

# On-demand Loop-Free Routing in Ad Hoc Networks Using Source Sequence Numbers

Hari Rangarajan\*

Email: hari@cse.ucsc.edu

\* Computer Engineering Department  
University of California at Santa Cruz  
Santa Cruz, CA 95064.

J.J. Garcia-Luna-Aceves\*<sup>†</sup>

Email: jj@cse.ucsc.edu

<sup>†</sup>Palo Alto Research Center  
3333 Coyote Hill Road  
Palo Alto, CA 94304.

**Abstract**—In any on-demand routing protocol, sources flood route requests (RREQ) to build routes to destinations, and each new RREQ is identified uniquely with a source-sequenced label (SSL) consisting of the source identifier and a locally generated sequence number. As a RREQ propagates, it creates a directed acyclic graph (DAG), because nodes relay each RREQ only once. We present the first framework for loop-free on-demand routing in ad hoc networks that is based directly on SSLs, rather than on independent mechanisms, which has been the way in which prior on-demand routing protocols have been designed. Extensive simulation results for simple protocol instantiations of our new framework operating in scenarios with 50 and 100-nodes under different traffic patterns show that our new protocols outperform AODV (Ad hoc On Demand Distance Vector), DSR (Dynamic Source Routing), and OLSR (Optimized Link State Routing).

## I. INTRODUCTION

Several routing protocols (e.g., [3] [2] [1]) have been proposed to deal with the fundamental problem of routing data packets across multiple hops in a mobile ad hoc network (MANET) using two different approaches. Pro-active approaches maintain routing information for all destinations, regardless of whether traffic exists for them. Reactive (on-demand) approaches maintain routing information for only those destinations for which traffic exists, and rely on flood search mechanisms to establish routes to “active” destinations. On-demand protocols are very attractive in scenarios with high mobility, and traffic patterns in which source nodes pick a few other nodes as destinations.

On-demand routing protocols establish routes to destinations in two phases: During the *route search* phase, the network is flooded with route requests (RREQ). Each RREQ is uniquely labeled by its source by means of a *source-sequenced label* (SSL), which consists of the identifier of the source and a source sequence number that is locally unique. SSLs prevent any node from processing the same RREQ multiple times, and nodes effectively build a directed acyclic graph (DAG) rooted at the source for a source-destination pair uniquely defined by the SSL and the destination identifier.

During the *route establishment* phase, RREQs received by the destination or an intermediate node with routing state for the destination are answered with route reply (RREP) messages that traverse the loop-free reverse paths along the DAG built in the route-search phase. Each node receiving a RREP establishes or updates its routing state for the destination specified in the RREP. This information is used for data forwarding and route maintenance.

Interestingly, the design of on-demand routing protocols to date has been such that the mechanisms used in the route-search phase to propagate RREQs are fairly independent of the mechanisms used during the route-establishment phase to establish and update routing-table entries for destinations. Clearly, SSLs are a necessity in any on-demand routing protocol for the identification of RREQs and their efficient processing. Furthermore, the loop-free paths that are built during the route-establishment phase must be part of the DAGs built during the route-search phase based on SSLs. This begs the question of whether on-demand loop-free routing can be attained using SSLs during both the route search and the route establishment phases of the routing process. In this paper, we present the first framework for loop-free on-demand routing based directly on SSLs. Our framework supports hop-by-hop packet forwarding, maintains routing tables that are always loop-free, and operates correctly in the presence of state loss, node failures, and unreliable message delivery.

Section II presents an approach for loop-free routing in which the destination must answer all RREQs. Each node relaying a RREQ associates a *relay-sequenced label* (RSL) with the SSL of the RREQ it forwards. The RREPs traversing the reverse paths along the DAG built by the RREQs cause nodes to switch successors and activate the route along this path. However, nodes can be associated with multiple RREQs, and the directed graph associated with the aggregate of all such RREQs need not be acyclic. Hence, to ensure loop-free routes to a destination, nodes use the RSLs to accept a RREP for only one SSL and drop the other replies, which essentially activates successor entries for a destination within a single DAG created by a RREQ SSL. This constitutes the first component of our framework.

Section III introduces the concept of a *viable successor set* (VSS), which is the set of nodes that a given node can safely

<sup>1</sup>This work was funded in part by the Baskin Chair of Computer Engineering at UCSC, the National Science Foundation under Grant CNS-0435522, the UCOP CLC under grant SC-05-33, and by the U.S. Army Research Office under grant No. W911NF-05-1-0246. Any opinions, findings, and conclusions are those of the authors and do not necessarily reflect the views of the funding agencies.

pick as loop-free successors for a given destination. We use this concept to extend the approach introduced in Section II to allow intermediate nodes to send replies to RREQs without creating loops. The basis for this approach is the use of label sets consisting of un-ordered collections of SSLs and RSLs with which nodes can identify viable successors towards destinations.

Section IV presents an alternative approach to extending the basic approach of Section II. This approach uses the SSL and distance to a destination as a new label for the destination, and uses it together with RSLs to ensure loop-free routing while allowing nodes with valid routes to destinations to answer RREQs.

Section V shows how instantiations of our new framework perform compared to two on-demand protocols (AODV, DSR) and a proactive link state protocol (OLSR [1]). Results are presented for 50 and 100-node networks with random traffic flows. Section VI provides our concluding remarks.

## II. ON-DEMAND ROUTING USING SOURCE SEQUENCE NUMBERS AND DESTINATION REPLIES

We present an approach for on-demand routing based on the SSLs and RSLs carried in RREQs, and on RREPs issued only by destinations. We call this approach DSLR (destination-controlled, source-sequenced labeled routing).

### A. Information Stored

Node  $A$  has a unique address for itself ( $A$ ) and maintains a 64-bit source sequence number  $ID_A$  that is strictly increasing, even after reboots. This can be achieved by deriving the source sequence number from a real-time hardware clock.

At node  $A$ , the routing-table entry for destination  $D$  consists, at a minimum, of the successor (next-hop)  $s_D^A$  and the start-identifier  $SID_D^A$ , which is used for checking if a certain neighbor can be used as a loop-free successor.  $SID_D^A$  is set equal to  $ID_D^{*A}$ , which is the last known value of  $ID_A(t)$  at time  $t$  when node  $A$  last added or updated its routing entry for  $D$ . Nodes can use optional metrics for choosing successors. Possible optional entries include the route-cost ( $d_D^A$ ), life-time of the route, and state of the route entry ( $rt_D^A$ ) (which can be valid or invalid). The route entry for a destination can be purged at any time-instant to save memory. When there is no routing-state, the value of  $SID_D^A$  is set to  $ID_A(t)$ , where  $t$  is the time that node  $A$  first originates or relays a RREQ for  $D$ ; otherwise, it is considered to be invalid ( $\infty$ ).

### B. Control Signaling

The basic signaling of DSLR consists of *route requests* (RREQ) sent by sources and forwarded by relays, and *route replies* (RREP) sent only by the destinations. Nodes notify route failures and broken links using *route errors* (RERR). Each RREQ is identified with a *source-sequenced label* (SSL) that consists of a (*source, identifier*) pair, where *source* is the node address originating the RREQ, and *identifier* is the source sequence number  $ID$  created by that node. RREPs generated by destinations in response to RREQs must carry the SSL used

to identify the RREQ. In addition to the SSL, each RREQ and RREP must carry the *relay-sequenced label* (RSL), which is locally assigned by the node relaying the RREQ or RREP, and is only significant within the one-hop neighborhood.

Node  $A$  is said to be *active* in a route computation for destination  $D$  (i.e., the RREQ) when it initiates a RREQ that 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 viable successor for destination  $D$  or the timer for its RREQ expires.

We define the following five rules for nodes to search for routes to destinations. RERR handling is omitted for brevity and is identical to ones used in previous on-demand routing protocols [3]. We use the notation  $ID_A$  to denote the current source sequence number at node  $A$ , and  $id_A^{req}$  and  $id_A^{rep}$  to denote the value carried in a RREQ or RREP.

- *Rule 1:* Node  $A$  must increment its  $ID_A$  each time it relays or originates a RREQ.
- *Rule 2:* If node  $A$  requires a route to destination  $D$ , it issues a RREQ identified by SSL  $(A, ID_A)$ . At the originating node, the RSL and the SSL for the same destination are identical.
- *Rule 3:* If node  $B$  ( $\neq D$ ) receives a RREQ identified by SSL  $(A, id_A^{req})$  from neighbor  $C$ , it caches the RSL  $(C, id_C^{req})$  for this SSL. Node  $B$  processes a RREQ identified by a SSL only once, and forwards a RREQ to its neighbors with the SSL created by the source of the request and its own RSL  $(B, ID_B)$ .
- *Rule 4:* When node  $D$  receives a RREQ from neighbor  $I$  that was issued by source  $A$  for node  $D$  itself, it sends a RREP carrying the same SSL  $(A, id_A^{req})$  of the RREQ, and the RSL  $(I, id_I^{req})$ .
- *Rule 5:* When node  $I$  receives a RREP for destination  $D$  identified by SSL  $(A, id_A^{rep})$  and carrying a RSL  $(I, id_I^{rep})$ , it can use it (if it is feasible) to update its routing table; if it does so, then it must find the pair  $(B, id_B^{rep})$  it cached for this  $(A, id_A^{rep})$ , and send a RREP to neighbor  $B$  with RSL  $(B, id_B^{rep})$  and SSL  $(A, id_A^{rep})$ .

*Theorem 1:* If rules 1 to 5 are followed, RREQs and RREPs do not loop in an error-free network.

*Proof:* For a given route computation  $(A, ID_A)$ , a node may be passive, engaged, or active. A node can become active in a route computation at most once, because it maintains the identifiers it assigns to the RREQs it originates. A router can engage in a route computation only when the the corresponding RREQ identified by  $(A, ID_A)$  has not been previously processed. Hence, any RREQ can traverse only a directed acyclic graph (DAG), which may be a directed tree if no node relays the RREQ more than once, and any path traversed by a RREQ is free of loops.

Because RREPs are forwarded along the reverse path traversed by the corresponding RREQs, it follows that the RREPs must traverse loop-free paths. ■

Note that RREQs may loop if nodes lose their cached state for a computation. However, RREPs will not loop, as long as nodes have a safe loop-free condition when accepting them. Rule 5 requires RREPs to be relayed only if they are accepted and this means that the routing tables must form a loop in order for RREPs to be relayed in loops.

### C. Sufficient Condition for Loop Freedom

Theorem 1 shows that the RREPs travel loop-free reverse paths because the RREQ identified by a unique SSL build a tree rooted at the source. It is easy to see that no loops can form if every node in that reverse path to the source is engaged in only one route computation for the destination. Based on this observation, we obtain a sufficient condition for loop-free routing when multiple RREQs are present by having a node only accept a single computation within a window of computations. After accepting a computation, the node drops all other route computations inside that window, and processes only computations from a new window.

We use the following terminology:  $id(A)_{DB}^A$  denotes the  $id$  from RSL  $(A, id)$  in the RREP for destination  $D$  sent by node  $B$  to node  $A$ .  $id(B)_D^A$  is the  $id$  from RSL  $(B, id)$  in the RREP that  $A$  receives from or transmits to a neighbor.

*Source Sequence-number condition (SSC):* Node  $A$  can change its current successor for destination  $D$  to node  $B$  at time  $t$ , if  $id(A)_{DB}^A(t) \geq SID_D^A(t)$ , where  $SID_D^A(t) = ID_D^{*A}(t)$ .

We establish the following Lemmas ( 1 and 2) when nodes obey rules 1 to 5, and use SSC for switching successors.

*Lemma 1:*  $SID_D^A(t_1) \leq SID_D^A(t_2)$ , where  $t_1 < t_2$ .

*Proof:* We know that  $SID_D^A(t_1) = ID_D^{*A}(t_1)$ . If node  $A$  never updates its routing table till time  $t_2$ , then  $SID_D^A(t_2) = SID_D^A(t_1)$ . On the other hand if node  $A$  updates route-entry for  $D$  at time  $t_2$ , then it can only be the case that  $ID_D^A(t_2) > ID_D^A(t_1)$ . Therefore  $SID_D^A(t_2) = ID_D^{*A}(t_2) > ID_D^{*A}(t_1)$ . Even if there is a reboot or state loss after time  $t_1$ , it is still true at time  $t_2 \geq t > t_1$ , that  $SID_D^A(t) = ID_A(t) > ID_D^{*A}(t_1)$ . Hence, the lemma is true. ■

*Lemma 2:* The event of reporting the value of  $id(B)_D^A$  at time  $t$  to neighbor  $B$  has a causal relation (denoted by  $\rightsquigarrow$ ) with the event that node  $A$  uses a value of  $id(A)_D^A$  to update its routing table at time  $t^-$ , where  $t = t^- + \epsilon$ , assuming that  $\epsilon$  is the processing time for updating the route table.

*Proof:* By Rule 5, node  $A$  can report a RREP at time  $t$  only if it used a RREP to update its routing table. Hence, node  $A$  must have used the value of  $id(A)_D^A$  reported by a RREP at time  $t^-$ . By Rule 3, node  $A$  must have a stored value of node  $B$ 's RSL for this RREP identified by an unique SSL. So, by Rule 5, node  $A$  reports the value of  $id(B)_D^A$  at time  $t$  from the cache after updating its routing table for a time  $\epsilon$ . Therefore, the events are causally related. ■

*Theorem 2:* If nodes use SSC to change successors, no routing table loops can form.

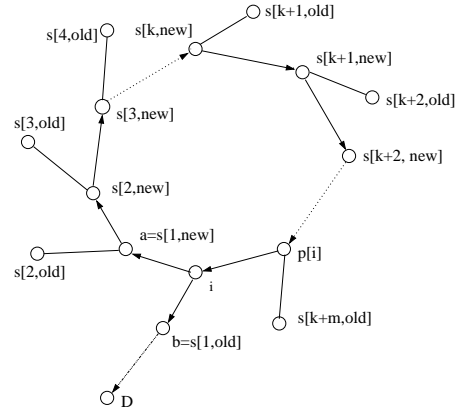


Fig. 1. SSC loop-free condition

*Proof:* The proof is by contradiction. Assume that, before time  $t$ , the directed successor graph for destination  $D$ , which we denote by  $S_D(G)$  is loop-free at every instant, and a loop  $L_D(G)$  is formed at time  $t$ . It is easy to see that no loops can be formed unless atleast one node changes its successor at time  $t$  to a node that is upstream of itself in  $S_D(G)$ .

Assume that  $L_D(t)$  is formed when node  $i$  makes node  $a$  its new successor  $s_D^i(t)$  after processing an input event at time  $t$ , where  $b = s_D^i(t_b) \neq a$  and  $t_b < t$ . Now,  $P_{aD}(t)$  must include  $P_{ai}(t)$ .

Let  $P_{ai}(t)$  consist of the chain of nodes  $\{a = s[1,new], s[2,new], \dots, s[k,new], \dots, i\}$  as shown in Figure 1. The notation indicates that node  $s[k,new]$  is the  $k^{th}$  hop in the path  $P_{ai}(t)$  at time  $t$ , and has node  $s[k+1,new]$  as its successor at time  $t$ .

The last time that node  $s[k,new]$  updates its routing table entry up to time  $t$  and sets  $s_D^{s[k,new]} = s[k+1,new]$  is denoted by  $t_{s[k+1,new]}$ , where  $t_{s[k+1,new]} \leq t$ . Therefore, it is true that  $s_D^{s[k,new]}(t_{s[k+1,new]}) = s_D^{s[k,new]}(t)$ . Because nodes joining  $P_{aD}$  do not switch to any new successors afterwards, it is also true that

$$SID_D^{s[k,new]}(t_{s[k+1,new]}) = SID_D^{s[k,new]}(t)$$

The time when node  $s[k,new]$  sends a reply that constitutes the last reply from such a node that is processed by node  $s[k-1,new]$  up-to time  $t$  is denoted by  $t_{s[k+1,old]}$ . Node  $s[k,new]$ 's successor at time  $t_{s[k+1,old]}$  is denoted by  $s[k+1,old]$ . Note that  $t_{s[k+1,old]} \leq t_{s[k+1,new]} \leq t$ , and that  $s[k+1,old]$  need not be the same as  $s[k+1,new]$ . It is also true that  $SID_D^{s[k,new]}(t_{s[k+1,old]}) \leq SID_D^{s[k,new]}(t_{s[k+1,new]})$ .

From Rule 5 and Lemma 2, when a node  $s[k,new]$  relays a RREP to node  $s[k-1,new]$  at time  $t_{s[k+1,old]}$  after updating its routing table at time  $t_{s[k+1,old]}^-$ , it must be true that

$$id(s[k,new])_{D_{s[k+1,old]}}^{s[k,new]}(t_{s[k+1,old]}^-) < ID_D^{*s[k,new]}(t_{s[k+1,old]})$$

Because SSC must be satisfied when node  $s[k,new] \in P_{aD}(t)$  makes node  $s[k+1,new] \in P_{aD}(t)$  its successor at time  $t_{s[k+1,new]}$ , it must be true that

$$\begin{aligned} id(s[k,new])_{D_{s[k+1,new]}}^{s[k,new]}(t) &= id(s[k,new])_{D_{s[k+1,new]}}^{s[k,new]}(t_{s[k+1,new]}) \\ &\geq SID_D^{s[k,new]}(t_{s[k+1,new]}) \end{aligned}$$

For a loop to be formed after  $t$ , it must be true that  $P_{aD}(t)$  exists. We now derive the following inequality along the path

$P_{ai} \subset P_{aD}$  at time  $t$ , if nodes satisfy SSC when switching successors. We use Lemmas 1 and 2.

$$\begin{aligned}
SID_D^i(t) &= ID_D^i(t) \leq id(i)_{Da}^i(t) = id(i)_{D}^i(t_{s[2,old]}) \rightsquigarrow id(a)_{D}^i(t_{s[2,old]}^-) < \\
ID_D^{*a}(t_{s[2,old]}) &\leq ID_D^{*a}(t_{s[2,new]}) = SID_D^i(t_{s[2,new]}) \leq id(a)_{D}^i(t_{s[2,new]}) = \\
id(a)_{D}^{s[2,new]}(t_{s[3,old]}) &\rightsquigarrow \dots \rightsquigarrow id(s[k,new])_{D}^{s[k,new]}(t_{s[k+1,old]}^-) < \\
ID_D^{*s[k,new]}(t_{s[k+1,old]}) &\leq ID_D^{*s[k,new]}(t_{s[k+1,new]}) = \\
SID_D^{s[k,new]}(t_{s[k+1,new]}) &\leq id(s[k,new])_{D}^{s[k+1,new]}(t) = \\
id(s[k,new])_{D}^{s[k+1,new]}(t_{s[k+1,old]}) & \\
\rightsquigarrow \dots \rightsquigarrow id(i)_{D}^i(t_b^-) &< ID_D^{*i}(t_b) \leq ID_D^{*i}(t) = SID_D^i(t)
\end{aligned}$$

The invariant conditions along this path lead to the erroneous conclusion that  $SID_D^i(t) < SID_D^i(t)$ . Hence, no loops can be formed when SSC is applied. ■

### D. Basic Route Maintenance

1) *Route Establishment*: Routes to destinations are established on demand when data packets destined for that destination are received. Node  $A$  that is *active* for destination  $D$  buffers such data packets. However, if node  $A$  is *passive* for destination  $D$  then it must become *active* and issue a RREQ as per Rule 2. Node  $A$  maintains a *RREQ timer* that is set to  $(2.ttl.latency)$  for every destination for which it is active, where  $ttl$  is the time-to-live of the broadcast flood and  $latency$  is the estimated per-hop latency of the network. If no usable RREPs are received, node  $A$  resends new RREQs with an increased  $ttl$  after the expiry of its timer. 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.

2) *Updating Routing Tables*: Node  $I$  sets  $s_D^I \leftarrow B$  when it accepts a RREP from neighbor  $B$  that satisfies SSC. If it has an associated route metric, it updates  $d_D^A \leftarrow d_D^{rep} + lc_B^A$ . Note that nodes can chose to accept only RREPs that will result in shorter-cost paths, although it is not necessary.

### E. Termination Properties

*Theorem 3*: All nodes in a connected component  $G$  not containing destination  $D$  invalidate their route entries for node  $D$  within a finite time.

*Proof*: From Lemma 1, RREPs generated by the destination cannot traverse loops. A finite time  $t$  after node  $D$  is partitioned from nodes  $n \in G$ , all RREPs must have been processed at the sources that originated the RREQs, and no more RREPs can be present in the network. The DASG for  $D$  defined by the successor entries of nodes in  $G$  is loop-free (Theorem 2), and in a finite time all nodes in the DASG must be notified with a route error (RERR) stating the unreachability of  $D$ . ■

*Theorem 4*: In a stable error-free connected network, a source  $S$  will establish a route to a destination  $D$  in finite time.

*Proof*: From Lemma 1, RREPs generated by the destination  $D$  will travel a loop-free reverse back path to the source

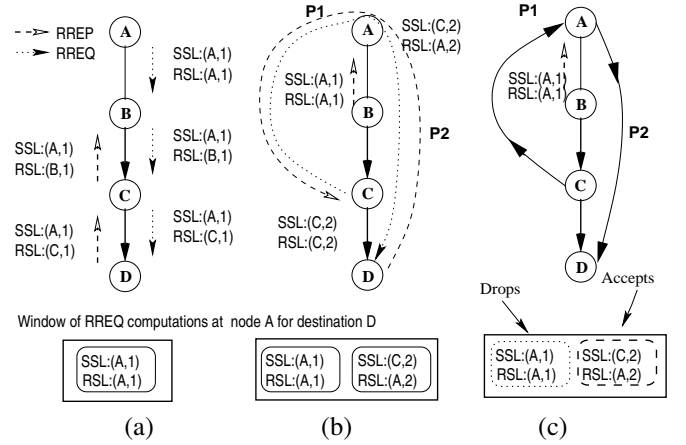


Fig. 2. Using SSLs and RSLs

$S$ . If every node along the path updates its routing table for  $D$  and relays the RREP, then the theorem is true. However, if a node is *engaged* in multiple route computations for  $D$ , then it may not satisfy SSC, and the RREP will be dropped. However, in such a case, if a node is engaged for 'n' different route computations, atleast one of them succeeds. Because along any path to the destination from the different sources, one RREP is always relayed, atleast one source must always establish a path to the destination irrespective of how many other route computations are on-going at the same time. Given that there are only a finite number of sources (nodes) in the network, and they retry new RREQs upon failure, eventually all sources must establish routes to the destination. ■

### F. Non-caching Option

Caching of the RSL associated with a RREQs SSL can be avoided at the relay nodes if the RREQs and RREPs themselves carry the entire list of RSLs generated at every relay node. This is beneficial when nodes have limited storage capacity. The message structure of such a RREQ or RREP would be a tuple  $\{SSL, RSL_1, RSL_2, \dots, RSL_n\}$ , where SSL is assigned by the originating node, and each  $RSL_i$ , where  $i \in \{1, \dots, n\}$ , is appended by the relaying node  $n_i$  and is carried as a list, rather than being cached at the intermediate nodes along the path. It is also possible for mixed-mode operation where some set of nodes can operate with the cache, while the ones that cannot can force the previous hop RSL to be carried in the message. For example, node  $n_i$  can flag a bit indicating that  $RSL_{n_{i-1}}$ , along with its new  $RSL_{n_i}$  must be carried in the RREQ. This ensures that enough information is present in the RREP generated so that node  $n_i$  can process and relay the RREP to node  $n_{i-1}$ .

### G. Example

Figure 2(a) shows the sequence of events in a four-node network when node  $A$  initiates a RREQ for destination  $D$  identified by SSL (A,1) along with a RSL (A,1) (that is the same as the SSL at the origin). Node  $B$  caches the RSL (A,1) for the SSL (A,1) and sends the RREQ with its RSL

(B,1). Similarly, node  $C$  caches the RSL (B,1) and sends out a RREQ with RSL (C,1). Destination  $D$  generates a RREP which is processed by nodes  $B$  and  $C$ . Assuming that the network has just begun operation, when the nodes  $A$ ,  $B$ , and  $C$  transmitted the RREQ they must set themselves a value of one for  $SID_D^A$ ,  $SID_D^B$ , and  $SID_D^C$ , respectively. Node  $C$  can accept the RREP (A,1) with a RSL (C,1) because SSC is satisfied and will switch its successor to node  $D$ . Similarly, node  $B$  will switch successors to  $C$ . Nodes  $B$  and  $C$  will set their respective  $SID_D^B$  and  $SID_D^C$  to two after updating their routing tables for  $D$ . Now, assume that the RREP relayed by node  $B$  is queued at the MAC layer at time  $t_1$ . Figure 2(b) shows the sequence of events after time  $t_1$ , when node  $C$  can no longer reach  $D$  due to a link failure. Node  $C$  sends a new RREQ that traverses the paths  $P_1$  and  $P_2$  through node  $A$  to reach destination  $D$ . The RREQ is identified by SSL (C,2) and node  $A$  relays it with a RSL (A,2). Note that node  $A$  still has a stored value of one for  $SID_D^A$ .

Figure 2(c) shows the state of the network at a time  $t_2 > t_1$  when a RREP is issued by the destination identified by (C,2) and the nodes along path  $P_1$  and  $P_2$  update their routing tables to establish a route to node  $D$ . Node  $A$  sets  $SID_D^A \leftarrow 3$  to establish its successor path along  $P_2$  after updating its routing table. When the RREP identified by SSL (A,1) queued at node  $B$  is finally transmitted and received by node  $A$ , it can form a loop if node  $A$  decides to switch successors to  $B$ . However, because SSC must be satisfied, node  $A$  can only accept RREPs carrying a RSL starting from (A,3) as  $SID_D^A = 3$ . Therefore, the RREP will be dropped and no loops are formed.

Figure 2 also shows the window of RREQ computations that node  $A$  is engaged or active for destination  $D$ . Node  $A$  becomes a part of two different DAGs created by the RREQs. However, the directed graph formed by the aggregate of RREQs with SSL (A,1) and (C,2) is not acyclic. Inside this computation window, node  $A$  becomes only a part of the DAG created by RREQ SSL (C,2), and drops the SSL (A,1). The window can consist of more RREQ computations, but only one of them is used, and the rest of the computations in the current window are dropped.

### III. ON-DEMAND ROUTING USING SOURCE SEQUENCE NUMBERS AND REPLIES FROM NODES WITH VALID ROUTES

We extend the basic framework of the previous section by deriving labels out of the SSL and RSLs carried in RREPs. These labels are then used to identify neighbors as viable loop-free successors to the destination, which allows intermediate nodes to generate replies. We call this approach Source-sequenced Labeled Routing (SLR).

For the purposes of discussing SLR, we use the non-caching version of routing based on SSLs presented in Section II-F. We later show how it can be simplified to the caching version.

#### A. Viable Successor Sets

We consider the generalized class of on-demand hop-by-hop ad hoc routing protocols that use a RREQ flood identified by

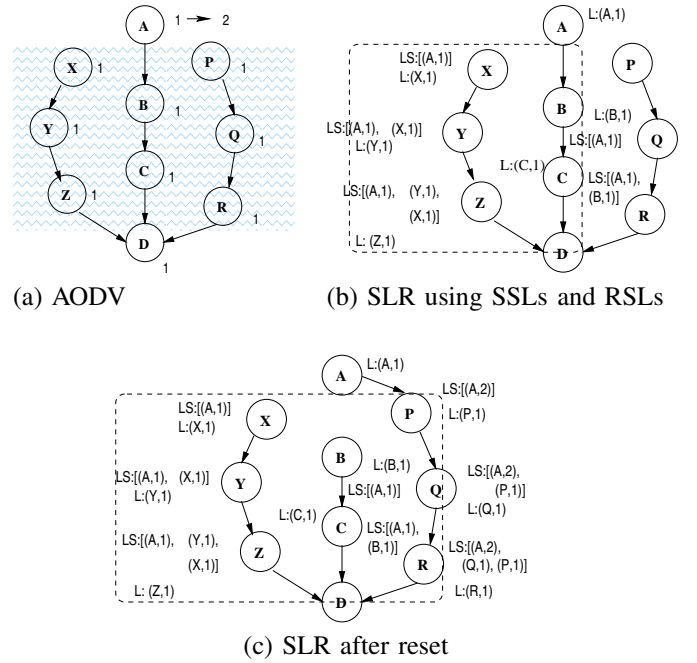


Fig. 3. Viable Successor Set Dynamics

an unique SSL to build a tree rooted at the source, and label one or more of the reverse paths traversed by RREPs along the tree. We define a *viable successor set* for a destination  $D$  at a node  $A$  at time  $t$ , denoted by  $VSS_D^A(t)$ , as the set of nodes that node  $A$  can use as a successor to destination  $D$  without causing any loops. We illustrate how VSS varies as a function of time when using a labeling scheme such as AODV.

Figure 3(a) shows a possible assignment of destination sequence numbers (as labels) when AODV is used as the routing protocol for a ten-node network, after node  $A$  attempts a route discovery for destination  $D$  and establishes a route at time  $t$ . Node  $A$  uses  $B$  as its next hop for destination  $D$  because it offers the shortest cost path. If at a later time  $t_1$ , link  $AB$  fails, then node  $A$  increases its destination-sequence number to two. Any new RREQs from node  $A$  can only be answered by a node that stores a destination-sequence number greater than or equal to two. Because of the labeling adopted by the nodes,  $VSS_D^A$  becomes empty ( $\emptyset$ ) at time  $t_1$  as none of the nodes in the shaded rectangle can satisfy the condition for generating a RREP. Consequently, RREQs from node  $A$  have to be answered by the destination.

#### B. Labeling with Source Sequence Numbers

To allow nodes with active routes to destination  $D$  to answer RREQs in SLR, each node stores a *label set* for the destination, which is an un-ordered collection of source-sequenced labels and relay-sequenced labels, together with its self source-sequenced label for the destination. The self source-sequenced label and label set for destination  $D$  maintained at node  $A$  are denoted by  $L_D^A$  and  $LS_D^A$ , respectively. The label set serves the purpose of other nodes identifying node  $A$  as a viable successor towards destination, while the stored self

sequenced-label is used by  $A$  to identify other nodes as safe viable successors. When routing state is lost, it is considered that  $LS_D^A = \emptyset$  and  $L_D^A = (A, \infty)$ . Nodes can also choose to drop any of the elements from their label sets for a destination at any time without affecting correct operation. We use the term sequenced-label (SL) to refer to a SSL or RSL.

We define the following operator ( $\succeq$ ) to determine if a sequenced-label  $SL_1 = (src_1, id_1)$  is fresher than another  $SL_2 = (src_2, id_2)$  as follows:

$$SL_1 \succeq SL_2, \text{ if } src_1 = src_2 \wedge id_1 \geq id_2$$

*Labeling Rule:* Let node  $n_i$  receive a RREP satisfying SSC for destination  $n_k$  carrying a list of SLs  $[(n_1, id_1), (n_2, id_2), \dots, (n_{i-1}, id_{i-1}), (n_i, id_i), \dots, (n_{k-1}, id_{k-1})]$ , where  $(n_1, id_1)$  is the SSL and the other sequenced-labels are RSLs appended by relaying nodes. Node  $n_i$  performs the following steps:

- Node  $n_i$  must set  $LS_{n_k}^{n_i} = LS_{n_k}^{n_i} - SL$ , if  $\exists SL'$ , such that  $SL' \in \text{RREP} \wedge SL' \succeq SL$ .
- Node  $n_i$  can assign itself a label set  $LS_{n_k}^{n_i} \subseteq LS_{n_k}^{n_i} \cup \{(n_1, id_1), (n_2, id_2), \dots, (n_{i-1}, id_{i-1})\}$ .
- Node  $n_i$  can set  $L_{n_k}^{n_i} = (n_i, id_i)$ . However, if node  $n_i$  relays the RREP, it must set  $L_{n_k}^{n_i} = (n_i, id_i)$  or to  $(n_i, \infty)$ .

We denote the individual elements of the label  $L$  with  $src^L$  and  $id^L$ . RREPs carry a label set ( $LS$ ) which is set to  $LS_D^A$ , where node  $A$  is transmitting the RREP for a destination  $D$ . The label set stored at node  $A$  for destination  $D$  reported by a neighbor  $B$  is denoted by  $LS_{DB}^A$ , and  $ID(LS_I)_{DB}^A$  is used to represent the  $id_I$  of the SL  $(I, id_I)$  reported in the label set.

*Source-Sequenced Labeling Condition (SSLC):* Node  $A$  can switch successors to its neighbor  $B$  for destination  $D$  at time  $t$ , if it is true that  $\exists L$ , such that  $L \in LS_{DB}^A(t)$  and  $L \succeq L_D^A(t)$  (i.e.,  $ID(LS_A)_{DB}^A(t) \geq id_D^A(t)$ ).

We establish the following lemmas (3, 4, and 5) for the labels stored at a node  $A$  when SSC is used for RREP processing, and *Rules 1 to 5* and the *labeling rule* are adhered to. We assume in Lemma 4 and 5 that there exists a node  $I$  in the network that is active or engaged in the route computation that node  $A$  is also engaged for.

*Lemma 3:*  $id_D^A(t_1) \leq id_D^A(t_2)$ , where  $t_1 < t_2$ .

*Proof:* Node  $A$  must change its label  $L_D^A$  for destination  $D$  only if it processes and relays a RREP. To accept a RREP at time  $t$  as per SSC, it must be true that the RREP carries a SL  $(A, id_A)$  such that  $SID_D^A(t) \leq id_A$ . Node  $A$  will re-label  $L_D^A = (A, id_A)$ , and can only accept RREPs carrying a SL  $(A, id)$ , where  $id > SID_D^A(t)$ , for SSC to be satisfied. This is true even after reboots or state loss from Lemma 1. Hence, the value of  $id_D^A$  is strictly non-decreasing with time, and the lemma is true. ■

*Lemma 4:*  $ID(LS_i)_{DB}^A(t_1) \leq ID(LS_i)_{DB}^A(t_2)$ , where  $t_1 < t_2$ .

*Proof:* For  $(I, id_I) \in LS_D^A$ , it must be true that node  $A$  engaged in a route computation for which node  $I$  is active or engaged as well. The proof is by contradiction. Let  $(I, id_I^1)$  and  $(I, id_I^2)$  be two SLs such that  $id_I^1 < id_I^2$ , and  $(I, id_I^2) \in LS_D^A$ . We will now prove that node  $A$  cannot accept a RREP

carrying a SL  $(I, id_I^1)$  and modify  $LS_D^A$ . Node  $A$  must have two SLs  $(A, id_A^1)$  and  $(A, id_A^2)$  engaged in a route computation along with node  $I$ 's SLs to receive the RREPs; although the relation between  $id_A^2$  and  $id_A^1$  is not known. After accepting the RREP carrying a SL  $(I, id_I^2)$ , node  $A$  must have set  $SID_D^A > \max(id_A^1, id_A^2)$ . This is true even after reboots or state loss because of Lemma 1. When node  $A$  receives the RREP carrying a SL  $(I, id_I^1)$ , it cannot satisfy SSC. Hence, node  $A$  will not modify its label set  $LS_D^A$ . Therefore, the lemma is true. ■

*Lemma 5:* A correspondence (denoted by  $\rightsquigarrow$ ) exists between the value of  $ID(LS_I)_{DB}^A$  reported at time  $t$  and the value of  $id_D^A$  at time  $t^- < t$ , where  $t^-$  denotes the time when  $ID(LS_I)_{DB}^A$  was added to or modified in the label set  $LS_D^A$ .

*Proof:* For  $ID(LS_I)_{DB}^A$  to be reported at time  $t$ , it must have been added to the label set at a time earlier than  $t$  when node  $A$  must have accepted a RREP satisfying SSC. Lemma 4 shows that the value of  $ID(LS_I)_{DB}^A$  does not decrease with time, and hence  $t^-$  must be the last time instant when  $ID(LS_I)_{DB}^A$  was modified or added to  $LS_D^A$ . As per labeling rules, there must exist a defined value for  $id_D^A$  at time  $t^-$ , although it can be modified at a later time. Hence, the value of  $ID(LS_I)_{DB}^A$  reported at time  $t$  corresponds to  $id_D^A$  at time  $t^-$ . ■

*Theorem 5:* If nodes follow labeling rules and use SSLC to change successors, then no routing-table loops can be formed.

*Proof:* Using the same argument as in Theorem 2, we derive the following inequalities along path  $P_{ai} \subseteq P_{aD}$  when nodes follow labeling rules and use SSLC to switch successors. We use Lemmas 5 and 3.

$$\begin{aligned} id_D^A(t) &\leq ID(LS_i)_{Da}^i(t) = ID(LS_i)_{Da}^a(t_{s[2,old]}) \rightsquigarrow id_D^A(t_{s[2,old]}^-) \leq \\ &id_D^A(t_{s[2,old]}) \leq id_D^A(t_{s[2,new]}) \leq ID(LS_a)_{Ds[2,new]}^a(t) \\ &= ID(LS_a)_{Ds[2,new]}^a(t_{s[3,old]}) \rightsquigarrow \dots \rightsquigarrow id_D^A(t_{s[k+1,old]}^-) \leq \\ &id_D^A(t_{s[k+1,old]}) \leq id_D^A(t_{s[k+1,new]}) \leq \\ ID(LS_s[k,new])_{Ds[k,new]}^s(t_{s[k+1,old]}) &= ID(LS_s[k,new])_{Ds[k+1,new]}^s(t_{s[k+1,old]}) \rightsquigarrow \\ \dots \rightsquigarrow id_D^A(t_{s[k+m,old]}^-) &\leq id_D^A(t_{s[k+m,old]}) \leq id_D^A(t_i) \leq \\ ID(LS_p[i])_{Di}^p(t) = ID(LS_p[i])_{Di}^i(t_b) &\rightsquigarrow id_D^A(t_b^-) id_D^A(t_b) \leq id_D^A(t) \end{aligned}$$

This leads to the erroneous conclusion that  $id_D^A(t) < id_D^A(t)$ . Hence, no loops can be formed. ■

### C. Simplified Labeling

We have previously introduced the Labeled Successor Protocol (LSR) [4], which can be considered as a simplified version of SLR's labeling scheme making use of only the SSL of a RREP. LSR can be derived from SLR using a subset of the original labeling rules as follows: When node  $A$  accepts a RREP satisfying SSC identified by SSL  $(S, ID_S)$  for destination  $D$ , it performs the following steps:

- Node  $A$  must set  $LS_D^A = \{(S, ID_S)\}$ .

- Node  $A$  can set  $L_D^A$  to  $(S, ID_S)$ , if  $S = A$ . If the RREP is relayed, then node  $A$  must set  $L_D^A = (A, \infty)$ .

The two stored labels used can be replaced with one as is the case of LSR [4] because it can be seen clearly that only one of the two labels are useful for loop-free checks. LSR lacks a mechanism to sequence RREPs received from the destination correctly. The simplified labeling of SLR used to realize LSR does not suffer from this limitation.

#### D. Example

Figure 3(d) shows an example of VSS dynamics using SLs. Source  $A$  starts a RREQ flood with SSL  $(A, 1)$ , which gets relayed along paths  $XYZ$  and  $BC$  after the nodes forward it with their respective RSLs. The RREPs generated by the destination (for the purposes of this example, we assume the destination replies to all received requests) are processed and the nodes set their label sets  $LS$  as shown in the figure. There is no explicit ordering of nodes here, and despite the higher hop count of  $X$ , node  $A$  can still switch to  $X$  as a viable successor applying SSLC. Here, the  $VSS_D^A(t) = \{X, Y, Z, B, C\}$ . Note that, in-addition to node  $A$ , other nodes can also identify their respective viable successors for the destination. Assume that at time  $t_1$ , node  $A$  labels path  $PQR$  as viable successors as shown in Figure 3(e). Because node  $A$  does not relay the RREP, it can still retain its old  $L_D^A = (A, 1)$ . Despite switching to a new path, node  $A$  can still use all the old successors at a later time and the  $VSS_D^A(t_1)$  is the complete set  $\{B, C, X, Y, Z, P, Q, R\}$ . Node  $A$  is able to determine all the nodes that were previously labeled as viable successors. However, as per the labeling rules, if node  $A$  had relayed the reply, it has to relabel  $L_D^A$  to  $(A, 2)$ , which will force  $A$  to lose viable successors from its old RREQ  $(A, 1)$ . The other reason the VSS can lose successors at a later time is if the relay nodes along the path drop SLs from their label set.

Figure 4(a) shows another example of the labeling for a network where source  $A$  has a route to destination  $D$ . At a later time, due to network mobility, node  $C$ 's link to  $D$  fails. Node  $C$  re-establishes a route to  $D$  through a path  $CEAFD$ , where  $E$  and  $F$  are new nodes that are in this vicinity due to mobility. Figure 4(b) shows the labeling at this time. Subsequently, if node  $B$ 's link to  $C$  fails, then node  $B$  establishes a new route through  $BAFD$ . Figure 4(c) shows the labeling at this time. Note that the re-labeling occurs in these cases because the RREP is generated by the destination. Note that in both these cases, the destination is the only node answering because node  $C$  or  $B$  cannot identify any viable successors. The label sets stored allow nodes to be identified as loop-free successors for a destination when SSLC is used. Figure 5 shows the labeling for the same set of events when LSR (using the simplified labeling scheme of SLR) is used as the routing protocol.

#### E. Route Search

We present three conditions for SLR that are used by nodes to search for loop-free viable successors across a single-hop or multiple-hops to find a route to the destination.

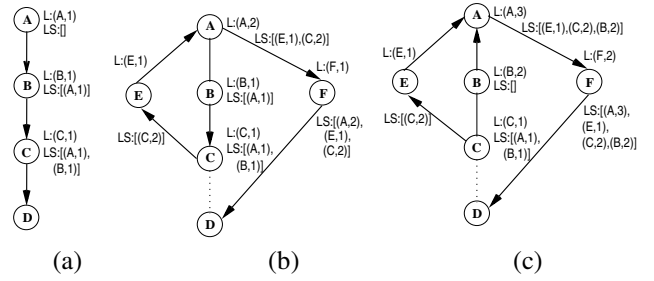


Fig. 4. SLR labeling

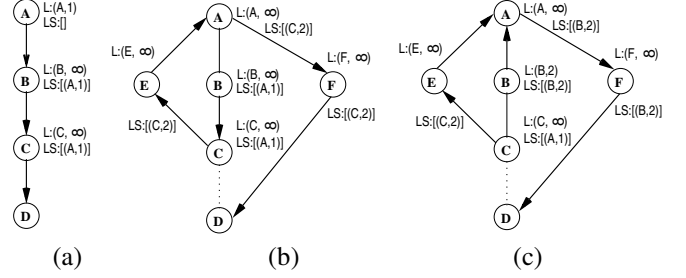


Fig. 5. LSR labeling

We redefine the is-a-subset-of or equal-sets operation ( $\subseteq$ ) between two label sets  $LS_1$  and  $LS_2$  as follows:

$$LS_1 \subseteq LS_2, \text{ if } \forall SL, SL \in LS_1, \text{ it is true that} \\ \exists SL', \text{ such that } SL' \in LS_2 \wedge SL' \succeq SL$$

Each RREQ carries a *common label set*  $CLS$  that represents the collection of self SLs of each node that transmitted the RREQ. The purpose of the  $CLS$  is to find a loop-free path from a successor whose RREP is usable at every node along the reverse path to the originating node. We use superscripts  $req$  and  $rep$  to represent the values carried in RREQs and RREPs, respectively. The conditions for initiating RREQs, relaying RREQs, and generating RREPs are as follows:

**RLSC:** (Reset Labeled Successor Condition). If node  $A$  must change  $s_D^A$  (after a route failure or if  $s_D^A = \phi$ ), then it must send a RREQ carrying  $CLS_D^{req} = L_D^A$ .

**GLSC:** (Generate Labeled Successor Condition). Node  $I$  can issue a RREP responding to a RREQ  $req$  for destination  $D$  if  $I$  has an active route to  $D$ , and  $CLS_D^{req} \subseteq LS_D^I$ .

**CLSC:** (Common Labeled Successor Condition). When node  $A$  relays a RREQ for destination  $D$ , it sets  $CLS_D^{req} = CLS_D^{req} \cup L_D^A$  in the relayed RREQ iff  $CLS_D^{req} \subseteq LS_D^A$ . Otherwise,  $CLS_D^{req}$  is set to  $\emptyset$ .

We illustrate an example of how the RREQ search progresses across multiple hops to find an intermediate node that can reply. Assume all nodes in Figure 3(e) except node  $R$  have expired their route for destination  $D$ . Node  $A$  sends a RREQ with  $CLS_D^{req} = (A, 1)$ . For simplicity we consider path  $PQR$ . Node  $P$  relays the RREQ with  $CLS_D^{req} = [(A, 1), (P, 1)]$  as per CLSC because  $(A, 1) \subseteq LS_D^P$ . Similarly, node  $Q$  relays the RREQ with  $CLS_D^{req} = [(A, 1), (P, 1), (Q, 1)]$ . Now, GLSC allows node  $R$  to initiate a RREP that will satisfy SSLC at

every one of the nodes Q, P, and A when the RREP traverses the reverse path.

When the simplified labeling scheme of SLR is used in LSR, nodes can only identify neighbors as successors. Route searches progressing more than a single-hop can only be answered by the destination.

#### F. Termination Properties

The proof that all nodes will invalidate their routing entries for a destination that is partitioned follows directly from Theorem 3 which shows that the property holds when the destination is the only node that can generate replies. In SLR, according to Theorem 5, the DASG is instantaneously loop-free and no nodes ever choose any nodes upstream in the DASG. This means that the RERRs that propagate along the DASG will force all nodes to invalidate their route entries for the partitioned destination.

*Theorem 6:* In an error-free stable connected network, a source will establish a route to a destination in finite time.

*Proof:* Let source  $A$  issue a RREQ for destination  $n_1$  that traverses a path  $P = \{n_k, n_{k-1}, \dots, n_i\}$  (where  $n_i$  can be  $n_1$ ), before reaching the destination  $n_1$  or a node that satisfies SLSC. If  $CLS_{n_1}^{req} = \emptyset$ , then the RREQ can only be answered by the destination, and the proof follows that of Theorem 3. Otherwise, when the RREP is transmitted along the reverse path to node  $n_{i-1}$  by node  $n_i$ , it is true that  $LS_D^{rep} = LS_D^{n_i} \supseteq CLS_D^{req}$  because of GLSC and CLSC. Hence, node  $n_{i-1}$  must be able to accept the RREP, which will satisfy SSLC. The same argument holds at every node that relays the RREP along the reverse path to source  $A$ . If a node along path  $P$  modifies its label set by processing another route computation, then the RREP will not be accepted and will not be relayed to the source. However, sources retry RREQs and there are only a finite number of nodes in the network. From the same argument as in Theorem 4, each source must be able to establish a route to the destination. ■

### IV. ON-DEMAND ROUTING USING SSLs AND DISTANCE INFORMATION TO ALLOW REPLIES FROM NODES WITH VALID ROUTES

As the last component of our loop-free routing framework based on SSLs and RSLs, we present an approach with which nodes with valid routes to a destination are allowed to answer RREQs using SSLs and distance information rather than label sets, which may become large in some scenarios. In this alternative approach to SLR, which we call Labeled Source-sequenced Routing with Distances (LSR-D), distances are paired with SSLs to create a single label that we call *source-sequenced distance label* (SSDL). SSDLs create a relative ordering of the distances along the path in which nodes are engaged for a particular RREQ SSL, and the source of the SSL can identify all nodes in the path as viable successors regardless of the distances. Showing that SSDLs can be safely used to maintain loop-freedom can be proven using an approach similar to the one presented for the prior approaches, and is omitted due to space limitations.

#### A. Sufficient Conditions for Loop-freedom

Each source-sequenced distance label (SSDL) is a tuple [(SSL), distance]. We now present the associated terminology, and operations on these labels. The freshness operator ( $\succ$ ) between two labels,  $SSDL_1 = [(src_1, id_1), d_1]$ , and  $SSDL_2 = [(src_2, id_2), d_2]$  is defined as follows:

$$SSDL_1 \succ SSDL_2 \text{ if } (src_1 = src_2 \wedge id_1 > id_2) \vee \\ (src_1 = src_2 \wedge id_1 = id_2 \wedge d_1 < d_2)$$

We denote the label stored for a known destination  $D$  at node  $A$  by  $SSDL_D^A$ . An invalid SSDL at a node  $A$  is considered to be  $[(A, \infty), 0]$ . The SSDL reported to node  $A$  by neighbor  $B$  for destination  $D$  is denoted by  $SSDL_{DB}^A$ . Each RREP carries the SSDL and distance metric ( $d$ ) at the relaying node for the destination denoted by  $d_D^{rep}$ . We denote the cost of the link from node  $A$  to  $B$  with  $lc_B^A$ .

*Distance Labeling Rule (DLR):* When node  $A$  accepts a RREP from neighbor  $B$  for destination  $D$  that satisfies SSC and that is identified by SSL  $(S, ID_S)$ , node  $A$  must set  $SSDL_D^A = [(S, ID_S), d']$ , where if  $S=A$  then  $d' = \infty$  else  $d' = d_D^{rep} + lc_B^A$ . Node  $A$  can choose not to modify a valid  $SSDL_D^A$  if it does not relay the RREP.

DLR allows nodes to assign or modify (reset) the stored SSDL for a destination. This may be done after the loss of state, or if the SSDL stored can no longer be used to determine any viable successors. Note that DLR requires SSC to be satisfied; therefore, nodes must still use the RSLs to determine which RREPs to accept and such RREPs must be generated by the destination. However, if nodes have valid SSDLs, they can choose a neighbor as a safe loop-free successor by comparing the freshness of the SSDLs as given by this sufficient condition for loop-freedom.

*Source-Sequenced Distance Labeling Condition (SSDLC):* Node  $A$  can switch successors to its neighbor  $B$  for destination  $D$  at time  $t$ , if it is true that  $SSDL_{DB}^A(t) \succ SSDL_D^A(t)$ .

Figure 6 shows the VSS for the same scenario discussed in Section III-D when SSDLs are used for labeling. We assume link costs to be unity. As the figure illustrates, a relay node  $B$  with SSDL  $[(A,1),2]$  can identify  $C$  with SSDL  $[(A,1),1]$  as a viable successor without using extensive label sets. Node  $A$  can still identify all nodes as viable successors, because its SSDL is set to the highest distance ( $\infty$ ). SSDLs also allow nodes to identify viable successors along different paths; for example,  $B$  can identify  $Y$  and  $Z$ .

#### B. Route Search

Each RREQ carries the freshest of the SSDLs stored at the nodes along the path traversed by the RREQ, which is denoted by FSSDL. We define an in-order operator ( $\sqcap$ ) between two SSDLs,  $SSDL_1$  and  $SSDL_2$ , as follows:

$$\sqcap(SSDL_1, SSDL_2) = \begin{cases} SSDL_2, & \text{if } (SSDL_2 \succ SSDL_1) \\ \phi, & \text{otherwise} \end{cases}$$

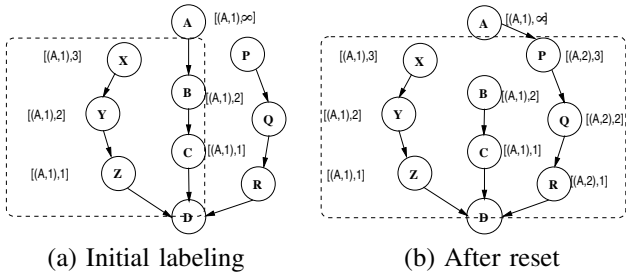


Fig. 6. Labeling with SSDLs

We briefly describe conditions similar to that of SLR for nodes to search for routes across one or more hops and when intermediate nodes with valid routes can reply. A node  $A$  issues a RREQ with  $FSSDL_D^{req} = SSDL_D^A$  for destination  $D$ , and node  $A$  relays RREQs with  $FSSDL_D^{req} = \sqcap(FSSDL_D^{req}, SSDL_D^A)$ . An intermediate node  $I$  having a valid active route can reply to a request if  $SSDL_D^I > FSSDL_D^{req}$ . A local-repair operation can be performed by intermediate nodes to repair routes locally using a neighbor query that is a RREQ with  $ttl$  set to one.

The in-order operator ( $\sqcap$ ) is used when relaying RREQs, because a previously expired path to the destination can be activated without modifying the stored SSDLs. For example, in Figure 6, node  $A$  can search a route to  $D$  through a path  $PQR$  because the the SSDLs are in-order and every RREQ will be relayed with the stored SSDL. However, if the RREQ traverses a path  $PBC$ , then  $FSSDL$  will be set to  $\phi$ , and the RREQ can only be answered by the destination, which will force the nodes along the path to reset their SSDLs. A local-repair can be performed by an intermediate node  $B$  without sending a RERR to source  $A$  when its current link to  $C$  fails. A one-hop RREQ query sent will be answered by node  $Y$  or  $Q$ .

## V. PERFORMANCE

We present results over varying loads and mobility for an instantiation of DSLR that selects shortest-cost paths, LSR, which adopts the simplified labeling scheme of SLR, and LSR-D, which uses SSDLs. We also present results for LSR-D-LR, which is LSR-D with the local-repair scheme. The protocols used for comparison are two on-demand protocols, DSR and AODV, and OLSR [1], which is a pro-active link state protocol. Simulations are run in Qualnet 3.5.2. AODV, DSLR, LSR, LSR-D, and LSR-D-LR, use an expanding-ring search scheme when flooding RREQs. AODV and DSLR set the  $ttl$  of the RREQs to the last-known hop-count of the destination.

Simulations are performed for 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 between randomly picked sources and destinations. 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 indicates that more data packets traverse more hops without reaching the destination necessarily.

Tables I and II summarize the results of the different metrics by averaging over all pause times for the 50 and 100 node networks with random flows. The columns show the mean value and 95% confidence interval. All our performance discussions focus on the average case because the confidence intervals overlap atleast slightly in most cases. The packet delivery ratio, the end-to-end delay, and the control overhead over various pause times is shown for a 100-node networks with 30-flows in Figure 7 (results for LSR-D are not shown). The vertical bars in the graphs indicate the 95% confidence intervals.

The performance results show that DSLR, LSR, and LSR-D outperform AODV, DSR, and OLSR. LSR has better packet delivery and lower control-overhead than DSLR across the different scenarios. This is because the labeling in LSR allows one-hop neighbors of the source to initiate RREPs, thus avoiding RREQ floods, whereas RREPs in DSLR can be initiated by the destination only. Note that if the elaborate labeling scheme of SLR is used, it is also possible for intermediate nodes that are across multiple hops from the source to initiate RREPs. The latency of DSLR is slightly better than of LSR because DSLR establishes more optimal (shortest-cost) paths because RREQ floods search for the destination after a route failure. In the case of LSR, the replies from intermediate nodes need not necessarily reflect the best path when nodes are mobile. The optimal forwarding of packets is also shown in the data-hops metric of DSLR, which is slightly lower than that of LSR. LSR-D's performance is equivalent to that of LSR across the different scenarios. However, the local-repair of LSR-D-LR shows a noticeable improvement in performance, particularly, in the 100-node, 30-flow scenarios where excessive RREQ flooding can cause congestion. The local-repair allows nodes to resolve failure without reporting a RERR to the source, which will then retry flooding RREQs. The key feature is that the RREQ sent with a  $ttl$  of one can elicit a reply from one of the neighbors. Although, AODV supports a local-repair operation, it floods the RREQ to locate the destination.

The performance of AODV, DSR, and OLSR suffer from

TABLE I

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	
	10	30	10	30	10	30	10	30	10	30
DSLRL	0.996±0.001	0.830±0.035	0.016±0.002	0.446±0.100	0.271±0.067	3.566±0.821	2.612±0.182	2.617±0.172		
LSR	0.995±0.001	0.859±0.038	0.017±0.002	0.480±0.170	0.313±0.082	2.229±0.657	2.628±0.186	2.678±0.223		
LSR-D	0.995±0.001	0.858±0.037	0.025±0.005	0.475±0.159	0.374±0.093	2.226±0.628	2.535±0.167	2.675±0.224		
LSR-D-LR	0.995±0.001	0.867±0.036	0.025±0.004	0.442±0.159	0.372±0.098	1.777±0.507	2.539±0.168	2.703±0.229		
AODV	0.994±0.002	0.765±0.055	0.016±0.003	1.010±0.356	0.270±0.066	4.423±1.289	2.576±0.179	2.951±0.324		
DSR	0.940±0.027	0.683±0.059	0.041±0.047	4.760±1.073	0.220±0.095	0.410±0.140	2.677±0.185	3.625±0.308		
OLSR	0.887±0.040	0.798±0.034	0.012±0.001	0.883±0.311	1.937±0.220	0.713±0.069	2.456±0.175	2.478±0.161		

TABLE II

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	
	10	30	10	30	10	30	10	30	10	30
DSLRL	0.991±0.004	0.690±0.033	0.034±0.006	0.728±0.107	0.907±0.237	11.845±1.733	3.851±0.307	3.864±0.194		
LSR	0.990±0.004	0.737±0.039	0.042±0.007	0.751±0.129	1.192±0.342	8.213±1.473	3.814±0.314	3.941±0.237		
LSR-D	0.989±0.005	0.738±0.039	0.069±0.013	0.754±0.133	1.462±0.412	8.248±1.503	3.690±0.302	3.961±0.244		
LSR-D-LR	0.988±0.005	0.757±0.033	0.069±0.013	0.669±0.118	1.353±0.405	6.229±1.154	3.747±0.294	4.020±0.259		
AODV	0.988±0.004	0.608±0.051	0.036±0.009	1.455±0.385	0.897±0.236	18.298±13.069	3.744±0.293	4.751±0.434		
DSR	0.876±0.050	0.618±0.049	0.099±0.057	5.125±0.782	0.859±0.353	1.243±0.405	4.257±0.317	6.141±0.499		
OLSR	0.821±0.063	0.612±0.041	0.022±0.002	3.371±0.532	11.795±1.575	5.423±0.669	3.583±0.256	4.014±0.277		

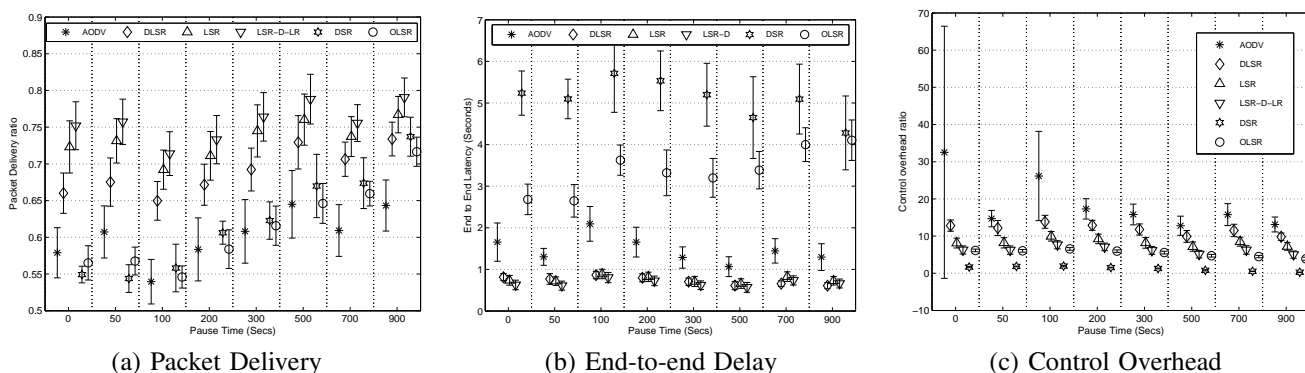


Fig. 7. Scenario with random sources and destinations (100 nodes, 30 flows, 120 pps)

the following problems: AODV suffers from excessive RREQ flooding. Note, however, that DSLR must also have RREPs sent by the destination, making it almost similar to AODV's case. We noticed that AODV was generating an excessive number of RREPs, and yet the number of RREPs received at the sources was far smaller. This could be caused by RREPs being forwarded by nodes using a reverse route entry, which is only valid for a limited time. Hence, with congestion, it is possible that these RREPs get delayed and are dropped. DSR suffers from stale caches and source-routes need not necessarily reflect the topology of the mobile network. OLSR suffers from temporary loops. The number of messages exchanged in OLSR is constant although the control overhead calculated depends on the number of flows in the network.

## VI. CONCLUSION

We have introduced the first routing framework for on-demand loop-free routing in MANETs based on the source sequence number that is used to identify RREQs originated uniquely. We presented three approaches within this framework (DSLRL, SLR, and LSR-D) that are robust to node failures, loss of information, and unreliable message deliv-

ery. They allow hop-by-hop routing of data packets while maintaining instantaneous loop-freeness of the routing tables without requiring time-stamps, source-routes, or any other synchronization techniques spanning single (i.e., packet filtering) or multiple hops. Performance results in 50 and 100-node networks with random traffic flows show that protocol instantiations of the proposed frameworks consistently deliver more data packets than DSR, AODV, and OLSR, while reducing control overhead and data packet latency.

## REFERENCES

- [1] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol," Request for Comments 3626, October 2003.
- [2] 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.
- [3] C. Perkins et al., "Ad hoc On-Demand Distance Vector (AODV) Routing," Request for Comments 3561, July 2003.
- [4] H. Rangarajan and J. J. Garcia-Luna-Aceves, "Efficient Use of Route Requests for Loop-free Routing in Ad hoc Networks," *IFIP Networking 2005 (LNCS)*, May 2005, Waterloo, Canada.