NetTopo: Beyond Simulator and Visualizer for Wireless Sensor Networks

Lei Shu[1], Chun Wu[1], Yan Zhang[2], Jiming Chen[3], Lei Wang[4], Manfred Hauswirth[1]
Digital Enterprise Research Institute, National University of Ireland, Galway
[1]{lei.shu, chun.wu, Manfred.hauswirth}@deri.org
[2]Simula Research Laboratory, Norway email: yanzhang@ieee.org
[3]Zhejiang University, China email: jmchen@ieee.org
[4]Dalian University of Technology, China email: lei.wang@dlut.edu.cn

Abstract

Simulators are needed for testing algorithms of wireless sensor networks (WSNs) for large scale scenarios. Deploying real WSN testbed provides a more realistic testing environment, and allows users to get more accurate test results. However, deploying real testbed is highly constrained by the available budget when the test needs a large scale WSN environment. By leveraging the advantages of both simulators and real testbed, an approach that integrates simulation environment and testbed can effectively solve both scalability and accuracy issues. Hence, the simulation of virtual WSN, the visualization of real testbed, and the interaction between simulated WSN and testbed emerge as three key challenges. In this paper, we present NetTopo for providing both simulation and visualization functions to assist the investigation of algorithms in WSNs. Two case studies are described to prove the effectiveness of NetTopo.

1. Introduction

Designing and validating algorithms pertaining to wireless sensor networks (WSNs) are among the most fundamental focuses of researchers. Simulators are widely used for the purpose of analysis in these tasks due to the fast prototyping and tackling large scale systems. However, even the best simulator still cannot simulate real wireless communication environment in terms of completeness and accuracy [1]. Taking this drawback of simulators into account, using real testbed to evaluate algorithms of WSNs is essentially necessary before applying them into commercial applications.

Using testbeds allows rigorous and replicable testing. However, there are two serious limitations on this approach in the following two conditions: 1) Large scale. Until today, it is still very expensive to buy a large number of sensor nodes for a large scale testbed. Especially, for most academic researches the cost for building a large scale testbed is not acceptable. 2) Not replicable environment. For some specific applications, e.g., monitoring an erupting volcano [1], deploying a testbed is unwanted since the devices are exposed to dangerous conditions which can cause serious damage.

Due to the complementary properties of simulators and testbeds, a better solution can be the integration of simulation environment and physical testbed. Having this integrated framework, applications can run partially in a simulation environment and partially in a physical WSN testbed which can solve both scalability and accuracy issues for the evaluation of algorithms in WSNs. This integration is specially motivated by the following two concrete scenarios:

- Researchers want to compare the performance of running a same algorithm in both simulator and real testbed. The comparison can guide researchers to improve the algorithm design and incorporate more realistic conditions. A good example is the applying of face routing algorithm in GPSR [2], which is proved to be loop free in theory but actually is not loop free in realist situations, due to the irregular radio coverage [3].
- A budget limitation prevents researchers from buying enough real sensor nodes but the research work has to base on a large scale WSN. For example, to evaluate the performance of sensor middleware [4], a large scale sensor network is needed. Researchers can actually do the research work by integrating a small number of real sensor nodes and a large number of virtual sensor nodes generated from the simulator.

The integration of simulation environment and physical testbed brings three major challenges:

- *Sensor node simulation*. Normally, a number of heterogeneous sensor devices can be used for building a WSN testbed. The integrated platform should not simulate only a specific sensor device, which means that the heterogeneous problem requires the integrated platform to be flexible enough to simulate any new sensor device.

- *Testbed visualization*. Sensor nodes are small in size and do not have user interfaces as displays or keyboards, which is difficult to track the testbed communication status. On the other hand, the communication topology in testbed is invisible, but researchers usually need to see the topology to analyze their algorithms. For example, when implementing a routing algorithm in the testbed, the actual routing path is expected to be visible.
- *Interaction between the simulated WSN and testbed*. The simulated WSN and the real testbed need to exchange information, e.g., routing packet. Their horizontal interconnection, communication, interaction, and collaboration are all emerging difficult problems that need to be addressed.

In this paper, we present an extensible integrated framework of simulation and visualization called NetTopo to assist investigation of algorithms in WSNs. With respect to the simulation module, users can easily define a large number of on-demand initial parameters for sensor nodes, e.g. residential energy, transmission bandwidth, radio radius, etc. Users also can define and extend the internal processing behavior of sensor nodes, such as energy consumption, bandwidth management. It allows users to simulate an extremely large scale heterogeneous WSN. For the visualization module, it works as a plug-in component to visualize testbed's connection status, topology, sensed data, etc. These two modules paint the virtual sensor nodes and links on the same canvas which is an integration point for centralized visualization. Since the node attributes and internal operations are user definable, it guarantees the simulated virtual nodes to have the same properties with those of real nodes. The sensed data captured from the real sensor nodes can drive the simulation in a pre-deployed virtual WSN. Topology layouts and algorithms of virtual WSN are customizable and work as user defined plug-ins, both of which can easily match the corresponding topology and algorithms of real WSN testbed. As a major contribution of this research work, NetTopo is released as open source software on the SourceForge. Currently, it has more than eighty java classes and 11,000 Java lines source codes. Users can freely download the latest version of NetTopo by accessing the NetTopo website [5].

The rest of the paper is: Section 2 positions our work with respect to the related work. Section 3 illustrates NetTopo architecture and Section 4 describes features of NetTopo. Section 5 presents two case studies provided in NetTopo as examples. Section 6 concludes this paper and describes the future work.

2. Related work

A large number of WSN simulators have been proposed by researchers till today. These simulators can be classified into three major categories:

- *Algorithm level*. Simulators [6-8] focus on the logic, data structure and presentation of the algorithms. AlgoSensim [6] analyzes specific algorithms in WSNs, e.g. localization, distributed routing, flooding, etc. Shawn [7] is targeted to simulate the effect caused by a phenomenon, improve scalability and support free choice of the implementation model. Sinalgo [8] offers a message passing view of the network, which captures well the view of actual network devices.
- *Packet level*. Simulators [9-12] implement the data link and physical layers in a typical OSI network stack. The ns-2 [9] is not originally targeted to WSNs but IP networks. SensorSim [10] is an extension to ns-2 which provides battery, radio propagation and sensor channel models. J-Sim [11] adopts loosely-coupled, component-based programming model, and it supports real-time process-driven simulation. GloMoSim [12] is designed for the parallel discrete event simulation capability provided by PARSEC.
- *Instruction level*. Simulators [13-15] model the CPU execution at the level of instructions or even cycles. They are often regarded as emulators. TOSSIM [13] simulates the TinyOS network stack at the bit level. Atemu [14] is an emulator that can run nodes with distinct applications at the same time. Avrova [15] is a Java-based emulator used for programs written for the AVR microcontroller produced by Atmel.

It is clear that none of these simulators has considered integrating with real WSN testbed. This point clearly distinguishes NetTopo from them.

In terms of visualization of real WSN testbed, there is much less related work. Octopus [16] is an open-source visualization and control tool for WSNs in TinyOS 2.x environment. It provides a graphical user interface for viewing the live WSN topology and allows users to control the behavior of one or many sensor nodes. The Surge [17] and the Mote-VIEW [18] are products of Crossbow company to visualize WSNs. They are capable of logging wireless sensor data to a database and to analyze and plot sensor readings. They are designed to support only Crossbow sensor nodes, thus they are not extensible. SpyGlass [19] visualizes WSN using a flexible multi-layer mechanism that renders the information on a canvas. TinyViz [13] is a

GUI tool of TOSSIM package of TinyOS. It visualizes sensor readings, LED states and radio links and allows direct interaction with running TOSSIM simulations. But these interactions are often ad-hoc as well as laborious and difficult to reproduce.

In short, most of existing visualization tools support only a single type of WSN and are highly coupled to the TinyOS. However, NetTopo is targeting at the visualization and control of WSN testbed where heterogeneous devices are used, e.g., wireless camera, Bluetooth based body monitoring sensor devices, and these devices are generally not TinyOS based.

3. NetTopo architecture

3.1. Modular components

From the high-level point of view, NetTopo consists of both simulation and visualization functions. These two functions need to interact with each other and access/manipulate some common resources. For focusing on the integration issues of them, we use component based NetTopo architecture, which is flexible enough for adding new components in the future. The basic architecture is illustrated in Figure 1.

Main Control and *Utility* are two components involved in all layers. *Main Control* is the core component working as a coordinator in charge of the interactions of other components. It can be regarded as an adaptor between input and output interfaces of other components and enables them to work smoothly. *Utility* provides some basic services, e.g., defined application exceptions, format verification, number transforms, dialogue wrappers.

File Manager is for the purpose of data persistence, e.g. logging runtime information, recording statistical results, keeping references of virtual sensor nodes, etc. Log information and statistical results are recorded as character streams into human readable format. References of virtual sensor nodes are stored as serialized format for easy recovery and reuse. All these references are encapsulated in *Virtual WSN*, which works like a runtime sensor nodes repository and also declares interface to allow other components to add new virtual nodes, delete particular nodes, retrieve the same type of nodes and their derived children, etc.

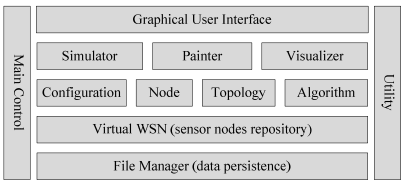
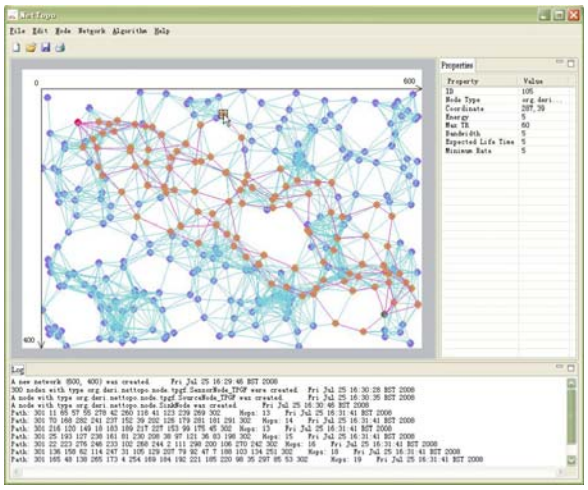


Figure 1. NetTopo Architecture

Node, *Topology* and *Algorithm* components are designed as highly extensible modules that can be regarded as plug-ins. *Node* represents a virtual sensor node. Virtual sensor nodes do not have fixed properties or structures. For example, sensor nodes can have very different sensing attributes: temperature, humidity, vibration, pressure, etc. To allow users to create their own virtual sensor nodes, an abstract interface named *VNode* is declared to define several basic methods representing actions of a real sensor node. Any user desired node that wishes to run on the simulator must implement the *VNode* interface. *Topology* stands for the topology to be deployed in *Virtual WSN*. Network topology can be various shapes, e.g., line, circle, triangle, tree. Users can flexibly implement any needed network topology. *Algorithm* represents an algorithm to be applied in the *Virtual WSN*. The algorithm can be any routing, clustering, scheduling, controlling algorithm, etc. Users can freely implement their needed algorithms for their specific studies.

The graphical user interface (*GUI*) in Figure 2 consists of three major components: a display canvas (on the upper left), which can be dragged in case of viewing a large scale WSN, a property tab for displaying node properties (on the upper right), and a display console for logging and debugging information. *Painter* is separated from the main *GUI* due to the frequent paining tasks. The painter is also designed as an abstract interface for various painting requirements, e.g., 2D or 3D. The specific painter used in Figure 2 is Painter_2D. Additionally, the painter encapsulates the lower painting API, interacts with the *Virtual WSN* and main *GUI* and provides advanced painting methods, e.g. it can paint a link between any two nodes by just using their ID information.



Property	Value
ID	105
Node Type	org.deri...
Coordinate	207, 39
Energy	5
Max TR	60
Bandwidth	5
Expected Life Time	5
Minimum Rate	5

```
A new network (600, 400) was created      Fri Jul 25 16:29:46 BST 2008
300 nodes with type org.deri.nettopo.node.tpgf.SensorNode_TPGF were created   Fri Jul 25 16:30:28 BST 2008
A node with type org.deri.nettopo.node.tpgf.SourceNode_TPGF was created      Fri Jul 25 16:30:35 BST 2008
A node with type org.deri.nettopo.node.SinkNode was created       Fri Jul 25 16:30:46 BST 2008
Path: 301 11 65 57 55 278 42 260 118 41 123 239 269 302     Hops: 13    Fri Jul 25 16:31:41 BST 2008
Path: 301 70 168 282 241 237 152 39 202 126 179 281 181 291 302   Hops: 14    Fri Jul 25 16:31:41 BST 2008
Path: 301 216 120 149 18 183 189 217 227 153 99 175 45 302  Hops: 13    Fri Jul 25 16:31:41 BST 2008
Path: 301 25 193 127 238 161 81 230 208 38 97 121 36 83 198 302   Hops: 15    Fri Jul 25 16:31:41 BST 2008
Path: 301 22 223 276 246 233 102 268 244 2 111 298 200 106 270 242 302  Hops: 16    Fri Jul 25 16:31:41 BST 2008
Path: 301 136 158 62 114 247 31 105 129 207 79 92 47 7 188 103 134 251 302     Hops: 18    Fri Jul 25 16:31:41 BST 2008
Path: 301 165 48 138 265 173 4 254 169 184 192 221 185 220 98 35 297 85 53 302      Hops: 19    Fri Jul 25 16:31:41 BST 2008
```

Figure 2. NetTopo main GUI (the TPGF [20] multipath routing algorithm is executed in the WSN)

Simulator and *Visualizer* represent the high level functions in NetTopo. The structure difference between these two components is that *simulator* is a built-in of NetTopo but *visualizer* is loaded as a plug-in. This is because different accessing interfaces (wrappers) are needed for different devices, e.g. the HTTP based connection is used for getting image streams from wireless camera and the socket connection is used for getting Crossbow sensor data. The common components they all utilize include *Virtual WSN*, *Painter*, *Node*, *Configuration* and *GUI*. Using these shared resources sometimes can cause synchronization problem, e.g. when both *Simulator* and *Visualizer* components need to add new sensor nodes in the graphical display canvas.

3.2. Interaction of components

In NetTopo, components interactively communicate with each other to achieve the functions of simulation and visualization. At the beginning, virtual nodes should be deployed and their attributes should be configured before the simulation starts. The *Algorithm* component loaded as plug-in decides how virtual sensor nodes will communicate and forward packets. Users' command drives the simulation which runs based on the specification of the loaded algorithm.

Figure 3 shows the interaction between the user and related components in a simulation scenario. *GUI* invokes the simulation interface provided by *Main Control* when receive the simulation request. *Main Control* directly forwards the task to *Algorithm* and wait for the result. Then, *Algorithm* searches the *Virtual WSN* and gets the references of starting virtual sensor nodes. In *Virtual WSN*, the nodes cooperatively behave according to the specified algorithm and return the results, e.g. searched routing paths. *Main Control* notifies the *Painter* to paint the results on *GUI* after receiving the results from the *Algorithm*.

The *Visualizer* works as a thread that shares some common components with *Simulator* including *GUI*, *Main Control*, *Painter* and *Configuration*. Sometimes, *Virtual WSN*, *Node* and *File Manager* can also be involved. This depends on the implementations of *Visualizer* for different real sensor devices.

Figure 4 shows the components interaction in a visualization scenario. Once the *Visualizer* thread is created, it runs concurrently with the *Simulator* thread. It then works in a loop to update the testbed information on *GUI* e.g. logging and painting added sensor nodes and connections, refreshing sensed data of each node, etc. until users manually interrupt this thread. This simple scenario only focuses on the

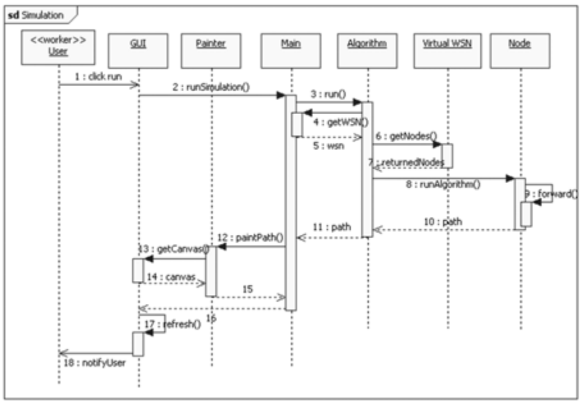


Figure 3. Component interaction in simulation

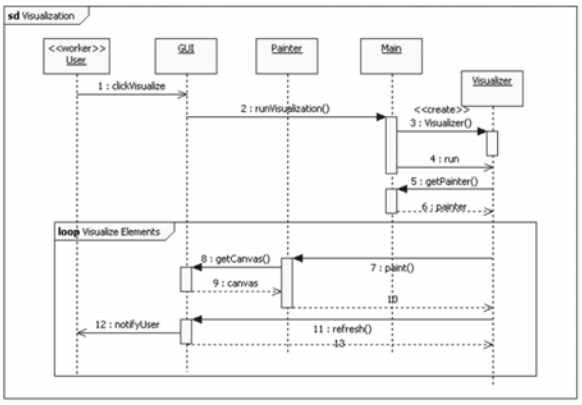


Figure 4. Component interaction in visualization

visualization of testbed, in which visualization and simulation components are running concurrently, but they actually do not interact with each other because no common virtual sensor nodes are used by both components. A further example of interaction between both components is presented in case study section.

4. Features of NetTopo

Features of NetTopo in current version can be classified in the following four categories.

1) Platform independent.

- NetTopo is implemented in Java language, which makes it portable between different operating systems.

2) Extensibility.

- *Configurable sensor nodes with defined attributes*. Users can define their own virtual sensor nodes with expected attributes. New type of nodes will be loaded as a plug-in, which provides an extra choice when users plan to deploy a WSN.
- *Customizable sensor network topology layout*. Users can define their own topology based on the

API described in the *Topology* component. This is helpful when users focus on studying a particular topology of the network.

- *User-defined algorithms and functions*. An algorithm can be composed of several functions, each of which acts for a particular purpose. User can debug a single function or add a new function without influencing others in the same algorithm.
- *Device based wrappers for visualization*. A wrapper is used to get information from sensor device. To visualize different hardware devices, users can create different wrappers to set up the connection for extracting information.
- *Integrating with GSN middleware*. GSN [3] is a sensor network middleware developed by us. It provides a large number of wrappers (currently more than 25 wrappers) for extracting data from heterogeneous sensor devices. This can help to reduce the workload to implement new wrappers for some GSN supported sensor devices.

3) Flexibility.

- *Single node deployment*. Users can deploy a single node in a given location. This is useful for a slight modification to the *virtual WSN* or placement for a sink node or source node.
- *Single node movement*. Users can move any node to any place in the WSN field (graphical display canvas) after deploying the sensor nodes. This is useful for updating some specific sensor node's location, e.g. move the sink node for several tests.
- *Random multi-node deployment*. Users can randomly deploy a specified number of sensor nodes. The random seed can be the irreproducible current time point of the running computer or any specified reproducible integer.
- *Specific multi-node deployment*. Users can deploy a specified number of sensor nodes based on pre-defined topologies to form some special shapes, e.g., users can deploy a circle by specifying the location of circle center, radius, and node number.
- *Repeated node deployment*. Users can repeatedly deploy different kind of sensor nodes in the *Virtual WSN*. This allows the deployment of heterogeneous sensor networks.

4) Practicability.

- *Data persistence for virtual WSN*. The network deployment state can be saved in a specific type of file using ".wsn" as the postfix. Users can base on these files to reuse the deployed *virtual WSN* or share these files with friends to discuss a common problem.
- *Snapshot for virtual WSN*. Users can capture a snapshot for a *virtual WSN* and save it as ".bmp" picture. This feature allows users to further analyze the simulation results and use the saved picture for sharing or writing papers.
- *Node manipulation*. Users can delete specified nodes, view the current properties of the nodes, modify the property values of a node before starting a simulation, search a node by its ID and disable nodes or kill nodes in a specified region to make a hole in the WSN or make an irregular WSN field.
- *Recording of simulation results*. NetTopo can save the simulation results in a specific type of file using ".report" as the postfix. Users can use normal text editor software to open it and read the simulation result. The simulation results are formulated into a unified format that allows users to further import them into Microsoft Office Excel to get the graphical results, e.g., curves and charts.

5. Case studies

To demonstrate the usability of NetTopo we present two case studies on simulation and visualization respectively as user examples. For simulation, two routing algorithms, GPSR [2] and TPGF [20], are implemented and compared based on the statistical results.

For visualization, a testbed composed of Crossbow Mica2 sensor nodes is visualized. Additionally, these real sensor nodes are considered as source nodes in a pre-deployed *virtual WSN*: when the sensed temperature value of any real node exceeds a threshold, which means an event is detected, it then automatically starts a simulation for exploring one/multiple routing paths in the integrated *virtual WSN*.

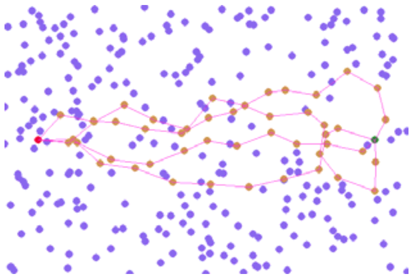
5.1. Simulation of two routing algorithms

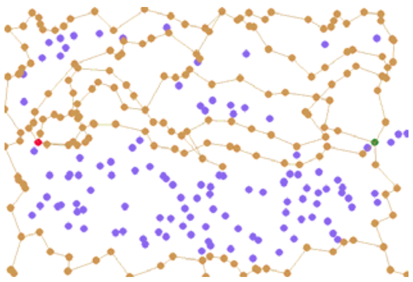
Users who do simulation of testing an algorithm not only expect to see visual results on the canvas but also need to gather related statistic information that can be used to analyze the algorithm performance. For example, users want to know how many paths can be searched by repeatedly using a same algorithm in the WSN and how many hops each path has.

However, providing such information of a single test on a specific WSN deployment is not enough, because to evaluate the algorithm performance, users generally need to simulate the same algorithm for many times while changing several input parameters to get the more convincible average results.

Following the above example, users also want to know the average paths number by applying the same algorithm in 100 runs with different random network deployment. And users even want to know the variation of the paths number along with the variation of value of input parameters such as network size, node number, and transmission radius. NetTopo provides an easy way for users to configure their input parameters for the purpose of simulating the same algorithm for many times.

Two routing algorithms TPGF and GPSR are implemented in NetTopo as examples. When applying them respectively in the network layer of WSN, different performance can be compared in various aspects. The major concentrated measurement metrics include: 1) the average number of paths by repeatedly using this same algorithm in the WSN; 2) the average path length from the source node to the sink node.


(a) Running TPGF in the *virtual WSN* with 4 routing paths when TR is set as 60 meters


(b) Running GPSR in the GG *virtual WSN* with 4 routing paths when TR is set as 60 meters

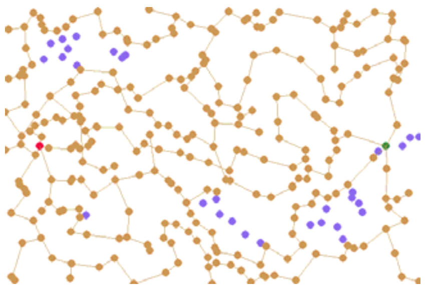
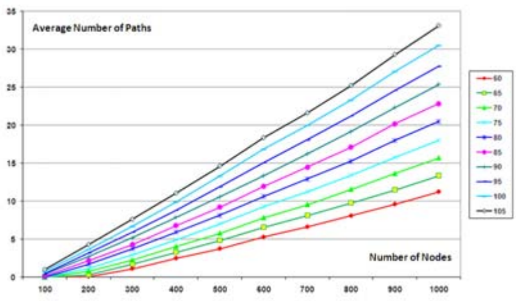
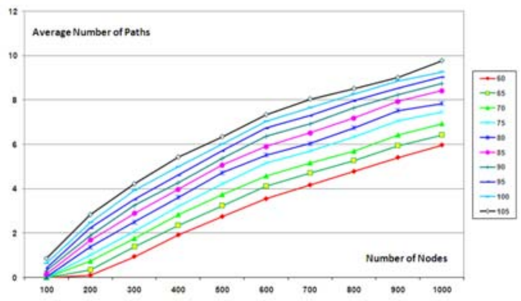

(c) Running GPSR in the RNG *virtual WSN* with 4 routing paths when TR is set as 60 meters

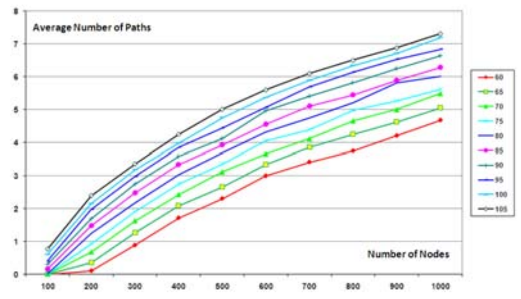
Figure 5. An example of the simulation of TPGF and GPSR



(a) TPGF: average number of paths vs. number of nodes



(b) GPSR on GG *virtual WSN*: average number of paths vs. number of nodes

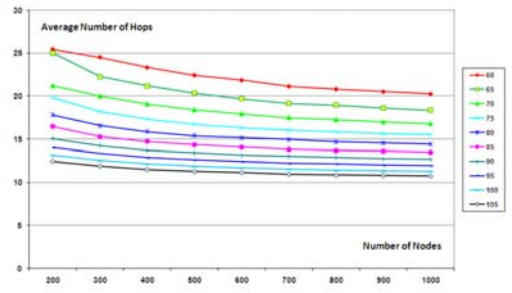


(c) GPSR on RNG *virtual WSN*: average number of paths vs. number of nodes

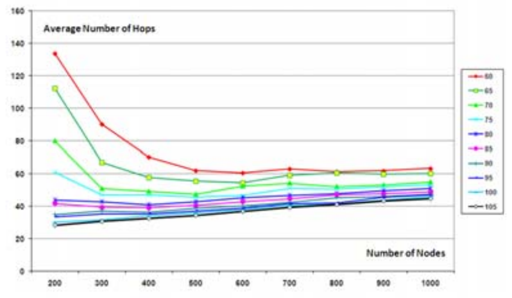
Figure 6. Average number of paths vs. number of nodes

As an example, Figure 5 shows the visual results on the canvas when running both TPGF and GPSR in the *virtual WSN*. In Figure 5, the red color node is the source node and the green color node is the sink node. Pictures (a), (b) and (c) give a direct impression to researchers that TPGF can have shorter average path length than that of GPSR in a single WSN deployment. However, having a single test result is not convincible; we need to do the test for many times. We would like to know the variation of these two metrics in the case of different conditions in terms of network density and transmission radius of sensor nodes.

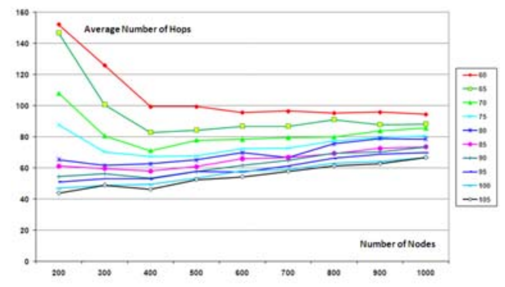
In order to simplify this case study, the network size is fixed in 600 × 400 (1 pixel on the canvas is considered as 1 meter). For each fixed number of sensor nodes (network density) and transmission radius (network degree), the average number of paths and the average path length are computed from 100 simulation results using 100 different random seeds for network deployment. Then, we change the node number (from 100 to 1000) and transmission radius (from 60 to 105) to obtain different values. By gathering all these average values together, lots of chart and figures can be drawn to reflect the execution performance of the algorithms. Figures 6 (a), (b) and (c) are the simulation results on the average number of paths that found by applying TPGF and GPSR respectively.



(a) TPGF: average path length vs. number of nodes



(b) GPSR on GG *virtual WSN*: average path length vs. number of nodes



(c) GPSR on RNG *virtual WSN*: average path length vs. number of nodes

Figure 7. Average path length vs. number of nodes

Figures 7 (a), (b) and (c) are the simulation results on the average path length that found by applying TPGF and GPSR respectively.

It is more convincible to use the statistical simulation results to reflect the impact on the execution performance of both GPSR and TPGF routing algorithms when using different transmission radius and number of sensor nodes for simulation.

5.2. Crossbow WSN testbed visualization

Crossbow WSN testbed consists of six Mica2 nodes. Figure 8 shows the whole network structure and flow of sensed data. The Crossbow driver called xServe is installed in gateway for converting sensed data into XML stream and providing a TCP/IP service on port 9005. NetTopo can be located on gateway or another computer that can communicate with the gateway.

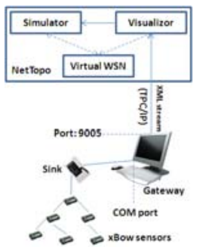
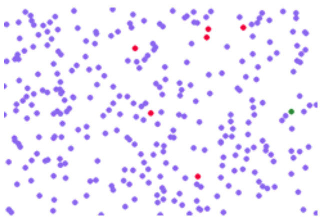


Figure 8. Crossbow WSN testbed visualization flow

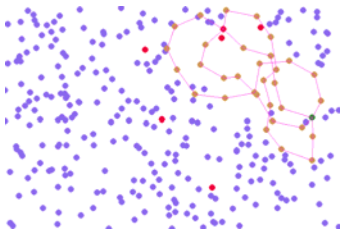
The sink node collects packets sent from sensor nodes. Each packet of any node includes lots of properties, e.g., its node ID and its parent node ID. By using a wrapper to set up a TCP/IP connection on port 9005, NetTopo can read the XML stream from the gateway, extract the node ID information and draw some round circles representing virtual sensor nodes on the canvas.

In addition to using *Painter* to update the GUI, this particular *Visualizer* component also creates virtual sensor nodes in the *virtual WSN*, which allows the references of these nodes to be obtained by *Simulator* for using in the simulation. Consequently, by getting the node ID and parent ID mapping information in the XML packet, NetTopo can easily draw the topology of the network connection. Nodes' latest properties and sensed data, e.g., voltage, temperature, humid, pressure. can also be periodically captured from the XML stream by setting a specific sampling rate. The values of all these properties are presented on the property tab of the main GUI and refreshed when new data arrive.

Furthermore, these six virtualized Mica2 nodes are considered as source nodes in the *virtual WSN*. When the temperature reading of any Mica2 node exceeds a threshold, the *Simulator* is involved to explore multiple routing in a pre-deployed *virtual WSN*, which include many other deployed simulated virtual sensor nodes.



(a) Six Crossbow nodes are virtualized as source nodes in the *virtual WSN*



(b) One Crossbow node explored 4 routing paths by using TPGF

Figure 9. An example of the integration of the testbed and the simulation environment

Figure 9 (a) shows the visualization of the six Crossbow nodes in the pre-deployed *virtual WSN*. Figure 9 (b) shows that one of the Crossbow nodes explored four routing paths by using TPGF.

6. Conclusion

In this paper, we present NetTopo, an integrated framework of simulation and visualization for WSNs. The friendly GUI makes it easy to use and the modular components enable it to be flexibly extended. NetTopo can support an extremely large scale network simulation by integrating simulated sensor networks and visualized testbed. It is very useful for a fast rapid prototyping of an algorithm.

All in all, currently NetTopo gets the first step into the whole vision where network simulators, visualizers and real physical testbeds are expected to be integrated to test and validate algorithms in WSNs.

7. Acknowledgement

The work presented in this paper was supported by the Lion project supported by Science Foundation Ireland under grant no. SFI/02/CE1/I131.

8. Reference

[1] C. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, J. Lees, “Deploying a Wireless Sensor Networks on an Active Volcano”, in *IEEE Internet Computing*, Vol. 10, No. 2, pp. 18-25, 2006.

[2] B. Karp, H.T. Kung, “GPSR: greedy perimeter stateless routing for wireless networks”, in Proceedings of the Annual International Conference on Mobile Computing and Networking (MobiCom 2000), Boston, USA, August.

[3] K. Seada, A. Helmy, R. Govindan, “Modeling and Analyzing the Correctness of Geographic Face Routing Under Realistic Conditions. Ad Hoc Networks”, doi:10.1016/j.adhoc.2007.02.008, pp. 855-871, 2007.

[4] K. Aberer, M. Hauswirth, A. Salehi, “Infrastructure for data processing in large-scale interconnected sensor network”, in *8th International Conference on Mobile Data Management* Mannheim, Germany, 2007.

[5] http://lei.shu.deri.googlepages.com/**nettopo**

[6] http://tcs.unige.ch/doku.php/code/**algosensim**/overview

[7] http://**shawn**.sourceforge.net/

[8] http://dcg.ethz.ch/projects/**sinalgo**/

[9] http://www.isi.edu/nsnam/**ns**/

[10] http://nesl.ee.ucla.edu/projects/**sensorsim**/

[11] http://www.**j-sim**.org/

[12] http://pcl.cs.ucla.edu/projects/**glomosim**/

[13] http://www.cs.berkeley.edu/~pal/research/**tossim**.html

[14] http://www.hynet.umd.edu/research/**atemu**/

[15] http://compilers.cs.ucla.edu/**avrora**/

[16] http://csserver.ucd.ie/~rjurdak/**Octopus**.htm

[17] http://www.cmt-gmbh.de/**surge**_network_viewer.htm

[18] http://www.**xbow**.com/Products/

[19] C. Buschmann, D. Pfisterer, S. Fischer, S.P. Fekete, A. Kröller, “**SpyGlass**: A Wireless Sensor Network Visualizer”, in ACM SIGBED Review. Vol. 2, No. 1, 2005.

[20] L. Shu, Z. Zhou, M. Hauswirth, D. Phuoc, P. Yu, L. Zhang, "Transmitting Streaming Data in Wireless Multimedia Sensor Networks with Holes", in Proceedings of the Sixth International Conference on Mobile and Ubiquitous Multimedia (MUM 2007), December 12-14, 2007. Oulu, Finland.