage that the developed algorithms can be tested on a target platform. However, this may place some restrictions of the target platform on the simulation. TOSSIM is not always the right simulation solution; like any simulation, it makes several assumptions about the target hardware platform, focusing on making some behaviour accurate while simplifying others [1]. TOSSIM can be used as a tool for absolute evaluation of some causes of the behaviour observed in real-world network deployments.

C. TOSSF

TOSSF [14] is a simulation framework that compiles a TinyOS application into the SWAN [15] simulation framework. It can be viewed as an improvement over TOSSIM with a primary focus on scalability. It allows simulation of a heterogeneous collection of sensor nodes and a dynamic network topology. TOSSF suffers from potentially long test-debug cycles because it does not provide a scripting framework for experimentation. Although it enables development of custom environmental models, the absence of a scripting framework requires those models to be compiled into the simulation framework. Given that both of these simulators are tightly coupled with TinyOS, they may be unsuitable for early prototyping, or developing portable WSN applications.

D. GloMoSim

GloMoSim [16] is a scalable simulation environment for wireless and wired network systems. Its parallel discrete-event design distinguishes it from most other sensor network simulators. Though it is a general network simulator, GloMoSim currently supports protocols designed purely for wireless networks. GloMoSim is built using a layered approach similar to the seven layer network architecture of the OSI model. It uses standard APIs between different simulation layers to allow rapid integration of models developed at different layers, possibly by different users.

As in NS-2, GloMoSim uses an object-oriented approach, however for scalability purposes; each object is responsible for running one layer in the protocol stack of every node. This design strategy helps to divide the overhead management of a large-scale network. GloMoSim has been found to be effective for simulating IP networks, but it is not capable of simulating sensor networks accurately [17]. Moreover, GloMoSim does not support phenomena occurring outside of the simulation environment, all events must be gathered from neighbouring nodes in the network. Finally, GloMoSim stopped releasing updates in 2000 and released a commercial product called QualNet.

E. Qualnet

Qualnet is a commercial network simulator tool released by Scalable Network Technologies [18] that is derived from GloMoSim. Qualnet significantly extends the set of models and protocols supported by GloMoSim. It also provides a comprehensive set of advanced wireless modules and user-friendly tools for building scenarios and analysing simulation results. Qualnet is a discrete-event simulator, as such, it is event driven and time aware. It uses a layered architecture that is run by each node. When a protocol resides in a particular layer at one node, the packets are passed down crossing the remaining layers at the sending node, across the network, and then up to the protocol stack at the receiving node. Qualnet has a modular design and an intuitive GUI that make it easy to use to learn and modify.

F. OPNET

OPNET [19] is a further discrete event, object oriented, general purpose network simulator. The engine of OPNET is a finite state machine model in combination

 ISBN: 978-1-61208-207-3

with an analytical model. It uses a hierarchical model to define each characteristic of the system. The top hierarchy level contains the network model, where the topology is designed. The second level defines the data flow models. The third level is the process editor, which handles control flow models defined in the second level. Finally, a parameter editor is included to support the three higher levels. The hierarchical models result in event queues for a discrete event simulation engine and a set of entities that handle the events. Each entity represents a node which consists of a finite state machine which processes the events during simulation.

Unlike NS-2 and GloMoSim, OPNET supports modelling sensor-specific hardware, such as physical-link transceivers and antennas. It also enables users to define custom packet formats. An attractive feature of OPNET is its capability of recording a large set of user defined results. Furthermore, the GUI (Graphical User Interface), along with the considerable amount of documentation and study cases that come along with the license are another attractive feature of the simulator. This GUI interface can also be used to model, graph, and animate the resulting output. The network operator is provided with editors that are required to simplify the different levels of modelling. Though model parameters can be changed, the simulation accuracy is influenced because OPNET is not open source software. Similar to NS-2, the object-oriented design of OPNET causes scalability problems. It does not have a high number of protocols publicly available possibly because of source code licensing constraints. Finally, OPNET is only available in commercial form.

The second class of simulators are application-oriented simulators, including EmStar [20], SENS [21], J-Sim [22], Shawn [23], and Dingo [24].

G. EmStar

EmStar [20] is a component based, discrete-event framework that offers a range of run-time environments, from pure simulation, distributed deployment on iPAQs [25], to a hybrid simulation mode similar to SensorSim. Emstar supports the use of simulation in the early stages of design and development by providing a range of simulated sensor network components, including radios, which provide the same interfaces as actual components. It supports hybrid mode with some actual components and some simulated components, and full native mode with no simulated components. As in TOSSIM, EmStar uses the same source code that runs at each of these levels to run on actual sensors. Amongst other simulators, such as TOSSIM, EmStar provides an option to interface with actual hardware while running a simulation. EmStar is compatible with two different types of node hardware. It can be used to develop software for Mica2 motes [26] and it also offers support for developing software for iPAQ based microservers. The development cycle is the same for both hardware platforms. The next step in the development cycle following the simulation is data replay. In this model, EmStar uses data collected from actual sensors in order to run its simulation. Leading directly from this, Emstar uses the half-simulation methodology similar to SensorSim's, where the software is running on a host machine and interfacing with a real physical communication channels. The final step in the development cycle is deployment.

EmStar combines many of the features of other WSNs simulators. Its component based design allows for fair scalability. Moreover, each aspect of the network can be logically fine-tuned due to its development cycle design. Because it targets a particular platform, many protocols are already available to be used. At the deployment step in the development cycle, only the configuration files have to be designed. This potentially adds constraints on the user as they must either ensure that the hardware configuration being used matches the existing configuration file, or they must write their own files.

The main goal of Emstar is to reduce design complexity, enabling work to be shared and reused, and to simplify and accelerate the design of new sensor network applications. While not as efficient and fast as other frameworks like TOSSIM, Emstar provides a simple environmental model and network medium in which to design, develop and deploy heterogeneous sensor network applications. When used as a migration platform from code to real sensor environment, the environment model may be sufficient for most developers. Another drawback of Emstar is that the simulator supports only the code for the types of nodes that it is designed to work with.

H. SENS

SENS [21] is a customisable component-based simulator for WSN applications. It consists of interchangeable and extensible components for applications, network communication, and the physical environment. In SENS, each node is partitioned into four main components: application, simulates the software application of the sensor node; network, handles incoming and outgoing packets; physical, reads sensed information; and environment, network propagation characteristics. Multiple different component implementations offer varying degrees of realism. For example, users can choose between various application-specific environments with different signal propagation characteristics. As in TOSSIM, SENS source code can be ported directly into actual sensor nodes, enabling application portability. Moreover, it provides a power module for development of dependable applications.

SENS defines three network models that can be used. The first successfully forwards packets to all neighbours, the second delivers with a chance of loss based on a fixed probability, and the third considers the chance of collision at each node. The physical component includes the non-network hardware for the sensor such as the power, sensors, and actuators. At a lower level, the environment

 ISBN: 978-1-61208-207-3

component models the physical phenomena and the layout. The layout model includes different types of surfaces, each affecting radio and sound propagation in a different way.

SENS is less customisable than many other simulators, providing no chance to alter the MAC protocol, along with other low level network protocols. SENS uses one of the most sophisticated environmental models and implements the use of sensors well. However, the only measurable phenomenon is sound.

I. J-Sim

J-Sim [22] is a component-based discrete event simulator built in Java and modelled after NS-2. The design of this simulator aims at solving many of the shortcomings of comparable object-oriented simulators like NS-2. J-Sim uses the concept of components instead of the concept of having an object for each individual node. J-Sim uses three top level components: the target node which produces stimuli, the sensor node that reacts to the stimuli, and the sink node which is the ultimate destination for stimuli reporting. Each component is broken into parts and modelled differently within the simulator; this eases the use of different protocols in different simulation runs.

J-Sim claim has several advantages over NS-2 and other simulators. First its component based architecture scales better than the object oriented model used by NS-2 and other simulators. Second, J-Sim has an improved energy model and the ability to simulate the use of sensors for phenomena detection. Like SensorSim, there is support for using the simulation code for real hardware sensors. However, J-Sim is comparatively complicated to use. While no more complicated than NS-2, the latter simulator is more popular and accepted in the sensor network research community and more community support is available, therefore, more people are keen to spend the time to learn how to use it.

Though it is scalable, J-Sim has a set of inefficiencies. First, there is unnecessary overhead in the intercommunication model. The second problem is inherited by most sensor networks simulators that are built on top of general purpose simulators, 802.11 is the only MAC protocol that can be used in J-Sim. Finally, Java is possibly less efficient than many other languages.

J. Dingo

Dingo [27] provides a workbench for prototyping algorithms for WSNs taking a top-down design methodology. Having no target platform means the full functionality of a programming language can be used. This eases the design process as prototype algorithms can be tested before optimisation for the target platform. Dingo consists of a fixed API, with customisable internals. It has a simple graphical user interface and a set of base classes, which are extended by the user to create simulation. Each simulated sensor node runs in its own thread and communicates using the same protocols that would be deployed on a physical node. Sensors are modelled using a pool of concurrent, communicating threads. Individual sensors are able to: (1) Gather and process data from a model environment; (2) Locate and communicate with their nearest neighbours; (3) Determine whether they are operating correctly and act accordingly to alter the network topology in case of faulty nodes being detected. Nodes may be configured differently to simulate a heterogeneous sensor network. Dingo comes with a set of application level routing packages including simple multi-hop flooding, MuMHR [28] and LEACH [29].

Dingo features a significant improvement in the simulation performance by giving the option to split the visualisation from the simulation. It provides tools for the simulation and deployment of high-level, Python code on real sensor networks. For example, Dingo-boom provides a two-way interface between MoteIV's Boomerang class motes and Dingo. Dingo-top is another tool which is used to dump network topology data to a text file and generate a graphical representation of that topology. Furthermore, Dingo has several features in the form of plugins. These can be activated/deactivated on the plugin menu.

As with SensorSimII, Dingo provides an extensible visualisation framework that aims at easing the life for sensor network debugging, assessment, and understanding of the software by visualising the sensor network topology, the individual node state, and the transmission of the sensed data. Dingo comes with an interface between the simulation environment and different hardware platforms, for example the Gumstix [30] platform. Also, Dingo allows mixed-mode simulation using a combination of real and simulated nodes. In Dingo, nodes have the ability to obtain their sensed data from a database or graphical objects like maps; this improves the fidelity of simulations as it makes it possible to check the simulation results against the real data.

Dingo focuses on the protocols and algorithms for higher layers of network state but it does not directly support sensor networks at the physical layer. It has major drawbacks which limit its functionality. Most of these drawbacks are due to the incomplete nature of the tool. These drawbacks are: (1) The lack for Media Access Control or MAC layer, communications to be handled by point-to-point systems. (2) No collision management procedure, partly due to the absence of the MAC layer.

K. NS-3

NS-2 [31] is an object-oriented discrete event simulator targeted at networking research. It is an open source network simulator originally designed for wired, IP networks. The NS-2 simulation environment offered great flexibility in studying the characteristics of WSNs because it includes flexible extensions for WSNs. NS-2 has a number of limitations: (1) It puts some restrictions on the customisation of packet formats, energy models, MAC protocols, and the sensing hardware models, which

 ISBN: 978-1-61208-207-3

limits its flexibility; (2), the lack of an application model makes it ineffective in environments that require interaction between applications and the network protocols. (3) It does not run real hardware code; (4) It has been built by many developers and contains several inherent known and unknown bugs. (5) It does not scale well for WSNs due to its object-oriented design; (6) Using C++ code and oTcl scripts makes it difficult to use.

To overcome the above drawbacks the improved NS-3 simulator [10] was developed. NS-3 supports simulation and emulation. It is totally written in C++, while users can use python scripts to define simulations. Hence, transferring NS-2 implementation to NS-3 require manual intervention. Besides the scalability and performance improvements, simulation nodes have the ability to support multiple radio interfaces and multiple channels. Furthermore, NS-3 supports a real-time schedule that makes it possible to interact with a real systems [10]. For example, a real network device can emit and receive NS-3 generated packets.

L. Shawn

Shawn is an open source discrete event simulator for WSNs. It is written in C++ and can be run in Linux/Unix and Windows environments. Shawn aims to simulate large- scale WSNs, where physically accurate simulations fail. The idea behind Shawn is to use abstract models to simulate the affects of a phenomenon rather than the phenomenon itself [23]. Users of Shawn can adapt the simulation to their needs by selecting the application preferred behaviour. The authors claim that Shawn provides a high abstraction level that hides a lot of the simulation details. Users are given full access to the communication graph, which allows them to observe nodes and their data [23]. However, there are some limitation in Shawn, for instance: Visualization output is not supported, MAC module is not extent, and also users need to do much programming [32].

III. DISCUSSION

Generally, real WSNs testbeds provide a more accurate, realistic, and replicable validation mechanism for algorithms and protocols. However, the cost of deployment and maintenance of large-scale testbeds limits their applicability. Moreover, the wide variety of available sensor hardware can make it rather difficult to replicate any results produced by real testbeds. Besides, in some applications, where dangerous conditions are being studied, e.g. chemical pollution, a real testbed is an unwanted choice. Out of these restrictions came the need for simulation as a tool for validating and testing algorithms/protocols. As shown in Section II, simulation tools are widely available and used by WSNs researchers. However, most of the existing simulators are incomplete and follow different approaches to investigate different problems. The variety of existing simulation tools has led to accuracy and authenticity issues that concern even the best simulators available today. Such issues also make it even more difficult to replicate and compare evaluation results from competing simulation systems. Simulation drawbacks also include the lack of visualisation tools, GUI's, poor documentation, absence of examples, amongst others.

To solve the dilemma of having an accurate but scalable and low-cost prototyping solution, we suggest the use of mixed-mode simulation as an effective midrange solution. Mixed-mode simulation is the integration of a simulated environment and a real testbed to improve both the accuracy and scalability of testing results. In other words, the mixed-mode simulation enables the simulation of algorithms partially in software and partially in a real hardware WSN testbed. A small number of simulation tools like NS-3 and Dingo already support this mode of simulation. This simulation mode allows researchers to compare the results of running the same algorithm in both simulation and on physical sensor hardware; the comparison allows the inclusion or the modelling of more realistic conditions in the simulation environment. A flexible mixed-mode simulator should support integration of heterogeneous sensor devices. Also, the simulation-testbed interaction remains a challenging task that needs to be addressed. For instance, the authors of Dingo describe in [33] a new Python library that implements synchronous message-passing concurrency to improve coordination between many hosts.

Yet, the choice of a suitable simulator is a difficult decision. There is no 'best' simulator; each simulator has specific features that work well in certain circumstances. The selection of a simulator depends mostly on the algorithmic feature to be evaluated. High level simulators like NS-2 gives an estimation about the applications and some middleware behaviour. Mid-level simulators, e.g. OMNET, provides more information about the physical layer components that are simulated without giving too much details. Low-level simulators provide accurate bit level estimations of the hardware as well as software performance. Regardless of the simulator, any simulations will always have weaknesses either due to non-realistic assumptions or modelling errors that may be present in the algorithm itself. Therefore, developing formal methods, e.g. using graph theory [34], to verify the correctness of new algorithms and protocols is also part of the testing or evaluation research.

Table 1 summarise and compares the reviewed simulation tools.

 ISBN: 978-1-61208-207-3

TABLE 1. SUMMARY ABOUT REVIEWED SIMULATION TOOLS

Simulators	Programming Language	GUI	General or Specific Simulator	Open Source	Main Features	Limitations
SensorSim	C++	No	Specifically designed for WSNs	Yes	-Power and communication protocol models Sensing channel and sensor models -Scenario generation -Support for hybrid simulations	-Limited in SensorSim project realism -Consider limited resources of sensor nodes. -Simulates the complete WSN protocol stack
TOSSIM	C++	Yes	Specifically designed for WSNs	Yes	-Can be targeted to motes without modification -Nodes share the exact same code image -The developed algorithms can be tested on a target platform	-Makes several assumptions about the target hardware platform -Focusing on making some behaviour accurate while simplifying others
TOSSF	C++	Yes	Specifically designed for WSNs	Yes	-Primary focus on scalability -Support heterogeneous nodes and dynamic topology	-Long test-debug cycles
GloMoSim	C/Parsec	Yes	General	Yes	-Supports protocols designed purely for wireless networks. -Built using a layered approach. -Uses standard APIs between different simulation layers.	-Not scapable of simulating sensor networks accurately -does not support phenomena occurring outside of the simulation environmen
Qualnet	C/C++	Yes	General	Comm-ercial	-Comprehensive set of advanced wireless modules and user-friendly tools	- The annual license is expensive
OPNET	C/C++	Yes	General	Comm-ercial	-Uses a hierarchical model to define each characteristic of the system -Capability of recording a large set of user defined results	- scalability problems
EmStar	C	Yes	Specifically designed for WSNs	Yes	-Supports hybrid mode -Provides an option to interface with actual hardware while running a simulation -Compatible with two different types of node hardware	-Supports only the code for the types of nodes that it is designed to work with
SENS	C++	No	Specifically designed for WSNs	Yes	-Multiple different component implementations	-Less customisable -Only measurable phenomenon is sound
J-Sim	Java	Yes	Specifically designed for WSNs	Yes	-Ability to simulate the use of sensors for phenomena detection -Support for using the simulation code for real hardware sensors	-Comparatively complicated to use -Unnecessary overhead in the intercommunication model
Dingo	Python	Yes	Specifically designed for WSNs	Yes	-Full functionality of a programming language can be used -Option to split the visualisation from the simulation	-Does not directly support sensor networks at the physical layer -Incomplete nature of the tool
NS-3	C++	No	General	Yes	-Supports simulation and emulation -Supports a real-time schedule -Ability to support multiple radio interfaces and multiple channels	- Some restrictions on the customisation. -Lack of an application model -Does not run real hardware code -Does not scale well for WSNs
Shawn	C++	No	Specifically designed for WSNs	Yes	-Able to simulate large- scale WSNs -Ability of selecting the application preferred behaviour -Full access to the communication graph	-Does not support visualization output -MAC module is not extent -Lots of programing is required

 ISBN: 978-1-61208-207-3

IV. CONCLUSION

This paper provides a comprehensive review of simulation tools that are widely used in the field of WSNs. The aim is to help researchers choosing the most appropriate simulation tools to evaluate their work. There are a variety of simulation tools with different capabilities. However, the authors believe that they are insufficient for testing and evaluating WSNs algorithms. This is because the simulation results can be unrealistic due to the incomplete or inaccurate simulation models. An immediate measure is to develop unified models, e.g. energy, for different simulators. This allows realistic comparisons between results produced by different simulators to be made. To improve authenticity and accuracy of simulation results, it is important that researchers make their simulation code available for download by other researchers. Moreover, researchers should dedicate more space in their papers to clearly describe their simulation setup. On the other hand, large-scale real testbeds are still infeasible due to their cost and complexity. It can be easily observed that the trend in the WSNs field is to use mixed-mode simulation as an interim solution. Finally, we believe that theoretical validation of algorithms can serve as a good means for evaluating many WSNs algorithms.

REFERENCES

[1] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire TinyOS applications," SenSys '03: Proc. of the 1st int.conference on Embedded networked sensor systems, 2003, pp. 126-137.

[2] P. Levis, et al., "TinyOS: An Operating System for Sensor Networks," In Ambient Intelligence, 2005.

[3] IARIA, "The Fifth International Conference on Sensor Technologies and Applications, SENSORCOMM 2011," 2006-2011; http://www.iaria.org/conferences2011/SENSORCOMM11.html.

[4] S. Kurkowski, T. Camp, and M. Colagrosso, MANET Simulation Studies: The Current State and New Simulation Tools, The Colorado School of Mines, 2005.

[5] E. Larsen, et al., "iOLSR: OLSR for WSNs Using Dynamically Adaptive Intervals," The Fifth Int.l Conference on Sensor Technologies and Applications, 2011, pp. 18 to 23.

[6] K. Shi, Z. Deng, and X. Qin, "TinyMQ: A Content-based Publish/Subscribe Middleware for Wireless Sensor Networks," Proc. SENSORCOMM 2011, The Fifth Int. Conference on Sensor Tech. and Applications, 2011, pp. 12 to 17.

[7] R. Behnke, J. Salzmann, P. Gorski, and D. Timmermann, "HDLS: Improved Localization via Algorithm Fusion," The Fifth Int. Conference on Sensor Tech. and Applications, 2011.

[8] S. Feldman and M. Feldman, "Tree-Based Organization for Very Large Scale Sensor Networks," The Fifth Int. Conference on Sensor Technologies and Applications 2011, pp. 45 to 50.

[9] F. Derogarian, J. Ferreira, and V. Tavares, "A Routing Protocol for WSN Based on the Implementation of Source Routing for Minimum Cost Forwarding Method," Proc. SENSORCOMM 2011, The Fifth International Conference on Sensor Technologies and Applications, 2011, pp. 85 to 90.

[10] nsnam, "NS-3," 2011; from http://www.nsnam.org/.

[11] S. Park, A. Savvides, and M.B. Srivastava, "SensorSim: a simulation framework for sensor networks," MSWIM '00: Proceedings of the 3rd ACM int. workshop on Modeling, analysis and simulation of wireless and mobile systems, 2000.

[12] C. Ulmer, "Wireless Sensor Probe Networks-SensorSimII," 2007.

[13] xbow, "Mica Mote," 2007.

[14] L.F. Perrone and D.M. Nicol, "A Scalable Simulator For TinyOS Applications," Simulation Conference, 2002. Proceedings of the Winter, vol. 1, 2002, pp. 679 - 687.

[15] A. Gahng-Seop, T.C. Andrew, V. Andras, and S. Li-Hsiang, "Supporting Service Differentiation for Real-Time and Best-Effort Traffic in Stateless Wireless Ad Hoc Networks (SWAN)," IEEE Transactions on Mobile Computing, vol. 1, no. 3, 2002, pp. 192-207; DOI 10.1109/tmc.2002.1081755.

[16] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a library for parallel simulation of large-scale wireless networks," PADS '98: Proceedings of the twelfth workshop on Parallel and distributed simulation, 1998, pp. 154-161.

[17] D. Curren, "A survey of simulation in sensor networks," 2005.

[18] S.N. Technologies, "QualNet Simulator," from http://www.scalable-networks.com/products/qualnet/.

[19] X. Chang, "Network simulations with OPNET," WSC '99: Proceedings of the 31st conference on Winter simulation, 1999, pp. 307-314.

[20] L. Girod, et al., "Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks," ACM Trans. Sen. Netw., vol. 3, 2007, pp. 13.

[21] S. Sundresh, W. Kim, and G. Agha, "SENS: A Sensor, Environment and Network Simulator," The 37th Annual Simulation Symposium (ANSS37), 2004.

[22] A. Sobeih, et al., "J-Sim: A Simulation Environment for Wireless Sensor Networks," ANSS '05: Proceedings of the 38th annual Symposium on Simulation, 2005, pp. 175-187.

[23] S.P. Fekete, A. Kroller, S. Fischer, and D. Pfisterer, "Shawn: The fast, highly customizable sensor network simulator," Proc. Networked Sensing Systems, 2007. INSS '07. Fourth Int. Conference on, 2007, pp. 299-299.

[24] S. Mount, R.M. Newman, E. Gaura, and J. Kemp, "SenSor: an Algorithmic Simulator for Wireless Sensor Networks," In Proceedings of Eurosensors 20, vol. II, 2006, pp. 400-411.

[25] hp, "iPAQs," 2000.

[26] xbow, "Mica Mote," 2012.

[27] S. Mount, "Dingo Wireless Sensor Networks Simulator," 2008.

[28] H. Mohammad, K. Alexander, and G. Elena, "MuMHR: Multi-path, Multi-hop Hierarchical Routing," SENSORCOMM '07: Proceedings of the 2007 International Conference on Sensor Technologies and Applications, 2007, pp. 140-145.

[29] W. Heinzelman, et al., "Energy-Efficient Communication Protocol for Wireless Microsensor Networks," Proceedings of the 33rd Int. Conference on System Sciences, 2000.

[30] Gumstix.com, "Gumstix way small computing," 2007.

[31] NS-2, "The Network Simulator," 2007.

[32] E. Kolega, V. Vescoukis, and D. Voutos, "Assessment of network simulators for real world WSNs in forest environments," Proc. Networking, Sensing and Control (ICNSC), 2011 IEEE Int. Conference on, pp. 427-432.

[33] M. Hammoudeh, "Modelling Clustering of Sensor Networks with Synchronised Hyperedge Replacement," ICGT '08: Proceedings of the 4th international conference on Graph Transformations, 2008, pp. 490-492.

[34] S. Mount, M. Hammoudeh, S. Wilson, and R. Newman, "CSP as a Domain-Specific Language Embedded in Python and Jython," Proc. Comm. Process Architectures IOS Press, 2009.

 ISBN: 978-1-61208-207-3