EasiSim: A Scalable Simulation Platform for Wireless Sensor Networks

Haiming Chen[12], Li Cui[1†], He Zhu[12]

[1]Institute of Computing Technology, Chinese Academy of Sciences

[2]Graduate School of the Chinese Academy of Sciences, Beijing, China

{chenhaiming, lcui, zhuhe}@ict.ac.cn

Changcheng Huang[3]

[3]Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada

huang@sce.carleton.ca

***Abstract*—Traditional simulators cannot meet the requirement of modeling large scale networks due to their deficiency in scalability. In this paper, we present a new simulator, namely EasiSim, for sensor networks on a large scale. EasiSim is featured by a *structure-based* modeling method and a *hierarchical* organization of the relevant functional components, including nodes, topology and scenario. The nodes are organized into a three-dimensional sorted list (3D list), which enables the node to process all the concurrent events in one batch, and therefore the running time may be reduced by an order of magnitude. Integrated with the other two upper layer components, which are topology and scenario, the proposed node organization method based on the 3D list makes the simulator not only scalable but also extensible. 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. The performance of EasiSim is evaluated through extensive simulations and compared with NS2 in terms of *real running time* and *memory usage*. The results show that EasiSim takes less time and less memory than NS2 to complete simulations with the same number of nodes during a same configured simulation time.**

I. INTRODUCTION

With the development of MEMS and wireless communication, the sensor network has become a hot research topic. Simulation is an effective approach, which has been widely adopted by the network researchers to evaluate the performance of networks. So far, a series of simulation tools have been developed to ease the 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 the sensor network precisely and running large scale experiments efficiently. Following is a brief overview on the existing network simulators.

A. 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' extension made NS support the wireless and mobile networks. Although NS has evolved substantially over the past few years, the basic architecture remains the same. The split-programming model and *object-oriented* architecture make it extensible, but no 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 unit. 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 problem of scalability.

Based on the simulation framework for the sensor network (SensorSim) proposed by Park [5], all the above mentioned general-purpose network simulators have been extended to support for the sensor networks, but never modified the architecture of the corresponding simulator 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.

B. Our contributions

In this paper, we design and implement a new simulator, namely EasiSim, specifically for the sensor network with the *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 such way, the number of events generated during simulation will be reduced by an order of magnitude.
2. To constitute the simulator in an extensible way, we design a *scenario* structure to integrate the topology with all the other components of the simulator, such as discrete event queue, clock. The hierarchical organization of the components, including nodes, topology and sce-

This paper is supported in part by the National Basic Research Program of China (973 Program) under Grant No. 2006CB303000, and National High Technology Research and Development Program of China (863 Program) under Grant No. 2007AA01Z2A9, and NSFC project under Grant No. 60572060.

† Corresponding author, Tel: +86-10-6260 0742, Fax: +86-10-6256 2701

nario, makes EasiSim not only scalable but also extensible.

3. To show the running progress of the simulation, we propose a *visualization* scheme based on a client-server model, which separate the graphical user interface (GUI) from the simulation modules and run them in a distributed way. Therefore, our proposed visualization scheme will not decrease the performance of the simulator in term of scalability.

The rest of the paper is organized as follows. We describe the node organization in Session II. Details of integrating the components of the simulator are presented in Session III. In Session IV, we elaborate on the visualization scheme of the simulator. We evaluate the performance of the simulator in Session V. At last, we make a brief conclusion and point out the future works in Session VI and VII respectively.

II. ORGANIZATION OF THE NODES

A. *Basic principle*

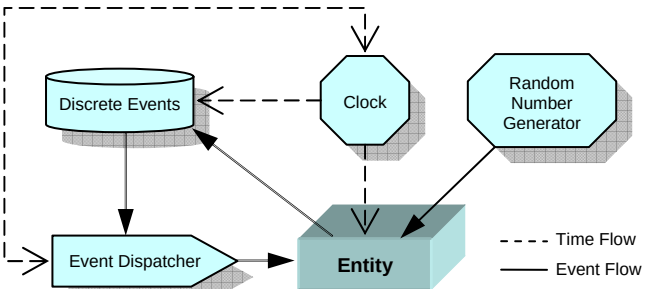

Fig. 1 Component architecture of the sensor network simulator

We design our simulator based on a sequential, discrete event driven framework. Like what the Fig.1 depicts, 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 noticing 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 new future events are supposed to be scheduled.

In the object-oriented and component-based network simulator, each node is modeled as an object. To update states of several nodes concurrently, it is required to generate some concurrent events and invoke all the corresponding nodes' methods in sequence. This can lead to lots of overheads in term of running time and memory usage. Radio propagation is such an event which needs update the states of all its neighbors, and it accounts for a large proportion of events in the 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*. Following are the details of the node structure and the topology structure.

B. *Node structure and topology structure*

```
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;
};
```

Fig.2 Definition of the node structure

Each node is modeled as a variable with the same structure as the Fig.2 shows. The foremost three fields refer to the type, identity and locate 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 couples of pointers to indicate the head and tail of each dimension of the sorted linked list. The last field in the node structure is just the pointer to the topology, which makes the node able to operate on the global information or any other node conveniently.

When initializing the simulator, each node in the sensor network is assigned a unique integer number as its identity, and put in a location expressed by the coordinates. 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.

C. *Advantages of the organization*

Supposing N nodes are uniformly scattered in a square area of S square meters, and the transmission range of each node is

R meter. If the radio propagation is modeled as disk, it is required to compute distances from the transmitting node to all the nodes in the square area in the traditional simulators.

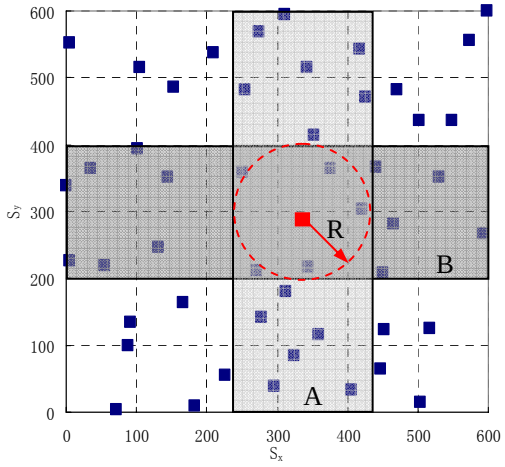

Fig. 3 The set of nodes affected by the propagation.

With support of the 3D list, when a propagation 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. Following steps illustrated in the Fig.3 are involved to determine the set of nodes affected by the propagation.

1. Locate the transmitting node in the list by any fast seeking algorithm.
2. Traverse forward and backward from the transmitting node and compare the x-coordinate and y-coordinate of the current node in the list and the transmitting node to determine the nodes in the intersection region of the light gray area (marked as A) and the dark gray area (marked as area B). Based on the sorted list, only these nodes in the light area are involved to compare its coordinates with the transmitting node. So the number of nodes involved in the 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 is p times and q times as much as the benchmark respectively. 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 $N\cdot\frac{2R}{\sqrt{S}}\cdot 2\cdot(1+q)\cdot T1$.

For the similar reason, the time to determine the set of nodes that can hear from the transmitting node is $N\cdot\frac{(2R)^2}{S}\cdot[3(1+p)+q]\cdot T1$.

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 the support of the 3D list. Take the above scenario as example, the time can be reduced by about 88.9%, when p is 2 and q is 3.

III. INTEGRATION OF THE COMPONENTS

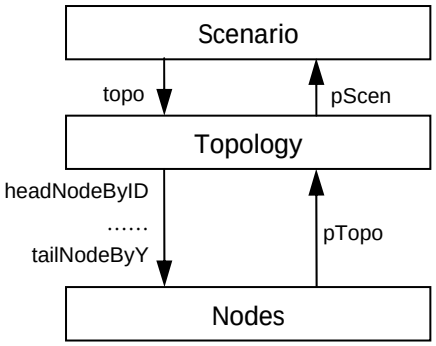

Fig. 4 Architecture to integrate the components of the simulator

The architecture to integrate the components of the simulator is presented in Fig.4. As the entity of the simulator, the topology is established 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 also a pointer to the topology structure, which makes the node operate on the other nodes conveniently.

Besides the topology, other components such as event list and simulation clock are integrated into an upper-layer structure named scenario. So the scenario structure mainly includes three fields, which are topo, cur_time and fel. It is worth noticing that in the topology structure, there exists 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 future event list 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 process, so the organization of the events should also be paid attention.

A. *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 (named freed list). When a new event is to be generated, the freed list is firstly checked to see whether there is a freed 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 and 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 freed list, and sum{N_t} without the aid of the freed 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 can not only reduce the frequency of allocating memory, but also inherits the merit of the traditional approach in memory usage.

IV. 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 NS2/Nam, which displays the flow of packets according to the trace file. Because the replaying depends on the trace file dumped during the 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 as well.

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 tell the simulator to connect the server before starting running by specifying the address and port on which the server is listening. During simulation running, 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 the packets formatting 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 in the coordinate of (100.0, 200.0) to represent the node. Other packet formats are also defined to visualize other objects in the simulator, such as radio propagation, link establishment and packet flowing.

In such 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 reduce the performance of the simulator a lot.

V. Performance Evaluation

The performance of the proposed simulator has been evaluated in terms of 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.

Performances of the EasiSim are compared with that of NS2 in terms of the above described metrics.

A. 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. All the nodes use the B-MAC (without sleeping) and route the data by flooding. Simulations were run on a Pentium-IV3.0 GHz processor with 1 Gbytes of RAM memory. The GUI ran in a remote machine.

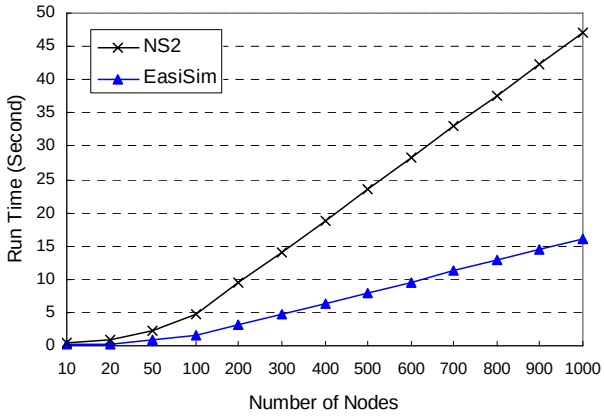

Fig. 5 Real running time versus number of nodes

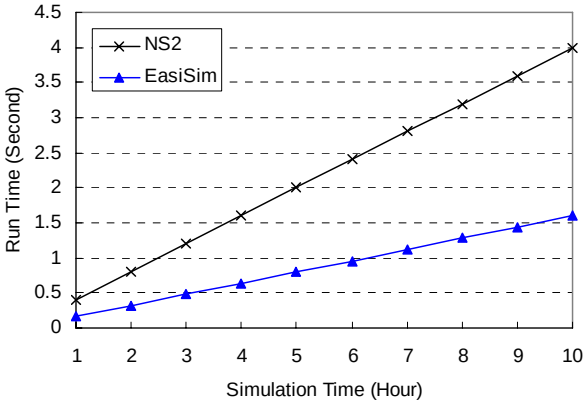

Fig. 6 Real running time versus simulation time

We compute the run time by recording the time when the simulator finished initialization, and when the simulator ends. All the results are the average of 5 repetitions to reduce random fluctuation. As Fig. 5 shows, for both EasiSim and NS2, 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 in higher linear rates. 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 NS2. This can be owed mainly to the efficient approach to merge the concurrent events described in session II.

B. *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 Fig.6, the running times on both EasiSim and NS2 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 NS2. Since the profits gained from event mergence 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 by a lot.

C. *Memory usage*

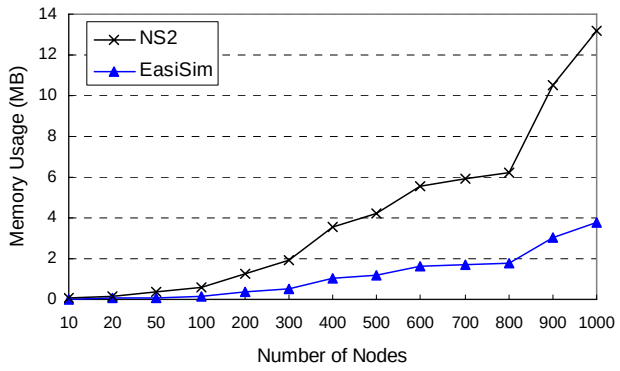

Fig.7 Memory usage versus number of nodes

The setting up of the experiments to evaluate the memory usage of the simulator is the same as what described in the part A of this section. Here, we record the volumes of memory space allocated for the nodes and the events in the simulator. Fig.7 shows the results of the experiments. We can see that EasiSim is always more memory efficient than NS2. The main reason leading to the result can be concluded as follows.

In NS2, every component of the node is modeled by an object, and the components then comprise the node. Each object in NS has a shadow in memory, so NS2 need twice more spaces than EasiSim to store the nodes in the network.

VI. 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 such 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*.

At last, 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 NS2 to complete simulations with the same number of nodes and configured simulation time.

In addition, 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 is supposed not to decrease the performance of the simulator in term of scalability.

VII. Future Works

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 B-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.

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, Nertherlands, Oct. 1993.

[3] A. Varga. “The OMNeT++ Discrete Event Simulation System,” In *Proceedings of the European Simulation Multiconference (ESM’01)*, Prague, Czech Republic, Jun. 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, 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.