A Path-Finding Algorithm for Loop-Free Routing

J.J. Garcia-Luna-Aceves and Shree Murthy

Abstract— **A loop-free path-finding algorithm (LPA) is presented; this is the first routing algorithm that eliminates the formation of temporary routing loops without the need for internodal synchronization spanning multiple hops or the specification of complete or variable-size path information. Like other previous algorithms, LPA operates by specifying the second-to-last hop and distance to each destination; this feature is used to ensure termination. In addition, LPA uses an inter-neighbor synchronization mechanism to eliminate temporary routing loops. A detailed proof of LPA's correctness and loop-freedom property is presented and its complexity is evaluated. LPA's average performance is compared by simulation with the performance of algorithms representative of the state of the art in distributed routing, namely an ideal link-state (ILS) algorithm, a loop-free algorithm that is based on internodal coordination spanning multiple hops (DUAL) and a path-finding algorithm without the inter-neighbor synchronization mechanism. The simulation results show that LPA is a more scalable alternative than DUAL and ILS in terms of the average number of steps, messages, and operations needed for each algorithm to converge after a topology change.**

1 Introduction

RIP [9] is widely used in internets today. However, it is based on the distributed Bellman-Ford algorithm (DBF) for shortest-path computation [1], which suffers from *bouncing effect* and *counting-to-infinity* problems. These problems are overcome in one of three ways in existing Internet routing protocols. OSPF [16] relies on broadcasting complete topology information among routers, and organizes an Internet hierarchically to cope with the overhead incurred with topology broadcast. BGP [12] exchanges distance vectors that specify complete path to destinations. EIGRP [2] uses a loop-free routing algorithm called DUAL [5], which is based on internodal coordination that can span multiple hops; DUAL also eliminates temporary routing loops.

Recently, distributed shortest-path algorithms [3, 5, 8, 10, 17, 19] that utilize information regarding the length and second-to-last hop (predecessor) of the shortest path to each destination have been proposed to eliminate the performance problems of DBF. We call these type of algorithms *path-finding algorithms*. Although these algorithms provide a marked improvement over DBF, they do not eliminate the possibility of temporary loops. All the loop-free algorithms reported to date rely on mechanisms that require routers to either synchronize along multiple hops [5, 11, 15], or exchange path information that can include all the routers in the path from source to destination [7]. This paper presents the loop-free path-finding algorithm (LPA) which is the first routing algorithm that is loop-free at every instant and does not use either of these two techniques.

Like previous path-finding algorithms, LPA eliminates the counting-to-infinity problem of DBF using predecessor information. Because each router reports to its neighbors the predecessor to each destination, any router can traverse the path specified by the predecessors from any destination back to a neighbor router to determine if using that neighbor as its successor would create a path that contains a loop (i.e., involves the router itself). Furthermore, a router detects a temporary loop within a finite time that depends on the speed with which correct predecessor information reaches the router, and not on the distance values of the paths offered by its neighbors; therefore, temporary loops are detected much faster than in DBF and its variations.

Of course, updates take time to be propagated and routers have to update their routing tables using information that can be out of date, which can lead to temporary loops. In LPA, when a router detects that it can create a routing table loop if it changes its successor to a destination, it blocks such a potential loop. The router accomplishes this by reporting an infinite distance for the destination to all its neighbors and by waiting for those neighbors to acknowledge its message with their own distances and predecessor information, before the router changes its successor. Because of the overhead involved, a router should not send a query every time it has to change its successor to a destination; a router decides when to block a potential loop by comparing the distances reported by its neighbors against a *feasible distance*, which is defined to be the smallest value achieved by the router's own distance since the last query sent by the router. The router is forced to block a potential loop with a query only when no neighbor reports a distance smaller than the router's own feasible distance. This feature accounts for the low overhead incurred in LPA to accomplish loop-free paths at every instant.

The rest of the paper is organized as follows. Section 2 presents the network model assumed in LPA. Section 3 provides a description of LPA. Section 4 and Section 5 provide a detailed proof of LPA's loop-freedom and convergence to

J.J. Garcia-Luna-Aceves is with the Computer Engineering Department, University of California, Santa Cruz, California. Shree Murthy is with Sun Microsystems, Inc., Mountain View, California

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

correct routing tables, respectively. Section 6 addresses the complexity of LPA, analyzes its average performance by simulation, and compares it with other algorithms. Finally, Section 7 presents our conclusions.

2 Network Model

A computer network is modeled as an undirected finite graph represented as $G(N, E)$, where N is the set of nodes and E is the set of edges or links connecting the nodes. For simplicity, each node represents a router that is a computing unit involving a processor, local memory and input and output queues with unlimited capacity. LPA can be applied to networks in which nodes represent address ranges [22]. A functional bidirectional link connecting nodes i and j is represented by (i, j) and is assigned a positive weight in each direction. The link is assumed to exist in both directions at the same time. All the messages received (transmitted) by a node are put in the input (output) queue and are processed on a first-come-first-serve basis. Each node has a unique identifier. Any link cost can vary in time but is always positive. The distance between two nodes in the network is measured as the sum of the link costs of the shortest path between the nodes. An underlying protocol assures that:

- Within a finite time, a node detects the existence of a new neighbor, loss of connectivity with a neighbor, or the change in the cost of an adjacent link.
- All packets transmitted over an operational link are received correctly and in the proper sequence within a finite time. This assumption is made for convenience. Reliable message transmission can be easily incorporated into the routing protocol (e.g., [16, 21]).
- All update messages, changes in the link-cost, link failures and link recoveries are processed one at a time in the order in which they occur.

When a link fails, the corresponding distance entry in the node's distance and routing tables are marked as infinity. A node failure is modeled as all the links incident on that node failing at the same time.

A *path* from node i to node j is a sequence of nodes in which (i, n_1), (n_x, n_{x+1}), (n_r, j) are the links in the path. A *simple path* from i to j is a sequence of nodes in which no node is visited more than once. The paths between any pair of nodes and their corresponding distances change over time in a dynamic network. At any point in time, node i is connected to node j if a path exists from i to j at that time. The network is said to be connected if every pair of operational nodes are connected at a given time.

3 LPA Description

3.1 Information Stored and Exchanged

In LPA's description, the time at which the value of a variable X of the algorithm applies is specified only when it is necessary; the value of X at time t is denoted by $X(t)$.

Each router maintains a *distance table*, a *routing table* and a *link-cost table*. The distance table at each router i is a matrix containing, for each destination j and for each neighbor k of router i, the distance and the predecessor reported by router k, denoted by D^i_{jk} and p^i_{jk}, respectively. The set of neighbors of router i is denoted by N_i.

The routing table at router i is a column vector containing, for each destination j, the minimum distance (D^i_j), the predecessor (p^i_j), the successor (s^i_j), and a marker (tag^i_j) used to update the routing table. For destination j, tag^i_j specifies whether the entry corresponds to a simple path ($tag^i_j = correct$), a loop ($tag^i_j = error$) or a destination that has not been marked ($tag^i_j = null$).

The link-cost table lists the cost of each link adjacent to the router. The cost of the link from i to k is denoted by d_{ik} and is considered to be infinity when the link fails.

An update message from router i consists of a vector of entries reporting incremental updates to its routing table; each entry specifies an update flag (denoted by u^i_j), a destination j, the reported distance to that destination (denoted by RD^i_j), and the reported predecessor in the path to the destination (denoted by rp^i_j). The update flag indicates whether the entry is an update ($u^i_j = 0$), a query ($u^i_j = 1$) or a reply to a query ($u^i_j = 2$). The distance in a query is always set to ∞.

Because every router reports to its neighbors the predecessor in the shortest path to each destination, the complete path that a router assumes to any destination (called its implicit path to the destination) at a given time t' is known by the router's neighbors at the subsequent time $t'' > t'$. This is done by means of a path traversal routine on the predecessor entries reported by the router. In the specification of LPA, the successor to destination j for any router is simply referred to as the successor of the router, and the same reference applies to other information maintained by a router. Similarly, updates, queries and replies refer to destination j, unless stated otherwise. Figures 1 and 2 specify LPA in pseudo-code. The rest of this section provides an informal description of LPA.

The procedures used for initialization are *Init1* and *Init2*; Procedure *Message* is executed when a router processes an update message; procedures *linkUp*, *linkDown* and *linkChange* are executed when a router detects a new link, the failure of a link, or the change in the cost of a link. We refer to these procedures as event-handling procedures. For each entry in an update message, Procedure *Message* calls procedure *Update*, *Query*, or *Reply* to handle an update, a query, or a reply, respectively. An important characteristic of all event-handling procedures is that they mark $tag^i_j = null$ for each destination j affected by the input event.

Router i initializes itself in passive state with an infinite distance for all its known neighbors and with a zero distance to itself. After initialization, router i sends updates containing the distance to itself to all its neighbors.

3.2 Distance Table Updating

When router i receives an input event regarding neighbor k (an update message from neighbor k or a change in the cost or status of link (i,k)), it updates its link-cost table with the new value of link d_{ik} if needed, and then executes procedure DT. The intent of this procedure is for the router to erase the outdated path information in the distance table by making path information from all neighbors consistent with the latest update. To accomplish this, DT updates the distance and predecessor of neighbor k as $D^i_{jk} = D^k_j$ and $p^i_{jk} = p^i_k$ for each destination j affected by the input event. In addition, DT determines whether the path to any destination j through any of the other neighbor of router i includes neighbor k. This is done by traversing the path specified by the predecessor entries reported by a neighbor from destination j towards node i. If the path implied by the predecessor reported by router b ($b \neq k$ and $b \in N_i$) to destination j includes router k, then node i assumes that b has outdated path information and substitutes the subpath from k to j reported by b as part of its path to j with the path to i reported by k itself. This is easily done by updating $D^i_{jb} = D^i_{kb} + D^k_j$ and $p^i_{jb} = p^k_j$.

The example in Figure 3 illustrates how procedure DT helps to expedite LPA's convergence. In the example, router x has reported to i its predecessors to y and j, and i infers that x's path to j is xyj. Router c has reported to i its predecessors to d, a, b and j, and i infers that c's path to j is $cdabj$. Router a has reported to i its predecessors to b and j, and i infers that a's path to j is abj. With these conditions, assume that a sends i an update stating that $D^a_j = \infty$ and $p^a_j = null$. Router i uses procedure DT to ensure that the path information from the other neighbors reflects the most recent update obtained from any other neighbors for any destination. Since c's path to j includes a, i substitutes the out-of-date subpath abj in c's path information with the information supplied by a, which makes the path from c to j non-existent. The path from x to j does not include a and i does not change x's information. Note that i does not have to wait for an update from c to infer that it should not use c as successor to j.

3.3 Blocking Temporary Loops

The example shown in Figure 4 illustrates the possibility of looping, even when path information is used. In the example, it is assumed that a has reported the implicit path aj to i and that b has reported the implicit path $bcdj$ to i. Furthermore, this path information is outdated, because c has changed its successor from d to e and the new path information has not reached i. If link (i,a) fails, simply using path information would permit i to use b as successor to j. However, this would create a temporary routing loop.

To eliminate temporary loops, a router i forces its neighbors not to use it as a successor (next hop) when it detects the possibility of creating a temporary loop before i changes its own successor. This is done using an interneighbor synchronization mechanism based on the notion of feasible distance.

The feasible distance of router i for destination j (denoted by FD^i_j) is the smallest value achieved by its own distance to j since the last time i initialized itself or sent a query reporting an infinite distance to j. LPA allows a router to use a neighbor k as its successor to destination j only if it satisfies the following condition.

Feasibility Condition (FC): If at time t router i needs to update its current successor, it can choose as its new successor $s^i_j(t)$ any router $n \in N_i(t)$ such that $D^i_{jn}(t) + d_{in}(t) = D_{min}(t) = Min\{D^i_{jx}(t) + d_{ix}(t) | x \in N_i(t)\}$ and $D^i_{jn}(t) < FD^i_j(t)$. If no such neighbor exists and $D^i_j(t) < \infty$, router i must keep its current successor. If $D_{min}(t) = \infty$ then $s^i_j(t) = null$.

FC is used to establish an ordering of routers along a given loop-free path to j, i.e., all the routers in a loop-free path to j have feasible distances to j that decrease as j is approached. If router i does not find neighbor that satisfies FC, it is forced to send a query to its neighbors reporting an infinite distance to j and waits for the replies before it can change its own route. Because every router uses FC to decide whether to adapt a successor or to block paths through itself (as described in Section 3.4), no temporary loops can exist.

3.4 Routing Table Updating

After procedure DT is executed, the way in which router i updates its routing table for a given destination depends on whether router i is *passive* or *active* for that destination. A router is passive if it has a *feasible successor*, or has determined that no such successor exists and is active if it is searching for a feasible successor. A feasible successor for router i with respect to destination j is a neighbor router that satisfies FC.

When router i is passive, it reports the current value of D^i_j in all its updates and replies. While router i is active, it sends an infinite distance in its replies and queries. An active router cannot send an update regarding the destination for which it is active, this is because any update sent during active state would necessarily have to report an infinite distance to ensure the correct operation of the inter-neighbor synchronization mechanism used in LPA.

If router i is passive when it processes an update for destination j, it determines whether or not it has a feasible successor, i.e., a neighbor router that satisfies FC.

If router i finds a feasible successor, it sets FD^i_j equal to the smaller of the updated value of D^i_j and the present value of FD^i_j. In addition, it updates its distance, predecessor, and successor making sure that only simple paths are used, as described in Section 3.5.

Router i then prepares an update message to its neighbors if its routing table entry changes. Alternatively, if router i finds no feasible successor, then it sets $FD^i_j = \infty$ and updates its distance and predecessor to reflect the information reported by its current successor. If $D^i_j(t) = \infty$, then $s^i_j(t) =$ null. Router i also sets the reply status flag

($r^i_{jk} = 1$) for all $k \in N_i$ and sends a query to all its neighbors. Router i is then said to be *active*, and cannot change its path information until it receives all the replies to its query.

Queries and replies are processed in a manner similar to the processing of an update described above. If the input event that causes router i to become active is a query from its neighbor k, router i sends a reply to router k reporting an infinite distance. This is the case, because router k's query, by definition, reports the latest information from router k, and router i will send an update to router k when it becomes passive if its distance is smaller than infinity. A link-cost change is treated as a number of updates.

Once router i is active for destination j, it may not have to do anything more regarding that destination after executing procedures RT and DT as a result of an input event. However, when router i is active and receives a reply from router k, it updates its distance table and resets the reply flag ($r^i_{jk} = 0$).

Router i becomes passive at time t when it receives replies from all its neighbors indicating that they have processed its query. As a result, router i is free to choose any neighbor that provides the shortest distance, if there is any. If such a neighbor is found, router i updates the routing table with the minimum distance as described for the passive state and sets $FD^i_j = D^i_j$.

A router does not wait indefinitely for replies from its neighbors because a router replies to all its queries regardless of its state. Thus, there is no possibility of deadlocks due to the inter-neighbor coordination mechanism.

If router i is passive and has already set its distance to infinity ($D^i_j = \infty$), and receives an input event that implies an infinite distance to j, then router i simply updates D^i_{jk} and d_{ik}, and sends a reply to router k with an infinite distance if the input event is a query from router k. This ensures that update messages will stop when a destination becomes unreachable.

Figure 5 illustrates the interneighbor coordination mechanism of LPA. The number adjacent to each link represents the weight of that link; U indicates updates, Q represents queries and R replies. The arrowhead from node x to node y indicates that node y is the successor of node x towards destination j; i.e., $s^x_j = y$. The label in the parenthesis assigned to node x indicates current distance (D^x_j) and the feasible distance from x to destination j (FD^x_j). Active nodes are indicated in black.

In the example (Figure 5), we assume that messages propagate across all links at the same speed, which is considered a *step*. Node processes all messages received in the previous step in zero time.

When link (a, j) fails, node a updates its distance table by setting the distances from d and b to j equal to ∞, because the paths to j reported by both b and d include a. After that, node a is unable to find a feasible successor to j, because $D^a_{jb} = D^a_{jd} = \infty > 1 = FD^a_j$. Accordingly, it sends a query to all its neighbors (Figure 5(b)).

When node d receives a's query, it updates its distance table as follows: it sets $D^d_{ja} = \infty$ because a reports $D^a_j = \infty$, and it sets $D^d_{jb} = D^d_{jc} = \infty$, because the paths to j reported by b and d include node a. Because d uses a to reach j, d must also update its routing table. After updating its distance table, the neighbor that offers the shortest distance to j is j itself. Furthermore, $D^d_{jj} = 0 < 2 = FD^d_j$, and d sends an update to all its neighbor with $D^d_j = 1 + 9 = 10$ and a reply to node a (Figure 5(c)).

When node b receives a's query (before receiving a new update from d), it must set $D^b_{ja} = D^b_{jd} = D^b_{jc} = \infty$, because all its neighbors have reported a path to j that include node a. Because b's own path to j includes node a, it must update its routing table. Node b sends a query to its neighbors because every distance to j through any neighbor is infinity (Figure 5(c)).

When a receives the replies from d and b, it makes node d its new successor and also sends a reply to b (Figure 5(d)). When c receives the update from d and the query from b, it makes e its successor, because $D^c_{je} = 1 < 4 = FD^c_j$ and e offers the shortest path to j among all of c's neighbors. Accordingly, c sends an update with its new distance of 10 and a reply to b's query.

Finally, when b receives all the replies to its queries, it sets d as its successor and sends updates accordingly (Figure 5(e)).

3.5 Ensuring Simple Paths

Procedure TRT ensures that any finite distance in the routing table corresponds to a simple path by allowing router i to select successors to destinations as only those neighbors that satisfy the following property:

Property 1: Router i sets $s^i_j = k$ at time t only if $D^i_{xk}(t) + d_{ik}(t) \leq D^i_{xp}(t) + d_{ip}(t)$ for every neighbor p other than k and for every node x in the path from i to j defined by the predecessors reported by neighbor k.

Let $P^i_{jk}(t)$ denote the path from k to j defined by the predecessors reported by neighbor k to router i and stored in router i's distance table at time t. Procedure TRT enforces Property 1 by traversing all or part of $P^i_{jk}(t)$ from j back to k using the predecessor information. This path traversal ends when either a predecessor x is reached for which $tag^i_x = correct$ or $error$, or neighbor k is reached. If $tag^i_x = error$, then tag^i_j is set to $error$ also; otherwise, neighbor k or a correct tag must be reached, in which case tag^i_j is set to $correct$. Lemma 2 shows that this traversal correctly enforces Property 1, without having to traverse an entire implicit path; as the simulation results presented in Section 6 show, this makes LPA considerably more efficient than other prior path finding algorithms [3, 10, 19].

3.6 Handling Topology Changes

When router i establishes a link with a neighbor k, it updates its link-cost table and assumes that router k has reported infinite distances to all destinations and has replied to any query for which router i is active; furthermore, if router k is a previously unknown destination, router i initializes the path information of router k and sends an up-

date to the new neighbor k for each destination for which it has a finite distance. When router i is passive and detects that link (i,k) has failed, it sets $d_{ik} = \infty$, $D^i_{jk} = \infty$ and p^i_{jk} = null; after that, router i carries out the same steps used for the reception of a link-cost change message in passive state. When router i is active and loses connectivity with a neighbor k, it resets the reply flag and resets the path information i.e., assumes that the neighbor k sent a reply reporting an infinite distance.

It follows from the above description that the order in which router i processes updates, queries and replies does not change with the establishment of new links or link failures. The addition or failure of a router is handled by its neighbors as if all the links connecting to that router were coming up or going down at the same time.

4 Loop Freedom in LPA

The successor graph for destination $j \in G$, denoted by $S_j(G)$, is a directed graph in which nodes are the same as the nodes of G and where directed links are determined by the successor entries in the nodal routing tables. Loop freedom of routing tables is guaranteed at all times in G if $S_j(G)$ is always a directed acyclic graph. If G is connected in steady state, when all routing tables are correct, $S_j(G)$ must be a directed tree whose links point to j.

It is clear that $S_j(G)$ would be loop free at every instant if a router sent a query reporting an infinite distance to its neighbors every time it needed to change successors, because no router would change $S_j(G)$ before blocking any potential loop by sending an infinite distance "upstream" the loop. However, it is not obvious that loop freedom is maintained at every instant when routers use FC to decide if they have to send a query before changing $S_j(G)$. The following theorem shows that this is the case, i.e., that LPA is free of loops at every instant. The proof is by contradiction.

Proposition 1: If a loop is formed in the successor graph $S_j(G)$ for the first time at time t, then some router i in that loop must choose an upstream router as its successor at time t.

By assumption, $S_j(G)$ is a directed acyclic graph before the loop is formed at time t. If a loop has to be formed at time t, there must be at least one router $k \in S_j(G)$ that changes its successor because the successor information can be changed only when an update occurs or when the router detects a change in a link cost or status. This implies that an upstream router will be chosen by some router x in the loop. □

Theorem 1 *In a network G, the successor graph $S_j(G)$ is loop-free at every instant t.*

Proof: The proof is by contradiction to FC.

Let G be a stable topology and let the successor graph $S_j(G)$ be loop-free at every instant before t. Let $C_j(t)$ be the loop formed in the successor graph at time t. It is evident that no loops can be created unless routers change successors and modify the successor graph $S_j(G)$, and it follows from Proposition 1 that at least one router must change its successor at time t and choose an upstream neighbor for a loop to be formed.

At time $t = 0$, when the network is first initialized, each router knows only how to reach itself. This is equivalent to saying that at time 0, $S_j(G)$ is a disconnected graph of one or more components, each with a single router. Therefore $S_j(G)$ is loop-free at time $t = 0$.

Let $t > 0$, and assume that a loop $C_j(t)$ is formed when router i makes router a (=$s[1, new]$) its new successor (Figure 6). This implies the path from a to j at time t, denoted by $P_{aj}(t)$, includes $P_{ai}(t)$.

Let path $P_{ai}(t)$ consist of a chain of routers $\{a, s[2, new]$, ..., $i\}$, as shown in Figure 6. Router $s[k, new]$ is the kth hop in the path P_{ai} at time t and $s[k+1, new]$ is its successor at time t. Router $s[k, new]$ sets $s_j^{s[k,new]} = s[k+1, new]$ at time $t_{s[k+1,new]} \leq t$ and makes no more updates to its successor in the time interval $(t_{s[k+1,new]}, t]$; therefore,

$$\begin{aligned} s_j^{s[k,new]}(t_{s[k+1,new]}) &= s_j^{s[k,new]}(t) \\ D_j^{s[k,new]}(t_{s[k+1,new]}) &= D_j^{s[k,new]}(t) \end{aligned}$$

Similarly, router $s[k+1, old]$ is router $s[k, new]$'s successor just before node $s[k, new]$ becomes the kth hop of path $P_{ai}(t)$ by making router $s[k+1, new]$ its successor at time $t_{s[k+1,new]} \leq t$.

Because all the routers in $C_j(t)$ must have a successor at time t, all of them must be passive at that time. If all the routers in $C_j(t)$ have always been passive before time t, it follows from Theorem 1 in [5] that router i cannot create $C_j(t)$; the proof of that theorem is based on the fact that FD^i_j can only decrease as long as router i is passive. The rest of the proof needs to show that $C_j(t)$ cannot be formed if at least one router in $P_{ai}(t)$ was temporarily active before time t.

Consider the case in which node $s[k, new] \in P_{aj}(t)$ is already passive before it updates its distance and successor to join $P_{aj}(t)$ at time $t_{s[k+1,\ new]} \leq t$. According to LPA, $D_j^{s[k,\ new]}(t_{s[k+1,\ new]}) = RD_j^{s[k,\ new]}(t_{s[k+1,\ new]})$; furthermore, according to FC it must be true that

$$\begin{aligned} D_{j\ s[k+1,\ new]}^{s[k,\ new]}(t_{s[k+1,\ new]}) &= D_{j\ s[k+1,\ new]}^{s[k,\ new]}(t) \\ &< FD_j^{s[k,\ new]}(t) \\ &\leq D_j^{s[k,\ new]}(t_{s[k+1,\ old]}) \end{aligned}$$

Hence, if router $s[k-1,\ new]$ processed the update that node $s[k, new]$ sent at time $t_{s[k+1,\ new]}$, then

$$\begin{aligned} D_{j\ s[k,\ new]}^{s[k-1,\ new]}(t) &= D_j^{s[k,\ new]}(t) \\ &= D_{j\ s[k+1,\ new]}^{s[k,new]}(t) + d_{s[k,new]s[k+1,\ new]}(t) \\ &> D_{j\ s[k+1,\ new]}^{s[k,new]}(t) \end{aligned}$$

However, if $s[k-1,\ new]$ did not process the update that node $s[k,\ new]$ sent at time $t_{s[k+1,\ new]}$, then

$$\begin{aligned} D_{j\ s[k,\ new]}^{s[k-1,\ new]}(t) &= D_j^{s[k,\ new]}(t_{s[k+1,\ old]}) \\ &> D_{j\ s[k+1,\ new]}^{s[k,\ new]}(t) \end{aligned}$$

because router $s[k+1,\ new]$ must be a feasible successor for router $s[k, new]$ to make it its successor at time $t_{s[k+1,\ new]}$. Therefore, if router $s[k,\ new]$ is already passive when it changes successor at time $t_{s[k+1,\ new]}$, then $D_{j\ s[k,\ new]}^{s[k-1,\ new]}(t) > D_{j\ s[k+1,\ new]}^{s[k,\ new]}(t)$.

Alternatively, consider the case in which router $s[k,\ new]$ is active from time $t_k < t$ to time $t_{s[k+1,\ new]}$ when it becomes passive again to join $P_{aj}(t)$. In this case, regardless of the value of $D_j^{s[k,\ new]}(t_k)$, router $s[k,\ new]$ must have sent a query to its neighbors with $RD_j^{s[k,\ new]}(t_k) = \infty$ at time t_k, and all of those neighbors must acknowledge that value of $RD_j^{s[k,\ new]}(t_k)$ before router $s[k,\ new]$ can make any changes to its distance at time $t_{s[k+1,\ new]}$.

When router $s[k-1,\ new]$ makes router $s[k,\ new]$ its successor when it joins $P_{aj}(t)$ at time $t_{s[k,\ new]} \leq t$, it may or may not have processed any update or query sent by node $s[k,\ new]$ at time $t_{s[k+1,\ new]} \leq t$ when that node joins $P_{aj}(t)$. In the first case,

$$\begin{aligned} D_{j\ s[k,\ new]}^{s[k-1,\ new]}(t) &= RD_j^{s[k,\ new]}(t_{s[k+1,\ new]}) \\ &= D_j^{s[k,\ new]}(t_{s[k+1,\ new]}) \\ &= D_j^{s[k,\ new]}(t) \\ &\geq FD_j^{s[k,\ new]}(t) \\ &> D_{j\ s[k+1,\ new]}^{s[k,\ new]}(t) \end{aligned}$$

In the second case, $D_{j\ s[k,\ new]}^{s[k-1,\ new]}(t) = RD_j^{s[k,\ new]}(t_k)$; this is impossible, because $RD_j^{s[k,\ new]}(t_k) = \infty$ and node $s[k-1,\ new]$ could not have chosen a neighbor reporting an infinite distance as its successor.

From the above argument it follows that, if a router $s[k, new]$ is passive at time t, then

$$D_{j\,s[k,new]}^{s[k-1,new]} > D_{j\,s[k+1,new]}^{s[k,new]}(t)$$

However, because all the routers in the loop $C_j(t)$ are passive at time t, traversing path $P_{ai}(t)$ leads to the erroneous conclusion $D_{ja}^i(t) > D_{ja}^i(t)$. This implies that a loop cannot be formed when $S_j(G)$ is loop free before time t and G has a stable topology. On the other hand, the handling of queries and replies in LPA is not modified with the establishment or failure of links, and G is loop-free when it is first stored before any router has a finite distance to any other node. Therefore, the theorem is true. □

5 Correctness of LPA

To prove that LPA converges to correct routing-table values in a finite time, we assume that there is a finite time T_c after which no more link-cost or topology changes occur. Procedure DT makes node i update its distance table in a way that the routing information assumed from all neighbors is consistent, as discussed in section 3.2. However, LPA is correct even if such consistency of neighbor information is not enforced, and is not assumed in the following proof.

Lemma 1 *LPA is live.*

Proof: Consider the case in which the network has a stable topology. When a router is in the active state and receives a query from a neighbor, the router replies to the query with an infinite distance. The router updates its distance table entries when either an update or a reply message is received in active state. On the other hand, when a router in passive state receives a query from its neighbor, it computes the feasible distance and updates its distance and routing tables accordingly. If the router finds a feasible successor, it replies to its neighbor's query with its current distance to the destination. If the router can not find feasible successor, it forwards the query to the rest of its neighbors and sends a reply with an infinite distance to the neighbor who originated the query. Accordingly, in a stable topology, a router that receives a query from a neighbor for any destination must answer with a reply within a finite time, which means that any router that sends a query in a stable topology must become passive after a finite time.

Consider now the case in which the network topology changes. When a link fails or is reestablished, an active router that detects the link status change simply assumes that the router at the other end of the link has reported an infinite distance and has replied to the ongoing query. Because an active router must detect the failure or establishment of a link within a finite time, and because router failures or additions are treated as multiple link failures or additions, it follows from the previous case that no router can be active for an indefinite period of time, and the lemma is true. □

Lemma 2 *TRT correctly enforces Property 1.*

Proof: TRT correctly enforces Property 1 if the tag value given by TRT at router i for destination j equals correct. This is true only when the neighbor n that router i chooses as successor to j offers the smallest distance from i to each node in its reported implied path from n to j.

First note that, procedure DT is executed before TRT and ensures that router i sets $D_{jb}^i = \infty$ if its neighbor b reports a path to b that includes i. Therefore, TRT deals with simple paths only.

According to procedure TRT, there are two cases in which a router stops tracing the routing table: (1) the trace reaches node i itself (i.e., $p_{xns}^i = i$), and (2) a node on the path to j is found with tag_x^i = correct. We prove that the correct path information is reached in both cases.

Case 1: Assume that TRT is executed for destination j after an input event. The tag for each destination affected by the input event is set to null before procedure TRT is executed. Therefore, if TRT is executed for destination j and node i (the source) is reached, the tag of each node in the path from i to j through neighbor n must be null. Therefore, the distance from i to j through n is the shortest path among all neighbors, because node i chooses the minimum in row entry among its neighbors for a given destination j, and the lemma is true for this case.

Case 2: If node x_1 with $tag^i_{x_1}$ = correct is reached, then it must be true that either node i or a node x_2 with $tag^i_{x_2}$ = correct is reached from x_1.

If node i is reached from x_1, then it follows from Case 1 that neighbor n offers the smallest distance among all of i's neighbors to each node in the implied subpath from n to x_1 reported by neighbor n. Furthermore, because x_1 is reached from j, node n must also offer the smallest distance among all of i's neighbors to each node in the implied subpath from x_1 to j reported by n. Therefore, it follows from Case 1 that the lemma is true. Otherwise, if x_2 is reached, the argument used when i is reached from x_1 can be applied to x_2. Because router i always sets tag^i_i = correct and TRT deals with simple paths only, this argument can be applied recursively only for a maximum of $h < \infty$ times until i is reached, where h is the number of hops in the implicit path from n to j reported by n to i. Therefore, Case 2 must reduce to Case 1 and it follows that the lemma is true. □

Lemma 3 *The change in the cost or status of a link is reflected in the distance and the routing tables of a router adjacent to the link within a finite time.*

Proof: Regardless of the state in which router i is for a given destination j, it updates its link-cost and distance table within a finite time after it is notified of an adjacent link changing its cost, failing, or starting up. On the other hand, router i is allowed to update its routing table for destination j only when it is in passive state for that destination. However, because LPA is live (Lemma 1), if router i is active for destination j, it must receive all the replies to its query regarding j within a finite time, i.e., when it becomes passive. When router i becomes passive for destination j, it executes Procedure TRT, which updates the routing-table entry for destination j using the most recent information in router i's distance table. This implies that any change in a link is reflected in the distance and routing tables of a neighbor router within a finite time T. □

Given Lemma 3 and our assumption about time T_c, a finite time must exist when all routers adjacent to the links that changed cost or status have updated their link cost and status information, and after which no more link-cost or topology changes occur. Let T denote that time, where $T_c \leq T < \infty$.

Theorem 2 *After a finite time $t \geq T$, the routing tables of all routers must define the final shortest path to each destination.*

Proof: Let $T(H)$ be the time at which all messages sent by routers with shortest paths having $H-1$ hops ($H \geq 1$) to a given destination j have been processed by their neighbors.

Assume that destination j is reachable from every router.

For any router a adjacent to j, it follows from Lemma 3 that, if router a's shortest path to j is the link (a, j), then router a must update $D^a_j = d_{aj}$ by time $T = T(0)$ and the theorem is true for $H = 0$.

Because LPA is loop free at every instant (Theorem 1), the number of hops in any shortest path (as implied by the successor graph) is finite. Accordingly, the proof can proceed by induction on H.

Assume that the theorem is true for some $H > 0$. According to this inductive assumption, by time $T(H)$, router i must have a correct routing-table entry for every destination for which it has a shortest path of H hops or less. Property 1 must be satisfied for all such destinations and LPA enforces it correctly (Lemma 2). On the other hand, from the definition of $T(H+1)$, it follows that any update messages sent by routers with shortest paths of H hops or less to j or any other destination have been processed by their neighbors by time $T(H+1)$. Therefore, if router i's shortest path to destination j has $H+1$ hops, Property 1 must be satisfied at router i for that destination by time $T(H+1)$, because all possible predecessors for destination j must satisfy Property 1 at router i and that router must have the correct information for link (i, s^i_j) at time $T(0) < T(H+1)$ (Lemma 2). It follows that the theorem is true for the case of a connected network.

Consider the case in which j is not accessible to a connected component C of the network. Assume that there is a router $i \in C$ such that $D^i_j < \infty$ at some arbitrarily long time. If that is the case, j must satisfy Property 1 through at least one of router i's neighbors at that time; the same applies to such a neighbor, and to all the routers in at least one path from i to j defined by the routing tables of routers in C. This is not possible, because C is finite and LPA is always free of loops and live, which implies that, after a finite time $t_f \geq T$, all paths to j defined by the successor entries in the routing tables of routers in C must lead to routers that have set their distance to j equal to ∞. Therefore, because C is finite, LPA is live, and messages take a finite time to be transmitted, it follows that destination j will fail to satisfy Property 1 at each router within a finite time $t \geq t_f$, and routers must then set their distances to infinity, and the theorem is true. □

Theorem 3 *A finite time after t, no new update messages are being transmitted or processed by routers in G, and all entries in distance and routing tables are correct.*

Proof: After time T, the only way in which a router can send an update message is after processing an update message from a neighbor. Accordingly, the proof needs to consider three cases, namely: router i receives an update, a query, or a reply from a neighbor.

Consider an arbitrary router $i \in G$. Because LPA is live (Theorem 1) and router i obtains its shortest distance and corresponding path information for destination j in a finite time after T (Theorem 2), router i must be passive within a finite time $t_i \geq T$.

If router i receives an update for destination j from router k after time t_i, router i must execute Procedure Update. If router i has no path to destination j, D^i_j must be infinity and router k must report an infinite distance as well, because router i achieves its final shortest-path at time t_i; in this case, router i simply updates its distance table. On the other hand, if router i has a path to destination j, then $D^i_j < \infty$ and router i must find that FC is

satisfied and execute Procedure TRT. Because an update entry is added only when the shortest distance or predecessor to j change, router i can send no update or query of its own.

If router i receives a query from a neighbor for destination j after time t_i, it must execute Procedure Query. If router i has no physical path to destination j, D_j^i must be infinity and router k must report an infinite distance in its query, because router i achieves its final shortest-path at time t_i; in this case, router i simply updates its distance table and sends a reply to router k with an infinite distance. On the other hand, if router i has a physical path to destination j, it must determine that FC is satisfied when it processes router k's query. Accordingly, it simply sends a reply to its neighbor with its current distance and predecessor to router j. Therefore, router i cannot send an update or query of its own when it processes a query from a neighbor after time t_i.

After time t_i, router i cannot receive a reply from a neighbor, unless it first sends a query after time t_i, which is impossible according to the above two paragraphs.

It follows from the above that, for any given destination, no router in G can generate a new update or query after it reaches its final shortest path and predecessor to that destination. Because every router must obtain its final shortest distance and predecessor to every destination within a finite time (Theorem 2), the theorem is true. □

6 Performance of LPA

6.1 Complexity

This section compares LPA's worst-case performance with respect to the performance of DBF, DUAL, and ILS. This comparison is made in terms of the overhead required to obtain correct routing-table entries assuming that the algorithm behaves synchronously, so that every router in the network executes a step of the algorithm simultaneously at fixed points in time. At each step, the router receives and processes all the inputs originated during the preceding step and if required, sends update messages to the neighboring routers at the same step. The first step occurs when at least one router detects a topological change and issues update messages to its neighbors. During the last step, at least one router receives and processes messages from its neighbors and after which the router stops transmitting any update messages till a new topological change has taken place. The number of steps taken for this process is called the *time complexity* (TC); the number of messages required to accomplish this is called the *communication complexity* (CC).

DBF has a worst-case time complexity of $O(|N|)$ and worst-case communication complexity of $O(|N^2|)$, where $|N|$ is the number of routers in the network G [5]. ILS requires that each change in the cost or status of a link be communicated to all the routers in the network; accordingly, it has $TC = O(d)$ (where d is the network diameter), because a link-state update must traverse the whole network, and $CC = O(E)$, because each update traverses each link at most once in ILS but each link has two states, one in each direction of the link. On the other hand, DUAL has $TC = O(x)$ and $CC = O(x)$, where x is the number of routers affected by the single topology change [5].

In [21], it is shown that the time complexity of an algorithm like LPA is $O(x)$ in the worst-case, where x is the number of routers affected in the change.

6.2 Average Performance

We compare LPA's performance by simulation with the performance of ILS, DUAL and a path-finding algorithm similar to LPA that we simply call the basic path finding algorithm (BPFA) [19]. The key differences between LPA and BPFA are that BPFA has no interneighbor coordination to block temporary loops, and each update causes a node to update its entire routing table, just like in other path-finding algorithms [3, 10].

A set of counters were used to instrument the simulations. These counters can be reset at various points. When the event queue empties, that is, when the algorithm converges, the values of these counters are printed. During each simulation step, a router processes input events received during the previous step one at a time, and generates messages as needed for each input event it processes. To obtain the average figures, the simulation makes each link (router) in the network fail, and counts the steps and messages needed for each algorithm to recover. It then makes the same link (router) recover and repeat the process. The average is then taken over all link (router) failures and recoveries. The routing algorithm was allowed to converge after each such change. The average is taken over all failures and recoveries. In all cases, routers were assumed to perform computations in zero time and links were assumed to provide one time unit of delay. For the failure and recovery runs, the costs were set to unity. Both the mean and the standard deviation were computed for each counter; the four counters used are

- *Events:* The total number of updates and changes in link status processed by routers.
- *Packets:* The total number of packets transmitted over the network. Each packet may contain multiple updates.
- *Duration:* The total elapsed time it takes for the algorithm to converge.
- *Operations:* The total number of operations performed by the algorithm. The operation count is incremented when an event occurs.

In this paper, we focus on simulation results for the ARPANET topology; similar simulation results for other network topologies appear elsewhere [18].

The simulation results for a single resource change are shown in Table 1. The table shows the average number of events (updates and link-status changes processed by routers), the average number of update messages, the average number of steps and the average number of operations

required by all the routers in the network for BPFA, LPA, DUAL and ILS to converge after a single topology change.

BPFA incurs fewer steps in the average than the rest of the algorithms after single failures. This is because Procedure DT of LPA, which is the basis of BPFA's operation, prevents the formation of temporary loops without the need for any internodal coordination. However, the results obtained for LPA after router or link failures are very encouraging. Because of the inter-neighbor synchronization scheme used in LPA, it can be expected that at least two additional steps are needed for convergence after a router failure, in addition to the steps required to propagate link-failure updates across the network. This is because a wave of queries must propagate from the routers detecting the failure of the links adjacent to the failed router, to the routers that are the farthest from the sources of updates.

The small difference between the number of steps required in LPA and BPFA indicates that LPA's inter-neighbor coordination mechanism achieves loop freedom at every instant with little overhead. Another important point of comparison between LPA and BPFA is the number of operations they require. In BPFA, the entire shortest-path tree defined in the routing table has to be traversed every time a router processes an input event. In contrast, in LPA, Procedure TRT traverses only those routing-table entries affected by the input event. The results clearly show that considerable efficiency is gained by the tagging mechanism used in LPA.

The graphs in Figures 7 and 8 depict the number of messages exchanged before LPA, DUAL and ILS converge for every link failing and recovering in the ARPANET topology after a single topology change. Similar graphs for each node failing and recovering are given in Figures 9 and 10, respectively. All topology changes were performed one at a time and the algorithms were allowed to converge after each such change before the next resource change occurs. The ordinates of the graphs represent the identifiers of the links and the nodes, while the data points show the number of messages exchanged after each resource change in each of these figures.

The simulation results show that LPA and DUAL have better overall average performance than ILS after the recovery of a single router or link. In the average, LPA requires fewer steps, messages, and CPU cycles than DUAL does after a single resource failure or addition. LPA also requires a comparable number of steps and update messages than ILS does after a single resource failure, but requires orders of magnitude fewer operations. Furthermore, the number of entries per update message in LPA is very small.

In the case of ARPANET, up to eight steps are needed to reach the farthest router, and two more steps are needed to handle the last query and reply after that. The simulation results show that approximately nine steps are needed in the average case for LPA's convergence after a router failure, compared to ILS's eight or nine steps.

In summary, the above results indicate that LPA constitutes a more scalable solution for routing in large internets than ILS and DUAL. Dynamics of LPA is discussed in [20].

6.3 Comparison with Prior Path-Finding Algorithms

LPA is the first path-finding algorithm that provides loop freedom at every instant. Reference [3] discusses loop freedom; however, the path finding algorithm presented in that work does not provide loop-free paths at every instant. The approach proposed in [3] relies on each router sending a query to its neighbors with the intended new routing-table entries, and waiting for the neighbors' replies before making the change. Data packets are held at a router waiting for its neighbors' replies. This approach incurs substantial communication overhead, because a router sends queries every time it tries to change its routing table, and also incurs unnecessary queueing delays for data packets.

Routing algorithms have been proposed in the past that provide loop-free paths at every instant by blocking potential loops. However, in these algorithms [7, 6], a router sends path information to its neighbors in update messages containing explicit labels of variable size that can contain the complete path in some cases. In contrast, LPA uses fixed-size entries in update messages, because path information is obtained from the predecessor entries.

LPA updates routing table entries using Procedure TRT, which ensures that only simple paths are used. This mechanism is similar to those proposed in [3, 17]; however, Procedure DT in LPA makes a router check the consistency of predecessor information reported by *all* its neighbors each time an input event is processed. In contrast, earlier path finding algorithms [3, 17, 10] check the consistency of the predecessor information only for the neighbor associated with the input event.

LPA is more scalable than the algorithms in [3, 10], because LPA updates only those entries of the distance and routing tables that are affected by the input event, rather than the entire tables, using tags similar to those used in [17]. In contrast, the algorithm in [10] uses a breath-first search on the entire distance table each time a router processes an input event; the algorithm reported in [3] makes sure that Property 1 is satisfied by all destinations every time an input event is processed, much like BPFA does.

7 Conclusions

We have presented and verified the first routing algorithm (LPA) that eliminates the formation of temporary routing table loops without internodal synchronization spanning multiple hops or the communication of complete or variable-length path information. LPA is based on the notion of using information about the second to last hop (predecessor) of shortest paths to ensure termination, and an efficient inter-neighbor coordination mechanism to eliminate temporary loops. The worst-case complexity of LPA for single recovery or failure is $O(x)$, x being the number

of routers affected by this recovery or failure. The performance comparison of LPA and BPFA confirms that LPA achieves loop-freedom with very limited additional overhead compared to similar path-finding algorithms. Our simulation results show that taking the average number of steps, messages, and operations into account, LPA is more efficient than DUAL and ILS. LPA works correctly when routing information is aggregated (e.g., masks are used to reduce routing-table size) [22] and paves the way towards developing an efficient routing protocol based on LPA for very large internetworks that is as simple as RIPv2 [13] and much more efficient than OSPF [16].

References

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

[2] R. Albrightson, J.J. Garcia-Luna-Aceves, and J. Boyle, "EIGRP–A Fast Routing Protocol Based on Distance Vectors," *Proc. Networld/Interop 94*, May 1994.

[3] C. Cheng, R. Riley, S. P. R Kumar and J. J. Garcia-Luna-Aceves, "A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect", *ACM Computer Comm. Review*, Vol.19, No.4, 1989, pp. 224–236.

[4] J. J. Garcia-Luna-Aceves, "A Fail-Safe Routing Algorithm for Multihop Packet Radio Networks", *Proc. IEEE INFOCOM 86*, 1986.

[5] —, "Loop-free Routing Using Diffusing Computations", *IEEE/ACM Transactions on Networking*, Vol. 1, No. 1, Feb, 1993, pp. 130–141.

[6] —, "Distributed Routing with Labeled Distances", *Proc. IEEE INFOCOM 92*, Vol.2, May 1992, pp. 633–643.

[7] —, "LIBRA: A Distributed Routing Algorithm for Large Internets", *Proc. IEEE Globecom 92*, Vol.3, Dec 1992, pp. 1465–1471.

[8] J. Hagouel, "Issues in Routing for Large and Dynamic Networks," IBM Research Report RC 9942 (No. 44055) Communications, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, April 1983.

[9] C. Hedrick, "Routing Information Protocol," RFC 1058, June 1988.

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

[11] J.M. Jaffe and F.M. Moss, "A Responsive Routing Algorithm for Computer Networks", *IEEE Trans. Comm.*, Vol.30, July 1982, pp. 1758–1762.

[12] Y. Rekhter, T. Li, "A Border Gateway Protocol 4 (BGP-4)," *Network Working Group Internet Draft*, Jan. 1994.

[13] G. Malkin, "RIP Version 2 – Carrying Additional Information", RFC-1723, Nov. 1994.

[14] J. McQuillan and D.C. Walden, "The ARPANET Design Decisions", *Computer Networks*, Vol.1, August 1977.

[15] P.M. Merlin and A. Segall, "A Failsafe Distributed Routing Algorithm", *IEEE Trans. Comm.*, Vol.27, Sept. 1979, pp. 1280–1288.

[16] J. Moy, "OSPF Version 2," RFC 1247, August 1991.

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

[18] S. Murthy, "Design and Analysis of Distributed Routing Algorithms", *Master's Thesis*, University of California, Santa Cruz, 1994.

[19] S. Murthy and J.J. Garcia-Luna-Aceves, "A More Efficient Path-Finding Algorithm", *28th Asilomar Conference*, Nov. 1994, pp. 229–233.

[20] —, "Dynamics of a Loop-Free Path-Finding Algorithm", *Proc. IEEE Globecom'95*, Nov. 1995, pp. 1347–1351.

[21] — "A Routing Protocol for Wireless Networks", *ACM Mobile Networks and Applications (MONET)*, Vol. 1, No. 2, 1996.

[22] — "Loop-Free Internet Routing Using Hierarchical Routing Trees", IEEE INFOCOM'97, 1997.

J.J. Garcia-Luna-Aceves (M) was born in Mexico City, Mexico on October 20, 1955. He received the B.S. degree in electrical engineering from the Universidad Iberoamericana, Mexico City, Mexico, in 1977, and the M.S. and Ph.D. degrees in electrical engineering from the University of Hawaii, Honolulu, HI, in 1980 and 1983, respectively.

He is an Associate Professor of Computer Engineering at the University of California, Santa Cruz (UCSC). Prior to joining UCSC in 1993, he was a Center Director at SRI International (SRI) in Menlo Park, California. He first joined SRI as an SRI International Fellow in 1982. His current research interest is the analysis and design of algorithms and protocols for computer communication. At UCSC, he leads a number of research projects sponsored by DARPA and ONR that focus on wireless networks and internetworking.

Dr. Garcia-Luna-Aceves has published more than 100 technical papers on computer communication, has been guest editor for IEEE COMPUTER (1985), and is an editor of the ACM Multimedia Systems Journal and the Journal of High Speed Networks. He has been Chair of the ACM special interest group on multimedia, General Chair of the first ACM conference on multimedia: ACM MULTIMEDIA '93, Program Chair of the IEEE MULTIMEDIA '92 Workshop, General Chair of the ACM SIGCOMM '88 Symposium, and Program Chair of the ACM SIGCOMM '87 Workshop and the ACM SIGCOMM '86 Symposium. He has also been program committee member for numerous IFIP 6.5, ACM, and IEEE conferences on computer communication. He received the SRI International Exceptional-Achievement Award in 1985 for his work on multimedia communications, and again in 1989 for his work on adaptive routing algorithms. He is a member of the ACM. (Email: jj@cse.ucsc.edu, Web Page: http://www.cse.ucsc.edu/research/ccrg/)

Shree Murthy (M '97 / ACM '97) received the B.E. in computer science from Bangalore University in 1990. She received her M.S. in computer science and engineering from the Indian Institute of Technology, Madras in 1993, and the M.S. and PhD degrees in computer engineering from the University of California, Santa Cruz in 1994 and 1996, respectively.

She is a Member of Technical Staff at the Internetworking Products Group of Sun Microsystems, Inc. where she is involved in the design and development of advanced switching products. Prior to joining Sun Microsystems, Inc. in 1996, she was with Sun Labs, where she worked on routing protocols for ATM networks. She received the NSF/ARPA Fellowship for Sigcomm'94 and IEEE grant for Globecom'95. Her research interests include routing and congestion control algorithms and protocols for both wired and wireless/mobile networks. (Email: shree.murthy@eng.sun.com)

```
Procedure Init1
when router i initializes itself
do begin
    set a link-state table with
    costs of adjacent links;
    N ← {i}; N_i ← {x | d_ix < ∞};
    for each (x ∈ N_i)
    do begin
        N ← N ∪ x; tag_x^i ← null;
        s_x^i ← null; p_x^i ← null;
        D_x^i ← ∞; FD_x^i ← ∞
    end
    s_i^i ← i; p_i^i ← i; tag_i^i ← correct;
    D_i^i ← 0; FD_i^i ← 0;
    for each (n ∈ N_i) do
        for each j ∈ N call Init2(n, j);
    for each (n ∈ N_i) do
    add (0, i, 0, i) to LIST_i(n);
    call Send
end

Procedure Init2(x, j)
begin
    D_jx^i ← ∞; p_jx^i ← null;
    r_jx^i ← 0;
end

Procedure Send
begin
    for each (n ∈ N_i)
    do begin
        if (LIST_i(n) is not empty)
        then send message with
          LIST_i(n) to n
        empty LIST_i(n)
    end
end

Procedure Reply(j, k)
begin
    r_jk^i ← 0;
    if (r_jn^i = 0, ∀n ∈ N_i)
    then if ((∃x ∈ N_i | D_jx^i < ∞)
       or (D_j^i < ∞))
        then call PU(j)
    else call AU(j, k)
end

Procedure Message
when router i receives a message
  on link (i, k)
begin
    for each entry (u_j^k, j, RD_j^k, rp_j^k)
      such that j ≠ i
    do begin
        if (j ∉ N)
        then begin
            if (RD_j^k = ∞)
            then delete entry
            else begin
                N ← N ∪ {j}; FD_j^i = ∞;
                for each x ∈ N_i
                  call Init2(x, j)
                tag_j^i ← null;
                call DT(j, k)
            end
        end
        else
            tag_j^i ← null; call DT(j, k)
    end
    for each entry (u_j^k, j, RD_j^k, rp_j^k) left
    such that j ≠ i
    do case of value of u_j^i
        0: [Entry is an update]
            call Update(j, k)
        1: [Entry is a query]
            call Query(j, k)
        2: [Entry is a reply]
            call Reply(j, k)
    end
    if (i = j)
    begin
        do case of value of u_j^i
            1: add (2, j, 0, null) to LIST_i(k)
        end
    end
    call Send
end

Procedure Update(j, k)
begin
    if (r_jx^i = 0, ∀x ∈ N_i)
    then begin
            if ((s_j^i = k) or (D_jk^i + d_ik < D_j^i))
            then call PU(j)
    end
    else call AU(j, k)
end

Procedure PU(j)
begin
    DT_min ← Min{D_jx^i + d_ix ∀ x ∈ N_i};
    FCSET ← {n | n ∈ N_i, D_jn^i + d_in = DT_min,
      D_jn^i < FD_j^i};
    if (FCSET ≠ ∅) then begin
      call TRT(j, DT_min);
      FD_j^i ← Min{D_j^i, FD_j^i}
    end
    else begin
        FD_j^i = ∞; r_jx^i = 1,  ∀x ∈ N_i;
        D_j^i = D_{j s_j^i}^i + d_{i s_j^i};
        p_j^i = p_{j s_j^i}^i;
        if (D_j^i = ∞) then s_j^i ← null;
        ∀ x ∈ N_i do begin
          if (query and x = k)
          then r_jk^i ← 0;
          else add (1, j, ∞, null)
            to LIST_i(x)
        end
    end
end

Procedure Query(j, k)
begin
    if (r_jx^i = 0∀x ∈ N_i)
    then begin
        if (D_j^i = ∞ and D_jk^i = ∞)
        then add (2, j, D_j^i, p_j^i)
          to LIST_i(k)
        else begin
            call PU(j);
            add (2, j, D_j^i, p_j^i)
              to LIST_i(k);
        end
    else
    begin
        add (2, j, ∞, null) to LIST_i(k)
        call AU(j, k)
    end
end
```

Figure 1: LPA Specification

```
Procedure TRT(j, DT_min)
begin
    if (D^i_j s^i_j + d^i_{s^i_j} = DT_min)
    then ns ← s^i_j;
    else ns ← b | {b ∈ N_i and D^i_jb + d_ib = DT_min};
    x ← j;
    if (p^i_xns = ns) tag^i_x = correct
    while (D^i_x ns + d_ins =
        Min{D^i_xb + d_ib  ∀  b ∈ N_i}
      and (D^i_xns < ∞) and (tag^i_x = null))
    do x ← p^i_x ns;
    if (p^i_x ns = i or tag^i_x = correct)
    then tag^i_j ← correct;
    else tag^i_j ← error;
    if (tag^i_j = correct)
    then begin
        if (D^i_j ≠ DT_min or p^i_j ≠ p^i_j ns)
        then add (0, j, DT_min, p^i_j ns)
            to LIST_i(x)  ∀x ∈ N_i;
        D^i_j ← DT_min; p^i_j ← p^i_j ns; s^i_j ← ns;
    end
    else begin
        if (D^i_j < ∞)
        then add (0, j, ∞, null)
            to LIST_i(x)  ∀x ∈ N_i;
        D^i_j ← ∞; p^i_j ← null;
        s^i_j ← null;
    end
end
Procedure Link_Up (i, k, d_ik)
when link (i, k) comes up do
begin
    d_ik ← cost of new link;
    if (k ∉ N) then begin
        N ← N ∪ {k}; tag^i_k ← null;
        D^i_k ← ∞; FD^i_k ← ∞;
        p^i_k ← null; s^i_k ← null;
        for each x ∈ N_i
            do call Init2(x, k)
    end
    N_i ← N_i ∪ {k};
    for each j ∈ N do
        call Init2(k, j);
    for each j ∈ N − k | D^i_j < ∞ do
        add (0, j, D^i_j, p^i_j) to LIST_i(k);
    call Send
end
```

```
Procedure Link_Down(i, k)
when link (i, k) fails do begin
    d_ik ← ∞;
    for each j ∈ N do begin
        call DT(j, k);
        if (k = s^i_j) then tag^i_j ← null
    end
    for k, D^i_jk ← ∞;
    p^i_jk ← null; N_i ← N_i − {k};
    delete r^i_jk;
    for each j ∈ (N − i) | k = s^i_j
        call Update(j, k)
    call Send
end
Procedure Link_Change (i, k, d_ik)
when d_ik changes value do begin
    old ← d_ik; d_ik ← new link cost;
    for each j ∈ N − i do begin
        call DT(j, k);
        if (D^i_j > D^i_jk + d_ik then begin
            tag^i_j ← null;
            call Update(j, k)
        end
        else begin
            if (k = s^i_j) call Update(j, k)
        end
    end
    call Send
end
Procedure AU(j, k)
begin
    if (k = s^i_j) then
        D^i_j ← D^i_jk + d_ik; p^i_j ← p^i_jk;
end
Procedure DT(j, k)
begin
    D^i_jk ← RD^k_j; p^i_jk ← rp^k_j;
    for each neighbor b do begin
        h ← j;
        if (p^i_hb ≠ null)
        do begin
            while (h ≠ i or k or b)
                do h ← p^i_hb;
            if (h = k) then
                D^i_jb ← D^i_kb + RD^k_j; p^i_jb ← rp^k_j;
        end
        if (h = i) then
            D^i_jb ← ∞; p^i_jb ← null;
    end
end
```

Figure 2: LPA Specification (cont...)

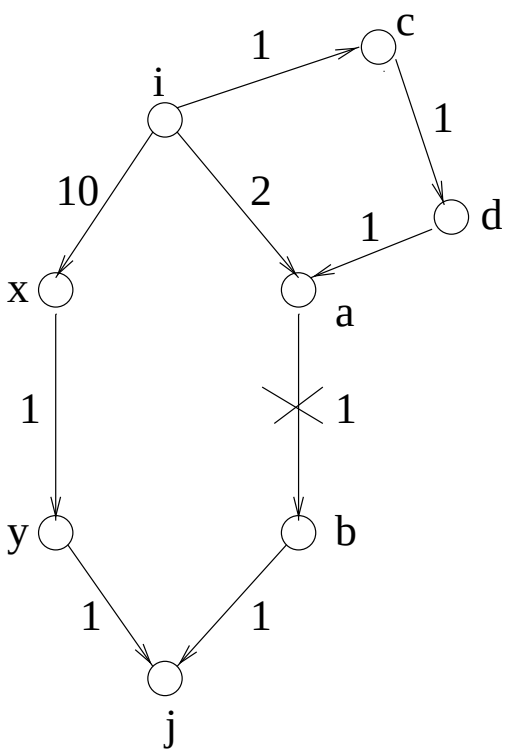

Figure 3: Updating Mechanism

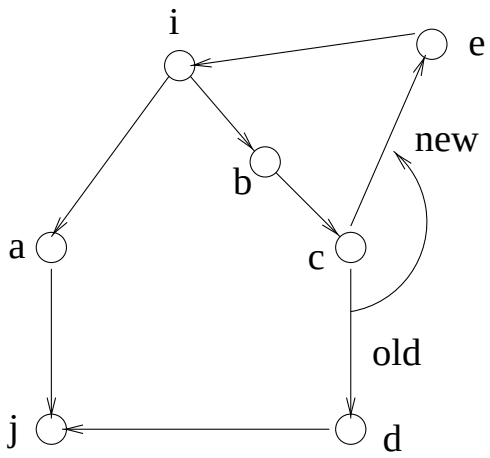

Figure 4: Possibility of a Loop

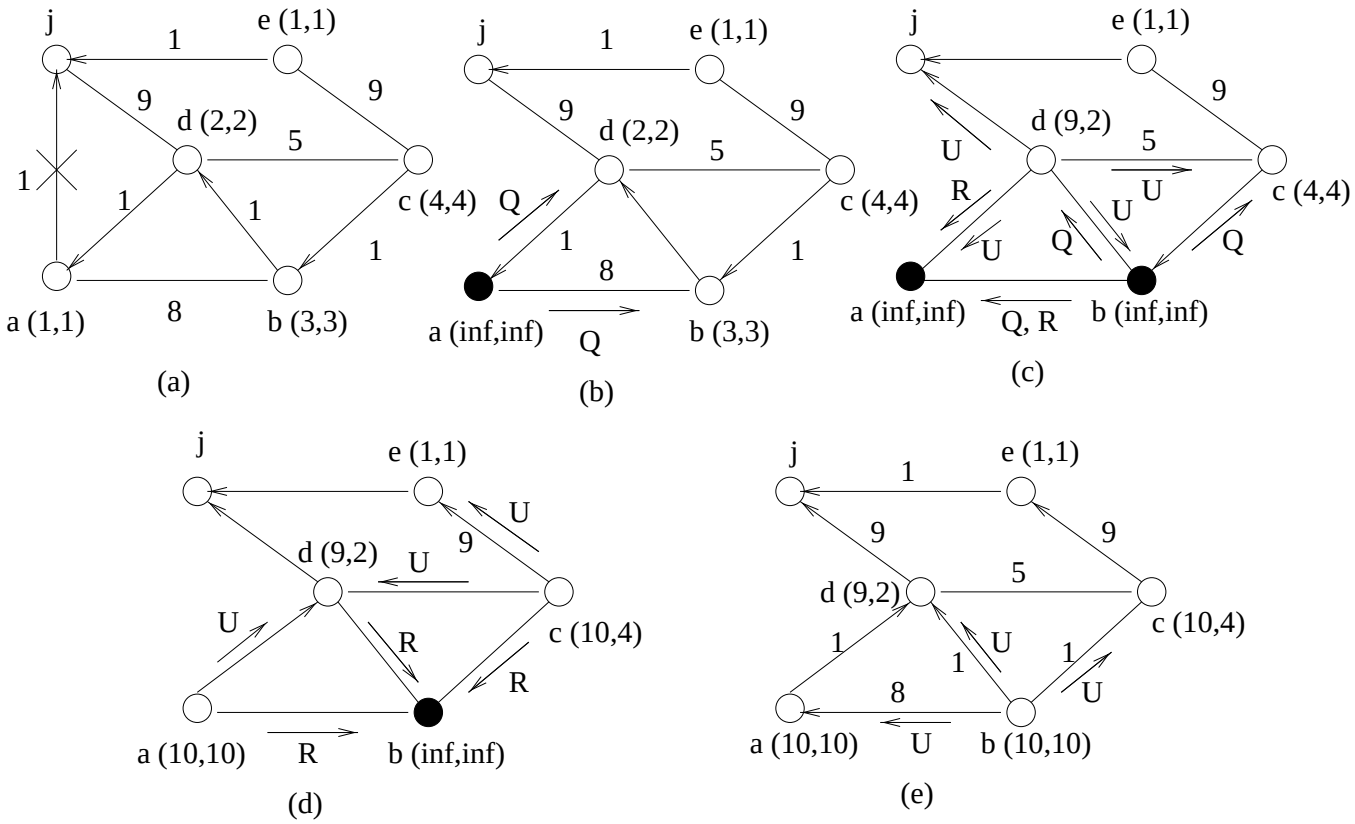

Figure 5: Example of LPA's Operation

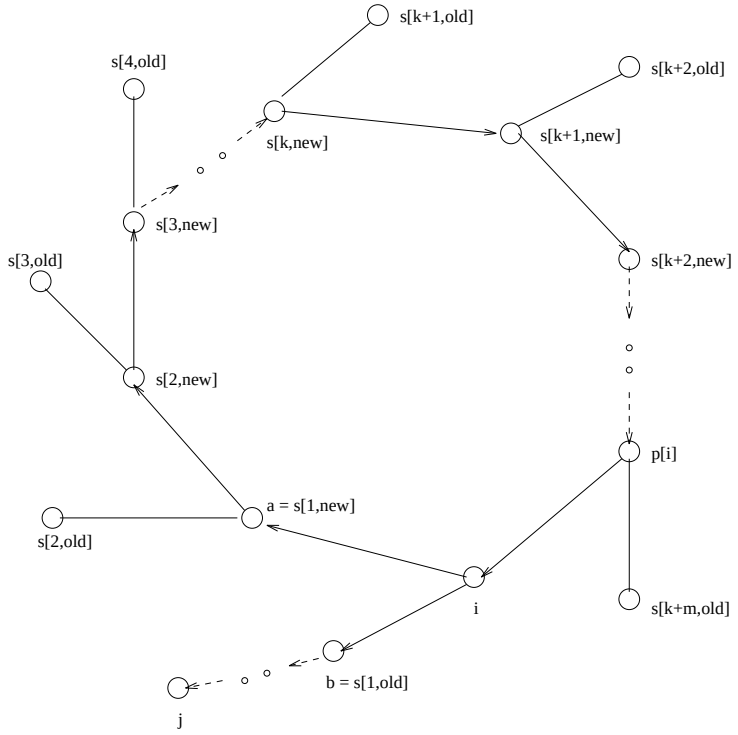

Figure 6: Loop in G

Table 1: Simulation Results for ARPANET

Parameter	BPFA		LPA		DUAL		ILS	
	mean	sdev	mean	sdev	mean	sdev	mean	sdev
Link Failure								
Event Count	962.1	392.9	587.3	381.5	720.9	449.1	160.0	0.0
Packet Count	96.5	45.9	126.1	59.8	266.8	97.3	158.0	0.0
Duration	7.16	1.75	9.24	3.39	15.1	3.45	8.5	0.74
Operation Count	843.90	594.5	385.6	190.8	813.9	449.1	25600.2	57. 121
Link Recovery								
Event Count	638.2	370.3	242.4	112.8	362.2	147.6	162.7	15.4
Packet Count	108.6	48.9	33.0	25.5	79.3	21.3	160.7	15.4
Duration	6.89	1.51	5.96	2.75	7.3	1.46	7.84	0.67
Operation Count	1144.9	620.1	213.2	56.4	454.2	147.6	26900.8	2477 .9
Router Failure								
Event Count	1350.8	373.8	646.5	424.4	1050.4	300.8	218.8	67.1
Packet Count	96.6	75.9	144.7	55.3	382.6	81.2	212.1	65.1
Duration	5.4	3.4	9.12	2.4	17.8	9.2	8.6	0.72
Operation Count	1803.8	407.4	589.5	271.3	1320.8	563.5	33356.7	10 766.2
Router Recovery								
Event Count	980.4	699.7	551.6	296.4	691.9	235.5	301.2	45.3
Packet Count	107.2	80.1	68.06	42.03	207.9	46.7	294.5	42.9
Duration	5.27	2.56	7.78	3.33	8.5	0.73	9.6	1.14
Operation Count	3252.0	1911.5	542.0	224.4	957.6	347.3	50102.2	79 30.4

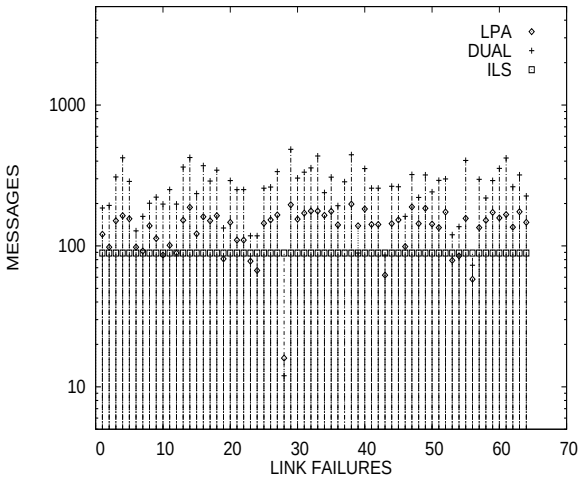

Figure 7: ARPANET Link Failure

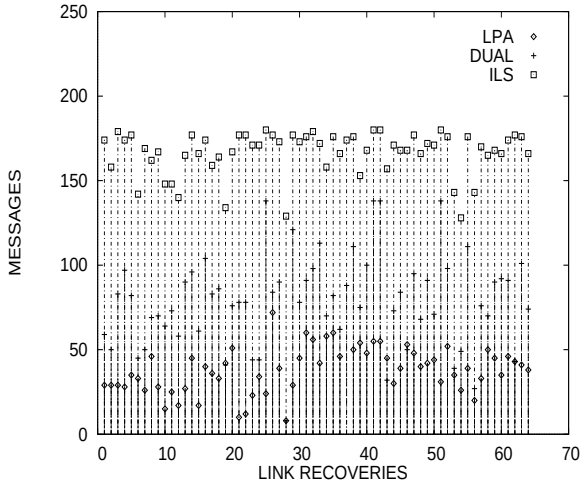

Figure 8: ARPANET Link Recovery

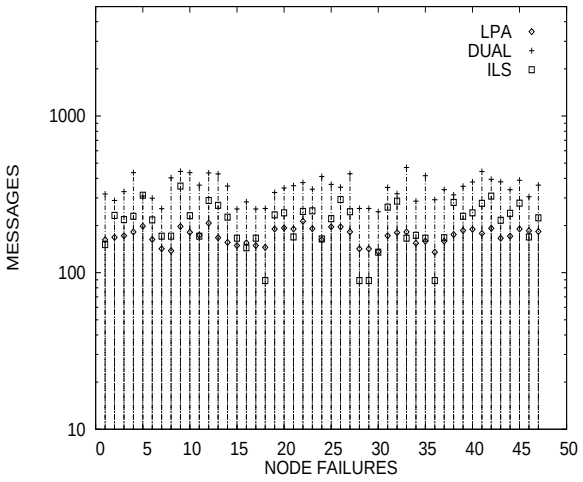

Figure 9: ARPANET Node Failure

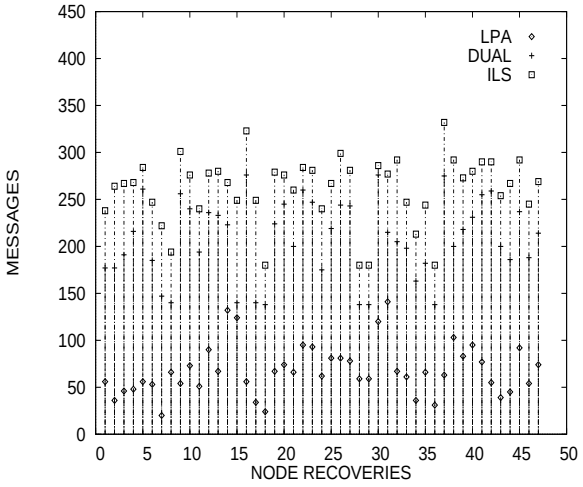

Figure 10: ARPANET Node Recovery