Efficient Use of Route Requests for Loop-Free On-demand Routing in Ad Hoc Networks *

Hari Rangarajan[1] and J.J. Garcia-Luna-Aceves[1,2]

[1] University Of California at Santa Cruz,
Computer Engineering Dept., Santa Cruz, CA 95064
[2] Palo Alto Research Center,
3333 Coyote Hill Road, Palo Alto, CA 94304
{hari, jj}@cse.ucsc.edu

Abstract. We present a new loop-free on-demand routing protocol for ad-hoc networks, the *Labeled Successor Routing (LSR)* protocol, which identifies loop-free successors to a destination using route-request labels (RRL). Each route request (RREQ) used during the on-demand destination search process is identified uniquely by a sequence number associated with the issuing source address. Route replies (RREP), which traverse loop-free paths created by RREQs, carry the associated RRL that is stored by nodes along the created successor path to the destination. Without requiring an additional mechanism for loop-freedom (e.g., per destination-sequence numbers or source-routing) LSR allows neighbors of a source to reply to RREQs, avoiding the destination being the only node capable of replying. Simulations results for scenarios consisting of networks of 50 and 100 mobile nodes show that LSR performs comparably or better than the Dynamic Source Routing (DSR) protocol, AODV, and the Optimized Link State Routing (OLSR) protocol.

1 Introduction

Several on-demand routing protocols have been proposed to date for mobile ad hoc networks (MANET). Such protocols establish routes to only those destinations for which there is traffic, and attempt to ensure loop freedom at every instant to limit control overhead. Because of node mobility and the need to setup routes when required, all on-demand routing protocols are characterized by the use of a route request flood to search for the destination. Each route request(RREQ) attempt by a source for a destination is represented by an unique (source address, flooding identifier) pair, to which we refer as a Route Request Label (RRL). RRLs are required to prevent RREQs from being processed multiple times, and to set up loop-free paths along which route replies generated by intermediate nodes or destinations traverse. However, the actual establishment and maintenance of routes in current on-demand routing protocols is independent of RRLs.

* This work was funded in part by the Baskin Chair of Computer Engineering at UCSC and by the National Science Foundation under Grant CNS-0435522.

R. Boutaba et al. (Eds.): NETWORKING 2005, LNCS 3462, pp. 1096–1107, 2005.

Current on-demand protocols use RRLs during route request floods, but use additional mechanisms to achieve loop-free routing. For example, the dynamic source routing protocol (DSR) [4] collects partial topology information during the RREQ flood, and uses path information collected to source-route data packets. The feasible label routing (FLR) [5] protocol collects path information much like DSR does, but achieves loop-free hop-by-hop routing of data packets based on just destination addresses by assigning lexicographically ordered labels, derived from path information, to nodes along a successor path to the destination. The Ad hoc On-demand Distance Vector protocol (AODV) [6] uses per-destination sequence numbers to maintain loop-freedom.

Recent work in the MANET Working Group of the Internet Engineering Task Force has focused on generic routing frameworks that are simple to implement, deploy, and test, while additional features can be added as necessary for improved performance. The most recent proposal as of this writing is the Dynamic MANET On-demand [7] Routing Protocol. It is based on AODV and allows additional capabilities to be added by allowing flexible control message parsing; however, DYMO attempts to maintain loop-free routing using the same per-destination-sequence numbers used in AODV. As a result, it is susceptible to the same performance and robustness issues found in AODV, namely: (a) most route requests have to be answered by the destination, and (b) it can suffer from temporary loops, de-facto partitions and count-to-infinity [2, 3].

In this paper we present a new loop-free routing approach that uses the information already needed in route requests to establish and maintain loop-free routes, and allows nodes other than the destinations to initiate route replies. Section 2 presents the design of the Labeled Successor Routing (LSR) protocol, which demonstrates the use of RRLs to achieve efficient loop-free routing. The basic idea behind LSR is very simple: A source floods a route request identified by an unique RRL, which creates a tree rooted at the source as the RREQs are processed only once by each node. When the REPs traverse loop-free reverse paths, nodes update their routing tables for the specified destination and store the RRL. If the source stored a RRL that belonged to a different source before issuing this new RREQ, it sequences to a new RRL; otherwise, it retains the previously stored RRL. At a later time, the source can pick any neighbor node that has a stored RRL that is more recent or the same as the one stored at the source, as a loop-free successor towards the destination. Section 3 illustrates the working of LSR with an example. Section 4 analyzes the correctness of LSR. Section 5 compares the performance of LSR against AODV, DSR and a proactive link state protocol (the Optimized Link State Routing or OLSR) [1]. Section 6 provides our concluding remarks.

2 Labeled Successor Routing (LSR)

The labeled successor routing (LSR), is an on-demand protocol based on a control signaling (RREQ, RREP and RERR) similar to that used in other on-demand routing protocols.

Each route request label RRL is a unique pair $\{src, id\}$, where src is a node address identifier and id is an integer created by src. The notation src^{RRL} is used to refer to the value of src in a RRL. An empty RRL is denoted by ϕ.

Route replies, and route requests are denoted by super-scripts req, and rep, respectively. For a destination D, node A stores a route-request label, RRL_D^A. The successor to destination D at A is denoted by s_D^A. $NRRL_D^{req}$ is the neighbor-query RRL of the request used to identify neighbors of the source that have a loop-free path to the destination. d_D^A is the distance (hop count) to the destination. lc_A^B is the link cost from node A to B. $DInit_D^{rep}$ is carried in a RREP to indicate whether the route reply was initiated by the destination.

The following relational operator is defined on two labels, $RRL_1 = \{src_1, id_1\}$, and $RRL_2 = \{src_2, id_2\}$, to establish ordering among RRLs:

$$RRL_1 \prec RRL_2 \;\; if \;\; src_1 = src_2 \;\wedge id_1 \geq id_2 \tag{1}$$

The above relational operator allow a source to compare if a neighbor can be chosen as a loop-free successor safely (i.e., without incurring a loop). LSR uses the following four conditions based on this operator to maintain loop-free paths:

*A*SC: (Accept Successor Condition). When node A receives a RREP from node B for destination D, then node A sets $s_D^A \leftarrow B$, if (i) $DInit_D^{rep} = 0$, $src^{RRL_D^A} = A$, and $RRL_D^{rep} \prec RRL_D^A$; or (ii) $DInit_D^{rep} = 1$. If $RRL_D^A = \phi$ or $src^{RRL_D^A} \neq A$, then node A should accept a RREP only if $DInit_D^{rep} = 1$ (i.e., generated by the destination).

*S*SC: (Start Successor Condition). Node I can issue a RREP responding to a RREQ for destination D if I has an active route to D, and $RRL_D^I \prec NRRL_D^{req}$. If node $I = D$, then it must set $DInit_D^{rep} = 1$ and issue a RREP.

*R*SC: (Relay Successor Condition). Node A, on processing a RREP rep, must relay the RREP only if $DInit_D^{rep} = 1$. Node A relays a RREQ for destination D only if A has not previously processed this RREQ, and sets $NRRL_D^{req} = \phi$.

*U*SC: (Update Successor Condition). If node A must change s_D^A, then it sets $d_D^A \leftarrow \infty$, and issues a new RREQ req. If $src^{RRL_D^A} = A$, then it sets $NRRL_D^{req} = RRL_D^A$, else $NRRL_D^{req} = \phi$.

USC allows a source to query its neighbors with its stored RRL to identify any loop-free paths to the destination. If no neighbor can answer the RREQ, then, as per RSC, the RREQ reaches the destination building a tree rooted at the source, and the RREP generated by the destination traverses one of the loop-free reverse paths along the tree. SSC allows neighbors to issue a reply that can be used at the source, and ASC allows the source to safely switch successors for a destination without causing any loops. If $NRRL_D^{req} = \phi$, then the RREQ can only be answered by the destination.

2.1 Information Stored and Exchanged

The routing table at node A maintains the following parameters for every destination D: the successor (s_D^A), the route-request label (RRL_D^A), the distance (hop count) to the destination (d_D^A), lifetime of route, and the state of route entry (rt_D^A): valid or invalid. If no entry for destination D exists, then it is considered equivalent to having a *null* RRL (ϕ). Node A maintains a monotonically increasing source sequence number ID_A, which additionally serves as a route-request identifier.

A RREQ consists of the tuple {*dst*, *src*, *rreqid*, $NRRL_{dst}$,$flags$}. The field *src* denotes the identifier of the source that is seeking a path to the destination (*dst*), the flooding identifier *rreqid* along with the source (*src*) represents the unique route request label for this RREQ generated for a destination, $NRRL_{dst}$ is the neighbor-query RRL to find neighbors of the source that have a loop-free path to the destination. $flags$ carries control bits.

A RREP consists of the tuple {*dst*, *src*, *rreqid*, RRL_{dst}, d_{dst},*ttl*, $flags$}. The field *ttl* states the lifetime of the route at the node relaying the RREP, *rreqid* is carried in the RREP to forward it along the reverse path to the source using information cached for the RRL (*src*, *rreqid*), and also, if required, serves to form the new RRL, RRL_{dst} is the label stored at the relaying node for *dst*, d_{dst} is the distance to the destination at the relaying hop, and $flags$ contains the *'DInit'* bit, which is set when the destination originates the RREP (if set then RRL_{dst} is set to ϕ).

The RERR is the tuple {*orig*, *unreachdests*}, where *orig* denotes the node originating the route errors, and *unreachdests* is the list of destinations that are not reachable at *orig*.

A node relaying a RREQ identified by a RRL (*src*, *rreqid*) caches the address of the node *revhop* that sent the RREQ, and is used to relay a RREP received for this (*src*, *rreqid*) pair along the reverse path. Cached entries are maintained for a period of time that is long enough, so that all RREPs for the RREQ with RRL (*src*, *rreqid*) will be received.

2.2 Route Maintenance

(A) Initiating a RREQ: Node A is said to be *active* in a route computation for destination D (i.e., the RREQ) when it initiates a RREQ for the destination, and the RREQ is uniquely identified by the pair (A, ID_A). A node relaying a RREQ (A, ID_A) originated by another node is said to be *engaged* in the RREQ. A node that is not active or engaged in a route computation for destination D is said to be *passive* for that destination.

At any given time, a node can be the origin of at most one RREQ for the same destination. The RREQ (A, ID_A) terminates when either node A attains a successor for destination D or the timer for its RREQ expires.

If node A is active or engaged for destination D and receives data packets for the destination, it buffers those data packets. If node A is passive for destination D and requires a route for destination D, it sets $ID_A \leftarrow ID_A + 1$, $reqid \leftarrow ID_A$, and *RREQ timer* $\leftarrow$ *(*2.ttl.latency) (where *ttl* is the time-to-live of the broadcast

flood and latency is the estimated per-hop latency of the network). If $src^{RRL_D^A} = A$, then $NRRL = RRL_D^A$, else $NRRL = \phi$. Node A then issues RREQ $\{D, A, reqid = ID_A, NRRL\}$.

If node A receives no RREP after the expiry of its timer for RREQ (A, ID_A) for destination D, it sends a new RREQ with an increased ttl. If node A does not receive a RREP for destination D after a number of attempts, a failure is reported to the upper layer. The number of hops that a RREQ can traverse is controlled externally from the RREQ by means of the TTL field of the IP packet in which a RREQ is encapsulated, or by other means.

(B) Relaying RREQs: When node B receives a RREQ $\{D, \ A, \ rreqid = ID_A, \ NRRL_D^{req}\}$ from node I, it first determines its own status for (A, ID_A). If B is active (i.e., $B = A$) or engaged (i.e., B has cached the RREQ (A, ID_A)) in the computation (A, ID_A), it silently drops the RREQ. Otherwise, node B is passive. In this case, if SSC is satisfied (i.e., $RRL_D^B \prec NRRL_D^{req}$), then node B issues a RREP (Section 2.2 (B)). Else, if SSC is not satisfied, node B becomes engaged and relays the new RREQ req' with $NRRL_D^{req'} \leftarrow \phi$.

A node may be engaged in multiple RREQs for the same destination, but relays a RREQ from the same origin only once by caching the reverse hop, $revHop = I$, for a given RREQ (A, ID_A) it forwards.

(C) Initiating and Processing RREPs: When node I processes a RREQ $\{D, A, rreqid = ID_A, NRRL_D^{req}\}$, if SSC is satisfied: $RRL_D^I \prec NRRL_D^{req}$, and $rt_D^A = valid$, it issues a RREP $\{D, \ A, \ ID_A, \ RRL_D^{rep}, \ d_D^{rep}, \ ttl\}$ with $RRL_D^{rep} \leftarrow RRL_D^I$ and $d_D^{rep} \leftarrow d_D^I$. At the destination $(I = D)$, D sets $DInit_D^{rep} \leftarrow 1$, $RRL_D^{rep} \leftarrow \phi$, and $d_D^{rep} \leftarrow 0$.

If node A receives a RREP $\{D, \ src = S, \ rreqid = ID_S, \ RRL_D^{rep}, \ ttl, \ d_D^{rep}, \ DInit\}$, it updates its routing table for destination D as described in Section 2.2(D). After updating its routing table, if $A \neq S$ and the RREP is not dropped, the node A increments ID_A, and the RREP is relayed along the reverse hop $revHop$ which is retrieved from the cache entry (A, ID_A). The RREP is relayed with the parameters $RRL_D^{rep} \leftarrow \phi$, $DInit_D^{rep}$ unchanged (must be one), and $d_D^{rep} \leftarrow d_D^A$.

(D) Adding, Updating, and Maintaining Routes: When node I receives RREP $\{D, \ A, \ ID_A, \ RRL_D^{rep}, \ ttl, \ d_D^{rep}, \ DInit_D^{rep}\}$ from neighbor B for destination D, it drops the RREP silently if ASC is not satisfied; otherwise, its routing table is updated as follows:

- Case (i), $DInit_D^{rep} = 1$,
  - if $I \neq A$, then node I sets $RRL_D^I \leftarrow [(A, ID_A^{rep})]$.
  - if $I = A$, and $RRL_D^{rep} \prec RRL_D^I$, then node I retains the same RRL_D^A; otherwise, sets $RRL_D^I \leftarrow [(A, ID_I)]$.
- Case (ii), $DInit_D^{rep} = 0$, $I = A$, and $RRL_D^{rep} \prec RRL_D^A$ then node I(i.e., A) retains the same RRL_D^A.

In the next step, node I sets $s_D^I \leftarrow B$; and updates route cost, $d_D^I \leftarrow d_D^{rep} + lc_I^B$, where lc_B^I denotes the link cost from node I to B. When $DInit_D^{rep} = 0$, nodes only switch to shorter cost paths.

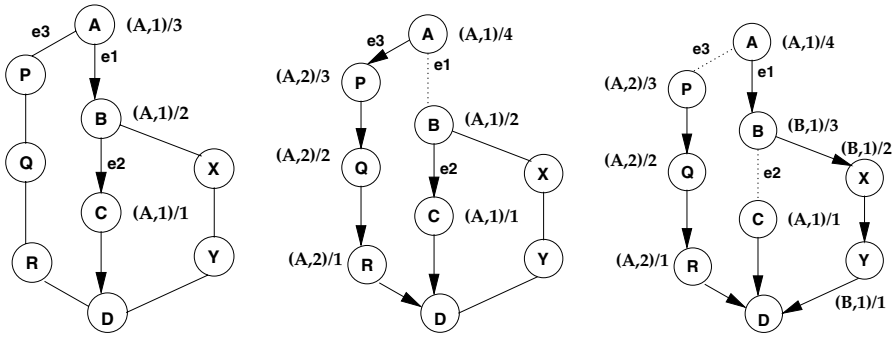
(E) Route Maintenance: Node A invalidates a route entry for destination D, with S as the next hop, in one of the following ways: (i) No data packet is forwarded using this route entry for *active_route_timeout* seconds (the time after which a route-entry expired); (ii) A link-failure notification for the next hop S is received; or (iii) A RERR is received, which indicates that D is no longer reachable through S. A node A invalidating an entry performs the following steps: It sets $rt_D^A = invalid$, $s_D^A \leftarrow \phi$, and $d_D^A \leftarrow \infty$. A route entry with state $rt_D^A = invalid$ can be purged at any time, to save memory (also, applies after a node reboot when nodes lose all state). For cases (ii), and (iii), node A sends a RERR to all the predecessors (either as a broadcast or separate unicasts), after determining the set of destinations affected by this event.

(F) Source Sequence Numbers: The source sequence number (ID_A) at a node A, serves the additional purpose of being the sequence number for route requests initiated by A. Because both purposes require a a monotonically increasing number (even after reboots), the ID_A must be based on a 64-bit real-time clock, which will avoid any wrap-around issues. Any repetition of the sequence number can cause route requests to be dropped at relay nodes because of previous cached state, and if old sequence numbers are repeated in RRLs then it can result in the formation of loops.

3 LSR Example

Figure 1 shows the directed acyclic successor graph (DASG) for destination D for a nine-node network at different instants of time. Link costs are unity. The figure shows, at each node for the destination D, a tuple $[(src, id)/hopcount]$, where (src, id) is the stored RRL, and *hopcount* is the distance to the destination. We illustrate the following sequence of events.

Node A has an active flow for destination D. Initially at time t_0, the routes are not active; and at all nodes, the RRL stored for D is ϕ. Node A initiates a route



(A) Time t_1 (B) Time t_2 (C) Time t_3

Fig. 1. LSR operation - Example

request req for destination D with parameters (D, A, $rreqid = ID_A = 1$, $NRRL = \phi$). Node B upon receiving the route request (A, $rreqid = 1$), caches the reverse hop ($revHop = A$), and relays a new RREQ req' with (D, A, $rreqid = 1$, $NRRL = \phi$). Similarly, node C relays the RREQ after caching reverse hop B. Note that the parameter $NRRL$ is set to ϕ because node B does not have an active route and the RREQ can subsequently be answered only by the destination.

A route reply rep is generated by node D upon receipt of the RREQ. The RREP (D, A, $rreqid = 1$, $RRL = \phi$, $DInit = 1$) is accepted by node C as it has $DInit$ set. Node C sets $s_D^C \leftarrow D$, $RRL_D^C \leftarrow (A, 1)$, $d_D^C \leftarrow 1$, and relays the RREP along the cached reverse hop. Similarly, because the RREP carries $DInit = 1$, nodes A and B switch successors and update their RRL and hopcounts for destination D as shown in Figure 1(A) at time t_1.

At time $t_1' > t_1$, link $e1$ fails. Node A issues another RREQ with $ID_A = 2$, and sets up a new successor path along nodes P, Q, and R. Similar to the previous illustration, all the nodes switch successors and update their routing tables to set an RRL with $(A, 2)$ and the associated hopcount. Figure 1(B) shows the network state at time $t_2 > t_1'$. At any time later than $t_2' > t_2$, if link $e3$ fails and link $e1$ comes back, node A can still switch successors to node B for destination D. This is because a RREQ carrying a $NRRL = (A, 1)$ will be answered by node B as it satisfies SSC, and because ASC is satisfied, node A can accept it. RRLs allow sources to switch to neighbors irrespective of any ordering based on distances (i.e., LDR) or path labels (i.e., FLR).

We illustrate loop-freedom in LSR with the following sequence of events after time $t_2'' > t_2'$. Assume node B becomes a source of data packets for destination D, and link $e2$ fails. Node B sends a RERR to A informing the loss of link to reach D. To recover from the failure of link $e2$, node B issues a new route request (D, B, $rreqid = 1$, $NRRL = \phi$), and on receiving a RREP from the destination with $DInit = 1$, labels the path along nodes X, and Y to destination D with $RRL = (B, 1)$. Assume at time $t_2''' > t_2''$, link B-X fails. Regardless of whether node B's RERR was received by node A or not, new route requests from B cannot be answered by node A or any node upstream of it, and consequently loop-freedom is maintained even if RERRs cannot be reliably delivered. Figure 1(C) shows the network state at time $t_3 \geq t_2'''$.

Note that node A can still identify node C as a loop-free successor to destination D, because $RRL_D^C = (A, 1)$. Due to mobility, if node A moves closer to node Q or node R, it can still use them as successors to the destination. To summarize LSR's operation in a nutshell: After issuing a route request flood identified by a RRL, a source can identify all nodes which processed and relayed the RREP as loop-free successors to the destination. There are two cases after which this no longer applies: the source re-labels itself (increases to a higher sequence number in its RRL), say node A changed RRL to $(A, 3)$ at a later time; or the relay nodes processed a RREP carrying a RRL from a different source, as in the case of node B, here, which changed RRL from $(A, 1)$ to $(B, 1)$.

4 Analysis

We show that LSR is loop-free at any instant and can ensure that a source can establish a path to a destination in a stable, error-free connected network within a finite-time. We also show that in a connected component, separated from the destination, all nodes invalidate their routing table entries for the destination within finite time, guaranteeing termination.

Theorem 1. *LSR is loop-free at every instant.*

Proof. The proof is by contradiction. Let $P = \{A, ..., B, n_1, n_2, ..., I, ...D\}$ be a successor path along the directed acyclic successor graph (DASG) for destination D, which is loop-free at any time before t. At time t, let node I accept a RREP *rep* from node B, which is upstream to create a loop. We show that when nodes in LSR update their routing tables, it is not possible for I to switch successors to B. There are two cases by which node I will process the update: Case (i), the RREP from B has $DInit = 1$, which means the RREP was initiated by the destination, and must have a traveled a loop-free reverse path by virtue of the unique source tree built using the RREQ (src,id) pair (Theorem 2, [5], pp.49). Hence, I must have relayed the RREP onto B, and cannot receive the RREP again. This is a contradiction, and I can never switch successors to B. Case (ii), node I can switch to A, if $DInit_D^{rep} = 0$, and $RRL_D^{rep} = RRL_D^B \prec RRL_D^I$. Therefore, nodes I and B must possess RRLs, $RRL_D^I = (I, ID')$, and $RRL_D^B = (I, ID'')$, respectively, such that $ID'' \geq ID'$. If $ID'' \geq ID'$, then it means that at a time $t^- < t$, node B must have relayed a RREP along a reverse path to I (which initiated the route request), and later switched to a node upstream of I. However, for node B to apply ASC to switch successors, it must have $src^{RRL_D^B} = B$ which is not possible at time t^-. Hence, at a time $t^- < t^B < t$, node B could not have switched upstream of node I unless node B or one of its downstream successor path nodes to D was part of a RREQ flood (S, ID_S), creating a loop-free reverse path P'. If node I belonged to this path P', then $src^{RRL_D^I}(t_B) = S$, where $S \neq I$. Therefore, in this case, at time $t^B \leq t^I \leq t$, node I cannot apply ASC to switch successors to B. If at any time $t^I \leq t_+^I < t$, node I adopted a RRL (I, ID'''), it must be true that $ID''' > ID''$ because ID_I is incremented on a RREP relay. Hence ASC will not be satisfied at time t, and no loops can be formed.

Theorem 2. *In a connected component G, partitioned from destination D, all nodes will invalidate their routing table entries for D within a finite time in the presence of link failures and node reboots.*

Proof. The one-hop neighbors of destination D, after partition, must detect the link failure to D within a finite time using a link layer notification scheme or otherwise (HELLO messages). Let t be the time after which the component G is partitioned from destination D and all RREPs initiated by the destination, with $DInit = 1$, have been processed by all nodes in G. Assuming default route error (RERR) message propagation from the neighbors of D along the directed

acyclic successor graph (DASG) for destination D in the connected component G, we need to prove that nodes will not re-learn routes from any upstream nodes in the directed acyclic successor graph for D after time t. There are two states, that the nodes are in: Case (i), node A has a $src^{RRL_D^A} \neq A$, or $RRL_D^A = \phi$ which could also be because of a routing table state loss. By ASC, node A can update its routing table only if $DInit = 1$ in the RREP. But, because the destination D is partitioned, it is not possible to receive any new RREPs with $DInit$ set. Case (ii), node A has $src^{RRL_D^A} = A$, and on a link failure or otherwise, it can switch to successors that have $RRL \prec RRL_D^A$. However, as per the argument in Theorem 1, the nodes chosen as successors cannot be upstream of A in the DASG for D. Therefore, RERR messages should propagate along the DASG, hop-by-hop, within a finite time and all nodes should invalidate their routing entries in finite time.

Theorem 3. *In a stable, connected, error-free network, a source S can establish a route to destination D within a finite time.*

Proof. Let a source S start a route request req, identified by (S, ID_S), for a destination D. The RREQ must carry either (i) $NRRL = \phi$, or (ii) $NRRL = (S, ID'_S)$, where $ID'_S < ID_S$. In case (i), only the destination can answer the RREQ, and the RREQ must traverse a path $P = \{n_k, n_{k-1}, ..., n_1, D\}$ to reach the destination. Because the RREP will carry $DInit = 1$(i.e., initiated by destination), all the nodes will update their routing tables and relay the RREP along the reverse path of P. Therefore, S will establish a path to the destination D. In case (ii), there are two possible sub-cases: (a) a neighbor n_k of S that has an active route to D, can satisfy SSC. The RREP issued will satisfy ASC at S. Because of Theorem 1, there must exist a valid loop-free successor path to D from n_k because the network is error-free and connected. Hence, S will establish a route to the destination; (b) if neighbor n_k does not satisfy SSC, then it resets $NRRL = \phi$, which means only the destination can answer the request. Hence, this case follows directly from the same argument as in Case (i). In all cases, because the messages propagate in finite time, the source S can establish a loop-free path to the destination D in finite time.

5 Performance

We present results for LSR over varying loads and mobility. The protocols used for comparison are DSR, AODV, and OLSR which are very well known and being considered in the MANET working group of the IETF. Simulations are run in Qualnet 3.5.2. The parameters are set as in [8].

Simulations are performed on two scenarios, (i) a 50-node network with terrain dimensions of 1500m x 300m, and (ii) a 100-node network with terrain dimensions of 2200m x 600m. Traffic loads are CBR sources with a data packet size of 512 bytes. Load is varied by using 10 flows (at 4 packets per second) and 30 flows (at

4 packets per second). The MAC layer used is 802.11 with a transmission range of 275m and throughput 2 Mbps. The simulation is run for 900 seconds. Node velocity is set between 1 m/s and 20 m/s. Flows have a mean length of 100 seconds, distributed exponentially. Each combination (number of nodes, traffic flows, scenario, routing protocol and pause time) is repeated for nine trials using different random seeds. We present four metrics. *Delivery ratio* is the ratio of the packets delivered per client/server CBR flow. *Latency* is the end to end delay measured for the data packets reaching the server from the client. *Network load* is the total number of control packets (RREQ, RREP, RERR, Hello, TC etc) divided by the received data packets. *Data hops* is the number of hops traversed by each data packet (including initiating and forwarding) divided by the total received packets in the network. This metric takes into account packets dropped due to forwarding along incorrect paths. A larger value for the data-hops metric corresponding to a poor delivery ratio indicates that more data packets traverse more hops without reaching the destination necessarily.

Table 1. Performance average over all pause times for 50 nodes network for 10-flows and 30-flows

Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
LSR	10	0.9960±0.0016	0.0174±0.0022	0.3125±0.0797	2.5983±0.1813
AODV	10	0.9945±0.0023	0.0169±0.0033	0.2700±0.0668	2.5763±0.1793
DSR	10	0.9400±0.0274	0.0411±0.0477	0.2202±0.0952	2.6775±0.1853
OLSR	10	0.8870±0.0406	0.0129±0.0017	1.9370±0.2202	2.4568±0.1754
LSR	30	0.8358±0.0444	0.6081±0.2247	2.8147±0.8116	2.8338±0.2767
AODV	30	0.7651±0.0553	1.0101±0.3564	4.4233±1.2898	2.9514±0.3241
DSR	30	0.6837±0.0590	4.7600±1.0732	0.4108±0.1401	3.6253±0.3087
OLSR	30	0.7980±0.0349	0.8834±0.3113	0.7137±0.0695	2.4781±0.1618

Table 2. Performance average over all pause times for 100 nodes network for 10-flows and 30-flows

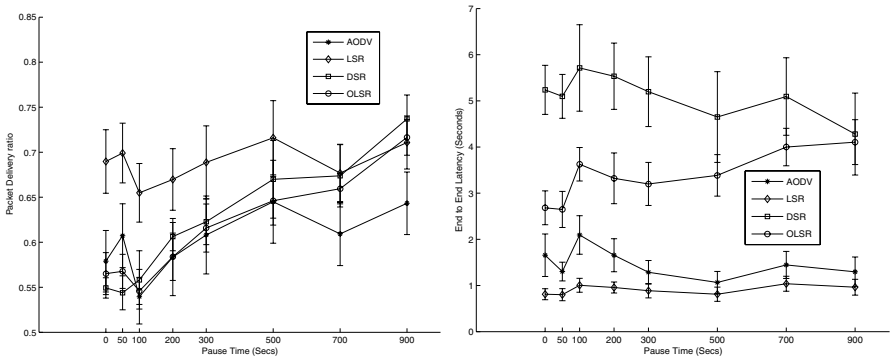
Protocol	Flows	Delivery Ratio	Latency (sec)	Net Load	Data Hops
LSR	10	0.9910±0.0036	0.0383±0.0061	1.1618±0.3221	3.8182±0.3134
AODV	10	0.9882±0.0045	0.0366±0.0095	0.8973±0.2368	3.7441±0.2935
DSR	10	0.8760±0.0505	0.0993±0.0577	0.8599±0.3535	4.2573±0.3170
OLSR	10	0.8218±0.0637	0.0222±0.0020	11.7954±1.5754	3.5838±0.2567
LSR	30	0.6882±0.0356	0.9088±0.1508	10.8084±1.6577	4.4653±0.3530
AODV	30	0.6082±0.0517	1.4558±0.3856	18.2987±13.0698	4.7513±0.4340
DSR	30	0.6183±0.0496	5.1253±0.7820	1.2432±0.4053	6.1410±0.4999
OLSR	30	0.6126±0.0415	3.3714±0.5324	5.4231±0.6695	4.0142±0.2774

Tables 1, and 2 summarize the results of the different metrics by averaging over all pause times for the 50-node and 100-node networks. The columns

show the mean value and 95% confidence interval. LSR has a very consistent performance across all scenarios and outperforms other protocols in most cases. Although the average results are statistically equivalent, the confidence intervals in most cases barely overlap.

In the highest load scenario (100 nodes, 30-flows), LSR has a packet delivery of 0.6882 ± 0.0356, and in the 50-nodes scenario with 30-flows, LSR's delivery ratio shares overlapping confidence intervals with AODV and OLSR. Figure 2(a) shows the delivery ratio for a 100-node network with 30-flows for different pause times. Confidence intervals (95%) are shown with vertical bars in the graphs. As reflected in the summarized average performance, LSR outperforms the other protocols by a wide margin in the high-mobility scenarios; DSR and OLSR have overlapping confidence intervals in the very low mobility scenarios.

In scenarios with 30-flows, LSR has a latency of 0.9088 ± 0.1508 seconds and 0.6081 ± 0.2247 seconds in the 100-nodes and 50-nodes scenario, respectively, better than the other routing protocols on the average. Figure 2(b) shows the data delivery latency for the 100-node network with 30-flows for different pause times. LSR has the lowest data packet latency at high mobility, and shares confidence intervals with AODV at low mobility.



(a) Delivery (100-nodes, 30-flow, 120pps) (b) Latency (100-nodes, 30-flow, 120pps)

Fig. 2. Performance results

The data hop count of all the protocols are comparable. Hence, LSR is delivering more data packets to destinations without dropping them due to congestion-triggered broken routes, and loops. The control overhead of LSR is better than that of AODV in all scenarios. DSR's and OLSR's control overhead cannot be directly compared since DSR uses the optimization to learn source-routes from data packets, and OLSR's control overhead is independent of the flows in the network.

6 Conclusion

We have shown that by using the same route request labels (RRL) needed as flood identifiers (i.e., a source address and an associated sequence number), it is possible to achieve loop-freedom even when nodes other than the destination reply. We presented a new on-demand loop-free routing protocol, the labeled successor routing (LSR) protocol, which labels each node processing and relaying a reply with the RRLs it processes. The source issuing the route-request flood can later identify the nodes that store the RRL as safe loop-free successors to the destination. Simulation results shows that LSR outperforms the on-demand and proactive routing protocols being addressed in the MANET Working Group of the IETF.

References

1. T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol," Request for Comments 3626, October 2003.
2. H. Rangarajan and J.J. Garcia-Luna-Aceves, "Making On-demand Routing Protocols Based on Destination Sequence Numbers Robust," ICC 2005, Seoul, Korea, May 2005.
3. J. J. Garcia-Luna-Aceves and H. Rangarajan, " A New Framework for Loop-Free On-Demand Routing Using Destination Sequence Numbers", The 1st IEEE MASS, October 25-27, 2004, Fort Lauderdale, Florida, USA.
4. D. Johnson et al, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)," IETF Internet draft, draft-ietf-manet-dsr-10.txt, July 2004.
5. H. Rangarajan and J.J. Garcia-Luna-Aceves, "Using Labeled Paths for Loop-free On-Demand Routing in Ad Hoc Networks," *Proc. ACM MobiHoc 2004*, Tokyo, Japan, May 24–26, 2004.
6. C. Perkins et al., "Ad hoc On-Demand Distance Vector (AODV) Routing," Request for Comments 3561, July 2003.
7. I. Chakeres, et al.,"Dynamic MANET On-demand Routing Protocol (DYMO)," IETF Internet Draft, draft-ietf-manet-dymo-00.txt, February 2005.
8. C. Perkins et al. "Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks," *IEEE Personal Communications*, 8(1):16 – 28, Feb 2001.