Finding Disjoint Paths in Networks*

Deepinder Sidhu and Raj Nair and Shukri Abdallah

Department of Computer Science
University of Maryland — BC
Baltimore, MD 21228
and
Institute for Advanced Computer Studies
University of Maryland — CP
College Park, MD 20742

Abstract

Routing is an important function implemented in computer communication networks. There has been extensive research interest in distributed algorithms for single path routing. In many instances, it is desirable to have multiple disjoint paths between pairs of nodes in a network. Multiple disjoint paths can increase the effective bandwidth between pairs of nodes, reduce congestion in a network and reduce the probability of dropped packets. This paper presents a distributed distance-vector algorithm that finds multiple disjoint paths to a destination. The algorithm includes the shortest path as one of the disjoint paths. We describe the algorithm, evaluate its performance using simulation and compare it with another disjoint-path routing algorithm. We conclude that our algorithm requires 3 to 4 times fewer messages to discover paths of comparable quality.

*This research was supported in part by the Department of Defense at the University of Maryland Baltimore County. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of Defense or the U.S. Government.

1 Introduction

Considerable effort has been devoted to the design of distributed algorithms for finding multiple disjoint paths from every node to a destination node in a network. One solution [1] requires every node to have complete knowledge of the network topology. Another scheme [2] finds multiple paths that are initial-link-disjoint (disjoint in the first link). The method of link-disjoint augmentation [3, 4] was used in [5] to construct a pair of disjoint paths of minimum total cost from every node to a destination. A later paper [6] extended the work in [5] to find K disjoint paths of minimum total cost from every node to a destination. Alternate path distance-vector routing algorithms [7, 8] find alternate disjoint and non-disjoint paths.

In this paper, we present the design of a distributed algorithm which constructs a set, S_x, of minimum cost node-disjoint paths from every node x to a destination node. Unlike other methods, this algorithm does not require any graph transformation and retains the shortest path in the set S_x.

The rest of the paper is organized as follows. In Section 2, we present the design of the algorithm. In Section 3, we compare the simulation results of this algorithm and the algorithm described in [5]. In Section 4, we present the conclusions.

2 Constructing Multiple Paths

In this section, we propose an approach for constructing a set of minimum cost disjoint paths from every node to a destination node in a network.

2.1 Notations and Concepts

A network is modeled as an undirected graph, $G = (V, E)$, with a vertex set, V, and a link set, E. A link in E connects a pair of nodes, x and y, in V and is denoted by (x, y). The cost associated with this link is represented by $c(x, y)$. We assume that (1) the cost $c(x, y)$ is a positive number which is the same in both directions; (2) there is an arbitrary and distinguished node, z, in V called the destination node; (3) an operational link provides error-free FIFO communication and its propagation delay is arbitrary, but finite; (4) each node sends, if required, a message to its neighbor(s) within finite time after receiving a message; and (5) no topological changes occur during the construction of the alternate paths.

A shortest-path tree, T, rooted at destination z contains minimum cost paths from each node, x, to destination z in graph G. A link pointing towards the destination is pointing *down-tree* whereas a link pointing towards the leaf nodes is pointing *up-tree*. T divides the links of E into *tree links* and *non-tree links*, where a tree link lies on the shortest path from a node to destination z. For a down-tree link (x, y), node x (y) is an up-tree (down-tree) neighbor of node y (x). Node y is called the *preferred neighbor* of node x as it provides the shortest path from node x to destination z. A non-tree link is called a *horizontal link*. Two nodes connected by a horizontal link are called *horizontal neighbors*.

Every path $Q(x, z) = (x, \dots, y, z)$ is assigned a *path identifier* which is the identifier of node y. Every node x has a path identifier which is the identifier of its shortest path. The shortest path from node x to destination z and its cost are denoted by $T(x, z)$ and $c(T(x, z))$ respectively. The cost of any path Q is denoted by $c(Q)$.

A *fork* is a node with two or more up-tree neighbors. An *f-segment* is a fork-free tree path segment that connects either a pair of forks or a fork and a leaf node. A *branch link*, (u, f), between a fork, f, and its up-tree neighbor, u, identifies the f-segment containing (u, f). A *branch identification list*, a minimum-length sequence of branch links, of a node $x(\neq z)$ identifies the f-segments which together provide maximum overlap with the shortest path $T(x, z)$. In Figure 1, $T(x, z) = (x, u_1, f_1, u_2, f_2, z)$ where f_1 and f_2 are the only forks in $T(x, z)$. The branch identification list of x is $\langle (u_2, f_2)(u_1, f_1) \rangle$. The branch links (u_2, f_2) and (u_3, f_2) uniquely identify the f-segments (f_2, u_2, f_1) and (f_2, u_3, l) respectively. The branch link (u_2, f_2) identifies a tree segment between the forks f_2 and f_1, and the branch link (u_3, f_2) identifies the tree segment between the fork f_2 and the leaf node l.

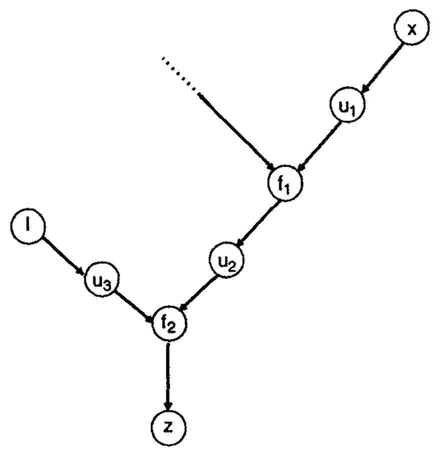

Figure 1: Example of an f-segment, branch-link and branch identification list

A horizontal neighbor, h, of node x can be up-tree, down-tree or neither with respect to node x. If the branch identification list of node h is a proper prefix of the branch identification list of node x, node h is down-tree from node x. If the branch identification lists of nodes x and h are identical and $c(T(h, z))$ is greater (less) than $c(T(x, z))$, node h is up-tree (down-tree) of node x. If the lists are different and neither list is a proper prefix of the other, node h is neither up-tree nor down-tree from node x .

2.2 Problem Statement

The goal of this paper is to find for every node $x \neq z$ a set S_x of disjoint paths from x to a destination node z in a graph G with the following properties:

P1 The set S_x contains the shortest path $T(x,z)$ from node x to the destination z.

P2 Any two paths Q, Q' in S_x are node-disjoint.

P3 A path Q in the set $S_x - \{T(x,z)\}$ contains no more than k horizontal links.

P4 For each path $(x,y)Q(y,z)$ in the set $S_x - \{T(x,z)\}$ and any path $(x,y)Q'(y,z)$ which is node-disjoint from every path in $S_x - \{P \mid c(P) \geq c((x,y)Q(y,z))\}$, $c(Q') \geq c(Q)$ or $(x,y)Q'(y,z)$ contains more than k horizontal links.

In the next section, we propose an algorithm that constructs the set S_x of paths with properties **P1** – **P4** from every node x to the destination z.

2.3 Distributed Algorithm

The solution to the multiple disjoint paths problem assumes that each node knows its preferred and up-tree neighbors. Disjoint paths with properties **P1**-**P4** are constructed by the exchange of two types of messages, *PID* and *ALT*, over the links of the graph G.

Messages

PID messages propagate information about shortest path costs, path identifiers and branch identification lists to all nodes. The general form of this message is *PID*(*nid*, *cst*, *pid*, *bil*) where:

nid : identifier of the node sending the message,
cst : cost of the shortest path from the sending node to the destination,
pid : path identifier of the sending node and
bil : branch identification list of the sending node with the form $\langle(n_1,n_1')(n_2,n_2')\ldots(n_m,n_m')\rangle$ where (n_i,n_i'), $1 \leq i \leq m$ are the only branch-links on the shortest path of the node sending the message.

ALT messages propagate information about the alternate paths through up-tree, down-tree or horizontal neighbors. The general form of such a message is *ALT*(*nid*, *cst*, *pid*, *bis*, *nhl*) where:

nid : identifier of the node sending the message,
cst : cost of the possible alternate path from the sending node to the destination,
pid : path identifier of the possible alternate path,
bis : set of branch links in the possible alternate path and
nhl : number of horizontal links in the possible alternate path.

Labeling Scheme

An important aspect of our distributed algorithm is a *labeling scheme* that labels each node with an identifier called the *path identifier*. Path identifiers are used to determine the disjointedness of paths and to uniquely identify a path in a set of disjoint paths. This scheme uses *PID* messages to inform a node about its path identifier and the path identifiers of its horizontal neighbors.

The destination node z starts a distributed computation by sending *PID*$(z, 0, -, \langle\rangle)$ messages to each neighbor x. The *PID* messages are propagated by the up-tree neighbors of the destination node z after receiving *PID* messages from z. The *PID* messages move up-tree towards the leaf nodes. Every node sends a *PID* message on each of its up-tree and horizontal links after receiving a *PID* message from down-tree. Each tree link carries one *PID* message in the up-tree direction, and each horizontal link carries two *PID* messages sent by the end nodes. No *PID* message is sent over a down-tree link.

An up-tree neighbor, x, of the destination, on receiving a *PID* message over the tree link (x,z), sends a $PID(x,c(x,z),x,bil_x)$ message to each up-tree neighbor, u. If x is a fork, bil_x is equal to $\langle(u,x)\rangle$; otherwise bil_x is $\langle\rangle$. Node x also sends a $PID(x,c(x,z),x,\langle\rangle)$ message to each horizontal neighbor. The destination node z ignores any *PID* message it receives over a horizontal link (x,z).

On receiving a $PID(y,c(T(y,z)),pid_y,bil_y)$ message from its down-tree neighbor y, an intermediate node x learns about its path identifier pid_y, cost of its shortest path $(c(T(y,z))+c(y,x))$, and its branch identification list bil_y. It propagates this information in the *PID* messages to each of its horizontal and up-tree neighbors. If node x is a fork, it appends the branch link, (u,x), to its branch identification list in the *bil* field of the outgoing *PID* message to each up-tree neighbor u.

The *PID* message propagation terminates at the leaf nodes of the shortest-path tree after the leaf nodes have sent and received *PID* messages on their horizontal links.

Node x, on receiving an $ALT(y,c',p',b',n')$ message from neighbor y describing the minimum cost alternate disjoint path, sends an $ALT(x,c'+c(x,y),p',b_v,n_v)$ message to every eligible neighbor v. In this figure, bis_x is the branch identification set of the shortest path of node x. The branch identification set b_v is constructed as follows :

```
1.0 if $v$ is the preferred neighbor of node $x$ then
    $b_v \leftarrow b'$

2.0 if $v$ is a horizontal neighbor of node $x$ then
    2.1 if $y$ is up-tree of $x$ and $x$ is a fork then
            $b_v \leftarrow b' \cup \{(y,x)\} \cup bis_x$
        else
            $b_v \leftarrow b' \cup bis_x$

3.0 if $v$ is an up-tree neighbor of $x$ then
    3.1 if $y$ is up-tree of $x$ and $x$ is a fork then
        3.1.1 if $bis_x \neq \emptyset$ then
                  $b_v \leftarrow b' \cup bis_x \cup \{(y,x),(v,x)\}$
              else
                  for each up-tree neighbor $u$ of $x$ do
                      $b_v \leftarrow b_v \cup \{(u,x)\}$
                  $b_v \leftarrow b_v \cup b'$
    3.2 if $y$ is preferred or horizontal neighbor
            and $x$ is a fork then
        3.2.1 if $bis_x \neq \emptyset$ then
                  $b_v \leftarrow b' \cup bis_x \cup \{(v,x)\}$
              else
                  for each up-tree neighbor $u$ of $x$ do
                      $b_v \leftarrow b_v \cup \{(u,x)\}$
                  $b_v \leftarrow b_v \cup b'$
    3.3 else
            $b_v \leftarrow b'$
```

Figure 2: Rules for constructing the branch identification set for an alternate path

Disjoint Paths

A node uses *PID* messages to inform its neighbors about alternate disjoint paths with one horizontal link. The *ALT* messages are used to inform the neighbors about paths with more than one horizontal link.

A horizontal neighbor, x, of the destination, on receiving a *PID* message over its horizontal link (x,z), determines that it has a disjoint path, (x,z), whose cost and path identifier are $c(x,z)$ and x respectively.

An intermediate node, x, on receiving a $PID(h,c(T(h,z)),pid_h,bil_h)$ message from a horizontal neighbor h, learns about a path, $Q=(x,h)T(h,z)$ and checks if Q can be included in the set S_x. Node x does not propagate the *PID* message it received over its horizontal link (x,h) any further.

Every node x saves the *PID* messages it receives from its down-tree and horizontal neighbors and the last *ALT* message it receives through each neighbor. When node x receives a *PID* or an *ALT* message from a neighboring node y describing an alternate path Q, node x includes the path Q in the set S_x if Q is disjoint from any other path in S_x of lower cost and cheaper than any other path offered by node y. If x includes Q in S_x, it excludes from S_x every path which intersects Q and has a cost greater than $c(Q)$. For every neighbor y of node x, if node x does not have a path in S_x through y, x reconsiders including in S_x the path offered by neighbor y. This is achieved by using the saved information received in the *PID* or last *ALT* messages from y.

Any two alternate paths are disjoint if their path identifiers are different and there is no common branch link in their branch identification sets. The absence of a common branch link in the two sets indicates that the two alternate paths do not intersect a common f-segment. The disjointness of an alternate path and the shortest path for a node is checked by constructing a branch identification set for the shortest path and comparing this set with the branch identification set of the alternate path. The branch identification set of the shortest path $T(x, z)$ is the set of branch links in the branch identification list of node x. Figure 2 contains the rules for constructing the branch identification set of an alternate path.

Each node x keeps track of the cost of its cheapest alternate disjoint path and its eligible neighbors to whom it sent *ALT* messages informing them about such a path. Neighbor eligibility is determined using the rules outlined in Figure 3. When node x gets an $ALT(y, c', p', b', n')$ message from node y describing an alternate disjoint path P with cost $c' + c(x, y)$ which is less than its cheapest alternate disjoint path, node x informs its eligible neighbors about the new path P using two sets of *ALT* messages. In the first set, node x sends an *ALT* message with a cost of -1 to every eligible neighbor w to cancel the previously offered path. In the second set, node x sends an $ALT(x, c' + c(c, y), p', b_v, n_v)$ message to every eligible neighbor v. If v is a horizontal neighbor, n_v is equal to $n' + 1$; otherwise, n_v is equal to n'. The branch identification set b_v is determined as shown in Figure 2. The neighbor receiving the cancellation message may not receive the *ALT* message describing the new path P.

Figure 4 shows the application of rules for constructing the branch identification sets of alternate paths, where the function "$f_{bis}(m)$" projects the branch identification set of an *ALT* message m.

A node, x, does not send *ALT* messages before it receives a *PID* message from its preferred neighbor. In addition, a leaf node x does not inform its eligible neighbors about any path P it received in a *PID* message before it receives *PID* messages from all its neighbors. This helps reduce the number of *ALT* messages containing sub-optimal paths sent by a leaf node. Any *ALT* message sent to an up-tree neighbor x of the destination z is not propagated any further. Destination z does not send any *ALT* messages. Any node whose branch identification set is empty may not propagate any *ALT* message it receives. This modification to *ALT* message propagation will simplify the rules in Figure 2. The "else" part of the "if" statements in 3.1.1 and 3.2.1 will not be needed. It should be noted that, in some special cases which depend on the network topology and link cost assignments, the rules in Figure 2 may cause a node x to exclude some alternate disjoint path from its set S_x. For further details and solutions, see [9].

Node x does not send an *ALT* message to a neighbor v informing it of a new minimum cost alternate disjoint path P if :

- **R1** *Path P was offered to x by v.* This prevents formation of a loop. In this case, node x sends the second cheapest alternate disjoint path to v.
- **R2** *The number of horizontal links in the path P exceeds k.* If node v is a horizontal neighbor, node x will not send it an *ALT* message because the number of horizontal links in path P exceeds the constraint in the problem statement.
- **R3** *Node v has a different path identifier than node x.* Node x does not inform node v about the path P because the cheapest alternate path that node v will have through node x is $((v, x)T(x, z))$ which node v will know about from a *PID* message received from x.
- **R4** *Node v is a horizontal neighbor that is either up-tree or down-tree from node x.* If node v is up-tree of x, then the path P is not disjoint from the shortest path of v. If v is down-tree of x, v will learn about the path P because node x propagates information about the path P along the tree path from x to v.
- **R5** *Node v is not an up-tree neighbor of x, and path P is non-disjoint from the shortest path of node v.* If the alternate path P is not disjoint from the shortest path of v, then node v will disregard this message.

Figure 3: Rules for determining eligibility of neighbors to receive *ALT* messages

Termination

The distributed computation of this algorithm terminates when no node in the network can send an *ALT* message to any of its neighbors. At this time, each node has found a set of disjoint paths. The complete algorithm and proof of its correctness is given in [9].

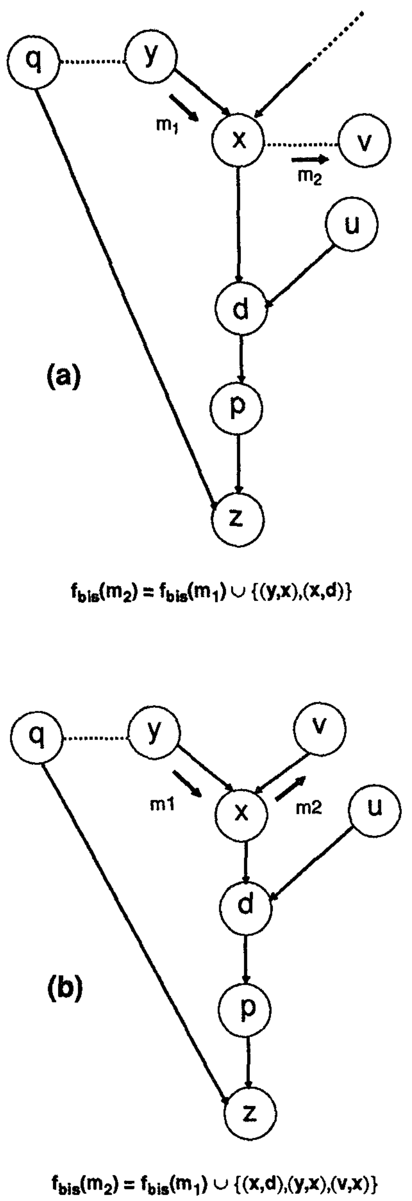
q
y
x
v
u
d
p
z
m_1
m_2
(a)
$f_{bis}(m_2) = f_{bis}(m_1) \cup \{(y,x),(x,d)\}$
q
y
v
x
u
d
p
z
m1
m2
(b)
$f_{bis}(m_2) = f_{bis}(m_1) \cup \{(x,d),(y,x),(v,x)\}$

Figure 4: Examples of changes to branch identification sets

Example

Figure 5, shows an example of a network graph G with a shortest-path tree, T, superimposed on it. The integer associated with a link is its cost. The tree shows the shortest paths from every node to the destination node z. The tree links are indicated by solid directed arrows and the non-tree links by dotted lines. A connected sequence of directed ar-

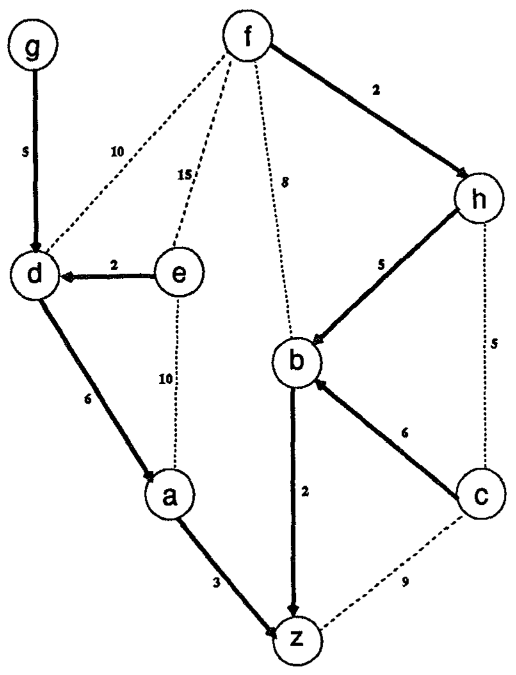

Figure 5: Shortest path tree for an example network

rows from a node to the destination z defines the shortest path from that node to z. This tree i rooted at z and has leaf nodes e, g, f and c.

Figures 6 and 7 illustrate the exchange of *PI* and *ALT* messages respectively. To trace the prop agation of *ALT* messages triggered by *PID* or othe *ALT* messages, ends of arrows are tagged. A ta of an *ALT* message is formed by concatenating th tag of the message which triggered this *ALT* mes sage and the identifier of the receiving node. W point out some special situations in this example.

The destination node z starts the distribute computation by sending *PID* messages to each c its neighbors.

The leaf node c sends *ALT* messages, tagge "1b" and "1h" about the path $Q = (c, z)$, to it neighbors b and h when it receives *PID* message

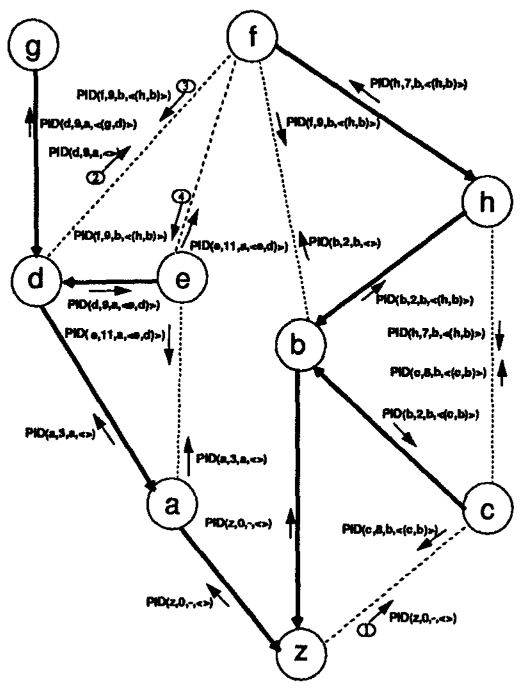
Figure 6: PID message exchanges

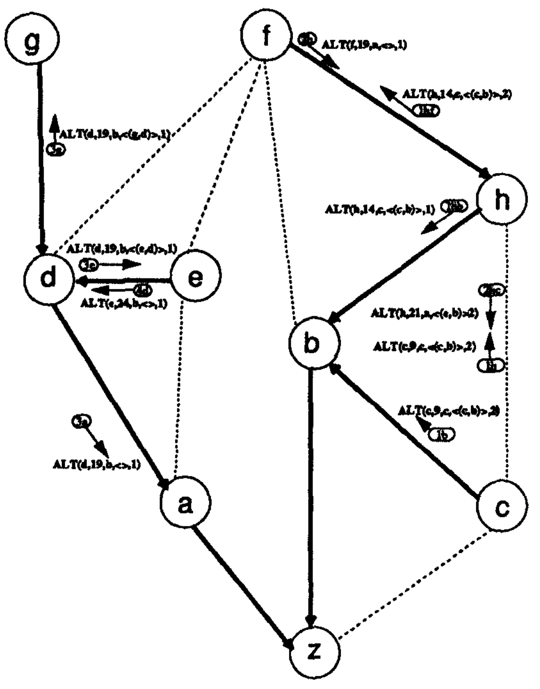
Figure 7: ALT message exchanges

from its neighbors z, b and h.

Node c considers node h as an eligible horizontal neighbor because (1) the path $Q = (c, z)$ is disjoint from the shortest path of node h, (2) the path identifiers of both the nodes are the same and (3) node h is neither up-tree nor down-tree from node c since their branch identification lists,$\langle(c, b)\rangle$ and $\langle(h, b)\rangle$ respectively, are neither identical nor proper prefixes of each other.

The *PID* messages which travel over the horizontal links (f, b) and (h, c) do not initiate *ALT* messages because the nodes b, f, h, and c have identical path identifiers. The same is true for the *PID* messages which travel over the horizontal link (a, e).

The leaf node f sends its down-tree neighbor, h, the *ALT* message tagged "2h" to inform it of an alternate path $Q = (f, d, a, z)$ with one horizontal link. The node f does not send *ALT* messages to nodes e and d because the path identifier of node f is different from that of nodes e and d. The node f does not send an *ALT* message to its horizontal neighbor b because b is down-tree from f. The leaf node g does not send an *ALT* message to its down-tree neighbor because it does not have any alternate path to offer.

The node h sends node c the *ALT* message tagged "2hc" informing it about its second shortest disjoint path $Q = (h, f, d, a, z)$ because node c is providing node h with the minimum alternate disjoint path $P = (h, c, z)$.

3 Simulation Results

In this section, we present simulation results based on our algorithm and the Ogier-Shacham algo-

Network (N, D, d)	Total No. Messages		Total No. Paths		Stretch Factor		Messages/ Path	
	sf	ss	sf	ss	sf	ss	sf	ss
(20,3,6)	3008	6005	1344	760 (1.71)	1.41	1.24	2.24	7.90
(20,7,3)	1267	5646	705	760 (8.95)	2.27	2.20	1.78	7.43
(20,10,3)	1775	6234	651	760 (10.79)	1.52	1.37	2.73	8.20
(20,2,12)	5610	8455	2026	760 (0)	1.32	1.19	2.77	11.13
(20,5,3)	1495	5096	747	760 (3.29)	1.68	1.53	2.00	6.71
(20,12,3)	882	2671	457	448 (0)	1.39	1.33	1.93	5.96
(20,9,2)	1241	4534	587	620 (6.13)	1.62	1.54	2.11	7.31
(20,5,3)	1470	4525	850	760 (0)	1.30	1.21	1.73	5.95
(19,4,4)	1746	3897	858	684 (1.75)	1.26	1.12	2.03	6.30

Table 1: Simulation Results

rithm. The simulation was performed on 9 arbitrarily chosen networks whose parameters are shown in column 1 of Table 1. The notation (N, D, d) represents a network of N nodes with diameter D and average node degree d. All networks are assigned arbitrary link costs except for the last two networks [10] which have unit link costs. The notation "sf" and "ss" refers to the Shortest-First (ours) and Shortest-Sum (Ogier-Shacham) algorithms respectively. To compare the quality of paths found by the two algorithms, we use the stretch factor for a path $P(x, z)$ which is defined as $c(P(x, z))/c(T(x, z))$. Table 1 summarizes the results of the simulations.

The second column shows the total number of messages generated by each algorithm to find multiple disjoint paths from every node to every other node. These results show that the Ogier-Shacham algorithm requires from one and a half to four and a half times more messages than our algorithm.

The third column shows the total number of disjoint paths found for every node in the network. The number in parentheses is the percentage of nodes whose shortest path is disturbed, that is the shortest path is not one of the two paths found by the Ogier-Shacham algorithm. The Ogier-Shacham algorithm had the shortest path as one of the paths over 90% of the time. In general, the number of paths found by each node is dependent on the network topology and the link cost assignments.

The fourth column shows the average stretch factor of the two shortest paths found for each node using our algorithm and the average stretch factor of the two paths found using the Ogier-Shacham algorithm. From the results, we conclude that for finding paths of comparable quality, our algorithm uses fewer messages than the Ogier-Shacham algorithm.

Our algorithm finds more paths than the Ogier-Shacham algorithm. It is important to compare the performance of our algorithm with that of Ogier-Shacham using the number of messages per path found. This is shown in the fifth column. In general, our algorithm requires 3 to 4 times fewer messages per path.

4 Summary and Conclusions

In this paper, we describe a distributed distance-vector algorithm for finding multiple node-disjoint paths in a computer communication network. This algorithm includes the shortest path as one of the disjoint paths. Multiple disjoint paths can be used to increase the effective bandwidth between pairs of nodes, reduce congestion in a network, and reduce the probability of dropped packets.

Using simulation, we compare our algorithm with an alternative approach that does not necessarily include the shortest path as one of its disjoint paths. This alternative approach finds a pair of disjoint paths such that the sum of the costs is minimized over all possible pairs of disjoint paths. We compare the two algorithms using simulation. Our algorithm requires 3 to 4 times fewer messages to discover paths of comparable quality. For a complete discussion of our algorithm to construct multiple disjoint paths, see [9].

Acknowledgement

The authors would like to thank Mr. P. Tsuchiya for helpful discussions.

References

[1] A. Itah and M. Rodeh. The multi-tree approach to reliability in distributed networks. In *Proc. 25th Symposium on FOCS*, 1984.

[2] D. M. Topkis. A K shortest path algorithm for adaptive routing in communications networks. *IEEE Transactions on Communications*, 36, 1988.

[3] J. W. Surballe. Disjoint paths in a network. *Networks*, 4, 1974.

[4] J. W. Surballe and R. E. Tarjan. A quick method of finding shortest pairs of disjoint paths. *Networks*, 14, 1984.

[5] R. Ogier and N. Shacham. A distributed algorithm for finding shortest pairs of disjoint paths. In *Proc. IEEE INFOCOM '89.*

[6] C. Cheng, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves. A distributed algorithm for finding K disjoint paths of minimum total length. In *Proc. 28th Annual Allerton Conference on Communication, Control, and Computing, Urbana, Illinois*, October 1990.

[7] P. F. Tsuchiya. The landmark hierarchy: Description and analysis. Technical Report MTR W87-87W00152, MITRE, June 1987.

[8] D. P. Sidhu, S. Abdallah, and R. Nair. A distance vector algorithm for alternate path routing. Submitted for publication, 1990.

[9] D. P. Sidhu, R. Nair, and S. Abdallah. A distributed algorithm for finding multiple disjoint paths. Submitted for publication, 1990.

[10] M. Garner, V. Haimo, I. Loobeek, D.Davis, and M. Frishkopf. Type-of-service routing: Modeling and simulation. Technical Report 6364, BBN Communications Corporation, January 1987.