The Development of A Realistic Simulation Framework with *OMNeT++*

Jiming Chen†, Jianhui Zhang†, Weiqiang Xu§,Lei Shu‡, Youxian Sun†
† Institute of Industrial Process Control, Zhejiang University, Hangzhou, P.R.China
jmchen@ieee.org, {jhzhang, yxsun}@iipc.zju.edu.cn
§ College of Informatics and Electronics, Zhejiang Sci-Tech University, Hangzhou, P.R.China
wqxu@zstu.edu.cn
‡ Digital Enterprise Research Institute, National University of Ireland, Galway
lei.shu@deri.org

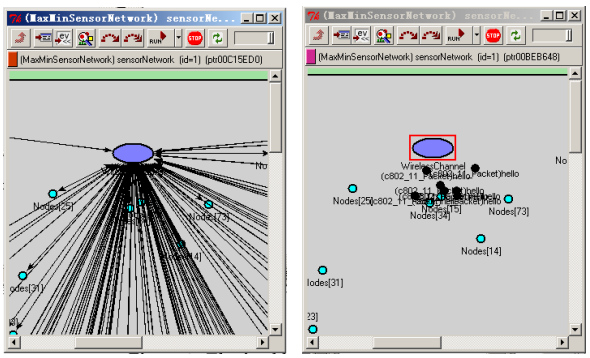
Abstract

It is necessary to use simulation tools in the study of wireless network. The object-oriented modular discrete event network simulator, OMNeT++*, can not only afford many network simulation but has strong GUI support and an embeddable simulation kernel. In this paper, a novel realistic simulation framework (called* R-simulator*) is proposed based on* OMNeT++*.* R-Simulator *has realistic GUI and strong "message" management while considers the node mobility and protocol stack.* R-Simulator *is also time-saving and memory-efficient. The design process of* R-Simulator *and its advantage on realistically displaying the network are detailedly described. Based on* R-Simulator*, we give examples to show its favorable performance.*

1 Introduction

Simulation tools have been widely used in network research. Because many problems need be analyzed by simulation while they are NP-hard. In all network simulation tools, *Oment++* is an appropriate simulation environment for network simulation because it is public-source, component-based, modular and open-architecture with strong GUI support and an embeddable simulation kernel [8]. Although there have been several simulation tools, such as NS2 [7], *OMNeT++* is more easy to study and use because of its convenience for model establishment, modularization, expandability and so on. Therefore, we design *R-Simulator* based on *OMNeT++*. There are also several simulation frameworks [10][1][5] [2][6][9][4], such as *SensorSimulator* created at Louisiana State University based on the *OMNeT++* environment [10]. The simulation framework is extensible for developers and researchers to investigate topological, phenomenological, networking, robustness and scaling issues, to explore arbitrary algorithms for distributed sensor nodes, and to defeat those algorithms through simulated failure. Besides the same purposes and capacity, our simulator is also different from *SensorSimulator* at the following aspects.

1) The interface of "SensorSimulator" in *TKENV* of *Oment++* is very tumultuous as shown in Figure 1. In Figure 1(a), *TKENV* displays the links between nodes and wireless channel module. Because the wireless channel module exists, all arrows point from other nodes to the same point. Therefore the display looks congested and jumbled, which is not convenient to debug and to differentiate the messages. The shortages still can not be relieved to obscure the links between nodes and wireless channel module because the messages transportation still results into congested and jumbled display, as shown in Figure 1(b). More fatal shortage is that the network topology is not intuitionistic.



(a) Display the connections (b) Display not the connections but the messages

Figure 1. The jumbled display of *SensorSimulator*

In *R-simulator*, no redundant links between nodes do exist, as shown in the examples in Section 3. *R-simulator* removes the wireless channel module as shown in Figure

2(b). Therefore, *R-simulator* has less submodule and less complex. The memory space and time are saved and the network topology can be intuitively visible.

2) The "Radio" submodule has very different function in *R-simulator* and *SensorSimulator*. In *SensorSimulator*, "Radio" submodule just stores the radio state and counts the transmission power consumption. In *R-simulator* (see Figure 2), "Radio" submodule not only has the function of that in *SensorSimulator* but stores the information of the link with its neighbors. In Section 2, we will give more detailed description on the module.

3) There is a "Sensor" module in *R-simulator* but no this kind of module in *SensorSimulator*. To add the module is necessary because one node may have one or more sensors and different kind of sensors have different character, such as sensing range.

4) *R-simulator* has a "Memory" module but *SensorSimulator* has not. In protocol stack, each layer may create many messages and data. When the number of messages and data is big, the requirement for memory space is also very big and sometimes is too huge to continue the simulation. So a "Memory' module can manege these messages and data, which can save memory space.

5) *R-simulator* has a "Mobility" module in addition. The module decides whether a node moves or not and what the mobility mode is.

6) *R-simulator* has a "DynSize", a "TR" and an "SR" module. "TR" indicates the radio transmission range and "SR" indicates the sensing range. These modules are all invisible in GUI. In Section 2, we will describe the function of these three modules in detail.

We also consider the node and linkage fault. When any of hardware components in a node or the software fails, the node fault may occur. We add a short program for the node or linkage fault in "Coordinator" module. Any module can start the program to turn off the node but not the whole network when it fails. The linkage fault may happen between one node and its neighbors. So we use "Radio" module to control the linkage fault.

The rest of the paper is organized as following. Section 2 describes the simulation model of *R-simulator* and its modules. Section 3 gives three simulation examples to show the capacity of *R-simulator*. In Section 4, we conclude our work and present some future work to improve our framework and the relative programs for some protocols.

2 Simulator Model and Modules

In this section, *R-simulator* framework and its modules are described and the "Network Layer" module is designed in detail as shown in Figure 2. Figure 2(b) shows the "Network" module, which represents a complete structure of the network layer. The *R-simulator* framework contains a "Target", a "Sensor Node" (see Figure 2(a)) and a "DynSize" module. The "Target" module can represent a locomotive car, a light or noise source, and so on. Here we do not give detailed model for this module in order to decrease the complexity of the "Network" module. Because we can simply use the "Sensor Node" module to represent it by closing the "Protocol stack" part of "Sensor Node" in Figure 2(a) and changing the transmission mode. The "DynSize" module in break-line frame isn't displayed in GUI, such as in the *TKENV*. The module dynamically decides how many neighbors each node has, and accordingly decides a array size in the "Radio []" module of the "Sensor Node" module. The `array` stores the `gates` between a node and its neighbors. Every `message` is transmitted to its neighbors through these `gates`, meanwhile a copy is sent to the "TR" module. Then "TR" module displays the transmission range of the message according to the node's transmission power level. Like "TR", the "SR" displays the sensing range of a sensor when the sensor senses the phenomenon around it. In the following subsection, we describe the "Sensor Node" module and its submodel "Network Layer".

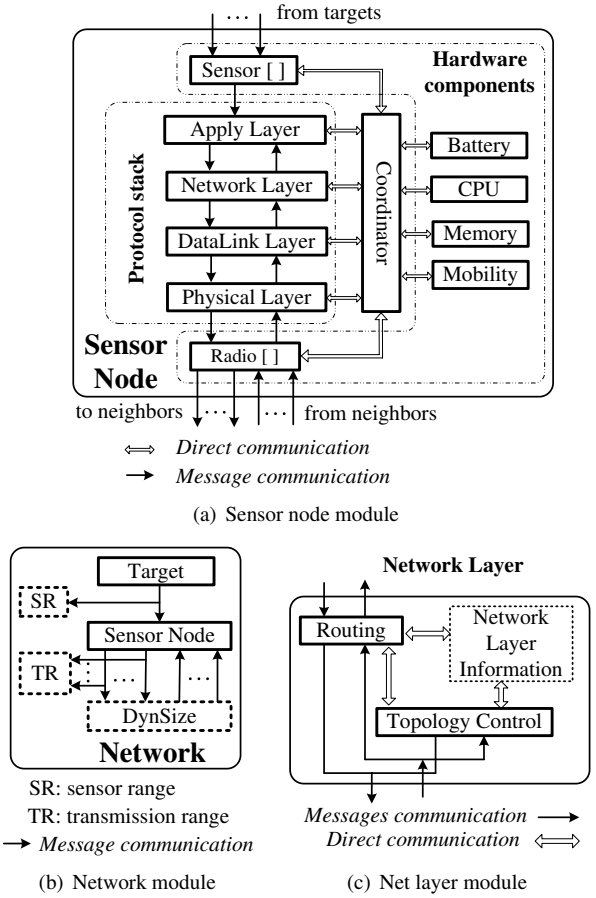


Figure 2. *R-simulator* framework and its modules

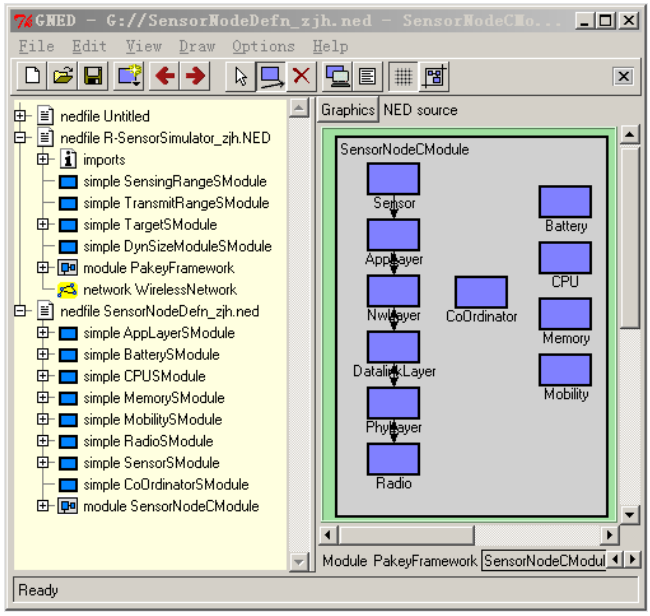


Figure 3. GNEG displays *R-SensorSimulator*

All of these models are specified through the Network Description Language (NED) and is displayed in GNED as shown in Figure 3. The figure shows the "Sensor Node" module and its submodules wrote in NED. The submodules, connected by double-headed arrows, communicate with each other by `message` and other isolated submodules directly communicate with each other. These submodules respectively correspond with those in Figure 2(a). NED files are not used directly but are translated into C++ codes by the NEDC compiler, then compiled by the C++ compiler and linked into the simulation executable. In Section 3, we use VC++ to compile these codes in Windows XP operating system in order to compile and debug these codes conveniently.

2.1 Sensor Node Module

Figure 2(a) shows the "Sensor Node" module, which logically composes of three parts: protocol stack, coordinator and hardware components. In the "protocol stack" part, there are four modules: "App Layer", "Network Layer", "Datalink Layer" and "Physical Layer". These modules respectively are mapped to four protocol layers: an application layer, a network layer, a data link layer and a physical layer.

The four layers have different functions [11]. The application layer affords many application protocols explored for many different areas, such as sensor management protocol, task assignment and data advertisement protocol. Sensor nodes are scattered densely in a field either close to or inside the phenomenon and they need be linked together in order to deliver data packets from one node to another. Therefore, the network layer should supply topology control protocols and routing protocols. The data link layer is responsible for the multiplexing of data streams, data frame detection, medium access and error control. It ensures reliable point-to-point and point-to-multipoint connections. The multiplexing refers to create and detect data frame. The physical layer is responsible for frequency selection, carrier frequency generation, signal detection, modulation and data encryption.

The "hardware components" part contains six modules: "Sensor[ ]", "Battery", "CPU", "Memory", "Mobility" and "Radio[ ]". The brackets added after "Sensor" and "Radio" indicate there are more than one `gate` in the "Sensor[ ]" and "Radio[ ]" modules. Since a node may equip several sensors for different phenomenons, which may have completely different characters, the node should be able to deal different sensor data at same time. Each `gate` in the "Sensor[ ]" module represents a sensor and has a propagation model under it to represent the sensor channel, through which the sensor decides which kind of stimuli it can receive and how the intensity of the stimuli if it can receive. The "Radio[ ]" module stores the `gates` through which one node communicates with its neighbors. Can a node be a neighbor of others? It is decided by the "DynSize" module in Figure 2(b). Suppose there is a node named N_1 and $n-1$ other nodes named N_2, $\cdots$, N_n when the whole network totally contains n nodes. The "DynSize" module checks whether N_2 is in the maximal transmission range of N_1 or not, whether N_3 is or not and so on. Till all other nodes are checked, the number of the neighbors of N_1 can be known. Therefore, the size of the `array` in "Radio[ ]" of N_1 also can be confirmed. Suppose that the position information of all nodes are stored in the file `Position.ini`, we use the below pseudocode to initialize nodes and to establish the connections from a node to its neighbors, therefore the size of the `arrays` in the "Radio[ ]" can be confirmed.

```
01.Get X, Y postion of all nodes from
   Position.ini;
02.for( each of all nodes, named N1 )
03.  for( other node, named N2)
04.    Compute the distance between them;
05.    if(the distance<=max communication
          radius)
06.     Set N2 as a neighbor of N1;
07.     Add one to the size of N1's Radio[];
08.     Link N1's Radio[] to N2's Radio[];
09.   endif
10. endfor
11.endfor
```

The "memory" module not only imitates the memorizer in a node but also manages the `messages` and `data` cre-

ated by the simulation programs. Therefore, its first function, also a basic function, is to determines the size of memory space in a node, the memory rate and so on. Another important function of the module is to manager the `messages` and `data`. During a long time or a large scale simulation, a large amount of `messages` and `data` are created and delay in the simulation system. Because they can not be dealt in time, they occupy large memory space. It increases the simulation cost and decreases the simulation efficiency. Furthermore, there are often many undisposed `messages` remained at the end of a simulation and *OMNeT++* deals them one by one, which costs much time. The "memory" arranges the `messages` and `data` and deletes them all together when the simulation ends, which saves memory and time. The "mobility" module decides whether a node is mobile or not. It gives the velocity, direction and acceleration if a node is mobile. The "Battary" and "CPU" respectively imitate the battery and processor in a node.

2.2 Network Layer Model

Since the topology control is important and necessary to decrease the power consumption and the network interference and to increase the network capacity in WSN [16], we design the "Network Layer" module as Figure 2(c). The module contains two sub-modules: "routing" and "topology control" (TC), and a "network layer information" container, which stores neighbor information and routing table as shown in Figure 4. In a simulation, the "routing" sub-module runs routings. The "topology control" sub-module executes TC protocols.

In the simulation examples of Section 3, the routing is GEAR [17] because it can deal with "routing void", which is often created when the nodes is deployed randomly and TC exists. Paper [16] lists many protocols including CBTC(α)[13], GLSS/FLSS [14]. We also propose new TC protocols: GAFT/LAFT while considering the effect of routing protocol and MAC protocol[12]. The simulations on these five TC protocols and a simulation without TC are designed and evaluated together. The simulation results are presented in Section 3.2. Before running the routing protocol, the TC protocols are first executed to obtain the neighbor information, as shown in Figure 4(a), including one-hop neighbors or two-hop neighbors if necessary. In Figure 4(b), "ID" refers to the node address, which is unique in the same network. "P_r" refers to the receive transmission power on the receive node radio when the transmission power of sender node is adjusted to the maximal value. "Link" indicates whether one node establishes logical link with its neighbors or not. TC protocols can be locally or globally run again if necessary. Based on "Neighbor information", the routing protocol builds routes among the network and obtains the information stored in "Routing table". Several source nodes (source 1, 2 and so on as shown in Figure 4(b)) may pass the same mediate node forwarding to their own target node (target 1, 2 and so on). Therefore, each node has one routing table and points at different next-hop node (see "Next Hop 11", "Next Hop 22" in Figure 4(b)) in different routes.

One-hop Neighbor	ID	ID 1	ID 2	$\cdots$
	P_t(dBm)	P_r 1	P_r 2	$\cdots$
	Link(Y/N)	No	Yes	$\cdots$
Two-hop Neighbor	ID	ID 1	ID 2	$\cdots$
	P_t(dBm)	P_r 1	P_r 2	$\cdots$
	Link(Y/N)	Yes	No	$\cdots$

(a) Neighbor information

Source 1	Source 2	$\cdots$	Source *k*
Target 1	Target 2	$\cdots$	Target *k*
Next Hop 11	Next Hop 21	$\vdots$	Next Hop *k*1
	Next Hop 22		$\vdots$
			Next Hop *kn*

(b) Routing table

Figure 4. Network layer information

There is other information, which can be stored in the "Neighbor information" module, when users design their own simulation. For example, the routing table may be very different when different routing protocols are executed.

3 Simulation Examples

In the section, three simulation examples based on *R-simulator* are given, which are the establishment of GEAR [17], the target tracking and the performance comparison among topologies. First, the simulation environment and some parameters are set. Then, the simulation results are presented and discussed.

3.1 Setting Simulation

A WSN contains several or hundreds of energy-limited nodes, each one has an omni-directional antenna. It also can gather the data from physical surrounding by the sensors on it. We suppose nodes can adjust their transmission power like MICA2/MICAZ [3]. A wireless sensor node can receive the signal from another node if it is within the transmission range of the sender. Otherwise, they relay the messages through multi-hop wireless links by using intermediate nodes. Consequently, each node also acts as a router, forwarding data packets for other nodes. In the following simulation examples, different number of nodes are randomly deployed uniformly in a $1000m \times 1000m$ region. The maximal transmission range of each node is $262.5m$.

In *R-simulator*, all `messages` are transmitted from one node to others through "Radio[ ]" module, which acts as

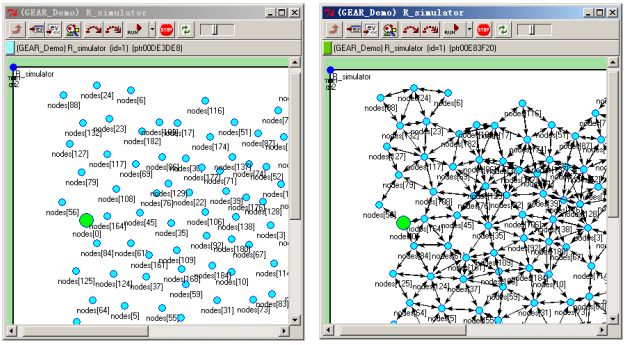
the wireless medium. We use the *free space propagation model* to predict the received signal power of each message packet [15]. In "Physical Layer", we control the data transmission rate and add up energy cost because of transmitting or receiving data. "DataLink Layer" is responsible for the multiplexing of data streams, data frame detection, medium access and error control. The multiplexing refers to create and detect data frame. The length of data frame is an important parameter in Medium Access Control(MAC). It is optimally about 59 byte in MICA2/Z for data frame [3]. We respectively packet broadcast (used by TC) and control message, such as RTS and CTS, into a single frame. The data is cut and packed into data frame according to the length of data frame. We use `cQueue` to define two queues: `Queue_of_message_frame` and `Queue_of_data_frame`, which respectively store the message and data frame. Here, MAC802.11 is used to afford medium access control and we write the following codes to control the header of message or data frames.

```
MAC_802_11_PACKET.h
struct MAC_802_11_Header{
  int Source_Node_ID;
  vector < int > Destination_Node_ID;
}
class MAC_802_11_Packet : public cMessage{
  protected:
    MAC_802_11_Header MACheader;
    ...
}
```

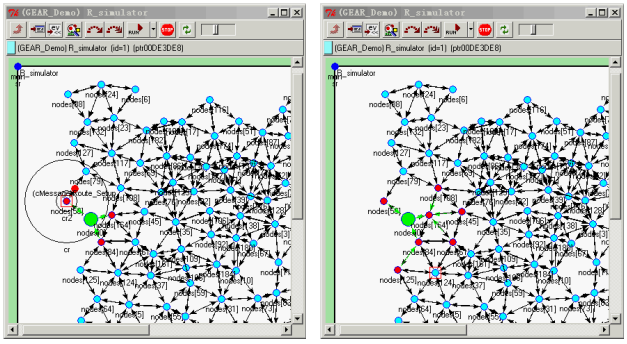
In "Apply Layer", there is no application program or protocol but a simple program to create data periodically or to receive and add up the data from bottom layer "Network Layer". The function of "Network Layer" is described in subsection 2.2.

3.2 Simulation Results

The first example is the simulation for GEAR and the simulation results are presented in Figure 5. All figures show a quarter of the whole simulation scenario. In Figure 5(a), nodes are deployed but they are not linked. The "DynSize" module establishes the linkages among all nodes as shown in Figure 5(b). Since there is no a "Wireless channel" like *SensorSimulator* at LSU, the whole simulation scenario is very shapely and the topology is also very clearly displayed. *R-simulator* is also very vivid as Figure 5(c) shows a node transmitting a message in its transmission range. A circle centered at the node represents the transmission range. When the message is transmitted to its target, the circle disappears. In other word, the circle is dynamically displayed by *R-simulator*. In Figure 5, the node with big size is the "sink". GEAR starts at the sink and any other node, which has established a route to the sink, becomes red. The links on the established route become green as shown in Figure 5(d).



(a) The nodes are deployed (b) The links between nodes are established

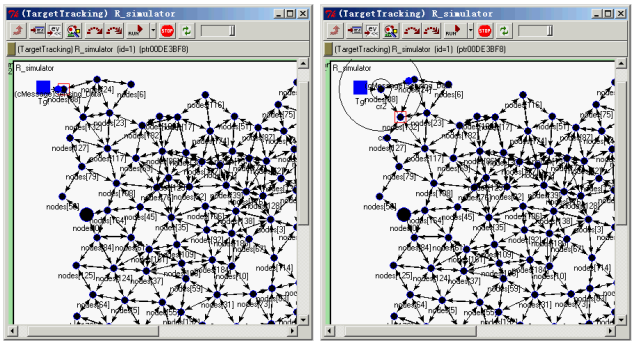


(c) A node is transmitting a message (d) The establishment of routing is in process

Figure 5. The implement for GEAR in R-simulator

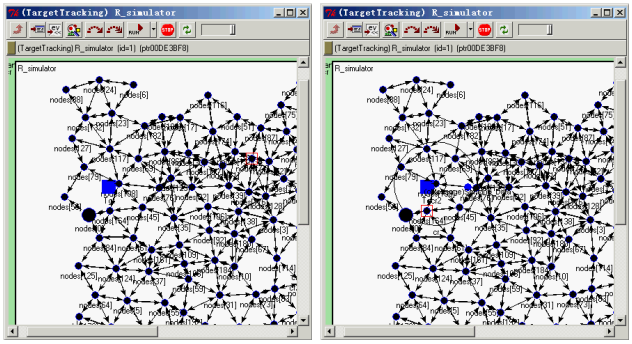
The second example is the target tracking. After the network implements the topology control and GEAR routing, the routes between all nodes and the sink are established as shown in Figure 5. Now the network can track a target, which is invading the network as shown in Figure 6(a). In this sub-figure, the little square in the upper left represents the target, which is closing to the network and being sensed by a node. The circle centered at the node represents the sensing range. In Figure 6, the dot with big size is the sink. The sink can be connected to a personal computer or a internet. Here all nodes sense the target by the periodical messages, which are broadcast by the target. In Figure 6(b), the node, which senses the target, is transmitting a message and sends the message to the sink through its neighbors. In the process, the target continues moving. After a while, it reach the middle of the network as shown in Figure 6(c). It also is sensed by a node as shown in Figure 6(d) and the node is sending a message to the sink through its neighbors.

The third example is the simulation for the comparison among the topology control protocols, which is described in Section 2.2. Note that the network layer information



(a) The target is invading and broadcast a signal

(b) A node "senses" the target



(c) The target is in the network

(d) A node senses the target

Figure 6. The target tracking in R-simulator

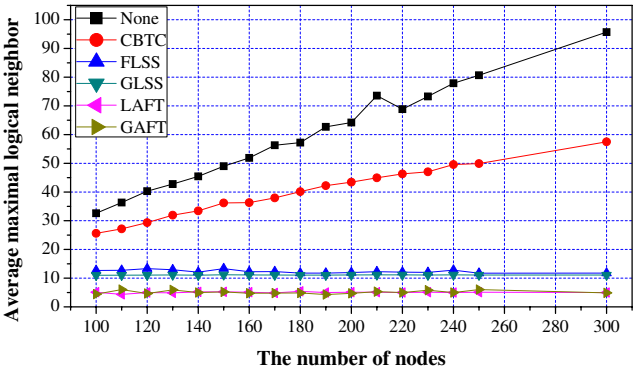


Figure 7. Comparison of None, CBTC($\frac{\pi}{3}$), $FGSS_2$, $FLSS_2$, $GAFT_2$ and $LAFT_2$ with respect to maximal average logical neighbor($k = 2$)

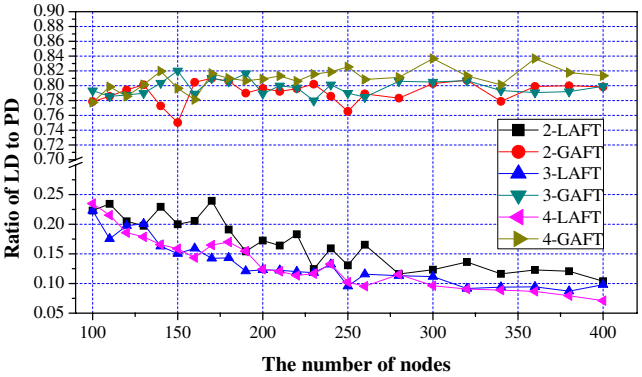


Figure 8. Comparison of GAFT and LAFT with respect to the ratio of LD to PD, where LD and PD refer to logical neighbor and physical neighbor

in Figure 4 should be stored in a `vector` container or a `Arraylist`, which can save memory, especially when the number of nodes reaches one hundred or more. Some statistic data are drawn into curves as shown in Figure 7 and 8. In Figure 7, k is the vertex connectivity of a network. GAFT/LAFT, GLSS/FLSS etc are all topology control protocols described in Section 2.2. One node establishes logical links with some of its neighbors after topology control. In the figure, logical neighbors (degree) of a node are those who have logical links with the node. In whole network, every node has different amount of logical neighbors so there must be a node has maximal amount. In Figure 7, there are one hundred samples under each case of the number of nodes. In Figure 8, the physical neighbors of a node is those who in the transmission range of the node. The figure shows the ratio of LD to PD under different connectivities.

4 Conclusion and Future Work

In this paper, we present our novel simulation framework—*R-simulator* and describe its structure and mechanism. Based on the framework, we give out three examples to test its performance. The results show that *R-simulator* is an extensible, object-oriented and event-based framework, which provides a favorable simulation framework for users and researchers.

We will improve the C\C++ programs generated from the NED files of *R-simulator*. It is important to low the time and space complexity for the simulation framework. Although there are many well known routing, MAC or topology control protocols, which have been compiled into C\C++ programs [8][7], they are not standard and have weak portability. So it is necessary to canonically compile some well known protocols into the C\C++ programs. Furthermore, the further effort will be give to compare the performance of *R-simulator* framework with others in the memory utilization, execution time and so on.

5 Acknowledge

This work is supported by NSFC-Guangdong Province Union Project under grants U0735003; NSFC under grants No.60604029, 60702081; NSF of Zhejiang Province under grants No.Y106384; the Science and Technology Project of

Zhejiang Province under grants No.2007C31038 and 863 High-tech Program under grants No.2007AA041201-1.

References

[1] "Castalia" simulation model. http://castalia.npc.nicta.com.au/.

[2] "Chsim" simulator. http://www.consensus.tudelft.nl/ software.html.

[3] Crossbow Technology Inc. http://www.xbow.com.

[4] MAC simulator. http://www.consensus.tudelft.nl/ software.html.

[5] "Mobility Framework" simulation model. http://mobility-fw.sourceforge.net/.

[6] "Nesct" simulation model. http://www.isi.edu/nsnam/ns//.

[7] The network simulator-ns-2. http://www.isi.edu/nsnam/ns//.

[8] Omnetpp. http://www.omnetpp.org/.

[9] "Positif" simulation model. http://www.consensus.tudelft.nl/software.html.

[10] Sensim. http://csc.lsu.edu/sensor_web/.

[11] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38:393–422, 2002.

[12] J.H.Zhang, J.M.Chen, Y. Wang, Y. Xiao, and Y.X.Sun. A simple algorithm for fault-tolerant topology control in wireless sensor network. In *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) to appear*, Cannes, France, 2008.

[13] L. Li, J. Y. Halpern, P. Bahl, Y. M. Wang, and R. Wattenhofer. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. In *Proceeding of ACM Symposium on Principles of Distributed Computing (PODC), Newport, RI, USA*, pages 264–273, Aug. 2001.

[14] N. Li and J. C. Hou. FLSS: a fault-tolerant topology control algorithm for wireless networks. In *Proceedings of the 10th ACM/IEEE Annual International Conference on Mobile Computing and Networking (MobiCom), New York*, pages 275–286. ACM press, 2004.

[15] T. S. Rapport. *Wireless communications: principles and practice*. Prentice Hall, Upper Saddle River, NJ, 2001.

[16] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comp Surveys*, 37(2):164–194, June 2005.

[17] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical report, UCLA Computer Science Department, May 2001. UCLA/CSD-TR-01-0023.