Wireless Sensor Network, **2009, 1, 467-474**
doi:10.4236/wsn.2009.15056 Published Online December 2009 (http://www.scirp.org/journal/wsn).

EasiSim: A Scalable Simulator for Wireless Sensor Networks

Haiming CHEN[1,2], Li CUI[1*], Changcheng HUANG[3], He ZHU[1,2]
[1]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China*
[2]*Graduate School of the Chinese Academy of Sciences, Beijing, China*
[3]*Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada*
Email: {*chenhaiming, lcui, zhuhe*}*@ict.ac.cn, huang@sce.carleton.ca*
Received July 11, 2009; *revised September* 21, 2009; *accepted September* 23, 2009

Abstract

Traditional simulators have deficiencies of scalability, thus they are not so efficient in running simulations with large-scale networks. In this paper, we present a new simulator, namely EasiSim, specifically for evaluating sensor net-works on a large scale. EasiSim is featured by organizing its core components, including nodes, topology and scenario, in a hierarchically structured approach. The hierarchically structured organization enables nodes to process all the concurrent events in one batch, hence reducing the running time by an order of magnitude. Moreover, we propose a visualization scheme based on a Client/Server model which separates the graphical user interface (GUI) from the simulation engine, and therefore the scalability of the simulator will not be decreased by complex GUI. Extensive simulations show that EasiSim outperforms ns-2 in terms of *real running time* and *memory usage*.

Keywords: Wireless Sensor Networks, Simulator, Scalability, Component Organization, Visualization Scheme

1. Introduction

With the development of MEMS and wireless communication, sensor networks have become a hot research topic. Simulation is an effective approach for evaluating the performance of networks. A series of simulation tools have been developed to ease design and deployment of network protocols. Most of these existing simulators are established for traditional wired or wireless networks, so they cannot fulfill the requirements of modeling sensor networks precisely and running large-scale experiments efficiently. We will start with a brief overview on existing network simulators below.

1.1. Related Works

ns-2 [1] is a discrete event driven general-purpose network simulator originally developed for modeling the transport control protocols and routing algorithms of wide-area Internet. The CMU Monarch project's extension made ns-2 support the wireless and mobile networks. Although ns-2 has evolved substantially over the past few years, the basic architecture remains the same. Its split-programming model and *object-oriented* architecture make it extensible, but not so scalable.

OPNET [2] is another discrete event driven general-purpose network simulator. OPNET is featured by its GUI-based modeler, which provides an intuitive way to customize modules, like different sensor-specific hardware units. However, the GUI-based modeler sacrifices simulation efficiency for convenience of customization. Therefore, it suffers from the same problem of scalability as ns-2.

Other general-purpose simulation platforms, like OMNeT++ [3] and J-Sim [4], are also widely used in simulating the traditional wired and wireless networks. Since these two simulators were designed from the beginning with module reusability in mind, they put less emphasis on the scalability of simulator.

Based on the simulation framework for sensor network (SensorSim) proposed by Park [5], all the above mentioned general-purpose network simulators have been extended to support for sensor networks, but never been modified in terms of their architectures to address the problem of scalability.

The problem of scalability is more severe for bit-level (or instruction-level) emulators, like TOSSIM [6] and ATEMU [7], than for the above mentioned packet-level simulators.

1.2. Our Contributions

In this paper, we design and implement a new simulator, namely EasiSim, specifically for sensor networks with *scalability* as our first goal. Differing from traditional object-oriented and component-based simulators, EasiSim is a discrete event driven *structure-based* simulator. The main contributions of this paper can be summarized into following three aspects.

1) To enables the node to process all the concurrent events in one batch, we propose a three-dimension sorted linked list (3D list) to organize all the *nodes* into a hyper-structure called *topology*. In this way, the number of events generated during simulation will be reduced by an order of magnitude, thus to decrease the running time.

2) To ease extension of the simulator, we design a *scenario* structure to integrate topology with all other components of the simulator, such as discrete event queue and clock. The hierarchical organization of the components, including nodes, topology and scenario, makes EasiSim not only scalable but also extensible.

3) To visualize the progress of a simulation, we propose a *visualization* scheme based on a client-server model, which separate the graphical user interface (GUI) from other simulation modules, and make GUI and simulation engine run in a distributed way. Therefore, our proposed visualization scheme will not affect performance of the simulator in terms of running time.

The rest of the paper is organized as follows. We describe the component organization in Session 2. Details of integrating the components of the simulator are presented in Session 3. In Session 4, we elaborate on the visualization scheme of the simulator. We evaluate the performance of the simulator in Session 5. At last, we make a brief conclusion and point out future work in Sessions 6 and 7 respectively.

2. Component Organization

2.1. Basic Principle

We design our simulator based on a sequential, discrete event driven framework. As shown in Figure 1, the simulator mainly consists of following five components.

1) Clock is a 64-bit integer to represent the current time of the simulator.

2) Random number generator is a collection of functions to generate the commonly used random numbers, such as uniformly distributed numbers.

3) Entity is the object to be simulated. For a sensor network, the entity is nothing but a collection of nodes.

4) Discrete event queue is a sorted event list ordered by the time when the event is scheduled to be processed.

5) Event dispatcher is a procedure responsible for fetching the latest event to be processed from the head of the discrete event queue and invoking the corresponding event processing procedure.

The time flow and event flow link the components together. It is worth noting that: 1) the clock of the simulator should be updated to the scheduled time of the dispatched event, before invoking the corresponding event processing procedure; 2) in the event processing procedure, states of more than one related nodes may be required to be updated concurrently and some future events are supposed to be scheduled.

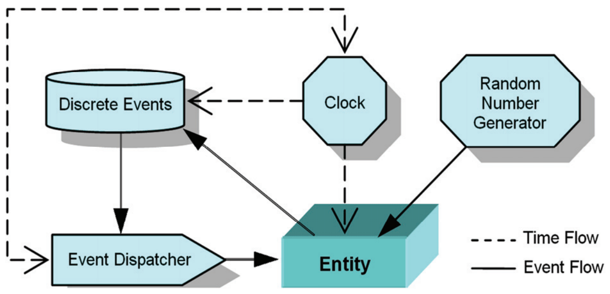

Figure 1. Component architecture of the sensor network simulator.

In the existing object-oriented and component-based network simulators, each node is typically modeled as an object. To update states of several nodes simultaneously, it is required to generate some concurrent events and invoke all the corresponding nodes' methods in sequence. This can lead to lots of overhead in terms of running time and memory usage. In wireless networks, packet transmission is such an event that requires updating the states of all the neighbors of the transmitting node, and it creates a large number of events which accounts for a large proportion of all the events in a sensor network. So we propose a new modeling method to establish an efficient structure to support updating several nodes' states in one batch.

We model each node in the sensor network as a structured variable rather than an object. Based on the node structure, we design a three-dimension sorted linked list to organize all the nodes into a hyper-structure called *topology*. We will describe the details of the node and topology structures next.

2.2. Node Structure and Topology Structure

Each node is modeled as a variable with the same structure, as shown in Figure 2. The foremost three fields refer to the type, identity and location of the node respectively. The next four fields contain all the necessary information about the settings and states of the protocol in each layer, which are followed by six pointers to link the nodes into a three-dimension sorted list (3D list). Each dimension is a doubly linked list sorted by the identity, the x-coordinate, and the y-coordinate of the node, respectively. This three-dimension sorted linked list is defined as a structure named topology. In the topology structure, it contains three pairs of pointers to indicate the head and tail of each dimension of the sorted linked

```
struct _node{
        NODETYPE nodeType;
        NODEID      nodeID;
        COORDINATE locate;

        PHYDATA phyData;
        MACDATA macData;
        ROUTEDATA routeData;
        APPDATA appData;

        struct _node *nextNodeByID;
        struct _node *preNodeByID;
        struct _node *nextNodeByX;
        struct _node *preNodeByX;
        struct _node *nextNodeByY;
        struct _node *preNodeByY;

        PTOPO pTopo;
};
```

Figure 2. Definition of the node structure.

list. The last field in the node structure is just the pointer to the topology, which provides the node with a convenient way to operate on the global information or any other node.

When initializing the simulator, each node is assigned a unique integer number as its identity, and a pair of coordinates as its location. According to the values of these attributes, the node is inserted into the three-dimension sorted linked list in ascending order, while the pointers in the topology structure keep track of the head and tail of each dimension of the list.

2.3. Advantages of the Organization

Suppose N nodes are uniformly scattered in a square area of S square meters, and the transmission range of each node is R meters. If the radio propagation is modeled as disk, in the traditional simulators, it is required to compute distances between the transmitting node and all the N nodes in the square area.

With support of the 3D list, when a transmitting event is scheduled during simulation running time, the transmitting node can be found in the list quickly by its identity, and then all its neighbors can be determined rapidly by traversing the list and calculating the distance (or signal attenuation) between the current node (receiving node) and the transmitting node because the list is sorted based on the physical locations of the nodes. Illustrated in Figure 3, the following steps are involved to determine the set of nodes affected by the propagation.

1) Locate the transmitting node in the list by any fast search algorithm.

2) Traversing forward and backward from the transmitting node along either the x-coordinate or the y-coordinate and comparing the x-coordinate and the y-coordinate of the current node in the list and the transmitting node to determine whether the node is located in the intersection region of the light gray area (marked as A) and the dark gray area (marked as area B). Because the list is sorted, only those nodes in the light shaded area are involved to compare their coordinates with the transmitting node. So the number of nodes involved in this step is reduced to $N\cdot\frac{2R\cdot\sqrt{S}}{S}=N\cdot\frac{2R}{\sqrt{S}}$.

3) Compute the distance from the transmitting node to each of the nodes in the intersection region of the light gray area and the dark gray area, to see whether it can hear from the transmitting node. The number of nodes involved in the step is reduced further to $N\cdot\frac{(2R)^2}{S}$.

Let T_1 be the benchmark of computation time, which is the time to do addition or subtraction operation. Multiplication operation time and relation operation time are p times and q times as much as the benchmark respectively, where both p and q are larger than one. Since it involves two subtraction operations and two comparison operations to determine whether one node is in the intersection area of A and B, the time to check all the nodes in the light gray area is $N\cdot\frac{2R}{\sqrt{S}}\cdot 2\cdot(1+q)\cdot \text{T1}$. For the same reason, we can know the time to determine the set of nodes that can hear from the transmitting node is $N\cdot\frac{(2R)^2}{S}\cdot\left[3(1+p)+q\right]\cdot \text{T1}$.

So the computational time to determine the neighbors of the transmitting node is reduced by a percentage of $\left[\frac{4R(1+q)}{\sqrt{S}(3+3p+q)}+\frac{4R^2}{S}\right]\times 100$ with adopting the 3D list. For example, the time can be reduced by about 86.1%, when p is 2 and q is 3.

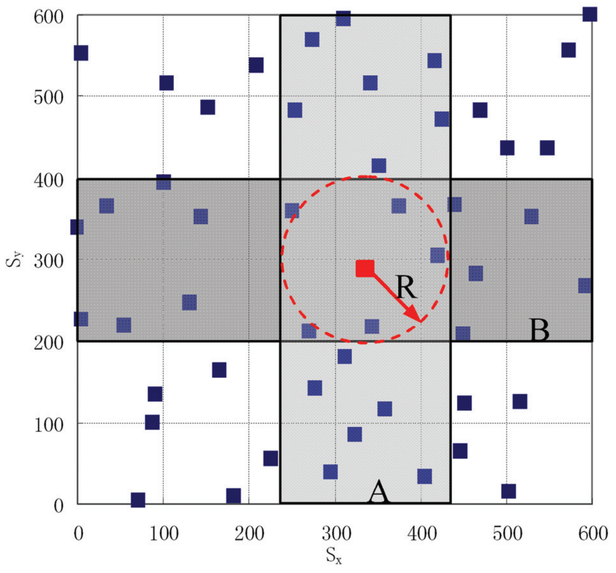

Figure 3. The set of nodes affected by the propagation.

3. Component Integration

To implement a discrete event driven simulator, we should integrate all the components together, mainly including discrete event queue and clock, besides nodes and topology. As described in the above section, we have integrated the nodes into the topology by organizing all the nodes in the network into a three-dimension sorted linked list. Each dimension is a doubly linked list, whose head and tail are indicated by a pointer respectively. In each node, there is also a pointer to the topology structure, which provides the node with a convenient way to operate on the other nodes.

As shown in Figure 4, the topology and other components, such as event queue and clock, are integrated into an upper-layer structure named scenario. The scenario structure mainly includes three fields, which are *topo*, *cur_time* and *fel*. The *topo* is the topology structure mentioned above, which keeps the pointers to the heads and tails of the three-dimension sorted linked list, and a pointer to the scenario. Through this pointer, nodes can access to all the data in the scenario directly. The *cur_time* is a 64-bit integer to indicate the current time of the simulator. *fel* is the event queue to organize all the events scheduled to be processed in the future. In the discrete event driven simulator, events are dynamically generated and released to drive the running of the simulator, which will involve lots of memory accesses, so the organization of the events should also be carefully designed.

3.1. Structure of the Event List

The traditional approach of managing the events is allocating memory once a new event is scheduled, and releasing it once it is processed. Since memory allocating operation is time consuming, we structure the event list as a 2-Dimension linked list to reduce the frequency of allocating memory. Each dimension is a sorted linked list. One is for organizing the future events (named scheduled list), and the other is for collecting the released events

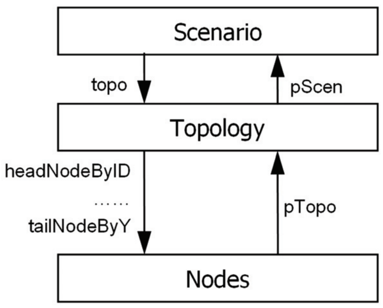

Figure 4. Architecture to integrate the components of the simulator.

(named free list). When a new event is to be generated, the free list is firstly checked to see whether there is a free event that can be "reused". If yes, the event will be renewed and moved to scheduled list, otherwise, a system call is invoked to allocate memory which is inserted into the scheduled list.

Supposing the instant number of events scheduled to be processed at time t during running time is N_t, the total times to invoke system calls for the events is $\max\{N_t\}$ with the aid of the free list, while it is $\mathrm{sum}\{N_t\}$ without the aid of the free list. The peak volume of memory allocated for the events are both $\max\{N_t\}$*L for the simulator, where L is length of the event.

So the structure of the event list not only reduces the frequency of allocating memory, but also inherits the merit of the traditional approach in memory usage.

4. Visualization of the Simulator

Visualization is also an important component of the simulator, though it is not indispensable. The traditional way to implement visualization is off-line replaying, like ns-2/Nam, which displays the flow of packets according to the trace file. Because the replaying depends on the trace file dumped during simulation, the *off-line* approach is time consuming and may influence the performance of the simulator in terms of scalability. Other tools integrate the graphic user interface with the simulation engine, which has bad effects on the scalability of the simulator.

We propose an *on-line* approach to show the progress of simulation by a separate process locally or remotely, based on a Client/Server model. A server process, which can be viewed as the graphic user interface (GUI) of the simulator, is firstly launched in the local machine or a remote machine. If the user requests to display the process of the packet flow or to visualize the states of the nodes, he can let the simulator connect with the server before starting running, by specifying the address and port on which the server is listening. During simulation, the server process is responsible for receiving and parsing the packets encapsulating the requests of visualization, which are generated and sent by the client.

For example, when the simulator finishes initialization, all the created nodes need to be displayed in the GUI. So the simulator will send a packet, like *Node sensor 0 100.0 200.0,* to the server. The first word *Node* in the packet tells the GUI to draw a node, and *sensor* indicates the type of the node. Different types of nodes, such as sensors and sink, may be illustrated by different shapes in the GUI. The number following the type of the node is its identity. When the server receives the packet, it will draw a circle or a rectangle at the coordinate of (100.0, 200.0) to represent the node. Other packets are also defined to visualize other objects in the simulator, such as radio propagation, link establishment and packet flowing.

In this way, the visualization of the simulator can be implemented without making significant modification to the established simulation engine. Since the GUI server can be run in a remote machine and the simulator can communicate with it in asynchronous way, the visualization will not affect the performance of the simulator.

5. Performance Evaluation

The performance of our proposed simulator, named EasiSim, has been evaluated in terms of the following metrics.

1) Real running time: real running time is the direct indicator of scalability. It is obvious that the real running time can be influenced by the *number of nodes* and the simulation time. So we will firstly examine the trend of change on real running time as the number of nodes increases, and then influence of the *configured simulation time* on the real running time will be examined.

2) Memory usage: total memory required to run simulation can also influence the scalability of the simulator, since more memory usage means more frequent operations on the system resource, which is very time consuming.

To compare its performance with ns-2, we first implement a unified suite of protocols, from the physical layer to the application layer, in both simulators. Details about how the protocols are designed and implemented have been presented in another paper [8]. Since the running time of simulation is directly related with the protocols implemented in the simulator, here we give a brief introduction of the protocols implemented by us, especially the MAC protocol and the routing protocol.

5.1. Protocol Implementation

5.1.1. Media Access Control (MAC) Protocol

Referring to the media control model integrated in the TinyOS [9], we designed a lightweight media access control protocol, named *TinyMAC*. Since the protocol should be refined further, we firstly established a simple model, which does not take the power saving mechanism into consideration. The finite state automaton of the protocol is shown as Figure 5.

From Figure 5, we can see that there are four states, namely idle (IDLE), initial back-offing (INIT-BO), congestion back-offing (CONG-BO) and transmitting (TRANS), existing in the automaton. The words beside the arcs, which are capitalized with C, stand for the conditions to trigger the change of the states. For example, supposing the current state is idle and a packet needs to be sent, if the current state of the physical layer is idle (that is C1), the node changes its state into initial back-

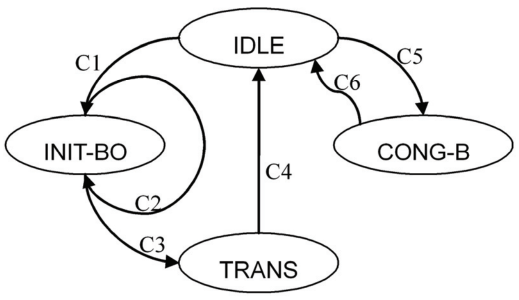

Figure 5. Finite state automaton of the TinyMAC protocol.

offing; otherwise (that is C5), the node starts congestion back-offing. When the congestion back-off ends (that is C6), it returns to idle and repeats the above described process. During initial back-offing, notification of state change (from idle to busy, that is C2) from the physical layer can stop its progress, but cannot make the current state changed. When the initial back-off ends (that is C3), the node starts transmitting the packet in the buffer. The node does not return to idle until the last bit of the packet is sent (this is C4).

In order to reduce the complexity of the media access protocol in the sensor network, we temporarily exclude the reliable transmission mechanism from the link layer. Referring to the media control model implemented in TinyOS, we adopt the following equations to determine the length of the initial back-off and congestion back-off.

$$InitBO = \mathrm{randInt}(0,31)+1 \quad (1)$$

$$CongBO = \mathrm{randInt}(0,15)+1 \quad (2)$$

The function of randInt(a, b) is defined to return a random number between a and b. So the maximum length of the initial back-off is 32 bytes, which implies about 13.3 milli-seconds when the channel bandwidth is 19.2 Kbps. For the same reason, the maximum length of congestion back-off is about 6.7 milli-seconds.

5.1.2. Routing Protocol

Routing protocols are considered specific for the sensor network, since it is suggested to provide support for data aggregation or fusion to reduce the volume of data and energy consumption [10]. In this paper, we don't put our efforts to design a data-centric or power-efficient routing protocol for the sensor network. Instead, we design a lightweight routing protocol based on flooding, which is referred to as *TinyFlood,* to evaluate performance of simulators.

Flooding is considered to be the simplest scheme to route data in the multi-hop network. But it is also well known that flooding can lead to inner explosion and overlapping, which can waste a lot of bandwidth and other network resources. To reduce the overhead resulting from the uncontrolled flooding, we make following improvements to avoid flooding loop and infinite re-forwarding of outdated packet.

One way to avoid looping packet in flooding is making the node remember the previously flooded packets. When an intermediate node receives a flooded packet, it firstly checks whether it has been flooded by the current node; if not, it will relay the packet by flooding and keep the packet in the memory; otherwise, it will drop the packet. For nodes in the sensor network, memory is severely constrained, so tracing the previously flooded packet seems impractical.

In *TinyFlood*, we make the packet carry the information of the route through which it has been flooded. When an intermediate node receives a flooded packet from the upstream, it checks whether itself has been included in the trace list of the packet; if not, add itself to the trace list and re-flood the packet to the downstream; otherwise, it will drop the received packet. Supposing the upper bound of the number of nodes in the sensor network is 65536, the length of identity of each node will not exceed 2 bytes. So the maximum length of the extra payload added to the packet depends on the TTL (Time To Live) configured by the user, which can be expressed as Equation (3). Through this improvement, the problem of flooding loop is solved without requiring large memory.

$$L_{max\_extra\_payload}=2\times \mathrm{TTL} \quad (3)$$

Considering the limited bandwidth of the sensor network, each packet cannot be so long, or else the network will always in burst state. Supposing the total packet size can not exceed 36 bytes, and the average length of data generated by the sensor node is 24 byes, it can be easily deduced from (3) that the TTL can not be more than 6.

Next, we run simulations to compare performance of our designed simulator with ns-2 in terms of the above mentioned metrics. Simulations were run on a Pentium-IV3.0 GHz processor with 1 Gbytes of RAM memory. The GUI ran in a remote machine.

5.2. Real Running Time versus Number of Nodes

The experiments were set up by putting 10 to 1000 nodes uniformly in a 1000 by 1000 meter square field. The transmission range of the node is 250 meters. The node in the left bottom corner is chosen to collect data and send the readings to the sink, which is in the right top corner of the field. The sensor nodes were configured to send the readings every 1 minute, and the simulation time is one hour. The length of the packet is 36 bytes, and the physical rate of the node is 19.2 kbps.

The running time is calculated as the difference between the time when the simulator finished initialization and the time when the simulation ends. All the results are the average of 5 runs with 95% confidence intervals.

As shown in Figure 6, for both EasiSim and ns-2, the running times are below 5 second when less than 100 nodes are put in the field. As more nodes are added to the scenario, the simulation times increase superlinearly. This can be attributed to the event explosion when the nodes become denser. However, the run time of EasiSim increases much more slowly than that of ns-2. This can be owed mainly to the efficient approach to merge the concurrent events described in Session 2.

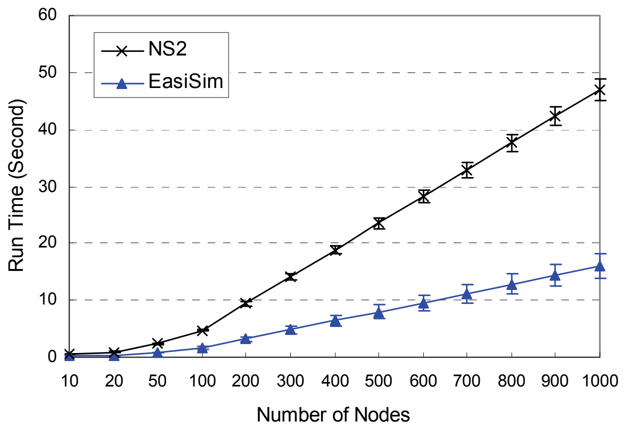

Figure 6. Real running time versus number of nodes.

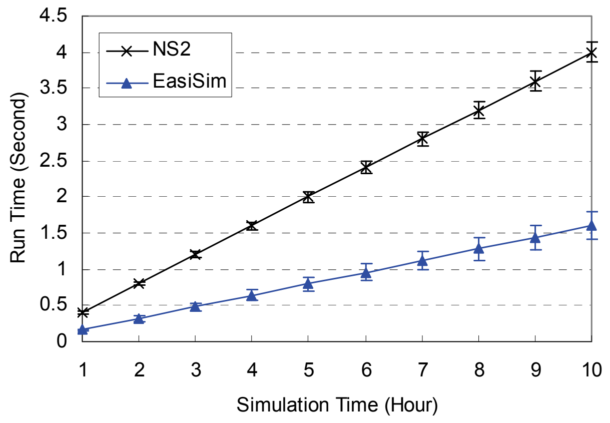

Figure 7. Real running time versus simulation time.

5.3. Real Running Time versus Simulation Time

In this experiment, we put 10 nodes uniformly in a 1000 by 1000 meter square field, and run the simulation with the same parameters as described in the former experiment for 1 hour to 10 hours.

As illustrated by Figure 7, the running times with both EasiSim and ns-2 rise with the increase of simulation time, because more events are generated to be processed. However, the running time of EasiSim is much less than that of ns-2. Since the profits gained from event merge can be neglected here, we can attribute the advantage of EasiSim to its structure based modeling method. Because all the data representing the state of each node is stored in a structured variable, rather than in an object, they can be accessed directly by the processing procedure without invoking other methods, the processing time can be reduced a lot.

5.4. Memory Usage

The setting up of experiments to evaluate the memory usage of the simulator is the same as we described in the above subsection. Here, we record the volumes of memory space allocated for the nodes and the events in the simulator. Figure 8 shows the results of the experiments. We can see that EasiSim is always more memory efficient than ns-2. The main reason leading to the result can be concluded as follows.

In ns-2, every component of the node is modeled by an object, and the components then comprise the node. Each object in ns-2 has a shadow in memory, so ns-2 needs twice more spaces than EasiSim to store the nodes in the network.

6. Conclusions

This paper presented a new simulator called EasiSim, for simulating sensor networks at large scales.

EasiSim is featured by the *structure-based* modeling method and the *hierarchical* organization of the components. As the fundamental components, the *node* structures are firstly organized into a three-dimension sorted linked list. Pointers to the head and the tail of each dimension of the 3D list are then organized into the hyper-structure called *topology*, through which all the nodes involved in the current event can be operated directly. In this way, some concurrent events can be merged and thereby the running time can be reduced by an order of magnitude. The topology structure is then integrated with other components of the simulator, such as the discrete event queue and the simulation clock, into the top-level structure named *scenario*.

We evaluate the scalability of our designed simulator in terms of *real running time* and *memory usage*. The results show that it takes less time and less memory for EasiSim than for ns-2 to complete simulations with the same number of nodes and configured simulation time.

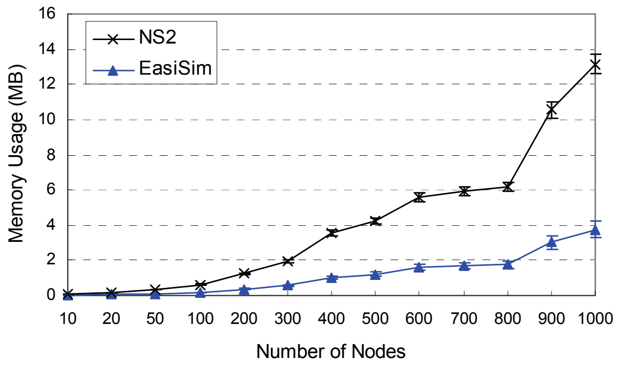

Figure 8. Memory usage versus number of nodes.

Furthermore, we proposed a visualization scheme based on a client-server mode, which enable the simulation and GUI processes to run in a distributed way. Therefore, our proposed visualization scheme does not decrease the performance of the simulator in terms of scalability.

7. Future Work

So far, we have established a scalable simulation platform for sensor networks. To evaluate the performance of the simulator, we also implemented the disk radio propagation module, the MAC protocol and the flooding protocol in the simulator.

As for our future works, we plan to extend the modules, including the radio channel modules, the environment modules and the networking protocol modules, to make the simulator support for modeling the sensor networks more precisely. A practical battery and energy module is also supposed to be implemented in the future days, since it is of vital importance for modeling the power efficiencies of different protocols and life time of the sensor nodes.

As more modules added to EasiSim, the scalability of the simulator will be reevaluated and its performance will be improved step by step. Besides that, the visualization scheme will be refined and its effects on the scalability of the simulator will be investigated more deeply.

8. Acknowledgements

This research is supported in part by the Chinese Academy of Sciences (CAS) - Croucher Funding Scheme for Joint Laboratories under Grant No. GJHZ200819, the National Basic Research Program of China (973 Program) under Grant No.2006CB303000, and the CAS Knowledge Innovation Program under Grant No. KGCX2-YW-110-3.

6. References

[1] The Network Simulator–ns-2. http://www.isi.edu/nsnam/ns.

[2] F. Desbrandes, S. Bertolotti, and L. Dunand. "OPNET 2.4: An environment for communication network modeling and simulation," In Proceedings of the European Simulation Symposium, pp. 609–614, Delft, Northerlands, October 1993.

[3] A. Varga. "The OMNeT++ Discrete event simulation system," In Proceedings of the European Simulation Multiconference (ESM'01), Prague, Czech Republic, June 2001.

[4] H. Tyan. "Design, realization and evaluation of a component-based compositional software architecture for network simulation," PhD thesis, Ohio State University, 2002.

[5] S. Park, A. Savvides, and M. B. Srivastava. "SensorSim: A simulation framework for sensor networks," In Proceedings of MSWiM'00, pp. 104–111, Boston, Massachusetts, USA, 2000.

[6] P. Levis, N. Lee, M. Welsh, and D.Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," In Proceedings of SenSys'03, pp. 126–137, November 2003.

[7] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J.S. Baras. "ATEMU: A fine-grained sensor network simulator," In Proceedings of SECON'04, pp. 145–152, October 2004.

[8] H. Chen, C. Huang, and L. Cui. "Lightweight protocol suite for wireless sensor networks: Design and evaluation," in Proceedings of IEEE ISCIT'07, pp. 1155–1160, Sydney, Australia, October 2007.

[9] A. Woo S. Hollar D. Culler J. Hill, R. Szewczyk, and K. Pister. "System architecture directions for networked sensors," In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00), pp. 93–104, November 2000.

[10] B. Kr ishnamachari, D. Estrin, and S. Wicker, "Modelling data-centric routing in wireless sensor networks," in Proceedings of the IEEE INFOCOM, 2002.