NetTopo: A framework of simulation and visualization for wireless sensor networks

Lei Shu a,⇑, Manfred Hauswirth b, Han-Chieh Chao c, Min Chen d, Yan Zhang e,f

a Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University, Japan
b Digital Enterprise Research Institute, National University of Ireland, Galway, Ireland
c Department of Electronic Engineering, National Ilan University, Taiwan
d School of Computer Science and Engineering, Seoul National University, Republic of Korea
e Simula Research Laboratory, Norway
f Department of Informatics, University of Oslo, Norway

A R T I C L E   I N F O

Article history:
Received 9 November 2009
Received in revised form 28 May 2010
Accepted 7 September 2010
Available online 27 September 2010

Keywords:
NetTopo
Research framework
Wireless testbed
Simulation
Visualization
Wireless sensor networks

A B S T R A C T

Network simulators are necessary for testing algorithms of large scale wireless sensor networks (WSNs), but lack the accuracy of real-world deployments. Deploying real WSN testbed provides a more realistic test 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 network simulator 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 an integrated framework called NetTopo for providing both simulation and visualization functions to assist the investigation of algorithms in WSNs. NetTopo provides a common virtual WSN for the purpose of interaction between sensor devices and simulated virtual nodes. Two case studies are described to prove the effectiveness of NetTopo.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

The increasing market requirements of wireless sensor networks (WSNs) applications drive the fast development of research in various aspects, e.g., sensor hardware design, sensor operating system development. Among all kinds of research issues in WSNs, various algorithms, e.g., routing algorithm, topology control algorithm, are highly concentrated. Designing and validating algorithms pertaining to WSNs are among the most fundamental focuses of researchers. Network simulators are widely used for the purpose of analysis in these tasks due to the fast prototyping and the capability of tackling large scale systems. However, even the best simulators are still not able to provide real condition simulation environments in terms of completeness, complexity and accuracy, e.g., theoretical model based simulation may trade accuracy for simulation performance [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. Testbeds allow for rigorous and replicable testing. Nevertheless, there are two serious limitations on this approach in the following two conditions: (1) Large scale. Till today, it is still very expensive to buy a large number of sensor devices for a large scale testbed, e.g., the price of a Crossbow professional kit with eight sensor nodes is around 2800 Euros. Especially, for most academic researches the cost for building a large scale

* Corresponding author. Address: Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University, Japan. Tel.: +81 6 6879 4513; fax: +81 6 6879 4514.
E-mail addresses: lei.shu@ieee.org (L. Shu), manfred.hauswirth@deri.org (M. Hauswirth), hcc@niu.edu.tw (H.-C. Chao), minchen@ieee.org (M. Chen), yanzhang@ieee.org (Y. Zhang).

1570-8705/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2010.09.003
testbed is not acceptable. (2) Not replicable environment. For some specific applications, e.g., monitoring an erupting volcano [2], rescue in a sudden earthquake and monitoring hazardous situations [3], deploying a testbed is unwanted since the devices are exposed to hazardous situations which can cause serious damage.

Even though simulation models are usually not able to represent the real environments with the levels of completeness and accuracy, simulators are still very important tools, because they enable rapid prototyping and early evaluation of sensor network applications and algorithms. Compared to the cost and time involved in setting up an entire testbed containing multiple networked computers, routers and data links, network simulators are relatively fast and inexpensive. Due to the complementary properties of simulators and testbeds, a better solution could be the integration of simulation environment and physical testbed. Using this technology, applications can run partially in a simulation environment and partially in a physical WSN testbed and interact with each other to create an environment where scalability issues, heterogeneous environments, etc. can be better studied.

Thus, the subject of this research work is about building a new software framework, which integrates simulation and visualization functions to assist the investigation of various 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 [4], which is proved to be loop free in theory but actually is not loop free in realistic situations, due to the irregular radio coverage [5].
- 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 any sensor middleware, e.g., [6], 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. On one hand, 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. Users also can define and extend the internal processing behavior of sensor nodes, e.g., energy consumption, bandwidth management. NetTopo 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 node attributes and internal operations are user definable, it guarantees simulated virtual nodes to have the same properties with those of real nodes. The sensed data captured from 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. Currently, NetTopo is released as open source software on SourceForge and its own website [7].

The rest of the paper is organized as follows: Section 2 presents an overview of the related work in terms of simulators and WSN visualization and positions our work with respect to the related literature. Section 3 illustrates design issues of the framework including architecture, mechanism of modular components, module interfaces and interaction between these interfaces. Section 4 describes the features of NetTopo and analyzes the implementation of the framework. It is focused on logic, programming techniques and interesting code of some core features. Section 5 presents two case studies provided in NetTopo as examples for testing the effectiveness of the framework. Section 6 concludes this paper.

2. Related work

Since NetTopo is an integrated framework of simulation and visualization for WSNs, network simulators and WSN visualization are two main aspects of related work. In this section, we have an overview of literature on both of them respectively.

2.1. Related work on WSNs simulators

Till today, a large number of WSN simulators have been proposed by researchers. These simulators can be classified
into three major categories based on the level of complexity:

- **Algorithm level.** Simulators [8–10] focus on the logic, data structure and presentation of algorithms. These simulators do not consider detailed communication models and, most commonly, they rely on some form of a graph data structure to illustrate the communication between nodes. AlgoSensim [8] analyzes specific algorithms in WSNs, e.g., localization, distributed routing, flooding. Shawn [9] is targeted to simulate the effect caused by a phenomenon, improve scalability and support free choice of the implementation model. Sinalgo [10] offers a message passing view of the network, which captures well the view of actual network devices. It was designed, but is not limited to simulate wireless networks. These approaches do not reproduce the ISO network stack and with no or very simplistic MAC layer protocol, they can simulate extremely large networks.

- **Packet level.** Simulators [11–14] implement the data link and physical layers in a typical OSI network stack. Hence, it is common for these types of simulators to find implementations of 802.11b or newer MAC protocols and radio models that account for propagation, fading, collision, and noise & wave diffraction. The most popular and widely used simulator is ns-2 [11], which is not originally targeted to WSNs but IP networks. SensorSim [12] is an extension to ns-2. It provides battery, radio propagation and sensor channel models. J-Sim [13] adopts loosely-coupled, component-based programming model, and it supports real-time process-driven simulation. GloMoSim [14] is designed for the parallel discrete event simulation capability provided by PARSEC. Compared to above open source simulators, some commercialized network simulation software (e.g. OPNET [35] and Qualnet [36]) provide good simulation environment with powerful standard modules. With the implementation of Zigbee protocol and IEEE 802.15.4 MAC protocol in its 14.0 version, OPNET becomes an excellent choice to simulate Zigbee based body sensor networks (BSN) or wireless body area sensor networks (WBASN). However, OPNET simulation requires a relatively long time when the number of sensor nodes is large (e.g., 5000 nodes). By comparison, Qualnet exhibits good performance in simulating large scale sensor network due to its scalability in wireless simulation.

- **Instruction level.** Simulators [15–17] model the CPU execution at the level of instructions or even cycles. They are often regarded as emulators. TOSSIM [15] is the most representative but not most accurate of the existing emulators. TOSSIM simulates the TinyOS network stack at the bit level. The communication between nodes is modelled through a probabilistic graph that, as side-effect, may reduce the overall accuracy of this tool. Atemu [16] is an emulator that can run nodes with distinct applications at the same time. For this reason, Atemu is more accurate but its scalability is limited. Avrova [17] is a Java-based emulator used for programs written for the AVR microcontroller produced by Atmel and the Mica2 sensor nodes. Being written in Java, it also benefits from the added portability across various computing platforms.

All these simulators make great contributions to the development of network simulation. They have their own advantages and limitations. Based on the above classification, the simulation framework in our NetTopo is by far algorithm oriented. Additionally, it is clear that none of these simulators has considered integrating with real WSN testbed, except one research on sQualNet [36], which is an extension of Qualnet and motivated by faithfully using real testbed to simulate the network models, e.g., energy consumption. But sQualNet did not take the real testbed visualization into consideration. This point clearly distinguishes NetTopo from them.

### 2.2. Related work on WSNs visualization

With respect to visualization of real WSN testbed, there is much less related work. Octopus [18] is an open-source dashboard to visualize and to control a WSN in the TinyOS 2.x environment. Octopus provides users with a graphical user interface (GUI) for viewing the live sensor network topology. It also allows the user to dynamically control the behavior of sensor nodes, e.g., the energy consumption, the sampling period, the radio duty cycle and formulating application queries to the nodes. The Surge Network Viewer [19] and the Mote-VIEW [20] 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. The Surge Network Viewer features topology and network statistics visualization as well as logging of sensor readings and the viewing of the logged data. The Mote-VIEW, working as monitoring software, covers essentially the same topics but presents a much cleaner user interface and more features. They are designed to support only Crossbow sensor nodes, thus they are not extensible. SpyGlass [21] visualizes WSN using a flexible multi-layer mechanism that renders the information on a canvas. Data emitted by individual sensor nodes is collected by gateway software running on a machine in the sensor network. It is then passed on via TCP/IP to the visualization software on a potentially remote machine. Visualization plug-ins can register to different data types and visualize the information on a canvas. TinyViz [22] 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. Using a simple plug-in model, users can develop new visualizations and interfaces for TinyViz. It also allows users to interact with a simulation through a GUI, 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 tries to target 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. Design of the framework

This section illustrates in detail the process of problemsolving and planning for the whole framework solution. We first analyze the requirement specification. Then we describe the architecture of NetTopo. Based on the architecture, modules are partitioned and presented in a hierarchy with specific description respectively. After that, we present the design of module interfaces and data persistence. For the purpose of utilizing NetTopo software and extending the framework for future customizable use, this section provides related high-level software design approaches.

3.1. Requirement specification

This subsection provides specific description of the behaviour of the framework to be developed. Related use cases are also included. The specification is divided into functional requirements and non-functional requirements based on project motivation, preliminaries of related work and using experience of researchers in Digital Enterprise Research Institute (DERI), National University of Ireland, Galway.

Functional requirements are listed as follows:

1. System level functions:
   * The system shall provide a graphical user interface on which network deployment, simulation and visualization are displayed.
   * The system shall allow the interaction and communication between simulation and visualization of WSNs.
   * The system shall be extensible to provide device-based wrappers for WSN visualization.
   * The system shall provide a set of high-level APIs for users to simulate their own algorithms.
   * The system shall provide functions for debugging and logging simulation process.
   * The system shall be able to record simulation results.
   * The system shall support for sensor data streams (XML streams) import and export.
   * The system shall provide file-related operations, including file saving, creating and opening, to save the deployment state and support for future reuse.
   * The system shall support WSNs 3D representation.

2. Node level functions:
   * The system shall provide basic symbols representing elements in a WSN including source node, sensor node, sink node, hole and link. All these symbols can be used to build users’ own expected network.
   * The system shall be extensible to allow users to define expected attributes of their own virtual sensor nodes.
   * The system shall allow users to create their own network with maximum freedom, e.g., adding, deleting, selecting, modifying, moving, searching, distributing sensor nodes and configuring their attributes, such as energy, expected lifetime, transmission radius.
   * The system shall allow users to deploy virtual sensor nodes and holes in pre-defined network topologies including line, circle, tree, grid and random or user-defined topologies.
   * The system shall allow users to kill virtual sensor nodes to make them as holes.
   * The system shall allow users to clear up all deployed virtual sensor nodes.

Non-functional requirements include:

* The system shall be programmed in Java so as to integrate with a java-based WSN middleware called Global Sensor Networks (GSN) [23], which is developed under the collaboration between DERI, Galway and EPFL, Lausanne.
* The system shall have a local operating system look and feel.
* The system shall warn users that they cannot re-configure virtual sensor nodes or WSN after simulation starts if they do so.
* The coordinate of any virtual sensor node to be created or modified must be different from others existed.

3.2. Architecture

As required and constrained by the requirement specification, the whole system environment involves the following five entities: user, NetTopo framework, persistent data, sensor devices and gateway. From users’ point of view, NetTopo, regarded as a software application, is located above an operating system in a computer. Persistent data stored in the local machine are transparent to the user. Besides the application, user is aware of real sensor devices. Due to the heterogeneity and hardware-specific properties of sensor devices, a gateway, working as an intermediate node, is incorporated for adapting communication between NetTopo and sensor devices.

Fig. 1 shows the NetTopo architecture. This figure shows not only the relationship of the five entities, but also an overview of the inner framework of NetTopo. User sends commands via Graphical User Interface (GUI) of NetTopo and waits for response, which is normally a visual change on GUI. Two main components in NetTopo are simulator and visualizer. They can interact with each other indirectly by sharing the same virtual WSN, which stores temporary data copied from persistent data. From network point of view, simulator component runs locally on the machine without communicating with outside entities. However, visualizer component should gather information from gateway. The gateway, working as an adaptor, is responsible for communicating with various kinds of hardware devices. This information exchange could be based on various protocols, e.g., TCP/IP Socket and HTTP Get, and it needs to setup the corresponding driver software of devices from which NetTopo can get all kinds of sensed data. Especially, the GSN middleware [6] can be installed in the gateway for gathering sensor data from multiple different real sensor networks and formalize these heterogeneous sensor data into a uniform format. By receiving the processed sensor data streams, mostly XML streams, from GSN, the NetTopo framework can be easily extended to support the visualization of multiple different real sensor networks.
Based on this architecture, all the external entities that NetTopo needs to communicate with are user, local computer and gateway. Inside NetTopo framework, four components including GUI, Virtual WSN, Simulator and Visualizer are presented to show the relationship between inner components and external entities. More specific information on NetTopo inner components and their interactions are illustrated in the following subsection.

3.3. Module design

In this subsection, we take a further look at the software modules of NetTopo framework. According to the requirement specification, Java is the programming language for NetTopo. This object-oriented programming language determines our design methodology. Modules are partitioned based on the functions and relation hierarchy. Class diagrams, sequence diagrams are involved. Each module could consist of one or more classes in which related methods are encapsulated. Interfaces of each module will be illustrated in next coming subsection.

From function level point of view, NetTopo consists of both simulation and visualization functions. These two functions are not just simply placed together and do their individual jobs, which means that they need to interact with each other and manipulate some common components and resources. To support this inner interaction, NetTopo is designed to be composed of several modules as illustrated in Fig. 2. These modules are flexible enough to add new components in the future for system extension.

3.3.1. Main control module

Main Control module, the core module involved in all layers working as a coordinator, is in charge of the interactions of other modules. It can be regarded as an adaptor between input and output interfaces of other modules, which enables them to work smoothly. Main Control module is mainly responsible for the following three tasks:

- Obtaining the references of all other modules. This guarantees that all other modules can be invoked by each other. For example, after deploying a new virtual sensor node, the system should not only reserve a block of memory for saving the node variable and its reference, but also it needs to notify GUI to update the canvas and then users know a node is added. Thus, the reference of GUI in Main Control is important.
- Forwarding commands and returning results. Normally, all kinds of commands sent from user should be forwarded to Main Control module. Then Main Control analyzes the commands, reads or modifies the configuration and then forwards the commands to a specific module responsible for the corresponding action. After getting the result, Main Control returns the results. For example, User send a starting simulation command, Main Control module collects all required information including configuration, algorithms, topologies and forwards command to simulation module. After getting results, it could make log files, update GUI, etc.
- Controlling multi-threads. Simulation and visualization functions are integrated in NetTopo, which causes the synchronization issues of multi-threads programming. Main Control module controls the life cycle of the threads and handles multi-threads concurrency problems.

Due to above functionalities and responsibilities, Main Control module is designed as a singleton pattern [31], which is a design pattern that is used to restrict the instantiation of a class to one object. This means that the class of Main Control module can have only one instance during the whole runtime of the system. The class diagram of...
the singleton pattern is shown in Fig. 3. MainApp is the name of the class representing the Main Control module.

There is only one single static variable MainApp representing the instance of the class, which includes a private construction method and a static public method for retrieving that instance. The private variable is initialized by calling the private construction method statically. This instance can be obtained by the public static method called getApp() each time.

3.3.2. Node module

Node module represents various kinds of virtual sensor nodes. 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-defined node that wishes to run on the simulator must implement the VNode interface. By using this interface mechanism, user can easily define a new sensor node and add preferred attributes to it without worrying about its compatibility with other modules of the framework. The class diagram in Fig. 4 shows the relation between VNode interface and other pre-defined sensor nodes classes. SensorNode class and SinkNode class directly implement VNode interface. SourceNode derives from SensorNode because any sensor node wishing to send sensed data could be regarded as a source node.

For the purpose of extensibility, Node module adopts factory method pattern [32]. Factory method pattern deals with the problem of creating objects without specifying the exact class of object that will be created. To be more specific, the factory method design pattern handles this problem by defining a separate method for creating the objects, which subclasses can then override to specify the derived type of objects that will be created. In Node module, VNodeFactory is a factory class which is in charge of which sensor node should be created during the runtime. Class diagram of VNodeFactory is shown in Fig. 5.

According to this factory method pattern, users can create their expected virtual sensor nodes just by providing the name of the node type. Consequently, by using Java reflexion mechanism, NetTopo can dynamically load the user-defined sensor node classes located in a certain plug-in file directory. It checks if the class implements the VNode interface. In addition, by using a property file recording the description of a sensor node and its corresponding class name, all user-defined sensor nodes can be loaded with descriptions in a list. Users can choose whichever one they like to create a node instance of that kind without manually creating the list of sensor nodes.

3.3.3. Topology module

Topology module stands for the WSN topology to be deployed, which describes the relationship and possible connections between sensor nodes. Network topology can be various types, e.g., line, circle, grid, triangle, tree, random topology. In practice, if users choose to deploy multiple sensor nodes in a ring topology. All they have to do is to enter the coordinate of the circle centre, circle radius and

---

**Fig. 3.** Singleton pattern of Main Control module.

**Fig. 4.** Class diagram in Node module.

**Fig. 5.** VNodeFactory class diagram.
number of nodes. Then a bunch of sensor nodes in a ring are created automatically at once. Additionally, to allow users to create their own network topology, an abstract interface named Topology is declared to define several basic methods representing actions of a network topology. Any user-defined topology that wishes to be used on the simulator must implement the Topology interface. The class diagram in Fig. 6 shows the relation between Topology interface and other pre-defined topology classes. The Coordinate is a class representing the node coordinates on canvas. For the purpose of extensibility, Topology module also adopts factory method pattern, which is similar with the Node module.

3.3.4. Algorithm module

Algorithm module represents an algorithm to be applied in the Virtual WSN. The algorithm can be any routing, clustering, scheduling, controlling algorithm, etc. To allow users to create their own algorithms, an abstract interface named Algorithm is declared to define several basic methods representing actions of an algorithm. Any user-defined algorithm that wishes to be applied by virtual sensor node and WSN on the simulator must implement the Algorithm interface.

Each algorithm in NetTopo can include several functions and keep the references of all these functions. In practice, in order to let users figure out the detailed logic sequence, an algorithm can be disassembled into several smaller logic sub functions. Taking GPSR algorithm [4] as an example, it first needs to draw the planarized graphs such as the Relative Neighborhood Graph (RNG) and Gabriel Graph (GG), then it adopts greedy forwarding or planar perimeters to find the next forwarding node. An interface called AlgorFunc is also defined to allow users to create sub functions for their algorithms.

The class diagram of Algorithm module is presented in Fig. 7. Algorithm and AlgorFunc have an aggregation relationship. For each Algorithm instance, it has no or multiple AlgorFunc instances. For each AlgorFunc instance, it has and only has one Algorithm instance. GPSR [4] and TPGF [24–26] are two algorithms implemented in NetTopo. Their representing classes are Algor_TPGF and Algor_GPSR which directly implement Algorithm interface. Several sub functions of each algorithm, e.g., TPGF_FindOnePath, TPGF_OptimizePath, TPGF_FindAllPaths, GPSR_GG, GPSR_RNG, GPSR_Perimeter, implement AlgorFunc interface and can be obtained by their corresponding algorithm. For simplicity, only three functions for each algorithm are presented in the diagram.

3.3.5. Virtual WSN module

Virtual WSN module works like a runtime sensor nodes repository. It can be regarded as a sensor nodes repository but the data, e.g., node IDs, node types, and node coordinates, are rather kept in memory. Virtual WSN module is mainly responsible for the following three tasks:

- Preserving references of all virtual sensor nodes. This allows that all deployed sensor nodes can be accessed by their handle references.
- Recording the network size when it is created. This is important because the software should be able to operate under stress or tolerate unpredictable or invalid input. The network size information can be used to check some input parameters. For example, the coordinates of a node to be deployed should not exceed the network region based on this size.
- Operating sensor nodes. The operations on sensor nodes include adding, deleting, viewing, modifying a sensor node, getting a collection of all sensor nodes with the same type, and verifying a node whether it has a duplicate coordinate with that of others.

Virtual WSN module has an association with Coordinate and VNode respectively. Class diagram of Virtual WSN module is presented in Fig. 8.

As mentioned in Main Control module, multi-threads are involved in the whole framework. Virtual WSN module, as a sensor nodes repository, is invoked by many other modules. All the resources in Virtual WSN module should be regarded as critical region. Consequently, this module should be thread-safe to guarantee the robustness and reliability of the software. More details on this issue is presented in implementation section.

3.3.6. GUI module

The main graphical user interface shown in Fig. 9 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. Additionally, the GUI provides menu bar and tool bar for action performance. It also allows user to adjust the size of each window pane. Other GUIs include dialogues and wizards.

GUI module encapsulates all related window resources. As required and constrained by the requirement specification, the system shall be platform independent and should have a local operating system look and feel. Thus, NetTopo adopts Standard Widget Toolkit (SWT) [27] to construct the graphical user interface. To display GUI elements, the SWT implementation accesses the native GUI libraries of the operating system using Java Native Interface (JNI) in a manner that is similar to those programs written using operating system-specific APIs. Though the implementation of the toolkit is unique for each platform, programs that call SWT are portable. NetTopo does not only use SWT as GUI construction components, but also use JFace [28], which is a layer that sits on top of the raw widget system, and provides classes for handling common UI programming tasks. It brings model view controller programming to the Standard Widget Toolkit.

3.3.7. Painter module

Painter module is separated from the main GUI due to the frequent painting tasks. The painter is also designed as an abstract interface for various painting requirements, e.g., 2D or 3D. The specific painter used in Fig. 9 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.

Fig. 7. Class diagram of Algorithm module.

Fig. 8. Class diagram of Virtual WSN module.
information. Painter module is responsible for painting all kinds of symbols, e.g., node, link, hole, focus. It interacts with Virtual WSN, Node, and GUI modules. The class diagram of Painter_2D is presented in Fig. 10. For each painting task, Painter_2D first obtains the reference of virtual WSN by Main Control, then it searches the nodes to be painted for their corresponding coordinate information in virtual WSN. Based on the searched information, it paints pixels on the canvas obtained from GUI and then notifies the GUI to update.

3.3.8. Other modules

Other modules include File Manager, Utility and Configuration. File Manager, which handles all related file operations, is responsible for data persistence, e.g., logging runtime information, recording statistical results, keeping virtual sensor nodes’ references. Log information and statistical results are recorded as character streams into human readable format. References of virtual sensor nodes are stored as serialized format in order to easily recover and reuse the inner structure of nodes. Utility module is
also a module involved in all layers described in Fig. 2. Utility module provides several basic and useful services, e.g., defined application exceptions, format verification, number transforms, dialogue wrappers. Configuration module provides a runtime repository for retrieving, modifying system configuration properties, e.g., node painting radius, default link color, default source node’s maximum transmission radius.

Simulator and Visualizer presented in Fig. 2 represent the high level functions in NetTopo. The structure difference between these two modules 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 Visualizer works as a thread that shares some common modules with Simulator, which include Virtual WSN, Main Control, Painter, Node, Configuration and GUI. Sometimes, File Manager can also be involved. The sharing of common modules depends on the implementations of Visualizer for different real sensor devices. Furthermore, using these shared resources sometimes can cause synchronization problem, e.g., when both Simulator and Visualizer modules need to add new sensor nodes in the graphical display canvas. The inner module interactions are illustrated in the following paragraph.

3.3.9. Interaction of modules

In NetTopo, components interactively communicate with each other to achieve the functions of simulation and visualization. At the starting point, virtual sensor nodes should be deployed and their attributes should be configured before the simulation starts. The Algorithm module loaded as plug-in decides how virtual sensor nodes will communicate and forward packets. User commands drive the simulation which runs based on the specification of the loaded algorithm.

Fig. 11 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.

Fig. 12 shows the modules 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, until users manually interrupt this thread. This simple scenario only focuses on the visualization of testbed, in which visualization and simulation modules are running concurrently, but they actually do not interact with each other because no common virtual sensor nodes are used by both modules. A further example of interaction between both modules is presented in case study section.

3.4. Data persistence

Data persistence is the ability to keep data or information around in the same state even after a program ends. Data persistence in NetTopo is needed due to the following three scenarios.
Users deploy a WSN, configure the attributes of virtual sensor nodes and test their own algorithms on it. However, the result could possibly not satisfy the expectation and users want to modify the algorithm and run it on the same virtual WSN environment deployed earlier so as to compare the results between these two. Consequently, preserving the configuration of virtual WSN is quite necessary for users to reuse the same pre-defined WSN.

Some runtime information and actions can be very helpful sometimes for users to debug their algorithms. Logging the information and recording the actions in file is a handy history material.

Researchers who do simulation of testing an algorithm not only expect to see visual results on the canvas but also need to gather some related statistics which could be used to analyze the algorithm performance. This kind of statistics information is often presented in a file with unified format that allows users to further import them into external software to get the graphical results.

Using database for data persistence is the most common approach. It has many advantages, e.g., greater data integrity and independence from applications programs as well as improved data security. However, with respect to the above three scenarios in NetTopo, database is relatively complex, time-consuming and not necessary. Simulation often prefers to high performance rather than the information of a single sensor node. In NetTopo, disk files are adopted as storage media. The format of the file varies in different scenarios.

For the purpose of preserving references of all sensor nodes and their attributes, Java serialization technique is used. Serialization is the process of saving an object onto a storage medium or to transmit it across a network connection link in binary form. When the resulting series of bytes is reread according to the serialization format, it can be used to create an accurate clone of the original object. VirtualWSN class implements Serializable interface so as to be serialized. Related classes including VNode and Coordinate also implements Serializable interface. As VirtualWSN works like a runtime sensor nodes repository, the storage file with format formed by serializing the VirtualWSN can be easily written and reread. Log information and statistics report are recorded in files with character format.

4. Implementation of core features

This section describes the features of NetTopo and analyzes the implementation of the framework based on the design vision presented in the previous section.

4.1. NetTopo features

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

- Platform independent
  - NetTopo is implemented in Java language, which makes it portable between different operating systems. Built upon SWT, the graphical user interface has a local operating system look and feel.
- Extensibility
  - Configurable sensor nodes with user-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 a type of sensor device. To visualize different hardware devices, users can create different wrappers to set up the connection for extracting information.

• 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 different 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 predefined topologies to form some special shapes, e.g., users can deploy a circle by specifying the location of circle centre, 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.

• 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.

4.2. Extensibility

To illustrate the extensibility of NetTopo in code level, we take Node module as an example. As mentioned in previous Section 3, Node module adopts factory method pattern. Fig. 13 shows the code segment of VNodeFactory class. VNodeFactory is a factory class which is in charge of which sensor node should be created. The factory creates object instance of VNode class by a public static method called getInstance(). By providing the name of the node type as an input parameter, it can create a new node object of that type. The type name is a concrete node class name, such as SinkNode (The exact name should include the package name as org.deri.nettopo.node.SinkNode). This code segment represents the power of Java reflexion mechanism. The factory method creates the instance by dynamically loading the node class: Class.forName(), when it is needed. Then the node’s default construction method is called by method newInstance().

The factory method can be invoked by a wizard page when user chooses which node on the list of GUI to be created. Fig. 14 shows a method called getNode() in a wizard page class. This method invokes the factory method in VNodeFactory by providing a node type name. However, the concrete name is not provided directly but got from a nodeNames array.

```java
public class VNodeFactory {
    public static VNode getInstance(String nodeName){
        try{
            Class nodeClass = Class.forName(nodeName);
            VNode node = (VNode)nodeClass.newInstance();
            return node;
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
}
```

Fig. 13. VNodeFactory class.
This nodeNames array preserves the concrete node type names by reading all node type names from a property file listed in Fig. 15. By using nodeDescription, which is a SWT list component of GUI, all the node type names can be displayed on the wizard GUI, and be chosen for creating the corresponding nodes. This mechanism allows the NetTopo framework to easily add new type of sensor node for extension.

4.3. Simulation

Simulation is driven by each individual algorithm which has multiple functions. For each algorithm and function, there is also a similar property list describing the type of an algorithm or function. In main GUI, all algorithms are gathered and displayed as menu items for user to choose. Each menu item has one or several sub menu item in which every sub menu item represents a function in that algorithm. All these algorithms and sub functions are created dynamically using the Java reflexion mechanism. The main program does not need to know which specific algorithm is loaded.

Fig. 16 shows the creation of sub functions of an algorithm. References of functions are obtained by calling the method getFunctions() in algorithm. Then the program traverses the functions array. For each function, the program gets its class’s name and then retrieves the corresponding description from the property list. The description can be displayed in a menu item. The action of a menu is also listened by a widget listener. Each time user click a menu item, the corresponding function starts to run.

4.4. Visualization

Visualization is started and stopped by user’s action as well. The specific implementation of the visualization

```java
public VNode getNode()
{
    return VNodeFactory.getInstance((String)nodeNames.getNodeDescription().getPropertyIndex());
}
```

Fig. 14. Method using factory method.

```java
# Node description
org.deri.nettopo.node.tpf.sensorNode_TPF = Wireless Sensor Node used in TPF
org.deri.nettopo.node.tpf.sourceNode_TPF = Wireless Source Node used in TPF
org.deri.nettopo.node.gpsr.sensorNode_GPSR = Wireless Sensor Node used in GPSR
org.deri.nettopo.node.gpsr.sourceNode_GPSR = Wireless Source Node used in GPSR
org.deri.nettopo.node.holeSinkNode = Sink Node
org.deri.nettopo.node.hostNode_GSN = GSN Host Node
```

Fig. 15. Node type description list.

```java
AlgorFunc[] functions = algorithm.getFunctions();
if (functions == null)
    continue;
for (int i = 0; i < functions.length; i++) {
    String funcName = functions[i].getName();
    final AlgorFunc function = functions[i];
    InputStream res_func = AlgorFunc.class
        .getResourceAsStream("function.properties");
    Properties prop = new Properties();
    prop.load(res_func);
    JMenuItem mi_algorFunc = new JMenuItem(submenu_algorFunc, SWT.PUSH);
    String funcDescription = prop.getProperty(funcName);
    if (funcDescription == null) {
        continue;
    }
    mi_algorFunc.setText(funcDescription);
    mi_algorFunc.addSelectionListener(new SelectionAdapter()
        {
        public void widgetSelected(SelectionEvent e) {
            function.run();
            // users can no longer change the network structure. */
            vsmlnodeComplete(true);
        }
    });
}
```

Fig. 16. Dynamical loading of algorithm functions.
depends on the hardware and its driver interfaces. This subsection illustrates how the visualizer is added to the main framework and how the visualization is stopped by user. A specific case study related to visualization is illustrated in next section. Fig. 17 shows the methods of starting and stopping the visualization. In start process, the Runnable is dynamically created in wizard and a thread which encapsulates this Runnable object then starts. With respect to stop process, the thread is arbitrarily interrupted by its pre-defined method interrupt().

4.5. Thread-safe operations on virtual WSN

As mentioned in Section 3, Virtual WSN module, working as a sensor nodes repository, should be thread-safe to handle the multi-threads concurrency problem. For implementation, Virtual WSN uses Java collection classes to store references of sensor nodes and their coordinates. There are HashMap class and HashTable class which can meet the needs that a node's reference can be quickly obtained by searching its ID. HashTable is thread-safe. HashMap is not thread-safe but has more efficiency. Taking the efficiency into account, HashMap is selected as a runtime storage class in NetTopo. In order to guarantee the Virtual WSN is thread-safe, it uses “synchronized” keyword in some code segments when HashMap is operated.

Fig. 18 shows the “synchronized” keyword used in a method called getNodes in Virtual WSN. This method retrieves all nodes with the node type. The parameter “name” is the node type name should include package name such as org.deri.nettopo.node.SensorNode mentioned. The parameter “derived” verify if the nodes to be retrieved include those nodes which inherit the nodes with the type name specified by the first parameter “name”. For example, SourceNode inherits SensorNode. If we call the method as: getNodes("org.deri.nettopo.node.SensorNode",

```java
private void cmd_visualizeNetwork();
if (wsn == null)
    cmd_newNetwork();
if(ws == null)
    return;
// Open the corresponding wizard */
Wizard_VisualizeNetwork wizard = new Wizard_VisualizeNetwork();
WizardDialog dig = new WizardDialog(sh_main, wizard);
int returnVal = dig.open();
if (returnVal == IDialogConstants.OK_ID) { // User finishes the
    // wizard
    Runnable run_vj = wizard.getRunnable();
    visualizer = new Thread(rus_vj);
    visualizer.start();
    mi_network_stop.setEnabled(true);
    notifyModified();
}

private void stopVisualization(){
if(visualizer!=null)
    visualizer.interrupt();
    mi_network_stop.setEnabled(false);
}

public Collection getNodes(String name, boolean derived){
    synchronized (this.allNodes){
      Collection nodes = new ArrayList();
      Iterator it = allNodes.values().iterator();
      while(it.hasNext()){
        Object node = it.next();
        String nodeTypeName = node.getClass().getName();
        if(nodeTypeName.equals(name)){
            nodes.add(node);
        }
        if(isDerivedClass(node.getClass(),name))
            nodes.add(node);
      }
      return nodes;
    }
```

Fig. 17. Starting and stopping visualization.

Fig. 18. Using synchronized keyword in virtual WSN.
true), all the nodes with the SensorNode type and all its derived types including SourceNode should be returned. If the second parameter is “false”, then only the nodes with SensorNode type will be returned. In this code segment, isDerivedClass method verifies whether the class specified by the first parameter is derived from the class specified by the second parameter.

5. Case studies for test

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 [4] and TPGF [24–26], are implemented and compared based on the statistical results. For visualization, a testbed composed of Crossbow sensor nodes is visualized.

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, e.g., 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. As an example, Fig. 19 shows the visual results on the canvas when running both TPGF and GPSR in the virtual WSN.

In Fig. 19, the red colour node is the source node and the green colour node is the sink node. Pictures (a)–(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. Users need to do the test for many times, and users 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 $600 \times 400$ (1 pixel on the canvas is considered as 1 m). 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, the node number is changed from 100 to 1000 and the transmission radius is changed from 60 to 105 to obtain different average values. By gathering all these average values together, many charts and figures can be drawn to reflect the execution performance of these algorithms.

5.2. Crossbow WSN testbed visualization

Crossbow WSN testbed consists of six sensor nodes. Fig. 22 shows the whole network structure and flow of sensed data, and Fig. 23 shows the visualization in a 3D office/house structure based model. The xbow 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. 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 xbow nodes are considered as source nodes in the virtual WSN. When the temperature reading of any xbow 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.

Fig. 24a shows the visualization of the six Crossbow nodes in the pre-deployed virtual WSN. Fig. 24b shows that one of the Crossbow nodes explored four routing paths by
using TPGF, which actually is an example for reflecting the integration between simulation and visualization for wireless sensor networks.

6. Cross-layer integrations with GSN [23]

As a sister project of the famous GSN middleware, NetTopo is designed to have tight cross-layer interactions with GSN to fully utilize GSN’s existing functions. Currently, two types of cross-layer integrations are designed as follows:

(1) NetTopo getting data from GSN, as shown in Fig. 25a: Basically, GSN is a powerful sensor network middleware because a large number of wrappers had been implemented inside GSN to allow it to gather various sensor streams from heterogeneous sensor devices. Although NetTopo also provides the flexibility to allow users to develop new wrappers for getting sensor streams from real sensor devices, but it may reduce users a lot of effort and time to use GSN directly for gathering the sensor stream, if this sensor device has already been supported by GSN. NetTopo can further get the gathered sensor streams from GSN for integrating into the simulation work.

(2) NetTopo posting data to GSN, as shown in Fig. 25b: GSN provides a powerful data storing and querying solution, which allows NetTopo to post the sensor streams into GSN, and further store them into GSN’s data repository. Users can use GSN’s standard query interface for checking their needed sensor data. This integration can save users a lot of implementation work for storing their sensor streams created by their own simulation algorithms.
7. Discussions

7.1. Is NetTopo aiming at a better simulator than other existing ones?

NetTopo is not aiming at a better simulator than other existing ones, especially, those commercialized special simulators, e.g., OPNET. The basic goal of NetTopo is to make it be a very good and useful platform for some tasks & scenarios, where both testbed evaluation and simulation are needed. However, during the design of NetTopo, we still tried our best to provide very good flexibility, extensibility and scalability in NetTopo.

7.2. How is the scalability and execution performance of NetTopo?

Supporting a large number of sensor nodes with reasonable execution performance is vital for NetTopo. In order to provide some quantity information for the execution...
performance of NetTopo, we conducted a serial test for running TPGF algorithm with different number of nodes in WSN, as shown in Fig. 26. The reason for using TPGF algorithm to test the performance is: In the final step of TPGF algorithm, to guarantee that no path can be further found, the algorithm needs to visit all unused sensor nodes in the network. The time complexity of TPGF is $O(n)$, where $n$ is the number of deployed sensor nodes.

We conducted these tests on a normal laptop computer (ThinkPad T500, Intel(R) Core(TM)2 Duo CPU T9900 @ 3.06 GHz, 2.99 GB of RAM, and Microsoft Window XP Professional N Version 2002, Service Pack 3). The experiment results are listed as following:

- When executing TPGF with 1000 nodes, it takes less than 2 s to return the results.
- When executing TPGF with 2000 nodes, it takes around 10 s to return the results.
- When executing TPGF with 3000 nodes, it takes around 40 s to return the results.
- When executing TPGF with 4000 nodes, it takes around 100 s to return the results.
- When executing TPGF with 5000 nodes, it takes around 180 s to return the results.
- When executing TPGF with 6000 nodes, it takes around 450 s to return the results.

7.3. Is it easy to implement new algorithm in NetTopo?

It is fairly easy to implement new algorithms in NetTopo, and the workload should be relative low, comparing with implementing new algorithm in NS2. Basically, when a user wants to implement a new algorithm, he/she needs to follow the following three steps:

1. Creating a new type of sensor node for his/her new algorithm, by rewriting a new node Java class and registering it into NetTopo. User needs to specify the node configuration in this class.
2. If needed, creating a new type of network topology, by rewriting a new topology Java class and registering it into NetTopo. User can also use the existing implemented network topology.
3. Writing his/her new algorithm. User can use existing implemented algorithms as examples to implement his/her new algorithm, and many utility functions had also been implemented.

The required background knowledge for NetTopo users are mainly two aspects:

1. Background knowledge on wireless sensor networks. Users should at least know what a wireless sensor network is, and its major network entities, e.g., sensor nodes, base station.
2. Background knowledge on Java language programming. Users should have a certain level knowledge & programming skill on Java, especially the object-oriented programming concept.
Normally, for a well educated bachelor student with good Java programming skill, it takes less than 2 weeks to learn the background knowledge of WSNs & NetTopo, and start to use it.

7.4. Is it possible to execute more than one algorithm in NetTopo at the same time?

The answer is yes!

Currently, besides the implemented TPGF and GPSR algorithms, one more new algorithm named as CKN [33] is implemented in NetTopo. In [33], Nath et al. proposed the Connected K-Neighborhood sleep scheduling algorithm (CKN) for duty-cycle based WSNs. CKN algorithm aims at allowing a portion of sensor nodes in the WSN to go to sleep but still keeping all the awoken sensor nodes connected. The number of sleeping nodes in the WSN when applying CKN algorithm can be adjusted when changing the value of K.

When executing CKN algorithm in NetTopo, it will generate a coordinated duty-cycling sensor network topology, and then, based on this generated network topology, the TPGF routing algorithm is executed to explore multiple routing paths. This research work is motivated by studying the impacts of duty-cycle on geographical multipath routing [34].

Furthermore, a Random WayPoint mobility model is implemented in NetTopo. It is feasible to run TPGF algorithm on CKN algorithm based network topology, while both source and sink nodes moving in the network by executing the Random WayPoint mobility model.

8. Conclusions and future work

We have presented NetTopo, a Java-based integrated framework of simulation and visualization for wireless sensor networks. The friendly GUI makes it easy to use and the modular components enable it to be easily extended. Due to the algorithm-oriented design, NetTopo supports the simulation for an extremely large scale network. It is useful for the rapid prototyping of an algorithm. The visualization function uncovers the real device based WSN topology and displays sensed data. Based on modular components design and common graphical resources, visualization process can drive the simulation. Generally, such integration takes the first step into the whole vision that applications can run partially in a simulation environment and partially in a physical WSN testbed and interact with each other to create an environment where algorithms can be much more accurately tested and validated. In our future work, we are going to further realize the routing packet transmission between the NetTopo and real sensor network testbed.

As a major contribution of this research work, currently, NetTopo is released as open source software on SourceForge and has its own website [7]. It has more than 120 Java classes and 20,000 Java lines source codes. Users can freely download the latest version of NetTopo by accessing the NetTopo website [7]. A conference paper related to it has been published in [29]. Moreover, more than 50 international researchers from over 10 different countries have subscribed NetTopo mailing list [30] and are willing to give their contributions to the development of NetTopo, e.g., a researcher named Yuanbo Han from Dalian University and Technology, China had implemented a mobility module in NetTopo, and another researcher named Taye Mulugeta from Addis Ababa University, Ethiopia is working the security aspect of TPGF routing protocol.

Acknowledgment

The work in this paper was supported by: (in part) the Lion project supported by Science Foundation Ireland under Grant No. SFI/02/CE1/1131, (in part) by the European Project CONET (Cooperating Objects NETwork of excellence) under Grant No. 224053. This research was partially supported by Grant-in-Aid for Scientific Research (S)(21220002) of the Ministry of Education, Culture, Sports, Science and Technology, Japan. The authors would like to thank Dr. Lei Wang (Dalian University of Technology, China) for reviewing and supporting this paper.

References


Lei Shu is a currently Specially Assigned Research Fellow in Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University, Japan, and was a Research Scientist in the Digital Enterprise Research Institute (DERI), at the National University of Ireland, Galway (NUIG). He received the B.Sc. degree in Computer Science from South Central University for Nationalities, China, 2002, and the M.Sc. degree in Computer Engineering from Kyung Hee University, Korea, 2005, and the Ph.D degree in Digital Enterprise Research Institute, NUIG, in 2010. He has published over 80 papers in related conferences, journals, and books. He has been awarded the MASS 2009 IEEE TCs Travel Grant and the Outstanding Leadership Award of EUC 2009 as Publicity Chair. He has served as guest co-editor and editor of 18 international journals. He has served as Program Co-chair of MMSN 2010, PMSN 2010, UBSN2010, MSSW 2009, MMSN 2009, PMSN 2009; Workshop Co-Chair of CIT 2010, EMC 2010; Publicity Co-Chair of BodyNets 2010, EMC 2010, EUC 2009, PiCom 2009, ASIT 2009, EmbeddedCom 2009, CPSE 2009; TPC members of more than 60 conferences including, MASS, IWCMC, BROADNETS, WICON, Tridentscom, DEXA, Chinacom, UIC, WORLDCOMP, etc. He has served as reviewer of more than 30 journals, including, IEEE Network Magazine, IEEE Transaction on Wireless Communications, IEEE Journal of Selected Areas in Communications (J-SAC), ACM/Springer Mobile Networks and Applications (MONET), and ACM/Springer Wireless Networks (WINET), etc. His research interests include wireless multimedia sensor networks, wireless sensor networks, context aware middleware, and sensor network middleware, and security. He implemented a new open source WSNs simulator & visualizer: NetTopo. He is a member of ACM and IEEE. Email: lei.shu@ieee.org.

Manfred Hauswirth is Vice-Director of the Digital Enterprise Research Institute (DERI), Galway, Ireland and professor at the National University of Ireland, Galway (NUIG). He holds an M.S. (1994) and a Ph.D (1999) in computer science from the Technical University of Vienna. From January 2002 to September 2006 he was a senior researcher at the Distributed Information Systems Laboratory of the Swiss Federal Institute of Technology in Lausanne (EPFL). Prior to his work at EPFL he was an assistant professor at the Distributed Systems Group at the TU Vienna. His main research interests are on semantic sensor networks, sensor networks middleware, large-scale semantics-enabled distributed information systems and applications, peer-to-peer systems, Internet of things, self-organization and self-management, Semantic Web services, and distributed systems security. He has published over 70 papers in these domains; he has co-authored a book on distributed software architectures and several book chapters on P2P data management and semantics. He has served in over 130 program committees of international scientific conferences and was program co-chair of the Seventh IEEE International Conference on Peer-to-Peer Computing in 2007 and general chair of the Fifth European Semantic Web Conference in 2008. He is a member of IEEE and ACM and is on the board of WISEN, the Irish Wireless Sensors Enterprise Led Network, and the scientific board of the Corporate Semantic Web research center at FU Berlin. He is a member of IEEE (Computer and Communication Societies) and ACM. Email: manfred.hauswirth@deri.org.
Han-Chieh Chao is currently a jointly appointed professor in the Institute of Computer Science and Information Engineering and the institute and department of electronic engineering at National Ilan University, Taiwan; jointly adjunct professor in the institute and department of electrical engineering at National Dong Hwa University, Taiwan; and honorary adjunct professor in Beijing Jiaotong University (985 University), Xiamen University (985 University), Lanzhou University (985 University) and Yantai University, China. His research interests include High Speed Networks, Wireless Networks, IPv6 based Networks, Digital Creative Arts and Digital Divide. He received his M.S. and Ph.D degrees in Electrical Engineering from Purdue University in 1989 and 1993 respectively. He has authored or co-authored 4 books and has published about 250 refereed professional research papers. He has completed 90 MSEE and 2 Ph.D thesis students. Dr. Chao has received many research awards, including Purdue University SRC awards, and NSC research awards (National Science Council of Taiwan). He also received many funded research grants from NSC, Ministry of Education (MOE), RDEC, Industrial Technology of Research Institute, Institute of Information Industry and FarEastTone Telecommunications Lab. Dr. Chao has been invited frequently to give talks at national and international conferences and research organizations. Dr. Chao is also serving as an IPv6 Steering Committee member and co-chair of R&D division of the NICI (National Information and Communication Initiative, a ministry level government agency which aims to integrate domestic IT and Telecom projects of Taiwan), Co-chair of the Technical Area for IPv6 Forum Taiwan, the executive editor of the Journal of Internet Technology and the Editor-in-Chief for International Journal of Internet Protocol Technology and International Journal of Ad Hoc and Ubiquitous Computing. Dr. Chao has served as the guest editors for Mobile Networking and Applications (ACM MONET), IEEE JSAC, IEEE Communications Magazine, Computer Communications, IEE Proceedings Communications, and Wireless Communications & Mobile Computing. Dr. Chao is an IEEE senior member, a Fellow of the Institution of Engineering and Technology (FIET), and Chartered Fellow of British Computer Society (FBCS).

Min Chen is an assistant professor in School of Computer Science and Engineering at Seoul National University (SNU). He was a research associate in Dept. of Computer Science at University of British Columbia (UBC) since Mar. 2009. He has worked as a Post-Doctoral Fellow in Dept. of Electrical and Computer Engineering at UBC for three years. Before joining UBC, he was a Post-Doctoral Fellow at SNU for one and half years. He has published more than 60 technical papers, including 32 journal papers. He is the sole author of a textbook OPNET Network Simulation (Tsinghua Univ. Press, 2004), Dr. Chen received the Best Paper Runner-up Award from The Fifth International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine) 2008. He was interviewed by Chinese Canadian Times where he appeared on the Celebrity Column in 2007. He has worked as session chairs in several conferences, such as VTC’08, QShine’08, ICACT’09, Trientcom’09, and ICC’09. He is a TPC co-chair of BodyNets 2010. He is a workshop chair of CHINACOM 2010. He is the co-chair of MMASN-09 and UBSN-10. He was the TPC chair of ASIT-09, ASIT 2010, TPC co-chair of PSI-09 and PSI-10, publicity co-chair of PCom-09. He serves as the corresponding guest editors for several international journals, such as ACM/Springer Mobile Networks and Applications (MONET) Special Issue on “Ubiquitous Body Sensor Networks”, International Journal of Communications System (IJCS) Special Issue on “Advances on Multimedia Communications”, International Journal of Sensor Networks (IJJSN) Special Issue on “Recent Advances in Sensor Integration”, and International Journal of Communication Networks and Distributed Systems (IJCNDS) Special Issue on “Mobile, Multimedia, Ad Hoc & Sensor Networks”. He is managing editor for IJAACS Journal. He is an IEEE senior member.

Yan Zhang received Ph.D degree in School of Electrical & Electronics Engineering, Nanyang Technological University, Singapore. He is a regional, associate editor or on the editorial board of: Wiley Wireless Communications and Mobile Computing, Security and Communication Networks; International Journal of Network Security; International Journal of Ubiquitous Computing; Transactions on Internet and Information Systems; International Journal of Autonomous and Adaptive Communications Systems; International Journal of Ultra Wideband Communications and Systems and International Journal of Smart Home. He is the Book Series Editor for the book series on “Wireless Networks & Mobile Communications” (Auerbach Publications, CRC Press, Taylor & Francis Group). He serves as guest co-editor for: IEEE Intelligent Systems, Security and Communication Networks, Wireless Personal Communications, Computer Communications, Journal of Universal Computer Science, Journal of Cluster Computing, EURASIP Journal on Wireless Communications and Networking, Journal of Wireless Personal Communications. He is serving as co-editor for several books. He serves as organizing chairs for many conferences, including Program Co-Chair (BROADNETS09, IWCMC09, UIC08, PCAC07, ISM07), Symposium Co-Chair (ChinaCom08-09), Industrial Co-Chair (ACM MobiHoc08), Workshop Co-Chair (ADHOCNETS09, APSCC08), Publication Chair (PSATS09, ISWCS07), and Track Co-Chair (ITNG09). He serves as TPC for numerous conferences, including ICC, GLOBECOM, WCNC, PIIMRC, VTC, CCNC, AINA, ISWCS, HPCC, etc. He received the Best Paper Award in the IEEE AINA07. From Aug. 2006, he works with Simula Research Laboratory, Norway. His research interests include resource, mobility, spectrum, data, energy and security management in wireless networks and mobile computing. He is a member of IEEE and IEEE ComSoc. Email: yanzhang@ieee.org