5.1 Introduction

A Mobile Ad hoc NETwork (MANET) is a multihop wireless network consisting of a number of mobile devices forming a temporary network. No established wired infrastructure or centralized network administration exists to support the communication in a MANET. A MANET differs from a wired network in many ways. Some of these are:

- Most users may neither wish nor be in a position to perform any administrative services needed to set up and maintain a network.
- Topology of interconnections is dynamic due to mobility of nodes.
- Communication is unreliable in wireless network and susceptible to vagaries of weather and many other form of interferences.
- Wired networks have high data bandwidth and more or less uniform link capacity whereas wireless networks have low bandwidth and widely varying link capacities.
- Wireless networks suffer from energy constraints, as mobile nodes operate on battery; and for portability reasons battery capacity is usually low.
- Wired networks are explicitly configured to have only one or a small number of router(s) connecting any networks, whereas no explicit links exist in an ad hoc network.
- Only limited physical security is possible in case of wireless network as communication is possible only through broadcast.

As a consequence of these and many other differences, communication and computing on a network of mobile hosts exhibit following notable characteristics;

1. all communication realizable by broadcast transmission.
2. the redundant paths in a wireless environment unnecessarily increase the size of routing updates that must be sent over the network, and
3. increases the CPU overhead required to process each update and to compute new routes.

Each mobile host has a fixed range within which it can broadcast messages. Any mobile host which appears inside the range of a transmitting host can listen/capture the messages sent by the latter. A mobile host enlists the support of others which comes directly in the wireless range of the former to forward messages to the desired destinations that is not directly reachable by the source. For example, consider the figure 5.1 where A, B and C represent three mobile hosts. The wireless ranges of these hosts are

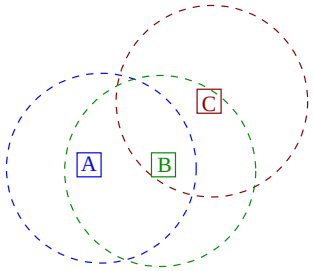


Figure 5.1: Ad hoc network

indicated by the respective circles having centers at A, B and C. A and B are within the wireless range of each other, so are B and C. But C can not directly receive any transmission from A as it is not in the range of A. If A were to send a message to C then B must cooperate to forward the same to C. Thus, the design of an efficient routing protocol in MANET poses a number of difficulties depending on the degree of severities of the constraints mentioned above. This explains why the area offers such as a rich source of topics for research.

A number of routing schemes have been proposed [14, 3, 12, 7, 15, 11, 8, 16] for mobile ad hoc networks. Most of these algorithms based on existing Internet routing algorithms. The performance evaluations of the routing algorithms can be based on a set of common parameters, namely,

- Distributiveness of the execution of the algorithm.
- Ability to find loop-free paths between a source and a destination.
- Ability to find cheaper routes between a source and a destination.
- Ability to restrict flooding the network with broadcast transmission during route discovery.

- Ability to quickly establish a route between a source and destination.
- Ability to avoid congestion at nodes by providing alternative routes between a source and a destination.
- Ability to maintain or repair a route between a source and a destination quickly, by localizing the route maintenance, when the network topology undergoes a change.
- Ability to provide quality of service (QoS).

An exhaustive study of the routing algorithms could be a book by itself. The interested readers may refer to an excellent collection of selected articles on ad hoc network [13]. Only a few important and widely cited routing schemes are the subject of our discussion here.

5.2 Classification of Routing Protocols

The routing in any network can be viewed abstractly as finding and maintaining the shortest-path between two communicating nodes. Each node maintains a preferred neighbour, which is the next hop to a destination. Each data packet will contain the address of the destination in its header. An intermediate node forwards a data packet to the next hop consulting the locally maintained table known as *route table*. The routing protocols differ with respect to the manner in which the tables are constructed and maintained. In a recent review Royer and Toh [17] have provided a classification of the routing protocols for ad hoc network. As shown in figure 5.2, the routing schemes can be classified into two broad classes, namely,

- Table driven or proactive, and
- Source initiated on-demand driven or reactive

The solid links in the above figure represent the direct descendants. For example, Fisheye State Routing (FSR) protocol is a direct descendant of Global State Routing (GSR) protocol. Whereas the broken links represent the logical descendants. Clusterhead Gateway Switching (CGSR) protocol is a logical descendant of Destination Sequence Distance Vector (DSDV) protocol. In this chapter we focus our discussion on the protocols labeled in blue. That leaves out only Signal Stability Routing (SSR) [18] and LMR

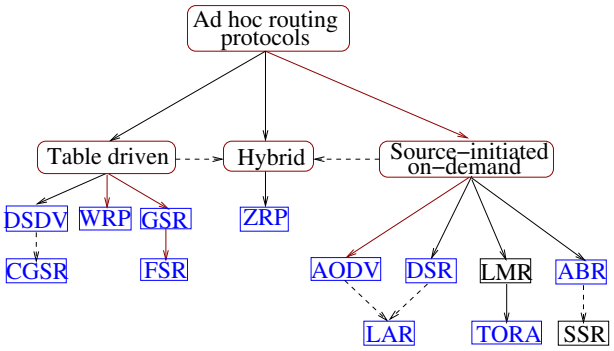


Figure 5.2: Classifying Routing Algorithms

(Lightweight Mobile Routing) [6]. The interested reader may refer to the relevant literature or the review article by Royer and Toh [17].

All MANET routing protocols are based on existing Internet protocols. There three approaches in design of Internet routing protocols:

- Distance vector,
- Link state, and
- Link reversal.

5.2.1 Distance Vector Routing

The abstract model of an underlying ad hoc network is a graph. Every node v maintains the table of distances d^x_{vw} for each destination node x, where $w \in L_v$ (adjacency list of v). In order to send a message from a source v to a destination x the next hop is selected among the neighbours of v which returns the minimum value of the metric d^x_{vw}. At an intermediate hop, the same rule is applied to forward the message to another hop. The process is repeated till the destination is reached. The message is, thus, forwarded from a source to a destination by a sequence of hops via the shortest known path between the two nodes.

A major problem which may arise in formation of route using a distance vector routing scheme is referred to as *count-to-infinity*. Figure 5.3 provides an illustration of the problem. Initially, A knows distance to C via B to be 2 units. Suppose the link B to C breaks (e.g., timer expires). But before

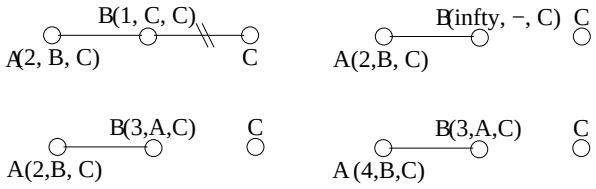


Figure 5.3: Count to infinity problem

B can advertise its own routing table, it receives update from A indicating there is a path of 3 units to C available through A. B updates the distance to C as 3 not knowing the fact that route via A includes itself. B then sends the update to A. After receiving the update from B, A updates distance to C as 4. The cycle of update exchanges keeps continuing resulting in count-to-infinity problem.

5.2.2 Link State Routing

Link state routing (also known as shortest path first or SPF routing) can be viewed as a centralized version of shortest path computation at each node. Each node maintains some (at least partial) view of entire topology together with the cost of each link. The two basic steps of link state routing are *flood* and *update*.

Flood To preserve a consistent of view of the network, each node periodically broadcasts (floods the network) the link cost of its outgoing links to all other nodes.

Update All nodes update (recompute) the shortest paths to each destination based on the new information reaching them. For computing shortest path, a link state protocol typically uses Dijkstra's algorithm.

Each node forwards data packets to the next hop based on the current best path to destination. Some of the important performance characteristics of link state routing schemes can be summarized as follows:

- They converge more quickly as the network undergoes changes.
- Require more bandwidth to broadcast route updates.

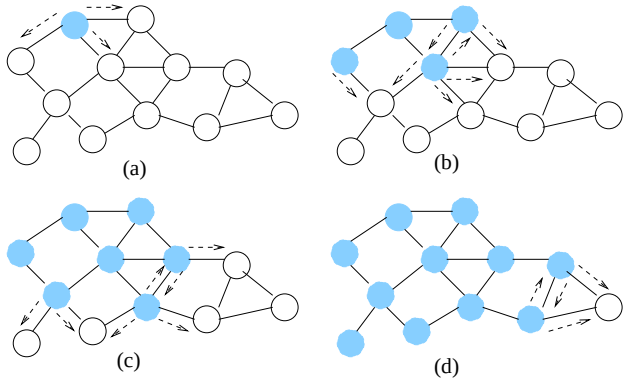


Figure 5.4: Propagation of LSP

- Require more CPU time for computing the shortest path.
- The information at a node may be inconsistent due to long propagation delays, partitioned network, etc.
- May lead to formation of loops.

Each link update is provided with a unique sequence number and comes associated with the initiator's identity. The initiator increments the sequence number each time it sends a new update. Thus a node can distinguish an earlier update from a recent one. It can drop an incoming update from an initiator if it has forwarded the same earlier (by comparing the sequence numbers of the two updates). The update is forwarded to all neighbours except for the neighbour from where the update was received. Figure 5.4 illustrates how link state update packet (LSP) is propagated over a network. The nodes in light blue are those which have received LSP. The propagation of LSP from a node in the network is in the direction indicated by arrows going out from those nodes in the figures. As clear from the figure the propagation will require $O(d)$, where d is the diameter of the network.

5.2.3 Link Reversal Routing

Link reversal routing (LRR) algorithms do not maintain full topology table which the proactive routing schemes such as DSDV do. Neither do they initiate route discovery each time a source needs a path to a destination

which reactive schemes like DSR or AODV. The differences in LRR approach with other routing protocols can be summarized as follows.

- It does not have to maintain global information like table driven algorithms such as DSDV.
- It does not have to go through a route discovery for each source destination pair unlike reactive routing protocols such as AODV or DSR.
- It has low overheads in terms of volume of control packets injected to the network and low latency in finding paths.

LRR algorithms maintain a directed acyclic graph (DAG) for each destination D with D as the only sink in the DAG. Other nodes in this DAG have both incoming and outgoing links. Without the loss of any generality, we can focus only on a single destination and the DAG associated with it. However, a node may become a destination for a message from other nodes at any time. Since a DAG has no cycle, any message from a node will not loop and will eventually reach the destination (sink), though it may have to traverse a longest path (critical path). The key issue in a LRR algorithm is to maintain the DAG correctly. This is done differently in different LRR based protocols. To fully appreciate the problem, we first introduce some definitions and investigate the problems in finding paths in a DAG.

Definition 1 *A DAG is a destination oriented DAG, if for each node N there exists a directed path from N to the destination D.*

We can formally prove the following result.

Lemma 1 *A DAG is not destination oriented if and only if there exists a node other than the destination that has no outgoing link.*

Proof: If a node $N \neq D$ has no outgoing link then no part of the graph can be reachable by a directed path from N. Therefore, *if part* of the lemma is obvious.

The backward part of the proof proceeds as follows. Let there be a a node N other than D which does not have any outgoing link. Since G is a DAG, there can not be a loop involving N. So, once a directed path from any node in G reaches N, it will fail to reach D subsequently. ∎

A destination orient DAG can be maintained by ensuring that:

1. Every node other than the destination has at least one outgoing link. This is done by link reversal.

2. The underlying network remains a DAG after performing the required link reversals.

There are two ways to perform link reversals namely,

Full link reversal: If a node $N \neq D$ has no outgoing link, it applies reversal to all incoming link turning each to an outgoing link. The full reversal propagate through the network. It stops when all nodes except the destination has at least one outgoing link.

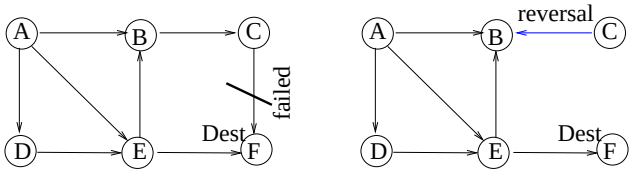
Partial link reversal: Every node N, other than the destination, keeps a list of its neighbouring nodes N' that have reversed the direction of the links (N, N'). At each iteration, each node N that has no outgoing links, reverses the directions of links (N, N') for all N' that do not appear in its list. Then it empties the list. If there is no such N' (all its neighbours appear in the list), N reverses all incoming links and empties the list.

Example

Figures 5.5 illustrates the full link reversal. The intial destination oriented graph is shown in part (a). The node labeled F is the destination. F is the only node without an outgoing link, but every other node has at least one outgoing link. When link $C \rightarrow F$ fails, C loses its only outgoing link. So, full link reversal is applied at C and its only incoming link $B \rightarrow C$ gets reversed. The result is shown in figure 5.5(b). Now B is left without an outgoing link. So, in second iteration link reversal applied to all incoming links of B. The result is shown in figure 5.5(c). It causes C to lose its only outgoing link. Therefore, another link reversal is applied to the link at C. Following this we have a new destination destination oriented graph as in figure 5.5(d).

5.3 DSDV

Destination Sequeneced Distance Vector (DSDV) routing is based on *distance vector routing* algorithm. It is a distributed version of the classical Bellman-Ford [5] shortest path algorithm modified for dynamic changing topology.



(a) Link $C - F$ fails, and C loses its only outgoing link.

(b) Link $C - B$ reversed.

(c) B loses its outgoing link $B-C$, so a full links reversal take place at B.

(d) Again C loses its outgoing link, so reversal takes place at C.

Figure 5.5: Full link reversal

The basic problem in employing the distance vector routing algorithm in mobile environment is formation of loops. Since, the nodes are mobile the topology of network changes rapidly, the information maintained at each node could become stale quickly. It may, therefore, introduce loops as well as exhibit the *count to infinity* problem.

The DSDV uses the routing tables maintained at the nodes for routing packets from a source to destination. Each routing table contains a list of destinations with the number of hops to reach each destination. Every entry in a routing table is tagged with a sequence number that assigned by the respective destinations. The source node consults its routing table for the destination and sends the data packet to the next hop in the path to the desired destination. The next hop repeats the actions and forwards to the node which is the next nearest hop to destination and so on, until the destination is reached. The major issue in DSDV is maintenance and update of routing tables.

5.3.1 Advertisement of Routes

All nodes periodically transmits the route updates or as soon as new information about the change in topology is available. However, there is no specific time interval for propagation of the updates. This follows from the fact that the mobile hosts are not expected to maintain any kind of time synchronization in movements.

Each mobile node advertises its own routing table to its current neighbors. Since the routes may change frequently due to movement of mobile nodes, the advertisement of routing table is carried out on a regular basis. The regular advertisement of routes restricts the possibility of disturbing the mobile hosts on *doze* mode. A node, in doze mode which misses the current route advertisements, will receive a route advertisement when it becomes active at some later point of time.

While forwarding of a message from a source to a destination requires support from other mobile hosts. But at the same time it is also desirable that nodes in doze mode are not unnecessarily disturbed. The routing packet broadcast by each node for a new route will consist of

1. The recent most sequence number of the route information to a destination as time stamped by the destination and known to the source.

2. The address of the destination.

3. The number of hops required to reach the destination.

A node receiving a new route information should broadcast the same to its neighbors. Apart from adding a new sequence number, the transmitting node increments the number of hops by one to indicate that the new route to destination includes itself as a hop. For the purpose of forwarding a packet, the route with most recent sequence number is used though the same is not typically used for route re-advertisement. The route re-advertisement is held back to allow settling time for a route information.

5.3.2 Propagation of updates

DSDV employs two different route updates: (i) incremental update, and (ii) full update. The route updates are propagated in form of an route update table. An entry in this table has three fields, namely,

1. The destination,
2. The route metric or number of hops, and
3. The sequence number

Usually incremental updates are small and periodically advertised. A route update table (also known as forwarding table) typically needs less than a single NPDU (Network Protocol Data Unit). As number of updates grows, due to increased mobility of nodes with time, just sending incremental updates will not work. The number of entries in the forwarding table becomes large to fit into a single NPDU. In this situation it is preferable to go for a full dump. Full dumps are transmitted relatively infrequently and require a number of NPDUs. The interval of time between two full dumps will depend on the frequency of movements of the mobile hosts.

5.3.3 Propagation of Link Break Information

When a node N detects a broken link to a neighbor M, then N generates a fresh sequence number and modifies the distance of each destination to ∞ for which M was a an intermediate hop. This is the *only instance* in DSDV where a sequence number for a route table entry is assigned by nodes other than the destinations. The sequence numbers for real (finite) routes are *even numbers* and sequence numbers of routes with broken links (route metric is ∞) are always *odd numbers.* A broken link is, thus, is easily identified as the one tagged with an odd sequence number.

5.3.4 Stability of requirements

DSDV requires a stability time for propagation of route information. It helps to absorb fluctuations in the route tables during the period of quick changes of network topology. It also eliminates the need to rebroadcast the route updates that arrive with the same sequence number. The new information about topology received at a node is allowed to settle for while before being advertised by the node. It is possible that during the interval of route settling time, the network experiences further changes in topology after the arrival of the *last* route update. The reason behind this can be attributed to continuity of node movements. If a node has just began to move then it is expected to continue move for some time before it reaches a desired place. By employing

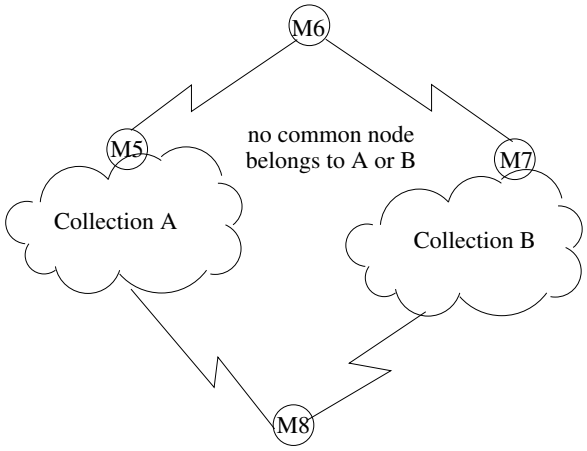


Figure 5.6: Flooding due to fluctuations in routes.

a delay in route advertisement, the problem re-advertisement of the routes in quick successions (which may lead to a *broadcast storm*) is controlled.

Furthermore, if the stability delay is not employed, flooding can occur even when a destination does not move. Such a scenario can be constructed as follows as depicted in Figure 5.6. The route update for the destination node $M8$ reaches a node say $M6$ from both $M5$ and $M7$. The collection of nodes in the network is such that paths reaching $M6$ from $M8$ can be grouped into two classes. One set of these paths is such that every path in it passes through $M5$ and do not interset any of the paths from the second set each of which passes through $M7$. Suppose that the route update from $M7$ reaches $M6$ 8 seconds earlier than that from a path via $M5$. Also let the number of hops in in a path via $M5$ is 20 and that in a path through $M7$ is 19. Suppose this happens every time a new sequence is issued by $M8$. Then the path information at $M6$ for $M8$ fluctuates back and forth every time a new sequence number is issued at $M8$. If any new route information gets propagated without stability delay, the receiving node will propagate the routes with new metrics in quick succession to its neighborhood. It causes the network flooding problem.

5.3.5 Guarantee for loop free paths

Assume that system is in steady state. All tables of all nodes have converged to actual shortest paths. Then the collection of next hops for a specific destination is logically an inverted rooted tree with the root at that destination. Let us consider the inverted tree for one destination say x, and examine the changes that occur due to movement of mobile nodes. $G(x)$ directed graph for x defined by the edges of the form $(i, n_i(x))$, where $n_i(x)$ is the next hop for destination x at node i.

Lemma 2 *The operation of DSDV ensures at every instant $G(x)$ is loop-free.*

Proof: Potentially a loop may be introduced if $n_i(x)$ changes. If $n_i(x)$ is set to *nil*, implying a link break, then no loop can be introduced. Assume that $n_i(x)$ is changed to a non-null next hop. There are two ways in which $n_i(x)$ at i can change when i receives an update about a new route to x from a neighbor, say k with sequence number $s_k(x)$, namely

1. $s_k(x) > s_i(x)$, where $s_i(x)$ is the sequence number for destination x in route table at i.

2. $s_k(x) = s_i(x)$, but metric $d_k(x) < d_i(x)$.

In the first case i cannot introduce a loop. Because i propagates sequence number $s_i(x)$ to its downstream neighbors only after receiving it from current next hop. Therefore, the sequence number stored at next hop is always greater or equal to that stored at the node i. In fact, the set of sequence numbers stored at intermediate nodes upstream from any node i to the destination x will form a non-decreasing series. If indeed, k introduced a loop then it implies that $s_k(x) \leq s_i(x)$ contradicting the initial assumption $s_k(x) > s_i(x)$.

The loop-free property in the second case follows from the fact that in presence of static or decreasing link weights, the algorithm always maintains loop-free paths. ∎

5.3.6 Forwarding table and update propagation

The forwarding (routing) table will be used to forward packets from a node to the next hop along route to destinations. This table at each node, stores the routing information (the next hop) for the destinations from that node. A basic routing table entry consists of following fields:

- *Destination*: the address of the destination node.
- *Next hop*: the next node along the path from the current node to destination.
- *Metric*: the number of hops required to reach the destination.
- *Sequence number*: the sequence number assigned by the destination to the current table entry.
- *Install time*: the time when the current entry was made. It helps to determine when the stale routes are to be deleted.
- *Stable data*: provides a pointer to a table holding information about stability of a route. It is used to damp fluctuations in the network.

If all links on a route to a destination are live then the corresponding sequence number for that destination must be even. An odd sequence number indicates that the route is broken. Install time is used to detect if any route has become *stale* (expired). However, install time is not very critical for working of DSDV algorithm, because detection of a link breakage is propagated through the ad hoc network immediately.

Stable data is pointer to a table that stores information about stability of routes to a destination. The route stability information is used to damp fluctuations. The stable data records last and average settling time for every route. When a node, say N, receives a new route (with newer sequence) N updates the routing table but will not immediately advertise the new route. The new route will be advertised only after 2*(*average settling time*). So, DSDV keeps two routing table, one which is used for routing and the other which is used for advertising the route updates. The advertised route table of a node is constructed from its stable data and as explained earlier have the following structure:

```
destination
metric
sequence_number
```

When advertising routes, the node which advertises the table places route to itself as the first entry. The nodes which have any significant route changes affecting them are placed after that in the advertised route table. A significant change could be a change in metric of the route or new sequence

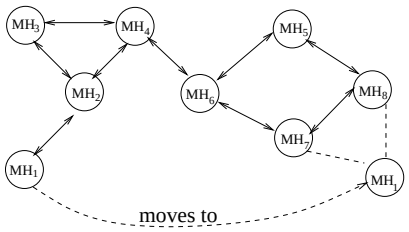


Figure 5.7: An example for execution of DSDV

number. The rest of the advertised route table is used to include all nodes whose route sequence numbers have changed. If too many updated sequence numbers are advertised then a update cannot be included in a single packet. To get around this problem a fair selection policy may be used to transmit the updates in round-robin fashion by several incremental update intervals.

5.3.7 Example

Consider the snap shot of the routes in an ad hoc network shown in figure 5.7. This figure is taken from the original paper [14]. Prior to the movement of mobile host MH_1 to new position as indicated in the figure, the forwarding table [14] for a node, say MH_4, could be as the one shown below.

Destination	Next Hop	Metric	Sequence Number	Install Time
MH_1	MH_2	2	S180_MH_1	T001_MH4
MH_2	MH_2	1	S384_MH_2	T001_MH4
MH_3	MH_2	2	S444_MH_3	T001_MH4
MH_4	MH_4	0	S236_MH_4	T001_MH4
MH_5	MH_6	2	S352_MH_5	T002_MH4
MH_6	MH_6	1	S278_MH_6	T001_MH4
MH_7	MH_6	2	S232_MH_7	T002_MH4
MH_8	MH_6	3	S190_MH_8	T002_MH4

Note that the above table does not include information concerning stable data. The advertised route table for the node MH_4 may look like the one shown below.

Destination	Metric	Sequence Number
MH_4	0	S236_MH_4
MH_1	2	S180_MH_1
MH_2	1	S384_MH_2
MH_3	2	S444_MH_3
MH_5	2	S352_MH_5
MH_6	1	S278_MH_6
MH_7	2	S232_MH_7
MH_8	3	S190_MH_8

Suppose now MH_1 moves to a new position as shown in figure 5.7 by a broken line; direction of move is indicated by the direction of the arrow. The new position is within the neighborhood of nodes MH_7 and MH_8. The old link between MH_1 and MH_2 gets snapped. The new route to destination MH_1 is advertised and a new metric is finally received by node MH_4 after sometime. The internal forwarding table at node MH_4 changes as follows.

Destination	Next Hop	Metric	Sequence Number
MH_1	MH_6	3	S580_MH_1
MH_2	MH_2	1	S486_MH_2
MH_3	MH_2	2	S634_MH_3
MH_4	MH_4	0	S856_MH_4
MH_5	MH_6	2	S502_MH_5
MH_6	MH_6	1	S378_MH_6
MH_7	MH_6	2	S358_MH_7
MH_8	MH_6	3	S390_MH_8

Notice the flag corresponding to MH_1 has been set to M indicating that the route entry has changed. The advertised route table also changes at MH_4. This table will have following new information

Destination	Metric	Sequence Number
MH_4	0	S856_MH_4
MH_1	3	S580_MH_1
MH_2	1	S486_MH_2
MH_3	2	S634_MH_3
MH_5	2	S502_MH_5
MH_6	1	S378_MH_6
MH_7	2	S358_MH_7
MH_8	3	S390_MH_8

Except for the node MH_1 the metric for all other nodes remain unchanged.

5.4 DSR

Dynamic source routing [7] is a reactive protocol. DSR uses source routing to route the messages. In other words, the entire sequence of hops from the source to the destination is embedded with message packets that gets exchanged between nodes. This is in contrast with other reactive routing protocols such as TORA [11], or AODV [15], where the sender just has to know only the next hop to destination. The advantages of keeping the route information with source are as follows.

- The requirement for periodic route advertisement is eliminated.
- In a less dynamic environment DSR provides valid routes for more often than not.
- DSR finds a route also when links are unidirectional.
- DSR initiates a route discovery in the case when route becomes invalid.

Periodic route advertisement not only consumes bandwidth, but it requires the nodes to be in connected state in order to receive the route advertisements. So, DSR by eliminating periodic route advertisements enables wireless devices to conserve battery power. When no significant node movements take place topology remains most static. So, if the routes to between source and destination pairs are known DSR can almost always provide valid routes. In the wireless environments, the quality of message transmission between hosts

may not be exactly same in both directions. In DSR, entire path from source to destination is embedded in each packet when it is sent. Even for sending route reply DSR initiates a route discovery. So, this approach routing works for the situation when links are unidirectional.

The basic assumptions for DSR to work properly are as follows.

- The hosts can move without notice, but they do so with a moderate speed with respect to packet transmission latency.
- The speed of movement of nodes is also moderate with respect to the wireless transmission latency of the underlying network hardware in use.
- The hosts can turn into *promiscuous* mode to listen to the activities within their respective wireless range.

5.4.1 Overview of the algorithm

Each node in the network maintains a route cache. All the routes learned so far by a node are stored in its route cache. In order to send a packet a node first checks the route cache to find a valid route to desired destination. If no route is found then a route discovery protocol is initiated to discover the route to the destination. The normal processing at the node continues, pending the discovery of route. The packets meant for the desired destination may be buffered at the node till such time when a route to the destination has been determined. The other alternative may be to retransmit the discarded packets which will eliminate the need to buffer the packet. Each entry in route cache is associated with an expiry time, after which the route entry is purged from the cache. It is possible that a route may become invalid due to any node, the source, the destination or an intermediate node, moving out of wireless transmission range, failing or being powered off. The monitoring the validity of the route is called route maintenance. When route maintenance detects a problem of the kind mentioned above, route discovery may be called again to discover a fresh route to destination.

5.4.2 Route discovery

The route discovery scheme works by flooding a *route request* (RREQ) packet. An RREQ packet contains *source ID*, *destination ID*, *route record* and a

unique *request ID*, set by the source. The request ID allows intermediate nodes to discard the duplicate route request packet and thus prevent repeated flooding. When a node N receives a RREQ packet, it reacts as follows:

Step 1 If the source ID and request ID matches with any of the recent RREQ packets seen by the node then discard the packet. No further action needed.

Step 2 Else if the address of the target ID matches with any of the recently seen RREQs then discard the packet and do not process any further.

Step 3 Else if target ID matches with the ID of the processing node then extract the route record from the request packet, the same would have accumulated in the route record of the request packet, and unicast a reply back to source by using the extracted route.

Step 4 Else append the address of self with the route record of the request packet and re-broadcast the request

The scheme for RREQ processing is quite straightforward. If an intermediate node N receives a RREQ it should first ensure that either the same RREQ or a RREQ for the same destination has not been forwarded by it earlier. In both these cases N is awaiting a route reply to arrive from the destination. There is also a slight anomaly in our description of Step 3 and the assumption that wireless transmission may not work well in both direction of a route. When a route reply has to be sent to initiator of the route discovery the possible alternatives are:

- If the destination has an entry for the source in its own route cache, this route may be used to send the reply packet.
- Otherwise, it is possible to use the reverse of route record extracted from the request packet. In this case we consider that the route is bidirectional (with symmetric links).
- The third possibility is to let the reply packet ride piggyback to source on a route request packet initiated by the destination for the discovery of a route to source.

5.4.3 Route maintenance

Since, DSR is a source initiated routing scheme, route maintenance is basically limited to monitoring link breakage along the route at the time of message transmission. There are three ways of monitoring the error in route.

1. Hop-by-hop acknowledgement.
2. Passive acknowledgement.
3. End-to-end acknowledgement.

In hop-by-hop acknowledgement, at each hop, the node transmitting the packet can determine which link of the route is not functioning, and sends an error packet to the source. But hop-by-hop acknowledgement scheme would require a low level (data link level) support. If the underlying network does not support such a low level acknowledgement mechanism, passive acknowledgement can be utilized to discover the route in error. The passive acknowledgement works as follows. After sending a packet to next hop a node promiscuously listens to transmission of the packet by the next hop to the subsequent hop. If no transmission could be detected then the it may be assumed that there is break in link between next hop and the hop following it. The other straightforward technique could be to seek explicit end-to-end acknowledgement by setting a bit in forwarding message packet itself, which indicates an acknowledgement should be sent to the source. The problem of sending explicit acknowledgement back to the source is quite similar to the problem of sending a corresponding route reply packet back to the original sender of a route request. If the wireless transmission between two hosts works equally well in both direction the acknowledgement can be sent in the reverse direction using the same route as the one used by the original message. Otherwise, the host detecting a link breakage sends the error packet back to the source if the former has an entry for the latter in its route cache. When there is no unicast route to the source, the error packet can be sent by the detecting host by piggybacking the error on a RREQ packet for the original sender.

5.4.4 Piggybacking on route discovery

When the links are unidirectional, we can use the concept of piggybacking of data along with the route discovery to amortize the message delivery time

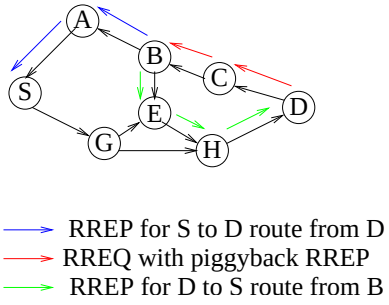


Figure 5.8: B unicasting piggyback data (D's RREP) to S

with route discovery delay. If a RREQ is propagated all the way upto a destination piggybacking has no problem except that the size of piggybacked data must be small. However, the piggybacked data will be lost if some intermediate node on the route that has a cached route to the destination sends a route reply from the cached data and discards the RREQ packet.

Without loss of generality, let us examine the problem for the case that a route discovery from source to destination is over and a route reply should now be sent from destination back to source. As shown in figure 5.8, a route $S \rightarrow G \rightarrow H \rightarrow D$ has been detected after a route discovery was initiated by S for D. Now a route reply (RREP) should be unicast to S by D. But suppose no such unicast route from D to S is known. Then D must now initiate a route discovery for the target S, but it sends RREP as piggyback data on RREQ for S. Suppose (an intermediate) node B on the route from D to S has a route, say $B \rightarrow A \rightarrow S$, in its route cache. So B can unicasts RREP for the route from D to S, via $B \rightarrow E \rightarrow H \rightarrow D$. But piggyback data (RREP) from D to S will be lost unless B unicasts to S the RREP being piggybacked by D with its RREQ to S. The blue arrows represent unicasting of RREP from D for S to D path by the intermediate node B. The green arrows represent unicasting of RREP from B for D to S route. The red arrows indicate the RREQ from D for destination S which carries piggyback RREP from D for the S to D path.

5.4.5 Handling route replies

Problems may arise when several mobile hosts receiving RREQ from an initiator send RREPs from their respective local route cache. Two noticeable

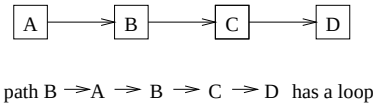


Figure 5.9: Loop formation in using cached data for route reply

problems arise due to asynchronous and autonomous operations of mobile hosts.

1. A number of hosts may send replies creating a flooding problem.

2. It is possible that the reply reaches the initiator from a host which is at a greater distance slightly ahead of time than that from a host at a lesser distance from the destination.

In order to avoid the problem of simultaneous replies and to eliminate replies indicating longer routes, following simple mechanism may be employed. Every node with a cached route to the destination delays the sending of RREP. The delay is set as $d = H * (h - 1 + r)$, where, H is a suitably chosen small constant, $0 < r < 1$ a randomly generated number, and h is the number of hops to reach the destination from the node sending the route reply. The delay d in sending route replies takes care of the situation where a reply with a poor route metric being sent earlier to the initiator than a reply with a better route metric. By operating in promiscuous mode a host may listen to all route replies reaching at any of its neighbours during the delay period. The host will not transmit the RREP packet if it listen to any RREP for the same target (the destination address for the RREQ packet) of a route request.

The more serious problem in using route cache for route reply is formation of a loop. Though the cached data for route entries themselves may not include loops, a loop may appear when a route gets established during the route discovery phase. For example, consider figure 5.9 where A has a cached route to destination D when node B initiate a route discovery for D. Node A sends the route to D from its cached data to B, which is $A \rightarrow B \rightarrow C \rightarrow D$. So the source route from B to D will become $B \rightarrow A \rightarrow B \rightarrow C \rightarrow D$. This route contains a loop, any message from B to D has to traverse $A \rightarrow B$ path once in backward direction and once in forward direction. To avoid such a loop formation, a possible approach could be as follows. If a route reply

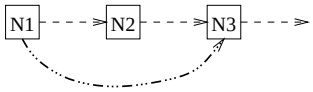


Figure 5.10: Reflecting shorter route updates

from cached data also includes the initiator, then this route is spliced into two parts. The first part is from the host that sent reply to the initiator and the second part is from the initiator to the target. The latter part is cached locally at the initiator. So, the entire path $B \rightarrow A \rightarrow B \rightarrow C \rightarrow D$ is spliced at A and the part $B \rightarrow C \rightarrow D$ is sent to B in route reply. However, DSR prohibits any node from sending a RREP for the route where the node itelf does not appear. There are two reasons why this is not allowed.

1. Firstly, if a node N is a part of the route it returns, the probability of route's validity increases. This is due to the fact, that N will get a RERR if route were invalid.

2. Secondly, if and when a route becomes invalid then node N, which originally sent the RREP, also gets the RERR when route is invalidate. This ensures that stale data is removed from N's route cache in timely manner.

5.4.6 Operating in promiscuous mode

As discussed earlier in the previous subsection, the problem of receiving several route replies and the routes longer than the shortest route to the destination can be eliminated by turning into promiscuous receive mode. Operating in promiscuous receive mode is found to be advantageous in reflecting shorter route updates, specially in mobile environment. Consider the situation as depicted in figure 5.10. Node N_1 transmits a packet to some destination through the nodes N_2 and N_3 with N_2 being the next hop and N_3 being the second hop on the route. Since nodes operate in promiscuous receive mode, N_3 receives the packet being sent from N_1 to N_2, and infers that the actual route can be shortened by eliminating the hop N_2. This situation can occur when N_1 moves closer to N_3. In this case N_3 sends an *unsolicited* RREP to N_1 which can then update the local route cache. But the major problem in operating in promiscuous receive mode is that the nodes must be power-on

most of the time. This runs counter to the problem of limited battery power in mobile hosts.

5.5 AODV

Ad hoc On-demand Distance Vector (AODV) [15] routing is an adaptation of the classical distance vector routing algorithm to the mobile case. It is a distributed routing algorithm, and like DSDV algorithm employs a sequence number for each entry in the routing table at a source. The sequence number is created either by the destination or by the leader of the group in a multicast group. A node always uses the route with highest sequence number from among the available alternatives routes to a destination.

AODV uses three basic types of messages:

1. Route REQuest (RREQ),
2. Route REPly (RREP) and
3. Multicast route ACTivation (MACT).

The first two types of messages have obvious meanings. MACT message is used along with RREQ and RREP to maintain multicast route trees. The significance of MACT message will be clear when multicast routings is discussed.

Each node along a route from a source to a destination maintains only the next hop entry in AODV. Therefore, the route table entry at each node is small. AODV reduces need for the system wide broadcasts by localizing the propagation of change in network topology. For example, if a link status does not affect the ongoing communication or the maintenance of a multicast tree then no broadcast occurs. Only a global effect is observed when a distant source attempts to use a broken link. One or more nodes that are using a link are informed when it breaks.

AODV also maintains a multicast tree for group communication. Hence multicast, broadcast and unicast all are integrated into a single protocol. The route tables also create or update the reverse route as a RREQ is being sent progressively from the source to the destination. So any node, on a path along which RREQ has been forwarded, can reach the source node.

5.5.1 Design decisions

Some of the design decisions in AODV are as follows.

- Routes not used are expired and discarded.
- If available, an alternative route is used after the primary route has expired.
- If available, an alternative route can be used to bypass a broken link on primary route.

By expiring non-used routes, maintenance of stale routes can be avoided. However, managing simultaneous aging process of multiple routes between a source and a destination is not easy. Although in theory use of alternative routes is possible, it is possible that the alternative route may also become invalid by the time primary route expires. Furthermore, bypassing broken link may not always be possible if all alternative routes use the same broken link.

5.5.2 Route tables

AODV does not attempt to maintain routes. The routes are discovered as and when needed, and maintained as long as they are used. AODV uses sequence number to eliminate loops. Every node maintains its own sequence number. A multicast group's sequence number is maintained by its leader. A route table and a multicast route tables are maintained at each node.

The four important parts of a route table entry are: `next hop` `destination sequence number`, `hop count`, and `life time`. A typical entry in a route table consists of the following fields:

```
Source and destination IP addresses
Destination sequence number
Hop count
Next hop
Expiry time
Routing flags
Last hop count
List of precursors
```

Each intermediate node N along a route maintains a list of active nodes which use N as the next hop to forward data packets to a given destination D. All such nodes are called precursors of N. A node is considered active if it originates or forwards at least one packet for the chosen destination within `active_timeout` period. The `active_timeout` period is typically 3000ms. The precursor list is required for route maintenance when a link breaks. The routes to a destination from all precursors of a node N will become invalid if and when the route from N to the destination becomes invalid.

The entries in a multicast route table are similar to route table except for following extra information:

- More than one next hops are stored.
- Each hop associated with activated flag and a direction.
- Route can be used only after activated flag has been set.
- Direction is either upstream or downstream relative to the group leader.

The requirement for maintaining these extra fields will become clear when the multicast route discovery and maintenance is discussed.

5.5.3 Unicast route discovery and maintenance

A node which needs a route to a destination broadcasts RREQ. Any intermediate node with a current route destination can unicast a RREP to source. The information obtained by RREQ and RREP messages help to build routing tables. Sequence numbers are used to eliminate the expired routes.

Creating a RREQ

A node wishing to send a packet to some destination, first searches the local routing table for a matching entry. If the source has a route then it forwards the packet to next hop. Otherwise a route discovery is initiated by creating a RREQ. The required fields for a RREQ are:

```
Src_ID
Dest_ID
Src_seqNo
```

```
Dest_seqNo
Broadcast_ID
```

The sequence number from source is used to refresh the reverse route to source. Broadcast ID and source ID pair uniquely identify a RREQ. After forwarding an RREQ, a node will store it in a RREQ cache for some time. It helps to restrict flooding. For example, if an old RREQ arrives again at a node possibly through an alternative path, then the node by examining RREQ cache can determine the freshness of RREQ and discard it.

RREQ also used for constructing the reverse path to the source, since AODV assumes bidirectional links. By storing the reverse next hop, an intermediate node will know the route to unicast a reply to the source when it arrives. The source broadcasts the RREQ and sets a time out for reply.

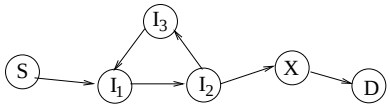
Processing a RREQ

On receiving a RREQ, a node first determines if the current RREQ was received earlier by checking its RREQ cache. If the source ID and the broadcast ID match an earlier RREQ existing in cache, the recipient discards the RREQ. Otherwise it sets up a reverse route entry for the source node in the route table. The reverse route entry consists of <`src_ID, src_seqNo, hop_cnt, nbr_ID`>, where `nbr_ID` is the ID of the neighbor from which the RREQ was received. For reverse path nbr_ID becomes the `next_hop`. As and when a RREP is received for the RREQ, the current node can forward the RREP to the source through the downstream neighbor on the reverse route towards the source. An intermediate node, receiving a RREQ, can also unicasts a RREP back to source if it has an unexpired entry for the destination, and the sequence number is of this entry is greater than or equal to the destination sequence number (last known sequence number) carried by the RREQ packet. Note that the unexpired path means that the path is active.

AODV provides a loop free route from a source to destination by employing sequence numbers like DSDV. A formal proof for the fact that AODV avoids formation of a loop is provided in Lemma 3.

Lemma 3 *AODV provides a loop free paths between source and destination pairs.*

Proof: The key idea behind the proof is that a route from a source to a destination imposes a logical ordering of the nodes along the path, based on destination sequence numbers and hop count. Where the assumptions are:



	I_1	I_2	I_3
Destination	D	D	D
Destination sequence no.	T_1	T_2	T_3
Next hop	I_2	I_3	I_1

Figure 5.11: A possible loop.

- Higher sequence number has precedence over hop count, and
- For the same sequence number, lower hop count has the precedence

AODV forces discovery of loop-free route from a source to destination by imposing the above conditions on the sequence numbers. According these, a node v can select w as its next hop on the path to a destination D provided

- the destination sequence number at v is less than the destination sequence number at w, or
- the destination sequence numbers are same but $DIST(v, D) < DIST(w, D)$

We can rephrase the sequence number conditions using the notion of downstream and upstream nodes on a path. If on path from v to D, w is closer to D than v, then w is considered as downstream to v with respect to D. The loop freedom is guaranteed because AODV never finds a path to a destination from a downstream node (w) via an upstream node (v). Let us assume that a loop exists as shown in the figure 5.11. The table shows the sequence number and the next hop for the route to destination D corresponding to the intermediate nodes I_1, I_2 and I_3. The loop is represented by the links $I_1 \rightarrow I_2$, $I_2 \rightarrow I_3$ and $I_3 \rightarrow I_1$. According to sequence number condition:

$$T_1 \leq T_2, T_2 \leq T_3, \text{ and } T_3 \leq T_2 \text{ so, } T_1 = T_2 = T_3 \tag{5.1}$$

Let H_i the number of hops from $I_i \rightarrow I_{i+1 \bmod 3}$. Now applying sequence number condition with relations 5.1 we must have $H_i = H_{i+1} + 1$. But for our example, $H_1 = 1$, $H_2 = 1$ and $H_3 = 1$. So $H_3 \neq H_1 + 1$, which is a contradiction. Therefore, a loop cannot exist. ∎

Expanding ring search

By localizing the search area for route discovery, it is possible to restrict flooding. For localizing the search, the trick is to use an expanding ring search by setting a TTL (time to live) value. The TTL specifies the diameter of the subsection of N/W in which RREQ must be flooded. If a route to the target not found in that subsection, a new RREQ with an increased TTL launched. The TTL value is recorded in route table. For subsequent route discovery to the same destination, the starting TTL is set the value in route table. This approach progressively enlarges section of the network in which route request packet is to be flooded. It localizes the route request to a portion of the network where the destination is most likely to be found.

RREP processing. An RREP packet has following five fields:

```
Src_ID
Dest_ID
Dest_seqNo
Hop_cnt
Lifetime
```

When any node receives a RREQ it can generate and send RREP for the RREQ provided that:

1. The node has an active route to the destination, and
2. The destination sequence number of the route being used by the node equal to or greater than the destination sequence number carried in the RREQ packet.

If an intermediate node generates an RREP, then it should include hop count to the destination, lifetime for the expiry of current route, and the destination sequence according to the information available in its own routing table.

The scenario where an intermediate node can send RREP is illustrated in Figure 5.12. Node I sets the hop count to 3 as it is 3 hops away from the destination node D. When no intermediate node along the path from source to destination has any active path, the RREQ eventually will reach the destination. On receiving RREQ, the destination node will generate and send a RREP.

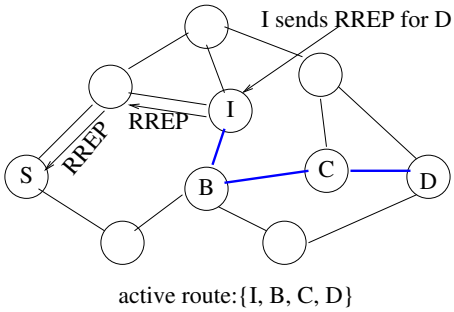


Figure 5.12: Intermediate node unicasting a RREP.

An intermediate node sets up the forward path to destination when it receives a RREP. First it creates a local route entry for the forward path as follows.

- Takes node ID from which RREP is received to define the next hop to destination.
- Takes ID of destination for indexing the route table.
- Adds 1 to hop count and store the same in the entry.
- Takes lifetime from RREP and places it in the entry.

After a route table entry has been created, the hop count carried by RREP is incremented by one. This ensure that if the route to destination were to pass through the current intermediate node than the hop count also includes the current node. Finally, the RREP is forwarded towards the source by using the next back hop or the next hop of the reverse route entry. Finally the RREP reaches the source, which can then start using the route. Clearly, RREP is a targeted unicast packet, because it is forwarded by using the next hop entries available in the routing tables of the nodes upstream towards the source.

Each time a route is used, its life time is updated. The route discovery process is explained in figure 5.13. The flooding of RREQ is shown by solid lines, the arrows denote the direction of flooding. The broken lines with arrows indicate the direction in which RREP traverses. The time outs received by the source from the nodes not having routes to the destination are also indicated in the figure.

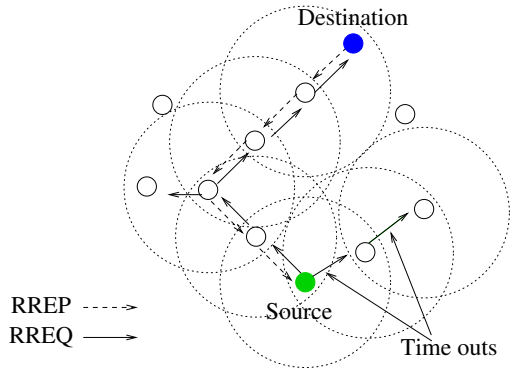


Figure 5.13: Route discovery

Route maintenance

A route is maintained as long as it is needed. The node movements affecting active paths are only considered. The route maintenance is for detecting link break on active paths and for building alternative paths when node movements occur. If the source moves, a route discovery can be re-initiated. When either the destination or an intermediate node moves then a RERR is sent to the source by the upstream end node (closer to source) of the broken link. Since precursor nodes are maintained for each destination at this end node, RERR can be sent to all nodes which has this end node as a hop in a valid route to the destination. The nodes receiving RERR mark the route to destination invalid and send RERR to all precursors. This way the breakage of a link becomes known to all the nodes which are using that link to communicate with the destination.

A source re-initiate route discovery when it receives RERR if the route is still needed. Figure 5.14 shows an active path from source S to destination D passes through four intermediate nodes $N0$, $N1$, $N2$, $N3$. After a while node $N3$ moves away to a new position. Due to breakage of links $(N1, N2)$ and $(N2, N3)$, the route from S to D is no longer valid. The route has to be invalidated by sending a route error packet (RERR) from $N2$ which is the farthest upstream intermediate node on the route. The broken arrows indicate the path that RERR traverses using precursors on the route from $N2$ to S.

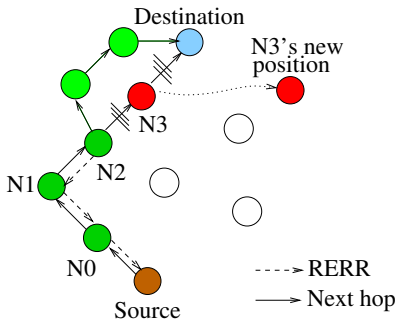


Figure 5.14: Route maintenance

Neighborhood information. Typically, the neighborhood information at a node is maintained by hello messages broadcast periodically from its neighbors nodes. If a node N has not sent anything within the last `hello_interval`, N sends a `hello` packet to inform the neighbors that it is still in the vicinity. Hello packet is in reality an unsolicited RREP. Hello packet not rebroadcast, as it carries a TTL = 1. A change in neighborhood of a node N is indicated if N fails to receive consecutive `allowed_hello_loss` of packets from another node which was previously in the neighborhood of N. The typical value of `allowed_hello_loss` = 2. On receiving the hello message from a neighbor, a node updates the lifetime of that neighbors. If entry for the neighbors does not exist, then the nodes creates one.

Link breaks. A link break on a route from a source to a destination is detected when the down stream neighbors of that link on the route fails to receive consecutive `allowed_hello_loss` hello packets in the usual hello interval. Link break is propagated upstream by sending a RERR packet. The RERR packet originates from the closest upstream node detecting the break. RERR is sent to each precursor node which have active route to destination.

5.5.4 Multicast Route Discovery and Maintenance

A multicast route discovery is essentially an extension of unicast route discovery. A node N initiates a multicast route discovery if

- Either N wants to send data to a group, or

- N wants to join a multicast group.

N initiates a multicast route discovery by creating a RREQ with destination IP address of the group. If N knows about the group leader G and has a valid path to G then it can unicast the RREQ by including the IP address of G, group's last known sequence number, and a join flag if N interested to join the group.

If RREQ is marked `join` then only members of group can respond. Otherwise, any node with fresh enough path to group can respond. A RREQ with `join` is processed by a node N as follows:

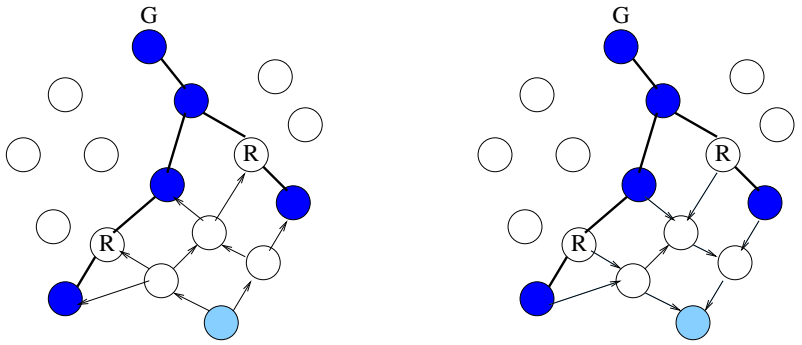
- If N is not a member of the multicast group, it creates a reverse entry for the source and rebroadcasts the RREQ.
- If N is a group member then it responds to RREQ by adding an *unactivated* entry for the source in its multicast table.

If N receives a RREQ (without `join` flag) for a group then RREQ is processed as follows:

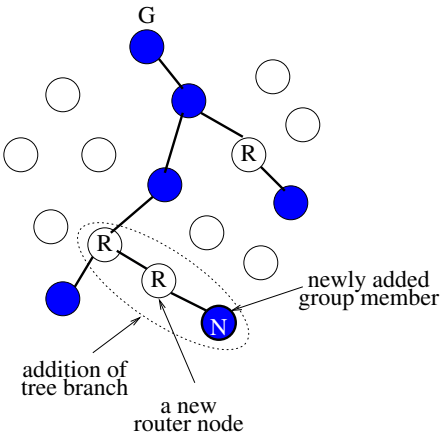
- If N is not a member of the group and does not have a route to the group it creates a reverse entry to the source and rebroadcasts RREQ.

The source waits for a time out to receive a reply. It can rebroadcast RREQ incrementing the broadcast ID by one. The source continues rebroadcast RREQ till receives a reply or number of broadcast becomes equal to `rreq_retries` after which it declare itself as the leader. Figure 5.15(a) illustrates how RREQ with `join` flag gets flooded in the network and a new tree branch is grafted into the multicast tree. The nodes labeled by R are not members of the multicast group, but are routers for the group. The new node N which wants to join the multicast group sends RREQ with `join` flag on. After RREQ has spread through the network RREPs originate from two router nodes and another group member. Then RREPs spread in the network as illustrated by Figure 5.15(b). Finally, Figure 5.15(c) shows that the new tree branch consisting of a new router node and N are added into the tree.

After a tree branch is added to the multicast tree nodes in the newly grafted branch should activated by sending an explict message on the branch. The reasons for explicit activation requirement are:



(a) Spreading RREQ. (b) Sending RREPs.



(c) Adding a new branch.

Figure 5.15: Grafting a new branch to multicast tree.

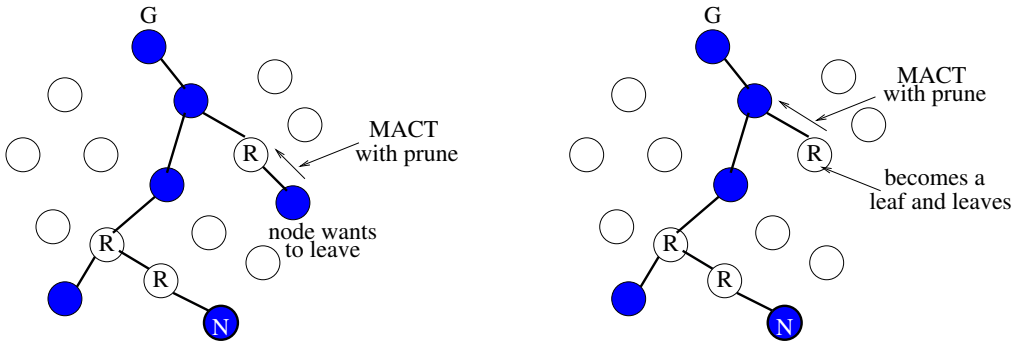
- RREPs for join request should trace out the path to the source so that one of the potential tree branches can be grafted into multicast tree.

- Apart form forwarding of the RREP with join request, the RREPs for nonjoin requests may also be forwarded in order to set up paths to multicast tree. Grafting a tree branch for nonjoin request is not warranted.

- Multicast data packets are sent as broadcast traffic. So, all the nodes which forwarded the RREPs to the source node have fresh routes to multicast tree. If no activation were needed, potentially, all these nodes can forward data packets to multicast group. This results in inefficient use of bandwidth. Therefore, only one of these routes should be used for forwarding data.

The source waits for the length of the interval for route discovery before it can use the path. The source first unicasts a MACT message to the node from which it received the RREP. The upstream neighbour sets actived flag against the source in its own route table before forwarding the same to the next hop upstream towards the originator. Each node on the route sets actived flag for hop, and forwards MACT to the next upstream neighbor until it reaches the originator of RREP.

MACT message is used also for deactivating an existing path if it is not needed any more. The deactivation proceeds as follows. If a leaf node (of the multicast group) wishes to leave the group, then the node may just prunes itself from the multicast tree. The next upstream hop deletes the entry corresponding to the deserter. But if a non-leaf node wishes to leave the group, it *can not* detach itself from the tree. Such nodes must continue to serve as a router for the multicast group. On receiving a MACT with prune flag, the next hop would delete the entry corresponding to the source of the MACT. If next hop is a router node and the deletion of the previous hop entry turns this a leaf node, then next hop would initiates its pruning from multicast tree by sending a MACT with prune flag. But if the next hop is a group member, then it may not want to revoke its group membership. The deletion of branch using MACT is shown in figure 5.16.



(a) Mact for leaving tree. (b) Tree after pruning.

Figure 5.16: Pruning after a member node leave the multicast group.

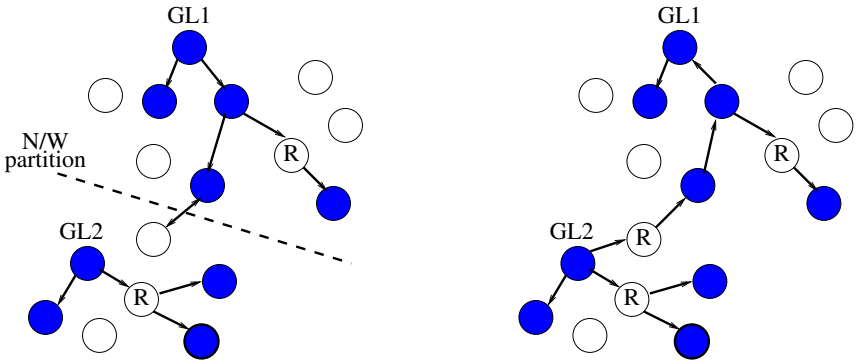
Maintenance of multicast routes

Maintenance of unicast route is need only as long as it is required. In the case of musticast routes, however, the multicast tree should be maintained for the entire lifetime of the group's existence. Every link requires maintenance so that each group member can access the multicast group. A link is assumed to have a lifetime which is equal to

$$\texttt{hello\_lifetime} = (1 + \texttt{allowed\_hello\_losses}) \times \texttt{hello\_interval}).$$

If no data packet is sent within a fixed interval called `hello-interval`, each node must receive a broadcast from its next hop neighbor in multicast tree. The broadcast may either be a RREQ, or a group hello or a hello packet. The hello packet is essentially a RREP with TTL value 1.

If a link $M \rightarrow N$ breaks then the downstream node N of the broken link initiates a route discovery for the multicast group. The upstream node of the broken link M may just a router for the multicast tree. If M becomes a leaf node after link break, it will try to detach itself. It will do so, sending a MACT with prune flag to its upstream node in the tree. However, MACT will be sent only after `prune_timeout`. The idea behind allowing a timeout is that M may still be available nearby N. If N initiates a route discovery to multicast tree, it may get connected to the multicast tree by an alternative path through M. For repairing the route, N sends a RREQ packet with a small TTL value, which includes also its distance of N from the group leader. The reason for a small TTL is that M may still be available nearby. Any



(a) Network partition. (b) Merging partitions.

Figure 5.17: Repairing link breakage in AODV.

node X, having an active route to multicast group, can respond to RREQ provided X is as close to the group leader as N. The above restriction prevents any node in the same side of break as N from responding. If this check is not enforced then a loop can occur if route replies were received from the nodes on both side of the link break. Figure 5.17 illustrates the route repair process. After N is able to add its subtree to multicast tree, it sends a MACT message to its next hop for activating the link. N also sends a MACT with update flag to its downstream neighbors. This message includes the new depth of N from the group leader. The downstream descendants in N's subtree update their respective depths from the group leader when they receive the MACT with update flag.

If node initiating repair does not receive any RREP after `rreq_retries`, then it is assumed that network is partitioned. Under this scenario, the partitioned nodes must also declare a new group leader. If N is a member of the multicast group then it can declare itself as the group leader. It sends then MACT message with update flag to its downstream neighbors. If N is not a group member and beecomes a leaf node after link break, then it can decide to prune itself by sending a prune message to next hop downstream. This action is also repeated at the downstream successor if N has only one neighbor. However, if N has more than 1 downstream successors then it selects one of the successors and sends a MACT with the group leader flag indicating the next node which is a member of the group receiving the MACT should become the new group leader. After the group leader is selected the

multicast group has more than 1 group leaders due to partition.

Topological changes over time may reconnect the two network partitions. The nodes in a tree partition learn about connectivity status when a GRouP Hello (GRPH) message is received from another partition. The ID of the group leader they receive will be different from known ID of the group leader. If a group leader receives GRPH message for the group for which it is the leader, then the leader with lower IP address $GL1$ initiate a reconnection by unicasting a RREQ with repair flag to the other group leader $GL2$ using the node from which it received GRPH. The RREQ also includes $GL1$'s multicast group sequence number. Every node in $GL2$'s tree which receives this RREQ forwards it towards $GL2$. When RREQ reaches $GL2$, it updates the group sequence number by taking the maximum of two and adding 1 to it. Then a RREP is sent back to $GL1$. This RREP also should have repair flag. As the RREP traverses back on the path from $GL2$ to $GL1$ every link along the path is oriented towards $GL1$, the nodes update their routing tables, activate the link. The tree partitions are connected when RREP reaches $GL1$ with $GL2$ becoming the leader. Every node is updated about the new group leaders identity by GRPH message which is periodically broadcast from the group leader as a unsolicited RREP with TTL value larger than the diameter of the network. It reaches every node in the tree which update their sequence number and the group leader's ID.

Bibliography

[1] Ballardie, T., Francis, P., and Crowcroft, J. Core based trees (cbt) an architecture for scalable inter-domain multicast routing. *SIGCOMM, ACM* (1993), 85–95.

[2] Carthy, P. M., and Grigoras, D. Multipath associativity based routing. In *Wireless On-demand Network Systems and Services (WONS 2005)* (2005), pp. 60–69.

[3] Chen, T. W., and Gerla, M. Global state routing: A new routing scheme for ad-hoc wireless networks. In *Proceedings of IEEE ICC'98* (1998), pp. 171–175.

[4] Chiang, C.-C., and Gerla, M. Routing and multicast in multihop, mobile wireless networks.

[5] Corman, T., Leiserson, C., Rivest, R., and Stein, C. *Introduction to Algorithms*. MIT Press, Cambridge MA, USA, 2001.

[6] Ji, L., and Corson, M. A lightweight adaptive multicast algorithm. In *Proceedings of Globecom'98* (1998), pp. 1036–1042.

[7] Johnson, D. B., and Maltz, D. A. DSR the dynamic source routing protocol for multihop wireless ad hoc networks. In *Ad Hoc Networking*, C. E. Perkins, Ed. Addison-Wesley, 2001, ch. 5, pp. 139–172.

[8] Ko, Y.-B., and Vaidya, N. H. Location-aided routing (lar) in mobile ad hoc networks. *Wireless Networks 6*, 4 (2000), 307–321.

[9] Marina, M. K., and Das, S. R. Ad hoc on-demand multipath distance vector routing. Tech. rep., Computer Science Department, April

2003. Earlier versions in IEEE ICNP (November 2001) and ACM SIGMOBILE Mobile Computing and Communications Review Special Feature on the First AODV Next Generation Workshop (July 2002).

[10] Murthy, S., and Garcia-Luna-Aceves, J. J. An efficient routing protocol for wireless networks. *ACM Mobile Newteorks and Application Journal* (1996), 183–197.

[11] Park, V. D., and Corson, M. S. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceeding of IEEE INFOCOM'97* (1997).

[12] Pei, G., Gerla, M., and Chen, T.-W. Fisheye state routing in mobile ad hoc networks. In *ICDCS Workshop on Wireless Networks and Mobile Computing* (2000), pp. D71–D78.

[13] Perkins, C. E., Ed. *Ad Hoc Networking*. Addison-Wesley, 2001.

[14] Perkins, C. E., and Bhagawat, P. Highly dynamic destination-sequenced distance-vector (dsdv) algorithm for mobile computers. *Computer Communication Review* (1994), 234–244.

[15] Perkins, C. E., and Royer, E. M. Ad-hoc on-demand distance vector routing. *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications* (1994), 90–100.

[16] Perlman, M. R., and Haas, Z. J. Determining the optimal configuration of the zone routing protocol. *IEEE Journal on Selected Areas of Communication 17*, 8 (1999), 61–81.

[17] Royer, E. M., and Toh, C.-K. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Magazine on Personal Communication 17*, 8 (1999), 46–55.

[18] Toh, C.-K. A novel distributed routing protocol to support ad hoc mobile computing. In *Proceedings of Fifteenth Annual Internation Phoenix Conference in Computers and Communication* (1996), IEEE, pp. 480–486.

[19] Zou, X., Ramamurthy, B., and Magliveras, S. Routing techniques in wireless ad hoc networks - classification and comparison. In

Proceedings of the Sixth World Multiconference on Systemics, Cybernetics, and Informatics, SCI 2002, July 15-18, 2002, Orlando, Florida, USA (July 2002), N. Callaos, Ed., vol. IV, IIIS.