

Neighborhood Aware Source Routing

Marcelo Spohn
Computer Engineering Department
University of California
Santa Cruz, CA 95064
spohn@cse.ucsc.edu

J.J. Garcia-Luna-Aceves
Computer Engineering Department
University of California
Santa Cruz, CA 95064
jj@cse.ucsc.edu

ABSTRACT

A novel approach to source routing in ad hoc networks is introduced that takes advantage of maintaining information regarding the two-hop neighborhood of a node. The neighborhood aware source routing (NSR) protocol is presented based on this approach, and its performance is compared by simulation with the performance of the Dynamic Source Routing (DSR) protocol. The simulation analysis indicates that NSR requires much fewer control packets while delivering at least as many data packets as DSR.

Keywords

On-demand routing, source routing, ad hoc networks, wireless mobile networks, link-state information

1. INTRODUCTION

On-demand routing protocols have been shown to be very effective for ad hoc networks. The success of a caching algorithm for an on-demand routing protocol depends of the strategies used for the deletion of links from the routing cache [5]. DSR has been shown to incur less routing overhead when utilizing a cache data structure based on a graph representation of individual links (link cache), rather than based on complete paths. DSR removes failed links from the link cache when a ROUTE ERROR packet reports the failure of the link and other links are removed by aging. The lifetime of a link is estimated based on a node's perceived stability of both endpoint nodes of the link. In some scenarios, the use of link caches has been shown to produce overhead traffic as little as 50% of that incurred with path caches [5].

The results reported on DSR indicate that a routing protocol based on link-state information can make better routing decisions than one based on path information, because the freshness of routing information being processed can be determined by the timestamp or sequence number assigned by the head node of the links. On the other hand, in an

ad hoc network, it is very easy for a given node to learn about the neighbors of its neighbors. Based on these observations, we introduce a new approach to link-state routing in ad hoc networks based on on-demand source routing and knowledge of links that exist in the two-hop neighborhood of nodes. We call this approach the *neighborhood aware source routing* (NSR) protocol. In NSR, a node maintains a partial topology of the network consisting of the links to its immediate neighbors (1-hop neighbors), the links to its 2-hop neighbors, and the links in the requested paths to destinations that are more than two hops away. Links are removed from this partial topology graph by aging only, and the lifetime of a link is determined by the node from which the link starts (head node of the link), reflecting with a good degree of certainty the degree of mobility of the node.

Section 2 describes NSR in detail. Section 3 presents the performance comparison of NSR and DSR using a 50-node network and eleven simulation experiments with different numbers of sources and destinations. The simulation results indicate that NSR incurs far less communication overhead while delivering packets with the same or better delivery rates as DSR using path caches. Section 4 presents our conclusions.

2. NSR DESCRIPTION

2.1 Overview

To describe NSR, the topology of a network is modeled as a directed graph, where each node in the graph has a unique identifier and represents a router, and where the links connecting the nodes are described by some parameters. A link from node u to node v is denoted (u, v) , node u is referred as the head node of the link and node v is referred as the tail node of the link. In this study it is assumed that the links have unit cost.

Routers are assumed to operate correctly and information is assumed to be stored without errors. All events are processed one at a time within a finite time and in the order in which they are detected. Broadcast packets are transmitted unreliably and it is assumed that the link-level protocol can inform NSR when a packet cannot be sent over a particular link.

The *source route* contained in a data packet specifies the sequence of nodes to be traversed by the packet. A source route can be changed by the routers along the path to the destination and its maximum length is bounded.

NSR does not attempt to maintain routes from every node to every other node in the network. Routes are discovered on an on-demand basis and are maintained only as long as they are necessary.

Routers running NSR exchange link-state information and source routes for all known destinations are computed by running Dijkstra's shortest-path first on the partial topology information (*topology graph*) maintained by the router. If NSR has a route to the destination of a locally generated data packet, a source route is added to the header of the packet and it is forwarded to the next hop. Otherwise, NSR broadcast a *route request* (RREQ) to its neighbors asking for the link-state information needed to build a source route to the destination. If the neighbors do not have a path, a RREQ is broadcast to the entire network and only the destination of the data packet is allowed to send to the source of the RREQ a *route reply* (RREP) containing the complete path to the destination. A router forwarding a data packet needs to send a *route error* (RERR) packet to the source of the data only when it is not able to find an alternate path to a broken source route or when the failure of a link in the source route needs to be reported to the source of the data packet in order to erase outdated link-state information.

NSR has the property of being loop-free at all times because any change made to the path to be traversed by a data packet does not include nodes from the traversed path.

A link to a new neighbor is brought up when NSR receives any packet from the neighbor. The link to a neighbor is taken down either when the link-level protocol is unable to deliver unicast packets to the neighbor or when a timeout has elapsed from the last time the router received any traffic from the neighbor.

A node running NSR periodically broadcasts to its neighboring nodes a *hello* (HELLO) packet. HELLO packets have a dual purpose: they are used to notify the presence of the node to its neighbors and as a way of obtaining reasonable up-to-date information about the set of links two hops away from the node. By having such link-state information refreshed periodically the nodes forwarding a data packet may not need to notify the source of the packet when a repair is done to a broken source route.

2.2 Routing Information Maintained in NSR

A node i running NSR maintains the node's epoch E_i , the node's sequence number SN_i , the average lifetime of the node's links to neighbors L_i , the broadcast ID, the neighbor table, the topology graph, the shortest-path tree, the data queue, the RREQ history table, and the RERR history table.

The node's epoch is incremented when the node boots up before NSR starts its operation. The node's epoch is the only data that needs to be kept in non-volatile storage because it is used to maintain the integrity of routing information across node's resets (a separate section describes its role in NSR's operation).

NSR uses *sequence numbers* to validate link-state information. All the outgoing links of a node are identified by the

same monotonically increasing sequence number. A node increments its sequence number when it needs to send a control packet and a link was brought up or taken down since the last time a control packet was transmitted.

The lifetime of the node's links to neighbors is updated periodically by applying a decay factor to the current lifetime and averaging the time the links to neighbors are up.

The broadcast ID is used with the node's address to uniquely identify a RREQ packet. The broadcast ID has its value incremented when a new RREQ packet is created.

An entry in the neighbor table contains the address of a neighbor, the time the link to the neighbor was brought up, the *neighbor ID*, and a *delete flag*. When the link to a neighbor is taken down the *delete flag* is set to 1 to mark the entry as deleted. The neighbor table has 255 entries, and the *neighbor ID* consists of the number of the entry in the table. An entry marked as deleted is reused when the link for the neighbor listed in the entry is brought up. This guarantees that the *neighbor ID* is preserved for some time across link failures, which is useful when building source routes based on this IDs, as described later.

The topology graph is updated with the state of the links reported in both control packets and data packets. The parameters of each link (u, v) in the topology graph consists of the tuple $(sn, cost, lifetime, ageTime, nbrID)$, where sn is the link's sequence number, $cost$ is the cost of the link, $lifetime$ is the link's lifetime as reported by u , $ageTime$ is the age-out time of the link, and $nbrID$ is the *neighbor ID* assigned by node u to its neighbor v .

Link-state information is only deleted from the topology graph due to aging. The lifetime of a link is determined by the head node of the link. All the outgoing links of a node are considered to have the same lifetime, which is computed according to a function that estimates the average time a link to a neighbor is up.

The shortest-path tree is obtained by running Dijkstra's shortest-path on the topology graph when a control packet is received, when the state of an outgoing link changes, when a link ages-out, and periodically if the shortest-path tree has not been updated within a given time interval.

Data packets locally generated at the node waiting for a route to the destination are kept in the *data queue*. The packet at the tail of the queue is deleted when the queue is full and a new packet arrives to be enqueued. A packet is also deleted from the queue if a certain period of time has elapsed since it was inserted into the queue. A leaky bucket controls the rate with which data packets are dequeued for transmission in order to reduce the chances of congestion.

Each node maintains in the RREQ history table, for a specific length of time, a record with the source address and broadcast ID of each RREQ received. A node that receives a RREQ with a source address and broadcast ID already listed on the table does not forward the packet.

A node can send a RERR packet for a node src , through

neighbor nbr , as a consequence of processing a data packet sent by node src to destination dst , after detecting the failure of the link (u, v) , only if there is no entry in the RERR history table for the tuple (src, dst, u, v, nbr) . An entry is deleted from the table after a certain period of time has elapsed since it was created. The RERR history table is a mechanism used to avoid the generation of a storm of RERR packets reporting the failure of the same link.

2.3 Routing Information Exchanged by NSR

NSR can generate four types of control packets: RREQ, RREP, RERR, and HELLO. Routing information is also sent in the header of data packets. RREQ and HELLO packets are broadcast unreliably and RREP and RERR are transmitted reliably as unicast packets. The link to the next-hop along the path to be traversed by RREP and RERR packets is taken down if after several retransmissions the link-layer fails to deliver the packet to the intended neighbor. All the packets transmitted by NSR have a field that keeps track of the number of hops traversed by the packets. A packet is not forwarded if it has traveled $MAX_PATHLEN$ hops, however, the link-state information it carries is processed.

RREP, RERR, and data packets contain a source route. The source route consists of the sequence of nodes to be traversed by the packet. The identification of a node in a source route does not need to be the node's address, it can be the *neighbor ID* assigned by the node that precedes it in the source route. The neighbor ID is encoded in 1 byte, representing a significant reduction in the overhead added by the source route in a data packet when the addresses of the nodes are encoded in several bytes (e.g., 4 bytes in IPv4 [6], or 16 bytes in IPv6 [2]).

Every packet but HELLOs is updated with the state of the link over which it was received (the receiving node is the head of the link and the neighbor which sent the packet is the tail of the link). The source of a data packet also adds to the source route the sequence number of the links along the path to be traversed. The receiver node processing a source route updates its topology graph with the state of the links traversed by the packet.

The link state information (LSI) reported by NSR for a given link consists of the cost of the link (encoded in 1 byte), the sequence number of the head of the link (encoded in 2 bytes), and the lifetime of the link (encoded in 4 bits). LSIs reported in control packets have an extra field (encoded in 1 byte): the *neighbor ID* assigned by the head of the link to the tail node.

A node relaying RREQ, RREP, and RERR packets adds to the packet its *neighborhood link state* (NLS) which consists of the LSIs for outgoing links to neighbors. The NLS also contains the *partialLSI* flag which is set when the node reports a partial list of its outgoing links due to packet size constraints. All the links in an NLS have the same sequence number and lifetime. After processing the LSIs received in an NLS with the *partialLSI* flag not set, the node sets to infinity the cost of all the links in the topology graph having the same head node of the NLS but with a smaller sequence number, and the sequence number of these links

is updated with the sequence number reported in the NLS. Consequently, the node that advertises its NLS does not have to report the set of links that were removed from its NLS due to failures.

HELLO packets carry the node's NLS and are not relayed by the receiving node. The receiver of a HELLO processes the NLS reported in the packet in the same way NLSs are processed when received in RREQ packets.

A RREQ packet contains the source node's address, the destination's address, the maximum number of hops it can traverse, and a broadcast ID which is incremented each time the source node initiates a RREQ (the broadcast ID and the address of the source node form a unique identifier for the RREQ). Two kinds of RREQs are sent: non-propagating RREQs which can travel at most one hop, and propagating RREQs which can be relayed by up to $MAX_PATHLEN$ nodes.

A RERR packet is generated due to the failure of a link in the source route of a data packet. The RERR contains the source route received in the data packet, the head node of the failed link, the LSIs having as head node the head of the failed link, and the LSIs for the links in the alternate path to the destination (if any).

2.4 Operation of NSR

The NSR protocol is composed of four mechanisms that work together to allow the reliable computation of source routes on an on-demand basis:

- **Connectivity Management:** by which a node can learn the state of those links on the path to nodes two hops away. The cost of repairing a source route due to link failures can be significantly reduced by having available up-to-date state of such links.
- **Sequence Number Management:** by which the sequence number used in the validation of link-state information is updated such that its integrity is preserved across node-resets and network partitions. This mechanism also ensures that RREQs are uniquely identified across node-resets.
- **Route Discovery:** by which the source of a data packet obtains a source route to the destination when the node does not already know a route to it.
- **Route Maintenance:** by which any node relaying a data packet is able to detect and repair a source route that contains a broken link, and by which the source of a data packet is able to optimize source routes. A broken source route may be repaired multiple times until it reaches the destination without needing to notify the source of the data packet of such repairs.

2.4.1 Connectivity Management

This mechanism is responsible for determining the node's lifetime L_i and the state of the links to neighboring nodes. The link to a new neighbor is brought up when any packet is received from the neighbor. The link to a neighbor is

taken down if the node does not receive any packet from the neighbor for a given period of time.

HELLO packets are broadcast periodically and have their transmission rescheduled when RREQ packets are transmitted.

The node's lifetime L_i is recomputed periodically based on the average time the links to neighbors are up. The minimum lifetime L_{min} is assumed to be 30 seconds and the maximum lifetime L_{max} is assumed to be 1800 seconds. When reported in an LSI the node's lifetime is encoded in 4 bits after being rounded down to the nearest of one of the following values (in seconds): L_{min} , 45, 60, 75, 90, 105, 120, 150, 165, 180, 240, 360, 480, 900, and L_{max} .

2.4.2 Sequence Number Management

NSR works with the assumption that only the source of data packets be notified of the failure of a link in the path being traversed by a data packet. Given that the cost of a link may change over time without being noticed by some nodes in the network that have the link in their topology graphs, link-state information must be aged-out to prevent routers from keeping stale routes. A node can ascertain whether the link-state information reported by a neighbor is valid by comparing the sequence number in the LSI against the sequence number stored in the topology graph for the same link. The router considers the received LSI as valid if its sequence number is greater than the sequence number stored for the same link, or if there is no entry for the link in the topology graph.

The sequence number used in the validation of link-state information consists of two counters maintained by the head node i of the link: the node's epoch E_i and the node's sequence number SN_i . Both E_i and SN_i are encoded in one byte each and have a value in the range [1, 254]. It is assumed that SN_i wraps around when its value is either 127 or 254 and it is incremented.

The value of SN_i is incremented whenever the router needs to advertise changes to its NLS. It is assumed that the time interval between node resets is greater or equal to $L_{min} = 30$ seconds, and that SN_i should wrap around in at least $L_{max} = 1800$ seconds, i.e., any node other than the head i of a link with lifetime set to L_{max} will have deleted the link from its topology graph by aging before E_i and SN_i wrap around. If SN_i wraps around before L_{max} seconds have elapsed since the previous wrap around, then E_i is incremented and SN_i is set to 1.

The procedure used to determine whether a value X based on SN_i or E_i is greater than a value Y is shown in Figure 1. When SN_i gets incremented to a value sn greater than 127 then all the nodes in the network have already aged-out all the links reported by i with SN_i in the range [1, $sn - 127$]. Likewise, when SN_i gets incremented to a value sn smaller than 128 then all the routers have already aged-out all the links reported by i with SN_i in the range [$128, sn + 127$].

From the perspective of any node $x \neq i$ in the network, the combination of E_i and SN_i is seen as an unbounded counter because the values of E_i and SN_i have a lifetime.

```

result ← FALSE;
if ( X ≤ 127 )
  A ← X; B ← Y;
else
  A ← Y; B ← X;

if ( B ≤ 127 )
  if ( A > B )
    result ← TRUE;
else if ( |A - B| > 127 )
  result ← TRUE;

if ( X = A and result = TRUE )
  X is greater than Y;
else if ( Y = A and X ≠ Y and result = FALSE )
  X is greater than Y;

```

Figure 1: Comparing values derived from E_i or SN_i

The sequence number of a received LSI for the link (u, v) is greater than the sequence number stored in the topology graph for the same link if the E_u component of the LSI's sequence number is greater than the respective E_u stored in the topology graph, or if the epochs are the same but the SN_u component of the LSI's sequence number is greater than the respective SN_u component in the topology graph.

The broadcast ID set by node i in a RREQ packet also consists of two counters: the node's epoch E_i and a 4-byte sequence number B_i . The source of a RREQ increments B_i before creating the RREQ for transmission. By having E_i as part of the broadcast ID, RREQs are uniquely identified across node resets.

2.4.3 Route Discovery

When NSR receives a data packet from an upper-layer and the router has a source route to the destination, the source route is inserted into the packet's header and the packet is forwarded to the next hop towards the destination. Otherwise, NSR inserts the data packet into the *data queue* and initiates the *route discovery* process, if there is none already in progress, for the data packet's destination by broadcasting a non-propagating RREQ. By sending non-propagating RREQs, NSR prevents unnecessary flooding when some neighbor has a source route to the required destination. If none of the neighbors send a RREP within a timeout period, a propagating RREQ is transmitted. Each time a propagating RREQ is transmitted the timeout period is doubled until a pre-defined number of attempts have been made, after which it is kept constant. After a pre-defined number of RREQs have been transmitted for a given destination, the route discovery process is restarted by sending a non-propagating RREQ if the *data queue* holds a packet for the destination.

When a node receives a RREQ, it processes all the LSIs in the packet and then checks whether it has seen it before by comparing the source address and the broadcast ID from the RREQ against the entries in the RREQ history table. The RREQ is discarded if the node has already seen it before, otherwise the node is said to have received a *valid RREQ*, and an entry is added to the RREQ history table with the values of the RREQ's source address and broadcast ID. Non-propagating RREQs are always considered valid RREQs.

The receiver node of a non-propagating RREQ sends a RREP if it has a source route to the destination of the RREQ. Since

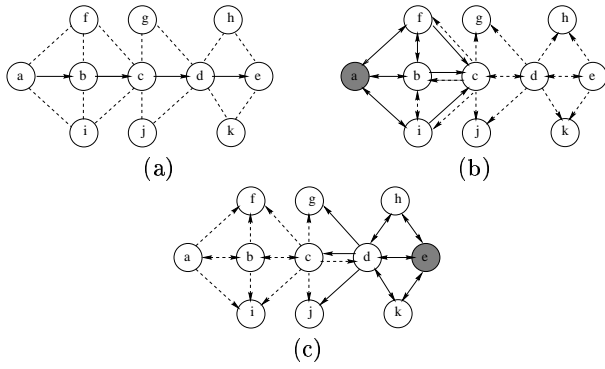


Figure 2: Link-state information learned from RREQ and RREP

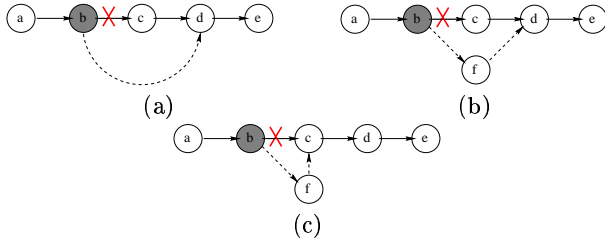


Figure 3: Repairs that can be applied to a source route in a RREP

a RREP to a non-propagating RREQ is not generated by the destination of the RREQ, the lifetime of the LSIs reported in the RREP must correspond to the time left for being aged-out from the node's topology graph.

If the node processing a valid RREQ is the destination of the RREQ then it sends a RREP back to the source of the RREQ. The source route contained in the RREP consists of the reversed path traversed by the RREQ packet. A node other than the destination of a valid RREQ adds its NLS into the packet before broadcasting it. Likewise, a node other than the destination of a RREP adds its NLS into the packet before forwarding it. The link-state information learned from RREQs and RREPs increases the chances of a node finding a source route in the topology graph and, consequently, increases the likelihood of replying to non-propagating RREQs. As an example, consider the network topology shown in Figure 2(a), where solid lines indicate the path traversed by the first RREQ packet received by destination node e from the source node a . The dashed lines in Figure 2(b) represent the links learned from the RREP received by node a from destination e . The dashed lines in Figure 2(c) represent the links learned from the RREQ received by node e from node a . The solid lines in Figures 2(b) and 2(c) correspond to those links learned from HELLO packets.

A node forwarding a RREP packet may change its source route if the link to the next hop has failed. The broken source route can be repaired if the node is able to find an alternate path having at most 2-hops to any of the nodes in the path to be traversed by the RREP packet. The or-

der with which the nodes from the broken source route are visited when seeking an alternate path is from the tail node towards the node that corresponds to the tail of the failed link. Figure 3 shows the types of repairs that can be applied to the source route (shown in solid lines) in a RREP: in Figure 3(a) the failure of the link (b, c) causes node b to replace links (b, c) and (c, d) by (b, d) , in Figure 3(b) the failure of the link (b, c) causes node b to replace links (b, c) and (c, d) by (b, f) and (f, d) , and in Figure 3(c) the failure of the link (b, c) causes node b to replace link (b, c) by (b, f) and (f, c) . An extra-hop can be added to a broken source route only if the length of the new path does not exceed MAX_PATHLEN hops.

2.4.4 Route Maintenance

A node forwarding a data packet attempts to repair the source route when either the link to the next hop or the link headed by the next hop in the path to be traversed has failed. The repair consists in finding an alternate path to the destination of the data packet, and may involve the transmission of a RERR packet to the source of the data packet.

The repair made by a forwarding node to the source route of a data packet does not trigger the transmission of a RERR packet if the following rules are satisfied:

- **Rule-1:** the node processing the packet is listed in the original source route received in the data packet.
- **Rule-2:** one of the nodes not yet visited by the data packet but listed in the original source route is at most two hops away from the router itself in the repaired source route.

The path traversed by a RERR packet consists of the reversed path traversed by the data packet, having as destination the source of the data packet that triggered its transmission. Rule-1 and Rule-2 guarantee that the source of the data packet is notified of all the link failures present in its source route. When the RERR reaches its destination, the source of the data packet updates its topology graph and recomputes its shortest-path tree.

Figure 4 illustrates the cases where repairs can be applied to the source route in a data packet without triggering the generation of RERRs. The links shown as solid lines in Figure 4 correspond to the source route carried by data packets originated at node a with destination f , and the dashed lines represent the links added to the new source route repaired by the nodes indicated with filled circles.

Figure 4(a) illustrates the fact that a node considers a source route as broken if any of the links in the next two hops following the node processing the packet has failed. In this particular case, node b receives a HELLO packet from node c reporting the failure of link (c, d) before b receives a data packet to be forwarded. The node forwarding a data packet may not have in its topology graph the link that is one hop away in the source route. In order to prevent the node from dropping the data packet, NSR allows the packet to be forwarded if the sequence number in the source route for

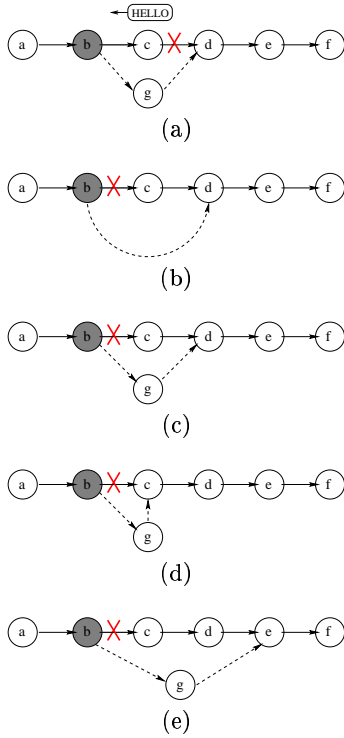


Figure 4: Type of repairs that can be applied to a source route in a DATA packet

the missed link is greater than the sequence number of any link reported by the neighbor.

The failure of the link (b, c) shown in Figure 4(c) makes the links (b, c) and (c, d) in the source route be replaced by the links (b, g) and (g, d) . The failure of the link (b, c) shown in Figure 4(d) causes the link (b, c) be replaced by the link (b, g) and the source route be extended in one hop by adding link (g, c) to it. The failure of the link (b, c) shown in Figure 4(e) causes the link (b, c) be replaced by link (b, g) and the source route be shortened in one hop by replacing the links (c, d) and (d, e) by link (g, e) .

If node g shown in Figure 4 receives a data packet with the source route repaired by node b and it detects the source route is broken, a RERR packet needs to be transmitted (even if g has an alternate path) since Rule-1 is not satisfied when the node attempts repairing the route.

Figure 5 illustrates the cases that trigger the transmission of a RERR packet when a source route in a data packet is detected to be broken. The links shown as solid lines in Figure 5 correspond to the source route carried by data packets originated at node a with destination f , and the dashed lines represent the links added to the new source route repaired by the nodes indicated with filled circles.

The generation of RERR packets reporting the failure of the same link are spaced by some time interval if the source route being processed was generated by the same source node, and the data packet is for the same destination, and the data

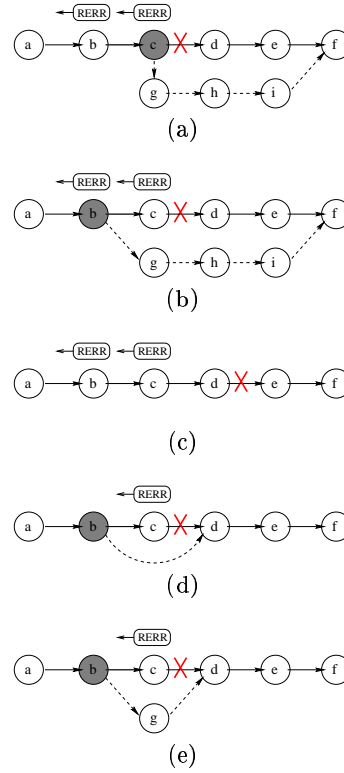


Figure 5: Broken source-routes leading to transmission of RERR packets

packet was received from the same neighbor that caused the transmission of the previous RERR packet. This mechanism prevents the generation of a RERR packet for every data packet in transit carrying the same source route. After transmitting a RERR packet the node updates its RERR history table by adding an entry with information about the RERR packet.

The failure of the link (c, d) shown in Figure 5(a) causes node c to transmit a RERR packet to the source of a data packet received for forwarding. The RERR packet reports the new source route to destination f , which consists of the links (c, g) , (g, h) , (h, i) , and (i, f) instead of the links (c, d) , (d, e) , and (e, f) . The data packet being processed has its source route updated accordingly and is forwarded to node g . When node b receives the RERR packet its topology graph is updated with the link-state information reported in the packet, its shortest-path tree is recomputed, its NLS is added to the packet, and the packet is then forwarded to a .

Node b shown in Figure 5(b) adds to the RERR packet received from node c an alternate path to f before forwarding the packet to a . NSR allows the node forwarding a RERR to add an alternate path to the RERR packet only if it is a neighbor of the head node of the failed link that triggered the generation of the RERR packet.

Node a shown in Figure 5(c) receives a RERR packet not reporting an alternate path to the destination.

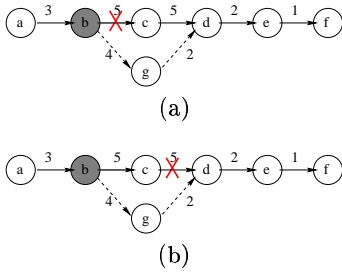


Figure 6: Using *neighbor IDs* in source routes

Node *b* shown in Figures 5(d) and 5(e) does not forward the RERR packet because it has an alternate path to the destination. The next data packet it receives from *a* has the source route repaired with the alternate path.

2.5 Using Neighbor IDs in Source Routes

The source route given in a data packet can be formed by the sequence of neighbor IDs mapped to each link along the path to be traversed by the packet, instead of being formed by node's addresses. Such approach makes the source route very compact and allows more data be carried in each packet. As an example consider the scenarios depicted in Figure 6. The numbers beside links shown as solid lines in Figure 6 correspond to the source route carried by data packets originated at node *a* with destination *f*, and the dashed lines represent the links added to the new source route repaired by the nodes indicated with filled circles. The number beside a link is the neighbor ID given by the head of the link to the tail node. The *neighbor table* contains the mapping between neighbor ID and the address of a neighbor. The entry for the node with neighbor ID 5 is not deleted from the neighbor table of node *b* when the link (*b, c*) fails (Figure 6(a)), allowing *b* to get the address of *c* and repair the source route accordingly. Because links are deleted from the topology graph only due to aging, node *b* in Figure 6(b) is able to identify the tail of the failed link (*c, d*) by looking for all the links in the topology graph having node *c* as the head of the link and a neighbor ID 5.

The source route received by node *f* in the data packet sourced at *a* contains the state of all the links in the reversed path traversed by the data packet. With the failure of either link (*b, c*) or link (*c, d*) the source route received by node *f* contains LSIs for the links (*d, g*) and (*g, b*). Node *f* cannot update the topology graph with the state of (*d, g*) and (*g, b*) if the links are not part of *f*'s topology graph and the source route lists neighbor IDs instead of node's addresses. The likelihood of finding alternate paths to destinations increases with up-to-date link-state information carried in data packets, especially when the data flows are bidirectional. For this reason, the source of data packets are required to periodically use addresses instead of neighbor IDs in the source routes.

3. PERFORMANCE EVALUATION

We run a number of simulation experiments to compare the average performance of NSR with respect to DSR. Both NSR and DSR use the services of a medium access (MAC) protocol based on an RTS-CTS-DATA-ACK packet exchange

for unicast traffic (similar to the IEEE 802.11 standard [1]). The promiscuous mode of operation is disabled on DSR because the MAC protocol uses multiple channels to transmit data. (Both NSR and DSR might benefit from having the node's network interface running in promiscuous mode.) The physical layer is modeled as a frequency hopping spread spectrum radio with a link bandwidth of 1 Mbit/sec, accurately simulating the physical aspects of a wireless multi-hop network.

3.1 Mobility Pattern

The simulation experiments use 50 nodes moving over a rectangular flat space of 5Km x 7Km and initially randomly distributed at a density of 1.5 nodes per square kilometer. Nodes move in the simulation according to the *random waypoint* model [3]. In this model, each node begins the simulation by remaining stationary for *pause time* seconds, it then selects a random destination and moves to that destination at a speed of 20 meters per second for a period of time uniformly distributed between 5 and 11 seconds. Upon reaching the destination, the node pauses again for *pause time* seconds, selects another destination, and proceeds there as previously described, repeating this behavior for the duration of the simulation.

The simulation experiments are run for the pause times of 0, 15, 30, 45, 60, 90, and 900 seconds, and the total simulated time in all the experiments is 900 seconds. A pause time of 0 seconds correspond to the continuous motion of the nodes, and in a pause time of 900 seconds the nodes are stationary.

3.2 Data Traffic Model

The overall goal of the simulation experiments is to measure the ability of the routing protocols to react to changes in the network topology while delivering data packets to their destinations. The aggregate traffic load generated by all the flows in a simulated network consists of 32 packets/second and the size of a data packet is 64 bytes. The data traffic load was kept small to ensure that congestion of links is due only to heavy control traffic.

We applied to the simulated network three different communication patterns: a pattern of N-sources and N-destinations (Nsrc-Ndst), a pattern of N-sources and 8-destinations (Nsrc-8dst), and a pattern of N-sources and 1-destination (Nsrc-1dst). For each communication pattern we run a number of simulation experiments with different number of data flows. The data flows consist of continuous bit rate traffic, all the flows in a simulation experiment generate traffic at the same data rate, and each node is the source of no more than one data flow. We run four simulation experiments with 8, 16, 32, and 50 sources for both the Nsrc-Ndst and the Nsrc-1dst patterns, and 3 simulation experiments with 16, 32, and 50 sources for the Nsrc-8dst pattern. The data flows are started at times uniformly distributed between 10 and 120 seconds of simulated time.

3.3 Protocol Configuration

The values for the constants controlling DSR operation during the simulations are those present in the *ns-2* implementation of DSR [4]. The values for the constants controlling NSR operation are listed below:

- Time between successive transmissions of RREQs for the same destination is 0.5 seconds. This time is doubled with each transmission and is kept constant to 10 seconds with the transmission of the sixth RREQ.
- The minimum lifetime of an LSI is 30 seconds, and the maximum lifetime is 1800 seconds.
- The average time interval between the transmission of HELLO packets is 59 seconds with a standard deviation of 1 second.
- The maximum number of entries in the *data queue*, RREQ history table, and RERR history table is 50, 200, and 200, respectively.
- The lifetime of an entry in the *data queue*, RREQ history table, and RERR history table is 30, 30, and 5 seconds, respectively.
- Data packets with a source route are removed from the *data queue* for transmission spaced from each other by 50 milliseconds.
- The maximum number of nodes (MAX_PATHLEN) traversed by any packet is 10.

3.4 Simulation Results

Figures 7, 8, and 9 summarize the comparative performance of NSR and DSR in all the simulation experiments. The coordinates having a value of 100 in the x-axis of Figures 10 and 11 correspond to the results obtained for networks with stationary nodes.

The number of control packets generated by NSR falls in the range [950, 6490], while for DSR is in the range [260, 83010]. From Figure 7 we can see that DSR is able to generate less than 6490 packets in only 24% of the experiments. The benefits brought by the ability of NSR repairing source routes without needing to inform the source of the data packets is noticeable when N sources send data packets to the same destination (Nsrc-1dst traffic pattern): most of the ROUTE REPLIES generated by nodes running DSR carry stale routing information, which leads to the transmission of more ROUTE REQUESTS. And, as shown in Figures 10(d), 10(e), and 10(f), this behavior is independent of the *pause time* (except when the nodes are stationary and the number of data flows is low). NSR may generate more control packets than DSR in some scenarios because HELLO packets are transmitted periodically. These are the cases depicted in Figures 10(a), 10(b) and 10(d) when the nodes do not move.

Figure 10 shows that the number of control packets transmitted by nodes running NSR deviates very little from the average among different *pause times* while DSR presents a large deviation from the average, especially when comparing the experiments with high node mobility against the experiments with stationary nodes. We can also see that the type of workload introduced in the network makes DSR to behave in an unpredictable manner while NSR is very little affected. In the simulation experiments with Nsrc-Ndst traffic pattern DSR generates up to 9.9 times more control packets than NSR, with Nsrc-1dst traffic pattern DSR generates up to 36.6 times more control packets, and with Nsrc-8dst traffic pattern DSR generates up to 15.7 times more control packets than NSR.

We observe from Figure 8(a) that, on average, NSR is able to deliver to the destinations more data packets than DSR: in 79% of the simulation experiments NSR delivered more than 50% of the data packets generated while DSR delivered more than 50% in 59% of the simulation experiments. The lack of a source route to the destination in the experiments with Nsrc-1dst traffic pattern caused DSR to discard a high number of packets awaiting for a route (Figure 8(c)). Figure 11 gives the average performance of NSR and DSR in terms of the percentage of data packets delivered to the destinations for a given *pause time* and traffic pattern. We see that the highest number of packets are delivered as the nodes become less mobile. This behavior is expected because all the packets enqueued for transmission at the link-layer are dropped after link failures, and link failures occur less frequently when the nodes in the network become more stationary.

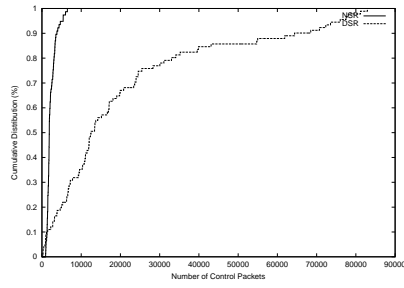
The end-to-end delay experienced by data packets (Figure 9) routed by NSR is similar to the end-to-end delay experienced by data packets routed by DSR. The curve shown in Figure 9(b) shows data packets routed by DSR having a smaller end-to-end delay than those routed by NSR. This is because the percentage of data packets delivered by DSR for the Nsrc-1dst traffic pattern was very low compared to NSR. We observe that most of the data packets routed by NSR traversed from 2 to 3 hops, while most of the packets routed by DSR traversed from 2 to 4 hops.

4. CONCLUSIONS

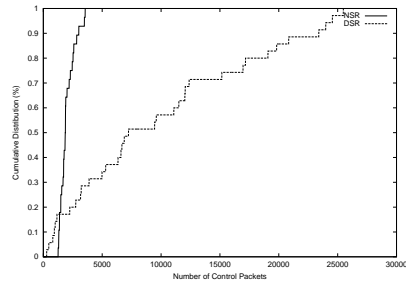
We have presented the neighborhood aware source routing protocol (NSR), which we derived from the performance improvements observed in DSR when link caches were used, and the ease with which nodes can inform their neighbors of their own neighbors. The key feature of NSR is that nodes reduce the effort required to fix source routes due to node mobility by using alternate links available in their two-hop neighborhood. Simulations demonstrate the advantages derived from the availability of such alternate paths. Future work focuses on comparing by simulation the performance of NSR with DSR when link caches are used.

5. REFERENCES

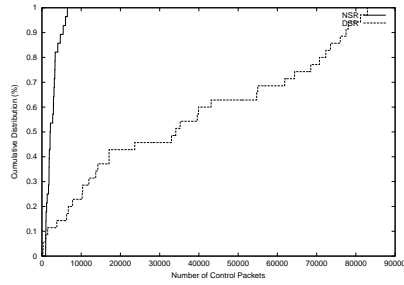
- [1] I. C. S. L. M. S. Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11*, 1997.
- [2] S. Deering and R. Hinden. Internet Protocol Version 6 (IPv6) Specification. *RFC*, 2460, 1998.
- [3] J. B. et al. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. *Proc. ACM MOBICOM*, October 1998.
- [4] K. Fall and K. Varadhan. ns Notes and Documentation. *The VINT Project, UC Berkeley, LBL, USC/ISI and Xerox PARC*, <http://www.isi.edu/nsnam/ns>, 1999.
- [5] Y.-C. Hu and D. Johnson. Caching Strategies in On-Demand Routing Protocols for Wireless Ad Hoc Networks. *Proc. ACM MOBICOM*, 2000.
- [6] J. Postel. Internet Protocol. *RFC*, 791, 1981.



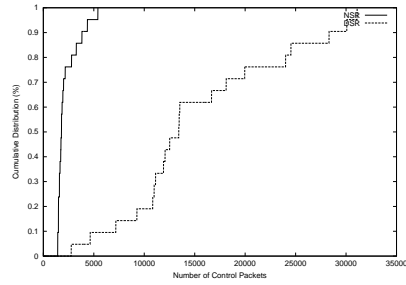
(a) All the traffic patterns



(b) Nsrc-Ndst pattern

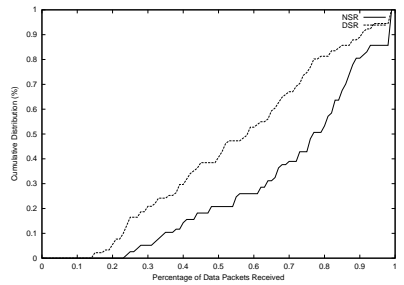


(c) Nsrc-1dst pattern

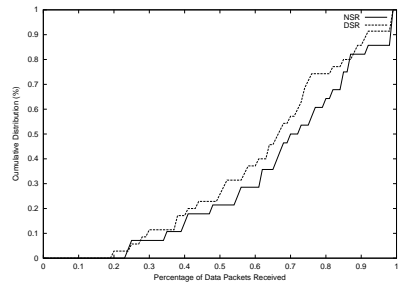


(d) Nsrc-8dst pattern

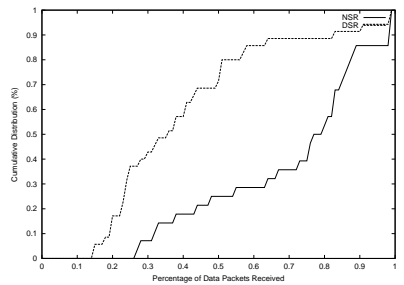
Figure 7: Cumulative distribution function for the number of control packets generated



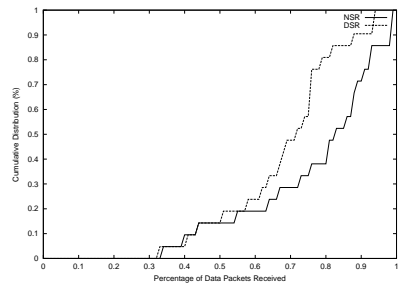
(a) All the traffic patterns



(b) Nsrc-Ndst pattern

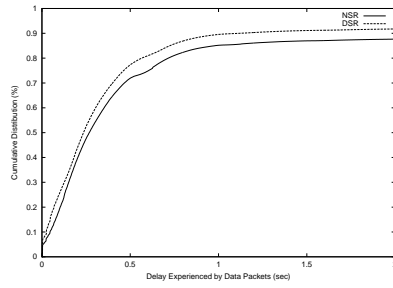


(c) Nsrc-1dst pattern

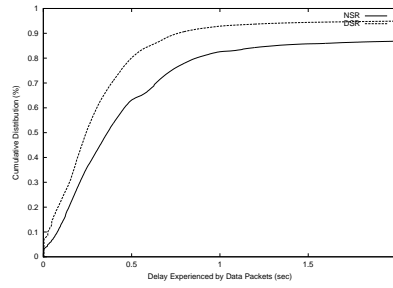


(d) Nsrc-8dst pattern

Figure 8: The cumulative distribution function for the percentage of data packets received

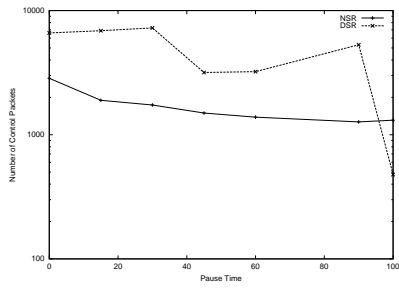


(a) All the traffic patterns

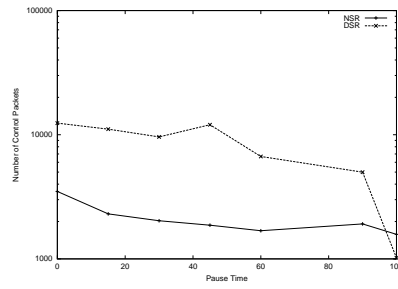


(b) Nsrc-1dst pattern

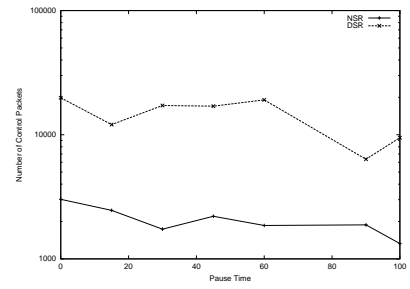
Figure 9: The cumulative distribution function for the delay experienced by data packets



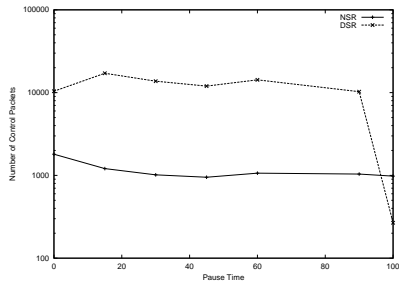
(a) 8 sources and 8 destinations



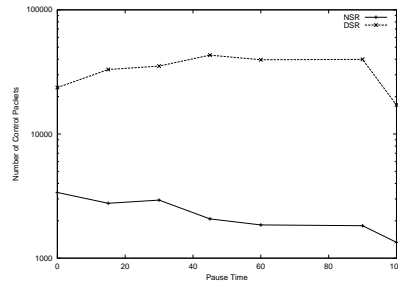
(b) 16 sources and 16 destinations



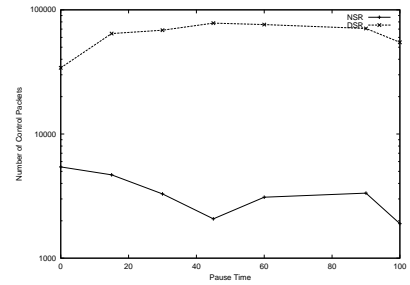
(c) 32 sources and 32 destinations



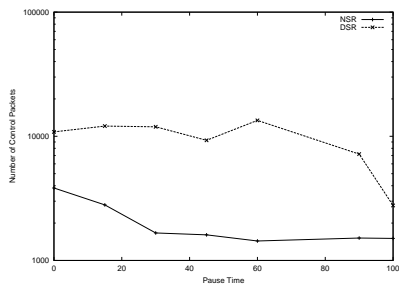
(d) 8 sources and 1 destinations



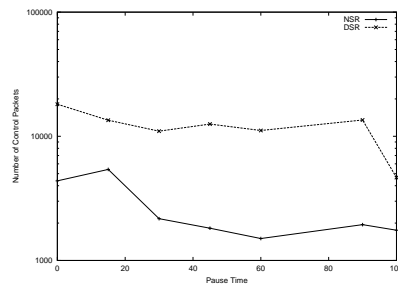
(e) 16 sources and 1 destinations



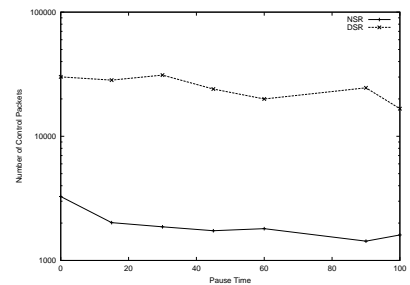
(f) 32 sources and 1 destinations



(g) 16 sources and 8 destinations

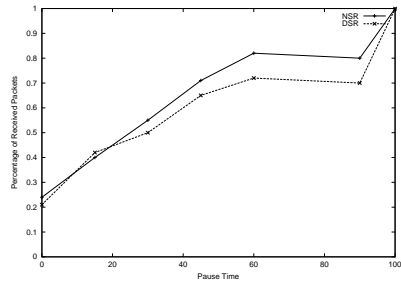


(h) 32 sources and 8 destinations

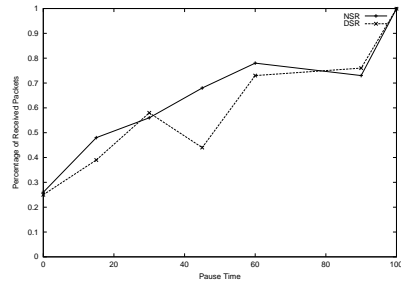


(i) 50 sources and 8 destinations

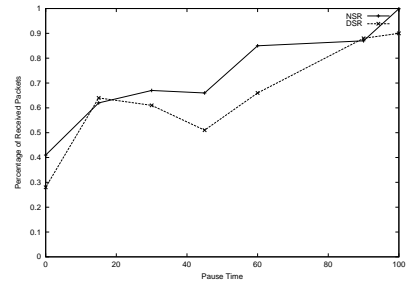
Figure 10: Number of control packets transmitted according to node mobility



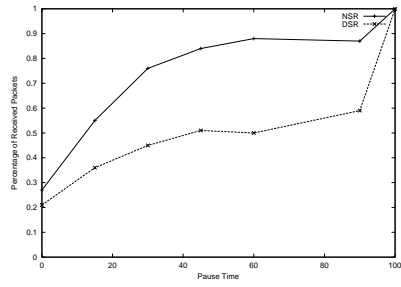
(a) 8 sources and 8 destinations



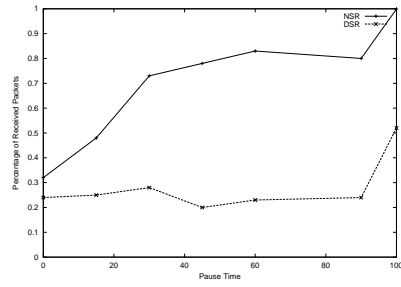
(b) 16 sources and 16 destinations



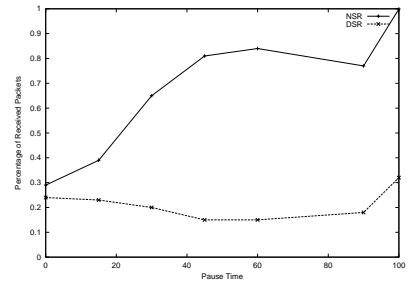
(c) 32 sources and 32 destinations



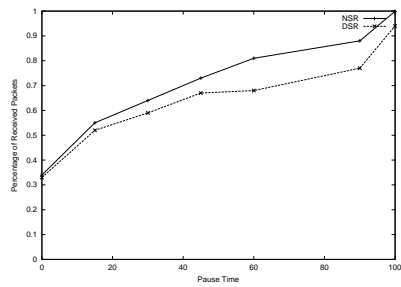
(d) 8 sources and 1 destination



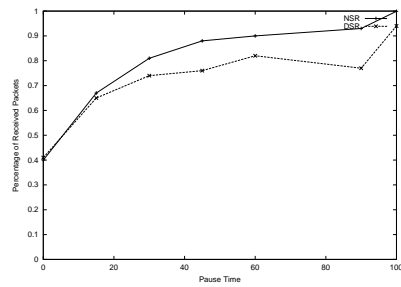
(e) 16 sources and 1 destination



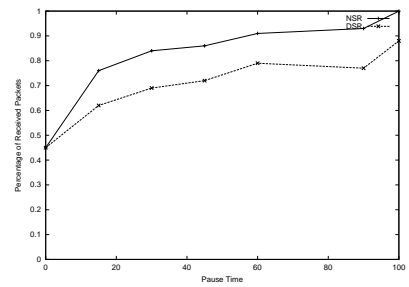
(f) 32 sources and 1 destination



(g) 16 sources and 8 destinations



(h) 32 sources and 8 destinations



(i) 50 sources and 8 destinations

Figure 11: Percentage of data packets received according to node mobility