On-demand Loop-Free Routing with Link Vectors

J.J.Garcia-Luna-Aceves and Soumya Roy
Computer Engineering Department
University of California, Santa Cruz
jj, soumya@soe.ucsc.edu

Abstract

We present the on-demand link vector (OLIVE) protocol, a routing protocol for ad-hoc networks based on link-state 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 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 route request 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 DSR, AODV, OLSR and TBRPF, in terms of control overhead, throughput, and average network delay, while maintaining loop-free routing with no need for source routes.

1 Introduction

Several on-demand routing protocols have been proposed to maintain routing tables efficiently in ad hoc networks [2], [3], [4], [5]. 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.

This work was supported in part by the US Air Force/OSR under grant F49620-00-1-0330 and by the Baskin Chair of Computer Engineering.

The dynamic source routing (DSR) protocol [1], [2] 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) 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 [3] 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 loosing its route to D. A node receiving the RREQ can send back a unicast route reply 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 route request. Otherwise, the node receiving the route request must forward the route request. 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 route requests.

The temporally-ordered routing algorithm (TORA) [4] 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 [5] 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. Sec. 2 provides a detailed description of OLIVE and illustrates its operation. OLIVE does not need internodal synchronization spanning multiple hops, the use of any packet-header information other than the destination for loop-free packet forwarding, or the use of destination-based sequence numbers. Like other on-demand routing protocols, route requests and route replies in OLIVE are sent for initial path set up. 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 set-up, while the network topology is used to compute plausible paths that become useful when the original paths break.

Sec. 3 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.

Sec. 4 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 has 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. Sec. 5 concludes our paper.

2 OLIVE

2.1 Motivation

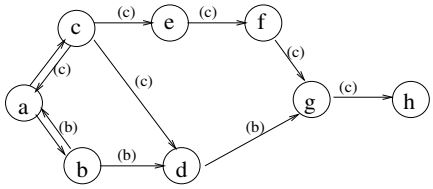


Fig. 1. Network topology known by node a.

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 b and 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. 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 h. However, this leads to a routing error; upon receiving a data packet destined to node h, node b would drop the data packet, because node b does not have a route to destination h.

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, it can be shown that this results in an NP-complete problem [13] 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, allowing local route repairs, and requiring no source routes or flow identifiers in the headers of data packets.

2.2 Principles of Operation

In OLIVE, the source of a data packet that has no route for the destination broadcasts a route request (RREQs) 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 towards 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.

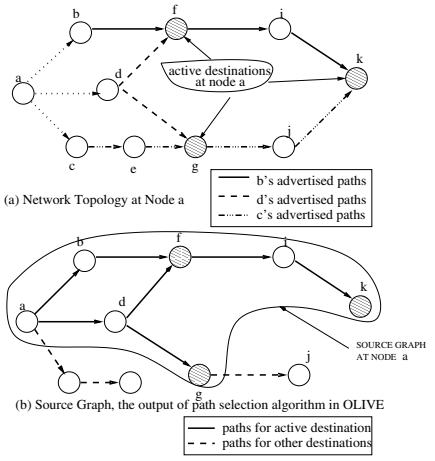


Fig. 2. Figure showing how path selections are done in OLIVE

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, $p_i^{kj}(cost)$ 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 $p_i^{xj}(cost) = Min\{p_i^{nj}(cost) \mid n \in N_i\}$.

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 $aceg$ and adg, of which the path adg is chosen because it has a smaller cost. Similarly, among the paths advertised for destination k, namely $adfik$ and $acegjk$, path $abfik$ is chosen. For destination f, the only path available is adf. Using only complete paths reported by neighbors avoids packet forwarding to nodes with no paths to destinations, the problem discussed in previous section in relation to on-demand 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 also makes use of *plausible paths*. A plausible path through a neighbor is either (a) a non-active path advertised by the neighbor, or (b) 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 non-active destination j at node i. However, the plausible path $adgj$ can be computed using Dijkstra's SPF algorithm, which comprises of 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. A node that receives the FRREQ and has an active route to the destination replies with a forced route reply (FRREP) that specifies an active path to the destination. A node that does not have any path to the destination of a FRREQ replies with a FRREP stating an empty active path. Other nodes will forward the FRREQ to the next hop of the plausible path. FRREPs are sent back to the origin of the FRREQ along the reverse path taken by the 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. 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 lost its path to a destination through reliable RERRs, or picks alternate routes that are shorter than the route that it advertised to its predecessors, instantaneous loop-freedom can be maintained when every node knows its predecessors.

2.3 Detailed Description

The operation of OLIVE can be classified into three phases: (a) route discovery for setting up new paths, (b) local route repair for finding alternate paths when the original breaks, and (c) route failure notification for updating neighbors of route failures.

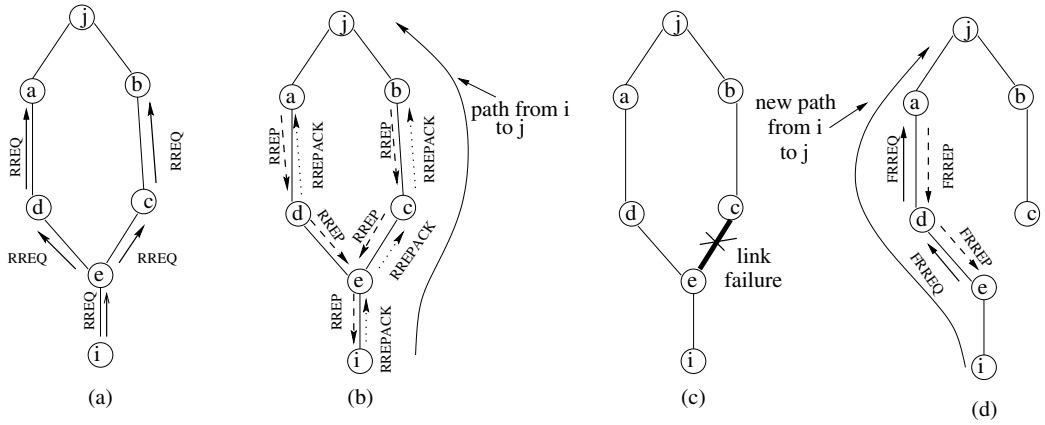


Fig. 3. Figure showing the route discovery and route repair methods in OLIVE

2.3.1 Route Discovery

Route requests (RREQ) are used to discover routes for unknown destinations, while RREPs carry routing information. 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 forwards it only if: (a) it has no valid route for the destination, (b) no RREQ initiated by the same source has been forwarded recently, and (c) 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 destination node waits for a back-off time proportional to the node's distance from the target before sending a broadcast RREP, and cancels its 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 forwards a RREP to a neighbor, it ensures that the neighbor is part of the predecessor list for the destination of the original RREQ.

An acknowledgment 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.

2.3.2 Local Route Repair

Communication overhead can be reduced drastically by localizing the impact of route failures. 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, forced route requests (FRREQ) are sent along those plausible paths to check their viability.

Each FRREQ carries information about the plausible path to the destination, such that each node on the path having an active route can compare the path with its current route for the destination. The node either forwards the FRREQ if it only has plausible paths to the destination, or responds with a forced route reply (FRREP), which either contains a valid route for the destination or contains no route. 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. When FRREQs do not yield any response within a certain time interval, the alternate paths are assumed to be non-existent and RERRs are sent to predecessors.

2.3.3 Route Failure Notification

Route errors (RERR) and RERR acknowledgments (RERRACK) are used for reliable transfer of route failure information and updating predecessor lists. A node sends a RERR to its predecessors to a given destination under any of four conditions: (a) A MAC- or network-layer mechanism states that the link to the next hop of its path to the destination failed, (b) a RERR is received from a neighbor for a destination for which the router is a relay and the router has no paths to the destination, (c) the active route to a destination expires and the router is unable to find alternate paths, and a RREP is received specifying a path with invalid links.

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.

2.4 Neighbor Relationship

A node considers the link to a neighbor to be up when any one of the following conditions happen: (a) The node receives a control packet from the neighbor for the first time, (b) it receives the first network-layer hello message from the neighbor, or (c) 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: (a) The router receives link-layer notification caused by the failure to deliver data packets across the link, (b) network-level hello messages are missed several consecutive times, (c) the link layer declares the link to be down either through the action of a neighbor protocol, or after several retransmissions, (d) acknowledgments for data packets are not received after repeated network-level retransmissions, and (e) 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.

2.5 Handling Link Sequence Numbers

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.

2.6 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 set-up 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 a's RREP is meant for node d, while node b's RREP is sent to node c. When node a sends its RREP to node d reporting its active route for node j, it will include node d in 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 c first. then node e sends a RREPACK to node c only and not to node d. When node d 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 d 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 i, which sends a RREPACK back to node e after it selects the final path, $iecbj$.

Fig. 3(c) and Fig. 3(d) illustrate 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 c) 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 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 d for destination j. Therefore, it might not receive updates regarding destination j from node d. Hence, node e sends a FRREQ, which is forwarded along the path $edaj$, and node a, which has an active route to j, replies with a FRREP, which traces back from a to d, and then to node e, setting up the new route $edaj$.

3 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 $\mathbf{SG_j} = (V, E')$, where $E' = \{(i, s_i^j) \mid i \in V$ and s_i^j is the successor for j 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 bi-directional. Furthermore, Theorem 1 below 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 proven based on 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 $PRED_i^j(t)$ is the set of predecessors for destination j as known to node i and $\overline{PRED}_i^j(t)$ 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 $\overline{PRED}_i^j(t) \subseteq PRED_i^j(t)$.

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

Let $a, b, c.....x$ 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 $P_i(t)^j$ 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, $P_a(t)^j(cost) \geq P_x(t)^j(cost)$. Assume that the last change in successor occurred at time $t' < t$, when node x chooses node a as the next hop. Therefore, $P_x(t)^j(cost) = P_x(t')^j(cost) > P_a(t'')^j(cost)$, where $P_a(t'')^j(cost)$ is the cost of the path advertised by a in its RREP or FRREP at time $t'' < t' < t$. This implies that $P_a(t)^j(cost) \geq P_x(t)^j(cost) = P_x(t')^j(cost) > P_a(t'')^j(cost)$, which means that node a has experienced an increase in the cost of its path to node j in the interval $(t", t]$. 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 level 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 loops. 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 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_j^v) for each $v \in S_j$ or $v \in R_j$, where s_i^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 RREQs, RREPs, FRREQs and FRREPs 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 $c \in S_j$ can have no path to destination j that ends at a node without a valid path to j.

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 RREQs and RREPs 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

4 Performance Comparison

We have compared the performance of OLIVE with the on-demand routing protocols (DSR [2], AODV [3]) 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 1 have 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 co-ordination function (DCF) for wireless LANs. The broadcast packets are sent unreliably and are prone to collisions. The physical layer approximates a two-Mbps DSSS radio interface The radio range is 250m and for all the simulations the run length is 600 seconds.

TABLE 1
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

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 1500m×300m. 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 seconds.

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 seconds and after the termination of the flow, within 1 second, the source randomly chooses another destination and starts another flow, which again lasts for 200 seconds.

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.

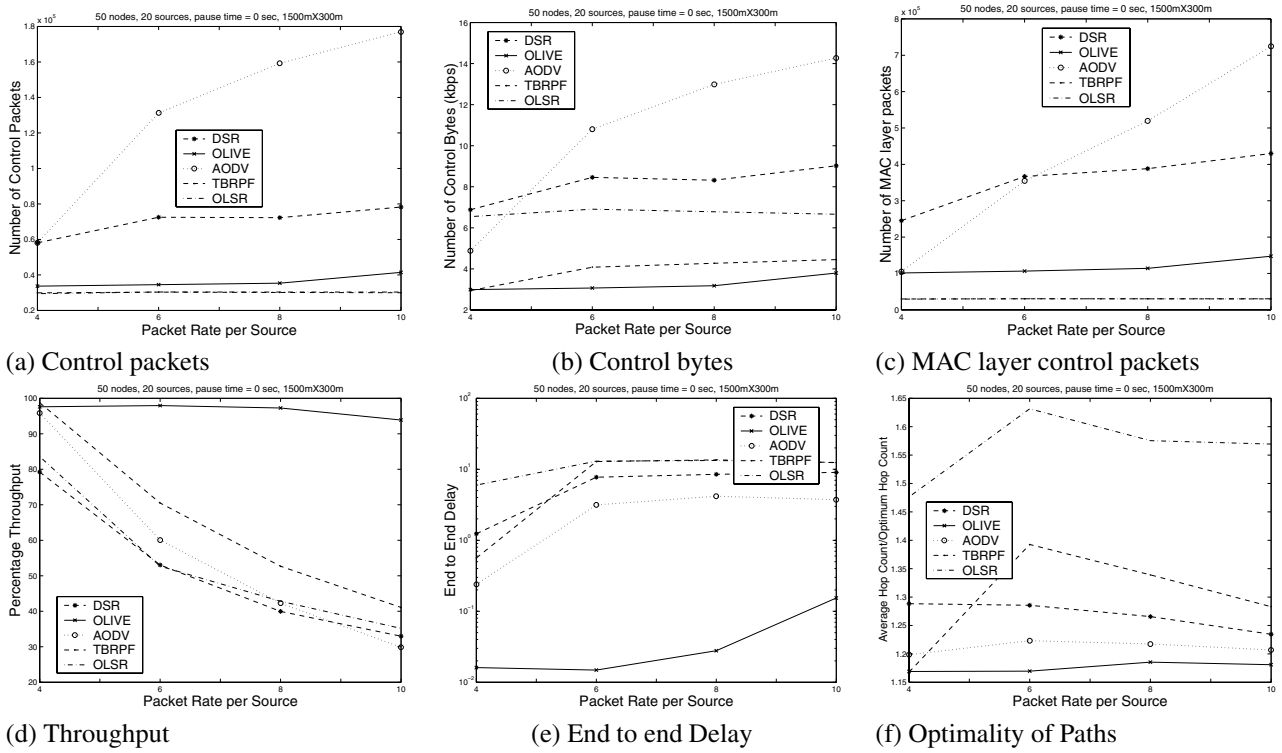
Control packet overhead: The total number of control packets sent out during the simulation. Each broadcast packet is counted as a single packet.

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



(a) Control packets (b) Control bytes (c) MAC layer control packets

(d) Throughput (e) End to end Delay (f) Optimality of Paths

Fig. 4. Performance in a 50-node network with 0 second pause time and 20 sources with varying packet load

and then arriving at the receiver application. This includes delays due to route discovery, queueing at IP and MAC layers, and propagation in the channel.

4.1 Effect of Traffic Load

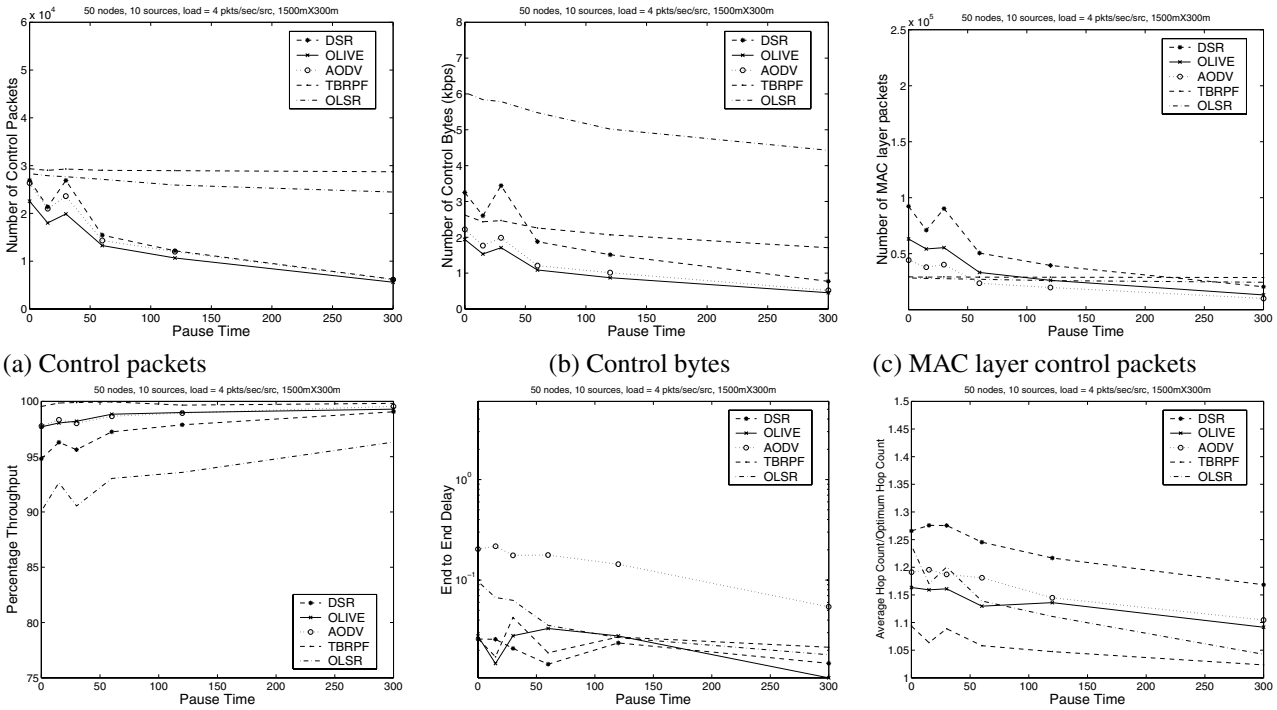
Fig. 4 shows the results of each protocol under varying packet loads when the number of sources is 20 and nodes are constantly moving.

The control overhead of OLIVE remains unchanged with the increase of load. DSR, TBRPF and OLSR exhibit similar behavior. However, AODV's 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 kbps/source or 10 packets/sec/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 remains are constant and very similar.

In terms of application-oriented metrics, like throughput or network delay (Fig. 4(d) and Fig. 4(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. 4(f)).

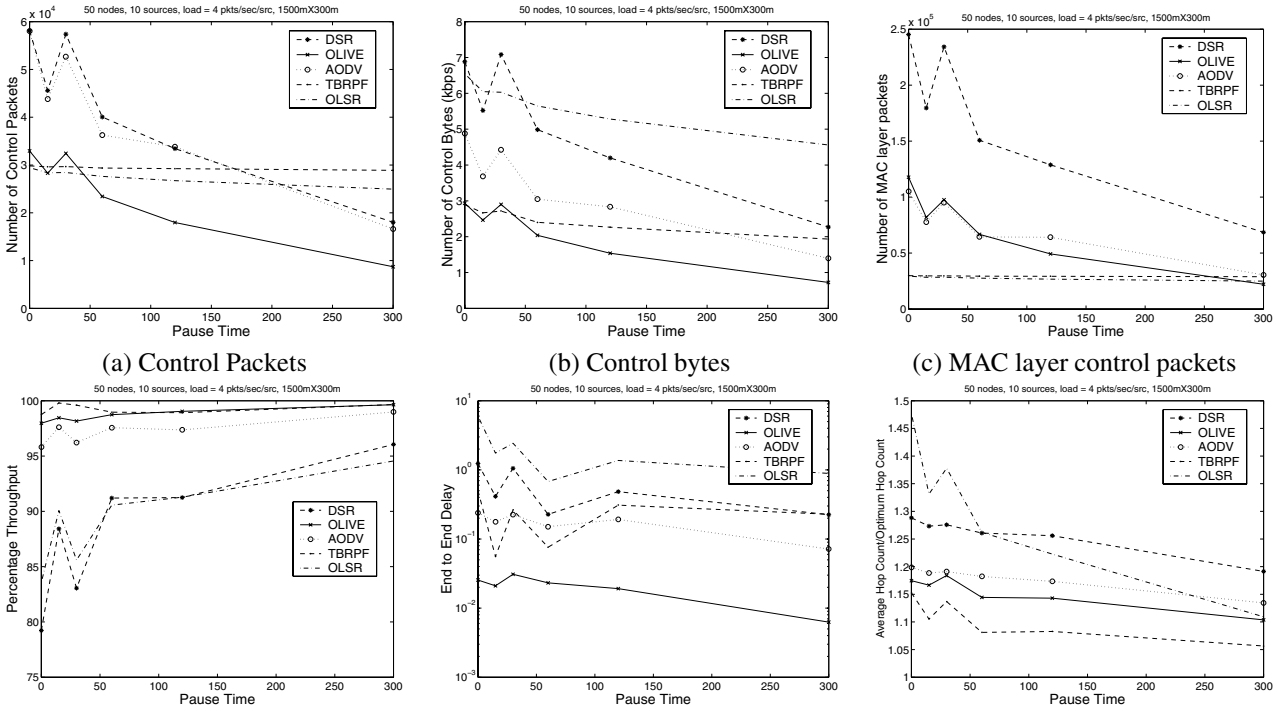
For TBRPF, data packets are always re-scheduled along alternate routes, when the original paths break. Therefore, they get circulated throughout the network and there is considerable packet loss due to looping and TTL timeout. In TBRPF, all the control packets are broadcast. Therefore, for sending any TBRPF control packet, no extra MAC layer handshake is necessary. However, because



(a) Control packets (b) Control bytes (c) MAC layer control packets

(d) Throughput (e) End to end delay (f) Optimality of paths

Fig. 5. Performance in a 50-node network with load of 4 packets/s/source and 10 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

Fig. 6. Performance in a 50-node network with load of 4 packets/s/source and 20 sources under varying mobility

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.

4.2 Effect of Mobility

Fig. 5 and Fig. 6 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 per second.

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 RTS/CTS handshakes, the difference between AODV and OLIVE in terms of MAC layer control packets (Fig. 5(c) and Fig. 6(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 to 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 (Fig. 5(d) and Fig. 6(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 (Fig. 5(f) and Fig. 6(f)), OLIVE is best among all the on-demand routing protocols. However, because TBRPF uses hello packets, it can learn about nodes that have moved before the on-demand protocols that depend on reception of control packets to detect neighbor connectivity.

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], [12]. The reason is likely to be the absence of promiscuous listening in DSR.

In terms of delay (Fig. 5(e) and Fig. 6(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.

4.3 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. 7 shows the number of packets that have been either sent along a loop, or dropped due to TTL timeouts or 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: (a) the source finds that the data packet has come back to it, and (b) 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. 7, 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 non-availability of routes, which implies that the topology information always helps data packets to be re-scheduled along alternate paths. However, in heavy-load scenarios, when the links fail frequently due to congestion, alternate paths are not always the correct choices.

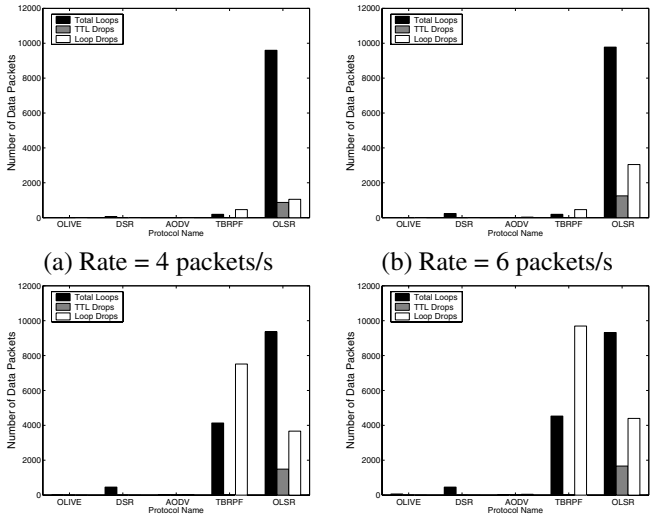
The control packet exchanges in OLIVE and AODV ensure instantaneous loop freedom. In DSR, the source routes carry information about path traversed and the path to be traversed; therefore, loops can be easily detected. Under high load, some data packets go into loops in DSR when data packets are salvaged 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 re-routing the packet using its own cached routing information. Because path traversal information is not checked for re-routing, 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.

5 Conclusions

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



(a) Rate = 4 packets/s (b) Rate = 6 packets/s

(c) Rate = 8 packets/s (d) Rate = 10 packets/s

Fig. 7. Loops for a 50-node network with 20 sources under four different load conditions

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 which nodes use it as next hop to an active destination, called predecessors for that 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 installed corresponding to obsolete alternate paths, a node unicasts route requests along an alternate path, so that its validity can be verified by a route reply.

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.

References

[1] D. Johnson et al, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)," IETF Internet draft, draft-ietf-manet-dsr-09.txt, April 2003.

[2] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*, Imielinski and Korth, Eds., vol. 353. Kluwer Academic Publishers, 1996.

[3] C. E. Perkins et al., "Ad hoc On-Demand Distance Vector (AODV) Routing," Request for Comments 3561, July 2003.

[4] V. D. Park and M. S. Corson, " A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks," *Proc. IEEE Conference on Computer Communications (Infocom)*, Kobe, Japan, April 7-12 1997.

[5] S. Roy and J. J. Garcia Luna Aceves, "Using Minimal Source Trees for On-Demand Routing in Ad Hoc Networks," *Proc. IEEE Infocom 2001*, Anchorage, Alaska, April 22-26, 2001.

[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 et al, "Topology Dissemination Based on Reverse-Path Forwarding (TBRPF)," *Request for Comments 3684*, February 2004.

[8] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol," *Request for Comments 3626*, October 2003.

[9] http://www.isi.edu/nsnam/ns/, "The Network Simulator - ns-2," ns-2.1b8.

[10] T. Clausen, "OLSR ns2 simulation code," *http://hipercom.inria.fr/olsr/*. INRIA, October 18 2000.

[11] J. Broch et. al., "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," *Proc. ACM Mobicom 98*, Dallas, Texas, October 25-30, 1998.

[12] P. Johansson et. al., "Scenario Based Performance Analysis of Routing Protocols for Mobile Ad-Hoc Networks," *Proc. ACM Mobicom 99*, Seattle, Washington, August 15-20, 1999.

[13] S. Roy, "On-demand Link-state Routing in Ad-hoc Networks," PhD Thesis, Computer Engineering, University of California, Santa Cruz.