

WSNGE: A Platform for Simulating Complex Wireless Sensor Networks Supporting Rich Network Visualization and Online Interactivity¹

Marios Karagiannis², Ioannis Chatzigiannakis³ and Jose Rolim²

² Centre Universitaire d' Informatique, Geneva, Switzerland. E-mail: {marios.karagiannis, jose.rolim}@unige.ch

³ Research Academic Computer Technology Institute (CTI) and University of Patras, Greece. E-mail: ichatz@cti.gr

Keywords: Wireless Sensor Networks, Multiprotocol simulation, Network Visualization

Abstract

In this paper we describe a new simulation platform for complex wireless sensor networks that operate a collection of distributed algorithms and network protocols. Simulating such systems is complicated because of the need to coordinate different network layers and debug protocol stacks, often with very different interfaces, options, and fidelities. Our platform (which we call WSNGE) is a flexible and extensible environment that provides a highly scalable simulator with unique characteristics. It focuses on user friendliness, providing every function in both scriptable and visual way, allowing the researcher to define simulations and view results in an easy to use graphical environment. Unlike other solutions, WSNGE does not distinguish between different scenario types, allowing multiple different protocols to run at the same time. It enables rich online interaction with running simulations, allowing parameters, topologies or the whole scenario to be altered at any point in time.

1. INTRODUCTION

A particularly promising and hot research area has been the design and analysis of *wireless sensor networks* (WSN), which has attracted researchers from very different backgrounds, such as hardware, software, algorithms, and data structures, as well as researchers from various application areas. Advancements have been made in the physical hardware level, embedded software in the sensor devices, systems for future sensing applications and fundamental research in new communication and networking paradigms. Although these research attempts have been conducted in parallel, in most cases they were also done in isolation, making it difficult to converge towards a unified global framework. We envision that future research on wireless sensor networks will exploit its theoretical and practical dimensions in parallel.

From a practical point of view, a joint approach of conducting research that combines both theory and practice must overcome considerable difficulties. Still, even if all skills

from all diverse specialties are acquired, the cost of setting up and maintaining an experimental facility of significant size can be very high. Deploying the experimental network into a realistic environment requires iteratively reprogramming dozens of nodes, positioning them throughout an area large enough to produce an interesting radio topology, and instrumenting them to extract performance data. This may have contributed to the fact that only few of these networks have yet been deployed [23, 19, 13].

An alternative to experimenting with actual sensor networks in order to validate and evaluate the performance of applications is software simulation. Simulators provide a number of advantages over real world deployments, such as, e.g., the ability to test a great variety of parameters in protocols and deployment settings, including repeatable scenarios, isolation of parameters, exploration of a variety of metrics and ease of development by leaving out all the details of deploying a real testbed. Currently, a number of simulators have been developed, most notable examples are **ns-2** [14], **OM-NeT++** [15], **OPNET Modeler** [3], **GTNetS** [2], **YANS** [12] and **Shawn** [10].

Even though most simulators provide the basic framework required to simulate a generic wireless network, in the context of wireless sensor networks, more facilities are required to produce realistic simulation scenarios that capture real world uses of these networks. For example, in real world scenarios it is uncommon for nodes to communicate without restrictions inside the network area, but still *obstacle modeling and simulation* is absent from most simulators. Even when considering wireless sensor networks, where several models assume nodes are prone to errors and hardware failures, there is usually no clearly defined functionality to *model failures*. We note that the largest part of the research papers which deal with such networks, and use simulation as an analysis tool, make *a lot* of simplifying assumptions to reduce the overall simulation complexity, be it in the simulation setup, the simulation scenarios, or the simulator used itself. Such oversimplifications are often required in order to conduct experiments with a great number of nodes in reasonable time, but most frequently they are imposed by the lack of functionality in the simulation tools. In this way the credibility of such research works is weakened. An analytic and meaningful discussion of common pitfalls and shortcomings in the simulation of wireless networks in general is included in [9] and [11].

¹This work has been partially supported by the IST Programme of the European Union under contract number IST-2005-15964 (AEOLUS) and ICT-2008-224460 (WISEBED)

1.1. Motivation & Our Contribution

In this paper we present WSNGE, a simulation platform, for the implementation, simulation, and evaluation of complex wireless sensor networks. WSNGE allows the simulation of multiple distributed algorithms that are executed concurrently on the nodes of the system and that are combined in order to implement the desired application. It focuses on providing a rich interface to monitor the execution of the system and interact with the system state during the simulation. The design goals of the WSNGE environment are to allow the protocol designer to create his own environment, to program in a natural style, very similar to the style of programming for the final real-world application, to be of substantial help to the application designer by providing means to monitor the execution of the system and to extract detailed statistical information, to be used both as a tool for experimental analysis and for demonstration or educational purposes, be easy to extend or integrate in other systems, to be efficient and easy to use and, finally, to be able to produce realistic simulation scenarios for wireless sensor networks.

Many systems (e.g., *ns*, *Shawn*) provide animations of the simulated system based on system traces extracted during the simulation. They are only animated visualizers and don't provide simulation capabilities. For this purpose WSNGE includes a GUI that can be used to monitor aspects of the simulation in *real time*. This feature is quite important since it allows to pause the simulation, make changes and resume the simulation in order to measure the effect of certain events on the robustness of the algorithm. The protocol designer is able to interact with the simulation while the experiment is still ongoing, introducing obstacles, packets, new topologies and new properties. To the extent of our knowledge, the existing solutions provide poor or no interaction with running simulations whatsoever. The GUI includes modules for designing the topology, constructing scenarios and viewing statistics. A powerful feature of WSNGE is that it provides multiple viewports of the *same* simulation with customizable information displayed on each, both to avoid clutter and to focus on certain aspects of the execution, making it an ideal platform for demonstration or educational purposes.

Another limitation of most of the available systems is that they offer a generic simulation environment, while our solution focuses on Wireless Sensor Network simulation. WSNGE provides a set of tools that make WSN simulation easier. It also provides WSN specific features like communication obstacles or battery life by default. We note that very few simulation platforms exist that are specialized in wireless sensor networks, most notable examples are **AlgoSenSim** [1] and **TRAILS** [4].

Another primary concern is to provide an open system that can be used as a scenario editor, online monitoring and presentation tool for other platforms. To support these func-

Simulator	Type	Size (# Nodes)	Visualization
ns-2/NAM	Generic	10^3	Offline
Shawn	Generic	10^6	Offline
OMNeT++	Generic	10^2	Offline
DAP/ADAPT	Generic	10^3	Online
WSNGE	WSN	10^5	Online

Table 1. Simulator characteristics

tionalties, WSNGE has a very flexible modular architecture where components have very low coupling so that they can be easily removed and replaced by external ones in order to come up with customized simulation platforms.

The use of WSNGE is rather intuitive and does not distract the developer from the main task, which is the design, development and testing of the algorithm.

2. PREVIOUS WORK

In this section we review four, of what we believe to be important representatives, software solutions that provide visualization for network topologies. Each network visualizer approaches the task with interesting ideas and innovations but at the same time retains its own philosophy. In Table 1 we attempt to classify these simulators based on their characteristics and capabilities.

2.1. Nam, the VINT Network Animator

NAM's [8] philosophy is to provide a detailed animation of the execution of the algorithm in test. It uses the trace files provided by other simulation environments (such as NS) to render a fully interactive animated representation of data flows, connections and network layout. NAM is only an animated visualizer and does not provide any simulation capabilities. It does however include tools for constructing simulation scenarios and extract them so that external tools can run them.

By using the time-indexed events from the trace files, NAM can provide offline re-runs of simulations. Online visualization is also possible using UNIX pipes as input. By taking advantage of the network properties descriptions, NAM provides its core feature, which is packet animation. Using graph layout drawing algorithms, it represents the physical properties, such as latency and bandwidth of links, and animates packets on the edges. The resulting animated graphs can be interacted with, providing extra information and statistics about their elements (packets, edges, nodes etc.).

Nam's scenario creation and editing capabilities consist of the scenario input facility which produces ns scripts for ns to execute and the usage of Nam as a visualizer during NS's scenario generation. The thing to note here is that Nam's visualization capabilities are restricted to relatively small scenarios

(roughly up to 100 nodes) while support for large network topologies scenarios is future work.

2.2. Shawn

Shawn [10, 18] is a discrete event simulator that focuses on large scale network simulations. Its philosophy is to use a higher level simulation model, that focuses on the algorithmic interesting part of a simulation, which is the effects of the execution of the algorithm as opposed to simulating all the mechanisms that lead to this effect. Shawn uses abstract and exchangeable models for lower level effects, like communication and transmission models instead of simulating every such effect in the most realistic way. This technique allows Shawn to run large scale experiments in a fraction of the time that would take similar simulators for the same experiments. By simulating the effects, Shawn allows the researcher to easily prototype algorithms rapidly, by developing the higher level functionality. By exchanging the lower level models, behaviour can be tested for different but similar cases.

Shawn's visualization features are plain. It mainly uses an external tool called Vis [22] to provide visualizations of discrete time instances of the simulation, including drawing of nodes and edges. Each visualized instance is recorded to PNG or PDF files. The nodes are the main component and represent the network processors. They have a customizable appearance in color, size and are generated automatically upon execution. Edges represent the communication connections between nodes and are not generated automatically.

2.3. OMNeT++

OMNeT++ [15, 21] is a discrete event simulator. Because of its open architecture, OMNeT++ has been used not only for network simulations but also for IT systems, queuing networks, hardware architectures and business processes. It provides an extensive GUI as well as command line options. OMNeT++'s philosophy is to provide a highly distinctive separation between the components of the simulation environment, so that frameworks for a very broad spectrum of applications can be introduced. Following this, OMNeT++ is being comprised by the simulation kernel, a compiler for its own topology description language (NED) [20], GUI and command line interfaces, vector plotting and scalars visualization tools, model documentation tools and other miscellaneous utilities.

NED is OMNeT++'s topology description language which can describe parameterized regular structures, using for example multiple or conditional connections. In this way, OMNeT++ can provide structures that are not characterized by fixed interconnection or fixed number of elements. Using this property, the system can avoid executing multiple simulation

runs for similar model topologies. NED also offers the flexibility to describe the topology using parameters.

The system's vector plotting and scalars visualization tools consist of Plove[16] and Scalars[17]. Plove is a tool to plot and analyze OMNeT++ output vector files. It provides plotting specific options like drawing style and smoothing and can output in various image format files. Scalars is a tool that also uses OMNeT++ output scalar files to draw bar charts and x-y plots. It also supports output in various image format files.

2.4. DAP/ADAPT

DAP/ADAPT [5, 6, 7] stands for Advanced Distributed Algorithms Platform and Testbed. Its main goals are to provide a platform which allows the user to develop algorithms in a common and well known programming language, algorithms which can be used in both simulation and real-world deployment, to support heterogeneous simulation environments and to be open and extensible for new application domains. In ADAPT every simulated process is actually implemented as a "real", in the operating system sense, process. Each process communicates through the ADAPT *client* library to a central core, called *Engine* that manages the simulation and provides the relevant functionality.

The Engine performs the simulation processing itself. It follows the discrete event simulation model thus its central module is an event queue handler. All actions are transformed to discrete events by the system and then fed to the queue for handling. Actions can originate from the processes or the GUI. The Engine also handles metrics and statistics.

The GUI that is provided with the system allows real time monitoring of the simulations, and is a completely different executable than the Engine itself, communicating with it using TCP/IP. This allows running the simulations in powerful machines while monitoring from one or more less powerful ones. ADAPT supports the creation of customized environments and complex simulation scenarios involving dynamic events (e.g., node failures, obstruction of node movement, etc.). Scenarios definitions in the mobility, obstacles and failures domain can be defined with time-specific actions that can be replayed at any time in the exact same way. The same protocol is used for scenarios that come from the GUI or from scenario files that can be loaded at any time.

3. ARCHITECTURE

Here we describe the architecture of WSNGE in greater detail. Its main functional components are its graphical user interface, the scripting engine and its internal simulation engine. Figure 1 shows the general architecture design and the interaction of the components. The choice of the optional external simulation engine or the integrated internal simulation engine is up to the researcher, depending if one wants to use WSNGE as merely a scenario editor, online manipulation and

monitoring tool for an external simulation framework (such as *Shawn*) or as a complete simulation package.

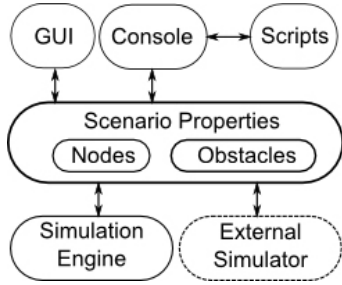


Figure 1. The High-level Architecture

In the integrated internal simulation engine, the communication between nodes takes place using data buffers. The simulation engine uses discreet time units, which we call *epochs*, to distinguish time slots where actions take place. Epochs are essentially iterations (i.e., rounds), which are completed when all active nodes in a network complete their respective actions. These actions can be queue processing for receiving data packets sent in the previous epoch, environment sensing or timed events. Within each epoch, each node processes its own data buffer queue until it is emptied. For each data packet injection, and depending on the algorithm used, none, one or more transmissions may occur from this node. The act of transmitting is equivalent to filling one slot of the destination node's packets buffer with a data packet. Apart from packet queue processing, events like sensing take place within each node's turn in the epoch. The simulator uses precalculated communication graphs, while applying real-time failures calculations, in order to speed up the simulation process.

Nodes attach themselves to predefined communication models (depending on the physical layer we wish to simulate) during the construction of the communication graph. Communication models include the Unit Disc Graph with fixed radius R or more realistic irrational graphs with the radius R varying randomly from R_1 to R_2 where $R_1 < R_2$. In the same way, each node can attach to a mobility model that enables its location to be altered each epoch following the model's behavior. While the default model is a dummy static model, other models can include random walk, circular or spiral walks or predefined paths.

3.1. Internal data abstraction

Each node is equipped with a packet buffer which can fill up with data packets that are either received by neighboring nodes or injected by the user. Each data packet is generic and can contain information about different functionality. For example, localization packets contain information about the anchor's location and localization algorithm while a geographic routing packet may contain the destination location. Packets

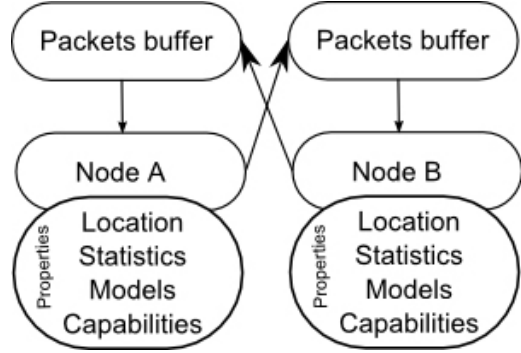


Figure 2. Two Interacting Simulated Entities

can be injected at any time using any of the available methods. Using the GUI it's also possible to view the buffer of any node at any given moment. The system does not distinguish between different scenario types while running the simulations. This means that at any time, just like in real testbeds, different kind of packets can be moving through the network, without them having to have any sort of compatibility between them, assuming that the software on the nodes knows how to handle each of them. In practice, this approach provides the researcher with a valuable tool, namely the ability to run multiple protocols on the same network at the same time, in order to directly compare their performances. For example, a number of different geographic routing protocols which share the same source and destination can be executed on the network and allowed to run step by step, allowing in turn the researcher to examine the difference in their behavior, without having to run the same simulation multiple times for each protocol. This is particularly useful, when used with the multiple views system (described in more details in the next section), for demonstration and presentation purposes.

3.2. Obstacles and faults

We feel that an important part of wireless sensor network simulation is shaping the network topology in a more natural way by introducing obstacles. Obstacles are categorized in two main categories. The first category is network areas that contain few or no nodes at all. These areas act like obstacles only when their border nodes have communication ranges that do not allow them to transmit over the gap. The second category is explicit obstacles that consist of line segments and can form various shapes like lines, triangles or circles. This type of obstacles do not allow communications to pass through them, so any part of the communication graph that crosses these lines segments is discarded. They also support mobility and toggling. Another important feature of WSNGE is the ability to fine tune the effect of faults both in communications and in other functions of the nodes, such as sensing capabilities, power leakage or total failure. This is optional,

so algorithms can be tested in an ideally functioning environment, or in a more realistic one which includes the possibility of failures. Failures are normally expressed by a probability $[0 \dots 1]$ for each action that represents the success rate. The probability can be set in the beginning to a fixed value and can change during the course of the simulation, in order to simulate effects like gradually failing nodes.

4. THE GUI AS A MONITOR AND AN ON-LINE INTERACTOR

A central goal of WSNGE is to provide the researcher a way to interact with the experiment online and directly. All operations are easy to control and initiate by simple combinations of keyboard and/or the mouse. Having this in mind, the system provides easy to use ways to manipulate the simulated environment, both during the setup and while the simulation is evolving. In fact, our system does not even distinguishes these phases.

Defining the topology of the network is simple and can be done with many different ways, depending on the situation. Individual nodes can be inserted in specific positions by using the mouse and keyboard shortcuts. If automatic deployment is what the user requires, uniform deployment of a given number of nodes in the whole area or in a specified area (again using intuitive GUI) is as easy as choosing the corresponding action from a menu.

The GUI of our system can be used in conjunction with the internal provided simulation engine or with external simulation solutions. In the later case, the GUI can be used as a powerful topology and scenario editor for simulating experiments in third party frameworks, like *Shawn* and *Algo-SenSim*. The interface can be either on-line or off-line (using exported *Shawn* scripts). The functionality that is available for each external simulation engine depends on the engine itself. Engines that do not support obstacles within their simulation, obviously will not benefit from the obstacles provided by WSNGE. A certain level of compatibility must exist between our platform and the external simulator, with specific elements being omitted when data cannot be exchanged (for example, if the external simulation does not support obstacles). The GUI provides a way to open multiple views of the running network. Each view window can provide different kind of information about the current network state, like Gabriel planarized subgraphs or connection subgraphs. While the main view is used for interaction, the other views can be used to examine the behavior of protocols or to focus on specific data.

5. SCRIPTING

WSNGE provides more than the GUI method to perform actions. Every single action can be performed in two more ways. The first way is using the GUI command line, which

can accept command by command a description of each action. For example, instead of inserting a node using the mouse, the user can insert the command `InsertAt 0.5 0.5` effectively inserting a node at the desired position. The same goes for all functionality of the GUI. Commands can be entered at any time during the simulation lifetime. The second way is using script files. These are simple text files that consist of commands and comments. The commands are executed sequentially until the end of file. An important functionality option is that a command can execute an external script file, both at the command line and within scripts, providing a way to better organize scripting. In this way, one script file can be about deployment while a second one which effectively can include the first one can be the simulation scenario description and execution. Organization is up to the researcher, so separating obstacles placement and node placement for example in different files, so that each can be reused separately, is perfectly possible. Because every aspect of the network description and scenario can be defined using script commands, exporting the situation status is being done using the scripting system itself. The system allows exporting in script files, which consist of commands that when rerun, will restore simulation state. This way, the researcher can edit these files to fine tune the scenarios, or create similar scenarios.

6. CASE STUDY

To demonstrate the capabilities of the WSNGE, we now show how easy and straightforward it is to simulate a wireless sensor network comprised of 5000 nodes. We wish to simulate an indoor network where nodes will use a localization algorithm to acquire positioning information and use a data propagation protocol that utilizes the virtual coordinates.

In order to develop a new protocol, the researcher must develop a function that implements the core functionality of the algorithm as well as its supporting data structures (e.g., node data structure) and link it to the GUI. Within the function, packet sends can take place if needed, since the function has access to the global data structures of the network. For our case study, in order to develop a new localization algorithm, the node data structure must be altered to include incoming localization packets count, and the function must describe the mechanism according to which localization packets will be generated and sent when each node becomes an anchor, or even before that (depending on the algorithm).

For the setup of the network we will use a simple script file, namely `network-simple.txt` (see listing 1). It creates a network of 5000 nodes, with size 1×1 units, using a specific random seed (making the process repeatable) and creates the communication graph for the network using a communication range of 0.1 units.

To initiate an algorithm, the researcher must inject the ap-

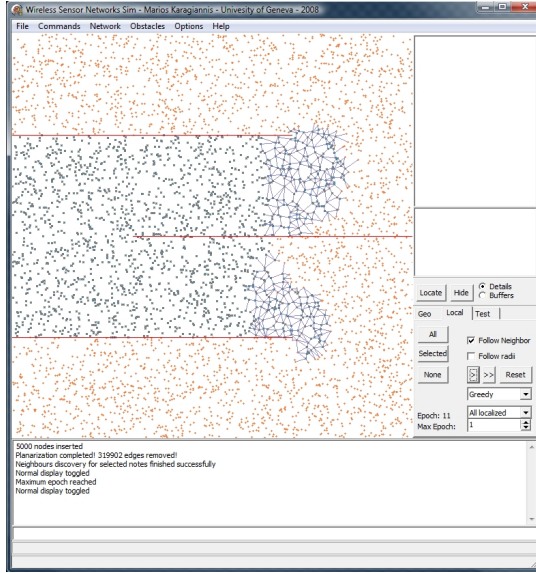


Figure 3. The Main Simulation Window

propriate packets or set up the conditions for the algorithm to begin. Based on the controls of Figure 4, in order to initiate a Greedy geographic routing algorithm, the steps that must be followed are: select the *Geo* tab, select *Greedy algorithm* from the *Geographic Routing* drop down menu, select the source node (by using the mouse to select the actual node from the network), click *source*, select the destination (again by selecting the actual node from the network), click *destination* and click *init*.

Listing 1. Script for simple network definition

```
seed 1024 // Change the random seed
insert 5000 // Position 5000 nodes uniformly
select all
radius 0.1 // Set communication range
discover // Create communication graph
```

At this moment, a greedy geographic routing packet has been injected to the source node, that contains all the necessary information to run the algorithm as well as some statistics (source node, destination location, hop count). By clicking *Next Epoch* the algorithm will forward all relevant packets and increase the epoch. By clicking *Run Until Completion*, the algorithm will run until all packets have been delivered to the destination or they fail to do so.

To run a localization algorithm, the researcher does not need to inject packets, because a newly localized node will decide, depending on the algorithm, to generate its own localization packets. The steps one must follow are: select *Local* tab, select *Greedy algorithm* from the *Localization Algorithms* drop down menu, select anchor node(s) (by selecting the actual nodes from the network), and finally click *Localize*. At this moment, by pressing *Next Epoch* or *Run Until*

Completion, the anchor nodes will generate the respective localization packets and broadcast them. Each newly localized anchor node will repeat this procedure until a set number of epochs is reached, all nodes are localized or there are no more localization packets to send.

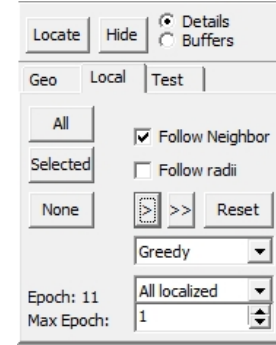


Figure 4. Controls for Controlling the Simulation

In order to insert obstacle segments by using the GUI, the researcher must follow those steps: use the menu system to select *Obstacles*, click *insert single* (for a single line segment) and drag the mouse to set the position of the obstacle. Alternatively, the researcher can use the console by inputting: *obstacle 0.5 0.5 1 1* to create an obstacle between points (0.5,0.5) and (1,1).

An alternative to setup the scenario using the GUI is to define a more detailed script file. We here use two script files, namely *network-detailed.txt* (see listing 2) and *obstacles.txt* (see listing 3) to describe some elements of the network. These configuration files insert 3 nodes at specific locations that are being localized, effectively making them the anchors for our localization algorithm testing. In the second file, 3 obstacles are introduced at specific locations. Finally, the first file resumes and creates the communication graph for the final network. The above procedure results in the network shown in Figure 3.

Listing 2. Script for detailed network definition

```
seed 1024 // Change the random seed
insert 5000 // Position 5000 nodes uniformly
insertat 0.1 0.45 // Position nodes
insertat 0.05 0.53 // at fixed points
insertat 0.15 0.53

localize 5000
localize 5001
localize 5002
select all
radius 0.1
script obstacles.txt // Obstacles script
discover // Create communication graph
```

Listing 3. Script for obstacle definition

```
obstacle 0 0.25 0.7 0.25
```

```
obstacle 1 0.5 0.3 0.5
obstacle 0 0.75 0.7 0.75
```

The above simulation was run step by step observing at each step the newly localized nodes for the purpose of this example. Alternatively, it could be run for a number of times, while providing little or no feedback. Notice that the same obstacles or network file can be reused separately, providing a tool for testing algorithms in a very controlled configuration. The above screenshot shows the network after 11 epochs running a simple greedy localization algorithm. It features two displays, one showing the full communication graph while the other shows a Gabriel Planarized subgraph.

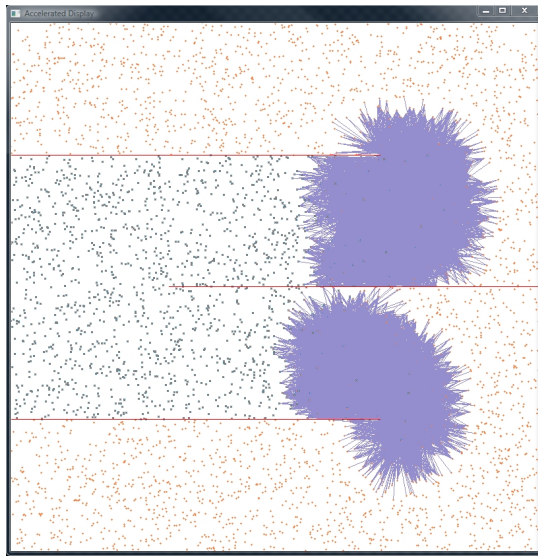


Figure 5. Auxiliary Display showing a stage of localization

For demonstration purposes (e.g., in a classroom) we can have multiple windows showing different information related to the execution of the simulation. For example, Figure 3 shows a Gabriel Planarized subgraph of the network after 11 epochs running a simple greedy localization algorithm. The full communication graph of the newly created anchors is displayed in Figure 5.

In fact, the user is not limited to controlling the simulator with one of the two methods; both can be combined. In Figure 6 we continue the scenario by introducing 4 obstacles (line segments) and extracted the Gabriel Planarized subgraph of the network to allow the use of a geographic routing algorithm that only works on planar graphs.

In Figure 7 we can view an instance of the network that runs 8 geographic routing protocols at the same time. In this case, the protocols shared the same sink node. This is a simple example of how easy it is to compare different algorithms' behaviors on the exact same network, step by step.

Regarding the ability of WSNGE to simulate large instances of wireless sensor networks, we have repeated the

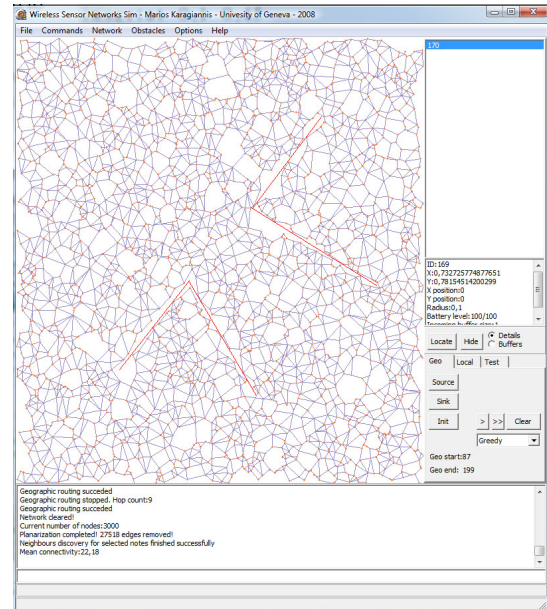


Figure 6. Gabriel subgraph of a network with obstacles

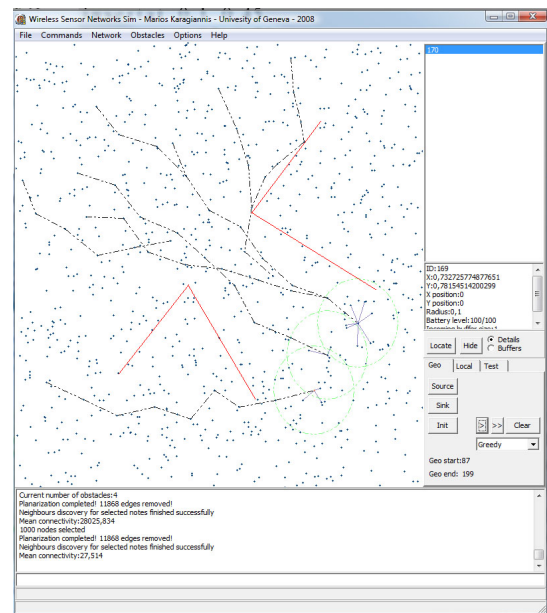


Figure 7. Executing 8 instances of the routing protocol

above experiment with larger number of nodes. In Table 2 we include the execution time and memory usage of our simulator when executed on a Quad core Q6600 (2.6 GHz) with 64KB L1 cache, 8MB L2 cache and 4 GB of memory operating Windows Vista. It is evident that even for extremely large networks ($[10^4 \dots 10^5]$) the execution time is less than a minute while the memory usage is less than 100 MB.

Size (nodes)	Memory (KB)	Time (ms)	Mean Conn.
1000	4948	514	28.844
2000	5968	1061	29.966
5000	9388	2449	30.7096
10000	14160	4712	27.5972
20000	23976	9781	24.6613
50000	57668	28096	30.38832
100000	107200	54304	25.26202

Table 2. Running times and Memory usage of WSNGE

7. CONCLUSIONS AND FUTURE WORK

We presented the design and implementation issues of a novel software platform, that especially enables the simulation of realistic application scenarios for wireless sensor networks. The primary goal of our simulator is to allow the protocol designer to interact with the simulation through a rich graphical interface. We strongly believe that this is a valuable tool that will help reduce the development cycle of applications for such networks.

Currently, we are in the process of integrating our simulator with **AlgoSenSim** [1] and **Shawn** [10]. Furthermore, we plan to refine the architecture of our framework, improve its generality and enrich the library of implemented modules with more realistic models.

REFERENCES

- [1] AlgoSenSim. URL: <http://tcs.unige.ch/doku.php/algosensim>.
- [2] The Georgia Tech Network Simulator (GTNetS). URL: <http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/>.
- [3] OPNET modeller, OPNET technologies, inc. <http://www.opnet.com>.
- [4] Ioannis Chatzigiannakis, Athanasios Kinalis, Georgios Mylonas, Sotiris Nikolettas, Grigorios Prasinos, and Christos Zaroliagis. Trails, a toolkit for efficient, realistic and evolving models of mobility, faults and obstacles in wireless networks. In *41th Annual ACM/IEEE Simulation Symposium (ANSS'08)*, pages 23–32. ACM/IEEE, SCS, April 2008.
- [5] Ioannis Chatzigiannakis, Athanasios Kinalis, A. Poulakidas, G. Prasinos, and C. Zaroliagis. DAP: A generic platform for the simulation of distributed algorithms. In *37th Annual Simulation Symposium (ANSS 2004)*, pages 166–177, 2004. IEEE Press.
- [6] Ioannis Chatzigiannakis, Christos Koninis, Grigorios Prasinos, and Christos Zaroliagis. Distributed simulation of heterogeneous systems of small programmable objects and traditional processors. In *6th ACM Workshop on Mobility Management and Wireless Access (MOBIWAC 08)*, pages 133–140. ACM, ACM, October 2008.
- [7] Distributed algorithms platform homepage. <http://ru1.cti.gr/LEP-DAP/>.
- [8] D. Estrin, M. Handley, J. Heidemann, S. McCanne, and H. Yu. Network visualization with Nam, the VINT network animator. *IEEE Computer*, 33(11):63–68, November 2000.
- [9] D. Kotz, C. Newport, R. Gray, J. Liu, Y. Yuan, and C. Elliott. Experimental evaluation of wireless simulation assumptions. In *Proceedings of the ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, 2004, pages 78–89, 2004.
- [10] Alexander Kröller, Dennis Pfisterer, Carsten Buschmann, Sándor P. Fekete, and Stefan Fischer. Shawn: A new approach to simulating wireless sensor networks. In *Design, Analysis, and Simulation of Distributed Systems (DASD05)*, pages 117–124, 2005.
- [11] S. Kurkowski, T. Camp, and M. Colagrosso. MANET simulation studies: The incredibles. *Mobile Computing and Communications Review*, 9(4):50–61, 2006.
- [12] Mathieu Lacage and Thomas R. Henderson. Yet another network simulator. In *WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator*, page 12, New York, NY, USA, 2006. ACM Press.
- [13] Koen Langendoen, Aline Baggio, and Otto Visser. Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In *20th International Parallel and Distributed Processing Symposium IPDPS*, page 8, 2006.
- [14] The network simulator – ns2. <http://www.isi.edu/nsnam/ns/>.
- [15] The OMNeT++ simulation system. <http://www.omnetpp.org/>.
- [16] Plove vector plotting tool. <http://www.omnetpp.org/external/plove.php>.
- [17] Scalars visualization tool. <http://www.omnetpp.org/external/scalars.php>.
- [18] Shawn. <http://shawn.sf.net>.
- [19] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *2nd international conference on Embedded networked sensor systems SENSYS'04*, pages 214–226, New York, NY, USA, 2004. ACM.
- [20] András Varga. Parameterized topologies for simulation programs. In *Proceedings of the Western Multiconference on Simulation (WMC'98) Communication Networks and Distributed Systems (CNDS'98)*, San Diego, CA, 1998.
- [21] András Varga. The omnet++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, June 2001.
- [22] Vis. <http://shawnwiki.coalesenses.com/index.php?title=Visualization>.
- [23] Pei Zhang, Christopher M. Sadler, Stephen A. Lyon, and Margaret Martonosi. Hardware design experiences in zebranet. In *2nd international conference on Embedded networked sensor systems SENSYS'04*, pages 227–238, New York, NY, USA, 2004. ACM.