

Randomized Distance-Vector Routing Protocol

Sangman Bak
Department of Computer Science
University of Houston
Houston, TX 77204-3475
1-713-797-3350
smbak@cs.uh.edu

Jorge A. Cobb
Department of Computer Science
University of Texas at Dallas
Dallas, TX 75083-0688, USA
1-972-883-2479
jcobb@utdallas.edu

1. ABSTRACT

The purpose of routing protocols in a computer network is to maximize throughput. Shortest-path routing protocols have the disadvantage of causing bottlenecks due to their single-path routing. That is, the shortest path between a source and a destination may become highly congested even when many other paths have low utilization. In this paper, we propose a routing scheme that randomly balances traffic over the whole network, therefore, it removes bottlenecks and increases network throughput. For each data message to be sent from a source s to a destination d , the proposed routing protocol randomly chooses an intermediate node e , and routes the data message along the shortest path from s to e . Then, it routes the data message via the shortest path from e to d . This increases the effective bandwidth between each pair of nodes. In our simulation results, we show that this proposed routing protocol distributes traffic evenly over the whole network and increases network throughput.

1.1 Keywords

Shortest path routing, randomized distance-vector routing, network throughput, load balancing, effective bandwidth

2. INTRODUCTION

In a wide-area store-and-forward computer network, such as Internet, routing protocols are essential. They are responsible for finding an efficient path between any pair of source and destination nodes in the network, and routing data messages along this path. The path must be chosen so that network throughput is maximized and message delay, message loss and other undesirable events are minimized.

There are mainly two types of routing protocols: source routing and destination routing (next-hop routing). In source routing, a source node determines the path that a data message must take [4], [6]. In destination routing, each node uses its routing table to store a next hop for each destination. Thus, the routing table specifies only one hop along the path from the current node to a destination. In this paper, we deal with destination-based routing, due to its prevalence in the Internet.

Destination routing protocols are classified into two types of routing protocols: distance-vector routing, used in the RIP Internet protocol [10], and link-state routing, used in the OSPF Internet protocol [13].

In a distance-vector routing protocol, each node maintains a routing table and a distance vector, which contains, respectively, a preferred neighbor for the shortest path to each destination in the network and the distance of the path to the destination. Each node has incomplete knowledge of the network topology, and knows only its neighboring nodes. From these neighbors, the node chooses a closest neighbor to each destination. At a stable state of the protocol, the path made by consecutive preferred neighbors to a given destination is the shortest path to the destination. Each node periodically sends its distance vector to each of its neighbors to inform it of any distance changes to any destinations. The node determines which neighbor is the closest to each destination by comparing the distance vectors of its neighbors [3], [7].

Link-state routing protocols require each node to maintain complete network topology information. Each node actively tests the status of the links between itself and

its neighbors. Then, it periodically broadcasts the local link status information to all other nodes. Since each node receives the local link status information from all other nodes, it is able to build a graph of the whole network topology and to compute the shortest path from itself to every other node [3], [8].

Unfortunately, destination routing protocols suffer performance degradation because all data messages are routed via the same shortest path to the destination until the routing tables have been updated. The problem with these routing protocols is that there are no mechanisms to alter the routing other than updating the routing tables. Routing table updates occur too slowly to respond to significant traffic fluctuations. Furthermore, increasing the frequency of routing table updates leads to unstable network behavior and an increase in network load due to routing state messages [14].

Note that the shortest path may be highly congested, even when many other paths to the destination are not congested. This congestion may trigger the loss of valuable messages [17]. Moreover, using the same path to the destination limits the maximum possible throughput to the destination to be at most the minimum capacity of the links along the shortest path to the destination.

Maximizing network throughput is an important goal in the design of routing algorithms and networks. An analysis in network flow theory, known as the max-flow min-cut theorem [5], shows that the distribution of the traffic load over the available paths between a source and a destination in the network, instead of using only one path of the minimum cost, increases the maximum possible throughput up to the capacity of the minimum cut separating these two nodes.

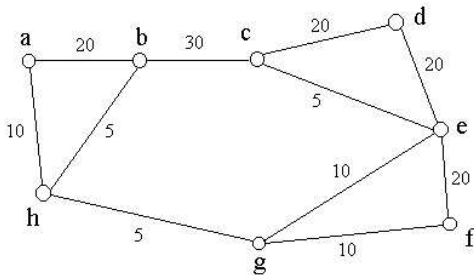


Figure 1: A Network Topology

For example, let's consider Figure 1. The number beside each link represents its capacity. Suppose that node **a** (source) wants to send the data messages to node **f** (destination). Suppose that we use the hop count in order to calculate the cost of a path in the network. Then the effective bandwidth (i.e. the capacity of the minimum cut) between node **a** and node **f** is 30, while the effective bandwidth of the shortest path (**a-h-g-f**) from node **a** to node **f** is 5.

So far, several techniques to increase the effective bandwidth between each pair of nodes and to improve performance have been proposed [2], [9], [14], [16]-[18]. These routing protocols improve performance by routing data messages via multiple paths to the destination. They maintain a stable routing table and meanwhile provide alternate paths to distribute data traffic when it is accumulating in some nodes of the network. When congestion and network failures do occur, instead of initiating route updating, the source temporarily forwards the traffic along the alternate paths and passes around the congested areas or failed areas. If the changes are persistent, the routing tables will be updated eventually when the next route updating time is due. Some techniques are Shortest Path First with Emergency Exits [17] based on link-state routing, multiple disjoint paths [16] based on distance-vector routing, and Dynamic Multi-path Routing [2] based on source routing. But these techniques require considerable processing overhead and non-negligible storage space, or increase the complexity of the routing algorithms.

In this paper we propose a routing scheme based on randomization that distributes traffic load over the whole network, and therefore, it improves the network throughput by distributing the traffic load over all available paths to the destination.

The rest of this paper is organized as follows. Section 3 presents the overview of the proposed routing protocol. Section 4 introduces the protocol notation to give a formal version of our routing protocol, which is given in Section 5. In Section 6 and 7 we present the simulation model and our results. In Section 8 and 9 we present future work and the conclusions, respectively.

3. OVERVIEW OF RANDOMIZED DISTANCE-VECTOR ROUTING PROTOCOL (RDVRP)

In this section, we sketch how RDVRP routes data messages to the destination. Each node creates data messages and receives data messages from its neighbors. The node should forward these data messages to its neighbors so that the number of links traversed by each data message is as small as possible, while at the same time attempting to distribute these data messages evenly throughout the network to avoid congestion and increase network throughput. Our scheme is based on the distance-vector routing algorithm.

3.2 Simple Randomized Distance-Vector Routing Protocol

In this subsection, we present a simple version of RDVRP. For each data message to be sent from a source node *s* to a destination node *d*, the proposed routing

scheme randomly chooses an intermediate node *e* among all the network nodes. It then routes the message via the shortest path from *s* to *e*. Then, it routes the message via the shortest path from *e* to *d*.

As an example, consider Figure 1 again. Suppose that node **a** (source) wants to send the data messages to node **f** (destination). For load balancing, node **a** should distribute the data messages evenly over the possible paths to node **f**. Node **a** may accomplish this by selecting randomly an intermediate node (say node **c**) among all the nodes in the network whenever node **a** sends a data message to node **f**, routing it to the intermediate node **c** via the shortest path between node **a** and node **c** and then routing it to destination **f** via the shortest path between node **c** and node **f**.

To accomplish this, each message must carry at least three pieces of information: the destination *d*, the intermediate node *e*, and a bit *b*. Bit *b* indicates whether the message has not yet reached *e* ($b = 0$), or it has already passed through node *e* ($b = 1$).

Therefore, the operation of the protocol is as follows. Initially, the source node *s* sends the message with $b = 0$, and routes it in the direction of node *e*. As long as $b = 0$, the message keeps being routed along the network until it reaches node *e*. At node *e*, *b* is updated to 1, and the message is routed towards node *d*. As long as $b = 1$, the message keeps being routed along the network until it reaches node *d*, and is delivered.

3.2 Bounding the Initial Routing Path

The protocol of the previous subsection, however, has a shortcoming. It is possible that a data message is routed to the destination via a very long path, much larger than the shortest path between the source and the destination.

For example, in Figure 1, suppose that node **a** wants to send a data message to node **b**, and it randomly chooses node **f** as the intermediate node. Then, the proposed algorithm routes the data message to node **f** via the shortest path (**a-h-g-f**) and then routes it to node **b** via the shortest path (**f-e-c-b**). Although there is a path of length 1 between node **a** and node **b**, the proposed algorithm may use the path of length 6.

Therefore, routing paths that are excessively long will waste network resources and reduce network throughput.

To remedy this problem, we introduce a parameter *k*, in order to exclude the nodes which are too far away from the source from being candidates for an intermediate node. The set of candidates is restricted to all those nodes whose distance to the source is at most *k*.

The value chosen for *k* affects delay, the path length, load balancing, and network throughput. If *k* is zero, the

length of traveled path is minimized because our routing protocol becomes the conventional distance-vector routing protocol, and thus the data message will be routed via a shortest path to the destination. On the other hand, if *k* is non-zero, a larger number of routing paths are available, which alleviates congestion and increases the effective bandwidth between these two nodes, but at the expense of increasing the length of the traveled path.

Choosing the appropriate value for *k* is crucial for the performance of the algorithm. Choosing a small value will increase the likelihood of bottleneck. On the other hand, choosing a large value will waste network resources by routing packets via excessively long paths. To reach a compromise between these two extremes, we choose *k* to be the average of the distance to each node reachable from the source.

4. PROTOCOL NOTATION

In this paper, we use a simple notation to define our routing protocol. A protocol is defined by a set of processes, $p[0], p[1], \dots, p[n-1]$. A process corresponds to a node in a computer network. A pair of neighboring processes is joined with a communications channel. Henceforth, we use the term process and node interchangeably.

A process is defined by a set of constants, a set of inputs, a set of variables and a set of actions. The actions of a process are separated by the symbol \parallel as follows:

begin *action.1* \parallel *action.2* $\parallel \dots \parallel$ *action.m* **end**

An action has the following form: guard \rightarrow statement. A guard is a Boolean expression, which refers to constants, inputs, and variables of the process. A statement is defined recursively as one of the following: skip, assignment statement, conditional (**if ... fi**), bounded loop (**for ... rof**) and a sequence of two statements separated by ";".

An action in a process is enabled if and only if the action's guard is true at the current state of the network. An execution step of a protocol consists of choosing any enabled action from any process, and executing the action's statement. Executions are maximal, i.e., either they consist of an infinite number of execution steps, or they terminate in a state in which no action is enabled. Executions are assumed to be fair, i.e., each action that remains continuously enabled is eventually executed.

The communication between processes is based on a message-passing model. For every pair of neighboring processes $p[i]$ and $p[j]$, there are a FIFO channel from $p[i]$ to $p[j]$ and a FIFO channel from $p[j]$ to $p[i]$. The statement **send** *data*(*var*) **to** $p[j]$ in process $p[i]$ appends a message of type *data* to the channel from $p[i]$ to $p[j]$, and the field in

the message is the current value of variable var in process $p[i]$.

In addition to Boolean expressions, guards in each process $p[i]$ are allowed to be of the form **rcv** data(var) **from any** $p[j]$. This guard is enabled iff there is a message of type $data$ at the head of an incoming channel of $p[i]$. If an action with this receive guard is chosen for execution, then, before its command is executed, the data message is removed from the channel, and its field is copied into the local variable var . Furthermore, variable j is set to the identity of the neighbor from whom the message is received.

Similar protocol notations are defined in [11] and [12].

5. SPECIFICATION OF RANDOMIZED DISTANCE-VECTOR ROUTING PROTOCOL

In this section, we present a formal version of the load-balanced routing protocol. Each process has a constant n with the number of the processes in the network and an input set N with the identities of its neighbors.

Each process $p[i]$ has several variables. Variable $inter$ stores candidates for intermediate nodes. That is, $inter$ stores the process id's of processes which are at most k hops away from process $p[i]$. Variable $rtb[j]$ stores the preferred neighbor to reach destination $p[j]$, and $hop[j]$ stores the distance to reach destination $p[j]$.

The load-balanced routing protocol is defined as follows.

```

process p [i: 0 .. n-1]
constants
n      : integer {number of nodes in the network}
inputs
N      : set of {j | p[j] is a neighbor of p[i]}
variables
k      : 0 .. n-1, {maximum length of 1st routing path}
inter  : set of {0 .. n-1} {possible intermediate
nodes of routing paths}
rtb    : array [0 .. n-1] of 0 .. n, {routing table}
hop    : array [0 .. n-1] of 0 .. n,
{hop count to each destination}
h      : array [0 .. n-1] of 0 .. n,
{neighbor's hop count to each destination}
e, d   : 0 .. n-1,
{message's intermediate node and destination}
b      : 0 .. 1,
{status bit: b=0 on 1st routing path, b=1 on 2nd
routing path}
x      : 0 .. n-1,
j      : element of N
begin
true →
k := average{hop[x] | 0 ≤ x < n ∧ 0 ≤ hop[x] < n};

```

```

inter := {x | hop[x] ≤ k}
[] true →
{create and route a new message to any destination}
b := 0;
d := any;
e := random(inter);
RTMSG
[] rcv data(b, e, d) from any p[j] → RTMSG
[] true → {send hop count to neighbors}
for each j in N do
send upd(hop) to p[j]
rof
[] rcv upd(h) from any p[j] →
UPDTBL

```

end

In the first action, process $p[i]$ computes k as the average hop count to each reachable destination in the network. Then, the set of candidate nodes for intermediate routing nodes is computed. In the second action, the process creates a data message and chooses a destination for the message. Also, an intermediate node is chosen at random from the intermediate candidates set. Then, the created data message is routed to a neighboring process using statement RTMSG, defined below.

In the third action, the process receives a data message and then routes the message to a neighbor using statement RTMSG. In the fourth action, the process sends a copy of its distance vector to each of its neighbors. In the last action, the process receives a copy of the distance vector from one of its neighbors, and updates its routing table and local distance vector using statement UPDTBL, which is defined below.

Statement RTMSG is defined as follows.

```

if d = i → {arrived, deliver message}
skip
[] d ≠ i ∧ b = 0 ∧ hop[e] = n →
{unreachable intermediate node}
skip
[] d ≠ i ∧ b = 0 ∧ hop[e] < n ∧ e ≠ i →
{reachable intermediate node}
send data(b, e, d) to rtb[e]
[] d ≠ i ∧ b = 0 ∧ hop[e] < n ∧ e = i →
{end of first routing path}
send data(1, e, d) to rtb[d]
[] d ≠ i ∧ b = 1 ∧ hop[d] = n → {unreachable destination}
skip
[] d ≠ i ∧ b = 1 ∧ hop[d] < n → {reachable destination}
send data(1, e, d) to rtb[d]

```

fi

In this statement, the process checks fields b , e , and d , of the message. If $d = i$, the message has reached its destination and is delivered. Otherwise, if $b = 0$, then the message is in its first routing path, and it is routed towards

process $p[e]$. If $b = 1$, the message is in its second routing path, and it is routed towards process $p[d]$.

Statement UPDTBL is defined as follows.

```

hop[i] := 0;
for each x, where  $x \neq i$ , do
  if  $rtb[x] = j \wedge (h[x]+1) \neq hop[x] \rightarrow \{p[i] \text{ currently routes}$ 
to  $p[x]$  via  $p[j]$ , and  $p[j]$ 's distance to  $p[x]$  has changed.}
     $hop[x] := \min(h[x]+1, n)$ 
     $[\text{rtb}[x] \neq j \wedge (h[x]+1) < hop[x] \rightarrow \{\text{found a shorter path}\}$ 
       $hop[x] := \min(h[x]+1, n);$ 
       $rtb[x] := j$ 
     $[\text{rtb}[x] \neq j \wedge (h[x]+1) \geq hop[x] \rightarrow \{\text{keep the current path}\}$ 
      skip
     $[\text{rtb}[x] \notin N \rightarrow \{\text{p[rtb[x]] is down}\}$ 
       $hop[x] := \min(h[x]+1, n);$ 
       $rtb[x] := j$ 
  fi;
rof

```

In the first case of this statement, if process $p[i]$ currently routes to a destination $p[x]$ through the neighbor $p[j]$, and $p[j]$'s distance to the destination changes, process $p[i]$ updates its distance to $p[x]$ accordingly. In the second case, if the neighbor $p[j]$ uses a shorter path to reach a destination $p[x]$, then process $p[i]$ chooses $p[j]$ as the next hop to destination $p[x]$. In the third case, process $p[j]$ does not have a shorter path to $p[x]$ than $p[i]$'s path, and thus, $p[i]$ does not update its tables. In the last case, the next hop neighbor to $p[x]$ is no longer connected to $p[i]$ due to a failure, and thus $p[i]$ chooses $p[j]$ as its next hop to $p[x]$.

6. SIMULATION MODEL

Our simulation studies were done on the Maryland Routing Simulator (MaRS) [1], which is a network simulator developed at the University of Maryland. A

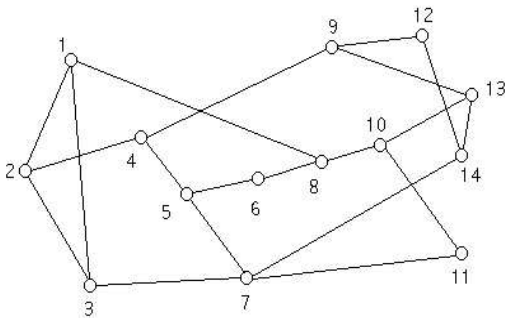


Figure 2: NSFNET Topology: 14 nodes, 21 bi-directional links, average degree 3

network configuration consists of a physical network (nodes and links), a routing algorithm and a workload.

6.1 Physical Network

In our simulation the physical network is the NSFNET topology in Figure 2. All links have the bandwidth, 1.5 Mbits/sec. There are no link or node failures. Each node has a buffer space of 50,000 bytes. The processing time of a data message at each node equals 1 μ sec. In order to calculate the cost of a path in the network, we use the hop count. The cost of a link between any two neighbors is 1. The propagation delay of each link is 1 msec.

6.2 Routing Algorithm

The Routing algorithms are SEGAL and RDVRP. SEGAL is a distance-vector and loop-free routing protocol. SEGAL uses a shortest-distance path for each pair of source and destination nodes. To get a better understanding of RDVRP, we compare the performance of RDVRP against SEGAL.

6.3 Workload

The workload is file transfer protocol (FTP). All FTP connections have the following parameters: the data message length equals 512 bytes, the inter-message generation time is 1 or 10 msec, and the window size is 500 messages. Connections start when the simulation begins and they are never-ending.

6.4 Performance Measures

We consider measures of throughput, message loss and link utilization. The measurement interval of each simulation is 50,000 msec.

- *Throughput*: Total number of data bytes acknowledged during the measurement interval divided by the length of the measurement interval.
- *Message loss*: Total number of the messages dropped during the measurement interval.
- *Link Utilization*: The data service rate divided by the link bandwidth.

7 SIMULATION RESULTS

In this section, we present the simulation results and some observations about those results.

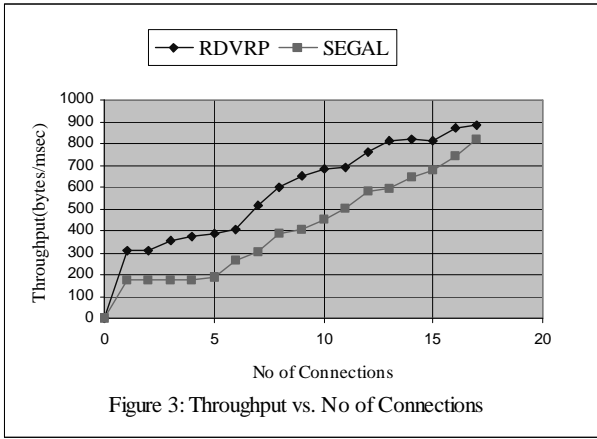


Figure 3 shows throughput versus the number of connections. The throughput first increases sharply, then levels off as network links reaches the saturation point. The saturation points in RDVRP are around 1, 10 and 14. The saturation points in SEGAL are around 1, 8 and 12. RDVRP is much better with respect to throughput.

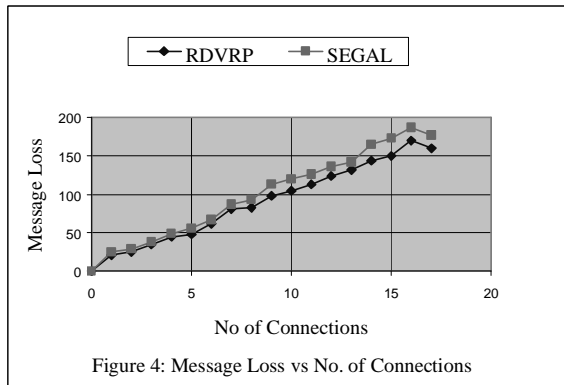
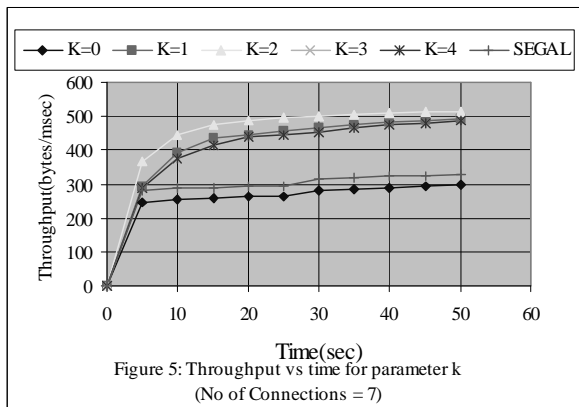


Figure 4 shows message loss versus the number of connections. The message loss in the two routing protocols almost linearly increases as the number of connections increases. RDVRP has less message loss at almost all times during the measurement interval.

Figure 5 shows throughput versus time for several



values of parameter k . RDVRP has a much better throughput than SEGAL does when k is more than 0.

Table 1 shows average link utilization for the measurement interval when the number of connections is 7. We can see that RDVRP more evenly distributes the data messages over the whole network than SEGAL does by investigating the link utilization over the whole network. And also, the total and the average of the link utilizations over the whole network indicate that RDVRP uses network resources more efficiently than SEGAL does.

Link	RDVRP	SEGAL
(1,2)	0	0
(2,1)	0.477867	0
(1,3)	0	0
(3,1)	0.116053	0
(2,3)	0.2048	0.02048
(3,2)	0.232107	0.116053
(2,4)	0.580267	0.955733
(4,2)	0.24576	0.0750933
(1,8)	0.709973	0
(8,1)	0	0
(4,5)	0.116053	0.423253
(5,4)	0.395947	0.273067
(3,7)	1	0
(7,3)	0	0
(5,6)	0.116053	0.361813
(6,5)	0.28672	0.0341333
(5,7)	0.402773	0.28672
(7,5)	0.0477867	0.361813
(4,9)	1	1
(9,4)	0.129707	0.0955733
(6,8)	0.116053	0
(8,6)	0.832853	0
(7,14)	1	0.36864
(14,7)	0.0477867	0
(7,11)	0.0341333	0.0341333
(11,7)	0	0.36864
(10,11)	0	0
(11,10)	0.587093	0
(8,10)	0.232107	0
(10,8)	0.587093	0
(9,12)	0.0136533	0.232107
(12,9)	0.0273067	0.04096
(10,12)	0.34816	0
(12,10)	0	0
(9,13)	0.587093	0.464213
(13,9)	1	0.0273067
(12,14)	0.02048	0
(14,12)	0.12288	0.24576
(10,13)	0	0
(13,10)	0	0
(13,14)	0.0273067	0
(14,13)	0.863787	0
Total	12.5096524	5.7854919
Average	0.29784887	0.13774981
Variance	0.11324603	0.05729708

Table 1: Link Utilization: (Measurement interval: 50,000 msec, Number of Connections: 7)

8 FUTURE WORK

We have many possible directions for future work.

The routing protocol proposed here is based on flat network architectures. But in the Internet the participating

nodes are partitioned according to a hierarchical structure. One direction for the future work is to apply our strategy to the hierarchical structure.

In this paper we used the average of the distances to the destinations in the routing table as the value of parameter k in order to exclude the nodes that are too far away from the source from being candidates for an intermediate node. Another possible future work is to determine the best value of parameter k for both balancing load and using a path of a distance as short as the conventional routing protocols do.

Our protocol uses an intermediate node even when the shortest path between source and destination are not congested. Using an intermediate node may increase the delay when the traffic load on every link along the shortest path is low. The third future work is to combine an adaptive routing and our strategy.

9 CONCLUSION

In this paper, we presented a randomized distance vector routing protocol to distribute the data traffic randomly over all available paths to a destination in the network for data load balancing. Our simulation results show that the proposed routing protocol has an improved performance with respect to throughput, message loss and link utilization under both light and heavy loads.

10 ACKNOWLEDGMENTS

It is our pleasure to thank the anonymous referees for their suggestion.

11 REFERENCES

- [1] Alaettinoglu, C., Dussa-Zieget, K., Matta, I., Gudmundsson, O., and Shankar, A. U. MaRS – Maryland Routing Simulator Version 1.0. Department of Computer Science, University of Maryland, 1991
- [2] Bahk, S. and Zarki, M. E., Dynamic Multi-path Routing and How it Compares with other Dynamic Routing Algorithms for High Speed Wide Area Networks, Proceedings of the 1992 ACM SIGCOMM Conference, Vol 22, Oct. 1992.
- [3] Christian Huitema, ROUTING IN THE INTERNET, Prentice Hall PTR, New Jersey, 1995.
- [4] David R. Cheriton, Sirpent: A high-performance internetworking approach, Proceedings of the 1989 ACM SIGCOMM Conference, Sep. 1989.
- [5] Dimitri P. Bertsekas, Linear Network Optimization: Algorithms and Codes, The MIT Press, 1991.
- [6] Dixon, R.C., Pitt, D.A., Addressing, bridging, and source routing (LAN interconnection), IEEE Network, Vol. 2, No. 1, Jan.1988.
- [7] J.J. Garcia-Luna-Aceves, A Minimum-Hop Routing Algorithm Based on Distributed Information, Computer Networks and ISDN Systems, Vol. 16, pp. 367-382, May 1989.
- [8] J.M. McQuillan, Ira Richer and E.C. Rosen, The New Routing Algorithm for the ARPANET, IEEE Trans. On Communications, Vol. COM-28, No. 5, pp. 711-719, May 1980.
- [9] Jorge A. Cobb and Gouda M. G., Balanced Routing, IEEE Proceedings of International Conference on Network Protocols, 1997.
- [10] G. Malkin, RIP Version 2, *Internet Request for Comments 1723*, Nov. 1994, Available from <http://www.ietf.cnri.reston.va.us>.
- [11] Mohamed G. Gouda, Protocol verification made simple: a tutorial, Computer Networks and ISDN Systems, Vol. 25, pp. 969-980, 1993.
- [12] Mohamed G. Gouda, The Elements of Network Protocol Design, A Wiley-Interscience Publication, John Wiley & Sons, Inc., 1998.
- [13] Moy J., Ospf Version 2, *Internet Request for Comments 1583*, March 1994. Available from <http://www.ietf.cnri.reston.va.us>.
- [14] S. Murthy and J.J. Garcia-Luna-Aceves, Congestion-Oriented Shortest Multipath Routing, Proc. IEEE INFOCOM 96, San Francisco, California, March 1996.
- [15] A. Segall and M. Sidi, A failsafe distributed protocol for minimum delay routing, IEEE Trans. On Commun., COM-29(5), 686-695, May 1981.
- [16] Sidhu D., Nair R., Abdallah S., Finding Disjoint Paths in Networks, Proceedings of the 1991 ACM SIGCOMM Conference.
- [17] Wang Z., Crowcroft J., Shortest Path First with Emergency Exists, Proceedings of the 1990 ACM SIGCOMM Conference.
- [18] W.T. Zaumen and J.J. Garcia-Luna-Aceves, Loop-Free Multipath Routing Using Generalized Diffusing Computations, Proc. IEEE INFOCOM 98, San Francisco, California, March 29, 1998.