MPATH: A Loop-free Multipath Routing Algorithm

SRINIVAS VUTUKURY
vutukury@cse.ucsc.edu
Computer Sciences Department
University of California
Santa Cruz, CA 95064

J.J. GARCIA-LUNA-ACEVES
jj@cse.ucsc.edu
Computer Engineering Department
University of California
Santa Cruz, California 95064
Networking and Security Center
Sun Microsystems Laboratories
Palo Alto, California 94303

Abstract—**We present a distributed routing algorithm for computing multiple paths between each source-destination pair in a computer network, such that the paths are loop-free at all times and are not necessarily of equal length. In this algorithm, routers exchange second-to-last hop on the shortest path to destinations in addition to shortest distances, which are used to prevent the well-know count-to-infinity problem. The safety and liveness properties of the algorithm are proved and its performance is analyzed.**

I. INTRODUCTION

RIP[8] and many other routing protocols based on the distributed Bellman-Ford algorithm (DBF) for shortest-path computation suffer from the *bouncing effect* and the *counting-to-infinity* problems, which limit their applicability to small networks using hop count as the measure of distance. In the past several years, much research has been devoted to fixing these problems. In one approach, routers exchange query and reply messages to synchronize distance updates, a technique that is sometimes called *diffusing computations* [1]. The loop-free routing algorithm DUAL[3], which is used in EIGRP [2], and several algorithms based on distance vectors have been proposed that use diffusing computations to overcome the counting-to-infinity problem of DBF [15], [11], [10], [20].

In another approach, routers exchange second-to-last hop to a destination in addition to distance information so that they can determine complete paths and prevent the count-to-infinity problem. These algorithms are often called path-finding algorithms or source-tracing algorithms [9], [14], [5]. All these algorithms eliminate DBF's counting to infinity problem, and some of them (LPA [5]) are more efficient than any of the routing algorithms based on link-state information proposed to date. Furthermore, LPA [5] is loop-free at every instant. The MPATH routing algorithm presented in this paper is a path-finding algorithm [17], [18]. MPATH differs from prior path-finding algorithms in that it uses the invariants, introduced in [19], to ensure multiple loop-free paths of unequal cost.

Another family of routing algorithms exchange link information to compute routing paths. These algorithms were first proposed and widely used because they do not suffer from the count-to-infinity problem of the distance vector algorithms. OSPF [12] and algorithms in [16], [13] are some that belong to this family, which exchange complete topology information. A couple of routing algorithms have been proposed that operate using partial topology information [4], [6], [7], [19] to eliminate the main limitation of topology-broadcast algorithms.

Except DASM[20] and MPDA[19], all of the above routing algorithms focus on the provision of a single path to each destination. A drawback of DASM is that it uses multi-hop synchronization, which can limit its scalability. In this paper, we present the first path-finding routing algorithm that (a) provides multiple paths of unequal cost to each destination that are free of loops at every instant and (b) uses a synchronization mechanism that spans only one hop, which makes it more scalable than routing algorithms based on diffusing computations spanning multiple hops.

The paper is organized as follows. Section II describes MPATH. Section III presents the correctness proofs showing that MPATH is loop-free at every instant, safe, and live. Section IV analyzes the complexity of MPATH. Section V provides concluding remarks.

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grants F30602-97-1-0291 and F19628-96-C-0038.

II. DISTRIBUTED MULTIPATH ROUTING ALGORITHM

A. Problem Formulation

A computer network is represented as a graph $G = (N, L)$ where N is set of nodes (routers) and L is the set of edges (links) connecting the nodes. A cost is associated with each link and can change over time, but is always positive. Two nodes connected by a link are called adjacent nodes or neighbors. The set of all neighbors of a given node i is denoted by N^i. Adjacent nodes communicate with each other using messages and messages transmitted over an operational link are received with no errors, in the proper sequence, and within a finite time. Furthermore, such messages are processed by the receiving node one at a time in the order received. A node detects the failure, recovery and link cost changes of each adjacent link within a finite time.

The goal of our distributed routing algorithm is to determine at each node i the successor set of i for destination j, which we denote by $S_j^i(t) \subseteq N^i$, such that the routing graph $SG_j(t)$ consisting of link set $\{(m,n)|n \in S_j^m(t),\ m \in N\}$ is free of loops at every instant t, even when link costs are changing with time. The routing graph $SG_j(t)$ for single-path routing is a sink-tree rooted at j, because the successor sets $S_j^i(t)$ have at most one member. In multipath routing, there can be more than one member in $S_j^i(t)$; therefore, $SG_j(t)$ is a directed acyclic graph with j as the sink node. There are potentially several $SG_j(t)$ for each destination j; however, the routing graph we will construct is defined by the successor sets $S_j^i(t) = \{k|D_j^k(t) < D_j^i(t), k \in N^i\}$, where D_j^i is the shortest distance of node i to destination j. We call such a routing graph the *shortest multipath* for destination j. After a series of link cost changes which leave the network topology in arbitrary configuration, the distributed routing algorithm should work to modify SG_j in such a way that it eventually converges to the shortest multipath of the new configuration, without ever creating a loop in SG_j during the process.

Therefore, our solution to the routing problem consists of first computing D_j^i using a shortest-path routing algorithm, and using it to compute S_j^i. Because D_j^k is node k's local variable, its value has to be explicitly or implicitly communicated to i. If D_{jk}^i is the value of D_j^k as known to node i, the problem now becomes one of computing $S_j^i(t) = \{k|D_{jk}^i(t) < D_j^i(t)\}$. However, because of non-zero propagation delays, during network transitions there can be discrepancies in the value of D_j^k and its copy D_{jk}^i at i, which may cause loops to form in SG_j. To prevent loops, therefore, additional constraints must be imposed when computing S_j^i. We show later that if the successor set

Procedure *INIT-PATH*

{*Invoked when the node comes up.*}

1. Initialize all tables.
2. Run $PATH$ algorithm.

End *INIT-PATH*

Algorithm *PATH*

{*Invoked when a message M is received from neighbor k, or an adjacent link to k has changed cost or when a node is initialized.*}

1. Run NTU to update neighbor tables.
2. Run MTU to update main tables.
3. For each destination j marked as *changed*,
   Add update entry [j, D^i_j, p^i_j] to the new message M'.
4. Within finite amount of time, send message M' to each neighbor.

End *PATH*

Fig. 1. The PATH Algorithm

at each node i for each destination j satisfy certain conditions called loop-free invariant conditions, then the snapshot at time t of the routing graph $SG_j(t)$ implied by $S^i_j(t)$ is free of loops.

B. Node Tables and Message Structures

As in DBF, nodes executing MPATH exchange messages containing distances to destinations. In addition to the distance to a destination, nodes also exchange the identity of the second-to-last node, also called predecessor node, which is the node just before the destination node on the shortest path. In this respect MPATH is akin to several prior algorithms [5], [14], [9], but differs in its specification, verification and analysis and, more importantly, in the multipath operation described in the next section.

The following information is maintained at each node i:

1. The *Main Distance Table* contains D^i_j and p^i_j, where D^i_j is the distance of node i to destination j and p^i_j is the predecessor to destination j on the shortest path from i to j. The table also stores for each destination j, the successor set S^i_j, feasible distance FD^i_j, reported distance RD^i_j and two flags *changed* and *report-it*.
2. The *Main Link Table* T^i is the node's view of the network and contains links represented by $(m,\ n,\ d)$ where $(m,\ n)$ is a link with cost d.
3. The *Neighbor Distance Table* for neighbor k contains D^i_{jk} and p^i_{jk} where D^i_{jk} is the distance of neighbor k to j as communicated by k and p^i_{jk} is the predecessor to j on the shortest path from k to j as notified by k.
4. The *Neighbor Link Table* T^i_k is the neighbor k's view of the network as known to i and contains link information derived from the distance and predecessor information in the neighbor distance table.
5. *Adjacent Link Table* stores the cost l^i_k of adjacent link to each neighbor k. If a link is down its cost is infinity.

Nodes exchange information using update messages which have the following format.

1. An update message can one or more update entries. An update entry is a triplet [j, d, p], where d is the distance of the node sending the message to destination j and p is the predecessor on the path to j.
2. Each message carries two flags used for synchronization: *query* and *reply*.

Procedure NTU

{*Called by PATH to process an event.*}

1. If event is a message M from neighbor k,
   a. For each entry [j, d, p] in M //Note $d = D^k_j$, $p = p^k_j$.)
      Set $D^i_{jk} \leftarrow d$ and $p^i_{jk} \leftarrow p$.
   b. For each destination j with an entry in M,
      Remove existing links (n, j) in T^i_k and add new link (m, j, d) to T^i_k, where $d = D^i_{jk} - D^i_{mk}$ and $m = p^i_{jk}$.
2. If link-down event, clear the neighbor tables of k;
3. If link-up or link-cost change event,update l^i_k;

End NTU

Fig. 2. Neighbor Table Update Algorithm

C. Computing D^i_j

This subsection describes the shortest-path algorithm PATH (1) and the next subsection describes how PATH is extended to compute multiple next-hops are determined based on D^i_j. INIT-PATH is called at node startup to initialize the tables; distances are initialized to infinity and node identities to a null value. PATH is executed in response to an event: either a receipt of an update message from a neighbor or detection of an adjacent link cost or link status (up/down) change. PATH invokes procedure NTU (Fig. 2), which first updates the neighbor distance tables and then updates T^i_k with links $(m,\ n,\ d)$, where $d = D^i_{nk} - D^i_{mk}$ and $m = p^i_{nk}$. PATH then invokes procedure MTU (Fig. 3), which constructs T^i by merging the topologies T^i_k and the adjacent links l^i_k. PATH is virtually identical to PDA [19], in the sense that though they differ in the information the routers exchange, internally they construct the partial topologies.

The merging process is straightforward if all neighbor topologies (T^i_k) contain consistent link information, but when two or more neighbors link tables contain conflicting information regarding a particular link, the conflict must be resolved. Two neighbor tables are said to contain conflicting information regarding a link, if either both report the link with different cost or one reports the link and the other does not. Conflicts are resolved as follows: if two or more neighbor tables contain conflicting information of link $(m,\ n)$, then T^i is updated with link information reported by the neighbor k that offers the shortest distance from the node i to the head node m of the link, i.e., $l^i_k + D^i_{mk} = min\{l^i_k + D^i_{mk} | k \in N^i\}$. Ties are broken in a *consistent* manner; one way is to break ties always in favor of lower address neighbor. Because i itself is the head of the link for adjacent links, any information about an adjacent link supplied by neighbors will be overridden by the most current information about the link available to node i.

After merging the topologies, MTU runs Dijkstra's shortest path algorithm to find the shortest path tree and deletes all links from T^i that are not in the tree. Because there can be more than one shortest-path tree, while running Dijkstra's algorithm ties are again broken in a consistent manner. The distances D^i_j and predecessors p^i_j are then obtained from T^i. The tree is compared with the previous shortest path tree and only the differences are then reported to the neighbors. If there are no differences, no updates are reported. Eventually all tables converge such that D^i_j give the shortest distances and all message activity will cease.

D. Computing S^i_j

MPATH described in this section determines the successor sets S^i_j, by enforcing the Loop-free Invariant conditions (described below) using a neighbor-to-neighbor synchronization.

Let FD^i_j, called the feasible distance, be an 'estimate' of the dis-

Procedure MTU

1. Clear link table T^i.
2. For each node $j \neq i$ occurring in at least one T_k^i,
   a. Find $MIN \leftarrow min\{D_{jk}^i + l_k^i | k \in N^i\}$.
   b. Let n be such that $MIN = (D_{jn}^i + l_n^i)$. (Ties are broken *consistently*. Neighbor n is the preferred neighbor for destination j). For each link (j, v, d) in T_n^i, Add link (j, v, d) to T^i.
3. Update T^i with each link l_k^i.
4. Run Dijkstra's shortest path algorithm on T^i to find new D_j^i, and p_j^i.
5. For each destination j, if D_j^i or p_j^i changed from previous value, set *changed* and *report-it* flags for j.

End MTU

Fig. 3. Main Table Update Algorithm

tance of node i to node j in the sense that FD_j^i is equal to D_j^i when the network is in stable state, but to prevent loops during periods of network transitions, it is allowed to be temporarily differ from D_j^i. The key to loop-free routing is in maintaining FD_j^i such that the following conditions are satisfied.

Loop-free Invariant Conditions(LFI)[19]:

$$FD_j^i(t) \leq D_{ji}^k(t) \qquad k \in N^i \tag{1}$$

$$S_j^i(t) = \{\, k \mid D_{jk}^i(t) < FD_j^i(t)\} \tag{2}$$

The invariant conditions (1) and (2) state that, for each destination j, a node i can choose a successor whose distance to j, as known to i, is less than the distance of node i to j that is known to its neighbors.

Theorem *1:* [19] If the LFI conditions are satisfied at any time t, the $SG_j(t)$ implied by the successor sets $S_j^i(t)$ is loop-free.

Proof: Let $k \in S_j^i(t)$ then from (2) we have

$$D_{jk}^i(t) < FD_j^i(t) \tag{3}$$

At node k, because node i is a neighbor, from (1) we have

$$FD_j^k(t) \leq D_{jk}^i(t) \tag{4}$$

Combining (3) and (4) we get

$$FD_j^k(t) < FD_j^i(t) \tag{5}$$

Eq.(5) states that, if k is a successor of node i in a path to destination j, then k's feasible distance to j is strictly less than the feasible distance of node i to j. Now, if the successor sets define a loop at time t with respect to j, then for some node p on the loop, we arrive at the absurd relation $FD_j^p(t) < FD_j^p(t)$. Therefore the LFI conditions are sufficient for loop-freedom. ■

The invariants used in LFI are independent of whether the algorithm uses link states or distance vectors; in link-state algorithms, such as MPDA, the D_{jk}^i are computed locally from the link-states communicated by the neighbors while in distance-vector algorithms, like the MPATH presented here, the D_{jk}^i are directly communicated.

The invariants (1) and (2) suggest a technique for computing $S_j^i(t)$ such that the successor graph $SG_j(t)$ for destination j is loop-free at every instant. The key is determining $FD_j^i(t)$ in Eq. (1), which requires

Procedure *INIT-MPATH*

{*Invoked when the node comes up.*}

1. Initialize tables and run MPATH.

End *INIT-MPATH*

Algorithm $MPATH$

{*Invoked when a message M is received from neighbor k, or an adjacent link to k has changed.*}

1. Run NTU to update neighbor tables.
2. Run MTU to obtain new D_j^i and p_j^i.
3. If node is *PASSIVE* or node is *ACTIVE* $\wedge$ last reply arrived,
   Reset *goactive* flag.
   For each destination j marked as *report-it*,
   a. $FD_j^i \leftarrow min\{D_j^i, RD_j^i\}$
   b. If $D_j^i > RD_j^i$, Set *goactive* flag.
   c. $RD_j^i \leftarrow D_j^i$
   d. Add [j, RD_j^i, p_j^i] to message M'.
   e. Clear *report-it* flag for j.

   Otherwise, the node is ACTIVE and waiting for more replies,
   For each destination j marked as *changed*,
   f. $FD_j^i \leftarrow min\{D_j^i, FD_j^i\}$
4. For each destination j marked as *changed*,
   a. Clear *changed* flag for j
   b. $S_j^i \leftarrow \{k | D_{jk}^i < FD_j^i\}$
5. For each neighbor k,
   a. $M'' \leftarrow M'$.
   b. If event is a *query* from k, Set *reply* flag in M''.
   c. If *goactive* set, Set *query* flag in M''.
   d. If M'' non-empty, send M'' to k.
6. If *goactive* set, become $ACTIVE$, otherwise become $PASSIVE$.

End $MPATH$

Fig. 4. Multi-path Loop-free Routing Algorithm

node i to know $D_{ji}^k(t)$, the distance from i to node j in the topology table T_i^k that node i communicated to neighbor k. Because of non-zero propagation delay, T_i^k is a time-delayed version of T^i. We observe that, if node i delays updating of FD_j^i with D_j^i until k incorporates the distance D_j^i in its tables, then FD_j^i satisfies the LFI condition.

MPATH (Fig. 4) enforces the LFI conditions by synchronizing the exchange of update messages among neighbors using *query* and *reply* flags. If a node sends a message with a *query* bit set, then the node must wait until a *reply* is received from all its neighbors before the node is allowed to send the next update message. The node is said to be in ACTIVE state during this period. The inter-neighbor synchronization used in MPATH spans only one hop, unlike algorithms that use diffusing computation that potentially span the whole network(e.g., DASM [20]).

Assume that all nodes are in PASSIVE state initially with correct distances to all other nodes and that no messages are in transit or pending to be processed. The behavior of the network, where every node runs MPATH, is such that when a finite sequence of link cost changes occurs in the network within a finite time interval, some or all nodes go through a series of PASSIVE-to-ACTIVE and ACTIVE-to-PASSIVE state transitions, until eventually all nodes become PASSIVE with correct distances to all destinations.

Let a node in PASSIVE state receive an event that results in changes in its distances to some destinations. Before the node sends an update message to report new distances, it checks if the distance D_j^i to any destination j has increased above the previously reported distance RD_j^i. If none of the distances increased, then the node remains in PASSIVE state. Otherwise, the node sets the *query* flag in the update

message, sends it to each neighbor, and goes into ACTIVE state. When in ACTIVE state, a node cannot send any update messages or increase FD_j^i. After receiving replies from all its neighbors the node is allowed to increase FD_j^i and report any changes that may have occurred since the time it has transitioned to ACTIVE state, and if none of the distances increased beyond the reported distance, the node transitions to PASSIVE state. Otherwise, the node sends the next update message with the *query* bit set and becomes ACTIVE again, and the whole cycle repeats. If a node receives a message with the *query* bit set when in PASSIVE state, it first modifies its tables and then sends back an update message with the *reply* flag set. Otherwise, if the node happens to be in ACTIVE state, it modifies the tables but because the node is not allowed to send updates when in ACTIVE state, the node sends back an empty message with no update information and the *reply* bit set. If a reply from a neighbor is pending when the link to the neighbor fails then an implicit reply with infinite distance is assumed, Because replies are given immediately to queries and replies are assumed to be given upon link failure, deadlocks due to inter-neighbor synchronization cannot occur. Eventually, all nodes become PASSIVE with correct distances to destinations, which we prove in the next section.

III. Correctness of MPATH

To show the correctness of MPATH, we prove the following: (1) MPATH eventually converges with D_j^i giving the shortest distances and (2) the successor graph SG_j is loop-free at every instant and eventually converges to the shortest multipath. PATH works essentially like PDA[19] except that the kind of update information exchanged is different; PDA exchanges link-state while PATH exchanges distance-vectors with predecessor information. Internally both represent this information as partial topologies communicated by the neighbors. So, the correctness proof of PATH is identical to PDA. The convergence of MPATH directly follows from the convergence of PATH because extensions to MPATH are such that update messages in MPATH are only delayed a finite amount of time. A node generates update messages only to report changes in distances and predecessor, so after convergence no messages will be generated. The following theorems show that MPATH provides instantaneous loop-freedom.

Theorem *2:* For the algorithm MPATH executed at node i, let t_n be the time when RD_j^i is updated and reported for the n-th time. Then, the following conditions always hold.

$$FD_j^i(t_n) \leq min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \quad (6)$$

$$FD_j^i(t) \leq FD_j^i(t_n) \qquad t \in [t_n, t_{n+1}) \quad (7)$$

Proof: From the working of MPATH in Fig. 4, we observe that RD_j^i is updated at line 3c when (a) the node goes from PASSIVE-to-ACTIVE because of one or more distance increases (b) the node receives the last reply and goes from ACTIVE-to-PASSIVE state (c) the node is in PASSIVE state and remains in PASSIVE state because the distance did not increase for any destination (d) the node receives the last reply but immediately goes into ACTIVE state. The reported distance RD_j^i remains unchanged during the ACTIVE phase. Because FD_j^i is updated at line 3a each time RD_j^i is updated at line 3c, Eq. (6) follows. When the node is in ACTIVE phase, FD_j^i may also be modified by the statement on line 3f, which implies Eq. (7). ■

Theorem *3:* (Safety property) At any time t, the successor sets $S_j^i(t)$ computed by MPATH are loop-free.

Proof: The proof is based on showing that the FD_j^i and S_j^i computed by MPATH satisfy the LFI conditions. Let t_n be the time when RD_j^i is updated and reported for the n-th time. The proof is by induction on the interval $[t_n, t_{n+1}]$. Let the LFI condition be true up to time t_n, we show that

$$FD_j^i(t) \leq D_{ji}^k(t) \qquad t \in [t_n, t_{n+1}] \quad (8)$$

From Theorem 2 we have

$$FD_j^i(t_n) \leq min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \quad (9)$$

$$FD_j^i(t_{n+1}) \leq min\{RD_j^i(t_n), RD_j^i(t_{n+1})\} \quad (10)$$

$$FD_j^i(t) \leq FD_j^i(t_n) \qquad t \in [t_n, t_{n+1}) \quad (11)$$

Combining the above equations we get

$$FD_j^i(t) \leq min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \quad t \in [t_n, t_{n+1}] \quad (12)$$

Let t' be the time when message sent by i at t_n is received and processed by neighbor k. Because of the non-zero propagation delay across any link, t' is such that $t_n < t' < t_{n+1}$ and because RD_j^i is modified at t_n and remains unchanged in (t_n, t_{n+1}) we get

$$RD_j^i(t_{n-1}) \leq D_{ji}^k(t) \qquad t \in [t_n, t') \quad (13)$$

$$RD_j^i(t_n) \leq D_{ji}^k(t) \qquad t \in [t', t_{n+1}] \quad (14)$$

From Eq. (13) and (14) we get

$$min\{RD_j^i(t_{n-1}), RD_j^i(t_n)\} \leq D_{ji}^k(t) \quad t \in [t_n, t_{n+1}] \quad (15)$$

From (12) and (15) the inductive step (8) follows. Because $FD_j^i(t_0) \leq D_{ji}^k(t_0)$ at initialization, from induction we have that $FD_j^i(t) \leq D_{ji}^k(t)$ for all t. Given that the successor sets are computed based on FD_j^i, it follows that the LFI conditions are always satisfied. According to the Theorem 1 this implies that the successor graph SG_j is always loop-free. ■

The following theorem shows that MPATH correctly computes the shortest multipath.

Theorem *4:* (Liveness property) A finite time after the last change in the network, the D_j^i give the correct shortest distances and $S_j^i = \{k | D_j^k < D_j^i, k \in N^i\}$.

Proof: The proof is similar to the proof of Theorem 4 in [19]. The convergence of MPATH follows directly from the convergence of PATH because the update messages in MPATH are only delayed a finite time as allowed at line 4 in algorithm PATH. Therefore, the distances D_j^i in MPATH also converge to shortest distances. Because changes to D_j^i are always reported to the neighbors and are incorporated by the neighbors in their tables in finite time $D_{jk}^i = D_j^k$ for $k \in N^i$ after convergence. From line 3a in MPATH, we observe that when node i becomes passive $FD_j^i = D_j^i$ holds true. Because all nodes are passive at convergence it follows that $S_j^i = \{k | D_{jk}^i < FD_j^i, k \in N^i\} = \{k | D_j^k < D_j^i, k \in N^i\}$. ■

IV. Simulation results

The simulations compare the control overhead and convergence times of MPATH, TOPB and DASM. TOPB is a link-state algorithm that closely approximates OSPF, which is a link-state algorithm for which commercial implementations exist and whose convergence time is fairly constant and depends on the diameter of the network. Ideally, MPATH should approach the convergence times of TOPB, that is the extra time needed to enforce loop-freedom should be negligible. We expect MPATH to have far less message overhead because of its reliance on only partial topology. On the other hand DASM is the only distance-vector routing algorithm to date that provides loop-free multipaths to each destination. DASM achieves loop-freedom through diffusing computations that span the whole network. In contrast, MPATH

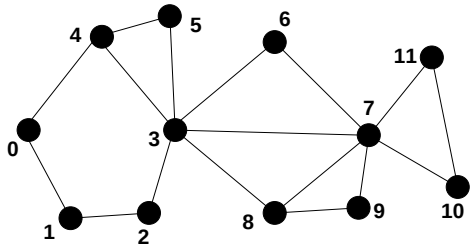

Fig. 5. CAIRN Topology used in simulations

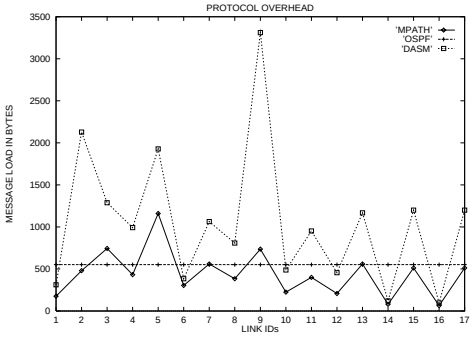

Fig. 6. Link failures. Message overhead

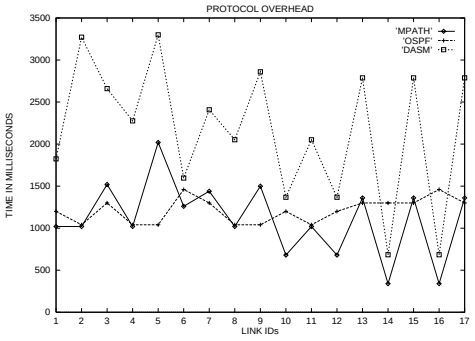

Fig. 7. Link failures. Convergence times

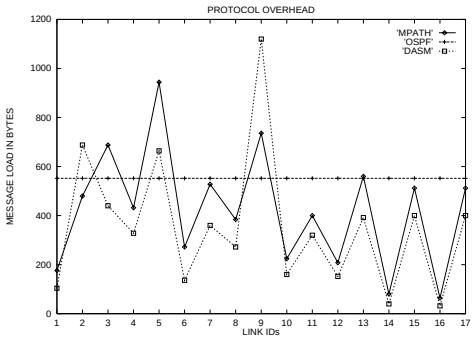

Fig. 8. Link recoveries. Message overhead

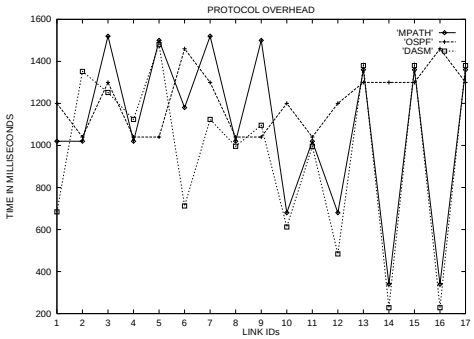

Fig. 9. Link recoveries. Convergence times

uses only neighbor-to-neighbor synchronization. It is interesting to see how convergence times are effected by the synchronization mechanisms. Also, it is not obvious how the control message overheads of DASM and MPATH compare.

The performance metrics used for comparison are the control message overhead and the convergence times. We use the event-driven real-time simulator CPT from Nokia and perform simulations on the CAIRN topology shown in Fig. 5 (www.cairn.net). For simplicity, we use a flat topology without area aggregation; there is no reason to believe area aggregation would favor one routing algorithm over others.

Two types of events are triggered in the network: link-status changes (link failures and link recovery) and link-cost changes. In practice links and nodes are highly reliable and change status much less frequently than link costs which are a function of the traffic on the link. For simplicity, We do not simulate node failures because of the problems resulting due to loss of sequence numbers by the nodes, which only effect the functioning of TOPB here.

We also restrict link-status changes to a single change; that is, only one link failure or link recovery can occur at any time during the measurement interval. Because in backbone networks the links and nodes in the network are highly reliable, simultaneous multiple topological changes are much less likely to occur and it is reasonable to assume that tables converge between topological changes. However, link costs of multiple links can change simultaneously and repeatedly before the tables converge to the latest costs. This is the case when near-optimal delay routing of [19] is used, in which the link costs are periodically measured and reported. For these reasons, we simulate only single link-status changes and multiple link-cost changes.

Link-status changes: Each link is made to fail and recover in turn, and the control message overhead and convergence times are measured in each case. The worst-case and the averages of control message overhead and convergence times are given in Table 1. Figs. (6)-(9) show the overheads associated with each event. For link failures and recoveries MPATH has lower average message overhead than TOPB, which is due to use of partial topologies. However, due to synchronization used for providing loop-freedom, the worst-case message overhead is higher for MPATH.. MPATH has larger overhead than DASM under link recoveries because, though neither invokes synchronization, MPATH exchanges predecessor information in addition to distances. Under link-failures, DASM requires more messages than MPATH because of the multihop synchronization that DASM uses. Same argument can be applied for the convergence times.

Multiple link-cost changes: When near-optimal routing framework is implemented as in [19], multiple links change cost simultaneously. To study the protocol behavior under such scenarios, costs of multiple links is changed simultaneously and the performance is measured. The average message overhead and convergence times are shown in the Table 1. MPATH has lower worst-case and average message overhead than TOPB and DASM. MPATH has lower worst-case and average convergence times than DASM. The average convergence time for MPATH is also lower than TOPB. Only in the worst-case, MPATH showed higher convergence times than TOPB, which is again due to synchronization used in MPATH.

V. Conclusions

We have presented the first path-finding routing algorithm that provides multiple paths between each source-destination pair that need not necessarily have equal costs and that are loop-free at every instant. The

TABLE 1

Control messages (bytes)			
	Worst-case	Avg	Std-dev
Link failures			
TOPB	555.00	555.00	0.00
DASM	3312.00	1052.70	792.19
MPATH	1160.00	443.29	266.06
Link recoveries			
TOPB	552	552	552
DASM	1120.0	353.41	266.43
MPATH	944	423.52	230.95
Link-cost changes			
TOPB	9384.00	9384.00	0.00
DASM	11520.00	10050.93	742.10
MPATH	6856.00	5272.53	702.51
Convergence times (ms)			
	Worst-case	Avg	Std-dev
Link failures			
TOPB	1.46	1.20	0.14
DASM	3.30	2.16	0.78
MPATH	2.02	1.11	0.42
Link recoveries			
TOPB	1.46	1.20	0.14
DASM	1.48	0.97	0.39
MPATH	1.52	1.08	0.37
Link-cost changes			
TOPB	5.48	5.48	0.00
DASM	9.82	7.75	0.71
MPATH	6.46	4.87	0.77

routing algorithm is designed around a set of loop-free invariant conditions and uses inter-nodal synchronization that spans no more than one hop. Using simulations, the performance of the routing algorithm, in terms of control message overhead and convergence times, is compared with other algorithms. The multiple next-hop choices that MPATH makes available at each node can be used for traffic load-balancing and minimizing delays in the network [19].

REFERENCES

[1] E.W.Dijkstra and C.S.Scholten. Termination Detection for Diffusing Computations. *Information Processing Letters*, 11:1–4, August 1980.

[2] D. Farinachi. Introduction to enhanced IGRP(EIGRP). *Cisco Systems Inc.*, July 1993.

[3] J.J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Trans. Networking*, 1:130–141, February 1993.

[4] J.J. Garcia-Luna-Aceves and J. Behrens. Distributed, scalable routing based on vectors of link states. *IEEE Journal on Selected Areas in Communications*, October 1995.

[5] J.J. Garcia-Luna-Aceves and S. Murthy. A path-finding algorithm for loop-free routing. *IEEE/ACM Trans. Networking*, February 1997.

[6] J.J. Garica-Luna-Aceves and M. Spohn. Scalable link-state internet routing. *Proc. International Conference on Network Protocols*, October 1998.

[7] J.J. Garica-Luna-Aceves and M. Spohn. Source tree adaptive routing. *Proc. International Conference on Network Protocols*, October 1999.

[8] C. Hendrick. Routing Information Protocol. *RFC*, 1058, june 1988.

[9] P. A. Humblet. Another Adaptive Distributed Shortest Path Algorithm. *IEEE Trans. Commun.*, 39:995–1003, June 91.

[10] J. M. Jaffe and F. H. Moss. A Responsive Distributed Routing Algorithm for Computer Networks. *IEEE Trans. Commun.*, 30:1758–1762, July 1982.

[11] P. M. Merlin and A. Segall. A Failsafe Distributed Routing Protocol. *IEEE Trans. Commun.*, 27:1280–1287, September 1979.

[12] J. Moy. OSPF Version 2. *RFC*, 1247, August 1991.

[13] R. Perlman. Fault-tolerant broadcast of routing information. *Computer Networks and ISDN*, 7, 1983.

[14] B. Rajagopalan and M. Faiman. A Responsive Distributed Shortest-Path Routing Algorithm with Autonomous Systems. *Internetworking: Research and Experience*, 2:51–69, March 1991.

[15] A. Segall. Optimal distributed routing for virtual line-switched data networks. *IEEE Trans. Commun.*, 27:201–209, January 1979.

[16] J. Spinelli and R. Gallager. Event Driven Topology Broadcast without Sequence Numbers. *IEEE Trans. Commun.*, 37:468–474, 1989.

[17] S. Vutukury and J.J. Garcia-Luna-Aceves. An algorithm for multipath computation using distance-vectors with predecessor information. *Proc. of ICCCN*, Oct. 1999.

[18] S. Vutukury and J.J. Garcia-Luna-Aceves. A Distributed Algorithm for Multipath Computation. *GLOBECOM'99*, 1999.

[19] S. Vutukury and J.J. Garcia-Luna-Aceves. A Simple Approximation to Minimum Delay Routing. *Proc. of ACM SIGCOMM*, Sept. 1999.

[20] W. T. Zaumen and J.J. Garcia-Luna-Aceves. Loop-Free Multipath Routing Using Generalized Diffusing Computations. *Proc. IEEE INFOCOM*, March 1998.