A More Efficient Path-Finding Algorithm

Shree Murthy and J.J. Garcia-Luna-Aceves*
Baskin Center for
Computer Engineering and Information Sciences
University of California
Santa Cruz, CA 95064

Abstract

In this paper, we present a new routing algorithm, which we call path-finding algorithm (PFA). It drastically reduces the possibility of temporary routing loops, which accounts for its fast convergence properties. Like other path-finding algorithms, PFA operates by specifying the second-to-last hop to each destination, in addition to the distance to the destination. A detailed proof of correctness and complexity is presented elsewhere. PFA's performance is compared quantitatively by simulation with DUAL (a loop-free routing algorithm) and an ideal link-state algorithm (ILS). A number of parameters, including the length of the messages and the number of steps required for convergence, are used in the comparison. The simulation results indicate that PFA constitutes a very efficient distance-vector algorithm. It provides about 50% improvement in performance compared to DUAL in terms of the convergence time and the number of updates after single link failures, and provides comparable or better convergence speed and traffic overhead than ILS, with orders of magnitude fewer CPU cycles.

1 Introduction

Routing in today's computer networks and Internet is accomplished by distributed shortest-path routing algorithms. The distributed Bellman-Ford (DBF) algorithm [1] has been used in many well known routing protocols. However, DBF takes a long time to converge after link failure or link-cost changes because of bouncing effect and counting-to-infinity problem [5]. Path-finding algorithms can be an attractive alternative to DBF for distributed routing as they eliminate counting-to-infinity problem. However, the path-finding algorithms proposed in the past incur substantial temporary loops in the paths specified by the predecessor information before they converge, which leads to slower convergence.

This paper presents a new algorithm, simply called *path finding algorithm* (PFA), that substantially reduces the number of cases in which routing loops can occur. Its performance is compared with the performance of an ideal topology broadcast (or link state) algorithm and DUAL, which has been shown to be a very efficient distance-vector algorithm. The main feature of PFA is the notion of *predecessor* or second-to-last hop. Using this information, each node can infer the path implicit in a distance-table entry without excessive overhead. Each node maintains the shortest-path spanning tree reported by its neighbors, and uses this information and the information regarding the cost of the adjacent links to generate its own shortest-path spanning trees. The fact that PFA reduces temporary looping accounts for its superior performance over DUAL and the ideal link state algorithm.

The rest of the paper is organized as follows. Section 2 presents the network model assumed in PFA and introduces the notation used throughout this paper. Section 3 provides the description of PFA along with an example illustrating key aspects of its operation. Section 4 presents simulation results showing that PFA constitutes an attractive alternative for the implementation of routing protocols that rely on the exchange of vectors of distances. Finally, Section 5 presents our conclusions.

2 Network Model

We have modeled a computer network as an undirected graph represented as $G(V, E)$, where V is the set of nodes and E is the set of links (or edges) connecting the nodes. Each node represents a router and is a computing unit involving a processor, local memory and input and output queues with unlimited capacity. A functional bidirectional link connecting the nodes is assigned a positive weight in each direction. A link is assumed to exist in both the directions at the same time. All messages received (transmitted) by a node are put in the input (output) queue on a first-come-first-serve basis and are processed in that order. An underlying protocol assumes that

- Every node knows its neighbors. This implies that, within a finite time a node detects the existence of a new neighbor or the loss of connectivity with a neighbor.
- All packets transmitted over an operational link are received correctly and in the proper sequence within a finite time.
- All update messages, changes in the link-cost, link failures and link recoveries are processed one at a

*This work was supported in part by the Office of Naval Research under Contract No. N-00014-92-J-1807 and by the Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0175

1058-6393/95 $4.00 © 1995 IEEE

```
Procedure Update(i, k)
when router i receives an update on link (i, k)
(0) begin
      update=0; RTEMP^i ← φ;
      DTEMP^{i,b} ← φ for all neighbors b
(1)   for each triplet (j, D^k_j, p^k_j) in V^{k,i}, j ≠ i do
      begin
            D^i_jk ← d_ik + D^k_j; p^i_jk ← p^k_j;
(2)         for all neighbors b
                if k is in the path from i to j in
                  the distance table through neighbor b
                then D^i_jb ← D^i_kb + D^k_j; p^i_jb ← p^k_j
      end
(3)   begin
            if there are b and j such that
            (D^i_jb < D^i_j) or ((D^i_jb > D^i_j) and (b = s^i_j));
            then Call RT_Update;
      end
(4)   begin
            if (RTEMP^i ≠ φ) then
              for each neighbor b do
              begin
                    for each triplet t = (j, D^i_j, p^i_j) in RTEMP^i do
                        if b is not in the path from i to j
                          then DTEMP^{i,b} ← DTEMP^{i,b} ∪ t;
                    send DTEMP^{i,b} to neighbor b;
              end
      end
end

Procedure Change(i, k, d_ik)
when d_ik changes value do
(7) begin
      update the distance table entry at node i
          D^i_jk ← d_ik + D^k_j; p^i_jk ← p^k_j;
      Go to Step (2);
end

Procedure Failure(i, k)
when link (i, k) fails do
(5) begin
      delete column k in D_i
      if there is a destination j such that k = s^i_j
      then Call RT_Update;
      Go to Step (4);
end

Procedure Recover(i, k, d_ik)
when link (i, k) comes up do
(6) begin
      insert column k in D_i
      respond as if a single entry in V^{k,i} = (k, d_ik, i)
      is received on link (i, k)
      copy whole routing table into DTEMP^{i,k} and
      send it to k
end

Function In_Path(Node,Neighbor,Dest, neigh)
begin
      p ← p^{Node}_{Dest,neigh};
      if (p = Node) then return(false);
      else if (p = Neighbor) then return(true);
           else In_Path(Node,Neighbor,p,neigh);
end

Procedure RT_Update
begin
      initialize all destinations to be unmarked;
      for any unmarked destination j do
      begin
          if there is no finite distance in row j
          then mark j as undetermined;
          else begin
                TV ← φ;
                pick any minimum distance D^i_jb
                c ← p^i_jb, TV ← TV ∪ c;
                repeat c ← p^i_cb, TV ← TV ∪ c;
                   until D^i_cb is not minimum of row
                   c or p^i_cb = i or p^i_cb is marked
                if ((p^i_cb is marked as undetermined) or
                  (D^i_cb is not minimum of row c))
                then mark each node in TV as undetermined
                else begin
                       mark each node in TV as determined
                       D^i_j ← D^i_jb; p^i_j ← p^i_jb; s^i_j ← b;
                     end
                end
      end
      copy the routing vector to RTEMP^i if the distance or
      predecessor has changed
end
```

Figure 1: PFA Specification

time in the order in which they occur.

When a link fails, the corresponding distance entry in a node's distance and routing tables are marked as infinity. A node failure is modeled as all links incident on that node failing at the same time. A change in the operational status of a link or a node is assumed to be notified to its neighboring nodes within a finite time. These services are assumed to be reliable and are provided by lower level protocols.

Throughout the paper the following notations is used:

j:	Destination node identifier $j \in N$
b, k:	Neighbor nodes
D^i_{jk}:	Distance entry at node i to destination j through neighbor k in the distance table
D^i_j:	Distance entry at node i to destination j in the routing table
p^i_{jk}:	Predecessor from i to j through k in the distance table
p^i_j:	Predecessor from i to j in the routing table
s^i_j:	Successor to destination j from a given node in the routing table
d_{ik}:	Link cost from i to neighbor k
N_i:	Set of neighbors of i

3 PFA Description

Figure 1 defines PFA. PFA consists of procedures used for topology changes (*Failure*, *Recover*, and *Change*), path traversal (*In_Path*), updating the routing table (*RT_Update*) and a procedure to process update messages (*Update*). The main feature of PFA is the notion of second-to-last hop or *predecessor*. Using predecessor information, each node can infer the path implicit in a distance entry without excessive overhead.

Each node maintains a *distance table*, a *routing table* and a *link-cost table*. The distance table is a matrix containing the distance and predecessor entries (path information) for all the destinations through all its neighbors. The routing table is a column vector of minimum distance to each destination and its corresponding predecessor and successor information. The link-cost table lists the cost of each link adjacent to the node; the cost of a failed link is considered to be infinity. An update message contains the source and the destination node identifiers, and the distance and predecessor for one or more destinations.

When a node i receives an update message from its neighbor k regarding destination j, the distance and the predecessor entries in the distance table are updated (Step 1). A unique feature of PFA is that, node i also determines if the path to destination j

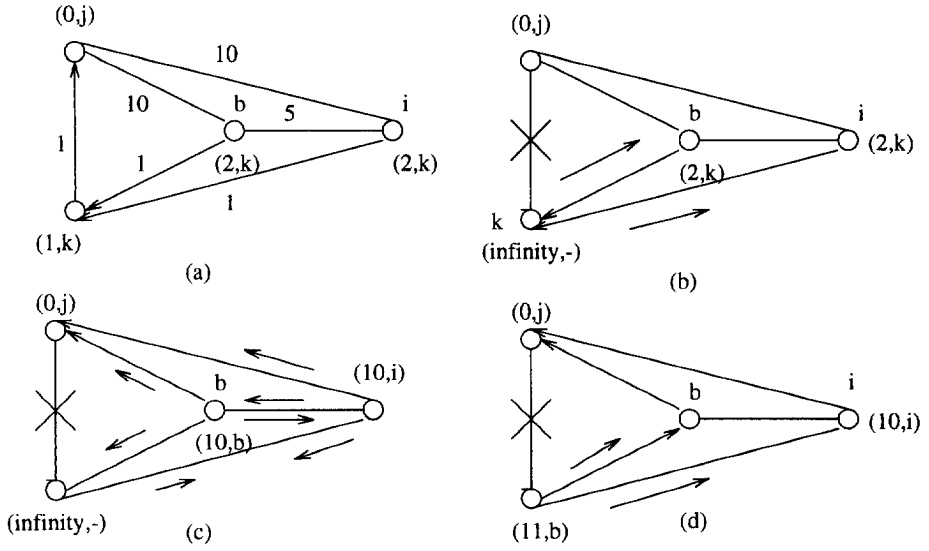

Figure 2: Example of the algorithm's operation

through any of its other neighbors $\{b \in N_i | b \neq k\}$ includes node k. If the path implied by the predecessor information reported by node b includes node k, then the distance entry of that path is also updated as $D^i_{jb} = D^i_{kb} + D^k_j$ and the predecessor is updated as $p^i_{jb} = p^k_j$. Thus, a node can determine whether or not an update received from k affects its other distance and routing table entries. Before updating the routing table, node i checks for all simple paths to j reported by its neighbors, and the shortest of these simple paths becomes the path from i to j (Procedure RT_Update). This implies that at each stage, node i checks for the simple paths and avoids loops. Link or node failures, recoveries and link-cost changes are handled similarly (Steps (5), (6) and (7)).

In contrast to PFA, which makes a node i check the consistency of predecessor information reported by *all* its neighbors each time it processes an event involving a neighbor k, all previous path-finding algorithms [2, 3, 4] check the consistency of the predecessor only for the neighbor associated with the input event. This unique feature of PFA accounts for its fast convergence after a single resource failure or recovery as it eliminates more temporary looping situations than previous path-finding algorithms.

The following example illustrates the working of the algorithm. Consider the four node network shown in Figure 2(a). Let PFA be used in this network. All links and nodes are assumed to have the same propagation delays. Link-costs are as indicated in the figure and are assumed to be the same in both the directions. Node i is the source, j is the destination and node k and b are the neighbors of node i. The arrows next to links indicate the direction of updates messages and the label in parentheses gives the distance and the predecessor to destination j. The figure focuses on update messages to destination j only.

When link (j, k) fails, nodes j and k send update messages to their neighboring nodes as shown in Figure 2(b). In this example, node k is forced to report an infinite distance to j as nodes b and i have reported node k as part of their path to destination j. Node b processes node k's update and selects link (b, j) to destination j. This is because of step(2) of the algorithm which forces node b to purge any path to node j involving node k. Also, when i gets node k's update message, i updates its distance table entry through neighbor k and checks for the possible paths to destination j through any other neighboring nodes. Thus, a node examines the available paths through its other neighboring nodes and updates the distance and the routing table entries accordingly. This results in the selection of the link (i, j) to the destination j (Figure 2(c)). When node i receives b's update reporting an infinite distance, node i does not have to update its routing table as it already has correct path information (Figure 2(d)). Similarly, updates sent by node k reporting a distance of 11 to destination j will not affect the path information of nodes i and b. This illustrates how step(2) of PFA helps in the reduction of the formation of temporary loops in the explicit paths.

The proofs of correctness, convergence and complexity of PFA are given elsewhere [6]. The worst-case complexity of PFA has been found to be $O(h)$ for single recovery/failure, h being the height of the tree.

4 Simulation Results

To gain insight into the average-case performance of the algorithms, we have developed simulations using an actor-based, discrete-event simulation language called *Drama* [7], together with a network simulation library. Link failures and recoveries are simulated by sending link status message to the nodes at the end points of the appropriate links. Node failures can be treated as all links connecting to that node going down

Table 1: Simulation Results for Los-Nettos

Parameter	PFA		DUAL		ILS	
	mean	sdev	mean	sdev	mean	sdev
Link-Failure						
Event Count	45.7	17.9	49.9	18.6	29.0	5.8
Packet Count	13.5	6.01	32.6	11.8	27.0	5.8
Duration	2.86	0.74	6.7	1.33	4.2	0.88
Operation Count	62.4	18.03	69.9	18.6	724.1	27.3
Link-Recovery						
Event Count	91.3	15.5	45.7	7.45	33.3	1.86
Packet Count	18.0	5.04	17.0	7.25	31.9	1.86
Duration	2.93	0.46	3.71	0.88	3.86	0.46
Operation Count	109.6	24.6	65.7	7.45	944.0	45.8
Node-Failure						
Event Count	135.6	79.5	73.0	25.4	31.8	9.6
Packet Count	39.8	17.5	45.5	3.26	26.7	7.19
Duration	5.82	2.85	6.91	0.99	4.09	0.51
Operation Count	195.0	112.2	123.9	50.2	702.9	204.3
Node-Recovery						
Event Count	221.1	117.9	94.3	40.5	56.2	13.4
Packet Count	30.4	10.3	41.0	12.4	51.1	10.8
Duration	3.18	0.38	4.7	0.44	4.4	0.5
Operation Count	274.9	136.4	145.2	66.5	1698.8	478.4

at the same time and the link cost changes can be treated as a link failing and recovering with a new link cost. All simulations are done assuming unit propagation time and zero packet processing time at each node. If a link fails when packets are in transit, the packets are dropped.

For simulating the routing algorithm, the node receives a packet and responds to it by running the routing algorithm and queueing the outgoing packets and processing the updates one at a time in the order in which they arrive. In our simulations, packet processing time is kept as zero. Drama's internals ensure that all the packets at a given time are processed before new updates are generated.

The performance of PFA has been compared with DUAL and an ideal link state (ILS) algorithm which uses Dijkstra's shortest-path algorithm [1] at each node. We have instrumented our simulations using the following four quantities:

Events: total number of updates

Packets: total number of packets transmitted over the network

Duration: total time elapsed for the algorithm to converge

Operations: total number of operations performed by the algorithm

Counters are used to instrument these quantities. These counters can be reset at various points and the values of these counters are printed when the algorithm converges (that is, when the event queue empties).

Simulation results are obtained for all single failures and recoveries of links and nodes. The routing algorithm was allowed to converge after each such change. The simulations were run on several network topologies after a series of tests on smaller topologies for debugging purposes. The results for link and node, failure and recovery for *Los-Nettos* are presented in Table 1. The results for other network topologies can be found elsewhere [6].

PFA has a better overall average performance than ILS after the recovery of a single node or link. The performance of PFA is comparable to ILS after the failure of a single node or a link. This is a remarkable improvement over DUAL which requires approximately twice the number of steps to converge than ILS after failures. For link failures, the operation count of PFA is almost the same as that of DUAL with the duration to converge being half that of DUAL. The number of packets (messages) exchanged among nodes is almost 50% less than that of DUAL. However, the event count and the operation count is about 2 to 3 times higher than DUAL; the operations count of PFA is up to an order of magnitude less than that of ILS.

5 Conclusion

In this paper, we have presented a new path-finding algorithm, PFA, that reduces the occurrence of temporary routing loops without the need for internodal synchronization mechanism or the exchange of complete path information. The performance of the al-

gorithm is compared with DUAL and an ideal link-state algorithm quantitatively by simulation. In terms of convergence time after link failures, PFA provides about 50% improvement in performance as compared to DUAL by exchanging lesser number of messages than DUAL and its performance is comparable to that of ILS. The worst-case complexity of the algorithm has been shown (elsewhere) to be $O(h)$, where, h is the height of the tree.

References

[1] D. Bertsekas and R. Gallager, *Data Networks*, Second Ed. Prentice Hall, Inc. 1992.

[2] C. Cheng, R. Reley, S. P. R Kumar and J. J. Garcia-Luna-Aceves, "A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect", *ACM Computer Communications Review*, 19 (4), 1989, pp.224–236.

[3] P.A. Humblet, "Another Adaptive Shortest-Path Algorithm", *IEEE Trans. Comm.*, Vol.39, No.6, June 1991, pp.995–1003.

[4] B. Rajagopalan and M. Faiman, "A Responsive Distributed Shortest-Path Routing Algorithm within Autonomous Systems," *Journal of Internetworking: Research and Experience*, Vol. 2, No. 1, March 1991, pp. 51-69.

[5] M.S. Sloman and X. Andriopoulos, "A Routing Algorithm for Interconnected Local Area Networks", *Computer Networks and ISDN Systems*, 1985, pp.109–130.

[6] Shree Murthy, "Design and Analysis of Distributed Routing Algorithms", *Masters Thesis*, University of California, Santa Cruz, June 1994.

[7] W. T. Zaumen, "Simulations in Drama", *Network Information System Center, SRI International*, Menlo Park, California, January 1991.