.Accelerating the Discrete Event Network Simulation by Direct Computing

Xiaofeng Wang and Xiangzhan Yu

School of Computer Science and Technology, Harbin Institute of Technology, Harbin, *RPC*

Summary
To reduce the computation requirement of large-scale network simulation, this paper presents a direct computing method for discrete event network simulation. In this method, many of the discrete events are replaced by direct computing, so the computation overhead for network simulation is decreased, yet the accuracy of simulation result is kept unchanged. Experiments show, compared with the traditional discrete event network simulator, the direct computing method can decrease the number of discrete events generated during the network simulation by over 50%, and can decrease the simulation running time by over 40%, without the loss of simulation accuracy; the memory required during simulation is also decreased.

Key words:
Discrete event simulation; network simulation; number of discrete events; direct computing

1. Introduction

Network simulation provides a technique platform for doing research on network behavior, and for evaluating network protocols, and it is of great value in science research and application. The predominating network simulators are realized based on discrete event simulation technology, which means the state of the simulation system changes only when a discrete event occurs. With the rapid development of computer networks, the size of topology for simulation grows to a very large scale, yet the amount of computation time required to perform large-scale discrete event network simulation is prohibitive [1]. There are two kinds of methods to attack this problem. One is to use PDES to realize the parallel network simulation [2]. The other is to increase the abstraction level of network simulation by establish more abstract models [3].

The purpose of this article is to reduce the running time for large-scale network simulation by introducing a more abstract model for discrete event simulation. This article brings forward a method for accelerating the discrete event network simulation by direct computing (DC). The DC method obtains part of the changes of simulation system state via direct computing, yet without loss of simulation accuracy. As the traditional discrete event network simulator (such as NS[4]) obtains all the changes of simulation system state via discrete events, this method can cut down the number of discrete events and reduce the simulation running time compared with the traditional method.

This article is organized as follows. In the next section, we present the related work about increasing the simulation abstraction level. In section 3, we put forward our DC method in detail. In section 4, we carry out experiments to demonstrate the accuracy and efficiency of the DC method. Section 5 concludes the paper and points out the future works.

2. Related Work

Increasing the simulation abstraction level is a key method to reduce the computation overhead of large-scale network simulation. Huang puts forward in her Ph.D thesis [3] several abstraction techniques for accelerating the network simulation and improving the scale of network simulation. One of the abstraction techniques is called end-to-end packet delivery, which means the packet for simulation is sent directly from the source node to the destination node and the queuing delay of the packet is ignored. The end-to-end techniques can cause an increasing loss of accuracy when the degree of congestion grows. In [5] and [6], a hybrid method of discrete event simulation and epidemiological model is developed for doing research on network worms. Although this method can cut down the number of discrete events and simulation running time greatly, the epidemiological model can reduce the accuracy of simulation. In [7~10], the fluid models are presented to reduce the simulation running time, but the fluid models can't assure the accuracy of each packet hop and are usually used to simulate background traffics. In [11], the complexity of network simulation is reduced by cutting down the number of hosts in the simulated topology, yet this method can cause the inaccuracy of packet loss rate. By combining sampling, modeling and simulation, [12] takes a sample of the network traffic for simulation, feeds it into a suitably scaled version of network, and then extrapolates from the performance of the scaled network to that of the original. This method can keep the accuracy of several statistical parameters of network simulation, yet it can't assure the accuracy of each packet hop.

As we can see, the methods mentioned above reduce the computation time of simulation at cost of accuracy, and they can't assure the accuracy of each packet hop. Our DC

This paper was supported by the National Basic Research (973) Program of China (2005CB321806)
Manuscript received May 5, 2007
Manuscript revised May 20, 2007

method can assure the accuracy of each packet hop, and by replacing discrete events with direct computing, it can reduce the computation time considerably.

3. The DC Method

3.1 Traditional Packet Transmission Simulation

NS is a good example of discrete event network simulation and has been widely used for network research [3], so we choose NS as our research platform. In NS, simulation of packet transmission can be depicted by Fig. 1. In Fig. 1, the simulation procedures of the packet transmission from node *N(0)* to node *N(1)* via "simplex link model" are listed as follows:

1) When a packet is delivered from *N(0)* to the "simplex link model", the model will first decide whether the packet should be dropped or not. If the packet is dropped, it should be delivered to the "Agent/Null" module; while, if not, the packet should be enqueued to the "Queue" module, and then wait for being processed.

2) When it is time to process the packet, which is triggered by an event previously inserted into the event list, the packet will be dequeued from the "Queue" module, and then delivered to the "Delay" module.

3) The "Delay" module firstly computes the packet's fixed delay, including propagation delay and transmission delay. Based on the fixed delay, an event ("Event 1" in Fig. 1) that will trigger to continue to process the packet is generated accordingly and inserted into the event list, and based on the transmission delay, another event ("Event 2" in Fig. 1) that will trigger to process the next packet in the "Queue" module is also generated and inserted into the event list.

4) When it is time to process "Event 1", the packet should be delivered to the "TTL" module. In the "TTL" module, the TTL of the packet is decreased by 1. If the TTL is then equal to 0, the packet should be dropped; while, if not, the packet can be delivered to *N(1)*.

5) When it is time to process "Event 2", this event will trigger the next packet to be dequeued from the "Queue" module, and then to be processed according to procedure 2) to 5).

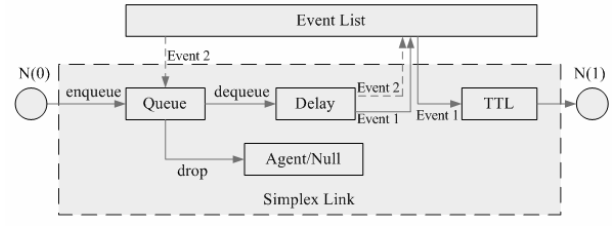

Fig. 1 Simplex link model in NS

3.2 Basic Idea of Direct Computing

From the analysis in the above section, we can see that the traditional simulation of one packet hop needs two discrete events. For large-scale network simulation, there are millions of packet hops for simulating, so the number of discrete events is enormous. Meanwhile, there is enqueue and dequeue operation for each packet hop, which also increases the computation overhead. For the sake of decreasing the number of discrete events and simplifying the simulation procedure of packet hop, we bring forward the DC method. Its basic idea is: simulation of packet hop doesn't need enqueue and dequeue operation; the queueing delay of packet is computed directly, and is not obtained by discrete events. As shown in Fig. 2, the DC method's simulation procedures of the packet transmission from *N(0)* to *N(1)* are described as follows:

1) When a packet is delivered from *N(0)* to the "simplex link model with direct computing", the model will first decide whether the packet can be directly sent to the "Abstract-Queue" module or not. If the packet can be directly sent, turn to procedure 4).

2) An event ("Event 3" in Fig. 2) that will trigger to continue to process the packet is generated and inserted into the event list.

3) When it is time to process "Event 3", the packet should be delivered to the "Abstract-Queue" module.

4) The "Abstract-Queue" module receives the packet, and it decides whether the packet should be dropped or not. If the packet is dropped, it should be delivered to "Agent/Null" module; while, if not, the queueing delay of the packet is computed directly, and the packet is then sent to the "Abstract-Delay" module.

5) The "Abstract-Delay" module computes the packet's fixed delay, including propagation delay and transmission delay. The total delay of the packet from *N(0)* to *N(1)*, which includes propagation delay, transmission delay and queuing delay, is recorded, and the packet is sent to the "TTL" module.

6) In the "TTL" module, the TTL of the packet is decreased by 1. If the TTL is then equal to 0, the packet should be dropped; while, if not, the packet can be delivered to *N(1)* with the total delay information.

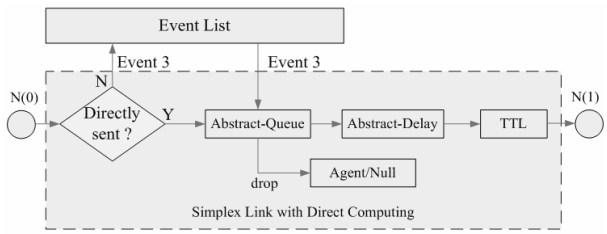

Fig. 2 Simplex link model with direct computing

Compared with the simplex link model, the direct computing link model does not have the "Queue" module for storing packets and the simulation of packet hop doesn't need the enqueue and dequeue operation; meanwhile, there are fewer discrete events, which are time-consuming as they should be inserted into and extracted from the events list. So, compared with the traditional discrete event network simulation, the DC method can reduce the computation overhead considerably.

When a packet *p* arrives at a direct computing link model *L*, *L* should first decide whether *p* can be directly sent or not: if every packet whose arrival time to *L* is smaller than *p* has been processed by *L*, *p* can be sent directly to the "Abstract-Queue" module; otherwise, *p* can't be sent directly. This condition is used to ensure that, all the packets which pass through the same link model can be processed in time order, and accordingly to assure the accuracy of simulation result. Without this condition, that is, in Fig. 2, all the packets coming from *N(0)* is directly sent to the "Abstract-Queue" module, the DC method can't assure the accuracy of simulation result (Fig. 3 shows us an example).

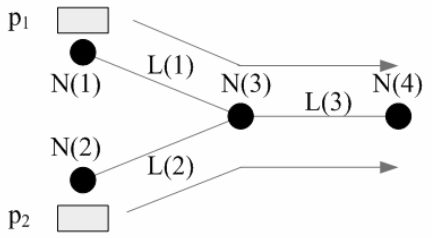

Fig. 3 Example of inaccuracy

In Fig. 3, there are packet flows from node *N(1)* to node *N(4)*, and from node *N(2)* to node *N(4)*. Suppose that the packet p_1 is generated by *N(1)* and should be sent to *N(4)*, and that the packet p_2 is generated by *N(2)* and should be sent to *N(4)*. Let t_1 denote p_1's generation time, t_2 denote p_2's generation time, t_3 denote p_1's arrival time to *N(3)*, and t_4 denote p_2's arrival time to *N(3)*. Suppose that $t_1 < t_2$, and the DC method will first process p_1: p_1 will be delivered to *N(3)*, and then be delivered from *N(3)* to *N(4)*. Having processed p_1, the DC method will then process p_2: p_2 will be delivered to *N(3)*, and then be delivered from *N(3)* to *N(4)*. Suppose that the propagation delay of link *L(1)* is much larger than that of link *L(2)*, so $t_3 > t_4$, that is, p_2 arrives at *N(3)* before p_1. Yet the direct computing link model between *N(3)* and *N(4)* processes p_1 before p_2. So, p_1 and p_2 can't be processed in time order, which may lead to inaccuracy in simulation.

If the packet can be directly sent, it will be sent to the "Abstract-Queue" module directly; while, if not, an event that triggers to continue to process the packet will be generated and inserted into the event list.

Clearly, in Fig. 2, a practical condition that the packet can be directly sent is that the degree of node *N(0)* is no greater than 2. If it is, the packet can be directly sent to the "Abstract-Queue" module without worrying about the inaccuracy of simulation result.

3.3 The Direct Computing Model

The key of the DC method is the direct computing link model, which contains two parts: parameters of the direct computing link model and the direct computing algorithm. We explain these two parts in detail as below.

The main parameters of the direct computing link model are listed as follows:

1. Bandwidth of the link (*B*);
2. Propagation delay of the link (*D*);
3. Queue length in bytes (*L*);
4. Queue length in packets (*C*);
5. Array of the lengths of packets in the queue (*P[]*);
6. The latest updating time of the queue (*T*).

Among these parameters, *B* and *D* are kept unchanged throughout the simulation, but *L*, *C*, *T* and *P* are updated constantly. If a packet is delivered to the link model and this packet should not be dropped, the four parameters should be updated accordingly. According to the value of *C*, we can decide whether the packet should be dropped or not, and according to the value of *L*, we can get the queuing delay of the packet. To update *C*, the information stored in the array *P* is needed. The array *P* contains the lengths of the packets currently in the queue.

Suppose that the packet *p* arrives at node *N(0)*, and is expected be delivered to the next node *N(1)* through link *L*, the direct computing algorithm for this packet hop can be described as follows.

The direct computing algorithm:

Parameters: l_p, t_a, *degree*, *B, D, L, C, P, T, max,* d_p /* length of *p*, arrival time of *p*, degree of *N(0)*, bandwidth of *L*, propagation delay of *L*, queue length in bytes, queue length in packets, array of the lengths of packets in the queue, the latest updating time of the queue, the maximum length of queue (DropTail mechanism), total delay of *p**/

// Check whether *p* can be directly sent or not:
1. if (*degree* ≤ 2) {
   // *p* can be directly sent:
   // Update the values of *L* and *C* of the link model:
2. $L = L - (t_a - T) \times B$;
3. if ($L < 0$) $L = 0$;
4. $T = t_a$;
5. According to the information stored in the array *P* and the

current value of L, get the value of C, which satisfies the following equation:

$$\sum_{i=1}^{C} l_i \le L < \sum_{i=1}^{C} l_i$$

, where l_i is the length of the last i^{th} packet of all the packets that arrive at *N(0)* before time t_a, and should be delivered to *N(1)* through link L; the value of l_i is stored in array P;

```
6.      Update the information stored in array P;
      //Decide whether the packet p should be dropped or not:
7.      if (C > max - 1) {
          // p is dropped.
8.        free(p);
9.        return;
10.     } else {
          // p is not dropped.
          // Get the queueing delay of p:
11.       d_q = L/B;
          // Update the values of L, C and P:
12.       C ++;
13.       L = L + l_p;
14.       Save l_p to array P;
          // Get the transmission delay of p:
15.       d_t = l_p/B;
          // Get the total delay of p:
16.       d_p = d_q + d_t + D;
17.       Get the current TTL of the packet p → ttl_p;
18.       if (ttl_p – 1 ≤ 0) {
            // p is dropped.
19.         free(p);
20.         return;
21.       } else {
            // send p to N(1):
22.         Decrease the TTL of the packet p by 1;
23.         Deliver p to N(1), and the time at which p arrives to
N(1) is t_a + d_p;
24.       }
25.     }
26. } else {
      // p can not be directly sent:
27.   Generate the event that will trigger to continue to process
the packet and insert it into the event list.
28. }
```

The direct computing algorithm first decides whether p can be directly sent or not according to the degree of *N(0)*. If p can be directly sent, line 2 to 14 in the algorithm is the processing procedure of p in the "Abstract-Queue" module; line 15 and 16 is the processing procedure of the "Abstract-Delay" module; line 17 to 23 is the processing procedure of the "TTL" module. If p can't be directly sent, an event will be generated. This event will be processed when the virtual time of network simulator shifts to t_a. Actually, the processing procedure triggered by this event is line 2 to 23 in the algorithm.

4. Experiments and Results

In this section, we do a series of simulation experiments to validate the efficiency of our DC method.

The hardware platform for the experiments is a machine with two 1.0GHz CPUs and 4GB RAM. We realize our DC method on the traditional network simulator NS. As the SNOOPy mechanism shows better performance than others [13], we use the SNOOPy mechanism as the event list mechanism in the network simulator.

We do the simulation of slammer worm propagation [14], and compare the performance of the traditional NS and the improved NS with our DC method (NS-DC). The topologies for slammer worm propagation simulation are generated by the Nem topology generator [15]. By the Nem, we generate 5 network topologies with 1000, 2000, 3000, 4000, 5000 nodes respectively. Each node whose degree is 1 is set as the infectious host. The scan rate of the slammer worm is 10 times per second. There is 1 infected host at the first second, and the simulation time of slammer worm propagation is 100 seconds.

The experiments results show that NS-DC has the same simulation results with NS, that is, the direct computing method does not decrease the accuracy of simulation. We select the simulation results of 5000 nodes' topology, and show the comparison between NS and NS-DC in Fig. 4. We can see that the simulation result, that is, the number of infected hosts during the worm propagation simulation, is the same.

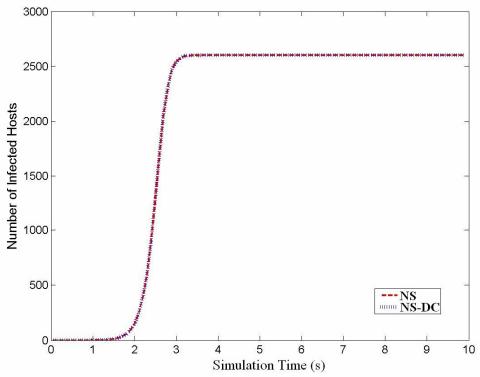

Fig. 4 Comparison of the number of infected hosts (5000 nodes)

Fig. 5 compares the number of discrete events generated during simulation. Experiments show, compared with NS, NS-DC can reduce the number of discrete events considerably (51.1% in average). More discrete events mean more computation overhead, so from Fig. 5, we can see that the direct computing method can reduce the computation overhead of network simulation considerably.

Fig. 6 compares the running time of simulation. As there are less discrete events for NS-DC, the running time

of NS-DC is decreased accordingly. Compared with NS, NS-DC can decrease the simulation running time by 40.1% in average. Comparison of running time shows that our direct computing method can improve the performance of the traditional network simulator NS efficiently.

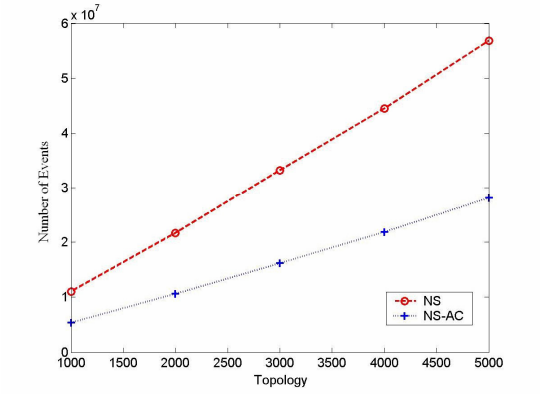

Fig. 5 Comparison of the number of events

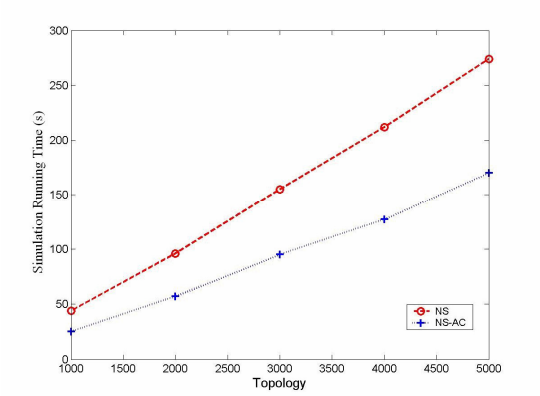

Fig. 6 Comparison of the simulation running time

Fig. 7 compares the memory overhead. During simulation, the whole routing table is maintained, whose memory overhead is $O(N^2)$ (N is the number of nodes in topology). Although the DC method can reduce the memory overhead, it is not clear in Fig. 7 as memory for routing table is large. Fig. 8 shows the absolute reduction.

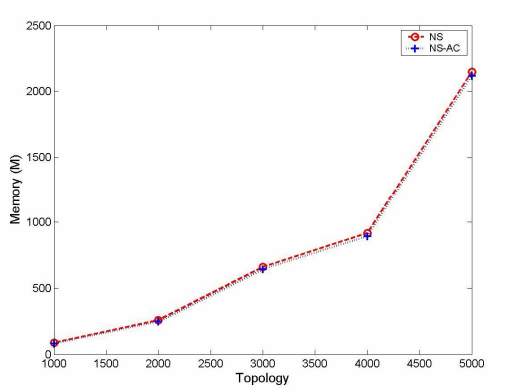

Fig. 7 Comparison of the memory overhead in simulation

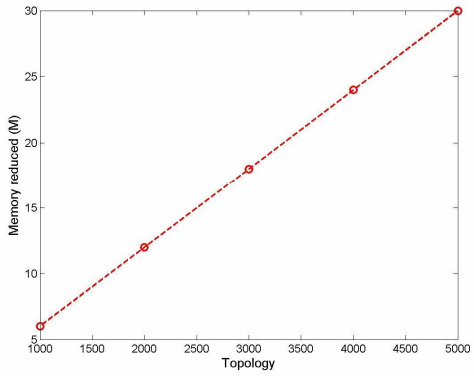

Fig. 8 Absolute reduction of the memory usage

From Fig. 4 to Fig. 8, we can see that, compared with the traditional discrete event network simulator, our DC method can reduce the number of discrete events, the simulation running time, and the memory required during simulation, while keep the accuracy of simulation result.

5 Conclusions and Future Works

We have presented a direct computing method for discrete event network simulation. In this method, the packet hop is depicted by direct computing, that is, the queueing delay of packet is computed and then the packet is sent to the next node directly; while in the traditional network simulator, the packet can't be sent to the next node directly and should be enqueued to and dequeued from the "Queue" module. Compared with the traditional network simulator, our direct computing method can decrease simulation running time considerably.

In this paper, we only realize the direct computing algorithm with DropTail mechanism. Now we are realizing the direct computing algorithm with RED mechanism.

References

[1] S. Floyd, V. Paxson. Difficulties in Simulating the Internet. IEEE/ACM Transactions on Networking. 2001, 9(4): 392~403

[2] R. M. Fujimoto. Parallel Discrete Event Simulation. Communications of ACM. 1990, 33(10):30~53.

[3] P. Huang. Enabling Large-scale Network Simulations: A Selective Abstraction Approach. Ph.D thesis, University of Southern California. 1999:51~54

[4] K Fall, K Varadhan. The NS Manual. http://www.isi.edu/nsnam/ns/doc/

[5] M. Liljenstam, D. Nicol, V. H. Berk and R. S. Gray. Simulating Realistic Network Worm Traffic for Worm Warning System Design and Testing. In Proceedings of the 2003 ACM workshop on Rapid Malcode (WORM'03), Washington, DC, USA, 2003:24~33

[6] M. Liljenstam, Y. Yuan, B. J. Premore and D. Nicol. A Mixed Abstraction Level Simulation Model of Large-Scale Internet Worm Infestations. In Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, Fort Worth, Texas, USA, 2002:109~116

[7] Y. Liu, F. L. Presti, V. Misra, D. Towsley and Y. Gu. Fluid Models and Solutions for Large-Scale IP Networks. In Proceedings of the 2003 ACM SIGMETRICS, San Diego, CA, USA, 2003:91~101

[8] Y. Gu, Y. Liu and D. Towsley. On Integrating Fluid Models with Packet Simulation. In Proceedings of IEEE INFOCOM 2004, Hong Kong, China, 2004:2856~2866

[9] F. Baccelli, D. Hong. Flow Level Simulation of Large IP Networks. In Proceedings of the Conference of the IEEE INFOCOM 2003, San Francisco, CA, 2003: 1911~1921

[10] B. Melamed, S. Pan and Y. Wardi. HNS: A Streamlined Hybrid Network Simulator. ACM Transactions of Modeling and Computer Simulation. 2004, 14(3): 251~277

[11] K. Below, U. Killat. Reducing the Complexity of Realistic Large Scale Internet Simulations. In Proceedings of IEEE GLOBECOM, San Francisco, USA, 2003: 3818~3823

[12] R. Pan, B. Prabhakar, K. Psounis and D. Wischik. SHRiNK: A Method for Enabling Scaleable Performance Prediction and Efficient Network Simulation. IEEE/ACM Transactions on Networking. 2005,13(5):975~988

[13] K. L. Tan, L.-J. Thng. SNOOPy Calendar Queue. In Proceedings of the 2000 Winter Simulation Conference, Orlando, Florida, 2000:487~495

[14] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford and N. Weaver. Inside the Slammer Worm. IEEE Magazine of Security and Privacy. 2003,1(4):33~39

[15] D. Magoni. Nem: A Software for Network Topology Analysis and Modeling. In Proceeding of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, Fort Worth, TX, USA, 2002:364~371

Xiaofeng Wang received his M.S. degree in communication engineering from the Harbin Institute of Technology in P.R.China in 2003. Since 2003, he has been a Ph.D. degree candidate in computer science from the Harbin Institute of Technology in P.R.China. His current research interests include network simulation, network security.

Xiangzhan Yu received the M.S. and Dr. Degree in Computer Science from Harbin Institute of Technology in 1997 and 2005 respectively. He has been working in HIT since 1997 and been an associate Professor in Computer science of HIT since 2005. His research interests include Disaster Tolerance, Network Security.