# Comparative Study of Wireless Network Simulators

Johannes Lessmann, Peter Janacik, Lazar Lachev, Dalimir Orfanus
University of Paderborn
Faculty of Computer Science and Mathematics
Fuerstenallee 11, 33102 Paderborn, Germany
{lessmann, pjanacik, lachev, orfanus}@upb.de

## Abstract

*In order to evaluate the behavior and performance of protocols for wireless networks, simulations are a good compromise between cost and complexity, on the one hand, and accuracy of the results, on the other hand. Since there are many simulators for wireless networks, it is often difficult to decide which simulator to choose. To help shed light on this issue, we present a case study in which four popular wireless network simulators were used to evaluate a well-known topology control protocol (SPAN). Within the case study, we describe outstanding and desirable but missing features of the simulators, outlining their strengths and weaknesses. Further, we compare the amount of effort needed for installation, familiarization, implementation (needed lines of code and lines for configuration) and visualization. As opposed to other simulator comparisons, we do not focus on the correlation of the individual simulation results, but try to compare the simulators from feature and usability point of view. This paper can help other researchers to quickly identify which simulator is most suitable for their needs.*

## 1. Introduction

Wireless networks are composed of nodes which communicate using wireless links. Typically, they work on achieving a common goal like environmental monitoring, communication, and so on. It is in the nature of such networks that communication between the nodes is unstable, since the quality of the wireless links is fluctuating heavily. Further, as wireless nodes are often small and thus very resource-constrained, it is also generally not feasible to implement algorithms with a large processing power or memory footprint. All this makes designing protocols for wireless networks a challenging task. Therefore, it is inevitable to thoroughly test protocols to be able to assess their performance in the anticipated application scenario.

In order to do so, there are two possible options. First, the developed concepts can be implemented and deployed on actual hardware. This is called testbed implementation. While testbeds might yield the most accurate results, there are several drawbacks, such as: the need to obtain hardware and the severely limited monitoring and debugging possibilities, as well as, high effort needed to create an artificial environment resembling the real application scenario. Hence, testbed implementations will generally be an option only for smaller numbers of nodes and during the later stages of the implementation phase.

A second option to test network protocols is simulation. Here, the physical world is modeled in a wireless network simulation software. This avoids the issues with testbed implementations outlined above. On the other hand, models can capture reality only to a limited extent, which implies that simulation results will generally not be as accurate as real implementations.

For network protocol designers, it is often difficult to decide which simulator to choose for a particular task. Therefore, we conducted a case study in which four network simulators (J-Sim, OMNeT++, ns-2, and ShoX) were selected and then used to implement a well-known topology control algorithm called SPAN [3]. Our focus in this case study was not to compare the correlation of the individual simulation results, as done in other comparison studies [15, 2], but rather to evaluate the candidates regarding their feature set and usability. We describe outstanding and desirable but missing features of the simulators, outlining their strengths and weaknesses. Further, we compare the amount of effort needed for installation, familiarization, implementation, debugging and visualization. The findings in our study will allow a quick yet deep enough understanding which can help other researchers to identify the wireless network simulator that is most suitable for their needs.

The remainder of this paper is organized as follows. In Section 2, we introduce previous work related to our study (i.e. other simulator comparisons). In Section 3, we briefly review SPAN in order to be later able to refer to certain as-

978-0-7695-3106-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ICN.2008.97

pects of SPAN in the case study. Section 4 is the main section of this paper where the four simulators and our findings are discussed in detail. The paper closes with a summary in Section 5.

## 2 Related Work

There are several surveys, comparisons, and also some case studies about wireless network simulators. They all differ with respect to the selection of evaluated simulators, the intention of the work (description or comparison), the focus of the potential comparison (credibility of results, features, performance, etc.) and the level of detail. Table 1 provides a short overview.

**Table 1. Overview of some previous network simulator comparisons**

| Paper | Type | Simulators | Focus |
|---|---|---|---|
| [20] | comparison | Opnet,ns-2 | setup,result accuracy |
| [19] | case study | ns-2, Opnet, GloMoSim | architecture, results |
| [17] | comparison | ns-2, cnet, JNS, Opnet, AdventNet, NCTUns | features (limited) |
| [16] | case study | J-Sim, ns-2, SSFNet | scalability, speed, memory requirements |
| [8] | comparison | Opnet, ns-2, QualNet, OMNeT++, J-Sim, SSFNet | suitability for critical infrastructures |
| [14] | comparison | ns-2, Avrora, Opnet, GloMoSim | architecture, functionality, extensibility, resource requirements |
| [12] | comparison | ns-2, TOSSIM | architecture, components, models, visualization |
| [11] | description | GloMoSim, ns-2, DIANEmu, GTNetS, J-Sim, Jane, NAB, PDNS, OMNeT++, Opnet, QualNet, SWANS | overview |
| [10] | comparison | SSF, SWANS, J-Sim, NCTUns, ns-2, OMNeT++, Ptolemy, ATEMU, EmStar, SNAP, TOSSIM | models, type of visualization |
| [9] | description | OMNeT++, REAL, ns-2, C++Sim, cnet, SSFNet, CLASS, SMURPH | overview |
| [4] | description | ns-2, GloMoSim, Opnet, SensorSim, J-Sim, Sense, OMNeT++, Sidh, Sens, TOSSIM, ATEMU, Avrora, EmStar | overview |
| [1] | comparison | Opnet, ns-2, OMNeT++, SSFNet, QualNet, J-Sim, Totem | availability/credibility of models, usability |
| [15] | case study | Opnet, ns-2, testbed | accuracy of results |
| [2] | case study | Opnet, ns-2, GloMoSim | accuracy of results |

Actually, all of the works listed in Table 1 consider different simulators or differ in their scope from this paper. The ones that are closest to our work are [8, 11, 10, 4, 1] as they include at least the three simulators J-Sim, OMNeT++ and ns-2, which we also consider. However, [8] examines their suitability for simulating the failure of critical infrastructures like electricity or telecommunication networks. This is very unrelated to what we present here. [11] and [4], although their list of simulators is huge, do not give a comparative study. Rather, they provide more or less unstructured descriptions of each of the simulators independently. In this paper, we aim at comparing simulators according to certain metrics. In [10], the authors give an overview about different issues in wireless sensor networks on a general basis. Only at the end of their work they present a table comparing the considered simulators according to their language, the available modules, and whether they have GUI support or not. Aside from the table, no detailed comparisons between the individual simulators are given.

The most detailed comparison is presented in [1]. The authors describe a variety of simulators including J-Sim, OMNeT++ and ns-2 according to the criteria listed in Figure 3. However, there are a number of important differences: (1) They consider all simulators from an industrial research point of view, hence, they focus on issues such as support for certain models (which are required for their project in mind), quality of human support, etc., which are rather less relevant for academic researchers. (2) They do not consider certain aspects which are important here, like installation issues, and discuss visualization and statistics only very briefly. (3) Their case study misses practical simulations and experiences: there are no practical experiences regarding installation, familiarization or implementation issues. This, however, is the focus of our case study.

## 3 SPAN

SPAN [3] is a topology control algorithm aiming at saving power without reducing the network capacity or loosing connectivity. It does so by electing and constantly adjusting a set of active nodes called coordinators. The coordinators form a forwarding backbone of the underlying network while the non-coordinator nodes can switch their radios to sleep mode. From time to time the non-coordinator nodes check if they should become coordinators. Similarly, the coordinator nodes regularly check whether there are enough other coordinators in their neighborhood in which case they can go to sleep mode.

In [3], SPAN is implemented on top of the 802.11 MAC layer's power-saving mode. For the routing layer, the authors of SPAN chose a simple greedy geographic forwarding approach, similar to GPSR [13], but without perimeter routing around voids. Principally, each routing protocol would do as well. As SPAN has to interact with both routing and MAC layer, in [3] it is implemented in the logical link control layer of the ISO OSI stack. For our implementations in the case study, we will follow their example.

## 4 Case Study

For our case study, we selected four different network simulators to implement the wireless topology control protocol SPAN, described in the previous section. We chose to select three well-established simulators: J-Sim [6], OMNeT++ [21] and ns-2 [5]. Further we took a look at a relatively new project, called ShoX [7]. The main reason why

we chose the former three simulators is because of their popularity in the research community. While other simulators like Opnet [18] are also popular, for our case study we only considered simulators which are freely available.

Opposed to J-Sim, OMNeT++ and ns-2, ShoX is comparably new. However, it is one of the few simulators which offer a comprehensive graphical user interface for configuration, visualization and statistics. Additionally, it was developed from the beginning targeted toward *wireless* networks, whereas most other popular simulators (including J-Sim, OMNeT++, ns-2 and Opnet) initially concentrated on *wired* networks and were later extended to support the wireless domain. Hence, for ShoX, there is no special wireless package as with the others, all functionality pertaining to wireless is an integral part of the software. We were especially interested to see in how far the focus on wireless instead of on wired networks has an impact on usability and the learning curve.

The case study was performed by four people (the authors). To ensure fairness, only one of them did the actual installation and implementation task for all four simulators. This was to rule out any influences caused by potentially different programming proficiency or knowledge of computer network protocols. For all simulators, the implementation of SPAN was conducted under Linux (Fedora 7) using the open-source platform Eclipse 3.3. The responsible student was not familiar with either of the four simulators before the case study. He had sufficient knowledge of popular programming languages like Java and C++. Before the start of the case study, he familiarized himself with SPAN, so that understanding problems with SPAN itself were not an issue during the implementation phase. During the case study, he carefully recorded each necessary step, questions that arose, efforts it took to find answers to them, features of the simulator that were helpful or confusing, and the time he needed for the individual parts. His progress and experiences were discussed weekly among the authors.

## 4.1 J-Sim

J-Sim (formerly JavaSim) is a network simulator written in Java. It is built according to the component-based software paradigm. In J-Sim terminology, this is called *autonomous component architecture* (ACA). Everything in J-Sim is a *component*: a node, a link, a protocol. Each component can be atomic or composed of other components. Connection between components is done through *ports*. Actually, there are three possible ways to connect ports: one-to-one, one-to-many, and many-to-many. On a more abstract level, J-Sim distinguishes two layers. The lower layer *Core Service Layer* (CSL) comprises every OSI layer from network to physical, the higher layer comprises the remaining OSI layers.

For wireless network simulations, J-Sim offers the Wireless Extension. Here, several components and their relationships are defined and extend the general CSL. Figure 1 gives an overview of the most important components. The only available MAC layer component in the Wireless Extension is 802.11 MAC.

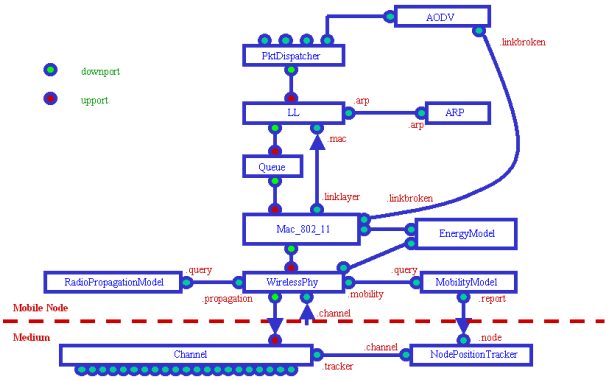


**Figure 1. Overview of components in J-Sim's wireless extension (source [6]).**

When a node wants to send a message, it goes through 802.11 MAC which decides when the packet is sent to the WirelessPhy. The latter determines the nodes current position from the MobilityModel and adds that position, the current transmission power and the antenna gain to the MAC frame. The receiving node's WirelessPhy consults the RadioPropagationModel to decide if the packet should be passed to the MAC. The EnergyModel is a collection of five energy consumption values (radio states send, receive, idle and off). When the energy is depleted, no packets can be sent and received anymore.

### 4.1.1 Installation

Once J-Sim is downloaded, it can be easily imported into Eclipse using Eclipse's "Java Project from Existing Ant Buildfile". However, some additional java archives for XML handling (jaxp, xalan and crimson) must be installed before J-Sim can be executed. The whole installation and configuration process took approximately 1,5 days, mainly because of the JVM problems.

### 4.1.2 Implementation and Documentation

J-Sim offers good introductory material with overviews and examples for small scenarios. However, it lacks a comprehensive manual. Several more specific questions remain undocumented, for example how to send broadcast. Also, we did not find any hint as to how a new packet is to be defined.

The problem was that the MAC layer, which had to communicate with SPAN (see Section 3), assumed out SpanPacket to have certain fields and parameters which were nowhere clearly expressed, but led to erroneous behavior nonetheless.

J-Sim uses Tcl for configuration of simulation scenarios. This requires a certain learning overhead. The binding between Java and Tcl (to be able to access Java objects and methods from Tcl) is pretty intuitive. There is also a graphical editor for the Tcl configuration files called gEditor.

The familiarization with the configuration part took us around two days. Another three days were spent to solve the problems mentioned above. The implementation itself, simulator-specific problems aside, took ten days. J-Sim offers both AODV and GPSR, therefore testing of SPAN was quite simple.

#### 4.1.3 Visualization and Statistics

J-Sim has no tool for network visualization itself. However, it allows generating trace files which conform to ns-2's nam (network animator) format. To plot simulation statistics, a special plot component is provided.

### 4.2 OMNeT++

OMNeT++ is a simulation platform written in C++. Like J-Sim, it has a component-based, modular and extensible architecture. Thus, its structure shares many properties with J-Sim's. The basic entity in OMNeT++ is a module. Modules can be composed of submodules or they can be atomic. Only atomic modules capture the actual behavior. Modules communicate with each other via messages through *gates*. Gates are linked to each other using *connections*. A connection can be associated with a propagation delay, error rate and data rate. Unlike J-Sim's ports, gates in OMNeT++ support only one-to-one communication.

Regarding simulation of wireless ad hoc networks, OMNeT++ relies on external extensions. The two most prominent ones are the INET Framework (IF) and the Mobility Framework (MF). While the latter is an extension explicitly designed for mobile ad hoc networks we chose it for our case study.

Figure 2 depicts the structure of OMNeT++/IF. Aside from the most important OSI layers, OMNeT++/IF provides two modules called blackboard and mobility. A blackboard is used to share cross-layer data. The mobility part is responsible for providing and updating a node's current position and establishing communication channels. MAC and PHY layers are composed into a single NIC (network interface card) module. The physical layer is split into a module which determines SNR characteristics and another one responsible for deciding whether a packet can be passed upwards.

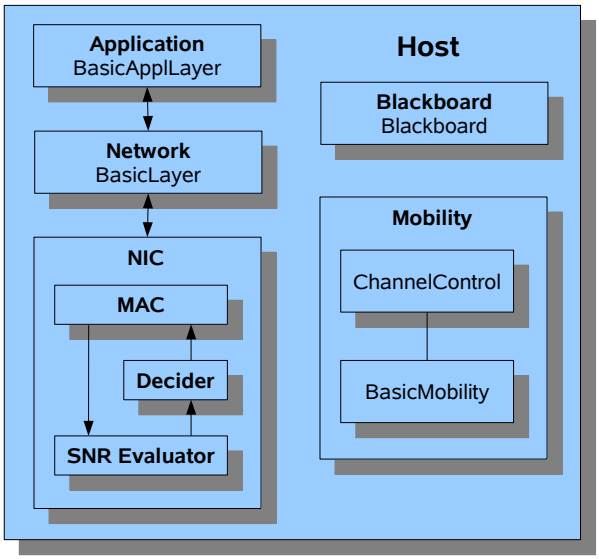


**Figure 2. Overview of the basic modules in OMNeT++/(INET Framework).**

#### 4.2.1 Installation

OMNeT++ (3.3) is installed using configure and make scripts. Before installing OMNeT++, the two additional packages Tcl/Tk and BLT (set of new commands and widgets) must be installed. Even though BLT was already installed on our system, it was somehow not found by the installation script. Wireless extensions are installed using command line make commands.

The time it took for installing OMNeT++ and importing to Eclipse was approximately three days. Some of the delay was caused by a "problem" with BLT. Another issue was to figure out how to integrate the IF with OMNeT++ (it slightly changes the build process but that is not documented well).

#### 4.2.2 Implementation and Documentation

OMNeT++ has a well-written fairly large user manual while IF has only an API documentation. OMNeT++ is very complex, thus careful consultation of the available documents is needed. To understand and run small examples took us approximately three days. Scenario configuration is done in so-called network description files.

One major drawback of OMNeT++ is that it does not have an energy model. Thus, while OMNeT++ certainly is a feature-rich and powerful simulation platform, it was not possible to implement and test SPAN completely. Another issue was finding a suitable routing protocol for testing. While there is an AODV implementation listed on the website, the referenced page refused to load. Whereas the MF

includes AODV, the IF does not and GPSR is not available with OMNeT++. Hence, we decided to use DYMO which integrates well with IF. These issues, especially the search for a suitable MAC and AODV implementation added another two days to our simulator-specific overhead time. Implementation itself (to the extent possible) lasted nine days.

#### 4.2.3 Visualization and Statistics

OMNeT++ is the only simulator with online visualization. Hence, users can pause the simulation and inspect or even directly change values in the models. It is also possible to change a node's appearance (color, size, shape, etc.) to reflect an inner state which the user wants to visualize. Statistics can be written to a trace file and displayed with external but commonly available tools like prove.

### 4.3 ns-2

The network simulator ns-2 is based on two languages: an object-oriented simulator, written in C++, and an OTcl (an object-oriented extension of Tcl) interpreter to execute user's command scripts. There are two class hierarchies: a compiled C++ one (which captures the protocol behavior) and an interpreted OTcl one for binding to the OTcl scenario configuration script.

Ns-2 offers a reduced OSI layer model in which the presentation and session layers are left out. For wireless network simulations, ns-2 offers a variety of features. It has an energy model, and both, traffic and movement patterns, can be easily generated. However, as opposed to the other three candidates, traffic and mobility are typically produced before the actual simulation start and are not so much an integral part of ns-2's architecture.

#### 4.3.1 Installation

There are basically two ways to obtain ns-2: by downloading the all-in-one package or only selected components and libraries. Unfortunately, each time the user code is modified, ns-2 itself will be recompiled. Our solution to this problem was to compile our code into a separate shared library and link that to the ns-2 kernel.

#### 4.3.2 Implementation and Documentation

From all the four simulators we tested, ns-2 clearly has the steepest learning curve, even though its documentation is comprehensive. For ns-2, there is a manual which is regularly updated. Further, there is an API for the C++ and OTcl classes (although the latter are not explained very thoroughly). Still, working with ns-2 requires learning many concepts. This starts with the object-oriented version of Tcl called OTcl which is used for scenario configuration. It also includes the structure of the configuration environment itself. This is unfortunately not as intuitive as with other simulators. We needed approximately eight days to become familiar enough with the complete environment.

In ns-2, the not very easy-to-use OTcl handles the task of describing the simulation scenario. To construct a binding between OTcl and the actual C++ classes, each C++ class must be accompanied by a corresponding OTcl class, causing a considerable overhead. For the implementation itself, we needed about 3.5 days. The short length of this timespan is due to the fact that we simply had to adapt our C++ classes from the ones we already wrote for OMNeT++. One of the major advantages of ns-2 is the huge pool of available features, offering a large number of external protocols already implemented.

#### 4.3.3 Visualization and Statistics

In order to visualize network behavior in ns-2, one must first of all call two scripts: one to generate a traffic trace file and another one to create a movement trace file. These two trace files can then be referenced as an input in the Tcl configuration for the actual simulation process (i.e. the network is simulated with the specified traffic and movement patterns). The simulation in turn generates a log file which can then be visualized using ns-2's network animator (nam). nam is similar to OMNeT++ in the way that it can visualize not only nodes, links, movements, packets, etc. but also changing node states by adapting the graphical appearance of the node. However, the possibilities in nam regarding a dynamic change of appearance are rather limited.

For plotting statistics, a function in the OTcl configuration file is used, which is called initially at simulation start time (or any other time), and which periodically calls itself. In each execution of this function, some statistical values may be written to a file. After the simulation end, an appropriate external tool (we used xgraph) is used for plotting the data.

### 4.4 ShoX

ShoX is an object-oriented network simulator written in Java, which was targeted at wireless networks from the beginning. By default, its architecture follows the OSI seven-layer model, although only five of them are present by default. However, all layers are derived from an abstract super-class and are defined by LayerType objects. Hence, it is straightforward to include additional layers at any position in the stack for special purpose simulations. ShoX does not use components as e.g. J-Sim or OMNeT++. Rather, protocols, energy management, propagation or mobility models, etc. are all derived from abstract super-classes which define the minimum interface and functionality. The

different entities in ShoX communicate through *events*. Devices are special kinds of components of a node like the network interface card, the power manager, the CPU or attached sensors.

In addition to the abstract OSI layer classes, there is a special "layer" called AirModule. Here, all channel related issues are handled (e.g. signal interference). PhysicalModel and InterferenceHandler of ShoX resemble the SNR Evaluator and Decider in OMNeT++. As Opposed to OMNeT++, forward error correction is handled by the actual layer implementations, which appears to more resemble reality.

Like J-Sim and ns-2, ShoX has an energy manager component. However, in ShoX, the energy manager is far more advanced making use of the concept of a device: different devices can be registered as power suppliers (e.g. solar panel) or power consumers (e.g. CPU, sensors).

| Aspect | J-Sim | OMNeT++ | ns-2 | ShoX |
|---|---|---|---|---|
| energy model | ✔ | – | ✔ | ✔ |
| 802.11 power-save | ✔ | – | ✔ | – |
| SPAN completed | ✔ | – | ✔ | – |
| AODV | ✔ | ✔ | ✔ | ✔ |
| DSR | – | – | ✔ | – |
| GPSR | ✔ | – | ✔ | ✔ |
| visualization | nam trace file, no own tool | online with model inspection, to go back, simulation must be repeated | trace file, can be viewed with nam | trace file, internal viewer |
| statistics | online plot, exporting to file must be done by user | trace file, can be displayed with plove | log file, can be displayed with xgraph | statistics file, internal viewer or export to gnuplot |
| strengths | + flexibility<br>+ Java based | + maturity<br>+ model inspection<br>+ GUI support | + model base<br>+ user base | + GUI support<br>+ visualization<br>+ architecture |
| weaknesses | - GUI support<br>- visualization capabilities | - energy model<br>- MAC competitors | - OTcl<br>- architecture | - documentation<br>- lack of models |

**Figure 3. Simulator feature matrix**

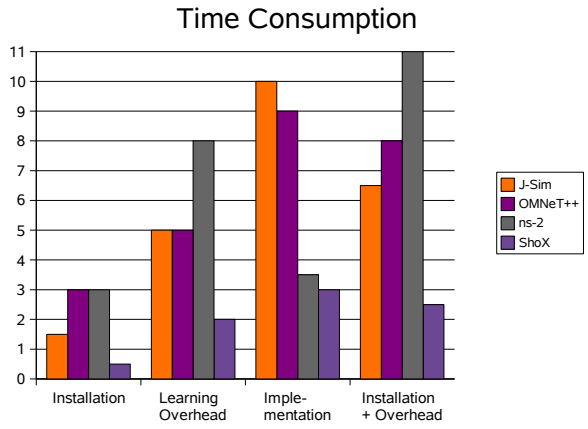


**Figure 4. Comparison of consumed time**

### 4.4.1 Installation

ShoX can be downloaded as a source package. However, we followed the recommendation on the website to instead directly use the more recent CVS version, since the release version (0.2) is rather outdated. Using Eclipse's "Projects from CVS", it is principally straightforward to import the CVS code into the tool. Unfortunately, no documentation is provided on the website on how to configure CVS, hence, the corresponding Sourceforge documentation must be consulted. After the setup in Eclipse, ShoX is started through Eclipse's run dialog. Despite the fact that a sufficiently detailed documentation is missing, trying to figure out the right configuration took us approximately half a day.

### 4.4.2 Implementation and Documentation

Although missing a user manual, ShoX provides an API documentation which contains explanations for most of the classes and their members. Getting familiar with ShoX took us about two days.

Scenario generation in ShoX is done through its GUI. Unlike in gEditor or gNED, there are no modules and links to be drawn. The configuration UI in ShoX is a wizard leading through the necessary steps for all needed elements. It appears that ShoX focuses on ease of use, which in some cases (e.g. configuration) reduces the amount of available possibilities. However, for our chosen protocol, SPAN, the approach is completely sufficient. We needed three days to implement SPAN. Again, we could adopt a lot of code from our J-Sim classes. Unfortunately, while the energy management of ShoX is the most advanced among all four simulators, its 802.11 MAC does not support the power-save mode.

### 4.4.3 Visualization and Statistics

Regarding visualization and statistics, ShoX is the most powerful and integrated simulation platform among the four candidates. It includes both, a network and a statistics visualizer, in the same GUI (from which also the configuration is done and the simulation started). Like OMNeT++ and nam, ShoX can visualize node movements, links and packets. Regarding node state representation, the mapping between the node state (which is logged in the simulation log file) and the desired graphical representation of that state in terms of node color, size, shape, labels, border color and border width can be changed retroactively and even while the visualization is running. In addition to the node state visualization, ShoX supports visualizing link states by changing the link appearance in the same fashion. ShoX also offers a statistics chart generation with three different chart types.

| | J-Sim | OMNeT++ | Ns-2 | ShoX |
|---|---|---|---|---|
| Lines of code | 873 | 942 | 1037 | 791 |
| Lines of code (configuration) | 255 | 253 | 180 | 76 |
| Number of classes | 6 | 5 | 5 | 6 |

**Figure 5. Comparison of lines of code needed**

## 5 Conclusion

In this paper, we have presented the results of a case study in which we compared the wireless network simulators J-Sim, OMNeT++, ns-2 and ShoX by implementing a simple topology control algorithm called SPAN. We evaluated strengths and weaknesses of each simulator with respect to installation, implementation issues and visualization capabilities. The results of our studies are summarized in Figures 3, 4 and 5. We have seen that none of the four simulators is a clear winner in all areas. Each of them also showed areas of relative weakness compared to the other candidates. OMNeT++ and ns-2 are the most mature ones. While OMNeT++ shined at GUI support, ns-2 profits from the large number of available models. Obviously, both support productivity. On the other hand, J-Sim attracts because of its flexible component-based architecture and ShoX is outstanding when it comes to visualization. Concerning the amount of effort it takes to become familiar with a simulator, we observed a clear order from ns-2, over OMNeT++ to J-Sim and ShoX. While we think this is on the one hand due to architectural decisions, part of it stems from the feature richness of ns-2 and OMNeT++, especially regarding their scenario configuration capabilities.

## References


[1] L. Begg, W. Liu, K. Pawlikowski, S. Perera, and H. Sirisena. Survey of simulators of next generation networks for studying service availability and resilience. Technical Report TR-COSC 05/06, Department of Computer Science & Software Engineering, University of Canterbury, Christchurch, New Zealand, February 2006.

[2] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of manet simulators. In *Proceedings of Principles of Mobile Computing (POMC) 2002*, Toulouse, France, October 2002.

[3] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networks*, 8(5):481–494, September 2002.

[4] D. Curren. A survey of simulation in sensor networks. Student project, `www.cs.binghamton.edu/~kang/teaching/cs580s/david.pdf`, 2007.

[5] DARPA/NSF. The network simulator - ns-2. `http://www.isi.edu/nsnam/ns/`.

[6] T. J.-S. developers. J-sim. `http://www.j-sim.org`.

[7] T. S. developers. Shox - a scalable ad hoc network simulator. `http://shox.sourceforge.net`.

[8] S. Duflos, G. L. Grand, A. A. Diallo, C. Chaudet, A. Hecker, C. Balducelli, F. Flentge, C. Schwaegerl, and O. Seifert. Deliverable d 1.3.2: List of available and suitable simulation components. Technical report, Ecole Nationale Supieure des Tommunications (ENST), September 2006.

[9] V. Efthimia. Free tools for network simulation. Master's thesis, University of Macedonia, Thessaloniki, 2006.

[10] E. Egea-Lopez, J. Vales-Alonso, A. Martinez-Sala, P. Pavon-Mari, and J. Garcia-Haro. Simulation scalability issues in wireless sensor networks. *IEEE Communications Magazine*, 44(7):64–73, July 2006.

[11] L. Hogie, P. Bouvry, and F. Guinand. An overview of manets simulation. In *Electronic Notes in Theoretical Computer Science, Proc. of 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord 2005)*, LNCS, pages 81–101, Namur, Belgium, April 2005. Elsevier.

[12] M. Karl. A comparison of the architecture of network simulators ns-2 and tossim. In *Proceedings of Performance Simulation of Algorithms and Protocols Seminar*. Institut fr Parallele und Verteilte Systeme, Abteilung Verteilte Systeme, Universit Stuttgart, 2005.

[13] B. Karp and H. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking*. 2000.

[14] A. Lemke and A. Sarkohi. Werkzeuge zur netzwerksimulation. In G. Wittenburg, editor, *Proceedings of Seminar Technische Informatik*. Freie Universit Berlin, June 2006.

[15] G. F. Lucio, M. Paredes-Farrera, E. Jammeh, M. Fleury, and M. J. Reed. Opnet modeler and ns-2 - comparing the accuracy of network simulators for packet-level analysis using a network testbed. *WSEAS Transactions on Computers*, 2(3):700–707, July 2003.

[16] D. Nicol. Comparison of network simulators revisited. `http://www.ssfnet.org/Exchange/gallery/dumbbell/dumbbell-performance-May02.pdf`, May 2002.

[17] P. Nov. Simulation of network structures. Master's thesis, Department of Software Engineering, Charles University in Prague, August 2006.

[18] I. Opnet Technologies. Opnet. `http://www.opnet.com`.

[19] R. Repp. Vergleich der verfahren simulation und emulation fr die evaluation von protokollen. Master's thesis, Institut fr Parallele und Verteilte Systeme (IPVS), Universit Stuttgart, December 2003.

[20] B. Schilling. Qualitative comparison of network simulation tools. Technical report, Institute of Parallel and Distributed Systems (IPVS), University of Stuttgart, January 2005.

[21] A. Vargas. Omnet++ - discrete event simulation system. `http://www.omnetpp.org`.