MDVA: A Distance-Vector Multipath Routing Protocol

Srinivas Vutukury, J.J. Garcia-Luna-Aceves

Abstract—**Routing protocols using the Distributed Bellman-Ford (DBF) algorithm converge very slowly to the correct routes when link costs increase, and in the case when a set of link failures results in a network partition, DBF simply fails to converge, a problem which is commonly referred to as the count-to-infinity problem. In this paper, we present the first distance vector routing algorithm MDVA that uses a set of loop-free invariants to prevent the count-to-infinity problem. MDVA, in addition, computes multipaths that are loop-free at every instant. In our earlier work we shows how such loop-free multipaths can be used in traffic load-balancing and minimizing delays, which otherwise are impossible to perform in current single-path routing algorithms [15].**

I. Introduction

Routing protocols construct tables at each node that specify for each destination the next-hop to use for data packet forwarding. It is required that the routing tables computed by them be free of loops when the network is stable. In dynamic environments, a more stringent requirement is that the routing tables be loop-free not only when network is stable but at *every instant* because, loops even if temporary can rapidly degrade performance. In our recent work [15], we described a load-balancing routing framework to obtain "near-optimal" delays, a key component of which is a fast responsive routing protocol that determines multiple successor choices for packet forwarding such that the routing graphs implied by the routing tables are free of loops even during network transitions. By load-balancing traffic over these multiple next-hop choices, congestion and delays can be significantly reduced. Our goal in this paper is therefore to develop a distance-vector routing algorithm that is suitable for implementing near-optimal routing as described in [15].

Though routing is a very old problem in computer networks, most of the solutions to date are unsuitable for load-balancing and implementing the near-optimal framework mentioned above. The widely deployed routing protocol RIP provides only one next-hop choice for each destination and does not prevent temporary loops from forming. Cisco's EIGRP [1] ensures instantaneous loop-freedom but can provide only a single loop-free path to each destination at any given router. The link-state protocol OSPF offers a router multiple choices for packet-forwarding only when those choices offer the minimum distance. When there is fine granularity in link costs metric, as in the case of optimal routing, there is less likelihood that multiple paths with equal distance exist between each source-destination pair, which means the full connectivity of the network is still not used for load-balancing. Also, OSPF and other algorithms based on topology-broadcast (e.g., [13], [10]) incur too much communication overhead when link costs change frequently. Also they do not provide instantaneous loop-freedom which is desirable especially when on-line link-cost measurement is used.

Several routing algorithms based on distance vectors have been proposed in the literature ([7], [8], [9], [11], [16] to name a few). However, with the exception of DASM[16], all of them are single-path algorithms. A few routing algorithms have been proposed that use partial topology information (refer [6], [12] and the references therein) to eliminate the main limitation of topology-broadcast algorithms; however, these algorithms are not loop-free at every instant. Recently, we introduced MPDA [15], which is the first routing algorithm based on link-state information that construct multipaths to each destination that are loop-free at every instant. In this paper, we present a new routing algorithm MDVA (Multipath Distance-Vector Algorithm), which is the first distance vector algorithm that uses the loop-free invariants introduced in [15], solves the count-to-infinity problem and computes multipaths to destinations. We provide formal proofs for the safety and liveness properties of MDVA, and compare its performance to other routing algorithms through simulations.

The paper is organized as follows. In Section II we discuss the main convergence problem facing a typical distance-vector algorithm and outline a solution that addresses those problems. Section III describes MDVA and Section IV provides the correctness proof for the algorithm. A performance comparison through simulations is provided in Section V. Section VI concludes the paper.

II. Overview of the Approach

A. Problem Formulation

A computer network is modelled as a graph $G = (N, L)$, where N is set of nodes (routers) and L is the set of edges (links). Let N^i be the set of neighbors of node i. The problem consists of finding the successor set at each router i for each destination j, denoted by $S_j^i \subseteq N^i$, so that when router i receives a packet for destination j, it can forward it to one of the neighbor routers in the successor set S_j^i. By repeating this process at every router, the packet is expected to reach the destination. If the routing graph SG_j, a directed subgraph of G, is defined by the directed link set $\{(m,n)|n \in S_j^m,\ m \in N\}$, a packet destined for j follows a path in SG_j. Two properties determine the efficiency of the routing graph constructed by the protocol: *loop-freedom* and *connectivity*. It is required that SG_j be free of loops, at least when the network is stable, because routing loops degrade network performance. In a dynamic environment, it is desirable that SG_j be loop-free at *every instant*, i.e., if S_j^i and SG_j are parameterized by time t, then $SG_j(t)$ should be free of loops at any time t. Observe that if there is at most one element in each S_j^i, then SG_j is a tree and there is only one path from

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grants F30602-97-1-0291 and N66001-00-1-8942.

Srinivas Vutukury is with the Computer Sciences Department and J.J. Garcia-Luna-Aceves is with the Computer Engineering Department at University of California, Santa Cruz, USA.

TABLE I
NOTATION

N	Set of nodes in the network
N^i	Set of neighbors of node i
S_j^i	Next-hop choices at i for destination j
SG_j	Routing graph implied by the S_j^i of destination j
D_j^i	Distance of node i to j as known to i
l_k^i	Cost of link (i,k)
D_{jk}^i	Distance of node k to j as reported by k to i
FD_j^i	Feasible distance is an estimate of D_j^i
RD_j^i	Distance to j reported by node i to its neighbors
SD_j^i	Best distance to j through S_j^i
WN_j^i	Set of neighbors that are waiting for replies
$\mathbf{G}(t)$	A bird's view of the network at time t
$\mathbf{D_j^i}(t)$	Distance of node i to j in $\mathbf{G}(t)$
$\mathbf{l_k^i}(t)$	Cost of link (i,k) in $\mathbf{G}(t)$

any node to j. On the other hand, if each S_j^i has more than one element, then SG_j is a directed acyclic graph (DAG), and has greater connectivity than a simple tree and therefore enabling traffic load balancing.

B. Solution Strategy

Given that there are potentially many directed acyclic graphs for a given destination in a graph, a question arises as to which DAG must be used as a routing graph? The routing graph should be uniquely defined and it should be easily computable by a distributed algorithm. The natural choice for routing graph is the one defined by the shortest paths. Accordingly, MDVA defines $S_j^i(t) = \{k | D_j^k(t) < D_j^i(t), k \in N^i\}$, where D_j^i is the cost of the shortest path from i to j measured as the sum of the costs of the links on the path. The routing graph SG_j implied by this set is unique and is called the *shortest multipath*. To compute D_j^i, distributed routing algorithms may exchange any information (distance-vectors or link-states), as long as they ensure that D_j^i's *converge* to the correct distances. Convergence is formally defined as follows. At time t, let $\mathbf{G}(t)$ denote the topology of the network as seen by an "omniscient observer", and let $\mathbf{D_j^i}(t)$ denote the distance of i to j in $\mathbf{G}(t)$. (Note that we use bold font to refer to quantities in $\mathbf{G}$.) Assume the network has stable configuration up to time t. We say that the network has converged to the correct values at t if $D_j^i(t) = \mathbf{D_j^i}(t)$ for all i and j. Now, if a sequence of link cost changes occur between t and t_c and none after t_c, then the routing algorithm is said to converge if at some time t_f, it satisfies $t_c \leq t_f < \infty$ and $D_j^i(t_f) = \mathbf{D_j^i}(t_f) = \mathbf{D_j^i}(t_c)$. In addition, during the convergence phase, the algorithm must ensure that SG_j's are loop-free at every instant. Note the distinction between $\mathbf{D_j^i}$ and D_j^i. $\mathbf{D_j^i}$ is the correct distance, whereas D_j^i is a local variable i and is an "estimate" of $\mathbf{D_j^i}$. D_j^i must eventually equal $\mathbf{D_j^i}$, if $\mathbf{D_j^i}$ does not change further.

In Distributed Bellman-Ford (DBF) algorithm, each node i repeatedly executes the equation $D_j^i = min\{D_{jk}^i + l_k^i \mid k \in N^i\}$ for each destination j, and every time D_j^i changes it reports it to all its neighbors. A known property of DBF is that it always converges, and converges fast when distance to destinations decrease [8]. However, convergence is slow if link-costs increase, and in the extreme case when link failures result in network partitions, DBF never converges. This is the well-known *counting-to-infinity problem* [14]. Intuitively, the count-to-infinity problem results due to "circular" computation of distances; that is, a node computes its distance to destination using a distance communicated by a neighbor, that happens to be the length of the path that passes through the node itself. The node using such a distance is unaware of this because nodes only exchange distance information with no path information.

Circular computation of distances that occur in DBF can be prevented if distance information is propagated along a DAG rooted at a destination. The key idea is that given a DAG, each node computes its distance using distances reported by the "downstream" nodes and reports its distance to "upstream" nodes. This method called *diffusing computations* was first suggested by Dijkstra et al [4] to ensure termination of distributed computation; a diffusion computation always terminates due to the acyclic ordering of the nodes. DUAL [5], the algorithm on which EIGRP [1] is based, uses diffusing computations to solve the count-to-infinity problem. Several other distance vector algorithms have been proposed that use diffusing computation to overcome the counting-to-infinity problem of DBF [8], [9], [16]. The algorithm suggested by Jaffe and Moss [8] allows nodes to participate in multiple diffusing computation of the same destination and requires use of unbounded counters, for which reason it may not be practical. In contrast, a node in DUAL and DASM [16] participates in only one diffusing computation for any destination at any one time and thus requires only a toggle bit. MDVA presented here follows the later approach.

Two questions arise regarding a diffusing computation:

1. Because there are potentially many DAGs for a given destination, which one should be used for diffusing computation?
2. How should a diffusing computation be performed in a dynamic environment in which the chosen DAG changes with time?

The answer to the first question is straightforward: the shortest multipath SG_j is the right choice given that computing SG_j is our final goal. The second question is not as straightforward. A SG_j used for carrying out a diffusing computation can be allowed to change if the following conditions hold: (1) SG_j is acyclic at *every instant*, and (2) at any instant, if a node reports a distance through a neighbor k in S_j^i, it must ensure that k remains in S_j^i until the end of the diffusing computation. That these conditions prevent circular computation of distances can be infered from the following argument. Assume that a circular computation occurs at time t involving nodes i_0, i_1, .. i_m. Let a node i_p, $1 \leq p \leq m$, compute its distance at $t_p \leq t$ using the distance reported by i_{p-1}, and i_0 computes its distance using the distance reported by i_m at t_0. Because i_{p-1} is held in the successor set of i_p for $1 \leq p \leq m$ and i_0 holds i_m until the diffusing computation ends, we have

$$i_0 \in S_j^{i_1}(t_1) \implies i_0 \in S_j^{i_1}(t)$$
$$i_1 \in S_j^{i_2}(t_2) \implies i_1 \in S_j^{i_2}(t)$$

$$
\begin{array}{rcl}
. & . & . \\
. & . & . \\
i_{m-1} \in S_j^m(t_m) & \Longrightarrow & i_{m-1} \in S_j^m(t) \\
i_m \in S_j^0(t_0) & \Longrightarrow & i_m \in S_j^0(t)
\end{array}
$$

Because the $SG_j(t)$ implied by $S_j^i(t)$, is acyclic at every instant t, the above relations indicate a contradiction. Therefore, circular computation is impossible if the above mentioned conditions are enforced. Notice that we intend to propagate the distances along the shortest-multipath SG_j which itself is computed using the distances. This "bootstrap" approach – computing D_j^i using diffusing computation along SG_j and simultaneously constructing and maintaining SG_j – is central to MDVA.

How can we ensure that SG_j is always loop-free? To do this we use a new variable FD_j^i, called the feasible distance, which is an 'estimate' of the distance D_j^i in 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 differ temporarily from D_j^i. Let D_{jk}^i be the distance of k to j as notified to i by k. To ensure loop-freedom at every instant FD_j^i, D_{jk}^i and S_j^i must satisfy the Loop-Free Invariant (LFI) conditions introduced in [15]. The LFI conditions capture all previous loop-free conditions ([5], [16]) in a unified form that simplifies protocol design. The conditions are

Loop-free Invariant Conditions(LFI) [15]:

$$
\begin{array}{rcll}
FD_j^i(t) & \leq & D_{ji}^k(t) \qquad k \in N^i & (1) \\
S_j^i(t) & = & \{\, k \mid D_{jk}^i(t) < FD_j^i(t) \} & (2)
\end{array}
$$

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 is reproduced here for convenience.

Theorem 1: [15] If the LFI conditions are satisfied at any time t, then the $SG_j(t)$ implied by the successor sets $S_j^i(t)$ is loop-free.

For proof of this theorem the reader is refered to [15]. The above theorem suggests that any distributed routing protocol (link-state or distance-vector) attempting to find loop-free shortest multipaths must compute D_j^i, FD_j^i and S_j^i such that the LFI conditions are satisfied, and such that at convergence $D_j^i = FD_j^i = \mathbf{D_j^i}$ = minimum distance from i to j.

III. Multipath Distance-Vector Algorithm

In essence, MDVA uses DBF to compute D_j^i and therefore SG_j, while always propagating distances along the SG_j to prevent count-to-infinity problem and ensure termination. Each node maintains a *main table* that stores D_j^i, the successor set S_j^i, the feasible distance FD_j^i, the reported distance RD_j^i, and SD_j^i, which is the shortest distance possible through the successor set S_j^i. The table also stores $WN_j^i \subseteq S_j^i$, the set of waiting neighbors in a diffusing computation. Each node also maintains a *neighbor table* for each neighbor k that contains D_{jk}^i the distance of neighbor k to j as communicated by k. The *link table* stores the cost l_k^i of adjacent link to each neighbor k. At startup time, a node initializes all distances in its tables to infinity and all sets to null. If a link is down its cost is considered infinity. The distance to unreachable nodes are considered to be infinity.

Nodes executing MDVA exchange information using messages which can have one or more entries. An entry or distance vector is of the form [*type*, j, d], where d is the distance of the node sending the message to destination j and the *type* is one of QUERY, UPDATE and REPLY. We assume that messages transmitted over an operational link are received without errors and in the proper sequence and are processed in the order received.

Nodes invoke the procedure $ProcessDistVect$ shown in Figure 1 to process distance vectors. An event is the arrival of a message, the change in cost of an adjacent link, or a change in status (up/down) of an adjacent link. When an adjacent link becomes available, the node sends an update message [UPDATE, j, RD_j^i] for each destination j over the link. When an adjacent link (i, m) fails, the neighbor table associated with neighbor m is cleared and the cost of the link is set to infinity, after which, for each destination the procedure $ProcessDistVect$(UPDATE, m, ∞, j) is invoked. Similarly, when an adjacent link cost to m changes, l_m^i is set to the new cost and $ProcessDistVect$(UPDATE, m, D_{jm}^i, j) is invoked for each destination j. When a message is received from neighbor k, $ProcessDistVect(type, k, d, j)$ is invoked for each entry $[type, j, d]$ of the message.

Computing distances to each destination can be performed independently. Hence, in the rest of the description, the working of the algorithm is described with respect to a particular destination j. A node can be in ACTIVE or PASSIVE state with respect to a destination j and is represented by variable $state_j^i$. A node is in ACTIVE state when it is engaged in a diffusing computation and waiting for replies from neighbors. Initially, we assume that all nodes are in PASSIVEstate. As long as link cost decrease, MDVA works identically to DBF and the nodes will remain in PASSIVEstate. This is because the condition on line 9 always fails and lines 17-24 are always executed. $ProcessDistVect$ works in such a way that when in PASSIVE state, the condition $D_j^i = FD_j^i = RD_j^i = min\{D_{jk}^i + l_k^i | k \in N^i\}$ always holds, which can be infered from lines 8 and 23. However, if the distance to a destination increases, either because an adjacent link cost changed or a message is received from a neighbor, the condition on line 9 succeeds and the node engages in a diffusing computation. A diffusing computation is initiated by sending query messages to all the neighbors with the best distance SD_j^i through S_j^i, and waiting for the neighbors to reply (lines 14-15). If the increase in distance is due to a query from a successor, the neighbor is added to WN_j^i to indicate that it is waiting for a reply so that a reply can be given when the node transits to PASSIVE state (lines 11-12). When all replies are received, the node can be sure that the neighbors have incorporated the distances that the node reported, and is safe to transit to PASSIVE state. At this point, FD_j^i can be increased and new neighbors can be added to S_j^i without violating the LFI conditions.

When in ACTIVE state, if a query message is received from a neighbor *not* in S_j^i, a reply is given immediately. On the other hand, if the query is from a neighbor m in S_j^i, a test is made to verify if SD_j^i increased beyond the previously re-

00. **procedure** $ProcessDistVect(et, m, d, j)$
01. { et is the type, m is the neighbor, d is the distance, j is the destination }
02. **begin**
03. **if** $(j = thisNode \wedge et = QUERY)$ **then send** [REPLY, j, 0] to m; **endif**
04. $D^i_{jm} = d$;
05. $D^i_j \leftarrow min\{D^i_{jk} + l^i_k | k \in N^i\}$;
06. $SD^i_j \leftarrow min\{D^i_{jk} + l^i_k | k \in S^i_j\}$;
07. **if** $(state^i_j = PASSIVE \vee state^i_j = ACTIVE \wedge last\ reply\ is\ received\ for\ j)$ **then**
08. $FD^i_j \leftarrow min\{D^i_j, RD^i_j\}$;
09. **if** $(D^i_j > RD^i_j)$ **then**
10. $state^i_j \leftarrow ACTIVE$;
11. **if** $(et = QUERY)$ **then**
12. $WN^i_{jm} \leftarrow m$;
13. **endif**
14. $RD^i_j \leftarrow SD^i_j$;
15. $\forall k \in N^i$, **send** [QUERY, j, RD^i_j] to neighbor k;
16. **else**
17. $state^i_j \leftarrow PASSIVE$;
18. **foreach** $k \in N^i$ **do**
19. **if** $(k \in WN^i_j \vee (k = m \wedge et = QUERY))$ **then send** [REPLY, j, D^i_j] to k;
20. **else if** $(RD^i_j \neq D^i_j)$ **send** [UPDATE, j, RD^i_j] to k;
21. **endif**
22. **done**
23. $RD^i_j \leftarrow D^i_j$;
24. $WN^i_j \leftarrow \phi$;
25. **endif**
26. **else**
27. **if** $(et = QUERY)$ **then**
28. **if** $(m \in S^i_j \wedge SD^i_j > RD^i_j)$ **then** $WN^i_j \leftarrow WN^i_j \cup m$;
29. **else send** [REPLY, j, RD^i_j] to m;
30. **endif**
31. **endif**
32. **endif**
33. $S^i_j \leftarrow \{k | D^i_{jk} < FD^i_j\}$;
34. **end**

Fig. 1. Distance vector processing in MDVA.

ported distance (line 28). If it did not, a reply is sent immediately. However, if SD^i_j increased, no reply is given and the query is blocked by adding m to WN^i_j. The replies to neighbors in WN^i_j are deferred until that time when the node is ready to transit to PASSIVE state. After receiving all replies, one of two things can happen: either the ACTIVE phase ends or it continues. If the distance D^i_j increased again after receipt of all replies, the ACTIVE phase is extended by sending new set of queries, otherwise the ACTIVE phase ends. In the case the ACTIVE phase continues, no replies are issued to the pending queries in WN^i_j. Otherwise, all replies are given and the node transits to PASSIVE state satisfying the PASSIVE-state invariant $D^i_j = FD^i_j = RD^i_j = min\{D^i_{jk} + l^i_k | k \in N^i\}$.

IV. Correctness Proofs

To prove the correctness of MDVA consider the following two mutually exclusive and exhaustive cases: (1) some link costs change, but the distances to destinations either decrease or remain unchanged (2) some link costs increase, resulting in an increase in distances to some destinations. MDVA works identical to DBF when distances to destinations only decrease and the same proof of DBF applies [2]. To state this formally, assume the network is stable up to time t and all nodes have the correct distances. At time t, the costs of some links decrease. Since the distances in the tables are such that $D^i_j(t) \geq \mathbf{D^i_j}(t)$, within some finite time t', $t \leq t' < \infty$, $D^i_j(t') = \mathbf{D^i_j}(t)$.

MDVA and DBF behave differently, when some link costs increase such that distances between some source-destination pairs increase. In this case, $D^i_j(t) < \mathbf{D^i_j}(t)$ for some i and j. Both DBF and MDVA first increase D^i_j to a value greater than $\mathbf{D^i_j}(t)$, after which the distances monotonically decrease until they converge to the correct distances. MDVA and DBF, however, differ on *how* they increase the distances. DBF does it step-by-step in small bounded increments until $D^i_j \geq \mathbf{D^i_j}(t)$. How-

ever, when $\mathbf{D_j^i}(t) = \infty$, this leads to the count-to-infinity problem. In contrast, MDVA uses diffusing computations to quickly raise D_j^i so that $D_j^i \geq \mathbf{D_j^i}(t)$, after which it functions similar to scenario 1 described above, and the distances converge to the correct values as before. After the end of all diffusing computations MDVA works just like DBF.

In summary, to show that MDVA terminates, it is sufficient to show that: (1) the SG_j are loop-free at every instant (Theorem 2), (2) every diffusing computation completes within a finite time (Theorem 3), and (3) there is a finite number of diffusing computations (Theorem 5). Finally, we show that MDVA converges to correct distances when it terminates in Theorem 5.

Theorem 2: For a given destination j, the SG_j constructed by MDVA is loop-free at every instant.

Proof: The proof is by showing that the LFI conditions are satisfied during every ACTIVE and PASSIVE phase. Let t_n be the time when the n^{th} transition from PASSIVEto ACTIVE state starts at node i for j. The proof is by induction on t_n. At node initialization time 0, all distance variables are initialized to infinite and hence $FD_j^i(0) \leq D_{jk}^i(0)$, $k \in N^i$. Assume the LFI conditions are true up to time t_n. Then

$$FD_j^i(t) \leq D_{jk}^i(t) \qquad t \in [0, t_n). \tag{3}$$

At any time t, from lines 6, 8, 14 and 23 in the pseudocode in Figure 1, and because $SD_j^i(t) \geq D_j^i(t)$, if follows that

$$FD_j^i(t) \leq RD_j^i(t) \tag{4}$$

and therefore, for t_{n-1} and t_n, we have

$$FD_j^i(t_{n-1}) \leq RD_j^i(t_{n-1}), \tag{5}$$

$$FD_j^i(t_n) \leq RD_j^i(t_n). \tag{6}$$

Let the queries sent at t_n, the start time of the n^{th} ACTIVE phase, be received at a particular neighbor k at $t' > t_n$. From Eq. (4) and the fact that the update messages sent, if any, between t_{n-1} and t_n specify non-increasing distances, we have

$$FD_j^i(t) \leq D_{jk}^i(t) \qquad t \in [t_n, t']. \tag{7}$$

Let t'' be the time when all replies are received and ACTIVE phase ends. During the ACTIVE phase the value of FD_j^i remains unchanged and no new RD_j^i is reported during this period (lines 27-31). Furthermore, during PASSIVE phase, only decreasing values of RD_j^i are reported. Then from Eq. (6) it follows that

$$FD_j^i(t) \leq D_{jk}^i(t) \qquad t \in [t', t''). \tag{8}$$

At t'', irrespective of whether the node transits to PASSIVE state or continues in the ACTIVE phase, from Eq. (4) we have

$$FD_j^i(t'') \leq RD_j^i(t''). \tag{9}$$

In the case that the ACTIVE phase finally ends, we have $FD_j^i(t) \leq D_{jk}^i(t)$ for $t \in [t_n, t'']$. In the PASSIVE phase, RD_j^i can only remain constant or decrease until the next ACTIVE phase at t_{n+1}. Therefore, the LFI conditions are satisfied in the interval $[t_n, t_{n+1})$. On the other hand, if the ACTIVE phase continues, new queries are sent at time t''. Assume all replies for these queries are received at time t'''. From similar argument as above, it follows that $FD_j^i(t) \leq D_{jk}^i(t)$ for $t \in [t_n, t''']$. Thus irrespective of how long the ACTIVE phase continues, the invariant holds between $[t_n, t_{n+1}]$. From induction, therefore, the LFI conditions hold at all times. It then follows from Theorem 1 that SG_j is loop-free at all times. ■

Theorem 3: Every ACTIVE phase has a finite duration.

Proof: An ACTIVE phase may never end due to either of the two reasons: *deadlock* or *livelock*. First we show a deadlock cannot occur. A node that transits to ACTIVE state with respect to a destination sends queries. If the transition is due to a query from a successor, the node defers the reply to this query until it receives the replies to its own queries. Because nodes wait for replies to their queries before replying to a query, there is a possibility of "circular" waits leading to a deadlock. But, this is impossible for the following reasons. First, a node in passive state immediately replies to a query if it does not increase distance to the destination (lines 19). If the query is from a successor that potentially increases SD_j^i, and the node is ACTIVE , the query is held until the ACTIVE phase ends (line 28). Because the SG_j's are loop-free at every instant (Theorem 2), a deadlock cannot occur. Thus, a node that issued queries to the neighbors will eventually receive all the replies and transits to PASSIVE state.

A livelock is a situation where a node endlessly has back-to-back ACTIVE phases without ever replying to the pending queries from the successors. A livelock cannot occur for the following reasons. An ACTIVE phase transition occurs either because the link-cost of an adjacent link increases or a query from a successor is received that increases SD_j^i. But, we know that a query from a successor is blocked if it increases SD_j^i. Because links can change only a finite number of times and there is only a finite number of neighbors for each node from which the node can receive queries, the node can only have finite number of back to back active phases. A node eventually sends all pending replies and enters PASSIVE state. A livelock, therefore, cannot occur. ■

Theorem 4: A node can have only a finite number of ACTIVE phases.

Proof: Assume towards a contradiction that there is a node that does go through an infinite number of PASSIVE to ACTIVE transitions. An active phase transition occurs either because of a query from a successor or a link-cost increase of an adjacent link. Because link costs can change only a finite number of times, the infinite PASSIVE-ACTIVE phase transitions must have been triggered by an infinite number of queries from a neighbor. Let that neighbor be k. Now, by the same argument, k is sending an infinite number of queries because it is receiving an infinite number of queries. But this argument cannot be continued for ever because there is only a finite number of nodes in the network. Because the reply to the neighbor in the successor set causing the phase transition is blocked and the routing graphs

are loop-free at every instant (Theorem 2), there must be a node that transits to ACTIVE state *only* because of adjacent link cost changes. This implies that a link must change its cost infinite number of times — a contradiction of assumption. Therefore, a node cannot have an infinite number of ACTIVE phases. ■

Theorem 5: After a finite sequence of link-cost changes in the network, the distances D_j^i converge to the final correct values.

Proof: Assume at time 0 that every node has correct distances to all the distances. In other words, $D_j^i(0) = \mathbf{D_j^i}(0)$. Assume that a finite number of link cost changes, link failures and link recoveries occur in the network between time 0 and t_c and after t_c no more changes occur. We have to show that at some time t_f, such that $t_c \leq t_f < \infty$, all nodes will converge to the correct distances. That is $D_j^i(t_f) = \mathbf{D_j^i}(t_c) = \mathbf{D_j^i}(t_f)$.

From Theorem 3 and 4, it follows that within a finite time after the last link change, all nodes transit to PASSIVE state and remain in PASSIVE state thereafter. Therefore, let t' be the time when the last ACTIVE phase ends in the network. We prove the following.

1. $D_j^i(t') \geq \mathbf{D_j^i}(t_c)$ for every i and j.
2. Between t' and t_f, all D_j^i's monotonically decrease and eventually converge to the correct distances $\mathbf{D_j^i}(t_c)$ at t_f. That is $D_j^i(t_f) = \mathbf{D_j^i}(t_c)$.

Part 1: Assume towards a contradiction that $D_j^i(t') < \mathbf{D_j^i}(t_c)$. Let $D_j^i(t') = (l_k^i(t') + D_{jk}^i(t'))$ for some $k \in K \subseteq N^i$. Assume that $D_j^k(t') \geq \mathbf{D_j^k}(t_c)$. Also assume that K has only one element. Because $\mathbf{D_j^i}(t_c) = \mathbf{l_k^i}(t_c) + \mathbf{D_j^k}(t_c)$ we have $l_k^i(t') + D_{jk}^i(t') \leq \mathbf{l_k^i}(t_c) + D_j^k(t')$, from which we can infer that either $l_k^i(t') < \mathbf{l_k^i}(t_c)$, or $D_{jk}^i(t') < D_j^k(t')$, or both. If $l_k^i(t') < \mathbf{l_k^i}(t_c)$, it implies that the link cost of (i,k) is not yet increased to $\mathbf{l_k^i}(t_c)$ via a link-cost change event. When it does, the condition on line 9 becomes true and an ACTIVE state transition is triggered. So all ACTIVE phases have not ended. Similarly, if $D_{jk}^i(t') < D_j^k(t')$, then there is message in transit, which when processed by i would trigger a PASSIVE-to-ACTIVE transition. This means that the ACTIVE phases have not yet ended. A contradiction of the assumption. Therefore, when ACTIVE phases end $D_j^i(t') \geq \mathbf{D_j^i}(t_c)$. When K has more than one element, each element will be removed from the successor set one after the other without triggering the ACTIVE transition until the last element, when the ACTIVE state transition finally occurs.

Part 2: After every node becomes PASSIVE at time t', all the messages in transit can only decrease the distances; otherwise, that would result in a transition to ACTIVE state. At this stage MDVA works essentially like DBF and the same proof of DBF applies here. Each time a distance is decreased, the new distance is reported. Because distances cannot decrease forever and are lower bounded by $\mathbf{D_j^i}(t_c)$, the distances will eventually converge to the correct distances $\mathbf{D_j^i}(t_c)$. ■

V. Performance Analysis

The storage complexity is of $O(|N^i||N|)$, as each of the N^i neighbor tables and the main distance table has a size of $O(|N|)$ entries. The computation complexity is the time taken to process a distance vector and it is easy to see that $processDistVector(.)$ takes $O(|N^i|)$. The time complexity is the time it takes for the network to converge after a set of link-cost changes in the network and the communication complexity is the amount of message overhead required for propagating a set of link-cost changes. In a dynamic environment, the timing and range of link cost changes occur in complex patterns that are often determined by the traffic on the network, because of which obtaining closed form expressions for time and communication complexity is impossible. An approximate analysis that is provided in [8] for the case in which communication is synchronous throughout the network also apply to MDVA.

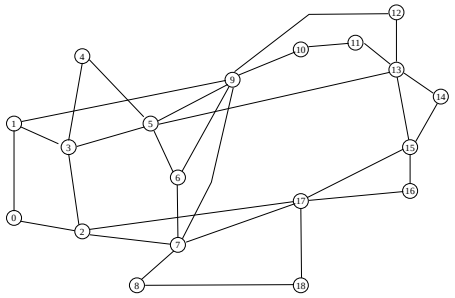

Fig. 2. Example topology

We use simulations to compare the control overhead and convergence time of MDVA with that of DBF, MPDA [15] and topology broadcast (TOPB). The main purpose of these simulations is to give some qualitative explanation for the behavior of MDVA. The reason for choosing DBF and TOPB is that DBF is based on vectors of distances and does not use diffusing computations, while TOPB represents an ideal upper bound on performance of the widely used routing protocols OSPF and IS-IS. The reason for choosing MPDA is that it has been shown to be very efficient compared to TOPB, in terms of communication overhead. MDVA achieves loop-freedom through diffusing computations that, in some cases, may span the whole network. In contrast, MPDA uses only neighbor-to-neighbor synchronization. It is interesting to see how convergence times and control message overheads are effected by the synchronization mechanisms. A comparison of several algorithms that does not include MPDA and MDVA is given in [3].

Simulations are performed on the topology shown in Fig.(2). The simulator used is an event-driven real-time simulator called CPT[1].

We assume the computation time to be negligible compared to the communication times. The bandwidth and propagation delays of each link are 5MB and 100μs respectively. In backbone networks, 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. This is particularly true in near-optimal routing of [15], in which the link costs are periodically measured and reported. For this reason, in this paper we focus on comparing the algorithms in scenarios when multiple link-cost changes occur.

In each experiment, all links are initially set at unit cost and then each link cost is changed by amounts determined by the

[1]We thank Nokia Wireless Routers for allowing us using the C++ Protocol Toolkit

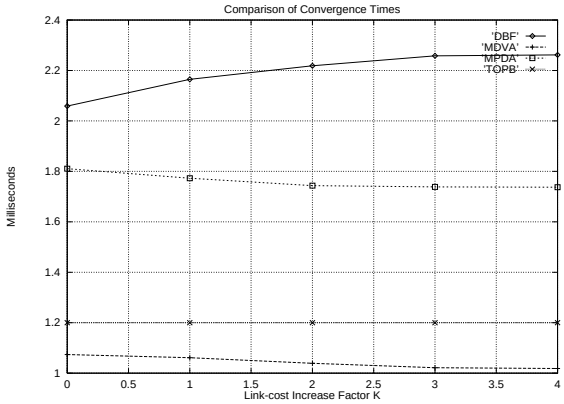

Fig. 3. Average convergence times. $\alpha = 1, \beta = 5$.

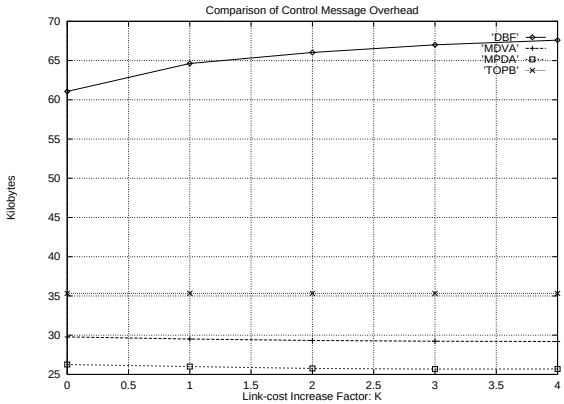

Fig. 4. Average message overhead. $\alpha = 1, \beta = 5$.

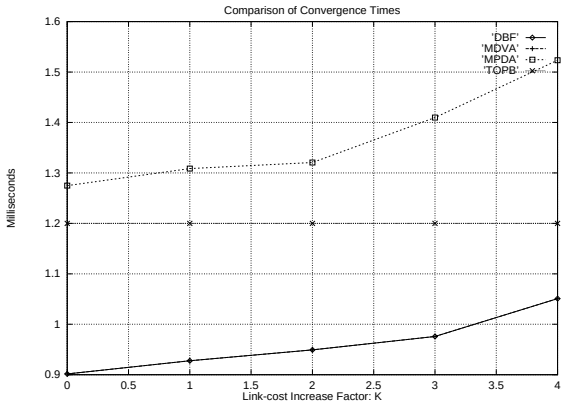

Fig. 5. Average convergence times. $\alpha = -0.1, \beta = -0.4$.

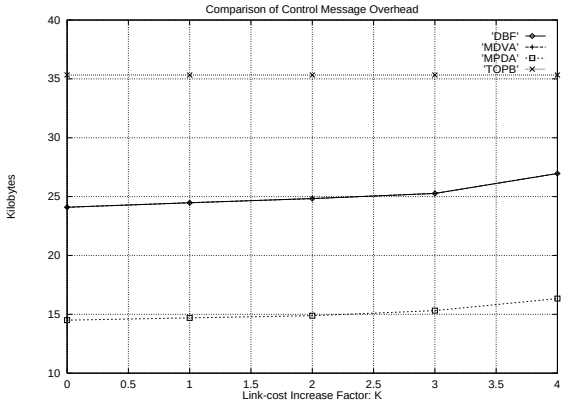

Fig. 6. Average message overhead. $\alpha = -0.1, \beta = -0.4$.

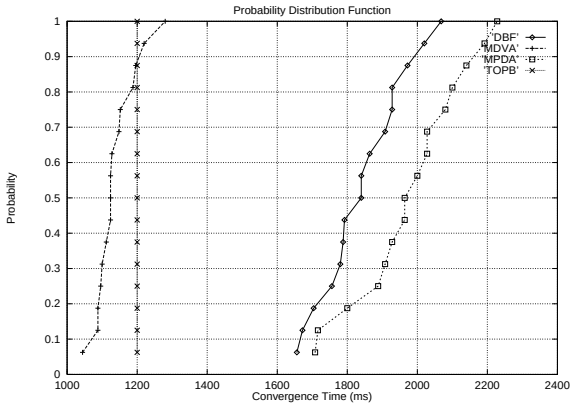

Fig. 7. PDF of convergence times. $\alpha = 1, \beta = 5, k = 1$.

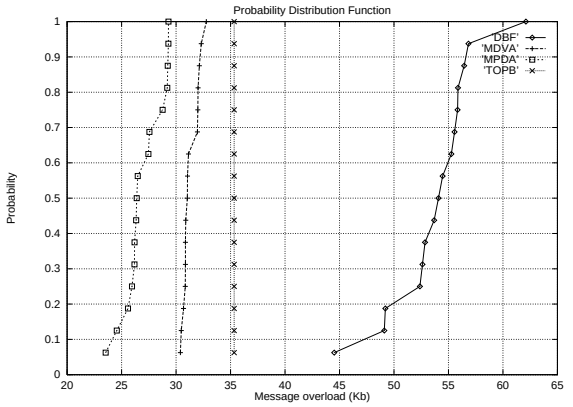

Fig. 8. PDF of message overhead. $\alpha = 1, \beta = 5, k = 1$.

formula $\alpha k + \beta r$, where r is a uniform random value in [0, 1]. The parameters of the experiment α and β are real values while k is a positive integer. After setting the new link costs, the convergence times and message overheads are measured for each routing algorithm. For each experiment with specific α, k and β, several trials are made using different random values for r. The averages and probability distributions obtained for each metric and for each set of trials are compared.

Fig. 3 and Fig. 4 show the average convergence time and average message load, measured over several trials, when the links costs are increased from initial unit cost to a cost using the formula $\alpha k + \beta r$ with $\alpha = 1, \beta = 5$ and $k = 0, 1, 2, 4$. As can be observed in Fig. 3 the average convergence times are best for MDVA. As can be seen in Fig. 4, the average message loads are also low and only MPDA has lower message overhead. Figures 5 and 6 show the averages when link costs decrease. Observe that DBF and MDVA perform identically as can be seen in the figures.

Fig. 7 and Fig. 8 show the complete distribution for convergence times and message overhead for the case $k = 1, \alpha = 1, \beta = 5$. Observe that the distributions are quite uniform compared to DBF. When k is increase to 5 from 1, the convergence times and message overheads of MDVA, as shown in Fig. 9 and Fig. 10, have not changed much, but the performance of DBF

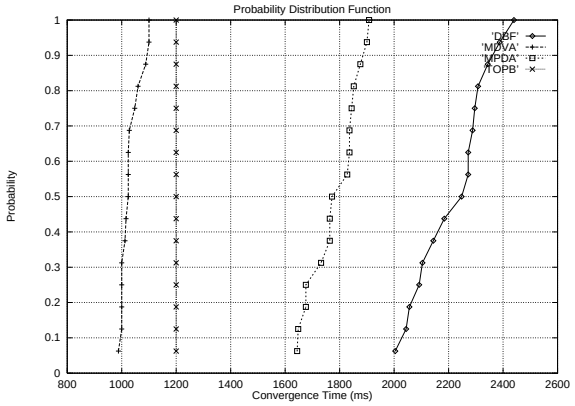

Fig. 9. PDF of convergence times. $\alpha = 5, \beta = 5, k = 1$.

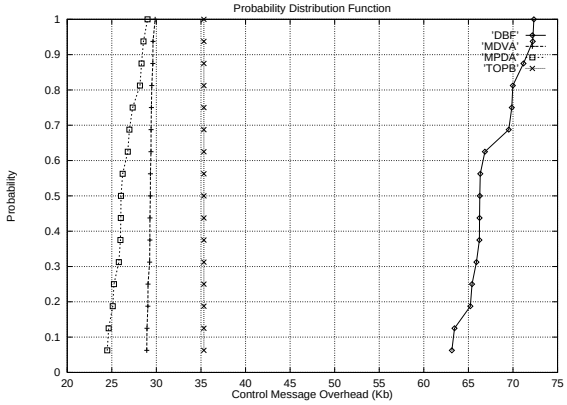

Fig. 10. PDF of message overhead. $\alpha = 5, \beta = 5, k = 1$.

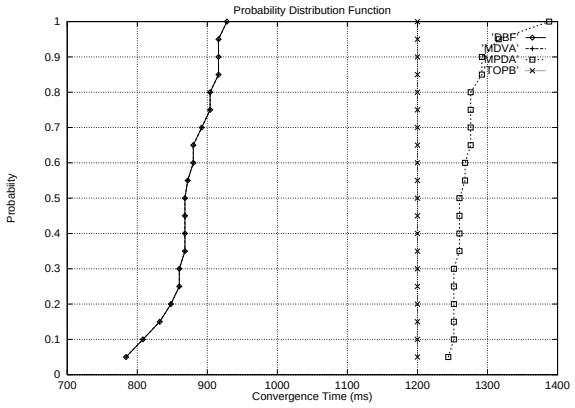

Fig. 11. PDF of convergence times. $\alpha = 0, \beta = -0.4$.

has degraded considerably. This is because of the counting-to-infinity problem, which is does not occur in MDVA.

Fig. 11 and Fig. 12 show the convergence time and message overhead distribution when link costs decrease ($\alpha = 0$, $\beta = -0.4$). (Note that we make sure that link costs do not become negative.) Observe that the performance of MDVA and DBF are much the same which is because MDVA essentially functions like DBF when distances to destinations decrease. From these simulations it appears that MDVA is a good choice if low convergence times are desired at the expense of high message overload while MPDA is preferable if low message overhead is desirable over convergence times.

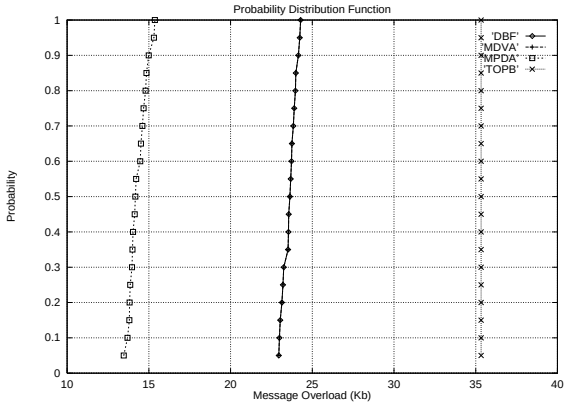

Fig. 12. PDF of message overhead. $\alpha = 0, \beta = -0.4$.

VI. Summary

This paper presented a new distributed distance-vector routing algorithm, MDVA, which is free from the count-to-infinity problem, provides multiple next-hop choices for each destination, and the routing graphs implied by them are always loop-free. The novelty of the algorithm lies in its design around a set of loop-free invariant conditions which ensures instantaneous loop-freedom and correct termination of the protocol. Formal proofs are presented to show MDVA's convergence, correctness and loop-freedom. Through simulation we have compared it to some currently used routing protocols.

References

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

[2] D. Bersekas and R. Gallager. Data networks. 2nd ed. *Prentice-Hall*, pages 404–410, 1992.

[3] I. Matta et. al. Transient and Steady-State Performance of Routing Protocols: Distance Vectors vs. Link State. *Journal of Internetworking: Research and Experience*, 6:59–87, 1995.

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

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

[6] 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.

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

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

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

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

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

[12] S. Roy and J.J. Garcia-Luna-Aceves. Using Minimal Source Trees for On-Demand Routing in Ad Hoc Networks. *INFOCOM 2001*, 2001.

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

[14] A. Tanenbaum. Computer networks. 3rd ed. *Prentice-Hall*, pages 357–358, 1996.

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

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