Simulation Tools for Wireless Sensor Networks

E. Egea-López, J. Vales-Alonso, A. S. Martínez-Sala, P. Pavón-Mariño, J. García-Haro*
Department of Information Technologies and Communications
Polytechnic University of Cartagena, Spain.
Email: {esteban.egea, javier.vales, alejandros.martinez, pablo.pavon, joang.haro}@upct.es
* Corresponding author. Address: Campus Muralla del Mar, 30202, Cartagena, Spain.
phone: +34 968 325314, fax: +34 968 325973, e-mail: joang.haro@upct.es

Keywords
WSN, survey, scalability.

Abstract

Wireless Sensor Networks (WSN) are formed by a large number of networked sensing nodes. It is rather complex, or even unfeasible, to model analytically a WSN and it usually leads to oversimplified analysis with limited confidence. Besides, deploying test-beds supposes a huge effort. Therefore, simulation is essential to study WSN. However, it requires a suitable model based on solid assumptions and an appropriate framework to ease implementation. In addition, simulation results rely on the particular scenario under study (environment), hardware and physical layer assumptions, which are not usually accurate enough to capture the real behavior of a WSN, thus, jeopardizing the credibility of results. However, detailed models yields to scalability and performance issues, due to the large number of nodes, that depending on application, have to be simulated. Therefore, the tradeoff between scalability and accuracy becomes a major issue when simulating WSN.

In this survey a suitable model for WSN simulation is introduced, together with guidelines for selecting an appropriate framework. In addition, a comparative description of available tools is provided.

1. INTRODUCTION

Wireless Sensor Networks (WSN) can be considered a particular type of Mobile *Ad-hoc* NETwork (MANET), formed by hundreds or thousands of sensing devices communicating by means of wireless transmission. Research on WSNs and MANETs share similar technical problems. But in WSNs, two specific factors arise:

- The envisioned applications and the operation of the protocol layers are usually driven by the physical variables measured by the sensors. Therefore, the dynamics of the physical parameters sensed by the network govern the network traffic, and even the topology.
- The energy is a primary concern in WSN. Usually, nodes run on non-rechargeable batteries. Therefore, the expected node lifetime is a fundamental element, that must be taken into account. On the contrary, in MANETs, energy is an important issue that should be optimized, although it is generally assumed that a node can recharge or replace its battery.

These constraints make unfeasible to analytically model a WSN and predict the actual performance of high-level protocols and network operation, which often leads to oversimplified analysis with limited confidence. Currently, the first real WSN applications are being explored and some of them are yet to come. Meanwhile, deploying and operating a test-bed to study the actual behavior of protocols and network performance supposes a great effort [1], [2].

Consequently, simulation is essential to study WSN, being the common way to test new applications and protocols in the field. This fact has brought a recent *boom* of simulation tools available to model WSN. However, obtaining reliable conclusions from research based on simulation is not a trivial task. There are two key aspects that should be evaluated before conducting experiments: (1) The correctness of the model and (2) the suitability of a particular tool to implement the model.

On one hand, there exists an increasing concern about the methodology and assumptions of simulations [3], [4]: Idealized hardware, protocols and non-realistic radio models can lead to mistaken results. A "good" model based on solid assumptions is mandatory to derive trustful results. But, including the required degree of detail adds strong computational requirements. The large number of nodes that may be involved in a WSN further stress the problem. The fundamental tradeoff is: Accuracy and necessity of detail versus performance and scalability.

On the other hand, implementing a complete model requires a considerable effort. A tool that helps to build a model is needed, and the user faces the task of selecting the appropriate one. Simulation software commonly provides a framework to model and reproduce the behavior of real systems. However, actual implementation and "secondary goals" of each tool differ considerably, that is, some may be designed to achieve good performance and others to provide a simple and friendly graphical interface or emulation[1] capabilities.

The aim of this paper is to provide some insight on the building blocks of a general simulation model for WSN, introducing its specific issues. Also, to facilitate newcomers the selection of the most appropriate tool for their needs, the most extended WSN simulation environments are reviewed.

The contents of this survey are organized as follows. Section 2 introduces a wide vision of a WSN simulator architecture. Section 3 compares the main parameters to select a simulation framework. Next, section 4 describes the

[1]By emulation, we mean the ability to combine simulated and real systems.

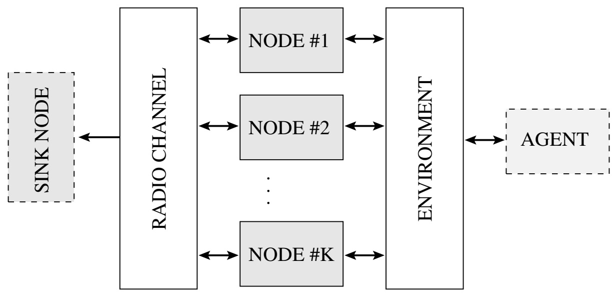

Fig. 1. Wireless sensor network model

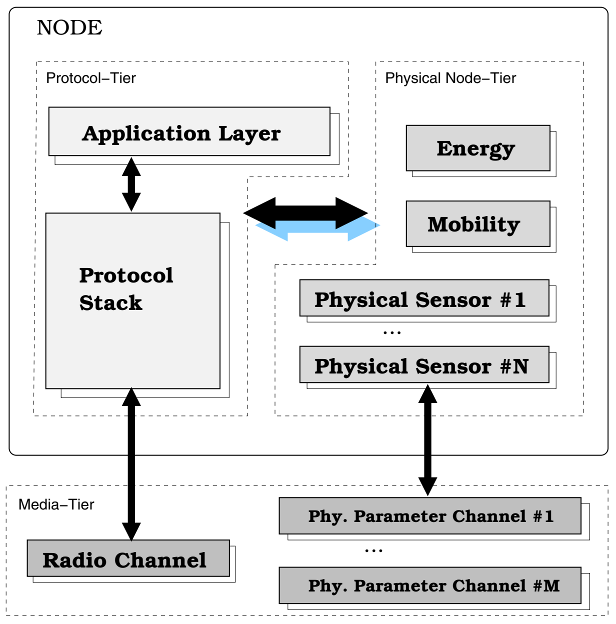

Fig. 2. Tier-based node model

capabilities and particular features of general and specific simulation packages, currently used to simulate WSN. Finally, section 5 summarizes the fundamental ideas of this paper.

2. A MODEL FOR WSN SIMULATION

Together with the development of simulation tools for WSN, their corresponding models have been introduced. The models include new components, not present in classical network simulators, as detailed power and energy consumption models or environment models. This section describes a general component model, derived from [5], [6], for WSN simulation tools. This model is suitable for most of the evaluation tools employed in on-going research on WSN.

2.1. Network model

Figure 1 depicts the general model at a network-wide scale. The following components are considered:

1) Nodes: Each node is a physical device monitoring a set of physical variables. Nodes communicate with each other via a common radio channel. Internally, a protocol stack controls communications. Unlike classical network models, sensor modes include a second group of components: The physical node tier, which is connected to the environment. Nodes are usually positioned in a two or three dimensional world. An additional "topology" component, not showed in figure 1 may control node coordinates. Depending on the application and deployment scenario, a WSN can contain from a few to several thousands of nodes. Section 2,2 describes the structure of a node.
2) Environment: The main difference between classical and WSN models is the additional "environment" component. This component models the generation and propagation of events that are sensed by the nodes, and trigger sensor actions, i.e. communication among nodes in the network. The events of interest are generally a physical magnitude as sound or seismic waves or temperature.
3) Radio channel: It characterizes the propagation of radio signals among the nodes in the network. Very detailed models use a "terrain" component, connected to the environment and radio channel components. The terrain component is taken into consideration to compute the propagation as part of the radio channel, and also influences the physical magnitude.
4) Sink nodes: These are special nodes that, if present, receive data from the net, and process it. They may interrogate sensors about an event of interest. The use of sinks depends on the application and the tests performed by the simulator.
5) Agents: A generator of events of interest for the nodes. The agent may cause a variation in a physical magnitude, which propagates through the environment and stimulates the sensor. This component is useful when its behavior can be implemented independently from the environment, e.g., a mobile vehicle. Otherwise, the environment itself can generate events.

2.2. Node model

Node behavior depends on interacting factors that cause cross-layer interdependencies. A convenient way to describe it is to divide a node into abstract tiers, as represented in Figure 2 .

- The *Protocol-tier* comprises all the communication protocols. Typically, three layers coexist at this tier: A MAC layer, a routing layer and a specific application layer. Note that the operation of the protocol tier usually depends on the state of the physical tier described below, e.g. a routing layer can consider battery constraints to decide on packet route. Hence, an efficient method to interchange tier information must be developed.
- The *physical-node* tier represents the hardware platform and its effects on the performance of the equipment. Actual composition of this tier may change depending on the specific application. The common elements of this tier are the set of physical sensors, the energy module and the mobility module. Physical Sensors describe the behavior of the monitoring hardware. Energy module simulates power consumption in the component hardware, a critical issue in WSN evaluation. Mobility module controls sensor position.

- The *media-tier* is the link of the node with the "real world". A node is connected with the environment through: (1) A radio channel, and (2) through one or more physical channels. Physical channels receive environmental events as described in section 2.1 .

3. FRAMEWORK SELECTION

Widespread research on WSN have raised a race involving many simulation tools and frameworks. The selection of a simulation framework for any type of network is a task that is worth to spend enough time. Indeed, this is particularly true for wireless sensors nets, because of the diversity and complexity of the simulation scenarios, protocols, and elements involved. In such an heterogeneous scope, different evaluation tools achieve different goals. This section identifies and discusses the main features to be considered in the selection of a WSN simulation framework. A comparative description of representative simulators follows in section 4 .

In a first step, existing WNS frameworks can be categorized in: (a) Specific add-ons to general purpose communication networks (section 4.1) and (b) WSN frameworks built from scratch (section 4.2).

3.1. The long-way road to simulation

Simulation design starts with a suitable description of the real system. Such description constitutes the *simulation model*, built up with the aid of common-simulation concepts like entities, attributes, events, channels, etc. Therefore, the modeler declares the structure of the simulation in terms of entities and their relations and implements the behavior of those entities and their response to events. Common simulation packages clearly separate implementation from model description and instantiation:

- The simulation engine and the basic model objects are provided as a set of software libraries in a high level programming language, usually Java or C++. This is the simulation API.
- Some kind of scripting (Tcl, e.g.) or mark-up language (XML, e.g.) is normally employed to support model description, that is, to establish (declare) relations between entities. Scripts allow a uniform and efficient approach to model description and configuration, model instantiation of simulation runs and runtime inspection.
- In addition, other utility libraries are often included such as graphical representation support or statistical data gathering and analysis.

Therefore, a simulation framework usually consists of a basic simulation library, a utility library, and some scripting support. The actual form the package is deployed depends on the implementation. Some packages provide tools that translate model scripts into objects in the implementation language to be compiled afterwards. Other packages bind library and scripting so that simulation objects can be instantiated from a script. Others provide a visual interface.

3.2. What may we expect from a good WSN simulator?

Usually, the key properties to select suitable simulation environment are:

1) Reusability and availability.
2) Performance and scalability.
3) Support for rich-semantics scripting languages to define experiments and process results.
4) Graphical, debug and trace support.

In this section, we focus on the impact of each feature in the context of the WSN.

Reusability and availability. Simulation is used to test novel techniques in realistic and controlled scenarios. Researchers are usually interested in comparing the performance of a new technique against existing proposals. Therefore, two key aspects are: Does the simulation tool include implementations of common models? How easy is to *modify* or *integrate* a new model with the existing ones?

The first question mainly depends on how long a framework has been used for, and how many people use it. Early and widely adopted frameworks have many available models and it is very likely that the new successful proposals will be added to next releases. The second aspect is closely related to the design of the package. A careful structure with clean interfaces and high modularity allows the user to easily add or change functionality. Ready-to-use models allow users to quickly build a realistic simulation scenario and focus on modeling more specific details of WSN. All the general-purpose packages include a more or less complete TCP/IP suite, which can be considered the minimum standard support. In addition, typical requirements for WSN simulators are: *Ad-hoc* routing support plus wireless MAC protocols, and propagation and mobility models to synthesize the physical node distribution. For example, these entities are commonly implemented: The AODV [7] for routing, the IEEE 802.11 [8] wireless MAC protocol, a path loss model [9] for propagation, and the random-waypoint-based mobility.

For specific tools the question is subtlety different: All the specific frameworks are able to execute *native* sensor code. Hence, every application, protocol or component developed for the actual sensor platform can be simulated or emulated. Only some specific parts are purely simulated, e.g. the radio channel or the physical media environment. Summing up, in this case protocols availability depends on the real availability of them for the target platform, and *viceversa*.

Performance and scalability. Performance and scalability is a major concern when facing WSN simulation. The former is usually bounded to the programming language effectiveness. The latter is constrained to the memory, processor and logs storage size requirements.

Additionally, the type of simulation implies some limits: Emulation mode and time-driven simulations operate in real time so they cannot be arbitrarily long.

Wireless simulations stress performance and scalability issues due to the increased complexity added by the interaction with the environment, radio propagation, mobility and power consumption. Simulation of several hundred of thousands of nodes remains a challenging problem.

Support for rich-semantics scripting languages to define experiments and process results. The configuration of a WSN typical trial requires to answer (at least) questions like: How many nodes are there in the test?, where is each node placed?, do nodes move?, all of them?, how?, which energy model is used?, how many physical environments are?, how they generate events?, which physical magnitudes should measure each node?, which statistics must be measured in the experiment?, which are the parameters of the radio model? The vast amount of variables involved in the definition of a WSN experiment requires the use of specific *input scripting* languages, with high-level semantics. Additionally, it is likely that large quantities of output data will also be generated through many replicas of the experiments. Therefore, a suitable *output scripting* language, that helps to obtain the results from the experiments quickly and precisely is desirable.

Graphical, debug and trace support. Graphical support for simulations is interesting in three aspects: (1) As a debugging aid. The primary and more practical way to quickly detect a bad behavior is to “watch” and follow the execution of a simulation. The key features that a graphical interface should support are: Capability of inspection of modules, variables and event queues at real time, together with “step-by-step” and “run-until” execution possibilities. These features make graphical interfaces a very powerful debugging tools. Note that the key is the ability to interact with the simulation. (2) As a visual modeling and composition tool. This feature usually facilitates and speeds the design of small experiments or the composition of basic modules. However, for large scale simulations, it is not very practical. (3) Finally, as result plotters, which allows quick visualization of results without a post-processing application.

4. WSN SIMULATION SOFTWARE

In this section the most relevant simulation environments used to study WSN are introduced, and their main features and implementation issues described and discussed. We basically focus on free, open-source, simulation tools.

4.1. General simulation packages

- NS-2 [10]. Discrete event simulator developed in C++. NS-2 is one of the most popular non-specific network simulators, and supports a wide range of protocols in all layers. It uses OTcl [11] as configuration and script interface. NS-2 is the paradigm of reusability. It provides the most complete support of communication protocol models, among non-commercial packages. Regarding WSN, NS-2 includes ad-hoc and WSN specific protocols such as directed diffusion [12] or SMAC [13]. Also, several projects intend to provide WSN support to NS-2 such as SensorSim [5] and NRL [14]. Both are extensions of NS-2 to support WSN modeling. However, SensorSim seems to be no longer available at [15]. NS-2 can comfortably model wired network topologies up to 1,000 nodes or above with some optimizations. This experiment size can be kept for wireless topologies using some new optimizations [16]. A disadvantage of NS-2 is that it provides poor graphical support, via Nam. This application just reproduces a NS-2 trace.
  NS-2 has been an essential testing tool for network research and, so, one could expect that the new *conventional* protocols will be added to future releases. However, new proposals for WSN are increasingly being tested in specific tools, e.g. TOSSIM or EmTOS (see section 4.2 for a description of both), because of the advantage of native sensor code simulation and the specific design of these tools for WSN. Therefore, it is unclear the availability of new WSN proposals for next releases of NS-2. This problem may be even worse for less used frameworks.
- OMNET++ [17]. Modular discrete event simulator implemented in C++. Getting started with it is quite simple, due to its clean design. OMNET++ also provides a powerful GUI library for animation and tracing and debugging support. Its major drawback is the lack of available protocols in its library, compared to other simulators. However, OMNET++ is becoming a popular tool and its lack of models is being cut down by recent contributions. For instance, a mobility framework has recently been released for OMNET++ [18], and it can be used as a starting point for WSN modeling. Additionally, several new proposals for localization and MAC protocols for WSN have been developed with OMNET++, under the Consensus project [19], and the software is publicly available. Nevertheless, most of the available models have been developed by independent research groups and do not share a common interface, what makes difficult to combine them. As an example, not even the localization and MAC protocols developed in the Consensus project are compatible.
- J-Sim [20]. A component-based simulation environment developed entirely in Java. It provides real-time process-based simulation. The main benefit of J-sim is its considerable list of supported protocols, including a WSN simulation framework with a very detailed model of WSNs, and a implementation of localization, routing and data diffusion WSN algorithms [6]. J-sim models are easily reusable and interchangeable offering the maximum flexibility. Additionally, it provides a GUI library for animation, tracing and debugging support and a script interface, named Jacl [21].
  J-Sim claims to scale to a similar number of wireless nodes than NS-2 (around 500) with two orders of magnitude better memory consumption but a 41% worse execution time [6].
- NCTUns2.0 [22]. Discrete event simulator whose engine is embedded in the kernel of a UNIX machine. The actual network layer packets are tunnelled through virtual interfaces that simulate lower layers and physical devices. This notable feature allows simulations to be fed with real program data sources. A useful GUI is available in

addition to a high number of protocols and network devices, including wireless LAN. Unfortunately, no specific designs for WSN are included.
On one hand, the close relationship between the simulation engine of NCTUns2.0 and the Linux kernel machine seems a difficulty (adding WSN simulation modules to this architecture is not a straightforward task). But, on the other hand, real sensor data can be easily plug into simulated devices, protocols and actual applications, just by installing these sensors in the machine.
NCTUns2.0 also has worthy graphical edition capabilities.

- JiST/SWANS [23]. Discrete event simulation framework that embeds the simulation engine in the Java bytecode. Models are implemented in Java and compiled. Then, bytecodes are rewritten to introduce simulation semantics. Afterwards, they are executed on a standard JVM. This implementation allows the use of unmodified existing Java software in the simulation, as occurs with NCTUns2.0 and UNIX programs. The main drawback of JiST tool, is the lack of enough protocol models. At the moment it only provides an *ad-hoc* network simulator called SWANS, built atop JiST engine, and with a reduced protocol support. The only graphical aid is an event logger. Jython [24] is used as a scripting engine.
JiST claims to scale to networks of 10^6 wireless nodes with two and one order of magnitude better performance (execution time) than NS-2 and GloMoSim respectively [23]. It has been also shown that it outperforms GloMoSim and NS-2 in event throughput and memory consumption, despite being built with Java.
- GloMoSim [25]. Simulation environment for wireless networks built with Parsec. Parsec [26] is a simulation language derived from C, that adds semantics for creating simulation entities and message communication on a variety of parallel architectures. Taking advantage of parallelization, it has been shown to scale to 10,000 nodes [27]. Several proposals for WSN protocols have been tested with it. Recently, a development kit for WSN has been released, sQualnet [28].
- SSFNet [29]. Set of Java network models built over the Scalable Simulation Framework (SSF). SSF is a specification of a common API for simulation, that assures portability between compliant simulators. There are multiple Java and C++ implementations of SSF. DartmouthSSF (DaSSF) [30], for instance, is a C++ implementation of SSF oriented to (parallel) simulation of very large scale communication networks.
Besides, specific extensions oriented towards *ad-hoc* networking exists, e.g., SWAN[2]. SWAN is being extended to be able to execute TinyOS code (see section 4.2), in a new framework called TOSSF [31].
- Ptolemy II [32]. Java packages that support different models of simulation paradigms (e.g. continuos time, dataflow, discrete-event). It also addresses the modeling, simulation and design of concurrent, real-time, embedded systems. Ptolemy models are constructed in an actor-oriented way, very similar to the component-based design of J-Sim. VisualSense [33] is a modeling and simulation framework for WSN built on Ptolemy II. Models can be developed by subclassing base classes of the framework or by combining existing Ptolemy models. Ptolemy visual edition assures a simple and intuitive graphical composition of models and result plotting.

[2]Notice that JiST/SWANS and SWAN are not related.

4.2. Specific WSN frameworks

This section describes the most relevant tools specifically aimed to emulate and simulate the WSN hardware and software (unlike the WSN extensions of the general network simulators described in the previous section). WSN scenarios are usually highly application-dependent, and subjected to hard constraints which cause, in turn, a tight coupling between layers. Therefore, dedicated tools may help to better capture these dependencies.

This approach also allows to simulate “real” application code, speeding up the migration from simulation to implementation, and facilitates testing and debugging of real applications. Emulation makes possible real time debug and analysis of information. The only drawback is that the user is tied to a single platform either software or hardware (typically MICA Motes [34]), and to a single programming language (typically TinyOS/NesC [35]). However, TinyOS and MICA motes are becoming the *de facto* platform for WSN, assuring somehow the “utility” of those tools.

Following environments are specifically designed for WSN research:

- TOSSIM [36]. Bit-level discrete event simulator and emulator of TinyOS, i.e. for each transmitted or received bit a event is generated instead of one per packet. This is possible because of the reduced data rate (around 40 kbps) of the wireless interface. TOSSIM simulates the execution of nesC code on a TinyOS/MICA, allowing emulation of actual hardware by mapping hardware interruptions to discrete events. A simulated radio model is also provided. Emulated hardware components are compiled together with real TinyOS components using the nesC compiler. Thus, an executable with real TinyOS applications over a simulated physical layer is obtained. Additionally, there are also several communication services that provide a way to feed data from external sources. The result is a high fidelity simulator and emulator of a network of TinyOS/MICA nodes. The goal of TOSSIM is to study the behavior of TinyOS and its applications rather than performance metrics of some new protocol. Hence, it has some limitations, for instance, it does not capture energy consumption. Another drawback of this framework is that every node must run the same code. Therefore, TOSSIM cannot be used to evaluate some types of heterogenous applications.
TOSSIM can handle simulations around a thousand of Motes. It is limited by its bit-level granularity: Performance degrades as traffic increases. Channel sampling is also simulated at bit level and consequently the use of a

CSMA protocol causes more overhead than would do a TDMA one.

- EmStar/EmSim/EmTOS [37] [38]. EmStar is a software framework to develop WSN applications on special platforms called *microservers*: Ad-hoc systems with better hardware than a conventional sensor. The EmStar environment contains a Linux microkernel extension, libraries, services and tools. The most important tools are:
  - EmSim: A simulator of the microservers environment. In EmSim every simulated node runs an EmStar stack, and is connected through a simulated radio channel model. It is not a discrete event but a time-driven simulator, that is, there is no virtual clock.
  - EmCee: An interface to real low-power radios, instead of a simulated radio model, obtaining radio emulation. EmStar source code (note that this code can be in any language) is used in the simulations.

  Additionally, the UCLA staff have developed EmTOS: An extension of EmStar that enables nesC/TinyOS applications to run in an EmStar framework. Thus, it opens the way to heterogenous systems of sensor and microservers. Simulation of microserver and sensor networks is also supported. In addition, EmTOS provides three modes of emulation: Pure emulation, where all the motes are emulated by software, "real mode", where all the motes are real, and "hybrid mode", where some motes are real and others are emulated. EmTOS reaches up to 200 modes and it is claimed that for over 500 nodes it would be necessary to distribute the simulation on several processors.
- ATEMU [39]. An emulator of the AVR processor (this processor is used in the MICA platform). While the operation of the mote is emulated instruction by instruction, the radio model is simulated. ATEMU also provides a library of other hardware devices, e.g., timers or transceivers. Therefore, a complete hardware platform is emulated, obtaining two advantages: (1) The capability of testing OS and applications other than TinyOS and (2) the capability of simulating heterogeneous networks with different sensors. They are achieved at the cost of high processing requirements and poor scalability.
- SENS [40]. A discrete event simulator implemented in C++. SENS utilizes a simplified sensor model with three layers (application, network and physical) plus an additional combined environment and radio layer. NesC code can be used directly on it.
- Prowler/JProwler [41]. A discrete event simulator running under MATLAB intended to optimize network parameters. JProwler is a version of Prowler developed in Java.
- SNAP [42]. A totally different approach. SNAP is defined as an integrated hardware simulation-and-deployment platform. It is a microprocessor that can be used in two ways: (1) As the core of a deployed sensor or (2) as part of an array of processors that performs parallel simulation. Again, "real" code for sensors can be simulated. By combining arrays of SNAPs (called Network on a Chip), it is claimed to be able to simulate networks on the order of 100,000 nodes.

5. CONCLUSIONS AND OPEN ISSUES

Simulation is an essential tool to study Wireless Sensor Networks due to the unfeasibility of analysis and the difficulties of setting up real experiments. This survey provides guidelines to help selecting a suitable simulation model for a WSN and a comprehensive description of the most used available tools.

Regarding availability of models, OMNET++, JiST and SSFNet lack of available protocol models compared to other simulators (specially, NS-2), which increases development time. Attending to the ability to compose models from basic pieces, the component or actor based packages J-Sim or Ptolemy II offer the maximum flexibility. Tools like NCTUns2.0 or JiST allow any, Linux or Java respectively, application to be used in a simulation. This feature greatly increases their possibilities. Specific tools such as TOSSIM, EMTOS or ATEMU are able to simulate real sensor code.

Regarding performance, one can expect better performance from C/C++ engines than from their Java counterparts. However, recent simulators like JiST/SWAN claim to perform better than NS-2 and GloMoSim (in its sequential version). Obviously, parallel simulations should perform and scale better than sequential ones. The tradeoff is a greater complexity of programming. Parallel simulators as GloMoSim (whose goal is performance rather than scalability) can simulate up to around 10,000 wireless nodes. DaSSF parallel tool, whose main goal is scalability, supports network topologies as large as 100,000 wired elements [43].

All the packages provide graphical support. OMNET++, NCTUns2.0, J-Sim and Ptolemy provide powerful GUI libraries for animation, tracing and debugging. All they include the aforementioned features such as inspection, modification of parameters at execution time, etc. OMNET++ and Ptolemy stand lightly up among them. On the contrary, JiST do not include other graphical interface than an event logger and viewer. Current support in NS-2 is the unelaborate and simple trace reproduction Nam tool. Specific tools also provide surprisingly rich GUIs. TinyViz is the TOSSIM visualization tool, an extensible Java application that provides useful debug information. Besides, it can control and drive the simulation elements. Users can develop their own plugins, which listen for TOSSIM events published by TinyViz and perform some action. EmView is a very similar tool, in this case written in C, for EmTos.

Additionally, Ptolemy-II and NCTUns2 provide graphical editors very simple to use. The graphical editors of the rest of packages are not that simple, so it is preferable most of the times to use their script-oriented way to create models.

We must also point out that there is a clear trend to use native code from actual devices (e.g., TinyOS/NesC) directly in simulations. *All* specific WSN frameworks have this capability.

As a final remark, credibility concerns about assumptions (mainly about radio channel) have been inherited from MANETs. Such concerns lead to complex models. Use of these detailed models may solve these accuracy issues. However, unlike MANETs, such solution limits the scalability of WSN experiments. New algorithms may alleviate this problem. Although, it is an open research field. New advances should contribute to improve scalability. Additionally, modeling problems arise when considering the new environment and the energy components. They also compromise scalability and accuracy. A deep study of these issues is mandatory for a better understanding and characterization of sensor networks and their corresponding simulators.

6. ACKNOWLEDGEMENTS

This work has been funded by the Fundacion Seneca, the research council of the Region de Murcia under the ARENA project (00546/PI/04), and by the Spanish Research Council under project reference TEC2004-05622-C04-02/TCM (ARPaq).

7. REFERENCES

[1] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson, “Wireless Sensor Network for Habitat Monitoring.” In *Proc. 1st ACM Int. Workshop on Wireless Sensor Networks and Applications*, Atlanta, GE, pp. 88–97, September 2002.

[2] D. Ganesan, D. Estrin, A. Woo, D. Culler, “Complex Behavior at Scale: An Experimental Study of Low-power Wireless Sensor Networks.” *Technical Report UCLA/CSD-TR 02-0013*, Center for Embedded Networked Sensing, University of California, Berkeley, February 2002.

[3] K. Pawlikowski, H. D. Joshua Jeong, J. S. Ruth Lee. “On Credibility of Simulation Studies of Telecommunication Networks.” *IEEE Communications Magazine*, vol. 40, no. 1, pp. 132–139, January 2002.

[4] D. Kotz, C. Newport, B. Gray, J. Liu, Y. Yuan, C. Elliot. "Experimental Evaluation of Wireless Simulation Assumptions.” In *Proc. of the 7th ACM/IEEE Int. Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM’04)*, Venice, Italy, pp. 78–82, October 2004.

[5] S. Park, A. Savvides, M. B. Srivastava. “SensorSim: A Simulation Framework for Sensor Networks.” In *Proc. ACM Modeling, Analysis and Simulation of Wireles and Mobile Systems (MSWiM 2000)*, Boston, MA, pp. 104–111, August 2000

[6] A. Sobeih, W. Chen, J. C. Hou, L. Kung, N. Li, H. Lim, H. Tyan, H. Zhang, “J-Sim: A simulation and emulation environment for wireless sensor networks.” In *Proc. Annual Simulation Symposium (ANSS 2005)*, San Diego, CA, pp. 175–187, April 2005.

[7] C Perkins, EM Royer, S Das, *Ad-Hoc On Demand Distance Vector Routing (AODV)*, IETF draft, 2000.

[8] *Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*, ANSI/IEEE Std. 802.11, 1999.

[9] T. S. Rappaport, *Wireless Communications, principles and practice*, Second Edition, Prentice Hall, 2002.

[10] The Network Simulator, NS–2 [Online]. Available: http://www.isi.edu/nsnam/ns/

[11] MIT Object Tcl. [Online]. Available: http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl

[12] C. Intanagonwiwat, R. Govidan, D. Estrin, J. Heidemann, F. Silva, “Directed diffusion for wireless sensor networking.” *IEEE/ACM Transactions on Networking*, vol. 11, issue 1, pp. 2–16, February 2003.

[13] W. Ye, J. Heidemann, D. Estrin, “Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks.” *ACM/IEEE Transactions on Networking*, vol. 12, pp. 493–506, 2004.

[14] NRL’s Sensor Network Extension to NS-2 [Online]. Available: http://nrlsensorsim.pf.itd.nrl.navy.mil/

[15] SensorSim: A simulation framework for sensor networks. [Online]. Available: http://nesl.ee.ucla.edu/projects/sensorsim/

[16] V. Naoumov, T. Gross, “Simulation of Large Ad Hoc Networks.” In *Proc. ACM Modeling, Analysis and Simulation of Wireles and Mobile Systems (MSWiM 2003)*, San Diego, CA, pp. 50–57, 2003.

[17] OMNET++ discrete event simulator. [Online]. Available: http://www.omnetpp.org

[18] Mobility Framework for OMNET++. [Online]. Available: http://mobility-fw.sourceforge.net

[19] Consensus: Collaborative Sensor Networks [Online]. Available: http://www.consensus.tudelft.nl

[20] J-Sim [Online]. Available: http://www.j-sim.org

[21] Jacl, Java implementation of Tcl8.x. [Online]. Available: http://www.tcl.tk/software/java

[22] NCTUns 2.0 Network Simulator and Emulator. [Online]. Available: http://nsl.csie.nctu.edu.tw/nctuns.html

[23] R. Barr, Z. J. Haas, R. van Renesse, “JiST: Embedding Simulation Time into a Virtual Machine.” In *Proc. 5th EUROSIM Congress on Modeling and Simulation*, Paris, France, September 2004

[24] Jython. [Online]. Available: http://www.jython.org

[25] Global Mobile Information Systems Simulation Library (GloMoSim). [Online]. Available: http://pcl.cs.ucla.edu/projects/glomosim/

[26] PARSEC: Parallel Simulation Environment for Complex Systems. [Online]. Available: http://pcl.cs.ucla.edu/projects/parsec/

[27] M. Takai, R. Bagrodia, K. Tang, M. Gerla, “Efficient Wireless Networks Simulations with Detailed Propagations Models.” *Kluwer Wireless Networks*, 7, pp. 297–305, 2001.

[28] sQualnet: A Scalable Simulation Framework for Sensor Networks. [Online]. Available: htpp://nesl.ee.ucla.edu/projects/squalnet/

[29] Scalable Simulation Framework (SSF). [Online]. Available: http://www.ssfnet.org

[30] Dartmouth SSF (SSF). [Online]. Available: http://www.crhc.uiuc.edu/ jasonliu/projects/ssf/

[31] L. F. Perrone, D. M. Nicol, “A Scalable Simulator for TinyOS Applications.” In *Proc. ACM 2002 Winter Simulation Conference*, San Diego, CA, pp. 679–687, 2002.

[32] Ptolemy II. Heterogeneous model and design. [Online]. Available: http://ptolemy.eecs.berkeley.edu/ptolemyII

[33] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, Y. Zhao. “Modeling of Sensor Nets in Ptolemy II.” In *Proc. Information Processing in Sensor Networks (IPSN)*, Berkeley, pp. 359–368, April 2004

[34] MICA Motes. [Online]. Available: http://www.xbow.com

[35] TinyOS: Open-source operating system for wireless embedded sensor networks. [Online]. Available: http://www.tinyos.net

[36] P. Levis, N. Lee, M. Welsg, D. Culler, “TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications.” In *Proc. 1st ACM Int. Conf. Embedded Networked Sensor Systems (SenSys)*, Los Angeles, CA, pp. 126–137, 2003.

[37] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, D. Estrin, “EmStar: A software Environment for Developing and Deploying Wireless Sensor Networks.” In *Proc. USENIX 2004*, Boston, MA, pp. 283–296, 2004.

[38] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, T. Schoellhammer, “A System for Simulation, Emulation and Deployment of Heterogeneous Sensor Networks.” In *Proc. 2nd ACM Int. Conf. Embedded Networked Sensor Systems (SenSys)*, Baltimore, MD, pp. 201–213, 2004.

[39] J. Polley, D. Blazakis, J. McGee, D. Rusk, J. S. Baras, M. Karir, “ATEMU: A Fine-grained Sensor Network Simulator.” In *Proc. 1st IEEE Int. Conf. Sensor and Adhoc Communication Networks (SECON’04)*, Santa Clara, CA, October 2004.

[40] S. Sundresh, W. Kim, G. Agha, “SENS: A Sensor, Environment and Network Simulator.” In *Proc. 37th ACM Annual Symposium on Simulation*, Washington, DC, pp. 221, 2004.

[41] PROWLER: Probabilistic Wireless Network Simulator. [Online]. Available: http://www.isis.vanderbilt.edu/projects/nest/prowler

[42] C. Kelly, V. Ekanayake, R. Manohar, “SNAP: A Sensor-Network Asynchronous Processor.” In *Proc. 9th ACM Int. Symposium on Asynchronous Circuits and Systems*, Washington, DC, pp. 24, 2003.

[43] J. Cowie, D. Nicol, A. Ogielski, “Modeling the Global Internet.” *Computing in Science and Engineering*, vol. 1, issue 1, pp. 42–50, January 1999.