Loop-Free Routing Using Diffusing Computations

J. J. Garcia-Lunes-Aceves, *Member, IEEE*

Abstract—**A family of distributed algorithms for the dynamic computation of the shortest paths in a computer network or internet is presented, validated, and analyzed. According to these algorithms, each node maintains a vector with its distance to every other node. Update messages from a node are sent only to its neighbors; each such message contains a distance vector of one or more entries, and each entry specifies the length of the selected path to a network destination, as well as an indication of whether the entry constitutes an update, a query, or a reply to a previous query. The new algorithms treat the problem of distributed shortest-path routing as one of diffusing computations, which was first proposed by Dijkstra and Scholten. They improve on algorithms introduced previously by Chandy and Misra, Jaffe and Moss, Merlin and Segall, and the author. The new algorithms are shown to converge in finite time after an arbitrary sequence of link cost or topological changes, to be loop-free at every instant, and to outperform all other loop-free routing algorithms previously proposed from the standpoint of the combined temporal, message, and storage complexities.**

I. Introduction

THE routing protocols used in most of today's computer networks are based on shortest-path algorithms that can be classified as distance-vector or link-state algorithms. In a distance-vector algorithm, a node knows the length of the shortest path from each neighbor node to every network destination, and uses this information to compute the shortest path and next node in the path to each destination. A node sends update messages to its neighbors, who in turn process the messages and send messages of their own if needed. Each update message contains a vector of one or more entries, each of which specifies, as a minimum, the distance to a given destination. In contrast, in a link-state algorithm, also called topology-broadcast algorithm, a node must know the entire network topology, or at least receive that information, to compute the shortest path to each network destination. Each node broadcasts update messages, containing the state of each of the node's adjacent links, to every other node in the network.

Several routing protocols based on distance-vector algorithms, called distance-vector protocols or DVP's in this paper, have been proposed for and implemented in computer networks including the old ARPANET routing protocol [18] and the NETCHANGE protocol of the MERIT network [26]. Well-known examples of DVP's implemented in internetworks are the Routing Information Protocol (RIP) [10], the Gateway-to-Gateway Protocol (GGP) [11], and the Exterior Gateway Protocol (EGP) [20]. All of these DVP's have used variants of the distributed Bellman–Ford algorithm (DBF) for shortest path computation [4]. The primary disadvantages of this algorithm are routing-table loops and counting to infinity [13]. A routing-table loop is a path specified in the nodes' routing tables at a particular point in time, such that the path visits the same node more than once before reaching the intended destination. A node counts to infinity when it increments its distance to a destination until it reaches a predefined maximum distance value.

A number of attempts have been made to solve the counting-to-infinity and routing-table looping problems of distance-vector algorithms by increasing the amount of information exchanged among nodes, or by making nodes to hold down the updating of their routing tables for some period of time after detecting distance increases. However, none of those approaches solves these problems satisfactorily [6], [7]. A recent DVP developed for internetwork routing, called the Border Gateway Protocol (BGP) [16], specifies the entire path from source to destination in update messages, as proposed by Shin and Chen [25], to *detect* the occurrence of loops.

On the other hand, link-state algorithms are free of the counting-to-infinity problem. However, they need to maintain an up-to-date version of the entire network topology at every node, which may constitute excessive storage and communication overhead in a large, dynamic network [24]. It is also interesting to note that the routing protocols using link-state algorithms, called link-state protocols (LSP), which have been implemented to date do not eliminate the creation of temporary routing-table loops [7]. Well-known examples of LSP's are the new ARPANET routing protocol [19], the OSI intradomain routing protocol [12], and the Open Shortest Path First (OSPF) protocol [2].

Whether link states or distance vectors are used, the existence of routing-table loops, even temporarily, is a detriment to the overall performance of an internet. This paper unifies new and previous results on loop-free routing using distance vectors into a new family of distance-vector algorithms [9] that are always loop-free, operate with arbitrary transmission or processing delays, assume arbitrary positive link costs, and provide shortest paths within a finite time after the occurrence of an arbitrary sequence of link-cost or topological changes. The approach used in these algorithms treats the distributed shortest-path routing problem as one of *diffusing computations*

Manuscript received June 1991; revised January 1992; recommended for transfer from the IEEE TRANSACTIONS ON COMMUNICATIONS by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Jeffrey Jaffe. This work was supported by SRI IR&D funds, by the U.S. Army Research Office under contract DAAL03-88-K-0054, and by the USAF-AFSC Rome Air Development Center, and the Defense Advanced Research Projects Agency under Contracts F30602-85-C-0186 and F30602-89-C-0015. This paper was presented in part at the ACM SIGCOMM '89 Conference, Austin, TX, Sept. 19–22, 1989.

The author is with the Department of Computer Engineering, University of California, Santa Cruz, CA 95064, and with SRI International, Menlo Park, CA 94025. (email: joaquin@NISC.SRI.COM)

IEEE Log Number 9206160.

[3], and matches or improves on the performance of previous loop-free distance-vector algorithms [5], [13], [17], [23].

This paper refers to routing-table loops simply as *loops,* and refers to a routing algorithm that is free of routing-table loops as a *loop-free routing algorithm.* The following sections introduce sufficient conditions for loop freedom using arbitrary routing algorithms, explain the application of diffusing computations to routing, describe and verify the new family of algorithms, and compare their performance with the performance of other algorithms in terms of their complexity and average response to topological changes.

II. Network Model and Notation

A network is modeled as an undirected connected graph in which each link has two lengths or costs associated with it—one for each direction—and in which any link of the network exists in both directions at any one time. A link-level protocol assures that:

- Every node knows its neighbors, which implies that a node detects within a finite time 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 messages, changes in the cost of a link, link failures, and new-neighbor notifications are processed one at a time within a finite time and in the order in which they occur.

Each node has a unique identifier, and link costs can vary in time but are always positive. The distance between two nodes is measured as the sum of the link costs in the path of least cost or *shortest path* between them. This same model can be applied to an internet in which routers are the nodes of the graph and networks are the edges of the graph [20]. For this case, the three services listed above are provided by a datagram service at the network level and a transport-level protocol similar to the transmission control protocol (TCP) [21].

Throughout this paper, the following notation is used:

G: a connected network of arbitrary topology
E: The set of links in G
N: The set of nodes in G
j: The identifier of destination node $j \in N$
(i,x): The link in E between nodes i and x
$s_j^i(t)$: The successor (or next hop) in the path to node j currently chosen by node i at time t
$l_k^i(t)$: The cost of the link from node i to neighbor node k, as known by node i at time t; the cost of a nonexistent link or a failed link is considered to be infinity
$V^i(t)$: The set of destination nodes node i knows at time t
$N_i(t)$: The set of nodes connected through a link with node i at time t—more formally, $N_i(t) = \{x | \exists (i,x), l_x^i(t) < \infty\}$; a node in that set is said to be a neighbor of node i.
$D_j^i(t)$: The current distance from node i to node j as known by node i at time t
$D_{jk}^i(t)$: The distance from node k to node j as known by node i at time t
$D_j^{*i}(t)$: The smallest value assigned to D_j^i up to time t
$D_{jk}^{*i}(t)$: The smallest value of D_j^k known by node i up to time t
$RD_j^i(t)$: The distance from node i to node j that node i can report to its neighbors at time t (and which need not equal $D_j^i(t)$)
$FD_j^i(t)$: The distance value used by node i to evaluate whether a feasibility condition is satisfied at time t; depending on the condition used, it can be equal to either $D_j^{*i}(t)$ or $D_{jk}^{*i}(t)$ (see Section III)
$P_{xj}(t)$: The path from node x to node j implied by the $s_j^i(t)$ entries for all $i \in G$ at time t.

The time at which the value of a variable applies is specified only when it is necessary.

III. Sufficient Conditions for Loop Freedom

Assume that an arbitrary link-state or distance-vector algorithm is used in G, such that a node updates its routing table independently of other nodes upon reception of messages or detection of changes in the status or cost of links. Also assume that each node maintains at least a *routing table* and a *topology table.* At time t, the routing table of node i consists of a column vector of $|V^i(t)|$ row entries; the entry for destination node j specifies at least $s_j^i(t)$ and $D_j^i(t)$. The topology table of a node i has enough information for node i to be able to compute D_{jk}^i, where $k \in N_i(t)$.

Accordingly, for each destination j, the successor entries of the nodal routing tables in G define another graph, denoted $S_j(G)$, whose nodes are the same nodes of G, and in which a directed edge exists from node i to node k if and only if node k is s_j^i. Obviously, loop freedom is guaranteed at all times in G if $S_j(G)$ is always a directed acyclic graph, which is called the *acyclic successor graph* (ASG) of G for destination j. In steady state, when all routing tables are correct, $S_j(G)$ must be a tree.

Node i is said to be *upstream* of node k in $S_j(G)$ if the directed chain P_{ij} from node i to node j includes node k. Similarly, node k is *downstream* of node i. A node x is said to be the predecessor of another node y for destination j if node y is node x's successor.

Unless specified otherwise, any mention to entries in nodal tables or update messages refers to destination node j. Similarly, references will be made to node j, distance to node j, ASC for node j, and successor toward node j simply by destination, distance, ASG, and successor, respectively.

Consider a node i for whom either $s_j^i(t') = s \neq null$ and $D_j^i(t') < \infty$, or $s_j^i(t') = null$ and $D_j^i(t') = \infty$. Assume that node i makes no changes to such distance or successor entries, until time $t > t'$. Each one of the following three conditions, which we call *feasibility conditions* for loop freedom, is *sufficient* to ensure loop freedom at every instant in G.

DIC: *Distance increase condition.* If at time t node i detects a link-cost decrease or a decrease in the distance reported by a neighbor, then node i is free to choose as its new successor any neighbor $q \in N_i(t)$ for whom $D_{jq}^i(t) + l_q^i(t) = \text{Min}\{D_{jx}^i(t) + l_x^i(t) | x \in N_i(t)\}$ and $D_{jq}^i(t) + l_q^i(t) < \infty$. On the other hand, if node i detects an increase in the cost of

a link or the distance reported by a neighbor, then it must maintain its current successor if it has any.

A less restrictive version of DIC is stating that at time t node i can choose as its new successor any neighbor $q \in N_i(t)$ for which $D^i_{jq}(t) + l^i_q(t) = \text{Min } \{D^i_{jx}(t) + l^i_x(t) | x \in N_i(t)\}$ and $D^i_{jq}(t) + l^i_q(t) \leq FD^i_j(t)$, where $FD^i_j(t) = D^{*i}_j(t)$. Note that this version of DIC behaves the same way as the previous versin as long as distances or link costs do not increase. On the other hand, when a node detects a distance or link-cost increase, the new version of DIC allows the node to change successors in some cases, while the previous version forces it to maintain the same successor in all cases.

CSC: *Current successor condition.* If at time $t > t'$ node i needs to change its current successor, it can choose as its new successor any neighbor $q \in N_i(t)$ for which $D^i_{jq}(t) + l^i_q(t) = \text{Min } \{D^i_{jx}(t) + l^i_x(t) | x \in N_i(t)\}$ and, if $s^i_j(t') = s, D^i_{jq}(t) \leq FD^i_j(t)$, where $FD^i_j(t) = D^{*i}_{js}(t)$. If no such neighbor exists, then node i must maintain its current successor if it has any.

SNC: *Source node condition.* If at time t node i needs to change its current successor, it can choose as its new successor any neighbor $q \in N_i(t)$ for which $D^i_{jq}(t) + l^i_q(t) = \text{Min } \{D^i_{jx}(t) + l^i_x(t) | x \in N_i(t)\}$ and $D^i_{jq}(t) < FD^i_j(t)$, where $FD^i_j(t) = D^{*i}_j(t)$. If no such neighbor exists, then node i must maintain its old successor if it has any.

The variable FD^i_j is called the *feasible distance* of node i for destination j.

While condition DIC was discussed in the literature prior to the work by Jaffe and Moss, they were the first to prove that DIC is sufficient for loop freedom [13] in DBF. The same proof applies to an arbitrary routing algorithm. Condition CSC has been previously proposed and proven by this author [8]. The following theorem proves SNC.

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

This is evident from the fact that $S_j(G)$ is directed and acyclic before $L_j(t)$ is created.

Theorem 1: Using SNC when nodes choose their successors is sufficient to ensure that $S_j(G)$ is loop free at every instant. □

Proof: Note that because the theorem must apply to any distance-vector or link-state algorithm, each node in G can be assumed to know an entire path from any other node to the destination. However, even if this is the case, the information maintained at a given node may be out of date. This forms the basis of the proof, which is by contradiction.

Assume that, before time $t, S_j(G)$ is loop free at every instant and a loop $L_j(t)$ is formed in $S_j(G)$ at time t. It is evident that no loop can be created unless nodes change successors and modify $S_j(G)$, and it follows from Proposition 1 that at least one node must change its successor at time t and choose an upstream neighbor for a loop to be formed. Therefore, this proof needs to show only that SNC is sufficient to ensure loop freedom when at least one node in G changes its successor at time t.

Assume that $L_j(t)$ is formed when node i makes node a its new successor $s^i_j(t)$ after detecting a change in $D^i_j = D^i_{jb} + l^i_b$ at time t, where $b = s^i_j(t_b) \neq a$ and $t_b < t$. Because of $L_j(t), P_{aj}(t)$ must include $P_{ai}(t)$.

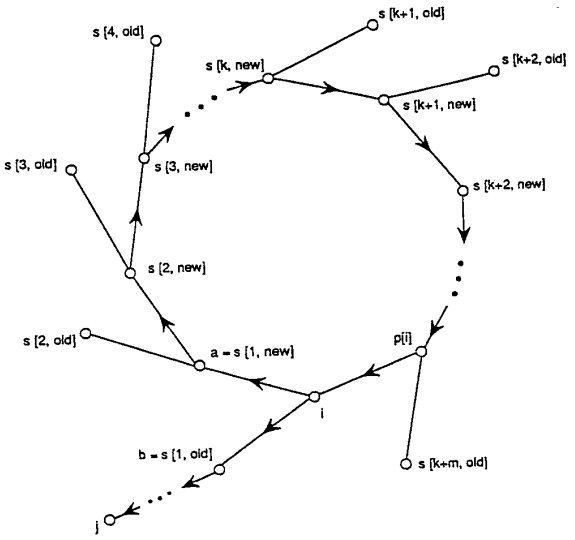

Fig. 1. Loop in G.

Let $P_{ai}(t)$ consist of the chain of nodes $\{a = s[1, \text{new}], s[2, \text{new}], \cdots, s[k, \text{new}], \cdots, i\}$, as shown in Fig. 1. According to this notation, node $s[k, \text{new}]$ is the kth hop in the path $P_{ai}(t)$ at time t and has node $s[k+1, \text{new}]$ as its successor at time t.

The last time that node $s[k, \text{new}]$ updates its routing-table entry up to time t and sets $s^{s[k,\text{new}]}_j = s[k+1, \text{new}]$ is denoted by $t_{s[k+1,\text{new}]}$, where $t_{s[k+1,\text{new}]} \leq t$. Therefore, it is true that

$$s^{s[k,\text{new}]}_j(t_{s[k+1,\text{new}]}) = s^{s[k,\text{new}]}_j(t)$$

and

$$D^{s[k,\text{new}]}_j(t_{s[k+1,\text{new}]}) = D^{s[k,\text{new}]}_j(t).$$

The time when node $s[k, \text{new}]$ *sends an update* that constitutes the last update from such a node that is processed by node $s[k-1, \text{new}]$ up to time t is denoted by $t_{s[k+1,\text{old}]}$.

Node $s[k, \text{new}]$'s successor at time $t_{s[k+1,\text{old}]}$ is denoted by $s[k+1, \text{old}]$. Note that

$$t_{s[k+1,\text{old}]} \leq t_{s[k+1,\text{new}]} \leq t,$$

and that $s[k+1, \text{old}]$ need not be the same as $s[k+1, \text{new}]$. Lastly

$$s^{p[i]}_j(t) = i, \qquad s^t_j(t_b) = b, \qquad \text{and } t_b < t.$$

Note that $D^{*i}_j(t_i) \leq D^i_j(t_i)$ at any time t_i, and $D^{*i}_j(t_2) \leq D^{*i}_j(t_1)$ if $t_1 < t_2$. Also note that, because SNC must be satisfied, when node $s[k, \text{new}] \in P_{aj}(t)$ makes node $s[k+1, \text{new}] \in P_{aj}(t)$ its successor at time $t_{s[k+1,\text{new}]}$ it must be true that

$$\begin{aligned} D^{s[k,\text{new}]}_{js[k+1,\text{new}]}(t) &= D^{s[k,\text{new}]}_{js[k+1,\text{new}]}(t_{s+1,\text{new}]}) \\ &< FD^{s[k,\text{new}]}_j(t_{s[k+1,\text{new}]}). \end{aligned}$$

Accordingly, because all link costs are positive and SNC must be satisfied by every node in $P_{ai}(t)$, traversing the

directed path $P_{ai}(t) \subset P_{aj}(t)$ at time t leads to the following inequalities:

$$\begin{aligned} FD_j^i(t) = D_j^{*i}(t) &> D_{ja}^i(t) = D_j^a(t_{s[2,\text{old}]}) \\ D_j^a(t_{s[2,\text{old}]}) \geq D_j^{*a}(t_{s[2,\text{old}]}) &\geq D_j^{*a}(t_{s[2,\text{new}]}) \\ &= FD_j^a(t_{s[2,\text{new}]}) > D_{js[2,\text{new}]}^a(t) \\ &\vdots \\ D_{js[k,\text{new}]}^{s[k-1,\text{new}]}(t) = D_j^{s[k,\text{new}]}&(t_{s[k+1,\text{old}]}) \\ &\geq D_j^{*s[k,\text{new}]}(t_{s[k+1,\text{old}]}) \\ &\geq D_j^{*s[k,\text{new}]}(t_{s[k+1,\text{new}]}) \\ &= FD^{s[k,\text{new}]}(t_{s[k+1,\text{new}]} > D_{js[k+1,\text{new}]}^{s[k,\text{new}]}&(t) \\ &\vdots \\ D_{ji}^{p[i]}(t) = D_j^i(t_b) \geq D_j^{*i}(t) &= FD_j^i(t). \end{aligned}$$

Because these inequalities lead to the erroneous conclusion that $FD_j^i(t) > FD_j^i(t)$, it follows that no loop can be formed in $S_j(G)$, and SNC is sufficient in this case.

The operation of any routing algorithm can be defined to be such that when the nodes in G are first initialized, each node knows only how to reach itself. This is equivalent to saying that a node has a routing table entry for each of the other nodes in the graph with infinite distance and no successors to them. Hence, at time 0, $S_j(G)$ is a disconnected graph of one or more components, each with a single node, and must be loop-free. Therefore, SNC is sufficient. □

From the above proof of SNC, it is clear that other similar feasibility conditions can be defined using a feasible distance that can only decrease. Although DIC, CSC, and SNC ensure loop freedom at every instant, none of them guarantees shortest paths in the resulting ASG. Deriving routing algorithms based on these feasibility conditions to achieve both loop freedom at every instant and shortest paths for each destination is the subject of the next section.

IV. Diffusing Computations

Dijkstra and Scholten [3] introduced the concept of *diffusing computations* to check the termination of a computation distributed among several nodes, such that the process starting a computation is informed when it is completed and such that there are no false terminations. The diffusing computation started by a node grows by sending *queries* and shrinks by receiving *replies* along an acyclic graph rooted at the source of the computation. The algorithm itself can be used to construct the acyclic graph.

The new family of routing algorithms presented in this paper is based on an adaptation of diffusing computations to distance-vector routing inspired on the distance-vector algorithm proposed by Jaffe and Moss [13]. It allows a given node i to modify FD_j^i in such a way that loop freedom and shortest paths are achieved when it changes the values of D_j^i or s_j^i. The rest of this paper refers to any algorithm in this family simply as a *diffusing update algorithm* or DUAL. Reference [9] provides a formal description of DUAL, whose operation is discussed in the rest of this section.

For each destination j, a change in the cost or status of a link that affects $S_j(G)$ causes one or more computations aimed at updating $S_j(G)$. A computation can be carried out by a node independently of others, which is called a *local computation,* or it can be a *diffusing computation* in which the node that originates the computation coordinates with upstream nodes in $S_j(G)$ before making any updates to the ASG.

At time t, node i is assumed to maintain l_k^i and D_{jk}^i for all $k \in N_i(t)$ as well as s_j^i, D_j^i, RD_j^i, and FD_j^i.

A. A Single Computation

Initially, no node in G is engaged in a diffusing computation for a new $S_j(G)$. A node not engaged in a diffusing computation is said to be *passive* (with respect to destination j). When a passive node i detects a change in a link cost or status at time t that changes the value of D_j^i or s_j^i, it first tries to obtain a new successor that satisfies a feasibility condition (DIC, CSC, SNC, or others), which is denoted by FC. Such a successor is called a *feasible successor.*

If node i finds a feasible successor at time t, DUAL behaves much like DBF; that is, node i carries out a *local computation* to update its distance and successor. More specifically, node i first computes the minimum of $D_{jq}^i(t) + l_q^i(t) = D_{\min}$ for all $q \in N_i(t)$. Secondly, node i updates $D_j^i = D_{\min}, RD_j^i = D_{\min}, s_j^i = k | k \in N_i(t), D_{jk}^i(t) + l_k^i(t) = D_{\min}$, and FD_j^i equal to the smaller of its previous value or $D_{jk}^i(t)$ if CSC is used, or $D_j^i(t)$ if SNC or DIC is used. Finally, if node i's updated distance is different than its previous distance, it sends an update to all its neighbors specifying $RD_j^i(t)$. Note that node i establishes no coordination with its neighbors before updating D_j^i and s_j^i.

On the other hand, if node i cannot find a feasible successor, then it sets D_j^i and RD_j^i equal to $D_{js}^i(t) + l_s^i(t)$, where s is its current successor. In addition, node i sets $FD_j^i = D_{js}^i(t)$ if CSC is used, or sets $FD_j^i = D_j^i(t)$ if SNC or DIC is used. After performing these updates, node i commences a *diffusing computation* by sending a query to all its neighbors in $N_i(t)$. Such a query simply contains $RD_j^i(t)$, i.e., node i's new distance through its current successor in $S_j(G)$. Node i is then said to be *active,* and cannot change its successor [i.e., it cannot change $S_j(G)$] or the values of RD_j^i and FD_j^i until it receives all the replies to its query.

Node i follows the same procedure outlined above for a local or diffusing computation after receiving an update from a neighbor while it is passive. If node i receives a query from a neighbor while it is passive, then node i attempts to find a feasible successor and sends a reply to its neighbor with D_j^i if it succeeds. Node i also sends an update to the rest of its neighbors if the value of D_j^i changes. If node i fails to find a feasible successor after processing its neighbor's query, then it becomes active by sending a query to all its neighbors; the value of RD_j^i in the query specifies node i's new distance through its current successor.

When at time t_p a node receives all the replies to the query it sent out, it becomes passive once more. At that time, node

i can be certain that all nodes that were upstream in $S_j(G)$ at time t_p have either modified their distances as a result of the distance reported by node i's query, or stopped being upstream nodes. Node i is, therefore, free to choose as its successor a neighbor that offers the shortest distance at time t_p. Accordingly, at time t_p, node i "resets" the value of FD_j^i to ∞, which ensures that FC will be satisfied by a node $n \in N_i(t_p)$ that offers the shortest distance. After node i makes node n its new successor, it sets $RD_j^i = D_j^i(t_p)$. If SNC or DIC is used, it also sets $FD_j^i = D_j^i(t_p)$; if CSC is used, it sets $FD_j^i = D_{jn}^i(t_p)$.

Node i uses a *reply status flag,* denoted r_{jk}^i to remember whether node k has sent a reply to node i's query. Node i is passive at time t if $r_{jk}^i(t) = 0$ for all $k \in N_i(t)$. Node i becomes active at time t by setting $r_{jk}^i = 1$ for all $k \in N_i(t)$.

Routing information is exchanged among neighbor nodes by means of update messages. Each update message consists of a distance vector of one or more entries, and each such entry consists of the identifier of a destination node j, RD_j^i, and a flag that specifies that the entry is an update (flag equals 0), a query (flag equals 1), or a reply (flag equals 2).

Obviously, the basic algorithm described above needs to be extended to handle multiple diffusing computations and topological changes.

The Jaffe–Moss algorithm behaves essentially like DUAL using DIC for the case in which a single diffusing computation exists.

Chandy and Misra proposed a distance-vector algorithm aimed at obtaining the shortest paths from a source to the rest of the network nodes. This algorithm is based on a simpler adaptation of Dijkstra and Scholten's algorithm than the one just described [1], and performs correctly only in fixed topologies.

Merlin and Segall [17], [23] proposed a distance-vector algorithm that propagates messages along an ASG much like queries and replies do in DUAL. However, critical differences between the two approaches stem from the way in which messages are processed. In the Merlin–Segall algorithm, a node i that receives a message from a node $k \in N_i$ other than its successor simply updates D_{jk}^i with the distance reported by node k. Node i keeps track of which neighbor has sent a message using a flag similar to the reply status flag of DUAL. Node i does not update its own distance until it receives a message from its successor; when that happens, node i sets D_j^i equal to Min $\{D_{jk}^i + l_k^i | k \in N_i$, message from k was received$\}$ and sends a message reporting that distance to all its neighbors, except its successor. When node i receives a message from all its neighbors, it sends a message reporting the current value of D_j^i to its current successor, computes a new shortest distance, and sets its new successor equal to that neighbor that provides the shortest distance. Because the messages a node sends to its successor or its other neighbors need not report its shortest distance, multiple iterations or cycles can be required for the Merlin–Segall algorithm to converge. Each update cycle is initiated by the destination node, and unbounded counters are used to keep track of cycle numbers. The need for the destination node to control each update cycle creates substantial communication overhead [22].

B. Multiple Computations

The Jaffe–Moss algorithm [13] supports multiple diffusing computations concurrently by maintaining *bit vectors* at each node. Bit vectors specify, for each neighbor and for each destination, how many queries need to be answered by the neighbor node and which one of such queries was originated by the node maintaining the bit vector. Unfortunately, the bit vectors used in the Jaffe–Moss algorithm can become exceedingly large in a large network with dynamic link costs and topology. The Merlin–Segall algorithm relies on unbounded sequence numbers to keep track of multiple computations that can occur concurrently from the standpoint of a node in the ASG.

Rather than handling multiple diffusing computations concurrently, DUAL makes sure that a node participates (i.e., is active) in at most one diffusing computation per destination at any given time. This is accomplished by means of the algorithm represented in Fig. 2, which assumes a stable topology. The state diagram of Fig. 2 shows the transition to a new state for node i, given its current state, the input event it receives, and whether or not FC is satisfied. An input event can be a change in the cost or status of a link, an update, a query, or a reply to a query.

The current state of node i is specified by the *query origin flag,* denoted o_j^i, and by the reply status flags. The possible states for node i are the following:

- Node i becomes active by relaying the query received from its successor, and experiences no distance increases after becoming active ($o_j^i = 3$ and $r_{jk}^i = 1$ for some $k \in N_i(t)$).
- Node i is active and either relayed the query in progress from its successor and has experienced at least one distance increase after becoming active, or is the origin of the query in progress and received another query from its successor after becoming active ($o_j^i = 2$ and $r_{jk}^i = 1$ for some $k \in N_i(t)$).
- Node i becomes active by originating a query, and experiences no distance increase and receives no query from its successor while it is active with its query ($o_j^i = 1$ and $r_{jk}^i = 1$ for some $k \in N_i(t)$).
- Node i is the origin of a query in progress and has experienced at least one distance increase after becoming active because of updates from its successor or link-cost increases ($o_j^i = 0$ and $r_{jk}^i = 1$ for some $k \in N_i(t)$).
- Node i is passive. Because $r_{jk}^i(t) = 0$ for all $k \in N_i(t)$ for node i to be passive, o_j^i is set equal to 1 in this state.

When node i is passive and processes a query from a node k other than its current successor s_j^i, if FC is not satisfied, it sends a reply to its neighbor with the current value of RD_j^i before it starts its own query. This way, node i can create a new diffusing computation without having to remember to send a reply to both k and s_j^i when it becomes passive.

When node i transitions from active to passive state and $o_j^i = 1$ or 3, then it "resets" the value of its feasible distance FD_j^i to ∞ before computing D_j^i and s_j^i. Hence, node i simply

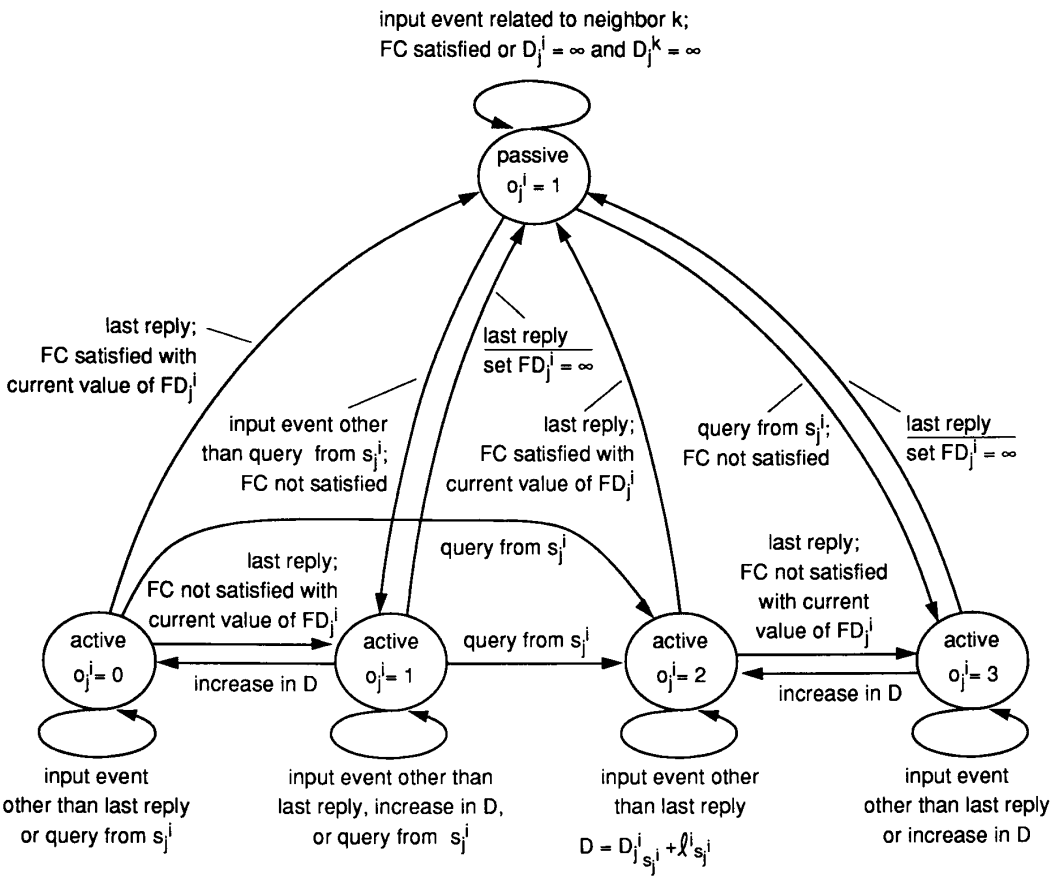

Fig. 2. Active and passive states in DUAL.

chooses the shortest path in this case. In contrast, when node i becomes passive and $o_j^i = 0$ or 2, it uses the value of FD_j^i at the time it became active to determine whether or not a feasible successor exists before computing D_j^i and s_j^i. Hence, node i processes any pending query or distance increases that occurred while it was active.

C. Handling Special Conditions and Topological Changes

Ensuring that updates will stop being sent in G when some destination is unreachable is easily done. If node i has set $D_j^i = \infty$ already and receives an input event (a change in cost or status of link (i, k), or an update or query fron node k) such that $D_{jk}^i + l_k^i = \infty$, then node i simply updates D_{jk}^i or l_k^i, and sends a reply to node k with $RD_j^i = \infty$ if the input event is a query from node k.

A node initializes itself in passive state and with an infinite distance for all its known neighbors, and with a zero distance to itself. After initialization, it sends an update containing the distance to itself to all its known neighbors.

When node i establishes a link with a neighbor k, it updates the value of l_k^i and assumes that node k has reported infinite distances to all destinations and has replied to any query for which node i is active. Furthermore, if node k is a previously unknown destination, node i sets $o_k^i = 1$, s_k^i = null, and $D_k^i = RD_k^i = FD_k^i = \infty$. Node i also sends to its new neighbor k an update for each destination for which it has a finite distance.

When node i is passive and detects that link (i, k) has failed, it sets $l_k^i = \infty$ and $D_{jk}^i = \infty$. After that, node i carries out the same steps used for the reception of a link-cost change in the passive state.

For a given destination, a node can become active in only one diffusing computation at a time and can, therefore, expect at most one reply from each neighbor. Accordingly, when an active node i loses connectivity with a neighbor n, node i can set $r_{jn}^i = 0$ and $D_{jn}^i = \infty$, i.e., assume that its neighbor n sent any required reply reporting an infinite distance. If node n is s_j^i, node i also sets $o_j^i = 0$. When node i becomes passive again and $o_j^i = 0$, it cannot simply choose a shortest distance; rather, it must find a neighbor that satisfies FC using the value of FD_j^i set at the time node i became active in the first place. In effect, this is equivalent to deferring the processing of the failure of link (i, s_j^i) that took place while node i was active, which may create another diffusing computation.

It, thus, follows that the FIFO order in which node i processes diffusing computations does not change with the establishment of new links or link failures. Note that the addition or failure of a node is handled by its neighbors as link additions or failures.

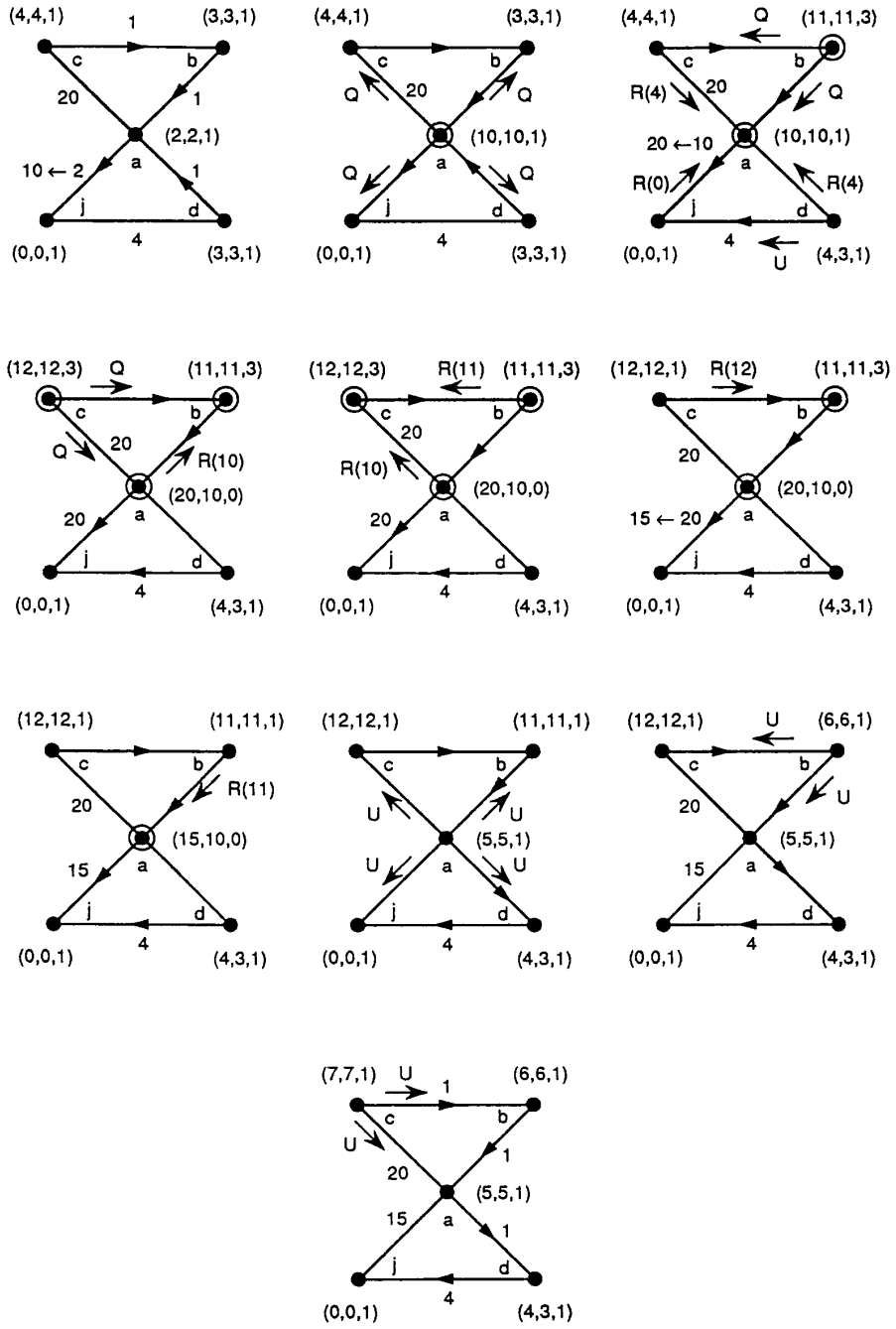

Fig. 3. Example of DUAL's operation using SNC.

V. Example

Consider the five-node network depicted in Fig. 3(a), in which DUAL is used with SNC. In this network, links and nodes have the same propagation or processing delays; link (a, j) first changes its cost from 2 to 10, then from 10 to 20 [Fig. 3(c)], and finally from 20 to 15 [Fig. 3(f)], and focuses on destination node j. In the figure, an arrowhead in the directed link from node x to node y indicates that node y is the successor toward node j (i.e., $s_j^x = y$). The label in parentheses assigned to node x indicates the value of D_j^x, followed by the value of FD_j^x and o_j^x. Fig. 3(b)–(j) show the changes in the routing table entry for node j at each node. An update or a query is indicated by an arrow adjacent to the link where it is transmitted, followed by a "U" or a "Q," respectively. A reply from node x is similarly identified with an "R" followed by the value of RD_j^x in parentheses.

When node a detects the cost increase of link (a, j), it determines that it has no feasible successor because none of its neighbors has a distance smaller than $FD_j^a = 2$. Accordingly, it becomes active by setting $r_{jb}^a, r_{jc}^a, r_{jd}^a$, and r_{jj}^a equal to 1, and sends a query to all its neighbors [Fig. 3(b)].

Node b forwards node a's query, because it has no feasible successor [Fig. 3(c)], while node d is able to find a feasible successor (node j itself, because $D_{jj}^d < FD_j^d = 3$ and $D_{jj}^d + l_j^d < D_{ja}^d + l_a^d$). While this is happening, link (a, j) increases its cost to 20, which makes node a set $D_j^a = 20$ and $o_j^a = 0$. The latter makes sure that node a uses $FD_j^a = 10$, not ∞, when it becomes passive and computes a new distance and successor. Note that node a replies to node b with $RD_j^a = 10$, which equals node a's distance when it became active in the first place [Fig. 3(d)]; this prevents node b from creating a loop through node c when it becomes passive [Fig. 3(g)].

When node c receives node a's query, it simply sends a reply [Fig. 3(c)] because it has a feasible successor. However, it becomes active when it receives the query from node b. Because $o_j^c = 3$ when node c receives all the replies to its query [Fig. 3(e)], it resets $FD_j^c = \infty$ to compute its new distance and successor, and sets $D_j^c = FD_j^c = 12$ accordingly [Fig. 3(f)]. Node b operates in a similar manner when it receives all the replies to its query [Fig. 3(g)].

When node a receives all the replies to its query [Fig. 3(h)], it applies SNC using the value $FD_j^a = 10$, which was set when node a became active in the first place. Because $D_{jd}^a = 4 < 10$ and $D_{jd}^a + l_d^a$ is the minimum for all of node a's neighbors, node a becomes passive, sends an update with $RD_j^a = 5$, and sets $o_j^a = 1$ and $D_j^a = FD_j^a = 5$. Nodes b and c eventually reach their shortest paths through uncoordinated updates.

Note that the cost changes in link (a, j) that occur while node a is active have no impact in the information conveyed by node a to its neighbors, until it becomes passive.

VI. Correctness of Dual

Once DUAL is proven to be loop-free, showing that it converges is simple. The details of the convergence proof for DUAL are presented in [9]; the proof is similar to the one shown in [5]. The rest of this section proves that DUAL is loop-free.

Because a routing-table loop is created with respect to a particular destination and because nodes coordinate with respect to the same ASG, loop freedom can be proven by focusing on a single ASG. The following theorem considers a graph G with stable topology in which DUAL is used with SNC, and demonstrates that DUAL is loop-free at every instant. Because topological changes do not affect DUAL's operation, this theorem suffices to prove loop freedom in an arbitrary network or internet in which topological changes can occur. The proof is essentially the same if DIC or CSC is used.

The graph consisting of the set of nodes upstream of node i that become active because of a query originated at node i is called the *active* ASG of node i and is denoted by $S_{ji}(G)$. The notation adopted in the proof of Theorem 1 is assumed in the rest of this section.

Theorem 2: $S_j(G)$ is loop-free at every instant if DUAL is used in G. □

Proof: The proof follows directly from the lemmas below. □

Lemma 1: When a node becomes passive, it must send a reply to its successor if it is not the origin of the diffusing computation. □

Proof: If node i receives a query from its successor (s_j^i) while it is passive, it sends a reply to its successor and sets $o_j^i = 1$ if it finds a feasible successor. Otherwise, if node i finds no feasible successor, it propagates the diffusing computation to all its neighbors, sets $o_j^i = 3$, and becomes active. Node s_j^i cannot send another query to node i until it receives node i's reply, and node i must send such a reply when it becomes passive because $o_j^i = 3$.

If node i receives a query from its successor while it is already active, node i must be the origin of the diffusing computation for which it is active $(o_j^i = 1)$ because its successor cannot send two consecutive queries without receiving node i's reply. After processing its successor's query, node i must set $o_j^i = 2$. When node i becomes passive, it either sends a reply to its successor and sets $o_j^i = 1$ after finding a feasible successor, or forwards its successor's query and sets $o_j^i = 3$ if no feasible successor is found. Hence, if node i receives a query from its successor while it is active, node i must send a reply to its successor when it becomes passive. Therefore, the lemma is true. □

Lemma 2: Consider that only a single diffusing computation takes place in G and that $S_j(G)$ is loop-free before an arbitrary time t. Then, independently of the state of other nodes in $P_{aj}(t)$, if node $s[k, \text{new}]$ is passive at time t (i.e., either it is passive immediately before time t or it becomes passive during time t), it must be true that

$$D_{js[k,\text{new}]}^{s[k-1,\text{new}]}(t) > D_{js[k+1,\text{new}]}^{s[k,\text{new}]}(t). \tag{1}$$

□

Proof: Let node $s[k, \text{new}] \in P_{aj}(t)$ be passive at time t. Consider the case in which that node does not reset $FD_j^{s[k,\text{new}]}$ when it updates its distance or successor to join $P_{aj}(t)$ at time $t_{s[k+1,\text{new}]} \leq t$, then according to SNC it must be true that

$$D_{js[k+1,\text{new}]}^{s[k,\text{new}]}(t) < FD_j^{s[k,\text{new}]}(t) \leq D_j^{s[k,\text{new}]}(t_{s[k+1,\text{old}]}).$$

Hence, if node $s[k-1, \text{new}]$ processed the message that node $s[k, \text{new}]$ sent last before time t, then

$$D_{js[k,\text{new}]}^{s[k-1,\text{new}]}(t) = D_j^{s[k,\text{new}]}(t) > D_{js[k+1,\text{new}]}^{s[k,\text{new}]}(t)$$

simply because all link costs are positive. On the other hand, if $s[k-1, \text{new}]$ did not process the message that node $s[k, \text{new}]$ sent last before time t, then

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

because SNC must be satisfied. Therefore, the lemma is true for this case.

On the other hand, if node $s[k, \text{new}]$ is passive at time t and resets $FD_j^{s[k,\text{new}]}$ when it joins $P_{aj}(t)$ at time $t_{s[k+1,\text{new}]} \leq t$,

then $o_j^{s[k,\text{new}]}$ must have been equal to 1 or 3, and node $s[k, \text{new}]$'s distance through its new successor must be the shortest. Therefore

$$D_j^{s[k,\text{new}]}(t) = D_j^{s[k,\text{new}]}(t_{s[k+1,\text{new}]}) \leq \quad (2)$$

$$D_{js[k+1,\text{old}]}^{s[k,\text{new}]}(t_{s[k+1,\text{new}]}) + l_{s[k+1,\text{old}]}^{s[k,\text{new}]}(t_{s[k+1,\text{new}]}).$$

Also, from time t_k when node $s[k, \text{new}]$ becomes active to time $t_{s[k+1,\text{new}]}$ when it becomes passive, node $s[k, \text{new}]$ cannot change its successor, node $s[k+1, \text{old}]$, and it cannot experience any increments in its distance through node $s[k+1, \text{old}]$, for otherwise it would have set $o_j^{s[k,\text{new}]} = 0$ or 2. Therefore,

$$\begin{aligned} D_j^{s[k,\text{new}]}(t_k) &= D_{js[k+1,\text{old}]}^{s[k,\text{new}]}(t_k) + l_{s[k+1,\text{old}]}^{s[k,\text{new}]}(t_k) \\ &= D_{js[k+1,\text{old}]}^{s[k,\text{new}]}(t_{s[k+1,\text{new}]}) \\ &\quad + l_{s[k+1,\text{old}]}^{s[k,\text{new}]}(t_{s[k+1,\text{new}]}). \end{aligned}$$

Substituting this result into (2), it follows that

$$D_j^{s[k,\text{new}]}(t) \leq D_j^{s[k,\text{new}]}(t_k). \quad (3)$$

Furthermore, node $s[k, \text{new}]$ must notify all its neighbors about its distance at time t_k, and all of them must reply to node $s[k, \text{new}]$ before time $t_{s[k+1,\text{new}]}$, which follows from the correctness proof of Dijkstra and Scholten's algorithm [3] (see also the proof of PIF [23]).

When node $s[k-1, \text{new}]$ updates its distance and makes node $s[k, \text{new}]$ its successor when it joins $P_{aj}(t)$ at time $t_{s[k,\text{new}]} \leq t$, it may or may not have processed any message sent by node $s[k, \text{new}]$ at time $t_{s[k+1,\text{new}]} \leq t$ when that node joins $P_{aj}(t)$. Therefore, it must be true that either

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

or

$$D_{js[k,\text{new}]}^{s[k-1,\text{new}]}(t) = D_j^{s[k,\text{new}]}(t_k).$$

In either case, it follows from (3) that

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

and the lemma is true in all cases. □

Lemma 3: If only a single diffusing computation takes place in G, then $S_j(G)$ is loop-free at every instant. □

Proof: By contradiction, assume that $S_j(G)$ is loop-free before an arbitrary time t and that node i is the first node in G to create a loop $L_j(t) \subset S_j(G)$ at time t after processing an input event, i.e., it establishes the last hop of the loop.

Let node $b = s_j^i$ when node i decides to change its successor at time t. Because an active node cannot change its successor after processing an input event, it follows that, for node i to create $L_j(t)$ at time t, it must be passive at time t and it must change its successor to $s_j^i(t) = a \neq b$.

When node i creates $L_j(t)$ at time t, it must be true that $P_{ai}(t) \subset P_{ij}(t)$. If all the nodes in $P_{ai}(t)$ are passive at time t, either all of them have always been passive before time t, or at least one of them was temporarily active before time t. If no node in $P_{ai}(t)$ was ever active before time t, it follows from Theorem 1 that node i cannot create $L_j(t)$ by making $a = s_j^i(t)$. Therefore, for node i to create $L_j(t)$ at time t in this case, at least one node in $P_{ai}(t)$ must have been temporarily active before time t.

Assume that all nodes in $P_{ai}(t)$ are passive at time t. The inequality in (1) leads to the erroneous conclusion that $D_{ja}^i(t) > D_{ja}^i(t)$ when $P_{ai}(t)$ is traversed at time t. Accordingly, node i cannot create $L_j(t)$ if all nodes in $P_{ai}(t)$ are passive at time t.

Now assume that node $s[k+n, \text{new}]$ $(n \geq 0)$ is the origin of the only diffusing computation in G and that the computation started at time $t_{k+n} \leq t$. Further assume that the nodes in the chain $P_{s[k,\text{new}]\,s[k+n,\text{new}]}(t) \subset P_{ai}(t)$ remain active at time t. Because node i must create $L_j(t)$ at time t, it must be true that $s[k, \text{new}] \neq i \neq s[k+n, \text{new}]$.

Because all the nodes in $P_{s[k,\text{new}]\,s[k+n,\text{new}]}(t)$ are active, all of them must have received a query reflecting the change in $D_j^{s[k+n,\text{new}]}$ made by node $s[k+n, \text{new}]$ at time t_{k+n}. Hence, from the correctness proof of Dijkstra and Scholten's algorithm [3] and the facts that all link costs are positive and a single diffusing computation exists in G, it follows that

$$\begin{aligned} D_{js[k+1,\text{new}]}^{s[k,\text{new}]}(t) &> D_j^{s[k+1,\text{new}]}(t) > \cdots > D_j^{s[k+n,\text{new}]}(t) \\ &= D_j^{s[k+n,\text{new}]}(t_{k+n}) > D_{js[k+n+1,\text{new}]}^{s[k+n,\text{new}]}(t) \end{aligned} \quad (4)$$

Because there is only one diffusing computation up to time t and at that time node $s[k+n+1, \text{new}]$ is passive and is the successor of the source of the diffusing computation, it follows that the nodes in $P_{s[k+n+1,\text{new}]i}(t)$, who are downstream of the source of the diffusing computation (node $s[k+n, \text{new}]$), have always been passive. Therefore, the proof of Theorem 1 implies that

$$D_{js[k+n+2,\text{new}]}^{s[k+n+1,\text{new}]}(t) > D_j^i(t_b).$$

Furthermore, (1) implies that

$$D_{js[k+n+1,\text{new}]}^{s[k+n,\text{new}]}(t) > D_{js[k+n+2,\text{new}]}^{s[k+n+1,\text{new}]}(t).$$

Substituting these two results in (4) yields

$$D_{js[k+1,\text{new}]}^{s[k,\text{new}]}(t) > D_j^i(t_b). \quad (5)$$

Because all of the nodes in $P_{ai}(t) - P_{s[k,\text{new}]\,s[k+n,\text{new}]}(t)$ and node i must be passive at time t, it follows from (1) that

$$D_{ja}^i(t) > D_{js[k,\text{new}]}^{s[k-1,\text{new}]}(t). \quad (6)$$

If node $s[k-1, \text{new}]$ processes the query sent by node $s[k, \text{new}]$ at time t_k, whether or not node $s[k-1, \text{new}]$ resets $FD_j^{s[k,\text{new}]}$ to update its distance for the last time before joining $L_j(t)$, it must be true that

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

Together with (5) and (6), this result leads to the conclusion that $D^i_{ja}(t) > D^i_j(t_b)$. However, as the next paragraph shows, this is a contradiction.

If node i does not reset FD^i_j when it updates its successor when it creates $L_j(t)$ at time t, then according to SNC, this means that node i can choose node a as its new successor at time t only if $D^i_{ja}(t) < D^i_j(t_b)$ is true, which contradicts the result of the previous paragraph. On the other hand, if node i resets FD^i_j to create $L_j(t)$, then it must be true that node i was active just before time t, that o^i_j equals 1 or 3 when it decides to make node a its new successor, and that it processed no query from node b or any input event that increased D^i_{jb} or l^i_b while it was active, for otherwise it must have set o^i_j equal to 2 or 0. Because in this case node i becomes passive at time t and node a is upstream of node i at that time, it follows from the correctness proof of Dijkstra and Scholten's algorithm that all the nodes in the path $P_{ap[i]}(t)$ must be passive immediately before time t, where $p[i]$ is the predecessor of node i at time t. Hence, (1) can be used to show that $D^i_{ja}(t) > D^{p[i]}_{ji}(t) = D^i_j(t_i)$, where $t_i < t$ is the time when node i becomes active. Furthermore, because D^i_{jb} or l^i_b cannot increase between times t_i and t, it must be true that $D^i_{ja}(t) > D^i_j(t_i) = D^i_{jb}(t) + l^i_b(t)$. This is in contradiction to SNC, which states that node i can set $a = s^i_j(t)$ only if $D^i_{ja}(t) + l^i_a(t) \leq D^i_{jb}(t) + l^i_b(t)$.

Alternatively, if by time t node $s[k-1, \text{new}]$ has not processed the query sent by node $s[k, \text{new}]$ at time t_k, then whether or not node $s[k-1, \text{new}]$ resets $FD^{s[k-1,\text{new}]}_j$ to update its distance for the last time before time t, node $s[k, \text{new}]$ was always passive before time t_k and therefore cannot reset $FD^{s[k,\text{new}]}_j$ to select $s[k+1, \text{new}]$ as its successor before it becomes active at time t_k. Given that node $s[k+n+1, \text{new}]$ has always been passive, and that no node

$$s[k+m, \text{new}] \in P_{s[k,\text{new}]\, s[k+n,\text{new}]}(t) \quad (1 \leq m \leq n)$$

could have reset $FD^{s[k+m,\text{new}]}_j$ to make node $s[k+m+1, \text{new}]$ its successor at time $t_{s[k+m+1,\text{new}]} < t_{k+m}$, the path-traversal technique used in Theorem 1 leads to the following inequality:

$$D^{s[k-1,\text{new}]}_{js[k,\text{new}]}(t) > D^{s[k+n+1,\text{new}]}_{js[k+n+2,\text{new}]}(t). \tag{7}$$

Because the nodes in $P_{s[k+n+1,\text{new}]i}(t)$ have always been passive, the proof of Theorem 1 implies that $D^{s[k+n+1,\text{new}]}_{js[k+n+2,\text{new}]}(t) > D^i_j(t_b)$. Combining this result with (7) and (6) leads again to the erroneous conclusion that $D^i_{ja}(t) > D^i_j(t_b)$. Therefore, node i cannot create a loop when it is passive at time t and a set of nodes in $P_{ai}(t)$ are still active at that time.

It follows from the above that no loop can be formed when $S_j(G)$ is loop-free before time t and a single diffusing computation occurs. However, as pointed out in the proof of Theorem 1, $S_j(G)$ must be loop-free at time $t = 0$, and so the lemma is true. □

Lemma 4: DUAL considers each computation individually and in the proper sequence. □

Proof: Consider the case in which node i is the only node that can start diffusing computations. If node i generates a single diffusing computation, the proof is immediate. If node i generates multiple diffusing computations, no node in $S_{ji}(G)$ can send a new query before it receives all the replies to the query for which it is currently active. Therefore, because all the nodes in $S_{ji}(G)$ process each input event in FIFO order, because all links transmit in FIFO order, and because each node that becomes passive must send the appropriate reply to its successor if it has any (Lemma 1), it follows that all the nodes in $S_{ji}(G)$ must process each diffusing computation individually and in the proper sequence.

Consider now the case in which multiple sources of diffusing computations exist in G. Note that once a node sends a query, it must become passive before it can send another query. Hence, a node can be part of only one active ASG at any given time. Furthermore, when node i becomes active as a result of a query from a neighbor k, only two things can happen. If $k = s^i_j$, then node i is not the origin of the query, forwards node k's query, and becomes part of the same active ASG to whom k already belongs. If $k \neq s^i_j$, then node i sends a reply to node k before creating a diffusing computation, which means that $S_{ji}(G)$ is not part of the active ASG to whom k belongs. Because the active ASGs of G have an empty intersection at any given time, it follows from the previous case that the lemma is true. □

VII. Performance Comparison

A. Complexity of Algorithms

This section compares DUAL's performance with the performance of DBF, an ideal link-state algorithm (ILS), and the Merlin–Segall algorithm. This comparison is made in terms of the time and communication overhead required by the various algorithms to converge to correct routing tables. Unfortunately, the time required for a given state of a distributed algorithm to occur depends on the timing with which the nodes execute the algorithm and on the delays incurred in internodal communication [3]. Consequently, this section assumes that the algorithms under study behave synchronously, so that every node in the network executes a step of the algorithm simultaneously at fixed points in time. At each step, a node receives and processes all input events originated during the preceding step and, if needed, creates an update message after processing each input event; all these update messages are transmitted in the same step. The first step occurs when at least one node detects a topological change and issues update messages to its neighbors. During the last step, at least one node receives and processes updates from its neighbors, after which all routing tables are correct and nodes stop transmitting updates until a new topological change takes place [13], [14]. Using this assumption, the performance of an algorithm is quantified in terms of the number of steps called *time complexity* or TC, and the number of messages called *communication complexity* or CC, required by each algorithm after a single change in the cost or status of a link.

Because of its looping problems [7], DBF performs very poorly after a link failure or a link-cost increase, namely

$TC = O(N), CC = O(N^2)$ [14], where N is the number of network nodes. On the other hand, because no loops can exist in DBF after the addition of a link or a reduction in the cost of a link [13], it has $TC = O(d)$ and $CC = O(E)$ in this case [15], where E is the number of links in the network and d is the diameter of the network.[1]

ILS requires that each change in the topology of the network be transmitted to every node; accordingly, it has $TC = O(d)$ and $CC = O(2E)$ after a single change in the cost or status of a link because each link-state update traverses each link at most once in each direction and there are two link costs per link.

The Merlin–Segall algorithm has $TC = O(d^2)$ [13], and its required update coordination results in a large number of update messages: $CC = O(N^2)$ [22].

In the worst case, DUAL has the same time and communication complexity as the Jaffe–Moss algorithm after a single link failure or link-cost increase, i.e., $TC = O(x)$ [13], where x is the number of nodes affected by the routing table perturbation. To verify this, we note that in the worst case all nodes upstream of a destination node j in $S_j(G)$ must freeze their routing-table entries for node j, which corresponds to the operation of the Jaffe–Moss algorithm. DUAL's communication complexity is $CC = O(6D \cdot x)$ after a single link failure or link-cost increase, where D is the maximum degree of a node. This value is derived from the fact that each adjacent link to a node affected by the perturbation may have to transmit a query, a reply, and an update in each direction. DUAL has the same complexity as DBF after a single link addition or link-cost reduction. To verify this, note that any node that receives a query reporting a distance decrease must be able to find a feasible successor. Accordingly, a node that sends a query after its distance decreases must receive immediate replies from all its neighbors, without those neighbors having to forward the query.

It is clear from the above results that DUAL's complexity is comparable to or better than the complexity of all previous distance-vector algorithms and comparable to the complexity of ILS. The only concern with DUAL's performance is when nodes fail or the network partitions because, in such cases, $x = N$.

B. Simulation Results

To obtain insight on the average performance of DUAL, DBF, and ILS in a real network, they were analyzed by simulation using the topologies of typical networks; SNC was used in DUAL. The simulation uses link weights of equal cost, zero link transmission delays, and the synchronous operation of DUAL described in the previous section. During each simulation step, a node 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 (node) in the network fail, and counts the steps and messages needed for each algorithm to recover. It then makes the same link (node) recover and repeats the process. The average is then taken over all link (node) failures and recoveries. The results of this simulation for MILNET are shown in Table I. The table shows the total number of events (updates and link-status changes processed by nodes), the total number of update messages transmitted, the total number of steps needed for the algorithms to converge, and the total number of operations performed by all the nodes in the network.

[1] The diameter of a network is the length of the longest shortest path in hops between any two of its nodes.

TABLE I
SIMULATION RESULTS FOR MILNET

Node Failure						
Parameter	DUAL		DBF		ILS	
	mean	*sdev*	*mean*	*sdev*	*mean*	*sdev*
event count	4500	4070	79300	51400	686	284
message count	1480	720	60500	335	680	282
steps	31.6	21.8	144	0	10.1	1.31
operation count	5370	4420	80200	51600	492000	204000
Node Recovery						
Parameter	DUAL		DBF		ILS	
	mean	*sdev*	*mean*	*sdev*	*mean*	*sdev*
event count	2390	1680	2850	1900	854	309
message count	644	166	743	259	848	307
steps	10.2	1.11	9.32	1.11	10.4	1.18
operation count	3260	2080	3720	2300	625000	227000
Link Failure						
Parameter	DUAL		DBF		ILS	
	mean	*sdev*	*mean*	*sdev*	*mean*	*sdev*
event count	2540	2120	11300	35200	506	90.2
message count	767	431	8010	20200	504	90.2
steps	20.3	10	25.2	45.5	9.92	1.26
operation count	2830	2120	11600	35200	366000	66000
Link Recovery						
Parameter	DUAL		DBF		ILS	
	mean	*sdev*	*mean*	*sdev*	*mean*	*sdev*
event count	1180	704	1200	698	510	90.8
message count	232	118	341	168	508	90.8
steps	8.71	1.67	8.17	1.61	9.36	1.05
operation count	1460	704	1480	698	371000	66000

The details of the simulation analysis appear elsewhere [27], [28]. However, it is worth noting that, as expected, DBF and DUAL have better overall average performance than ILS after the recovery of a single node or link. As it was also expected, DBF performs very poorly after a node failure because of the counting-to-infinity and looping problems. The simulation results of Table I also indicate that, insofar as overhead traffic and number of steps needed for convergence, the average performance of DUAL is comparable to the performance of ILS. In this respect, the only concern with DUAL is its performance after nodal failures; in this case, it requires almost twice as many messages and more than twice as many steps as ILS. On the other hand, the CPU utilization in ILS is two orders of magnitude larger than in DUAL. Accordingly, DUAL appears to be a more scalable solution for routing in large networks and internets than ILS.

VIII. SUMMARY AND CONCLUSIONS

The preceding sections described a new family of algorithms for distributed shortest-path routing called diffusing update algorithms or DUAL and proved that it provides loop-free paths at every instant regardless of the operational conditions of the network or internet. These results unify previous work on loop-free distance-vector algorithms by Jaffe and Moss and this author, and present a complete proof of loop freedom

for these types of algorithm for the first time in a journal publication.

The previous section showed that DUAL matches or outperforms previous loop-free distance-vector algorithms, and that a practical DVP based on DUAL can be implemented in a network or internet with a performance comparable to or better than that of an LSP.

The only concern regarding DUAL's performance is after node failures and network partitions, because in such cases all network nodes have to be involved in the same diffusing computation. Fortunately, the performance degradation of DUAL after node failures and network partitions can be easily dealt with in practice. A solution is to require that a node wait for a fixed "hold down" time to allow the node to receive updated information from any downstream neighbors that can be taken as feasible successors before the node is allowed to update its routing table. No hold-down time is needed when no feasible successors are found. The time complexity of DUAL with hold-down time is $TC = O(h)$, where $h < x$ is the length of the longest chain of the nodes participating in the diffusing computation. This is easily verified by noting that, with the synchrony assumption, a node that waits for a hold-down time must receive and process the update messages from all its downstream neighbors before it can send its own update message. In most networks and internets, a hold-down time proportional to the transmission delay incurred between two adjacent routers is enough to let a router near a destination receive updated information from all neighbors downstream before it is allowed to update its routing table. However, estimating an effective hold-down time depends on such factors as the topology of the network and the average link and processing delays experienced by control messages.

ACKNOWLEDGMENT

The author wishes to thank Z. S. Su for many fruitful discussions that helped in the design of DUAL, and W.T. Zaumen for all the work that went into DUAL's simulation.

REFERENCES

[1] K. M. Chandy and J. Misra, "Distributed computation on graphs: Shortest path algorithms," *Commun. ACM,* vol. 25, no. 11, pp. 833–837, Nov. 1982.
[2] R. Coltun, "OSPF: An internet routing protocol," *ConneXions,* vol. 3, no. 8, pp. 19–25, Aug. 1989.
[3] E. W. Dijkstra and C. S. Scholten, "Termination detection for diffusing computations," *Inform. Process. Lett.,* vol. 11, no. 1, pp. 1–4, Aug. 1980.
[4] L. R. Ford and D. R. Fulkerson, *Flows in Networks.* Princeton, NJ: Princeton Univ Press, 1962.
[5] J. J. Garcia-Luna-Aceves, "A distributed loop-free, shortest-path routing algorithm," *Proc. IEEE INFOCOM'88,* Mar. 1988.
[6] ——, "A minimum-hop routing algorithm based on distributed information," *Comput. Netw. and ISDN Syst.,* vol. 16, no. 5, pp. 367–382, May 1989.
[7] ——, "Loop-free internet routing and related issues," *ConneXions,* vol. 3, no. 8, pp. 8–18, Aug. 1989.
[8] ——, "A unified approach for loop-free routing using link states or distance vectors," *ACM Comput. Commun. Rev.,* vol. 19, no. 4, pp. 212–223, Sept. 1989.
[9] ——, "Diffusing update algorithms for message routing in computer networks and internetworks," Invention Disclosure P-3089, SRI Int., Menlo Park, CA, Sept. 1991.
[10] C. Hedrick, "Routing information protocol," RFC 1058, Netw. Inform. Cent., SRI Int., Menlo Park, CA, June 1988.
[11] R. Hinden and A. Sheltzer, "DARPA internet gateway," RFC 823, Netw. Inform. Cent., SRI Int., Menlo Park, CA, Sept. 1982.
[12] Int. Stand. Org., "Intra-domain IS–IS routing protocol," ISO/IEC JTC1/SC6 WG2 N323, Sept. 1989.
[13] J. M. Jaffe and F. M. Moss, "A responsive routing algorithm for computer networks," *IEEE Trans. Commun.,* vol. COM-30, no. 7, pp. 1758–1762, July 1982.
[14] M. J. Johnson, "Updating routing tables after resource failure in a distributed computer network," *Networks,* vol. 14, no. 3, pp. 379–392, 1984.
[15] ——, "Analysis of routing table update activity after resource recovery in a distributed computer network," in *Proc. Seventeenth Hawaii Int. Conf. Syst. Sci.,* Honolulu, HI, 1984, pp. 96–102.
[16] K. Lougheed and Y. Rekhter, "Border gateway protocol 3 (BGP-3)," RFC 1267, SRI Int., Menlo Park, CA, Oct. 1991.
[17] P. M. Merlin and A. Segall, "A failsafe distributed routing protocol," *IEEE Trans. Commun.,* vol. COM-27, no. 9, pp. 1280–1288, Sept. 1979.
[18] J. McQuillan, "Adaptive routing algorithms for distributed computer networks," BBN Rep. 2831, Bolt Beranek and Newman Inc., Cambridge, MA, May 1974.
[19] J. McQuillan *et al.,* "The new routing algorithm for the ARPANET," *IEEE Trans. Commun.,* vol. COM-28, May 1980.
[20] D. Mills, "Exterior gateway protocol formal specification," RFC 904, Netw. Inform. Cent., SRI Int., Menlo Park, CA, Dec. 1983.
[21] J. B. Postel, "Transmission Control Protocol," RFC 793, Netw. Inform. Cent., SRI Int., Menlo Park, CA, Sept. 1981.
[22] M. Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis.* Menlo Park, CA, Addison-Wesley, 1986, ch. 6.
[23] A. Segall, "Distributed network protocols," *IEEE Trans. Inform. Theory,* vol. IT-29, no. 1, pp. 23–35, Jan. 1983.
[24] J. Seeger and A. Khanna, "Reducing routing overhead in a growing DDN," in *Proc. MILCOM'86,* Monterey, CA, Oct. 1986, pp. 15.3.1–15.3.13.
[25] K. G. Shin and M. Chen, "Performance analysis of distributed routing strategies free of ping-pong-type looping," *IEEE Trans. Comput.,* vol. COMP-36, no. 2, pp. 129–137, Feb. 1987.
[26] W. D. Tajibnapis, "A correctness proof of a topology information maintenance protocol for a distributed computer network," *Commun. ACM,* vol. 20, pp. 477–485, 1977.
[27] W. Zaumen and J. J. Garcia-Luna-Aceves, "Dynamics of distributed shortest-path routing algorithms," *ACM Comput. Commun. Rev.,* vol. 21, no. 4, pp. 31–42, Sept. 1991.
[28] ——, "Dynamics of link-state and loop-free distance-vector routing algorithms," *J. Internetwork.,* vol. 3, pp. 161–188, 1992.

J. J. Garcia-Luna-Aceves 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 in the Department of Computer Engineering at the University of California, Santa Cruz. He is also Director of the Network Information Systems Center of SRI International, which he joined as an SRI International Fellow in 1982. His current research interests include the analysis and design of distributed network control algorithms and multimedia information systems. He has published more than 60 technical articles related to computer communication research. He has been Guest Editor of IEEE COMPUTER (1985) and the *ACM SIGCOMM Computer Communication Review* (1988), and is an Editor of the *Multimedia Systems Journal* (Springer International). He is General Chair of the first ACM conference on multimedia: ACM Multimedia '93. He was Program Chair of the IEEE Multimedia '92 Workshop, General Chair of the ACM SIGCOMM '88 Symposium, Program Chair of the ACM SIGCOMM Symposia of 1986 and 1987, and member of the technical program or organizing committee of numerous IEEE and ACM SIGCOMM conferences, IFIP 6.5 conferences, and the High Performance Distributed Computing (HPDC) symposia of 1992 and 1993. He is ACM SIGCOMM Conference Coordinator, and is a member of the steering committees for the ACM Multimedia Conference Series and the IEEE Multimedia '93 Workshop. He received the SRI Exceptional Achievement Award in 1985 for his work on multimedia communications, and again in 1989 for his work on distributed routing algorithms.

Dr. Garcia-Luna is a member of the ACM and a pioneer member of the Internet Society.