Reducing the Size of Routing Tables for Large-scale Network Simulation

Akihito Hiromori, Hirozumi Yamaguchi†, Keiichi Yasumoto‡,
Teruo Higashino† and Kenichi Taniguchi†

Graduate School of Engineering Science, Osaka University
1-3 Machikaneyamacho, Toyonaka, Osaka 560-8531, JAPAN
hiromori@ics.es.osaka-u.ac.jp
TEL: +81-6-6850-6607 FAX: +81-6-6850-6609

† Graduate School of Information Science and Technology,
Osaka University
1-3 Machikaneyamacho, Toyonaka, Osaka 560-8531, JAPAN
{h-yamagu,higashino,taniguchi}@ist.osaka-u.ac.jp
TEL: +81-6-6850-6607 FAX: +81-6-6850-6609

‡ Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5 Takayamacho, Ikoma, Nara 630-0192, JAPAN
yasumoto@is.aist-nara.ac.jp
TEL: +81-743-72-5251 FAX: +81-743-72-5259

December 16, 2002

Abstract

In simulating large-scale networks, due to the limitation of available resources on computers, the size of the networks and the scale of simulation scenarios are often restricted. Especially, routing tables, which indicate the directions to forward packets, are considered to consume memory space. A simple routing table requires $O(N^2)$ space where N is the number of nodes. An algorithmic routing recently proposed by Heung et al. only requires $O(N)$ space for representing routing tables, however this can be applied in the case that all the routes between two nodes are contained in a spanning tree (i.e. very limited routing strategies are allowed). In this paper, we propose a new method to reduce a capacity of routing tables under any routing strategy. In our method, given a simple routing table, we can find a (near-optimal) algorithmic routing based table that can represent a part of the given routing table in a smaller size. Our experimental results have shown that our method could reduce about 90 % of the routing table size compared with a simple routing table in hierarchical networks.

current node	destination node	neighbor node
1	2	2
⋮	⋮	⋮
1	n	2
2	1	1
⋮	⋮	⋮
n	$n-1$	$n-1$

Figure 1: General Routing Table

1 Introduction

In recent years, network simulation has played an important role to evaluate the performance of network protocols. Network simulators simulate the behavior of network-layer routers (simply say *nodes* hereafter), packets delivered between them and upper-layer components such as servers and clients. It enables us not only to measure the performance of existing network protocols on various network architectures but also to design and validate brand-new protocols.

One of the major problems in simulating large-scale networks is that the size of networks allowed depends directly on the capabilities of computers on which simulators run. To overcome this essential problem, two methodologies have mainly been investigated to improve the efficiency of network simulation, *i.e. parallelization of simulation* and *abstraction of networks*. A good example of the first category can be found in a simulator tool GlomoSim[1], which allows parallel execution of its simulation engine on multiple computers. On the other hand, in the second category, network topologies are "abstracted" to decrease the necessary computing power to simulate the behavior of nodes. For example, network simulator ns-2[2] has a functionality to replace a path consisting of multiple links with a single link whose link delay is equal to the end-to-end delay of the original path so that nodes on the path need not be simulated. This may greatly be effective in evaluating end-level behavior, however, may not be applicable in the case of link-level measurement.

The major factor that makes a network simulator consume computing resources (memory capacity in particular) is that the size of the routing table of a network to be simulated often becomes huge. Most network simulators, before starting simulation, setup and keep simple but huge routing tables where a packet's outgoing link at a node is directly obtained from its destination node, in order for fast processing. An example of this routing table is shown in Fig. 1. Each column (entry) consists of three terms which indicate the names of a current node where a packet arrives, a destination node of the packet and a neighbor node of the current node to which the packet is forwarded. Obviously, in this table structure, looking up a neighbor node is $O(1)$ for a given pair of a current node and a destination node while the size of the table is $N \times (N-1) = O(N^2)$ where N is the number of nodes.

Recently, an alternative methodology has been investigated that reduces the sizes of routing tables used in network simulators[3, 4, 5]. Especially, Huang et al. have proposed *algorithmic routing approach* [5] that greatly reduces the size of routing tables in a special case where any route between two nodes is covered in a unique spanning tree. In this case, instead of a general routing table explained in Fig. 1, a simulator may keep a spanning tree as a routing table whose size is only $O(N)$ while a lookup operation needs $O(logN)$. In

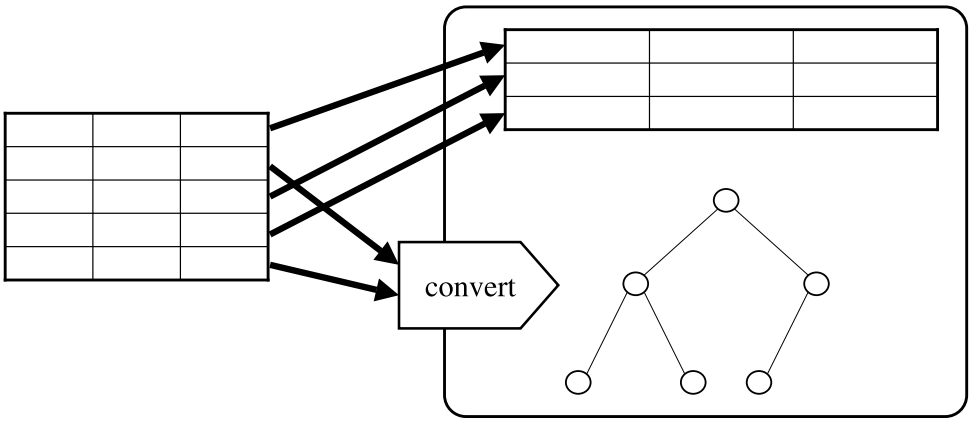


Figure 2: Concept of Proposed Method

Ref. [5], they have also shown that on typical network topologies, almost routes in a unique spanning tree still keep minimum hops. This means that the algorithmic routing method is effective in network simulation assuming minimum hop based routing.

In this paper, we extend the algorithmic routing method to allow generic routings, keeping the size of routing tables as small as possible and lookup speed as fast as possible. The main idea is as follows. Given a network to be simulated and a general routing table of the network as shown in Fig. 1, our method represents the table as the combination of a spanning-tree based routing table and a general routing table, by translating a part of the given general routing table into the spanning-tree based one (see Fig. 2). The size of our routing table depends on how many entries in the given general routing table can be covered by the spanning-tree based routing table. For this purpose, we present a heuristic algorithm that can find a near-optimal spanning tree in polynomial time complexity.

The problem to find an optimal spanning tree is formulated as an enumeration problem to seek all the possible spanning trees and then compute the number of entries in the given general routing table covered by each tree. Here, since the number of possible spanning trees exponentially increases as the number of links on the given network increases, this problem is considered as a combinatorial optimization problem whose complexity is considered as NP-hard. Our experimental results have shown that spanning trees found by our heuristic algorithm could cover about 90 % of the entries in the given routing tables on hierarchical networks.

The rest of the paper is organized as follows. In Section 2, we summarize the algorithmic routing proposed in [5] which is the basis for our method. In Section 3 we overview our method and its details are explained in Section 4. Section 5 gives experimental results and Section 6 concludes the paper.

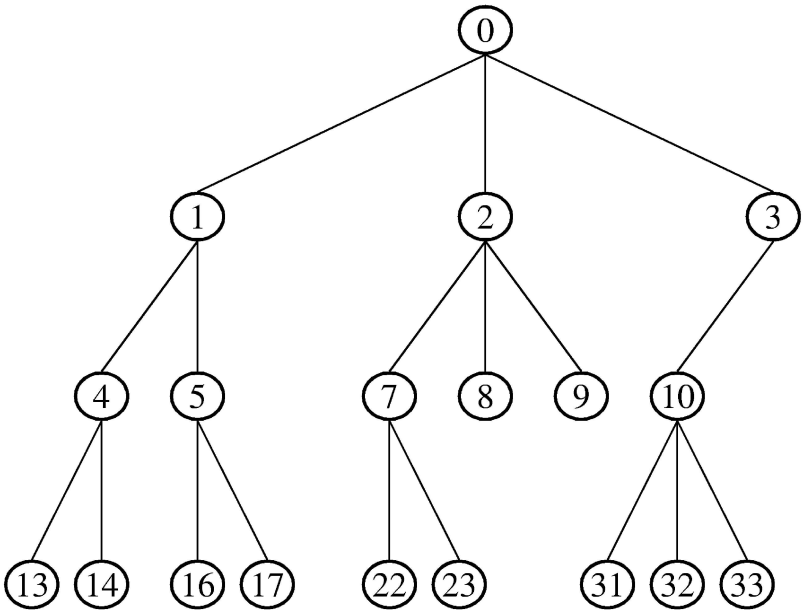


Figure 3: Example Spanning Tree

2 Algorithmic Routing Approach

The algorithmic routing approach proposed in [5] reduces the size of routing tables by giving special node numbers to all the nodes in the case that any route between two nodes is included in a unique spanning tree. Using such special node numbers, the neighbor node to which a packet is forwarded can be determined from the current node that the packet arrives and the destination node of the packet. In Fig. 3, we show an example spanning tree whose nodes have such special node numbers. In this example, the node number 0 denotes the root node. We assume that each node with node number n has at most k children nodes whose node numbers are $n \times k + 1$,..., $(n + 1) \times k$. Assuming this numbering rule, we can calculate each node's parent/children nodes' node numbers algorithmically. As a result, each node does not need to keep the routing table that directly contains neighbor nodes information. That is, we can calculate such a neighbor node on-the-fly algorithmically, although we need some computation time for calculating it. On the other hand, we can reduce the size of the routing table into $O(N)$ by only keeping the connection relation between nodes in the spanning tree, which only requires $O(N)$ space.

In Fig. 4, we show an algorithm next_hop(A, B) which calculates the neighbor of the node "A" to which a packet arriving at "A" and going to the node "B" should be forwarded. The time complexity of this algorithm is $O(logN)$. Using this algorithm, we can decide whether the packet's destination node B is one of the current node A's descendant nodes or not. If so, the incoming packet is forwarded to the child node whose descendant node is B. If not, the incoming packet is forwarded to its parent node. For example, since node 7 is a

```
next_hop( A, B )
begin
  while ( B > 0 )
   B_parent = (B-1)/k

   if ( B_parent == A)
    return B
   end

   B = B_parent
  end
  return (A-1)/k
end
```

Figure 4: Algorithm for Deciding Neighbor Node in Algorithmic Routing Approach

child node of node 2 and node 22 is a child node of node 7 (i.e. node 22 is a descendant node of node 2), the packet arriving at node 2 whose destination node is 22 is forwarded to node 7. On the other hand, since node 5 is not an descendant node of node 2, the packet whose destination node is 5 is forwarded to the parent node 0 of node 2.

3 Basic Idea of Our Method

In order to apply the algorithmic routing approach explained above, there is a restriction that all the routes must be included in a unique spanning tree. In this paper, we remove this restriction and propose a new method where a spanning-tree based routing table and a general routing table are combined, keeping the size of the routing table as small as possible and lookup speed as fast as possible.

In the proposed method, we assume that a network to be simulated and a general routing table of the network as shown in Fig. 1 are given. Then, we construct a spanning tree. If an entry in the general routing table can be represented by the spanning tree using the algorithmic method (we say that the entry is *covered* by the spanning tree hereafter), then the entry is represented by the spanning-tree based routing-table. If not, we represent the entry in the general routing table as it is. This indicates that the routing information is represented by combining the spanning-tree based routing table and the general routing table. The algorithmic routing method only needs $O(N)$ space for representing the routing information, and it does not depend on the number of entries which are covered by the spanning-tree based routing table. Therefore, if we can maximize the number of entries which can be covered by the spanning-tree based routing table, we can minimize the size of the general routing table necessary for representing the rest of the entries, and thus the space required for entire routing information is reduced.

In Section 4, we propose an algorithm which constructs a near-optimal spanning tree which covers as large number of entries as possible.

4 Finding a Near-Optimal Spanning Tree

In this section, we present a heuristic algorithm to find a near-optimal spanning tree st. In our algorithm, for each edge in a given network, we construct a spanning tree which contains the edge. The construction is done in a greedy manner. Then, we select the best one from these spanning trees. We explain the spanning-tree construction algorithm below.

We denote each entry in a given routing table as $< n_i, n_j >$ where n_i is a current node and n_j is a destination node. For an entry $< n_i, n_j >$, let $next(n_i, n_j)$ denote a neighbor node in the given routing table. For a given spanning tree st and an entry $< n_i, n_j >$, let $next_{sp}(st, n_i, n_j)$ denote a neighbor node determined by the algorithmic routing on st. If $next(n_i, n_j)$ and $next_{sp}(st, n_i, n_j)$ are identical, the entry $< n_i, n_j >$ in the given general routing table can be represented by the algorithmic routing on st, and we say that this entry is *coverable* by st. If $next(n_i, n_j)$ and $next_{sp}(st, n_i, n_j)$ are not identical, the entry $< n_i, n_j >$ cannot be represented, and we say that it is uncoverable by st. We assume that the number of nodes in the given network is N. In our spanning-tree construction algorithm, starting from an edge, edges are added one by one until a spanning tree is constructed. In adding an edge, we select one so that the number of entries which are newly made uncoverable by adding the edge is smallest. As a result, we can construct a spanning tree st where the number of entries covered by st is considerably large, in reasonable computing complexity.

Let t denote a tree on the given network. For a tree t, t' denotes a tree obtained by adding an edge e to t. Also, $ST(t')$ denotes the set of all the spanning trees which contain t' as a subtree. For an edge e and a tree t, $URE(t, e)$ denotes the set of entries which are made uncoverable by adding e to t. Let k denote the maximum degree of the given network. When we add an edge to t, the number of candidate edges is at most $O(k \times ($ the number of nodes in $t))$. In these candidate edges, we select the edge e where $|URE(t, e)|$ is smallest. Whether an entry is in $URE(t, e)$ or not can be determined by the following two conditions. We denote the edge e as (n_u, n_v) where n_u is the node contained in t.

1. If there exists an edge $e' = (n_v, n_w)$ where n_w is contained in t ($n_w \neq n_u$), the entries $< n_v, * >$ whose neighbor nodes are n_w and the entries $< n_w, * >$ whose neighbor nodes are n_v, are uncoverable.

   Adding e' to t' causes a closed path, so e' cannot be contained in any spanning tree in $ST(t')$. Thus entries, whose current nodes and neighbor nodes are the end nodes of e', cannot be covered by any spanning tree in $ST(t')$. Thus these entries are in $URE(t, e)$.

2. Let n_z denote a node contained in t. Entries $< n_z, n_v >$ (or $< n_v, n_z >$) whose neighbor nodes are not identical to $next_{sp}(t', n_z, n_v)$ (or $next_{sp}(t', n_v, n_z)$), are uncoverable.

   Since we adopt a greedy way to select edges, t' is contained in the obtained spanning tree. This means that all the routes on t' must follow the algorithmic routing since those routes will contain in the obtained spanning tree. Therefore, for all entries whose current nodes and destination nodes are contained in t', their neighbor nodes must be identical to those computed by the algorithmic routing. The entries which do not satisfy this condition should be in $URE(t, e)$.

We will explain how to calculate $URE(t, e)$ using Fig. 5. Fig. 5 (a) shows an intermediate tree t (represented by thick lines). Now candidate edges to add t are 2-5, 4-5, 4-7 and 6-7. Here we explain only the cases for 4-7 and 6-7. Fig. 6 shows entries in a given routing table which are related to these two edges. By adding 6-7 to t (Fig. 5 (b)), $< 4, 7 >$ is made

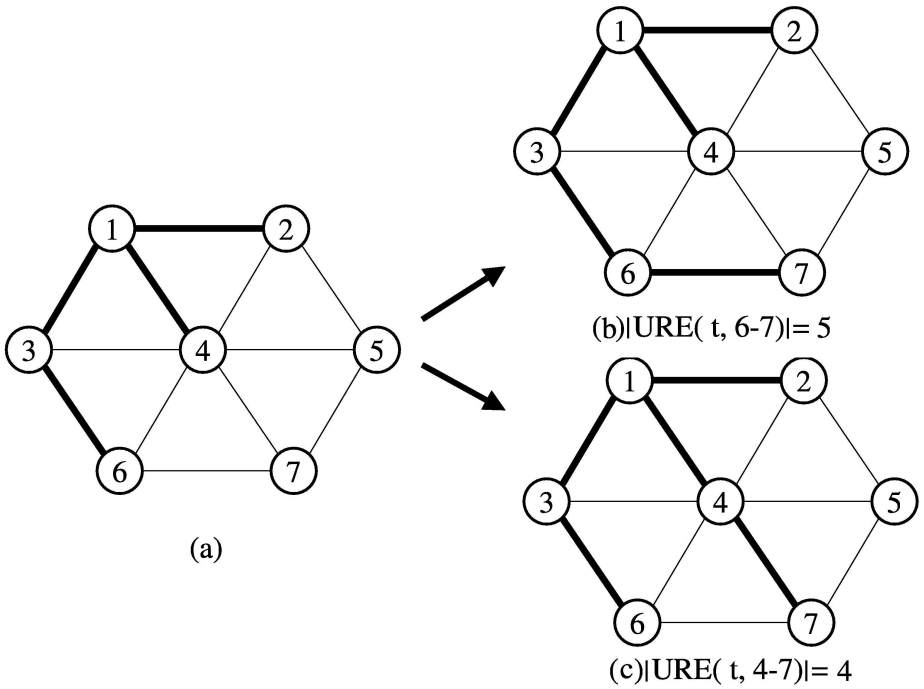


Figure 5: Caclulation of $URE(t, e)$

uncoverable by condition (1) because adding 4-7 results in a closed path. Also, $< 1, 7 >$ is made uncoverable by condition (2) because a packet arriving at node 1 is indicated to be forwarded to node 4 in the given routing table, but to be forwarded to node 3 by the algorithmic routing using this tree. Similarly, $< 2, 7 >$, $< 7, 1 >$, $< 7, 2 >$ and $< 7, 4 >$ are made uncoverable by condition (2). However, $< 2, 7 >$ is not in $URE(t$, 6-7) because this entry was already made uncoverable by condition (1) when edges 1-2 and 1-4 were added to the tree. Therefore, $|URE(t$,6-7)| is 5. On the other hand, by adding 4-7 to t (Fig. 5 (c)), $< 6, 7 >$ by condition (1) and $< 3, 7 >$, $< 7, 3 >$ and $< 7, 6 >$ by condition (2) are made uncoverable. So, $|URE(t$,4-7)| is 4. Therefore, 4-7 is added to t.

After adding $N - 2$ edges, we can get a spanning tree st. Our spanning-tree construction algorithm is shown in Fig 7. In Fig. 7, we start with an edge ie. Then for each edge e in $TE(t)$, the set of edges which do not cause closed paths to t, we compute $URE(t, e)$. Then we select the one whose $|URE(t, e)|$ is minimum, and add it to t. This is iterated until a spanning tree is obtained.

In our spanning-tree construction algorithm, adding an edge to a tree, the number of candidate edges is at most $O(k \times N)$. So the number of calculations of $URE(t, e)$'s is also $O(k \times N)$. The time complexity to compute each $URE(t, e)$ is $O(N)$. Thus, the time complexity of adding an edge is $O(k \times N^2)$. On constructing the spanning tree, we add $N - 2$ edges. Thus, the time complexity of our spanning tree construction algorithm is $O(k \times N^3)$. In our method, we adopt our spanning-tree construction algorithm for all edges in the given

current node	destination node	neighbor node
⋮	⋮	⋮
1	7	4
⋮	⋮	⋮
2	7	4
⋮	⋮	⋮
3	7	6
⋮	⋮	⋮
4	7	7
⋮	⋮	⋮
6	7	7
⋮	⋮	⋮
7	1	4
7	2	4
7	3	6
7	4	4
⋮	⋮	⋮
7	6	6
⋮	⋮	⋮

Figure 6: An Example of a Given Routing Table(this corresponds to the example in Fig. 5)

$t = \{ie \in E\}$

for $i = 1$ to $N - 2$
  foreach e in $TE(t)$
    compute $URE(t, e)$
  end
  find e where $|URE(t, e)|$ is minimum
  $t = t \cup e$
end

Figure 7: Spanning-Tree Construction Algorithm

network. Since the number of edges in the network is at most $N \times (N - 1)$, and the time complexity of our method is $O(k \times N^5)$ where N is the number of nodes and k is the maximum degree of the network.

5 Experimental Results and Discussion

In the proposed method, as more entries are covered by a spanning tree, the total size of routing tables will be smaller. In this section, we have carried out some experiments to confirm what percentage of entries are covered by the spanning tree which the proposed algorithm has computed.

The number of entries which can be covered by a spanning tree depends largely on the characteristics of the routes. In order to carry out our experiments in route sets with different characteristics, we assume that all routes are shortest paths and apply our algorithm to two different types of topologies: (i) hierarchical topology and (ii) random topology. Here, in each hierarchical topology generated, the routes converged to some specific links, and formed nearly a tree. On the other hand, in each random topology generated, the routes were uniformly distributed to all links. For these topologies, we have measured the percentage of entries covered by the spanning trees computed by our algorithm.

Since the proposed algorithm is a heuristic one, it cannot always compute the optimal spanning tree which covers the maximum number of entries. So, in the experiments, we have compared the spanning tree computed by our algorithm with the one in the optimal case, with respect to the number of entries covered by the tree.

5.1 Experiments and Evaluation for Hierarchical Topologies

In the first experiment, we have used Tiers [6], one of hierarchical topologies consisting of WAN, MAN and LAN. We expect that large portion of entries can be reduced from the general routing table using a spanning tree since the Tiers topology is nearly a tree. Here, each LAN is a network with the star topology which can be considered as a tree. So, even when the number of nodes in each LAN increases, most of them can be covered by a spanning tree. On the other hand, each node in MAN is considered to have multiple adjacent nodes. So, for entries with such a node, we think that a spanning tree can cover only a part of them.

In the experiment, we have measured the percentage of the entries covered by a spanning

tree to all entries for the following two cases: (a) fix the numbers of nodes in WAN and MAN, respectively, and vary the number of nodes in LAN from 20 to 600 by 20 nodes; (b) fix the numbers of nodes in WAN and LAN, and vary the number of nodes in MAN from 20 to 400 by 10 nodes.

The results are depicted in Fig. 8 (a) and (b), respectively. Here, the vertical axis means the percentage of the entries covered by the spanning tree, and the horizontal axis does the number of nodes in LAN or MAN. In Fig. 8 (a), we see that the percentages were over 90 % for some cases. For both cases (a) and (b), the percentages did not vary so much and they were between 85 % and 90 % while the number of nodes increases. These results show that the proposed method is fairly useful for the reduction of routing tables for networks with nearly tree topologies like Tiers, independently of the number of leaf and intermediate nodes.

Among entries in the given routing table, only the entries covered geometrically by a spanning tree can be covered in the spanning tree based routing. Therefore, in the proposed method, for the near tree topologies like hierarchical topologies, independently of routing algorithms, most of entries can be covered by the spanning tree based routing. Thus those entries can be removed from the general routing table. In the Internet, networks form hierarchical topologies due to their nature. So, the proposed method is useful for network simulations assuming the Internet.

5.2 Experiments and Evaluation for Random Topologies

It is important to know what percentages of entries are covered by a spanning tree depending on network topologies. For the purpose, we applied the similar experiment to random topologies. Here, we have measured the percentages of entries covered by the spanning trees when the number of nodes changes from 20 to 240 by 20 nodes.

The result is shown in Fig. 9. Here, the vertical and horizontal axes mean the percentages and the numbers of nodes, respectively. Unlike hierarchical topologies, in random topologies, the percentages were 60 % and 30 % for 20 nodes and 200 nodes, respectively. The result shows that it is more difficult to cover many entries by a spanning tree in topologies with many edges such as random topologies than hierarchical topologies.

5.3 Comparison with the Optimal Spanning Tree

In order to show the usefulness of our method, we have compared the spanning tree computed by our algorithm with the optimal tree with respect to the number of entries covered by the tree. In the experiment, we computed the optimal spanning tree by generating all possible spanning trees and by evaluating the number of entries covered by each tree. It takes much time to compute the optimal spanning tree. So, we used 20 nodes and 15 nodes for the hierarchical topology and for the random topology, respectively. We simulated 50 times for each topology, and measured the percentages of the entries covered by the spanning tree computed by our algorithm to the entries covered by the optimal tree.

For both topologies, in half of all simulation cases, the percentages were 100 %. Even in other simulation cases, the percentages were over 95 %. This shows that our algorithm can generate the spanning tree close to the optimal spanning tree.

6 Conclusion

In this paper, we have proposed a new method to reduce the size of general routing tables used in network simulators. Our method converts a part of a given general routing table that can represent any routing strategy into a spanning tree based routing table proposed in [5] to reduce the entire size of the routing table. Our main contribution is that our method allows simulators to handle simulation scenarios under any routing strategy as small size of routing tables as possible. Experimental results have shown that spanning trees found by our heuristic algorithm could cover about 90 % of the entries in given routing tables on hierarchical networks.

We are now planning to incorporate our method into network simulator ns-2 for large-scale network simulation. We are also considering a Tabu search based approximate algorithm to find spanning trees to improve the performance of our algorithm. Moreover, instead of using a single spanning tree to reduce the size of a routing table, using multiple spanning trees is considered effective. This is a part of our future work.

References

[1] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a library for parallel simulation of large-scale wireless networks," in *Proceedings of the 12th Workshop on Parallel and Distributed Simulations(PADS'98)*, 1998.

[2] MASH Research Group, University of California, Berkeley, "The network simulator ns-2," 2000. http://www-mash.cs.berkeley.edu/ns/.

[3] G. F. Riley, R. Fujimoto, and M. H. Ammar, "Stateless routing in network simulations," in *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, May 2000.

[4] G. F. Riley, M. H. Ammar, and E. W. Zegura, "Efficient routing with Nix-Vectors," in *Proceedings of IEEE Workshop on High Performance Switching and RoutingHPSR 2001*, 2001.

[5] P. Huang and J. Heidemann, "Minimizing routing state for light-weight network simulation," in *Proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, August 2001.

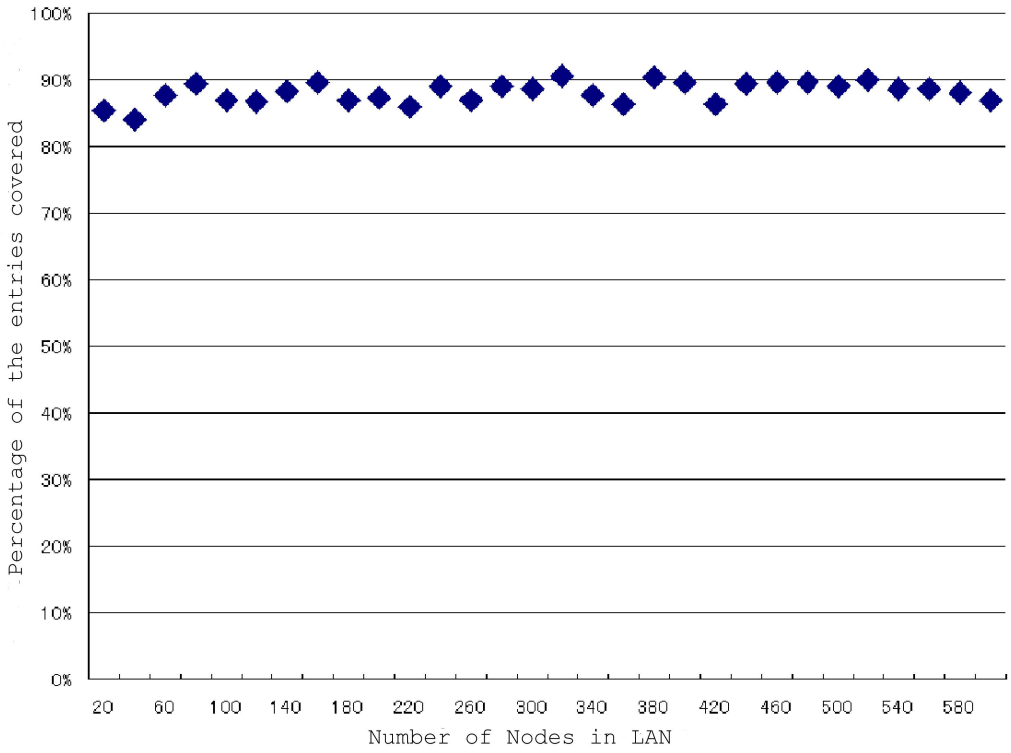
[6] K. Calvert, M. Doar, and E. Zegura, "Modeling internet topology," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160–163, 1997.

[7] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. McCanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala, "Improving simulation for network research," Tech. Rep. 99-702b, University of Southern California, March 1999.

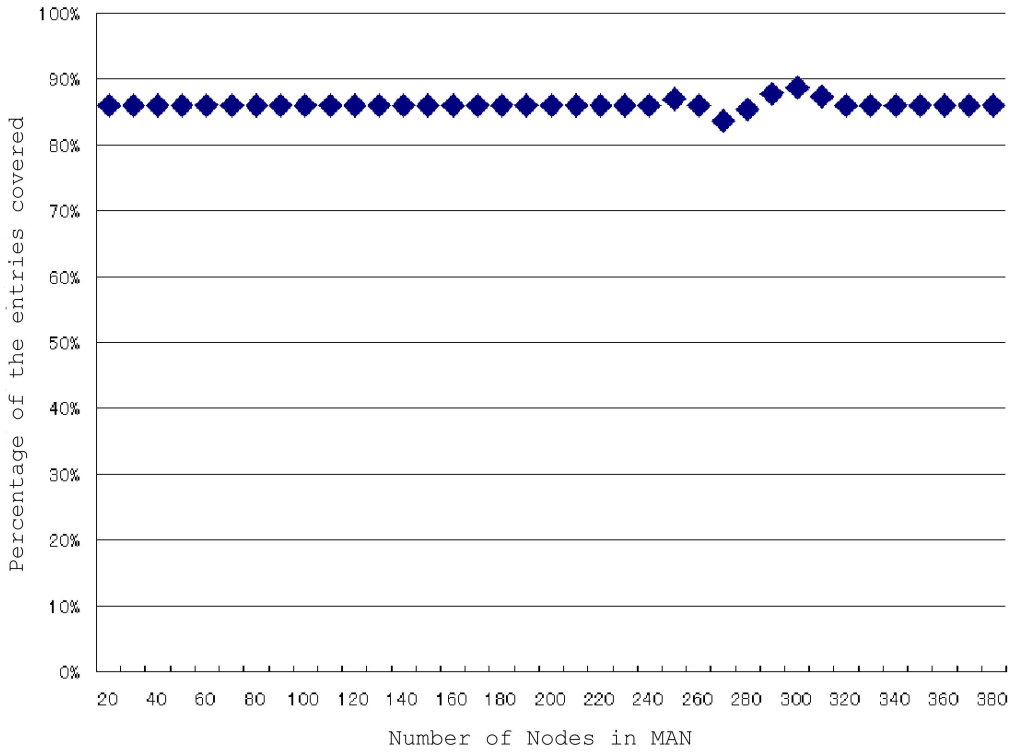
[8] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in network simulation," *IEEE Computer*, vol. 33, pp. 59–67, May 2000.

[9] S. Floyd and V. Paxson, "Difficulties in simulating the Internet," *IEEE/ACM Transactions on Networking*, vol. 9, pp. 392–403, August 2001.

[10] J. Heidemann, K. Mills, and S. Kumar, "Expanding confidence in network simulation," *IEEE Network Magazine*, vol. 15, pp. 58–63, Sept./Oct. 2001.



(a) When the Number of Nodes in LAN Increases



(b) When the Number of Nodes in MAN Increases

Figure 8: Percentage of the Entries Covered by Spanning Trees in Hierarchical Topologies

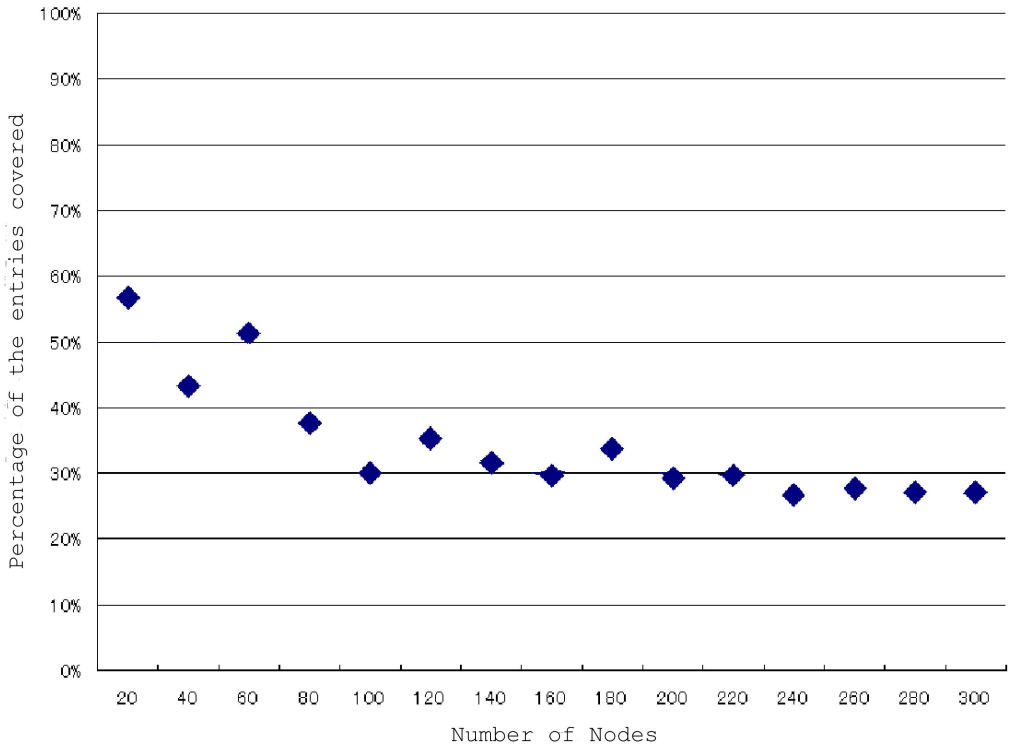


Figure 9: Percentages of the Entries Covered by Spanning Trees in Random Topologies