# **UC Santa Cruz**

# **UC Santa Cruz Previously Published Works**

## **Title**

On-Demand Loop-Free Routing With Link Vectors

# **Permalink**

https://escholarship.org/uc/item/1pd8347b

# Journal

IEEE Journal on Selected Areas in Communications, 23(3)

# **Author**

Garcia-Luna-Aceves, J.J.

## **Publication Date**

2005-03-01

Peer reviewed

# On-Demand Loop-Free Routing With Link Vectors

J. J. Garcia-Luna-Aceves, Senior Member, IEEE, and Soumya Roy

Abstract—We present the on-demand link vector (OLIVE) protocol, a routing protocol for ad hoc networks based on linkstate information that is free of routing loops and supports destination-based packet forwarding. Routers exchange routing information reactively for each destination in the form of complete paths, and each node creates a labeled source graph based on the paths advertised by its neighbors. A node originates a broadcast route request (RREQ) to obtain a route for a destination for which a complete path does not exist in its source graph. When the original path breaks, a node can select an alternative path based on information reported by neighbors, and a node can send a unicast RREQ to verify that the route is still active. A node that cannot find any alternate path to a destination sends route errors reliably to those neighbors that were using it as next hop to the destination. Using simulation experiments in ns2, OLIVE is shown to outperform dynamic source routing, ad hoc on-demand distance vector, optimized link-state routing protocol, and topology broadcast based on reverse-path forwarding, in terms of control overhead, throughput, and average network delay, while maintaining loop-free routing with no need for source routes.

Index Terms—Ad hoc networks, on-demand, link-state routing, loop-freedom.

#### I. INTRODUCTION

EVERAL on-demand routing protocols have been proposed to maintain routing tables efficiently in ad hoc networks [1]-[4]. Two key features of on-demand routing protocols are that routing information is maintained at a given router for only those destinations to which data must be sent, and the paths to such destinations need not be optimum. The basic differences among on-demand routing protocols are how they communicate routing information to obtain paths to destinations, how they use and maintain this information and the manner in which data packets are routed. Maintaining loop-free routes at every instant becomes a necessity in ad hoc networks with dynamic topologies, because routing loops increase packet-delivery latencies and reduce the number of packets delivered to the intended destinations. Current on-demand routing protocols adopt different techniques to prevent temporary loops.

The dynamic source routing (DSR) protocol [1], [5] is an example of protocols that attain loop-free routing using source routes. In DSR, a route request (RREQ) sent to find a given destination records its traversed route, and a route reply (RREP)

Manuscript received October 15, 2003; revised December 3, 2004. This work was supported in part by the U.S. Air Force Office of Scientific Research (OSR) under Grant F49620-00-1-0330 and in part by the Baskin Chair of Computer Engineering, University of California, Santa Cruz, CA.

The authors are with the Computer Engineering Department, University of California, Santa Cruz, CA 95064 USA (e-mail: jj@soe.ucsc.edu; soumya@soe.ucsc.edu).

Digital Object Identifier 10.1109/JSAC.2004.842564

sent by a node in response to the RREQ specifies the complete route between the node and the destination. Routers store the discovered routes in a route cache. The basic scheme in DSR is for the header of every data packet to specify the source routes to their intended destinations.

The ad hoc on-demand distance vector (AODV) protocol [2] is an example that supports incremental packet forwarding and maintains loop-free routing by using a sequence number for each destination. When node A needs to establish a route to a destination D, it broadcasts a RREQ to its neighbors. The RREQ specifies a sequence number for the destination that A increases after losing its route to D. A node receiving the RREQ can send back a unicast RREP along its shortest path to node A only if it has a valid route to node D and the sequence number stored for node D is no less than the sequence number in the RREQ. Otherwise, the node receiving the RREQ must forward the RREQ. Increasing the sequence number for a destination when routes must be changed prevents several nodes with valid and shorter paths to the destination from being used, and in many cases makes the destination the only node that can answer the RREQs.

The temporally ordered routing algorithm (TORA) [3] uses a link-reversal algorithm [6] to maintain loop-free multipaths that are created by a query-reply process similar to that used in DSR and AODV. The limitation with TORA and similar approaches is that they require reliable exchanges among neighbors and coordination among nodes over multiple hops, which incurs more signaling overhead compared to AODV and DSR.

Several routing protocols have been proposed in which a node receives partial or complete link-state information from its neighbors, stores that information in a topology graph, and computes a shortest-path routing tree from such a graph using a shortest-path algorithm locally. The topology broadcast based on reverse-path forwarding (TBRPF) [7] and the optimized link-state routing protocol (OLSR) [8] disseminate complete topology information to all routers, which in turn can compute routes to each destination using a local shortest-path algorithm. None of these protocols eliminates temporary routing loops.

The source tree on-demand adaptive routing (SOAR) protocol [4] was the first to use link-state information on demand. Each router provides its neighbors with a "source tree" consisting of preferred paths to destinations for which the router has traffic. However, SOAR requires data packets to carry the path traversed by the packet in order to avoid routing loops, which incurs as much overhead as the basic source routing scheme of DSR.

This paper presents the on-demand link vector (OLIVE) protocol, which is an on-demand routing protocol based on partial link-state information that supports loop-free hop-by-hop (incremental) routing. Section II provides a detailed description of OLIVE and illustrates its operation. OLIVE does not need internodal synchronization spanning multiple hops, the use of any

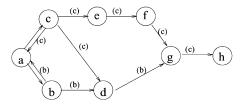


Fig. 1. Network topology known by node a.

packet-header information other than the destination for loopfree packet forwarding, or the use of destination-based sequence numbers. Like other on-demand routing protocols, RREQs and RREPs in OLIVE are sent for initial path setup. Routers running OLIVE exchange path information and the paths received at a node from all its neighbors combine to give a partial network topology. The paths advertised are used for active route setup, while the network topology is used to compute plausible paths that become useful when the original paths break.

Section III demonstrates that OLIVE is loop free at every instant, and that it is safe and live, i.e., that routers find the paths to destinations within a finite time if the network is not partitioned. Loop-freedom is ensured by maintaining an up-to-date list of predecessors at every node for each destination and by informing the predecessors of the route failures reliably. Intermediate nodes between the source and destination can locally repair paths without informing the predecessors.

Section IV compares the performance of OLIVE with the performance of DSR, AODV, TBRPF, and OLSR, which are the four routing protocols for ad hoc networks being considered in the IETF MANET working group. The experiments consider the effect that traffic load and mobility have on the performance of the protocols, and also the role that looping plays. The results of our experiments show that OLIVE provides the most attractive performance based on the metrics we analyze. Section V concludes our paper.

### II. OLIVE

#### A. Motivation

In proactive routing protocols based on link-state information, it is straightforward to compute paths at a router by first aggregating all link-state information received from its neighbors, and then running a shortest-path algorithm locally (e.g., Dijkstra's shortest-path first). The aggregation of those paths is a tree, which is referred to as "source tree." However, similar steps cannot be taken in on-demand routing protocols based on link-state information, because it is not true that every router maintains paths to every destination. Fig. 1 shows the network topology as known to node a based on inputs from neighbors band c. Neighbor b has advertised paths  $\{b, a\}, \{b, d, g\}$ , while node c has advertised paths  $\{c, a\}, \{c, e, f, g, h\},$  and  $\{c, d\}.$ The links in each of those paths aggregate to form the topology as given in Fig. 1. A node deletes information of a link if it is of finite cost and all neighbors have removed it from their advertised paths. Links with infinite cost are never deleted to reduce communication overhead in on-demand routing. The label set of each link in any node's network topology indicates the list of neighbors that have advertised that link to that node. For example, the label set of link (g,h) is  $\{c\}$ , which implies that neighbor c has advertised link (g,h). Using Dijkstra's SPF algorithm, the shortest path for destination h would be abdgh, i.e., node a would pick node b as the next hop to reach destination b. However, this leads to a routing error; upon receiving a data packet destined to node b, node b would drop the data packet, because node b does not have a route to destination b.

Intuitively, the best node a can do to compute a "source tree" subject to the on-demand constraint is trying to obtain a source tree that renders the minimum number of routing errors. However, as shown in the Appendix, this results in an NP-complete problem and cannot be implemented efficiently. The design of OLIVE is motivated by the need for an approach to attain loop-free routing using link-state information on-demand without computing source trees, allowing local route repairs, and requiring no source routes or flow identifiers in the headers of data packets.

#### B. Principles of Operation

In OLIVE, the source of a data packet that has no route for the destination broadcasts a RREQ to its neighbors. The destination or nodes having active routes to the destination respond with route replies (RREPs). RREPs contain paths for destinations and are sent back toward the originator of the RREQ, very much as in DSR. The aggregate of path information obtained in RREPs constitutes a node's labeled network topology. Each path stored in the topology is considered to be active for a finite lifetime, which is renewed when the path is used to forward data. Link-based sequence numbers are used to select the most recent information in case of conflicts of link-state information.

If a router needs to change its current route for a destination after an input event, it first tries to select the shortest path among those complete paths for the destination advertised by its neighbors that are still active. Let  $p_i^{kj}$  be the path for destination j advertised to node i by neighbor  $k, p_i^{kj} \cdot (\cos t)$  be the cost of that path, and  $N_i$  be the set of neighbors of node i. The path-selection algorithm is very simple: Path  $p_i^{xj}$  is chosen if  $p_i^{xj} \cdot (\cos t) = \min\{p_i^{nj} \cdot (\cos t) | n \in N_i\}$ .

Fig. 2 illustrates the way in which the path selection algorithm operates. Fig. 2(a) shows the network topology formed by combining paths advertised by neighbors b, c, and d. Node a has active flows with nodes g, f, and k and, therefore, needs to set up routes for them. For destination g, the advertised paths are aceq and adq, of which the path adq is chosen because it has a smaller cost. Similarly, among the paths advertised for destination k, namely, ab fik and aceqik, path ab fik is chosen. For destination f, the only path available is adf. Note, abf is also a plausible path to reach node f. However, in OLIVE routes for a destination are installed using paths explicitly advertised by neighbors and plausible paths are used if there is no such explicit path and the plausible path have been verified to be valid. Using only complete paths reported by neighbors avoids packet forwarding to nodes with no paths to destinations, the problem discussed in the previous section in relation to on-demand link-state routing. As shown in Fig. 2(b), the aggregate of the selected paths form the labeled source graph at node a.

To reduce the overhead incurred with RREQs, a node makes use of *plausible paths*. A plausible path through a neighbor is

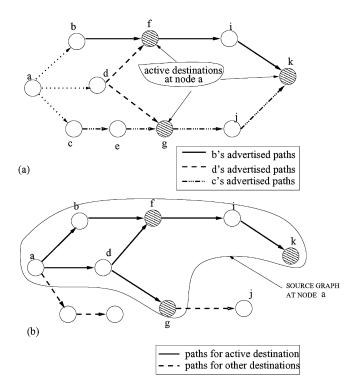


Fig. 2. Showing how path selections are done in OLIVE.

either: 1) a nonactive path advertised by the neighbor or 2) a path that has not been advertised by the neighbor but that can be computed from the topology using a local path-selection algorithm. For example in Fig. 2, no path has been advertised for nonactive destination j at node a. However, the plausible path adqj can be computed using Dijkstra's SPF algorithm, which comprises links that belong to paths advertised for other destinations g and k. The plausible paths are always validated using control messages before the actual data packet forwarding. When a node needs to change or create a route for a destination for which there is no active path in the labeled source graph, the node looks for plausible paths to the destination. Before a node installs in its routing table a route corresponding to a plausible path, it unicasts a forced route request (FRREQ) to the next hop of the path to verify that the path exists. FRREQs are forwarded by nodes in the plausible path till it reaches a node which has an active path for the destination or is the destination itself. This node sends the forced route reply (FRREP) and FRREPs containing the current information about the route traverse the reverse path to the source of FRREQs. When the originator of a FRREQ receives a reply, an alternate path to a destination is selected only if it has the same or lower cost than the path that the node announced to its predecessors. Every node maintains a list of predecessors for each active destination and if the node has no predecessors for a destination, it can pick an alternate path of any length. In case no alternate path is possible, the predecessors are informed of route failures through router error (RERR) packets sent reliably. Because each node either informs all its predecessors that it has lost its path to a destination through reliable RERRs, or picks alternate routes that are shorter than or equal to the route that it advertised to its predecessors, instantaneous loop-freedom can be maintained when every node knows the predecessors for active destinations. The elaborate proof has been given in Section III.

#### C. Detailed Description

The operation of OLIVE can be classified into three phases: 1) route discovery for setting up new paths; 2) local route repair for finding alternate paths when the original breaks; and 3) route failure notification for updating neighbors of route failures.

1) Route Discovery: Route requests (RREQs) are used to request for routes for unknown destinations. The route discovery process is always initiated by the source of data packets when it does not have active or plausible routes for a destination for which it has an active flow. RREQs by the source are either limited to neighbors or sent throughout the network with some limited scope. When a node receives a RREQ, it forward it only if: 1) it has no valid route for the destination; 2) no RREQ initiated by the same source has been forwarded recently; and 3) the RREQ has not traversed beyond the zone within which the route search has to be limited.

Route replies (RREP) are sent in response to RREQs and carry path information for a destination. To reduce duplicate RREPs, a RREP sent in response to a RREQ is broadcast to all neighbors. A node waits for a backoff time proportional to the node's distance from the target before sending a broadcast RREP, and deletes its scheduled RREP if it receives a broadcast RREP from another node in response to the same RREQ. RREPs are forwarded using unicast packets. Before a node forward a RREP to a neighbor, it ensures that the neighbor is part of the predecessor list for the destination of the original RREQ. Maintaining the correct up-to-date list of predecessors is the key to maintaining loop-freedom. Also, on receiving a RREP, a route is only installed for the destination of the original RREQs. The subsets of the paths advertised in RREPs are considered as plausible routes but not as valid routes for intermediate nodes.

An acknowledgment to RREP (RREPACK) signifies that the neighbor has selected the route through this node for data forwarding. Hence, if no RREPACK is received from a neighbor within a specified time period, it is removed from the predecessor list.

The lifetime for a route is *active\_route\_timeout*, while the lifetime of a predecessor entry is  $2 \times active\_route\_timeout$  and the entries get refreshed during data packet forwarding.

2) Local Route Repair: A route fails when a node detects a local link-failure or receives information from its successor about the failure of a downstream link. In such scenarios, the communication overhead can be reduced drastically by localizing the impact of route failures because if link failures are always advertised to predecessors, then the sources of the active flows have to reinitiate network-wide route discovery leading to high communication overhead. In OLIVE, local route repair is done using the plausible paths computed with the local link-state information. When the original path to a destination breaks, plausible paths of equal or lower cost than the cost of the original path are computed using the information in the network topology available locally. However, before data packets are sent over such paths, FRREQ are sent along those plausible paths to check their viability.

Fig. 4 describes how FRREQs are handled by nodes. Each FRREQ carries information about the plausible path to the destination, such that each node on the path having an active route

can compare that path with its current route for the destination. A node responds with a forced route reply (FRREP), under two conditions: 1) when it contains a valid route for the destination (steps 9 and 13) and 2) contains no route (step 18) and the path specified in the FRREQ is not correct. If a FRREP carries no path information, it specifies the first link in the plausible path of the FRREQ that has infinite cost such that the original sender can update its topology. A node forward the FRREQ along the path specified in the control packet, if: 1) it has no valid route for the destination, and the next hop in the advertised path of FRREQ is still reachable (step 16) or 2) the valid route is longer than the path advertised in the FRREQ, and the next hop matches the successor as specified in the tail of the path of FRREQ (step 11). The second condition is an optimization which ensures that FRREQs are sent nearer to the destination and most up-to-date path information is sent back to the original sender of FRREQ. When the originator of a FRREQ receives a reply, an alternate path to a destination is selected only if it has the same or lower cost than the path that the node announced to its predecessors. When FRREQs do not yield any response within a certain time interval, the alternate paths are assumed to be nonexistent and route errors (RERRs) are sent to

3) Route Failure Notification: Route errors (RERR) and RERR acknowledgments (RERRACK) are used for reliable transfer of route failure information and updating predecessor lists. When a node detects a failure of an active path (either directly by detecting a broken link or indirectly by receiving an RERR from a downstream node), the node generates a RERR only if it does not have any alternate path to the destination of equal or lower cost.

A node sends a RERRACK in response to a RERR to notify the sender of the RERR that it has stopped using the node as a successor to a given destination. The sender of a RERRACK is no longer considered a predecessor to the destination indicated in the RERR. This route failure notification mechanism ensures that either the actual path used for data forwarding is not longer than the path known to a node or the node has the most up-todate view of the path.

There is also a special case, where if the received RREP specifies a path with outdated links, RERR is sent to the sender of RREP.

#### D. Neighbor Relationship

A node considers the link to a neighbor to be up when any one of the following conditions happens: 1) The node receives a control packet from the neighbor for the first time; 2) it receives the first network-layer hello message from the neighbor; or 3) a neighbor protocol at the link layer advertises the presence of a new neighbor.

A node decides that its link with a neighbor is down when any of the following conditions is true: 1) the router receives link-layer notification caused by the failure to deliver data packets across the link; 2) network-level hello messages are missed several consecutive times; 3) the link layer declares the link to be down either through the action of a neighbor protocol, or after several retransmissions; 4) acknowledgments for data packets are not received after repeated network-level retransmissions;

and 5) no data or control packet has been received for a certain time interval.

If no network-layer hello mechanism is available for neighbor discovery, and a neighbor is silent for a certain time (i.e., the router has not received any data packet or control packet from it for that period of time), in OLIVE a neighbor is assumed to be down and all link entries advertised by it are declared to be invalid.

#### E. Handling Link Sequence Numbers

Each link has a sequence number which is used to resolve conflicts in case of conflicting link-state information. Only the head node of a link can assign or change the sequence number of the link and reports of links with higher sequence numbers are trusted over reports of the same links with lower sequence numbers. If there is no entry for a link, then the router trusts the first link-state entry that it receives. Link sequence numbers can be avoided by having nodes trust the link-state value reported by the neighbor with the shortest path to the head of the link.

#### F. Example of OLIVE Operation

Fig. 3(a) shows an ad hoc network of seven nodes. Let us assume that node i has data packets for node j and has to set up a route for it, and only nodes a and b have valid routes for node j. Therefore, when node i sends a single-hop RREQ for j to its neighbors, it receives no RREP. Then, it sends a network-wide RREQ, as shown in Fig. 3(a). Fig. 3(b) shows how the RREPs and the RREPACKs are exchanged between routers for the path setup from node i to node j. When nodes a and b receive a RREQ, they responds with a RREP containing a path for node j. Node d receives node a's RREP, while node c receives node b's RREP. When node a sends its RREP in response to node d's RREQ, reporting its active route for node j, it includes node din its predecessor list for destination j, such that node d can be notified when the route for node j changes. When node d selects its route for node j through node a, it sends a RREPACK to node a. Similar messages are also exchanged between nodes c and b. At node e, RREPs with paths to node j are received from nodes c and d. Both paths are of equal cost; therefore, node e accepts the route it first receives.

Let us assume that node e hears the RREP from node e first. Then, node e sends a RREPACK to node e only and not to node e. When node e does not receive any RREPACK from node e within a certain time interval, it removes node e from its list of predecessors. An entry for the advertised route of neighbor e will be maintained at node e and this information will be helpful to set up plausible paths when the original route breaks. Node e sends a RREP to node e, which sends a RREPACK back to node e after it selects the final path eebj.

Fig. 3(c) and (d) illustrates how alternate paths are created when original paths break. Assume that link (e,c) fails. Because node e's successor for destination j (i.e., node e) is not reachable, node e removes the route for node j from its routing table. It then attempts a local route repair based on its link-state information. Node e computes the plausible path edaj for destination j using Dijkstra's SPF algorithm. However, node e is not the current predecessor of node e for destination e from node e is might not receive updates regarding destination e from node e.

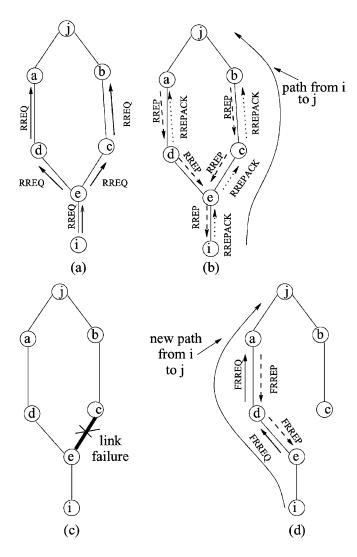


Fig. 3. Showing the route discovery and route repair methods in OLIVE.

```
Procedure processFRREO @ node i for destination j
        adv\_dist = distance from i to destination j according to the path in FRREQ
        dist = distance of the route installed at i for destination j
2
        dist = INFINITY if there is no route
        if (dist \neq INFINITY)
4
             nhop = successor in i's route for destination j.
6
        adv_next_hop = next hop at node i to j according to the path in FRREQ
        if (dist \neq INFINITY AND dist > adv_dist)
8
             if (nhop≠ adv_next_hop)
                  send FRREP with its path
10
               else
                   send FRREQ to nhop
11
12
         if (dist \neq INFINITY AND dist \leq adv_dist)
13
              send FRREP
14
         if (dist = INFINITY)
15
              if (adv_next_hop exists)
16
                   forward FRREO to adv_next_hop
17
                   send FRREP with no path
18
```

Fig. 4. Procedure: Processing of FRREQ.

Hence, to verify the plausible path node e sends a FRREQ, which is forwarded along the path edaj, and node a, which has an active route to j (assuming the active route at node d has timed out), replies with a FRREP, which traces back from a to d, and then to node e, setting up the new route edaj. If the route set up at node d would have not timed out, node d would have

also responded with a FRREP, containing the required active route.

#### G. Critique of OLIVE With Respect to Other Routing Protocols

Several major features of OLIVE address many important shortcomings of the on-demand routing protocols (DSR [1], AODV [2]) and proactive routing protocols (TBRPF [7], OLSR [8]) that have been proposed for standardization in the IETF working group on mobile ad hoc networks (MANET). Here, we describe some of those features and try to explain whether we can achieve those with the existing routing protocols.

- OLIVE ensures instantaneous loop-freedom when nodes maintain consistent neighbor relationship information by keeping an up-to-date predecessor list and checking the validity of any path before using it. OLIVE supports hop-by-hop data packet forwarding. DSR ensures loop-freedom by adding extra overhead in data packets in the form of source routes or flow identifiers. AODV like OLIVE supports hop-by-hop packet forwarding. However, its use of destination-based sequence numbers for preventing loops leads to frequent route discovery and high control overhead. TBRPF and OLSR support hop-by-hop packet forwarding. However, they can both suffer from permanent and transient loops when control packets get lost, and all the necessary nodes do not have a consistent view of the network. Loops can be averted by checking the paths before usage, as done in OLIVE, but that would become expensive for TBRPF and OLSR, in which paths need to be maintained for all destinations all the time.
- When an active path breaks in OLIVE, any intermediate node can select an alternate path without informing the predecessors. That would make the number of RERRs depend on the number of active destinations only and not both on number of active destinations and sources of active flows as happens in DSR and AODV. The number of control packets in TBRPF and OLSR due to network changes is only dependent on the number of nodes in network and does not depend on the number of active flows.
- Selection of old defunct paths are always averted in OLIVE by ensuring the viability of any path before using it for data packet forwarding. This would reduce the wasteful usage of bandwidth when the data packets are routed along wrong paths before being finally discarded. DSR, OLSR, and TBRPF do not have any mechanism to ensure the viability of alternate paths and, hence, suffer from usage of invalid paths. Use of timers per link or per path can only improve the situation though in a nonadaptive and nonscalable manner. In AODV, due to automatic increment of destination sequence number after route failures and rediscovery of new routes, correct paths are mostly chosen. However, as we also see in Section IV, this process consumes high bandwidth because often valid paths with lower cost are not considered in path computation.

SOAR [4] shares the same philosophy as OLIVE and uses on-demand link-state information to create routing tables.

However, in terms of operation they differ considerably. OLIVE exchanges path information and uses the links in the path to create the partial network topology. On the contrary, SOAR uses source trees as the basis of routing information exchange, and uses the path traversed information in the data packets to prevent permanent loops. However, SOAR cannot avert transient loops. Since our main aim is to come up with a routing solution that ensures instantaneous loop freedom, supports hop-by-hop packet forwarding and does not incur control overhead in data packets, we have not included SOAR in our subsequent analysis. In our performance analysis, we have considered DSR, AODV, TBRPF, or OLSR to compare against OLIVE to investigate the suitability of OLIVE as an alternate routing approach to the established routing protocols.

#### III. OLIVE CORRECTNESS

The following two conditions have to be satisfied for OLIVE to be correct.

Safety Property: Given a network G = (V, E), and a destination  $j \in V$ , the successor graph for destination  $j\mathbf{S}\mathbf{G}_j = (V, E')$ , where  $E' = \{(i, s_i^j) | i \in V \text{ and } s_i^j \text{ is the successor for node } j \text{ at node } i\}$ , is a directed acyclic graph at every instant.

Liveness Property: Within a finite time following an arbitrary number of changes in network conditions and flows, all nodes in the network have correct paths for each reachable destination, to which they have active flows.

The above conditions leave open the possibility of nodes trying to find persistently paths to destinations belonging to the partitioned network. In practice, the routing protocol can infer that a destination appears to be unreachable after a few failed attempts, and it is up to the higher level protocol or application to determine whether or not to continue looking for paths to unreachable destinations.

To show that OLIVE is correct, we assume that all information is stored correctly and routers operate according to the specifications of OLIVE. Also, changes in status of adjacent links are notified within a finite time and the links are bidirectional. Furthermore, Theorem 1 assumes that, after an arbitrary sequence of link cost changes, topology changes, and traffic flow changes, there is a sufficiently long time for OLIVE to stop computing routes to active destinations.

The instantaneous loop freedom in OLIVE can be proved based on its two important properties. First, when a node needs to adopt a new route of higher cost than its previous route, the node reports an infinite distance to its predecessors and adopts its new route only after its predecessors have removed it as their successor. Second, a node sets up a route for a destination only if the selection of the new route results from a path specified in a FRREP or RREP.

The above means that, at any instant of time t if  $\mathrm{PRED}_i^j(t)$  is the set of predecessors for destination j as known to node i and  $\overline{\mathrm{PRED}}_i^j(t)$  is the set of nodes at time t who have selected node i as successor to reach node j according to an omniscient observer, then  $\overline{\mathrm{PRED}}_i^j(t) \subseteq \mathrm{PRED}_i^j(t)$ .

We can show by contradiction that loops can never form in OLIVE if the above two conditions are always satisfied.

Let  $a, b, c, \ldots, x$  be the nodes that form a loop for destination j. Let the next hop for node x be node a, the next hop for node a be node b, and so on. Let  $P_i^j(t)$  be the path at node i for destination j at time t. Let us assume that node a is the node in the loop whose length of the path for node j is maximum at time t. Therefore,  $P_a^j(t) \cdot (\cos t) \geq P_x^j(t) \cdot (\cos t)$ . Assume that the last change in successor occurred at time t' < t, when node x chooses node a as the next hop. Therefore,  $P_x^j(t) \cdot (\cos t) =$  $P_x^j(t') \cdot (\cos t) > P_a^j(t'') \cdot (\cos t)$ , where  $P_a^j(t'') \cdot (\cos t)$  is the cost of the path advertised by a in its RREP or FRREP at time t'' < t' < t. This implies that  $P_a^j(t) \cdot (\cos t) \ge P_x^j(t) \cdot (\cos t) =$  $P_x^j(t') \cdot (\cos t) > P_a^j(t'') \cdot (\cos t)$ , which means that node a has obtained a new path to node j in the interval (t''t] and the cost of the new path is greater than the cost of the previous path. According to the specifications of OLIVE, a node sends a RERR, when its current path breaks and there is no alternate path of equal or lower cost. In that case, node a must send a RERR to node x advertising an infinite cost, and node x must, therefore, release node a as its next hop at time  $t_x < t$ . Hence, a loop cannot form, which contradicts our assumption that there is a loop at time t.

Loop freedom in OLIVE relies on the reliable exchange of RERRs between a node and its predecessor for a destination. However, when the link between a node x and its predecessor y for a given destination fails due to mobility or other effects, node x must avoid accepting a new route to the destination before node y has stopped using x as a successor. To be safe, node x must apply a hold-down time after detecting a broken link to node y, such that no packets to the destination for which y is a predecessor are forwarded until the hold-down time expires. The length of the hold-down time that node x must apply depends on how quickly nodes can detect link changes. If a neighbor protocol exists at the link layer, predecessor and successor can detect the broken link very quickly. However, detecting a broken link based on acknowledgments to data packets at the network layer can take much longer due to queuing delays.

Theorem 1: Within a finite time after the last change in network conditions and traffic flows, all nodes which are sources of data packets have correct paths to the destinations.

Summary of Proof: As described above, OLIVE ensures that the successor graph as visible to an omniscient observer  $\mathbf{SG}_j$  does not contain any transient loop. Let  $S_j \subset V$  be the set of sources that have active flows for destination j, and  $R_j \subseteq V$  be the set of nodes acting as relays. Let  $t_1$  be a time when  $\mathrm{SG}_j$  is correct and loop-free, and assume an arbitrary but finite sequence of link-state and traffic-flow changes. Let  $t_2$  be the time when the last link or traffic-flow change occurs. The proof needs to show that the routing tables for destination j at every node converge to correct values within a finite time after  $t_2$ . In this context, the active graph  $\mathbf{AG}_j$  is formed by the links  $(v, s_v^j)$  for each  $v \in S_j$  or  $v \in R_j$ , where  $s_v^j$  is the successor for node j at node v. Because  $\mathbf{AG}_j \subseteq \mathbf{SG}_j$ ,  $\mathbf{AG}_j$  is guaranteed to be a DAG.

Assume that the theorem is not true, then  $\mathbf{AG}_j$  must be a forest, because OLIVE is loop free. We need to consider two cases: a source node located in the same connected component as j, and a node with no physical path to node j. We note that link sequence numbers cannot change after time  $t_2$ .

Consider first the case of a source node in the same connected component as j. A source node  $v \in S_j$  with an active flow for j in a connected component containing node j must either: 1) have a path that reaches j; 2) have a path that ends in a node with no path for j; or 3) not have a path for j.

By induction on the number of hops away from the head of any given link, all the links in a path to j reported in a RREP sent within a finite time after time  $t_2$  are assigned their last sequence number, given that OLIVE requires a node to correct a neighbor advertising an older link sequence number. Given that RREQ's, RREP's, FRREQ's, and FRREP's traverse loop-free paths, it follows that a source node with a path to j must have a correct path after some finite time  $t > t_2$ .

Consider a node a that has no path to j at time  $t > t_2$ . If node b uses node a as successor to reach j, it implies that node a has reported a path of finite distance for node j to node b and has node b as a predecessor for the route to node j. Hence, node a must have had an active path to node j and it must have reported the loss of its path to node b through reliable RERRs a finite time after  $t_2$ , and node b must receive such RERRs, because no more link changes can happen after time  $t_2$ . Accordingly, node b must stop using node a as its next hop within a finite time after  $t_2$ . By induction on the number of hops away from any node without a path to j, it is easy to show that any source node a a valid path to a0.

A source node s that never attains a valid path to j after time  $t_2$  must generate RREQs an infinite number of times. However, the connected component where j and s are located is finite and RREQ's and RREP's traverse loop-free paths only, because they specify the paths they traverse. Furthermore, as pointed out before, a finite time after  $t_2$  any link communicated in a RREP must have its final sequence number (and correct state). Hence, because no link changes can occur after time  $t_2$ , source s must receive some of the RREPs being generated and they must contain correct paths. Therefore, node s cannot generate an infinite number of RREQs after time  $t_2$ .

A similar approach can be used to show that, within a finite time after time  $t_2$ , no source node in a different component than node j's component can have an active path to j. Q.E.D

#### IV. PERFORMANCE COMPARISON

We have compared the performance of OLIVE with the on-demand routing protocols (DSR [1], AODV [2]) and proactive routing protocols (TBRPF [7], OLSR [8]) that have been proposed for standardization in the IETF working group on mobile ad hoc networks (MANET). The performance evaluation has been done in the ns2 simulation platform [9], using the code of DSR, AODV, and TBRPF provided with the simulator. The TBRPF code conforms to version 4 of IETF draft of TBRPF. For OLSR, we have used the code available from INRIA [10] and have added the code for handling link-layer notifications of adjacent link failures. The specifications of the OLSR code match those in version 3 of OLSR IETF draft. The constants used for DSR, AODV, and TBRPF are the same as in the original code, while the constants given in Table I have

TABLE I CONSTANTS OF OLIVE

Constants	Value
active_route_timeout	5.0 s
predecessor_lifetime	2 × active_route_timeout
olive_rreq_gap_time	0.4 s
olive_rrep_gap_time	0.4 s
rerr_ack_wait	0.5 s
reply_ack_wait	1 s
one_hop_traversal	0.1 s
frreq_time_out_value	2×one_hop_traversal × tot_hops

been used for OLIVE. The AODV and DSR codes conform to their latest implementations available in ns2 [9].

The link-layer implements the IEEE 802.11 distributed coordination function (DCF) for wireless local area networks (LANs). The broadcast packets are sent unreliably and are prone to collisions. The physical layer approximates a 2-Mb/s DSSS radio interface The radio range is 250 m and for all the simulations the run length is 600 s.

TBRPF, DSR, AODV, OLSR, and OLIVE use link-layer indications about the failure of links when data packets cannot be delivered along a particular link. Except for the notification of the link-layer about links going down, none of the protocols has any other interaction with the lower layer. In particular, promiscuous listening was disabled for both DSR and OLIVE. ARP has also been disabled and, for the sake of simplicity, the IP addresses of the nodes are used as the MAC addresses.

For our simulations, we have 50 nodes moving over a rectangular area of  $1500 \text{ m} \times 300 \text{ m}$ . The movement of the nodes in the simulation is according to the random waypoint model [11]. Values of *pause time* used are 0, 15, 30, 60, 120, and 300 s.

Each flow is a peer-to-peer constant bit-rate (CBR) flow and the data packet size is kept constant at 512 bytes. Each flow continues for 200 s and after the termination of the flow, within 1 s, the source randomly chooses another destination and starts another flow, which again lasts for 200 s.

The following six metrics are used to compare the performance of the routing protocols.

- Packet delivery ratio: The ratio between the number of packets sent out by the sender application and the number of packets correctly received by the target destinations.
- Control packet overhead: The total number of control packets sent out during the simulation. Each broadcast packet is counted as a single packet.
- *Control byte overhead*: The total number of control bytes used in the control packets.
- *Total number of MAC packets*: The total number of packets sent at the link-layer for exchange of routing information. They include RTS, CTS, the control packets and the ACKs.
- *Optimality of paths*: Ratio of the actual number of hops to the optimal number of hops possible based on the given topology.
- Average end-to-end delay: The end-to-end delay measures
  the delay a packet suffers after leaving the sender and then
  arriving at the receiver application. This includes delays due
  to route discovery, queueing at Internet protocol (IP) and
  medium access control (MAC) layers, and propagation in the
  channel.

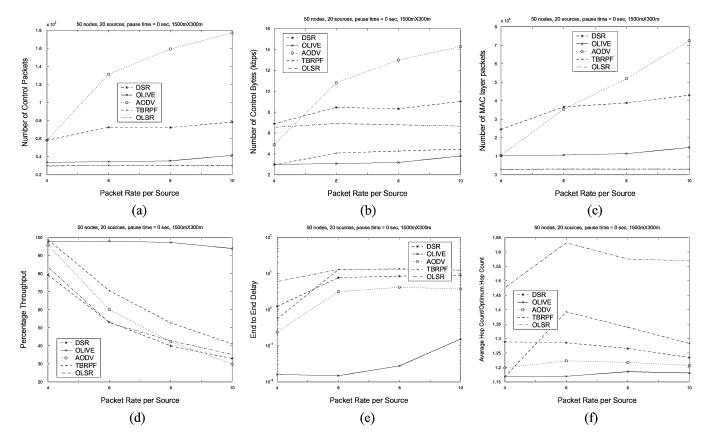


Fig. 5. Performance in a 50-node network with 0 s pause time and 20 sources with varying packet load. (a) Control packets. (b) Control bytes. (c) MAC layer control packets. (d) Throughput. (e) End-to-end delay. (f) Optimality of paths.

#### A. Effect of Traffic Load

Fig. 5 shows the results of each protocol under varying packet loads when the number of sources is 20 and nodes are constantly moving.

The control overhead of OLIVE remains unchanged with the increase of load. DSR, TBRPF, and OLSR exhibit similar behavior. However, AODVs control overhead increases with load. Even when nodes are physically close, links are assumed to fail when data packets cannot be delivered along those links. Such perceived link failures affect AODV drastically, because it generates network-wide RREQs with increased sequence numbers that in many cases have to be resolved by the destinations. The number of MAC-level control packets increases slightly for DSR throughout all scenarios, while for OLIVE it increases only when the network load is maximum (41 kb/s/source or 10 packets/s/source). This is because of the increased number of broadcast RTSs, for which there is no collision avoidance mechanism. Hello packets form the major percentage of packets for TBRPF or OLSR, and their control overheads remain constant.

In terms of application-oriented metrics, like throughput or network delay [Fig. 5(d) and (e)], the performance of OLIVE is the best and is almost unaffected by network load. On the other hand, the performance of DSR and AODV degrade significantly. This can be attributed to the opportunistic use of plausible paths in OLIVE. In TBRPF, DSR, or OLSR, when original paths break, packets are rescheduled along alternate paths, without ascertaining their feasibility. This leads to higher waiting times in queues and more congestion. In contrast,

OLIVE uses FRREQs and FRREPs when active routes are broken to test the viability of alternate paths, and data packets are forwarded only if new paths are usable.

OLIVE has a high range of delay values, like DSR, TBRPF, OLSR, or AODV, which implies that data packets in OLIVE also have long waiting times at the link-layer interface. However, its 95 percentile delay value is far lower than that of DSR, OLSR, AODV, or TBRPF, which shows that it has better average delay performance. Network topology information in OLIVE also helps in finding shorter paths [Fig. 5(f)].

For TBRPF, data packets are always rescheduled along alternate routes, when the original paths break. Therefore, they get circulated throughout the network and there is considerable packet loss due to looping and TTL time-out. In TBRPF, all the control packets are broadcast. Therefore, for sending any TBRPF control packet, no extra MAC layer handshake is necessary. However, because the broadcast TBRPF packets can lead to collisions, its performance degrades significantly with load. the same is the case for OLSR, whose proactive mechanism of route maintenance is almost similar to that of TBRPF. This is also true for AODV, where broadcast RREQs constitute the majority of its signaling.

#### B. Effect of Mobility

Figs. 6 and 7 show the performance of the protocols under varying mobility of network nodes for a 50-node network with 10 sources and 20 sources, respectively, with each source generating packets at the rate of 4 packets/s.

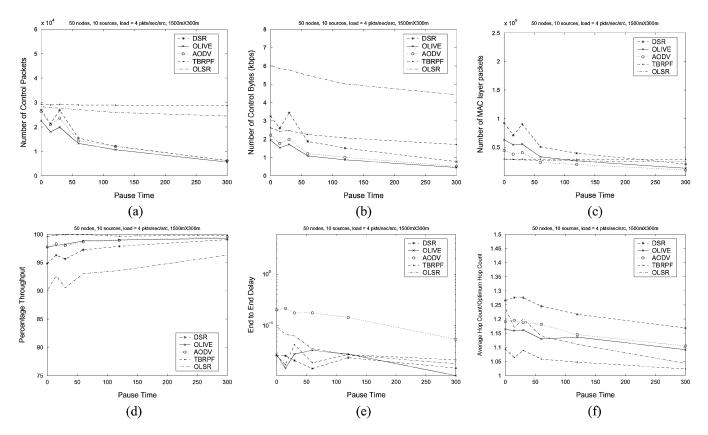


Fig. 6. Performance in a 50-node network with load of 4 packets/s/source and ten sources under varying mobility. (a) Control packets. (b) Control bytes. (c) MAC layer control packets. (d) Throughput. (e) End-to-end delay. (f) Optimality of paths.

Control overhead in OLIVE, DSR, and AODV decreases with lesser node mobility. Higher mobility implies higher rate of route failure leading to higher control overhead. For TBRPF and OLSR, the control overhead remains almost unchanged with mobility, because they rely on periodic hello packets sent independently of the reliability of links. For 10 and 20 sources, the control overhead of OLIVE in terms of both bytes and packets is less than DSR, AODV, TBRPF, or OLSR.

Again, because the majority of control packets in AODV is broadcast RREQs that do not use ready-to-send/clear-to-send (RTS/CTS) handshakes, the difference between AODV and OLIVE in terms of MAC-layer control packets [Figs. 6(c) and 7(c)] is not as significant as the difference in the number of network-level control packets.

In general, when the number of sources is ten, the on-demand routing protocols use fewer network-layer control packets compared with proactive routing protocols, while for the high mobility scenarios with higher number of sources proactive routing protocols start performing better. In all scenarios, OLIVE has less control overhead than DSR or AODV.

The number of data packets delivered is similar in OLIVE, TBRPF, and AODV [Figs. 6(d) and 7(d)]. DSR suffers a higher loss of data packets in all scenarios, because of the use of stale routing information in the RREPs. OLSR suffers considerable loss of data packets due to routing loops and TTL timeouts.

In terms of path optimality [Figs. 6(f) and 7(f)], OLIVE is best among all the on-demand routing protocols. In all our experiments, we have found that the paths in DSR tend to be longer than the paths in AODV, contrary to the results in [11] and [12].

The reason is likely to be the absence of promiscuous listening in DSR.

In terms of delay [Figs. 6(e) and 7(e)], OLIVE performs better or equal to the other protocols. Queueing at the link layer is the main cause for the delay experienced by data packets in each of the routing protocols.

#### C. Bandwidth Lost to Routing Loops

We quantify the amount of bandwidth wasted in each routing protocol due to packets going in loops or staying in the network for a considerable time. The experiment is done for low to heavy load scenarios when the number of sources is 20.

Fig. 8 shows the number of packets that have been either sent along a loop, or dropped due to time-to-live (TTL) timeouts or loop detection. Loops can be detected in two ways in any routing protocol in which no traversed path information is present in the packet headers: 1) the source finds that the data packet has come back to it and 2) a forwarding node detects that it is passing the packet to a node that has actually forwarded the data packet.

In our experiments, packet traces are used to detect the number of packets that have gone in loops. From Fig. 8, we see that bandwidth is wasted in TBRPF and OLSR due to looping and the effect becomes more pronounced at heavier loads. This is because neither OLSR nor TBRPF ensures instantaneous loop freedom, and exchange unreliable control packets that renders longer convergence times. These two routing protocols have very few packets dropped due to nonavailability of routes, which implies that the topology information always helps data packets to be rescheduled along alternate paths. However, in

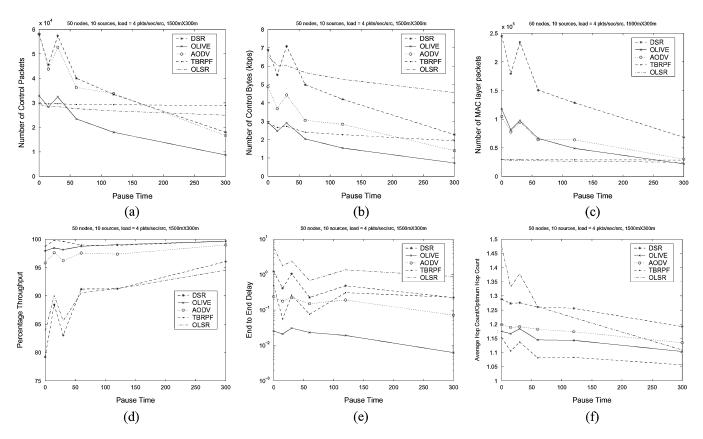


Fig. 7. Performance in a 50-node network with load of 4 packets/s/source and 20 sources under varying mobility. (a) Control packets. (b) Control bytes. (c) MAC layer control packets. (d) Throughput. (e) End-to-end delay. (f) Optimality of paths.

heavy-load scenarios, when the links fail frequently due to congestion, alternate paths are not always the correct choices.

The control packet exchanges in OLIVE and AODV ensure instantaneous loop freedom. In DSR, the source routes carry information about the path traversed and the path to be traversed; therefore, loops can be easily detected. Under high load, some data packets go into loops in DSR when data packets are salvaged at intermediate nodes. When an intermediate node in DSR finds that the next link in the source route is no longer available, it salvages the data packets by rerouting the packet using its own cached routing information. Because path traversal information is not checked for rerouting, loops can form.

Though AODV and OLIVE maintain instantaneous loop freedom of routing tables in the simulations we ran, looping of packets can still occur during the transient states of the routing tables due to inconsistent views of neighboring links. However, this effect is not persistent, and is unavoidable in order to deliver packets to their destinations.

#### V. CONCLUSION

We have presented the on-demand link-vector (OLIVE) protocol, which is the first protocol to ensure loop-freedom at every instant using link-state information on demand, while allowing destination-based hop-by-hop routing instead of requiring source-routed data packets.

We have shown that selecting paths on-demand cannot be approached based on the "source trees" used in proactive routing

protocols. Therefore, routers in OLIVE exchange path information and these paths combine to give a partial network topology. A path selection algorithm is then run to compute paths to destinations and these paths aggregate to form source graphs. Source graphs are reported incrementally in the form of separate paths.

OLIVE has been shown to be loop-free at every instant and to find correct paths to destinations in finite time. Loop-freedom is attained in OLIVE by ensuring that a node always knows its predecessors for an active destination, and by allowing localized route repairs using alternate paths whose length is not longer than the paths announced to its predecessors for the destination. To ensure that routes are correctly installed corresponding to alternate paths derived from old information, a node unicasts RREQs along the alternate paths, so that validity can be verified by a RREP.

Our simulation results show that OLIVE performs much better than the routing protocols being discussed for standardization in the IETF MANET working group, in terms of control overhead, throughput, and delay.

#### APPENDIX

BUILDING A SHORTEST-PATH SOURCE TREE WITH ON-DEMAND ROUTING CONSTRAINT IS AN NP-COMPLETE PROBLEM

Building routes to destinations using a link-state routing protocol entails computing a source tree at a given node based on the topology information known to it. The topology consists of the node's adjacent links and the links advertised to it by its neighbors. Assuming that the identity of a neighbor

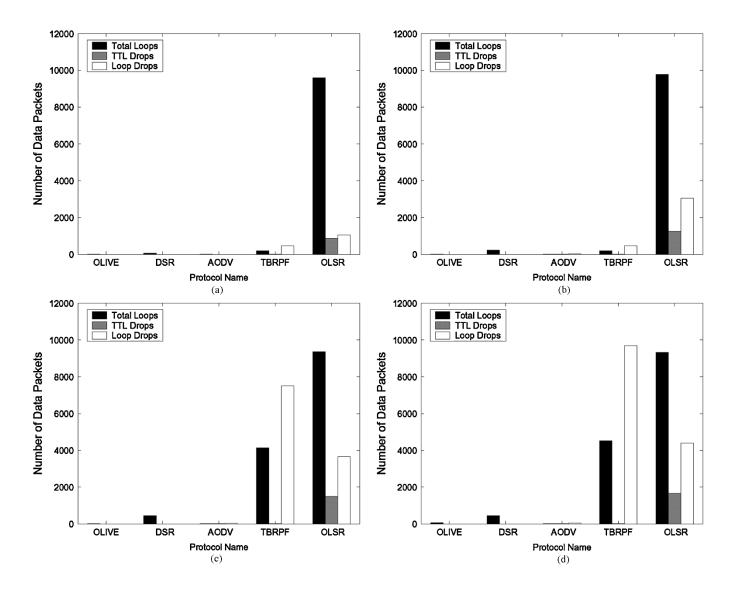


Fig. 8. Loops for a 50-node network with 20 sources under four different load conditions. (a) Rate = 4 packets/s. (b) Rate = 6 packets/s. (c) Rate = 8 packets/s. (d) Rate = 10 packets/s.

is a boolean variable (termed as label), a link in any node's topology graph will have a label set, which consists of labels corresponding to the neighbors which have advertised that link to that node. A label within a link's label set will be assigned the value "one," if the final path to any destination contains that link and passes through the neighbor which corresponds to that label. Therefore, for a link e, if the label set,  $S_e$  is  $\{l_1, l_2, l_3\}$ at any random node, the Boolean value assignment would be  $\{l_1 = 1, l_2 = 0, l_3 = 0\}$ , if the link is included in the final path to any destination through the neighbor which corresponds to label  $l_1$ . Labels  $l_2$  and  $l_3$  correspond to other neighbors and no path through these neighbors includes link e. If the link is in paths through multiple neighbors, then multiple labels in a label set would be assigned the value "one" and if the link is excluded from all paths, then the label set will have no label assigned the value "one."

Based on the previous Boolean assignment strategy, the problem of computing a source tree with on-demand routing constraint (termed as the OPT-TREE problem) can be defined as the following.

Question: What is the maximum-vertex source tree that can be formed at node i, such that the edges on the path from node i to any node v in that source tree will have the same label set to one?

This constraint on the label assignment on any path ensures that all links in the path through a neighbor have been advertised by the neighbor only and, hence, there would be no scope of wrong packet forwarding.

As the next step toward solving the OPT-TREE problem, we reframe the optimality problem of OPT-TREE as a decision problem (termed as k-TREE problem): Is there a source tree having k vertices and rooted at source i, such that edges on the path from node i to any vertex v will have the same label set to one? By varying the value of k from one to |V| and applying the solution of the k-TREE decision problem, we have the solution for the optimality problem. We will show that the k-TREE problem is an NP-complete problem and, therefore, the OPT-TREE problem is also an NP complete problem. As the first step to prove that the k-TREE problem is NP-complete, we first define the SPAN-TREE problem.

Instance: Graph, G = (V, E), a given node i, which can be termed as the source, a collection of labels C, and the set of labels,  $S_e$  corresponding to each edge  $e(S_e \subseteq C)$ .

Question: Can we construct a tree rooted at node i containing |V| nodes, such that the edges on the path from node i to any node v have the same label set to one?

We first show that SPAN – TREE  $\in$  NP. Let the source tree formed be G = (V', E'). We first verify V' = V, and then we verify that in the spanning tree formed, all the edges in the path from the source to any node will have the same label and this can done in O(|V|) time.

The next step is to prove that SPAN-TREE is NP-hard. For that purpose, the well-known 3-SAT problem [13], [14] has been reduced to SPAN-TREE problem. In the well-known 3-SAT problem, the question is to find whether a logical formula in the 3-CNF form is satisfiable or not. An example of a 3-SAT problem is how to assign values to the literals,  $x_1, x_2$ , and  $x_3$  such that the logical formula  $(x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3)$  is satisfiable, i.e., has an output equal to one.

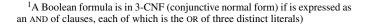
The reduction algorithm consists of three steps. For our purpose, we will show how the satisfiability of the above logical formula can be reduced to SPAN-TREE problem. The first step is to convert the logical formula  $(x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3)$  to the following form:  $(x_1 \lor \neg x_1 \lor y) \land (x_2 \lor \neg x_2 \lor y) \land (x_3 \lor \neg x_3 \lor y) \land (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3)$ , where y is a dummy literal which always assumes the boolean value zero.

The modified formula is essentially the same as the original formula because the first three clauses are always equal to one, irrespective of the values set for  $x_1, x_2$ , and  $x_3$ . This step of reduction is a trivial one, because it extends a logical formula by adding a clause corresponding to each literal  $x \in C$ , where each clause is  $(x \vee \neg x \vee 0)$  and C is the set of all literals. The first step of the reduction can be done in O(m) time, where m is the cardinality of the set C.

The modified logical formula is next represented in the form of a source graph. First, we create a node i that represents the root or the source and subsequently a new node is created for representing each of the clauses in the modified logical formula. For each such node, there are three incident edges and each edge has a label corresponding to each literal in that clause.

The graph is created stepwise by starting from the first clause and moving toward the right of the logical formula and representing each clause by a node. For each literal  $x_i$ , the last node  $v_{\rm last}$  which has been created with an edge having a label  $x_i$  incident on it, is remembered. If a new node  $v_{\rm new}$  is created for a new clause, one of whose literals is again  $x_i$ , an edge from  $v_{\rm last}$  to the new node  $v_{\rm new}$  is added and is assigned the label  $x_i$ . Also,  $v_{\rm new}$  becomes  $v_{\rm last}$ . This process makes the distance of node  $v_{\rm new}$  from node i always greater than the distance to node  $v_{\rm last}$  from node i if the path is traced from node i to node  $v_{\rm new}$  along the edges with label  $x_i$ .

Using the above algorithm, the graph shown in Fig. 9 has been created corresponding to the modified logical formula. Initially,



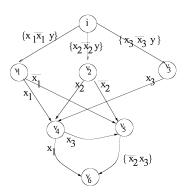


Fig. 9. Representation of the logical formula  $(x_1 \lor \neg x_1 \lor y) \land (x_2 \lor \neg x_2 \lor y) \land (x_3 \lor \neg x_3 \lor y) \land (x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3)$  in the form of a directed graph during second step for the reduction of 3-SAT problem to a SPAN-TREE problem.

 $v_{\rm last}$  is i for any literal. Nodes  $v_1, v_2$ , and  $v_3$  are joined to the source i by edges and they correspond to the new clauses that were added to the original logical formula. Node  $v_4$  is created for the fourth clause of the modified formula with three incident edges having labels  $x_1, x_2$ , and  $x_3$ , respectively, and these three edges, respectively, connect  $v_4$  with  $v_1, v_2$  and  $v_3$ . Then, node  $v_5$  is created,  $v_5$  joins node  $v_4$  by an edge with label  $x_3$  since  $v_4$  was the last node that has been created with an incident edge having a label  $x_3$ . Similarly, all the other edges and nodes are created. If there are multiple edges connecting the same two nodes, they are merged into one edge and the new edge will have the combined label set. This representation can be done in polynomial time by constructing (m+n) nodes and (3n+m) edges, where m is the number of literals and n is the number of clauses in the original 3-CNF formula.

The next step of the reduction algorithm is to replace each node (except those nodes directly connected to the source) in the above graph by three nodes each with three incident edges, each edge having a different set of labels. This step is shown in Fig. 10, where a node with three incident edges with labels  $x_1, x_2$ , and  $x_3$  is replaced by three nodes with the labels of the incident edges on each of the three nodes being, respectively,  $[\{\neg x_1\}, \{a\}, \{b\}], [\{x_2\}, \{b\}, \{c\}], \text{ and } [\{\neg x_3\}, \{c\}, \{d\}]. \text{ This }$ reduction is similar to the reduction algorithm used for the reduction of the 3-SAT problem to the 1in3-SAT problem [15].<sup>2</sup> The edges between nodes are made using the same logic as used for drawing edges between nodes in Fig. 9. In Fig. 11, we show the final representation of the graph. Note that the node  $v_4$  is replaced by three nodes  $(v_4^1, v_4^2, v_4^3)$  with modified incident edges and the labels of the three edges incident on the three nodes are, respectively,  $[\{\neg x_1\}, \{a_1\}, \{b_1\}], [\{x_2\}, \{b_1\}, \{c_1\}],$ and  $[\{\neg x_3\}, \{c_1\}, \{d_1\}].$ 

The next step is to show that there exists a solution to the 3-SAT problem if and only if there is a solution to SPAN-TREE, i.e., if and only if there exists a spanning tree in which all the edges in the path from the source to any node will have the same label. First, we show that if there exists a solution for 3-SAT there will also be a solution in SPAN-TREE. The steps of the proofs, presented here have been motivated by the proofs

<sup>&</sup>lt;sup>2</sup>In 1in3-SAT, the problem is to find whether a logical formula in the 3-CNF form is satisfiable or not if exactly one literal in a clause can be set to one.

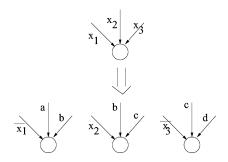


Fig. 10. Final step in the representation of 3–CNF formula in the form of a graph, where each node of the graph of Fig. 9 is replaced by three nodes (associated with each edge, its label set has been shown).

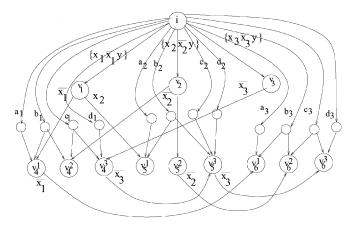


Fig. 11. Final representation of the 3-CNF logical formula after all three steps of reduction.

given in [15]. If the 3-SAT problem has a solution, then the logical formula is satisfiable and each clause should be equal to one and at least one literal of each clause must be equal to one. As described earlier a node that represents a clause  $(x_1 \lor$  $x_2 \vee x_3$ ) is replaced by three nodes with the labels of the incident edges on each of the three nodes being, respectively,  $[\{\neg x_1\}, \{a\}, \{b\}], [\{x_2\}, \{b\}, \{c\}] \text{ and } [\{\neg x_3\}, \{c\}, \{d\}].$  We now show that, depending on the values of  $x_1, x_2$ , and  $x_3$  for the clause  $(x_1 \lor x_2 \lor x_3)$ , we can have different Boolean assignments for labels a, b, c, and d, based on which if there exists a solution for the 3-SAT problem, we can have a solution for SPAN-TREE problem. Let us assume that  $x_2 = 1$  in the clause  $\{x_1, x_2, x_3\}$ . In that case, following assignment for the labels a, b, c, and d leads to a solution for the SPAN-TREE problem:  $a = x_1, b = 0, c = 0, d = x_3$ . Using such an assignment, the nodes representing clauses  $(\neg x_1 \lor a \lor b), (x_2 \lor b \lor c)$ , and  $(\neg x_3 \lor c \lor d)$  will have exactly one edge incident on each of them with each edge having one label set to one. Therefore, given that  $x_2 = 1$ , by using appropriate values for a, b, c, and d, if there exists a solution for 3-SAT, every node in the final graph has exactly one incident edge whose exactly one label has been set to one, which implies that all nodes in the final graph will form a spanning tree. Given that, there is exactly one incident edge on a vertex with exactly one label set to one, there can be no cycle, because in that case for at least one node (excluding the source

which has no incident edge), among all labels for all its incident edges there have to be at least two labels which are set to one, which cannot be true. Moreover, a forest cannot be formed because apart from the source node i, each node represents a clause of a satisfiable logical formula and, therefore, each node must have exactly one incident edge with exactly one label set to one.

Let us assume that  $x_2 = 0$ . In that case, there are three possibilities for the values assigned to  $x_1$  and  $x_3$  and they are: 1)  $x_1 = 1$  and  $x_3 = 1$ ; 2)  $x_1 = 1$  and  $x_3 = 0$ ; and 3)  $x_1 = 0$  and  $x_3 = 1$ .

To get a spanning-tree solution:

- Condition 1) leads to a = c = 0, d = 1, b = 1.
- Condition 2) leads to b = 1, a = c = d = 0.
- Condition 3) leads to a = b = d = 0, c = 1.

The case of  $(x_1 = 0 \text{ and } x_3 = 0)$  is not possible because at least one literal in each clause has to be equal to one.

The next step is to show that, if there exists a solution for the SPAN-TREE problem, there exists also a solution for the 3-SAT problem. The proof is by contradiction. Let us assume that, although there exists a solution to the SPAN-TREE problem, there exists no solution for the 3-SAT problem. For such case,  $x_1 = x_2 = x_3$  each is equal to zero for at least one clause,  $(x_1 \lor x_3 \lor x_2)$ . Then, a = 0, d = 0. Also, b = 0 or c = 0 (but not both b and c). However, if either one of these two literals, b or c is equal to one, then the condition that in the spanning tree there is only one incident edge with exactly one label set to one is no more satisfied. For the first node [that represents  $(\neg x_1 \lor a \lor b)$ ] or the third node [that represents  $(x_2 \lor b \lor c)$ ], we will get two incident edges that will have labels set to one, which is a contradiction to the original assumption that a spanning tree has been formed in which case each node can have exactly one label set to one among all labels for all of its incident edges. Therefore, the 3-SAT problem has a solution.

We have, thus, shown that the SPAN-TREE problem is NP-hard and that  $SPAN-TREE \in NP$ , which implies that SPAN-TREE is an NP-complete problem.

Now, the next step is to show whether the k-TREE problem is an NP complete problem. This can be a proof by restriction [14]. The SPAN-TREE problem is a restricted case of k-TREE problem, where the value of k is |V|.

Therefore, the OPT-TREE problem, i.e., the problem of finding the source tree with the optimal number of nodes with on-demand routing constraints is an NP-complete problem, because the decision problem to which the optimality problem has been reframed is an NP-complete problem.

#### REFERENCES

- D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*, T. Imielinski and H Korth, Eds. Norwell, MA: Kluwer, 1996, vol. 353.
- [2] C. E. Perkins, E. M. Royer, and S. R. Das. (2003, Feb.) Ad hoc on-demand distance vector (AODV) routing. Internet Draft, Mobile Ad Hoc Networking Working Group. [Online]. Available: http://www.potaroo.net/ietf/ids/draft-ietf-manet-aodv-13.txt
- [3] V. D. Park and M. S. Corson, "A highly adaptive distributed routing algorithm for mobile wireless networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Kobe, Hyogo, Japan, Apr. 7–12, 1997, pp. 1405–1413.

- [4] S. Roy and J. J. G. L. Aceves, "Using minimal source trees for on-demand routing in ad hoc networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Anchorage, AK, Apr. 22–26, 2001, pp. 1172–1181.
- [5] Y. Hu and D. Johnson, "Caching strategies in on-demand routing protocols for wireless ad hoc networks," in *Proc. ACM Int. Conf. Mobile Comput. Netw. (MobiCom)*, Boston, MA, Aug. 6–11, 2000, pp. 231–242.
- [6] E. Gafni and D. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Trans. Commun.*, vol. 29, no. 1, pp. 11–15, 1981.
- [7] R. G. Ogier, F. L. Templin, B. Bellur, and M. G. Lewis. (2003, Apr.) Topology broadcast based on reverse-path forwarding (TBRPF). Internet Draft. [Online]. Available: http://www.potaroo.net/ietf/ids/draft-ietf-manet-tbrpf-08.txt
- [8] P. Jacquet et al.. (2003, Apr.) Optimized link state routing protocol [Online]. Available: http://hipercom.inria.fr/olsr/draft-ietf-manet-olsr-09.txt
- [9] The Network Simulator-ns-2, http://www.isi.edu/nsnam/ns/. ns-2.1b8.
- [10] T. Clausen. (2000, Oct.) OLSR ns2 simulation code. INRIA. [Online]. Available: http://hipercom.inria.fr/olsr/
- [11] J. Broch et al., "A performance comparison of multi-hop wireless ad hoc network routing protocols," in Proc. ACM Int. Conf. Mobile Comput. Netw. (MobiCom), Dallas, TX, Oct. 25–30, 1998, pp. 85–97.
- [12] P. Johansson et al., "Scenario based performance analysis of routing protocols for mobile ad hoc networks," in Proc. ACM Int. Conf. Mobile Comput. Netw. (MobiCom), Seattle, WA, Aug. 15–20, 1999, pp. 195–206.
- [13] T. H. Cormen, C. H. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1999.
- [14] M. Garey and D. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness. San Francisco, CA: Freeman, 1979.
- [15] Z. Zhan. (2000) One in three SAT (1in3SAT) problem. [Online]. Available: www.eecs.wsu.edu/cs516/notes34.ps



J. J. Garcia-Luna-Aceves (S'75–M'77–SM'02) received the B.S. degree in electrical engineering from the Universidad Iberoamericana, Mexico City, Mexico, in 1977, and the M.S. and Ph.D. degrees in electrical engineering from the University of Hawaii, Honolulu, in 1980 and 1983, respectively.

He is a Professor of Computer Engineering at the University of California, Santa Cruz (UCSC). He directs the Computer Communication Research Group (CCRG), which is part of the Information Technologies Institute of the Baskin School of Engineering,

UCSC. He has been a Visiting Professor at Sun Laboratories and a Consultant on protocol design for Nokia. Prior to joining UCSC in 1993, he was a Center Director at SRI International (SRI), Menlo Park, CA. He has published a book and more than 225 refereed papers.

Dr. Garcia-Luna-Aceves received the SRI International Exceptional-Achievement Award in 1985 and 1989. He has been Program Co-Chair of ACM MobiHoc 2002 and ACM Mobicom 2000; Chair of the ACM SIG Multimedia; General Chair of ACM Multimedia'93 and ACM SIGCOMM'88; and Program Chair of the IEEE MULTIMEDIA'92, ACM SIGCOMM'87, and ACM SIGCOMM'86. He has served in the IEEE Internet Technology Award Committee, the IEEE Richard W. Hamming Medal Committee, and the National Research Council Panel on Digitization and Communications Science of the Army Research Laboratory Technical Assessment Board. He has been on the Editorial Boards of the IEEE/ACM TRANSACTIONS ON NETWORKING, the Multimedia Systems Journal, and the Journal of High-Speed Networks.



Soumya Roy received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, in 1998, and the M.S. and Ph.D. degrees in computer engineering from the University of California, Santa Cruz, in 2000 and 2003, respectively, where his Ph.D. dissertation under Prof. J. J. Garcia-Luna-Aceves was on "on-demand link-state routing in ad hoc networks."

Presently, he is associated with Ipsum Networks, Philadelphia, PA. His research interests are in

routing, overlay networks, network measurements and algorithms.