Title

Distributed, scalable routing based on link-state vectors

Authors

Behrens, J.
Garcia-Luna-Aceves, J.J.

Publication Date

1994-10-01

Peer reviewed

Distributed, Scalable Routing Based on Link-State Vectors

Jochen Behrens J.J. Garcia-Luna-Aceves
University of California
Santa Cruz, California 95064
jochen, jj@cse.ucsc.edu

Abstract

A new family of routing algorithms for the distributed maintenance of routing information in large networks and internets is introduced. This family is called link vector algorithms (LVA), and is based on the selective diffusion of link-state information based on the distributed computation of preferred paths, rather than on the flooding of complete link-state information to all routers. According to LVA, each router maintains a subset of the topology that corresponds to the links used by its neighbor routers in their preferred paths to known destinations. Based on that subset of topology information, the router derives its own preferred paths and communicates the corresponding link-state information to its neighbors. An update message contains a vector of updates; each such update specifies a link and its parameters. LVAs can be used for different types of routing. The correctness of LVA is verified for arbitrary types of routing when correct and deterministic algorithms are used to select preferred paths at each router. LVA is shown to have smaller complexity than link-state and distance-vector algorithms, and to have better average performance than the ideal topology-broadcast algorithm and the distributed Bellman-Ford algorithm.

1. Introduction

An internetwork consists of a collection of interconnected domains, where each domain is a collection of such resources as networks, routers, and hosts, under the control of a single administration. All the work in inter-domain and intra-domain routing has proceeded in two main directions: protocols based on distance-vector algorithms (DVA), which we call distance-vector protocols, characterized by BGP [26], IDRP [22], RIP [18], Cisco's IGRP [4], and EIGRP [1]; and protocols based on link-state algorithms (LSA), which we call link-state protocols, characterized by the inter-domain policy routing (IDPR) architecture [33], ISO IS-IS [21] and OSPF [28].

The key advantage of distance-vector protocols is that they scale well for a given combination of services taken into account in a cost metric. Because route computation is done distributedly, distance-vector protocols are ideal to support the aggregation of destinations to reduce communication, processing, or storage overhead. However, although DVAs have been proposed that eliminate the looping problems of old distance-vector protocols like EGP and RIP [13], an inherent limitation of using distance vectors is that routers exchange information regarding path characteristics, not link or node characteristics. Because of this, the storage and communication requirements of *any* distance-vector protocol (e.g., BGP and IDRP) grows proportionally to the number of combinations of service types or policies [23]; therefore, supporting many types of service together with different types of policies using any distance-vector protocol is inherently complex.

Because LSAs replicate topology information at routers, they avoid the long-term looping problems of old distance-vector protocols. More importantly, an LSA exchanges information regarding link characteristics, which means that the complexity of providing multiple types of services and policies grows linearly with the service types and policies, not their combinations. However, a key disadvantage of today's link-state protocols is that they require routers to broadcast complete topology information by flooding. As pointed out by Estrin and others [7], [8], this approach does not scale well.

The main scaling problems of today's link-state protocols are three: flooding consumes excessive communication resources, requiring each router to compute routes using the same topology database at every router consumes excessive processing resources (e.g., see the results shown in [37]), and communicating complete topology information is unnecessary if a subset of links in the network is not used in the routes favored by routers. Today's link-state protocols (e.g., OSPF) cope with the scaling problems inherent in any LSA by organizing the network or internet into areas connected by a backbone; however, this imposes additional network configuration problems and, as the results in [16] indicate, at least one DVA (DUAL [13]) using areas outperforms OSPF even in relatively small networks.

In summary, the routing algorithms used in today's routing protocols have inherent scaling problems. DVAs need to communicate routing information among routers on a per path basis, which leads to a combinatorial explosion of service types and policies. LSAs require the same topology information to be replicated at all routers, which consumes excessive communication and processing resources in very large internets, or leads to additional network management problems. Surprisingly, although the inherent limitations of LSAs and DVAs are well known, all of the existing rout-

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

ing protocols or proposals for routing in large internets are based on these two types of algorithms [6], [8], [9].

This paper presents a new method for distributed, scalable routing in computer networks called *link vector algorithms*, or LVA. The basic idea of LVA consists of asking each router to report to its neighbors the characteristics of each of the links it uses to reach a destination through one or more preferred paths, and to report to its neighbors which links it has erased from its preferred paths. Using this information, each router constructs a *source graph* consisting of all the links it uses in the preferred paths to each destination. LVA ensures that the link-state information maintained at each router corresponds to the link constituency of the preferred paths used by routers in the network or internet. Each router runs a local algorithm or multiple algorithms on its topology table to compute its source graph with the preferred paths to each destination. Such algorithm can be any type of algorithm (e.g., shortest path, maximum-capacity path, policy path) and the only requirements for correct operation are for all routers to use the same algorithm to compute the same type of preferred paths, and that routers report all the links used in all preferred paths obtained.

Because LVAs propagate link-state information by diffusing link states selectively based on the distributed computation of preferred paths, LVAs reduce the communication overhead incurred in traditional LSAs, which rely on flooding of link states. Because LVAs exchange routing information that is related to link (and even node) characteristics, rather than path characteristics, this approach reduces the combinatorial explosion incurred with any type of DVA for routing under multiple constraints [23]. Aggregation of information can take place in an LVA by adapting the area-based routing techniques proposed for DVAs in the past [12], [25], [27], [35]. In contrast, aggregation of information in traditional LSAs is difficult, because routers need to define different levels of topologies in order to use topology broadcast methods.

The following sections introduce the network model assumed throughout the rest of the paper; describe LVA; show that LVA converges to correct paths a finite time after the occurrence of an arbitrary sequence of link-cost or topological changes under the assumption that all routers run the same local algorithm(s) for the computation of preferred paths; compare its performance with that of LSAs and DVAs in terms of its complexity under the assumption that a shortest-path routing algorithm is used at every router to compute preferred paths; and compare the average performance of LVA against the performance of the ideal LSA and the Distributed Bellman Ford (DBF) algorithm.

2. Network Model

To describe LVA, an internet is modeled as an undirected connected graph $G = (V, E)$, where V is the set of nodes and E the set of edges. Routers are the nodes of the graph and networks or direct links between routers are the edges of the graph. Each point-to-point link in such a graph has two lengths or costs associated with it—one for each direction. Any point-to-point link of the graph exists in both directions at any one time. For a multipoint link, the cost of the link is assumed to be the same in all directions, and exists in all directions at any one time. An underlying protocol assures that

- Every node knows who its neighbors are, 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 messages 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 are detected.

Each router has a unique identifier, and link costs can vary in time but are always positive. Furthermore, routers are assumed to operate correctly, and information is assumed to be stored without errors. The same model can be applied to a computer network, and is the model used in the DVAs and LSAs reported in the past.

3. Basic Method

The basic idea of LVA's design consists of asking each router to report to its neighbors the characteristics of every link it uses to reach a destination through a preferred path. The set of links used by a router in its preferred paths is called the *source graph* of the router. The topology known to a router consists of its adjacent links and the source graphs reported by its neighbors. The router uses this topology information to generate its own source graph using one or more local algorithms, which we call *path selection algorithms*. In the case of shortest-path routing, Dijkstra's algorithm or the Bellman-Ford algorithm could be used as a path-selection algorithm [3]. A router derives a routing table specifying the successor, successors, or paths to each destination by running a local algorithm on its source graph; of course, this can simply be a sub-algorithm of the path-selection algorithms used.

In addition to the parameters of a link, the record of each link entry in the topology table must contain the set of neighbors that reported the link. This can be implemented by means of a bit vector. Since a router's neighbors can be sorted, a single bit per neighbor suffices to indicate whether it is member of the set for a given link or not. The structure of the vector is modified when neighbors are added or deleted.

The basic update unit in LVA is a link-state update reporting the characteristics of a link; an update message contains one or more updates. For a link between router x and router or destination y, router x is called the *head node* of the link in the x to y direction. For a multipoint link, a single head node is defined. The head node of a link is the only router that can report changes in the parameters of that link.

The main complexity in designing LVA stems from the fact that routers with different topology databases can generate long-term or even permanent routing loops if the information in those databases is inconsistent on a long-term or permanent basis.

Sending updates specifying only those links that a router currently uses in its preferred paths is not sufficient in LVA, because a given router sends incremental updates and may stop forwarding state information regarding one or more links that are not changing the values of their parameters. When this happens, it is not possible to ascertain if the

router is still using those links in preferred paths if routers' updates specify only those links currently used in preferred paths. Simply aging link-state information would lead to unnecessary additional control traffic and routing loops, specially in very large internets. Therefore, to eliminate long-term or permanent routing loops, routers must not only tell its neighbor routers which links they use in their preferred paths, but also which links they no longer use. Accordingly, routers using LVA send update messages with two types of update entries: *add updates* and *delete updates*. An add update reports a link that should be added to the source graph of the sending router or whose information should be updated; a delete update specifies a link that should be deleted from the source graph of the sending router.

A router reports its source graph to its neighbors incrementally; therefore, a typical update message in LVA contains only a few add and delete updates. Of course, when a router establishes a new link, it has to send its entire source graph to the new neighbor; this is equivalent to the LSA case in which a router sends its entire topology table to a new neighbor, or the DVA case in which a router sends its entire routing table to a newly found neighbor.

An update specifies all the parameters of the link (just like in an LSA) and a router sends an update in a message only when a link is modified, added, or deleted in its source graph, not when the same unmodified link is used for a modified set of preferred paths. Therefore, the number of update messages and the size of update messages do not necessarily increase with the number of paths that a router uses.

Because of delays in the routers and links of the internet, the add or delete updates sent by a router may propagate at different speeds along different paths. Therefore, a given router may receive an update from a neighbor with stale link-state information. The consistency of link-state information can be controlled on a link-by-link basis taking advantage of the fact that the only router that can change the information about a given link is its head node. More specifically, a distributed termination-detection mechanism is necessary for a router to ascertain when a given update is valid. Termination-detection mechanisms based on sequence numbers similar to those used in a number of LSAs and the associated protocols [3], [28], [29] or diffusing computations [14] can be used. Section 5. presents an instance of an LVA using sequence numbers.

The importance of termination-detection mechanisms for LVA is illustrated in Figure 1. In the figure, link (x, y) is assumed to change cost from 10 to 1 and then fail. Because of latencies in the network, router a receives updates from b regarding link (x, y) before it receives similar update from c. Because of that delay, unless there is a way for routers to determine when an update is stale, a link value of 10 and infinity may circulate among routers a, b, and c forever. The same problem occurs in any LSA, of course [3].

4. Differences With Previous Methods

Before proceeding with a detailed description of a specific LVA, it is worth showing how LVA's basic design differs from prior algorithms based on link-state information. There are three types of prior algorithms that have been or can be used to compute preferred paths based on link-state information:

- *Link-state algorithms* (LSA): This type of algorithms are also called *topology broadcast algorithms*. In an LSA, information about the state of each link in the network is sent to every router by means of a reliable broadcast mechanism, and each router uses a local algorithm to compute preferred paths.
- *Path-finding algorithms* (PFA): These are DVAs that exchange distance vectors containing the length and second-to-last hop (predecessor) of the shortest path to a destination.
- *Path-vector algorithms* (PVA): These are DVAs in which routers exchange distance vectors whose entries specify complete path information for any destination they need to reach.

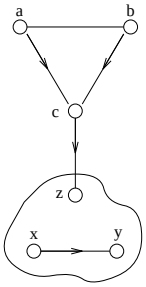

Fig. 1. Possibility of nontermination in LVA

4.1 Differences with LSAs

The key difference between LSAs and LVA is that a link-state update propagates to all routers in an LSA, while in LVA the update propagates to only those routers that use the corresponding link in a path to a destination. Therefore, the reliable broadcast mechanism used in LSAs to ensure that all routers with a physical path to a source of link-state updates receives the most recent updates within a finite time (e.g., see [2], [14], [10], [19], [24], [29]) is not applicable to an LVA. Furthermore, as argued before, a router using LVA must explicitly state which links it stops using.

Figure 2 illustrates how LVA reduces storage and communication overhead compared to LSAs, even for the case of a fairly compact topology. Figure 2(a) shows the complete topology. For simplicity, it is represented as an undirected graph and it is assumed that both directions of each link have the same cost. An LSA would require each router to maintain a copy of the entire topology, with an entry for each link in each direction.

Figures 2 (b) through (e) show the partial topology known at various routers (the black node is the router holding the information). Solid lines represent the links that are part of the source graph of the respective router, dashed links represent links that are part of the router's topology table but not of its source graph. Arrowheads on links indicate the direction of the link stored in the router's topology table. A link with two arrowheads corresponds to two links in the topology table.

Router x's source graph shown in Figure 2(b) is formed by the source graphs reported by its neighbors y and z (these are formed by the links in solid lines shown in Figures 2(c) and (d)) and the links for which router x is the head node (namely links (x, y) and (x, z)).

It is clear from this example that LVA reduces the amount of topology information each router needs to know to provide

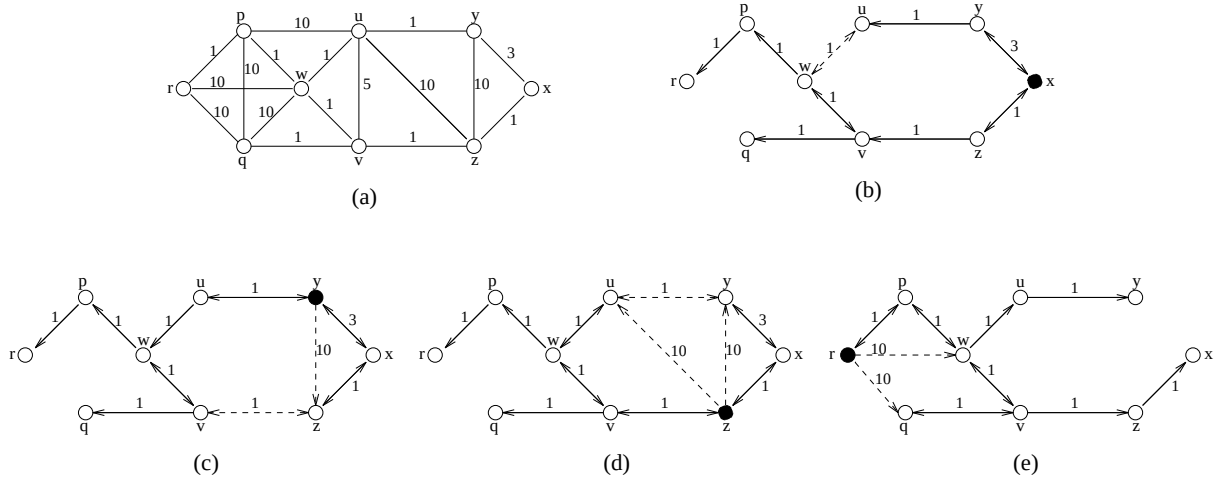

Fig. 2. Example topology

shortest paths to all destinations. For example, router r knows about its links to routers w and q, but routers x, y, and z do not have such links in their topology databases.

A router's topology table may contain a link in only one direction (e.g., link (y, u) in Figure 2(b)). This is because a router's source graph contains links only in the directions of its preferred paths. While this detail is not important in the present example, in an internet, the "links" between routers may be paths through networks that need not have the same cost in both directions.

Given that LVA reduces the amount of topology information stored at each router, an obvious question is whether this results in fewer routes being available to the router. Because a router's source graph contains the links in all its preferred paths, a router using LVA for a given type of routing (e.g., shortest path) has the same number of paths available than with an LSA for the same type of routing. In our example, each router obtains the same shortest-path spanning tree than it would obtain with any LSA.

Obviously, in the worst case, each router's source graph contains all the links in the network and LVA requires the same communication and storage overhead as an LSA. The number of updates and size of updates in LVA are bounded by a number proportional to the number and size of updates in an LSA, because in that case update messages contain add updates reporting changes to the parameters of network links, just as in an LSA. The average size of updates is difficult to characterize analytically; however, the simulation results presented in Section 8. indicate that updates in LVA are small.

4.2 Differences with PFAs

Recently, several PFAs algorithms have been proposed (e.g., see [5], [11], [17], [20], [30]). The basic idea in a PFA is for each router to maintain the shortest-path spanning tree reported by its neighbors (i.e., those routers connected to it through a direct link or a network), and to use this information, together with information regarding the cost of adjacent links, to generate its own shortest-path spanning tree. An update message exchanged among neighbors consist of a vector of entries that reports incremental or full updates to the sender's spanning tree; each update entry contains a destination identifier, a distance to the destination, and the second-to-last hop in the shortest path to the destination.

Another PFA by Riddle [32] is similar to the ones just mentioned in that a router communicates information regarding the second-to-last hop in the shortest path to each known destination. However, it uses exclusionary trees, rather than shortest-path spanning trees, and the cost of the link between the second-to-last hop and the destination, rather than the distance to the destination. An exclusionary tree sent from router x to router y consists of router x's entire shortest-path tree, with the exception of the subtree portion that has node y as its root. Riddle's algorithm does not use incremental updates.

Of course, just as it is done in Riddle's algorithm, any path-finding algorithm can use the cost of the link between the second-to-last hop and the destination, rather than the distance to the destination. However, there are two key differences between LVA and PFAs, namely:

- The set of preferred paths used by a node to reach other nodes need not constitute a tree in LVA and it is always a tree in a path-finding algorithm.
- In the path-finding algorithms proposed to date, there is no notion of how recent the path information reported by a neighbor is.

There are many reasons why routers may want to communicate link-state information of preferred paths that do not correspond to a tree. For example, if multiple shortest paths are desired, a router will communicate links along multiple preferred paths to each destination. Another example is the case in which a router communicates links along multiple preferred paths to each destination because different criteria are used to derive each path (e.g., delay, reliability, administrative constraints).

Because there can be multiple links leading to the same node in the subgraph of preferred paths communicated by a router, a router that receives an incremental update from a neighbor cannot simply assume that the link from node a to node b communicated by its neighbor can substitute any previously reported link from another node c to node b by the same neighbor, as it is done in all path-finding algorithms but Riddle's. On the other hand, transmitting entire subgraphs of preferred paths, as it is done in Riddle's

algorithm, becomes unacceptable in large networks and internets. Therefore, the update mechanisms used in path-finding algorithms to update the subset of link states maintained at each router are not applicable to LVA.

4.3 Differences with PVAs

The existing internet routing protocols based on PVAs (BGP [26] and IDRP [31]) do not exchange link-state information per se. However, such information can be exchanged in a PVA by including it as part of the information for each hop of the reported path in an update or update entry. This, however, would become very inefficient when the size of the network and the number of link-state parameters are large, or when multiple preferred paths to each destination are desired. Some savings can be obtained by specifying a path bit vector, instead of specifying the path hops explicitly. To use the path bit vector, the links of the network have to be ordered and a maximum number of links has to be defined, such that the presence or absence of the link in the preferred path can be indicated by a 1 or a 0 in a vector containing as many bits as links there are in the network. Unfortunately, this approach forces the routers to agree on the number and order of links; therefore, adding and deleting links requires all the routers to coordinate their update activity, which limits the benefits of a PVA over an LSA.

A more subtle difference between LVA and any PVA using link-state information is that routers using LVA determine whether or not an update to a link state is valid based on the timeliness of that update alone, just as in an LSA. In contrast, a router using a PVA that communicates link-state information still has to operate on a path-oriented basis, i.e., the timeliness of an update refers to an entire path, not its constituent links; therefore, even if a router is able to ascertain that a given update is more recent than another, that update may still use link-state information that is outdated (e.g., regarding links that are far away in the path). To eliminate the possibility of using stale link-state information in an adopted path, each link of the path could be validated (with a sequence number, for example), but this becomes inefficient in a large internet.

Even if the above limitations of PVAs are overlooked and PVAs are used to report path information only, the fact is that LVA provides routers with the same path information that a PVA provides, but with far less overhead. This is the case because a router that uses a given link in one or more preferred paths reports that link only once in LVA, while it has to include the link in each preferred path it reports using a PVA. LVA makes reporting complete path information unnecessary for supporting either source routing or hop-by-hop routing.

As the number of paths that a router can use per destination grows, the number of updates communicated and stored in PVAs grows. Furthermore, if multiple types of service are provided, the complexity of PVAs grows proportional to the number of combinations of service parameters. In contrast, the complexity of LVA grows with the number of service parameters, because each update simply adds more parameters to describe a link.

Figure 2(b) helps to illustrate the above points for the case of shortest-path routing. Using LVA, router x requires eight update entries of one link state each to inform its neighbors about its preferred paths. In contrast, a PVA requires the same number of update entries, but many of those entries contain redundant information (e.g., the update entry for the path from x to p is redundant with the entry for the path from x to r). Although some paths that are fully contained in other paths can be simply implied by the paths that contain them, paths may overlap in many different ways in a large internet.

The same type of savings become apparent for the case of multipath routing. In LVA, if router x decides to use the additional path containing link (u, w) to reach router w, it simply has to send an add update with the parameters of the link. In contrast, a PVA requires the router to send an update with the entire new path, which contains a substantial amount of redundant information.

5. An LVA Based on Sequence Numbers

This section describes a concrete example of an LVA designed for shortest-path routing that validates updates by means of sequence numbers. We denote this embodiment of LVA by LVA-SEN. The information regarding each link in a router's topology table is augmented to include the sequence number of the most recent update generated by the link's head node for that link.

A sequence number is associated with each link; it consists of a counter that can be incremented only by the head node of the link. For convenience, a router need to keep only one counter for all the links for which it is the head node, which simply means that the sequence number a router gives to a link for which it is the head node can be incremented by more than one each time the link parameters change values.

The concrete algorithm, LVA-SEN, is shown in Figure 3. For simplicity, the specification assumes that unbounded counters are used to keep track of sequence numbers and that each router remembers the sequence number of links deleted from its topology table long enough for the algorithm to work correctly. The use of finite counters for sequence numbers is addressed in [15]. The specification assumes the following data structures at node i:

- a topology table TT_i with entries (i, j, l, ts, r), for the link (i, j), where l represents the cost of the link, ts its sequence number, and r represents the set of reporting nodes;
- a spanning tree ST_i containing the edges of the tree that is used for routing and their sequence numbers;
- the sequence number t;
- the set of neighbors N_i, and
- the latest sequence numbers reported by the neighbors.

The update messages sent between nodes are vectors of tuples $(i, j, l, ts, type)$, representing link (i, j), its cost l, its sequence number ts, and the label $type$.

Procedure *init* in Figure 3 is used to initialize the network. The sequence number counter is set to zero, and the set of neighbors of a router is determined through information from an underlying protocol. The link to each neighbor, with the appropriate information on the router's operational parameters, is stored in the topology table. At this point, the shortest path tree is essentially the same as the topology table. The information about all its outgoing links is sent to all neighbors. This procedure can only be used to initialize the network, when no information about topology is known at any node. The same mechanism used by a node to recover from a crash must be used to add a new node subsequently.

```
procedure init (i);
   begin
      t = 0
      N_i = {x|∃(i, x), l^i_x ≤ ∞}
      ST_i = ∅
      message = ∅
      u_message = ∅
      for all x ∈ N_i do
         t_x = 0
         message = message ∪ (i, x, l^i_x, t, add)
      end for
      update (i, i, message)
   end init

procedure update (i, n, message)
   begin
      u_message = ∅
      updated = update_topology_table (i, message, u_message)
      if u_message ≠ ∅
         send (n, u_message)
      end if
      u_message = ∅
      if updated then
         build_shortest_path_tree (i, TT_i, NewST_i)
         build routing table
         compare_trees (i, ST_i, NewST_i, u_message)
         remove marked links from TT_i
         if u_message ≠ ∅ then
            for all x ∈ N_i do
               send (x, u_message)
            end for
         end if
         ST_i = NewST_i
         t = t + 1
      end if
   end update

procedure link_down(i, j)
   begin
      message = ∅
      N_i = N_i − {i} (but keep sequence number)
      for all (k, m) ∈ TT_i do
         TT_i(k, m).r = TT_i(k, m).r − {j}
         if TT_i(k, m).r = ∅ or TT_i(k, m).r = {i} then
            message = message ∪ {(TT_i(k, m), delete)}
         end if
      end for
      message = message ∪ {(i, j, ∞, t, delete)})
      update (i, i, message)
   end link_down

procedure link_up(i, j)
   begin
      N_i = N_i ∪ {j}
      update (i, i, {(i, j, l^i_j, t, add)})
      u_message = ∅
      for all (k, m) ∈ ST_i do
         u_message = u_message ∪ (TT_i(k, m), add)
      end for
      send(j, u_message)
   end link_up

procedure link_change(i)
   begin
      update(i, i, {(i, j, l^i_j, t, add)})
   end link_change

procedure node_up (i);
   begin
      t = 0
      message = ∅
      N_i = {x|∃(i, x), l^i_x < ∞}
      for all x ∈ N_i do
         TT_i = TT_i ∪ (i, x, l^i_x, t, {i})
         ST_i = ST_i ∪ (i, x, t)
      end for
      build new routing table
      for all x ∈ N_i do
         send (x, query)
      end for
      answers_received = 0
      while answers_received < |N_i| do
         receive (answer)
         t = max{t, answer.t}
         update_topology_table (i, answer, u_message)
      end while
      t = t + 1
      for all x ∈ N_i do
         TT_i(i, x).t = t
      end for
      build_shortest_path_tree(i, TT_i, NewST_i)
      build new routing table
      ST_i = ∅
      u_message = ∅
      compare_trees (ST_i, NewST_i, u_message)
      for all x ∈ N_i do
         send (x, u_message)
      end for
      ST_i = NewST_i
      t = t + 1
   end node_up

procedure answer_query(i, j)
   begin
      if (i, j) ∉ ST_i then
         ST_i = ST_i ∪ (i, j)
         build routing table
      end if
      for all (k, m) ∈ ST_i
         u_message = u_message ∪ TT_i(k, m)
      end for
      u_message.t = t_j
      send (j, u_message)
   end answer_query

procedure update_topology_table (i, message, u_message)
   begin
      updated = false
      for all m = (j, k, l, ts, type) do
         if type = add then
            if (j, k) ∈ TT_i then
               if TT_i(j, k).t < m.ts then
                  TT_i(j, k) = m
                  TT_i(j, k).r = {message.source }
                  updated = true
               else if TT_i(j, k).t = m.ts and i ≠ message.source then
                  TT_i(j, k).r = TT_i(j, k).r ∪ { message.source }
                  updated = true
               end if
            else if (i ≠ j or message.source = i) and (TT_i(j, k).t <= m.ts) then
               TT_i = TT_i ∪ m
               updated = true
            end if
            if TT_i(j, k).t > m.ts then
               if (j, k) ∈ ST_i then
                  u_message = u_message ∪ (TT_i(j, k), add)
               else
                  u_message = u_message ∪ (TT_i(j, k), delete)
               end if
            end if
         else if type = delete
            if TT_i(j, k).t < m.ts then
               if (j, k) ∈ TT_i then
                  mark (j, k) as deleted
                  updated = true
               else
                  TT_i(j, k).t = m.ts
               end if
            else if TT_i(j, k).t = m.ts then
               if (j, k) ∈ TT_i then
                  TT_i(j, k).r = TT_i(j, k).r−{message.source}
                  if (TT_i(j, k).r = ∅ or TT_i(j, k).r = {i}) and i ≠ message.source then
                     mark (j, k) as deleted
                     updated = true
                  end if
               end if
            else if TT_i(j, k).t > m.ts then
               if (j, k) ∈ ST_i then
                  u_message = u_message ∪ (TT_i(j, k), add)
               else
                  u_message = u_message ∪ (TT_i(j, k), delete)
               end if
            end if
            if TT_i(j, k).l = ∞ and TT_i(j, k).t < m.ts then
               TT_i(j, k).t = m.ts
            end if
         end if
         if j = message.source and j ∈ N_i then
            store sequence number of neighbor
         end if
      end for
      return updated
   end update_topology_table

procedure compare_trees (i, ST_i, NewST_i, u_message)
   begin
      for all (j, k) ∈ NewST_i, ((j, k) ∉ ST_i or NewST_i(j, k).ts > ST_i(j, k).ts) do
         u_message = u_message ∪ (j, k, TT_i(j, k).ts, TT_i(j, k).l, add)
      end for
      for all (j, k) ∈ ST_i, (j, k) ∉ NewST_i do
         if i = j then
            u_message = u_message ∪ (j, k, t, TT_i(j, k).l, delete)
         else
            u_message = u_message ∪ (j, k, TT_i(j, k).ts, TT_i(j, k).l, delete)
         if
      end for
   end compare_trees
```

Fig. 3. LVA-SEN Specification

Procedures *update* and *update_topology_table* are the core part of LVA-SEN. Each time a router receives a message from one of its neighbors, or if there is some change in the cost of an outgoing link reported by a lower-level protocol, these procedures are performed to update all the data structures held at the router.

For all link-state updates in the received message, their sequence numbers must be checked. If the sequence number is older or of the same age as the current content of the topology table, then the information is discarded. Otherwise, there are two possible cases:

1. The label of the information is 'add'. Then this link state is added to the topology table (which can mean that it replaces an old entry), or the reporting node is added to the set of nodes that reported that link.
2. The label is 'delete'. If there is an entry concerning this link in the topology table, then the reporting node is removed from the set of reporting nodes, and the link is deleted from the topology table if that set becomes empty and the link is not an outgoing link of finite length. Otherwise, if there is no entry, the message is discarded.

A shortest-path algorithm (e.g. Dijkstra's algorithm [3]) is run on this updated topology table to construct a new shortest-path tree. This tree is used to compute the new routing table, using for example a depth-first search in the shortest-path tree. Then the new tree is compared to the old tree (procedure *compare_trees*), and the update message that will be send to the neighbors is constructed from the differences of the two trees. Note that a link in the tree is considered different if its sequence number is changed. If the different link is in the new tree, then an add update about this link is added to the update message. If the link is in the old tree but is not in the new one, then a delete update is added. In addition, the link is removed from the topology table, unless it is an adjacent link and its length is not infinity. If any of the link informations refer to the state of an outgoing link of the node itself, then it gets a current sequence number.

Finally, the update message is sent to all the neighbors, the sequence number counter is incremented, the old shortest-path tree is discarded and the new one becomes the current tree.

If a link cost changes, then the node at which this link origins will be notified by the link level protocol. It then runs *update* with the appropriate message as input. This holds for simple changes in link cost as well as link failure. In the latter case, the link cost is set to infinity. The same thing is done for a new link or a link that comes up again after a failure.

In the case of a failing node, all its neighbors are notified about the failure of their links to the failed node. They then remove the failed node from the list of reporting nodes for all affected links, and therefore obtain an accurate picture of the topology after running the *update* procedure.

We assume that the upcoming node does not 'remember' any information that it previously had, in particular it does not know the last sequence number it used. It is not sufficient to simply run procedure *init* because other nodes would discard all update messages with a zero sequence number. In addition, the neighbors of the node would not send sufficient topology information for the node to regain its needed knowledge. After initializing its data structures, the upcoming node sends a query to all its neighbors. In response, they send back their complete shortest-path trees, plus the latest sequence number they received from the node (nodes store sequence numbers of neighboring nodes, which are updated when a link of some neighbor is changed). The node collects all this information, updates its topology table and sequence number, and then performs the same steps as in the procedure *update*.

To illustrate the exchange of updates in LVA-SEN, consider the topology in Figure 2. Assume that link (p, w) fails. Both endpoints of the link note this failure and call procedure *link_down*. The update message sent from w to its neighbors contains delete updates for links (w, p) and (p, r) and add updates for links (w, r) and (r, p). Link (w, p) is deleted from w's topology table because it failed, while (p, r) is only removed from the source graph, because w cannot use this link on the path to r anymore. Instead, the route to r is now link (w, r). This link was not previously used; therefore, it is included in the message in an add update, as is (r, p), which lies on the new route to p. Similarly, node p will send a message to its neighbors containing delete updates for links (p, w) and (w, v), and add updates for links (p, u) and (p, q).

6. Correctness of LVA

This section shows that LVA is correct for multiple types of routing. The proof of correctness makes use of the assumptions introduced in Section 2. and the additional assumptions that there is a finite number of link cost changes up to time t_0, that no more changes occur after that time, and that routers can correctly determine which updates are more recent than others. The correctness of the particular mechanism used to determine which updates are valid in LVA-SEN is addressed in [15].

Correctness for LVA means that, within a finite time after t_0, all routers obtain link-state information that allow them to compute loop-free paths that adhere to the constraints imposed by the local algorithms they use to compute preferred paths, and to forward packets incrementally.

Because our proof of correctness is intended for many different types of routing, not only shortest-path routing, we must specify what we mean by the correct operation of a path-selection algorithm.

To define what a correct path-selection algorithm is, consider first the case in which each router in the network has complete and most recent topology information in its topology table and runs the same path-selection algorithm on it. In this case, it is evident that, for permanent loops to be avoided, the way in which the path-selection algorithm chooses routes must be deterministic.

Assuming that the same deterministic path-selection algorithm is executed at each router using a complete and most recent copy of the topology, the preferred paths at any router for each destination constitute a directed acyclic graph (DAG). Furthermore, the union of the DAGs of any set of routers for the same destination in the network is also a DAG. Therefore, there are no permanent loops in the routing tables computed in this case.

Definition 1: A correct path-selection algorithm is a one that produces the same loop-free paths when it is provided with the same complete and correct topology information.

As we have stated in the description of LVA, all routers use the same path-selection algorithm to compute the same type of preferred paths (e.g., shortest path, maximum capacity), and report all the links used in all the preferred paths obtained through all the path-selection algorithms. Therefore, the rest of this section can assume that a single path-selection algorithm is executed at every router.

Because the topology tables of different routers running LVA need not have the same information, we cannot use the notion of having all topology tables containing the same information to ensure correct paths. The following definition specifies what a topology table should have for loop-free paths to be produced in LVA.

Definition 2: A router is said to have *consistent* link-state information in its topology table if it has the most recent link-state information regarding all the links for whom it is the head node, and the most recent link-state information corresponding to each of its neighbor's most recent source graph.

Theorem 1: A finite time after t_0, all routers have consistent link-state information in their topology tables and the preferred paths computed from those tables are correct.

Proof: Because the deterministic path-selection algorithm used at each router is assumed to be correct, all the proof needs to show is that

1. All routers eventually stop updating their topology and routing tables, and stop sending update messages to their neighbors.
2. All routers obtain consistent link-state information needed to compute correct preferred paths within a finite time after t_0.

These two properties are proven in the following two lemmas.

Lemma 1: LVA terminates within a finite amount of time after t_0.

Proof: First note that there is a finite number of links in the network and that, by assumption, a finite number of link-state changes occur up to time t_0, after which no more changes occur.

By assumption, for each direction of a link whose parameters change, there is one router (the head node of the direction of the link) that must detect the change within a finite time; such a router updates its topology table and must then update its source graph. As a result of updating its source graph, the router can send at most one add update reporting the change in the state of the adjacent link, and at most one add or delete update for each of the links that have been added to or deleted from preferred paths as a result of the change in the adjacent link. Therefore, for any link l_i in the network, its head node can generate at most one update for that link after time t_0.

A given router x_1 that never terminates LVA must generate an infinite number of add or delete updates after time t_0. It follows from the previous paragraph that this is possible only if x_1 sends such updates as a result of processing update messages from its neighbors; furthermore, because the network is finite, x_1 must generate an infinite number of updates for at least one link l_1. Because the network is finite, at least one of those neighbors (call it x_2) must send to x_1 an infinite number of update messages containing an update for either link l_1 or some other link l_2 that makes x_1 generate an update for link l_1. It follows from the previous paragraph and the fact that the network is finite that x_2 can send an infinite number of updates regarding link l_1 or l_2 to x_1 only if at least one of its neighbors (call it x_3) generates an infinite number of updates for either link l_2 or some other link l_3 that makes x_2 generate updates regarding link l_1 or l_2. Because the network is finite, it is impossible to continue with the same line of argument, given that the head node of any link can generate at most one update for that link after time t_0. Therefore, LVA can produce only a finite number of updates and update messages for a finite number of link-state changes and must stop within a finite time after t_0. **q.e.d.**

Lemma 2: All routers must have consistent link-state information in their topology databases within a finite time after t_0.

Proof: The definition of consistent link-state information at a router implies that the router knows all the links it needs to compute correct preferred path, and that the router has the most recent link-state information regarding all the links in its topology table.

Proving that the router receives all the link-state information required to compute correct preferred paths can be done by induction on the number of hops h of a preferred path. What needs to be shown is that the router knows all the links on that path within a finite time after t_0.

Consider some arbitrary preferred path from a router i to some destination. For $h = 1$, the preferred path consists of one of router i's outgoing links. Because of the basic assumption that some underlying protocol provides a router with correct information about its adjacent links within a finite time after the link-state information for such links changes, the lemma is true for this case. For $h > 1$, assume that the claim is true for any preferred path with fewer than h hops.

Consider an arbitrary preferred path of length $h > 1$ from some router i to a destination j. Let k be router i's successor on this path (i.e., the first intermediate router). Then, the subpath from k to j must have length $h - 1$, and it must be one of router k's preferred paths to j. Denote this path by P_{kj}. By the inductive hypothesis, router k knows all the links on P_{kj}. Because router i also knows (as in the base case) the most recent information about link l_k^i within a finite time after t_0, it suffices to show that router k indeed sends the link information in path P_{kj} to its neighbor router i.

Assume that P_{kj} is a new path for router k, then router k must update its source graph. Because P_{kj} is a new path for router k, the information in the updated source graph concerning P_{kj} is different than the information in the old source graph. Therefore, router k must include this information as add updates in the update message that it sends to its neighbors. Because router i is one of those neighbors, it must receive from k all the information on P_{kj} within a finite time after t_0.

By assumption router k can determine which link-state information is valid (i.e., up to date). Accordingly, if P_{kj} is already one of router k's preferred paths, but experiences a change in the information of some of its constituent links, then those links with updated link-state information will be considered different in the new source graph as compared to the old source graph. Therefore, router k must send the updated link-state information in P_{kj} to its neighbor i in add updates.

The same inductive argument holds for link-state changes resulting in links being deleted from a preferred path. In this case, an intermediate router that decides that a link should no longer be used in any of its preferred paths sends a delete update, which is propagated just like an add update. This completes the first part of the proof.

Having shown that a router receives the most recent information about the links used in its source graph within a finite time after t_0, it remains to be shown that it also receives the most recent information about all the links that are in its topology table, but not part of the source graph of preferred paths. There are two possible cases to consider of links in a router's topology table that are not used in its source graph:

- An adjacent link to the router.
- A non-adjacent link is in the source graph reported by some of the router's neighbor.

In the first case, it is obvious that the lemma is true because of the basic assumption of some underlying protocol providing the node with correct information about adjacent links within a finite time. The second case follows almost immediately from the first part of this proof. Because every neighbor of the router sends the appropriate add or delete updates about links added to or deleted from its source own graph, it must be shown that each such neighbor obtains consistent information about changes in its source graph, which was shown to be the case in the first part of this proof. **q.e.d.**

Once Theorem 1 has been shown to be true it is easy to show that these routing tables do not contain any permanent loops.

Corollary 1: The routing tables created by LVA do not contain any permanent loop.

Proof: Lemma 2 shows that the topology information at all routers is consistent within a finite amount of time after any change in link information. The topology information held at any router is a subset of the complete topology, and this subset contains all the information needed at this router to compute the correct preferred paths. Therefore, the preferred paths computed from any router's subset of the topology information must be a subset of the DAG computed in the case of each router having complete topology information. Any subset of a DAG is still a DAG; and the union of any such DAGs also forms a DAG, because that union is also a subset of the DAG obtained with complete topology information. Hence, the routing tables computed by LVA with a correct path-selection algorithm do not contain permanent loops. **q.e.d.**

7. Complexity of LVA

This section quantifies the communication complexity (i.e., number of messages needed in the worst case), time complexity (number of steps), computation complexity, and storage complexity [13] of LVA for shortest-path routing after a single link change.

7.1 Communication Complexity

The number of messages per link cost change is bounded by twice the number of links in the network. To prove that this is the case, it suffices to show that any update can travel each link at most twice.

Assume that an update concerning link l arrives at some arbitrary node n for the first time; there are two possibilities to consider:

1. The link is used in the source graph of n. If this is the case, the corresponding link-state information is sent to some neighbor n_1 over some link l_1. There are two possibilities at this router:
   (a) n_1 uses l: If the information was already known and used at n_1, then no further update will be sent over l_1 (or any other link adjacent to n_1). If it was not previously known at n_1, then an update will be sent to all neighbors of n_1, including one over l_1 to n. From n, no further update with information concerning l will be sent over l_1, until newer information becomes available.
   (b) n_1 does not use l: n_1 will not sent any update with information concerning l, in particular none over l_1.
2. The link is not used at n, in which case no further update will be sent.

From the above, it follows that the number of messages is at most in the order of the number of links in the network ($O(|E|)$).

7.2 Time Complexity

If the cost of links is not directly related to the delays incurred over such links, the number of steps required for any link change is $O(x)$, where x is the number of nodes affected by the change. This can be shown by the following argument: the information about a changed link travels along all the shortest paths that contained the link before the change, and also along all shortest paths that will contain the link after the change. No other router than those along the paths and their neighbors will be notified about the change.

In the worst case, all the affected routers lie along one long path, thus causing $O(x)$ communication steps. In general, the paths on which the information is forwarded together with the affected routers form a directed, acyclic graph, and the upper bound for the steps required is given by the length of the longest simple path in that graph.

Because link failures and recoveries are handled as special cases of link cost changes, and router failures are perceived by the network as link failures for all their links, it is clear that $O(x)$ communication steps are also incurred in these cases. The case of a recovering node involves the nodes getting the complete source graphs from its neighbors, which takes no more steps than the number of neighbors, before the links of the routers again are handled as changing their cost to some finite value. Hence, the same upper bound of $O(x)$ applies.

This worst case is the same as the complexity of any DVA. On the other hand, if the link costs reflect the delay of the links, the complexity for LVA reduces to $O(d)$, where d is the (delay) diameter of the network. The reasons for this are that the information travels along the shortest paths and a router receiving new information can trust the neighbor that reports the most recent link-state for the associated link; most importantly, the node will discard older information from other neighbors. Therefore, a router does not have to wait for link state updates to reach it through the slower paths, as is in the case in DVAs. The flooding technique used in LSAs also takes $O(d)$.

7.3 Complexity of Computations at Routers

The most important routines to analyze are *update* and *update_topology_table*. Most other procedures just call *update* with the appropriate input message. One part of this procedure is the shortest path finding algorithm (Dijkstra). Therefore, the overall complexity is at least $O(|V|^2)$. The complexity of the main loop of procedure *update_topology_table* is determined by the size of the update message. In the worst case, this message could contain information about every link, resulting in running time $O(|E|) \leq O(|V|^2)$. This case seems highly unlikely, though.

In "normal" cases, we would expect an update message to contain information about some path plus possibly a second path that has to be deleted. A path can have at most length $|V| - 1$, leading to an expected complexity of $O(|V|)$. The amount of work in the other loops is bounded by the number of nodes in the network.

Note that there also is the hidden complexity of accessing the topology table. This problem can be solved using a (dynamic) hash table, which has an expected constant access time.

7.4 Storage Complexity

In the worst case, the topology table of each router maintains the whole topology, making the storage requirement $O(|V|^2)$. In addition, both the shortest path tree and the routing table require $O(|V|)$ storage, which is also the case for link state algorithms.

On the average, we expect the storage for the topology table to be by far smaller than $O(|V|^2)$. Because the goal is to keep as sparse a subset of the whole topology as possible, we hope that the required storage space is closer to $O(|V|)$. This seems realistic, even the small topology shown as example in Figure 2 revealed a significant saving of required space when compared to an algorithm that stores the complete topology at all routers. In contrast, the LSAs used today have to store the complete topology. Though the storage required for DVA is linear in the number of routers, routers have to store the routing tables of their neighbors. Therefore, DVAs' storage requirement really become $O(|V||N_k|)$ at router k.

8. Simulation

In this section we compare LVA-SEN, DBF, and a generic LSA in terms of time and communication complexity. Communication complexity is measured as the number of updates that are required for the algorithm to converge and the size of these updates. The time complexity is given in steps: when a node receives an update message, it compares its local step counter with the sender's counter, takes the maximum and increments the count. In all three algorithms, update messages are processed one at a time, in the order in which they arrive. Both LVA-SEN and LSA use Dijkstra's algorithm to compute the local shortest-path tree.

The results presented are based on simulations for the DOE-ESNET topology [36]. The graphs show the results for every single link changing cost from 1.0 to 2.0 (Fig. 4,5), every link failing (Fig. 6, 7) and recovering (Fig. 8, 9), as well as every node failing (Fig. 10, 11) and recovering again (Fig. 12, 13). All changes were performed one at a time, and the algorithms had time to converge before the next change occurred. The ordinate of Figures 4 to 9 and Figures 10 to 13 represent identifiers of the links and the nodes, respectively, that are altered in the simulation. In Figures 4, 6, 8, 10, and 12, the data points show the number of updates and the size of the updates, while in Figures 5, 7, 9, 11, and 13, they show the number of steps needed for convergence.

As expected, LSA shows almost constant behavior for all single link cost changes (Figures 4,5). Similar to DBF, the effort changes considerably from case to case with LVA-SEN, but is always less than in LSA. With LSA, each of the update packets contains exactly one link state, the size of packets remains small with LVA-SEN, too, at an average of 1.36 links per packet.

Figures 6 and 7 show almost the same general behavior for link failures, the exception being DBF suffering from 'counting to infinity' in some cases. In almost all cases, LVA-SEN needs fewer update messages and fewer steps than LSA; the size of the messages is bigger than for link changes, with a mean of 2.61.

When a failed link recovers, DBF shows its strengths and is superior to both LVA-SEN and LSA. Again, LSA behaves very uniform for all the simulated link recoveries. The performance of LVA-SEN varies considerably from case to case, but remains always better than LSA (Figures 8,9). The mean packet size is less than three links per packet; since the packet size of LSA is no longer one in this case (due to the packets containing complete topology information sent over the recovering link), LVA-SEN almost always requires less overall information to be sent.

For failing nodes, LSA usually has the best performance of the three algorithms. DBF always suffers from 'counting to infinity'. In almost all cases, LSA converges faster than LVA-SEN, it needs fewer steps and updates (Figures 10,11). Although the mean packet size for LVA-SEN is very moderate (1.5 links per packet), less information is sent through the network with LSA.

Similar to the recovery of a single link, DBF is superior to LSA when a node comes up, and LVA-SEN performs even better than DBF. It needs fewer steps and updates than the other algorithms (Figures 12,13). Although the mean packet size for LVA-SEN has its highest value here (3.12), this is true for the other algorithms, too, such that LVA-SEN requires the least amount of information to be sent through the network.

Overall, the results of our simulations are quite encouraging. In terms of its overhead, LVA-SEN behaves much like DBF when link costs change or resources are added to the network, and behaves much like the ideal LSA when links or routers fail. This is precisely the desired result, and there are a number of simple ways to improve LVA-SEN's behavior after a link or router failure, which involve establishing a "hold down" on the updating of a router's source graph [15].

9. Conclusions

We have presented a new method for truly distributed routing in computer networks and internets using link-state information. LVAs enjoy nice scaling properties: like DVAs, LVAs scale well with the number of destinations by aggregating information; like LSAs, LVAs scale well with the number of service types because routers communicate link properties, not path properties in their updates.

An important contribution of this paper is to show that

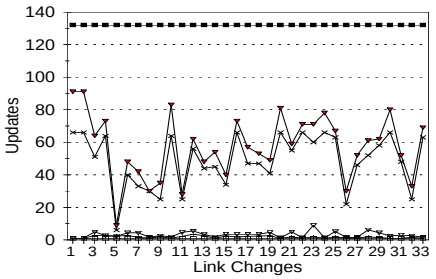

Fig. 4. Updates for link changes, DOE-ESNET

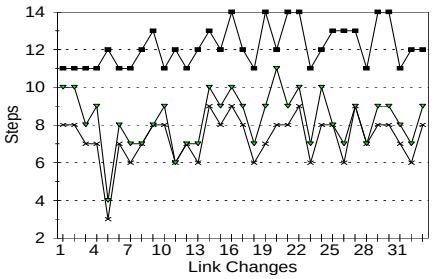

Fig. 5. Steps for link changes, DOE-ESNET

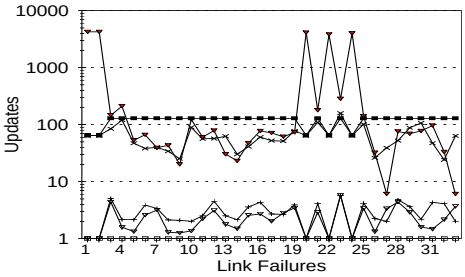

Fig. 6. Updates for link failures, DOE-ESNET

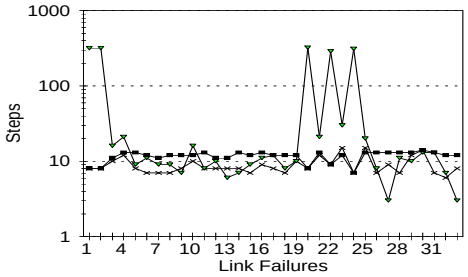

Fig. 7. Steps for link failures, DOE-ESNET

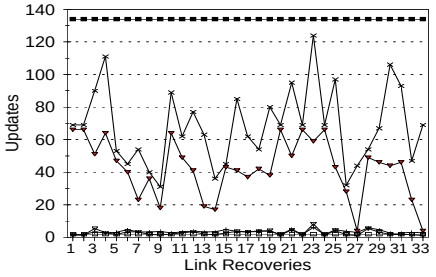

Fig. 8. Updates for link recoveries, DOE-ESNET

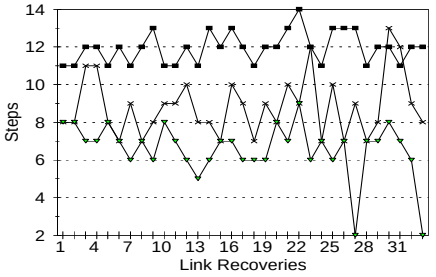

Fig. 9. Steps for link recoveries, DOE-ESNET

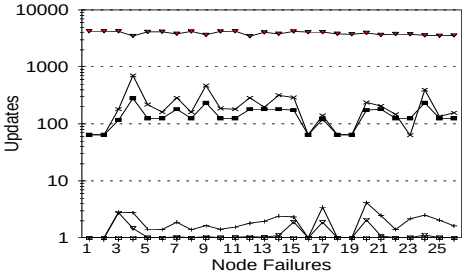

Fig. 10. Updates for node failures, DOE-ESNET

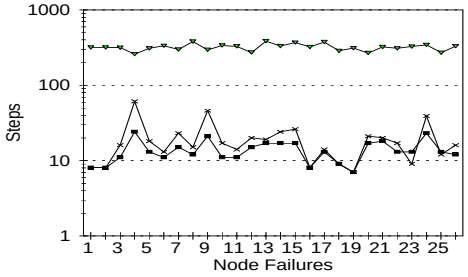

Fig. 11. Steps for node failures, DOE-ESNET

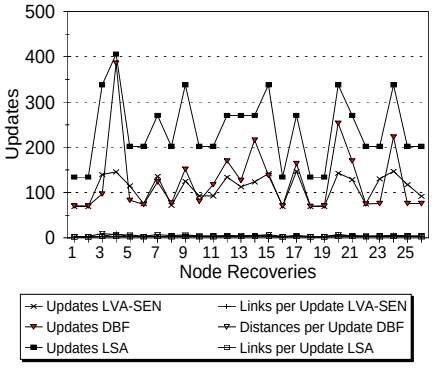

Fig. 12. Updates for node recoveries, DOE-ESNET

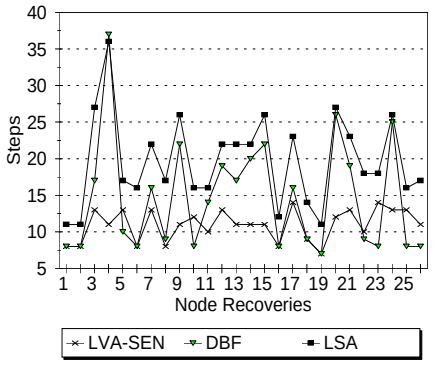

Fig. 13. Steps for node recoveries, DOE-ESNET

LVA is correct under different types of routing, assuming that a correct mechanism is used for routers to ascertain which updates are recent or outdated.

LVAs open up a large number of interesting possibilities for internet routing protocols. To name a few, LVAs can be the basis for the first routing protocols for packet radio networks based on link-state information. Equally important, LVA can be used to develop intra-domain routing protocols that are based on link-state information but require no backbones and can take advantage of aggregation schemes developed for DVAs. Finally, LVAs make PVAs obsolete. Work is continuing to improve the average-case performance of LVAs, and to apply LVAs to policy-based routing, routing with multiple constraints, and hierarchical routing based on link vectors.

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*, Las Vegas, Nevada, May 1994.

[2] B. Awerbuch, I. Cidon, and S. Kutten, "Communication-Optimal Maintenance of Replicated Information," *Proc. IEEE FOCS '90*, pp. 492-502, August 1990.

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

[4] L. Bosack, "Method and Apparatus for Routing Communications among Computer Networks," U.S. Patent assigned to Cisco Systems, Inc., Menlo Park, California, February 1992.

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

[6] J.N. Chiappa, "A New IP Routing and Addressing Architecture," Unpublished Draft, 1991.

[7] D. Estrin and K. Obraczka, "Connectivity Database Overhead for Inter-Domain Policy Routing," *Proc. of IEEE INFOCOM '91*, Miami, Florida, pp. 265-278, April 1991.

[8] D. Estrin, Y. Rekhter, and S. Hotz, "Scalable Inter-Domain Routing Architecture," *Computer Comm. Review*, Vol. 22, No. 4, 1992.

[9] D. Estrin, M. Steenstrup, and G. Tsudik, "A Protocol for Route Establishment and Packet Forwarding across Multidomain Internets," *IEEE/ACM Trans. on Networking*, Vol. 1, No. 1, February 1993, pp. 56-70.

[10] E. Gafni, "Generalized Scheme for Topology-Update in Dynamic Networks," *Lecture Notes in Computer Science* (G. Goos and J. Hartmanis, Eds.), No. 312, pp. 187-196, 1987.

[11] J.J. Garcia-Luna-Aceves, "A Fail-Safe Routing Algorithm for Multihop Packet-Radio Networks," *Proc. of IEEE INFOCOM '86*, Miami, Florida, April 1986.

[12] —, "Routing Management in Very Large-Scale Networks," *Future Generation Computing Systems (FGCS)*, North-Holland, Vol. 4, No. 2, pp. 81-93, 1988.

[13] —, "Loop-Free Routing Using Diffusing Computations," *IEEE/ACM Trans. Networking*, Vol. 1, No. 1, 1993.

[14] —, " Reliable Broadcast of Routing Information Using Diffusing Computations," *Proc. IEEE Globecome 92*, Orlando, Florida, December 1992.

[15] J.J. Garcia-Luna-Aceves and J. Behrens, " Distributed, Scalable Routing Based on Vectors of Link States," Unpublished Report, Baskin Center for CE & CIS, University of California, Santa Cruz, CA, 1994.

[16] J.J. Garcia-Luna-Aceves and W.T. Zaumen, "Area-Based, Loop-Free Internet Routing," *Proc. IEEE INFOCOM 94*, Toronto, Canada, June 1994.

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

[18] C. Hedrick, "Routing Information Protocol," RFC 1058, Network Information Center, SRI International, Menlo Park, CA, June 1988.

[19] P.A. Humblet and S.R. Soloway, "Topology Broadcast Algorithms," *Computer Networks and ISDN Systems*, Vol. 16, pp. 179-186, 1989.

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

[21] International Standards Organization, 1989: "Intra-Domain IS-IS Routing Protocol," ISO/IEC JTC1/SC6 WG2 N323, September 1989.

[22] International Standards Organization, "Protocol for Exchange of Inter-domain Routing Information among Intermediate Systems to Support Forwarding of ISO 8473 PDUs," ISO/IEC/JTC1/SC6 CD10747.

[23] J.M. Jaffe, "Algorithms for Finding Paths with Multiple Constraints," *Networks*, Vol. 14, pp. 95-116, 1984.

[24] J.M. Jaffe, A.E. Baratz, and A. Segall, "Subtle Design Issues in the Implementation of Distributed, Dynamic Routing Algorithms," *Computer Networks and ISDN Systems*, Vol. 12, pp. 147-158, 1986.

[25] L. Kleinrock and F. Kamoun, "Hierarchical Routing for Large Networks: Performance Evaluation and Optimization," *Computer Networks*, Vol. 1, pp. 155-174.

[26] K. Lougheed and Y. Rekhter, " Border Gateway Protocol 3 (BGP-3)," RFC 1267, SRI International, Menlo Park, CA, October 1991.

[27] J. McQuillan, "Adaptive Routing Algorithms for Distributed Computer Networks," BBN Rep. 2831, Bolt Beranek and Newman Inc., Cambridge MA, May 1974.

[28] J. Moy, "OSPF Version 2," Network Working Group Internet Draft, November 1992.

[29] R. Perlman, "Fault-Tolerant Broadcast of Routing Information," in *Computer Networks*, North-Holland, Vol. 7, pp. 395-405, 1983.

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

[31] Y. Rekhter, "Inter-Domain Routing Protocol (IDRP)," *Internetworking: Research and Experience*, Wiley, Vol. 4, No. 2, June 1993, pp. 61-80.

[32] G.G. Riddle, "Message Routing in a Computer Network," U.S. Patent assigned to AT&T Bell Telephone Laboratories, Inc., Patent Number 4,466,060, August 1984.

[33] M. Steenstrup, "Inter-Domain Policy Routing Protocol Specification: Version 1," Internet Draft, May 1992.

[34] J. Spinelli and R. Gallager, "Event Driven Topology Broadcast without Sequence Numbers," *IEEE Trans. Commun.*, Vol. 37, pp. 468-474, May 1989.

[35] P. Tsuchiya, "The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks," *Computer Comm. Review*, Vol. 18, No. 4, 1988, pp. 43-54.

[36] W. Zaumen and J.J. Garcia-Luna-Aceves, "Dynamics of Distributed Shortest-Path Routing Algorithms," *Computer Comm. Review*, Vol. 21, No. 4, pp. 31-42, September 1991.

[37] —, "Dynamics of Link-State and Loop-Free Distance-Vector Routing Algorithms," *Journal of Internetworking: Research and Experience*, Vol. 3, No. 4, pp. 161-188 December 1992.