Chapter 7
Routing Protocols for Mobile Ad Hoc Network

7.1 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 communication in a MANET. A MANET differs from a wired network in many ways. Some of these are attributable to general characteristics of wireless networks, while the others are due to the characteristics specific to MANETs [4]. Some of the typical characteristics a MANET are:

- The users in a MANET may neither wish nor be in a position to perform any administrative services needed to set up or to maintain a network.
- The topology of a MANET changes dynamically due to mobility of the nodes.
- Each node in a MANET is completely autonomous and acts both an ordinary node and a router.
- A node in a MANET has relatively less resources compared to a node in wired network.
- All nodes have similar capabilities and identically responsible for communication in a MANET.
- Any mobile node may either join, or leave MANET at any point of time.

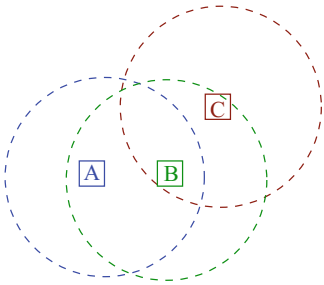
The problems associated with wireless connectivity combined with the limitations of MANET create a number of difficulties in designing routing protocols. In particular, these protocols are expected to address and resolve the following important issues.

1. Communication in a MANET is broadcast based. Unless carefully designed, routing protocols may generate high volume of unnecessary traffic in the network leading to flooding and congestion.
2. In order to participate in discovery and maintenance of the routes, the nodes in a MANET should be in active mode. However, no node should be forced to wake up when it is operating in low energy mode to save its battery power.

© Springer Nature Singapore Pte Ltd. 2017

R.K. Ghosh, *Wireless Networking and Mobile Data Management*,
DOI 10.1007/978-981-10-3941-6_7

Fig. 7.1 Ad hoc network



3. The processing load due to discovery and maintenance of routes should be distributed as much evenly as possible among the nodes in a MANET.
4. Many redundant paths have to be maintained between every pair of source and destination. This may causes unnecessary increase in the size of routing updates that must be sent over the network.

Each node in a MANET has a fixed wireless range. Any mobile host which appears inside the range of a transmitting host can listen/capture the messages being transmitted. In a MANET, every active node is expected to participate in forwarding a message towards the destination node. If a destination host is not directly reachable from a source then all hosts which receive the transmission from the source, forward the same to their respective neighbors. The neighbors, in turn, repeat the same until the message finally reaches the destination. For example, consider Fig. 7.1 where A, B and C represent three mobile hosts. The wireless ranges of these hosts are indicated by the respective circles having centers at A, B and C. The figure indicates that A and B are within the wireless range of each other, so are B and C. But C cannot receive any transmission directly 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 a rich source of problems for research.

A number of routing schemes have been proposed [1, 8, 9, 11, 12, 14–16] 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 in execution.
- 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 a 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. An interested reader 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.

7.2 Classification of Routing Protocols

Routing in any network can be viewed abstractly as finding and maintaining the shortest-path between two communicating nodes in a weighted graph. Each node maintains a preferred neighbor, which is the next hop on the path to reach a destination. Each data packet should have the address of the destination in its header. An intermediate node forwards a data packet to the next hop closer to destination by consulting the locally maintained table known as route table. The routing protocols differ in the manner in which the routing tables are constructed and maintained. Based on characteristics of the routing protocols, Royer and Toh [17] have suggested a classification scheme. As shown in Fig. 7.2, the routing schemes can be classified into two broad classes, namely,

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

Examples of distance vector based table driven protocols are Destination Sequenced Distance Vector (DSDV) [14], Wireless Routing Protocol (WRP) [10] and Cluster Gateway Switch Routing (CGSR) [2]. DSDV is a distributed version of Bellman-Ford shortest path algorithm which relies on destination sequence numbering scheme to avoid loops. CGSR protocol is a logical descendant DSDV protocol. But, it uses

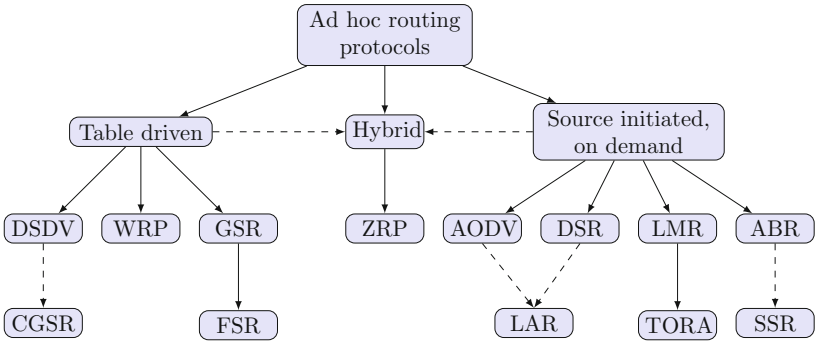


Fig. 7.2 Classifying routing algorithms

a clustered multihop organization scheme for network instead of a flat structure. WRP [10] does not use sequence number, but uses second-to-last hop information for each destination. It forces each node to check consistency of its predecessor's information to avoid count-to-infinity problem. Fisheye State Routing (FSR) [12] protocol is a direct descendant of Global State Routing (GSR) [1] protocol. GSR is a link state routing algorithm it exchanges vector of link states among the neighbors during route discovery.

Examples of source initiated reactive protocols are Ad hoc On-demand Distance Vector (AODV) [15], Dynamic Source Routing (DSR) [8], Lightweight Mobile Routing (LMR) [7] and Associativity-Based Routing (ABR) [18]. AODV [15] is inspired by DSDV protocol. Instead of proactively maintaining routes, AODV reactively discovers routes when needed. Like DSDV, it uses destination sequence number to avoid formation of loops. In DSR [8], each source provides the complete path to a chosen destination for routing a packet. Each node maintains a route cache for the routes to other nodes that it is aware of. The route cache of a node is updated as the node learns about new routes to more destinations. Location Aided Routing (LAR) [9] uses location information to reduce flooding during route discovery. The major problem in LAR is that it requires nodes to be aware of their location information. Route discovery and establishment is similar to AODV and DSR.

LMR [7] is a link reversal based routing scheme based on Gafni-Bertsekas [6] algorithm. The objective of LMR is not to find shortest path but to find any path between a source and destination pair. Temporarily Ordered Routing Algorithm (TORA) [11] is a direct descendant of LMR. The key feature of TORA is localization of control messages in the neighborhood where the topology changes occurs in the network. In order to accomplish it, the nodes maintain routing information about immediate neighbors. ABR [18] obtains stable route by selecting routes on the basis of degree of association stability among mobile nodes. Every node sends out periodic beacons to notify its existence which is used to update the respective associativity ticks of the current node with the beacon dispatch nodes. Association stability of one node with another is defined by connectivity over time and space. Signal Stability Routing (SSR) [5] is a logical descendant of ABR. It chooses route based on signal strength and location stability.

In this chapter our attempt is not to give a full review, but focus discussion on a few interesting protocols which exposes the characteristics and the abstraction of routing problem in a network with dynamic topology.

All MANET routing protocols are based on existing Internet routing protocols. Primarily three design approaches are used for Internet routing protocols, namely,

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

Our discussion in this chapter is restricted only to distance vector based routing algorithms. Focusing only on distance vector algorithms may appear to be an incomplete exploration of the area. However, the reasonbehind this decision is to introduce

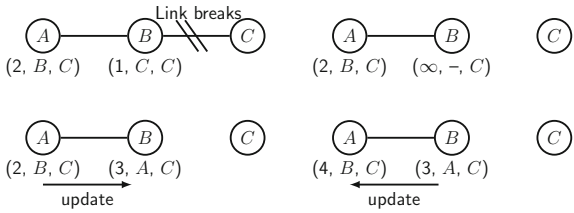


Fig. 7.3 Count to infinity problem

the problem domain to the reader rather than just throwing up research challenges. Furthermore, we believe that ad hoc networks are mostly low diameter and small sized networks. Therefore, the research efforts made in design of multihop, scalable routing algorithms have only theoretical value with no practical utility.

7.2.1 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 the next 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 possible path between the two nodes.

A major problem that may arise in formation of a route using a distance vector routing scheme is referred to as *count-to-infinity*. Figure 7.3 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 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 keep continuing resulting in count-to-infinity problem.

7.3 Destination-Sequenced Distance Vector Routing

Destination Sequenced Distance Vector (DSDV) is a distance vector based routing protocol. It is essentially a distributed version of the classical Bellman-Ford [3] shortest path algorithm modified for dynamically changing network topology. The fundamental problem in employing the distance vector routing algorithm in a mobile environment is formation of loops. Since, the nodes are mobile, the change in network topology occurs rapidly. Consequently, the information maintained at each node

quickly becomes stale. It may, therefore, not only introduce loop but may also exhibit *count to infinity* problem.

DSDV maintains routing tables at each node for routing packets from a source to a destination. Each routing table contains a list of destinations with the number of hops to reach a destination. Each entry of a routing table is tagged with a sequence number assigned by the corresponding 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 the destination and so on, until the destination is reached. The major issue in DSDV is the maintenance of the routing tables.

7.3.1 Advertisement of Routes

All nodes periodically transmit the route updates, or as soon as any new information about the change in topology is available. However, there is no fixed time interval for the propagation of 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 allows the mobile hosts to continue in *doze* mode and resume as they wish. So, even if a node (not in active mode) misses out the current route advertisements, it receives a route advertisement on turning to active mode.

Forwarding a message from a source to a destination requires support from other mobile hosts. However, the nodes operating in doze mode should not be forced to participate in forwarding of a message. A route advertisement packet broadcast by a node consists of the following important information.

1. The recent most destination sequence number of the route to a destination as known to the source.
2. The address of the destination.
3. The number of hops required to reach the destination.

A node on receiving a new route information should broadcast the same to its neighbors. Apart from adding a new broadcast 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. The broadcast sequence number is different from the destination sequence number. A broadcast sequence number helps to eliminate repeated forwarding of the same route advertisement. Unless stated otherwise, a sequence number always means a destination sequence number.

For the purpose of forwarding a data 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 some settling time for a new route information.

Two different route updates are advertised: (i) incremental update, and (ii) full update. The incremental updates are small and advertised more frequently than the full updates. An incremental update 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 do 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 may require a number of NPDUs. The interval of time between two full dumps will depend on the frequency of movements of the mobile hosts.

7.3.2 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 an intermediate hop. This is the *only instance* in DSDV where a sequence number for a route table entry is generated by nodes other than the destinations. The sequence numbers for real (finite) routes are always *even numbers*. The sequence numbers of the routes with broken links (with route metric ∞) are always *odd numbers*. A broken link is, thus, easily identified as the one tagged with an odd sequence number.

7.3.3 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 a 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 *latest* 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 pauses at a place for a while. By employing a delay in route advertisement, the problem re-advertisement of the routes in quick successions (which may lead to a *broadcast storm*) is controlled. 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 Fig. 7.4 adopted from the example from the original DSDV paper [14]. 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 intersect any of the paths from the second set each of which passes through $M7$. Suppose $M6$ receives the route update from

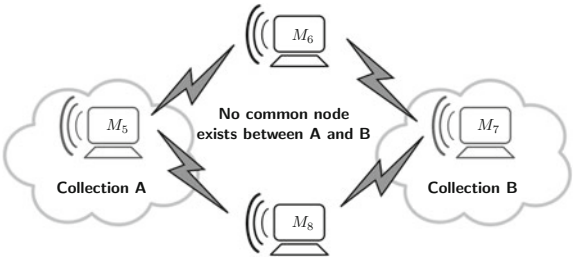


Fig. 7.4 Flooding due to fluctuations in routes

$M7$ eight seconds earlier than that from a path via $M5$. Also let the number of hops 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 for $M8$ at $M6$ 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 propagates the routes with new metrics in quick succession to its neighborhood. It causes flooding in the network.

7.3.4 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 any destination D is logically an inverted rooted tree with the root at D.

Consider an inverted tree for one destination, say x, and examine the changes that occur due to movements of the nodes. Let $G(x)$ be the directed graph of 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 7.1 *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*, it implies a link break. Therefore, a loop cannot be formed.

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 the 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 the sequence number $s_i(x)$ to its downstream neighbors only after receiving it from any of its current neighbors. Therefore, the sequence number stored at any of the neighbors of i 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 forms a non-decreasing series. If indeed, k were to introduce a loop, then it would mean $s_k(x) \leq s_i(x)$ which contradicts the assumption $s_k(x) > s_i(x)$.

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

7.3.5 Forwarding Table and Update Propagation

A forwarding (routing) table stored at each node is used to forward packets from the node to the next hop along a route to a destination. A basic routing table entry consists of the 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*: is a pointer to a structure holding information of a new route which is likely to supersede the current route after sometime.

If all links on a route to a destination are live then the corresponding sequence number for that destination must be even. 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.

The stable data is a pointer to a structure that stores the information about the stability of routes to a destination. The stability information is used to dampen fluctuations routes. The fluctuations may happen due to continuity of the node movements that has not yet settled. The stable data records the last and the average settling time for every route. Therefore, when a node, say N, receives a new route R, N does not immediately advertise R unless it is a route to a previously unreachable destination.

A new route to an existing destination is advertised only after 2*(*average settling time*). Essentially, DSDV keeps two routing tables. One table is used for forwarding the packets, and the other one is used for advertising of the route updates. The advertised route table of a node is constructed from its stable data, and has the following structure:

```
destination
metric
sequence_number
```

For advertising routes, a node places the route to itself as the first entry. Then all the nodes which have experienced significant change in topology, since the previous advertisement, are placed. A change in route metric is considered as a significant change. 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.

7.3.6 Example

Consider the snap shot of the routes in an ad hoc network shown in Fig. 7.5. This figure is adopted from the original paper [14]. Prior to the movement of a mobile host MH_1 to a new position as indicated in the figure, the forwarding table [14] for a node, say MH_4, could be as shown in Table 7.1. The above table does not include

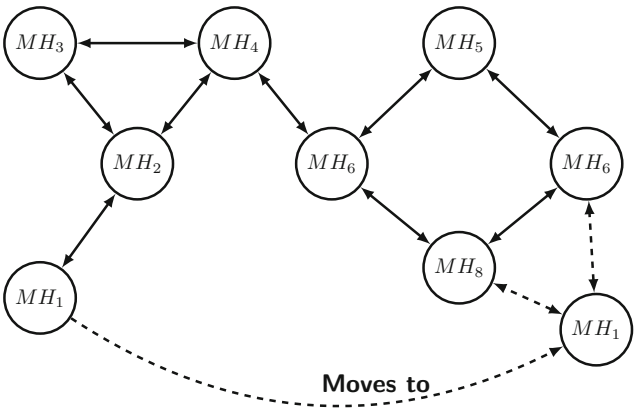


Fig. 7.5 An example for execution of DSDV

Table 7.1 Forwarding table of MH_4

Destination	Next hop	Metric	Flag	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

Table 7.2 Advertised route table for node MH_4

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

information concerning stable data. However, the advertised route table for the node MH_4 may be as the one shown in Table 7.2. Suppose now MH_1 moves to a new position as shown in Fig. 7.5 by a dashed line; the direction of movement is indicated by the arrow head. 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 indicated by Table 7.3. Notice that the flag for MH_1 has been set to M indicating that the route entry has changed. The install time of the entry also has changed. The advertised route table also changes at MH_4. This table has new information as illustrated by Table 7.4. Except for the node MH_1 the metrics for all other nodes remain unchanged.

Table 7.3 Change in forwarding table of node MH_4

Destination	Next hop	Metric	Flag	Sequence number	Time instal
MH_1	MH_6	3	M	S580_MH_1	T500_MH4
MH_2	MH_2	1		S486_MH_2	T001_MH4
MH_3	MH_2	2		S634_MH_3	T001_MH4
MH_4	MH_4	0		S856_MH_4	T001_MH4
MH_5	MH_6	2		S502_MH_5	T002_MH4
MH_6	MH_6	1		S378_MH_6	T002_MH4
MH_7	MH_6	2		S358_MH_7	T002_MH4
MH_8	MH_6	3		S390_MH_8	T002_MH4

Table 7.4 Change in forwarding table of node MH_4

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

7.4 Dynamic Source Routing

Dynamic source routing [8] is a reactive protocol. DSR uses source routing to route the messages. In other words, the entire sequence of hops from a source to a destination is embedded with message packets that gets exchanged between the two nodes. This is in contrast with the other reactive routing protocols such as TORA [11], or AODV [15], where the sender just has to know the next hop to the destination. The advantages of keeping the route information with the source are as follows.

- The requirement for periodic route advertisement is eliminated.
- In a less dynamic environment DSR provides valid routes 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 also 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, i.e., network topology remains mostly static. If the routes 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 exactly be the same in both directions. DSR may initiate a route discovery in order to send a route reply. More specifically, DSR approach to routing does not require the network links to be bidirectional.

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 to learn about new routes.

7.4.1 Overview of the Algorithm

Each node in the network maintains a route cache. The routes learned by a node are stored in its route cache. In order to send a packet, a node first checks the route cache for a valid route to the desired destination. If no route can be found, then a route discovery is initiated to discover a route to the destination. The normal processing at the node continues, pending the route discovery. The packets meant for the desired destination may be buffered at the sending node till such time when a route to the destination has been determined. Alternatively, the packet may be discarded and retransmitted after the path to the destination has been discovered. This eliminates 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) on the path moving out of wireless transmission range, failing or being switched off. 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 the destination.

7.4.2 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 RREQ and to prevent network flooding.

The RREQ processing is quite straightforward. When an intermediate node *N* receives a RREQ, it should first ensure that either the same RREQ or a RREQ for

Algorithm 1: DSR algorithm.

```
begin
    if source ID and request ID matches with any recent RREQ packets then
        discard the packet // Duplicate RREQ.
    end
    else
        if address of target ID matches with any recently seen RREQs then
            discard the packet // RREQ for same target from a different
                source.
        end
        else
            if target ID matches with the ID of the processing node then
                extract the route record from the request packet; // Accumulated in
                    the route record of RREQ packet
                Unicast a reply back to source by using the extracted route;
            end
            else
                // Processing node is an intermediate node.
                append address of self with the route record of RREQ packet;
                re-broadcast RREQ;
            end
        end
    end
end
```

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. So, it need not forward another copy of the RREQ. The processing of RREQ has been described in Algorithm 1. There is also a slight anomaly in sending a unicast reply. When a route reply has to be sent to the 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 RREQ packet initiated by the destination for the discovery of a route to source.

7.4.3 Route Maintenance

Since, DSR is a source initiated routing scheme, route maintenance is basically limited to monitoring link breaks along a 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 the hop-by-hop acknowledgement would require a low level (data link layer) 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 the 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 a 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 directions 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 broken link, 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.

7.4.4 Piggybacking on Route Discovery

When the links are unidirectional, the concept of piggybacking of data along with the route discovery can be used to amortize the message delivery time 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 is 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 the 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 the destination back to the source. As shown in Fig. 7.6, 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

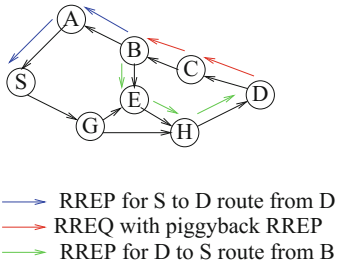


Fig. 7.6 *B* unicasting piggyback data (*D*'s RREP) to *S*

RREQ for the route from D to S, via $B \rightarrow E \rightarrow H \rightarrow D$. But piggybacked 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.

7.4.5 Handling Route Replies

Problems may arise when several mobile hosts receiving RREQ from an initiator send RREPs from their respective local route caches. Two noticeable 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 longer distance slightly ahead of time than that from a host at a shorter 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 listens 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 Fig. 7.7 where A has a cached route to destination D when

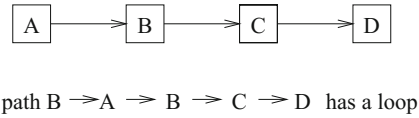


Fig. 7.7 Loop formation in using cached data for route reply

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

7.4.6 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 Fig. 7.8. Node N1 transmits a packet to some destination through the nodes N2 and N3 with N2 being the next hop and N3 being the second hop on the route. Since nodes operate in promiscuous receive mode, N3 receives the packet being sent from N1 to N2, and infers that the actual route can be shortened by eliminating the hop N2. This situation

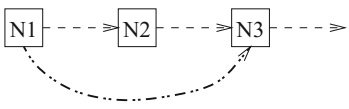


Fig. 7.8 Reflecting shorter route updates

can occur when N1 moves closer to N3. In this case N3 sends an *unsolicited* RREP to N1 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.

7.5 Ad hoc On-demand Distance Vector Routing

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 maintained by a node. 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 the 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 routing is discussed.

Each node along a route from a source to a destination maintains only the next hop entry in AODV. Therefore, the size of route table at each node is small. AODV reduces the need for system wide broadcasts by localizing the propagation of changes in the 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. A global effect is observed only 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 pushed progressively from the source towards the destination. So any node, on a path along which RREQ has been forwarded, can reach the source node by using the reverse pointers.

7.5.1 Design Decisions

Some of the design decisions in AODV are as follows.

- The routes not used are expired and discarded.
- After the primary route has expired, an alternative route is used, if one is available

- If available, an alternative route can be used to bypass a broken link on the primary route.

By expiring the unused 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, the use of alternative routes is possible, the same 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.

7.5.2 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. The sequence number of a multicast group is maintained by the leader of the group. 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 3000 ms. The precursor list is required for route maintenance when a link breaks. The routes to a destination from all the precursors of a node N 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 the following additional information:

- More than one next hops are stored.
- Each hop associated with activated flag and a direction.
- A 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 discovery and the maintenance of multicast routes are discussed.

7.5.3 Unicast Route Discovery and Maintenance

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

7.5.3.1 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 the preferred 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 `Src_seqNo` number is used to refresh the reverse routes to a source. The `Broadcast_ID` and `Src_ID` pair uniquely identify a RREQ. After forwarding a RREQ, a node stores the same in a RREQ cache for sometime. 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 it, and discard the RREQ. It prevents an RREQ from looping around.

Since AODV assumes bidirectional links, a RREQ is also used for constructing a reverse path to a source. By storing the reverse next hop, an intermediate node can unicast a received RREP to the source. The source broadcasts the RREQ and sets a time out for the reply. Timeout for the reply ensures that a node does not wait indefinitely for receiving a RREP. `Dest_seqNo` is the sequence number of the last known path to the destination.

7.5.3.2 Processing a RREQ

On receiving a RREQ, a node first checks its RREQ cache to determine if the current RREQ was an old RREQ already seen by the node. If the source ID and the broadcast ID match a RREQ present in the cache, the recipient node 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. Thus, for the 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 the source if it has an unexpired entry for the destination, and the sequence number of this entry is greater than or equal to the destination sequence number (the 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 7.2.

Lemma 7.2 *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 the destination sequence numbers and hop count. The proof is based on the following two key assumptions:

- A higher sequence number has precedence over hop count, and
- For the same sequence number, lower hop count has the precedence.

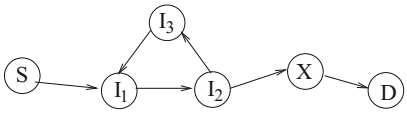
AODV forces discovery of loop-free route from a source to destination by imposing the above conditions on the sequence numbers. According these, a node v can select w as its next hop on the path to a destination D provided

- 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 a 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 Fig. 7.9. 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 the sequence number condition:

$$T_1 \leq T_2, T_2 \leq T_3, \text{ and } T_3 \leq T_2 \text{ so, } T_1 = T_2 = T_3 \tag{7.1}$$

Fig. 7.9 A possible loop



	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

Let H_i the number of hops from $I_i \rightarrow I_{i+1 \bmod 3}$. Now applying the sequence number condition with relations 7.1 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.

7.5.3.3 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 of the N/W, a new RREQ with an increased TTL is launched. The TTL value is recorded in the route table. For subsequent route discovery to the same destination, the starting TTL is set to the value in the route table. This approach progressively enlarges the 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 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:

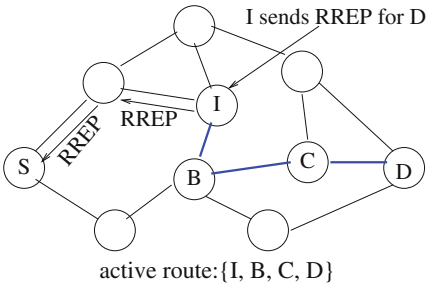


Fig. 7.10 Intermediate node unicasting a RREP

1. The node has an active route to the destination, and
2. The destination sequence number of the route currently 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 the 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 Fig. 7.10. 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.

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

- Takes the node ID from RREP to define the next hop to destination.
- Takes the ID of destination for indexing the route table.
- Adds 1 to hop count and stores the same in the entry.
- Takes the 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 ensures that if the route to destination were to pass through the current intermediate node, then the hop count would also include 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 and a route entry corresponding to destination is created. 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 Fig. 7.11. The flooding of RREQ is shown by solid lines, the arrows denote the direction of flooding. The dashed 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.

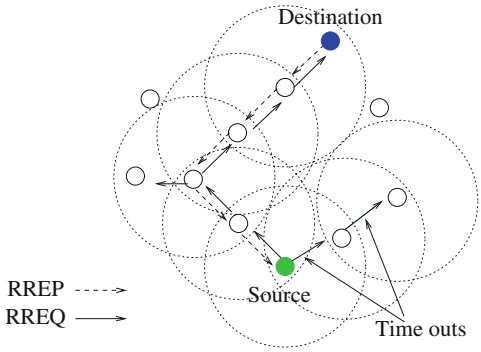


Fig. 7.11 Route discovery

7.5.3.4 Route Maintenance

A route is maintained as long as it is needed. The movements of the nodes affecting active paths are only considered. The route maintenance is responsible primarily for:

1. Detecting link breaks on active paths, and
2. Building alternative paths when node movements occur.

Routes may become invalid due to movement of nodes. For example, if the source node moves, a route discovery should be re-initiated if the path to destination is still needed. Every node must maintain its neighborhood information to detect any movement as soon as it happens.

Neighborhood information
Typically, the neighborhood information at a node is maintained by periodic broadcast of hello messages from its neighbors. If a node N has not sent anything (either a message or a hello message) 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 is 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 a hello message from a neighbor, a node updates the lifetime of that neighbor. If an entry for the neighbor does not exist, then the node creates one.

Link breaks
A link break on a route from a source to a destination is detected when the downstream 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 upstream end node detecting the break. As explained earlier, every node maintains a list of precursors for each destination. A precursor is an upstream neighbor of the node that uses the current node as the next hop for a valid route to a chosen destination. For each broken link, the upstream end node (the end node closer to the source)

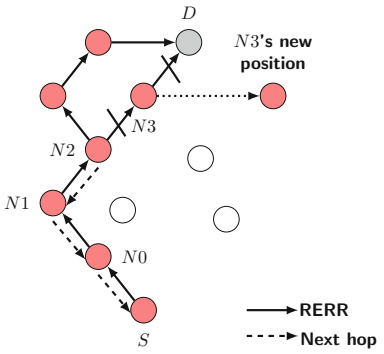


Fig. 7.12 Route maintenance

sends RERR to all the nodes in its list of precursors. These precursors are exactly those nodes which use the upstream end node as the next hop for a valid route to the destination.

A source re-initiate route discovery when it receives a RERR packet from each of its downstream neighbors has next hop entry for an active route to the destination. Figure 7.12 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 dashed arrows indicate the path that RERR traverses using precursors on the route from $N2$ to S.

7.5.4 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, the last known sequence number of the group, and a `join` flag if N is 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 it receives a reply or the number of broadcast becomes equal to `rreq_retries` after which it declare itself as the leader. Figure 7.13a 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 two group members. Then RREPs spread in the network as

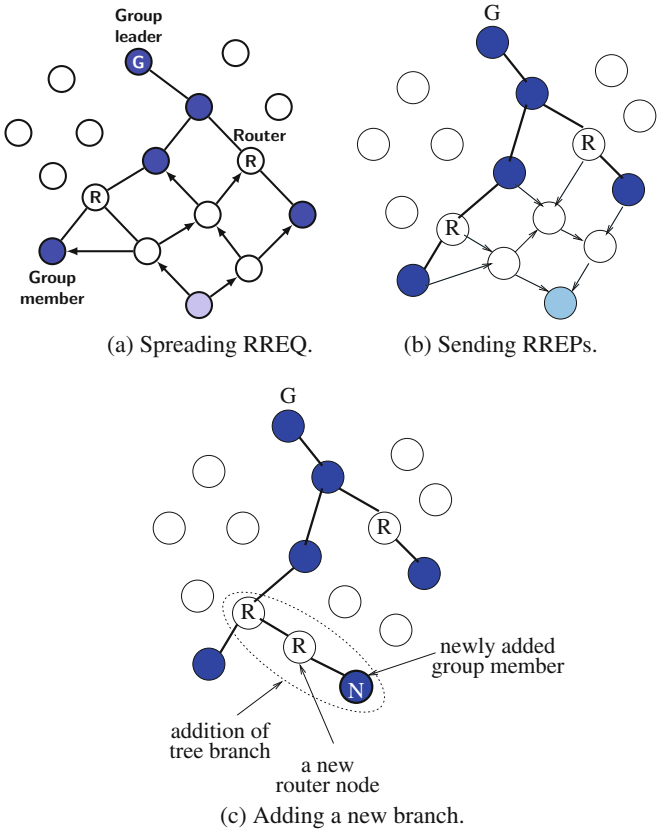


Fig. 7.13 Grafting a new branch to multicast tree

illustrated by Fig. 7.13b. Finally, Fig. 7.13c 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, the nodes in the newly grafted branch should be activated by sending an explicit *activation* message on the branch. The reasons for requirement of an explicit activation are:

- 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.
- Besides forwarding of the RREP with join request, the RREPs for non-join requests may also be forwarded in order to set up paths to multicast tree. Grafting a tree branch for non-join 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 explicit activation were not needed, potentially, all these nodes which sent RREPs for a branch can forward data packets to multicast group.

The last point is particularly important, as it may lead to inefficient use of bandwidth when all nodes involved may potentially create large amount of network traffic. Therefore, only one of possible branches should be allowed to forward data to multicast group.

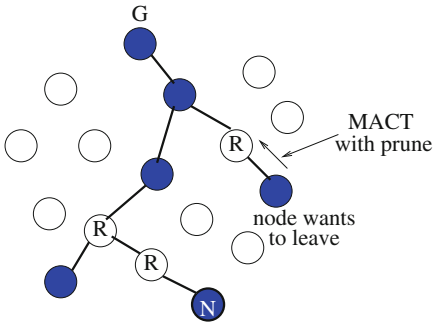
The source waits for the length of the interval for route discovery before it can use the path. At first, the source unicasts a MACT message to the node from which it received the RREP. The upstream neighbour sets `active` flag against the source in its own route table before forwarding the same to the next hop upstream towards the originator. Each node on the route sets `active` 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 *cannot* 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, a node *N* deletes the entry corresponding to the source of the current MACT. If *N* is a router and the deletion of the previous hop turns into *N* a leaf node, then *N* would initiate its pruning by sending a MACT with prune flag. But if the *N* is a group member, then it may not want to revoke its group membership. The deletion of branch using MACT is shown in Fig. 7.14.

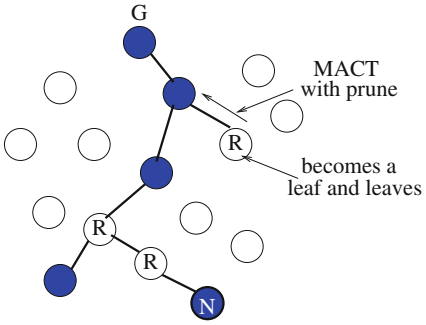
7.5.4.1 Maintenance of Multicast Routes

A unicast route is maintained only as long as it is required. However, in the case of musticast routes, a multicast tree should be maintained for the entire lifetime of the existence of the group. 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

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



(a) Mact for leaving tree.

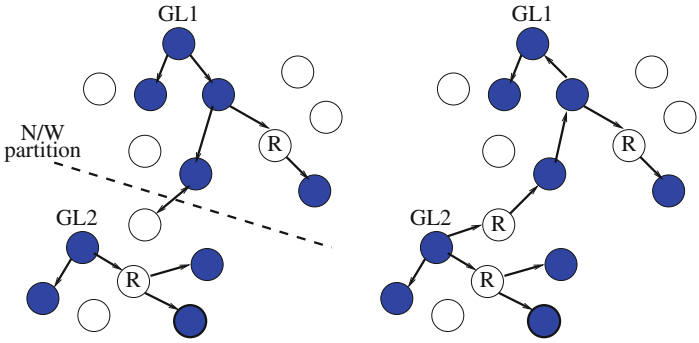


(b) Tree after pruning.

Fig. 7.14 Pruning after a member node leave the multicast group

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, as stated earlier, is 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 be a router for the multicast tree. If M becomes a leaf node after link break, it tries to detach itself. It does so, by sending a MACT with prune flag to its upstream node in the tree. However, MACT is 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 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 7.15 illustrates the route repair process. After N is able to add its subtree to multicast



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

Fig. 7.15 Repairing link breakage in AODV

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 becomes 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 leader's identity by GRPH message. GRPH is periodically broadcast from the group leader as an 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.

7.6 Zonal Routing Protocol

Zonal Routing Protocol (ZRP), proposed by Haas and Perlman [16], is a dynamic zone based hybrid routing scheme. The scope of proactive scheme is restricted to a local neighborhood of each node called its *zone*. A route to a distant node is determined by querying a subset of nodes in the network. The protocol relies on the following important observation.

- Routing can be efficient if the topology changes in network can be quickly disseminated in the local neighborhood.

The above basic idea of restricting the propagation of changes in network topology only to the local neighborhood where the changes take place is also used by fishey state routing (FSR) [12]. However, the granularity of propagation in FSR is controlled only by varying the periodicity of update propagation according to distance. FSR distributes information about topology changes over the whole network at some regular though larger time interval compared to DSDV.

ZRP combines the characteristic of creating a route on-demand as in reactive routing schemes with the property of fast convergence that is typical to proactive routing algorithms. It restricts the propagation of topology changes to the local neighborhoods of the involved nodes, and creates a route between a pair distant nodes on-demand. Only active routes between distant nodes are maintained as the topology changes. Topology changes are not propagated over the entire network.

7.6.1 Routing Zones

A node maintains routes to all destinations within its local neighborhood called its *routing zone*. In other words, a node's routing zone consists of all nodes which may be reached from it by a fixed number of hops, say 1 to 2. This fixed number, denoted by ρ, defines the routing zone is called *zone radius*. For example, in Fig. 7.16 the routing zone for s consists of nodes $\{a, b, c, e, h, i, j, k\}$ which are within two hops from s. The nodes $\{e, f, g, h, i, k\}$ are peripheral nodes of the routing zone of s. The radius ρ of a routing zone is adjustable, and usually a small constant. Since it is not possible to control or predict the presence of the number of mobile nodes in an area, the number of nodes in a routing zone may potentially be very large if ρ is not small.

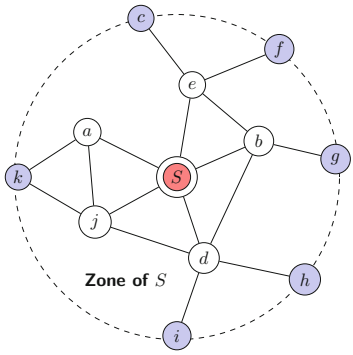


Fig. 7.16 Two-hops routing zone of S

A large routing zone would generate a large amount of update traffic. This apart, the diameter of an ad hoc network usually does not exceed 5 or 6. Consequently, the very purpose of ZRP is defeated if a large zone radius is selected. In other words, with large zone radius, ZRP no longer remains a hybrid protocol.

Each node proactively maintains information about the routes to all other nodes within its routing zone. Therefore, the updates are propagated locally. Periodic routing updates do not flood the entire network, and full dumps for routing updates are not needed. The nodes use a proactive protocol called *IntrAzone Routing Protocol* (IARP) to maintain the routing information in its routing zone. It is interesting to observe that nothing prevents a node from using a different proactive schemes from other nodes. However, the problem in this approach is that different control packets of varying length and structures will be needed to maintain intrazone routes. Usually DSDV [14] is used for IARP.

7.6.2 Interzone Routing

An *IntErzone Routing Protocol* (IERP) is used for discovery of route on-demand to the destinations beyond a source's own routing zone. Theoretically, IERP can be based on any reactive protocol. It does not matter which reactive protocol is chosen for IERP as long as that protocol is able to exploit the information about local routing zone topology maintained by IARP to guide the query for route discovery. This is achieved by delivering the route request queries from a receiving node to the peripheral nodes in its routing zone. The delivery service is called *bordercasting*. The bordercasting is illustrated by Fig. 7.17. The source labeled S has a zone radius $\rho = 2$. The nodes $\{a, b, c, d, e, h, i\}$ belong to its routing zone. If S were to send a datagram to D and does not have a valid route to D then a route discovery is initiated by S. Since, D does not belong to routing zone of S, S bordercasts a route request to the peripheral nodes $\{e, f, g, h, i\}$ of its routing zone. Each node receiving bordercast checks for availability of the destination in their respective zones before a

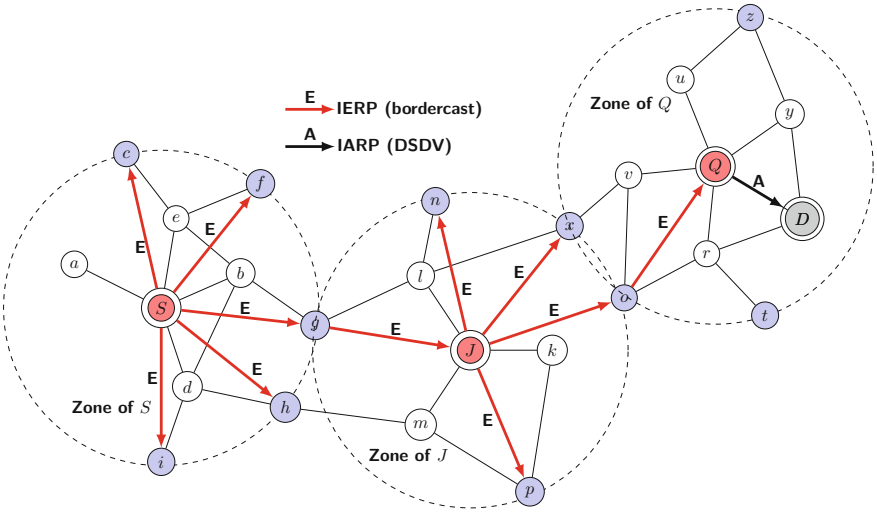


Fig. 7.17 Bordercasting and operation of IERP

repeat bordercast of the query in their own zones. So g bordercasts the request to J. Since, D does not belong to J's routing zone, J again bordercasts route discovery to n, x, o and p. Node o's bordercast finally reaches Q. Since, D belongs to routing zone of Q, the route discovery is over. The bordercasts collectively construct a tree. The branches of the tree are indicated in Fig. 7.17 by solid links labeled T. The forwarding path $S \rightarrow g \rightarrow J \rightarrow Q$, thus, gets created. The discovery process is complete when a route reply is sent back to the source. The procedure for route discovery by a source S to a destination D is summarized below:

- S first checks its own routing zone to find D. If D is found, then S sends the packet to D using local routing table of S.
- Otherwise (D does not belong to routing zone of S), S bordercast (unicasts to border nodes) RREQ to peripheral nodes of its zone.
- If any of the peripheral nodes finds D in its zone, then it responds by sending RREP to S. If a peripheral node does not have a path to D then it re-bordercasts the RREQ.

In Fig. 7.17, the dotted circles represent the routing zones of the respective nodes S, J and Q. Routing zones of g, o, and other involved nodes have not been shown as these zones are not relevant to the route. Node Q finds D in its routing zone, so it sends RREP to S. The routing zones of other nodes involved in route discovery procedure are not explicitly shown as it will clutter the figure. However, the reader can imagine how a bordercast is carried out.

7.6.3 Bordercast Tree and Query Control

A bordercast tree is defined by a root node r and the peripheral nodes of r's routing zone. The root of the tree appends complete forwarding path map in a RREQ packet. For example, node S include the path map as shown below.

Query message	
Border node	Relay node
e	c
f	c
g	b
h	d
i	d

The overhead of including forwarding path in RREQ grows linearly with the size of routing zone. Therefore, the number of hops defining a routing zone should be a small constant ρ. Bordercast effectively introduces an exponential path multiplying scheme into route discovery procedure. Therefore, though the route discovery is expected to be fast, control of flooding remains an issue.

ZRP supports a distributed construction of bordercast tree. This allows an interior node of a routing zone to participate in construction of bordercast tree. Each node proactively tracks topology of a region extending beyond its own routing zone. An interior node x of a bordercast tree constructs an entire tree ρ hop tree from the root node by proactively tracking a topology of the region consisting of $\rho + \rho - 1 = 2\rho - 1$ hops away from it. Figure 7.18 depicts extended routing zone for relay interior node c of S's routing zone. Maintaining an extended routing zone adds overhead to IARP, because router re-advertisements should include information about extended zone. However, it helps interior nodes to save on query traffic overhead in the reactive route discovery (IERP). So there is a trade-off between overhead of maintaining an extended routing zone and saving in RREQ traffic.

The main motivation of query control mechanism is to ensure that search for destination is guided from the source in all outward directions. It serves to reduce flooding, and also to guide the query reaching the destination quickly. The implementation framework is to prevent a query from re-entering a covered zone. Figure 7.19 illustrates how the query should be dispersed from a source node in all directions [16] so that it reaches targeted destination and prevented from re-entering already covered regions.

Two types of advance query detection techniques QD1 and QD2 are used for early termination of route discovery queries [16]. QD1 is concerned with direct relay of bordercast messages. When an interior node relays a bordercast message, it prevents messages flowing over downstream branches leading to peripheral nodes inside the covered region of the network. As Fig. 7.18 demonstrates, node c has full knowledge of S's routing zone. Therefore, c can terminate re-bordercast of S's query back to its routing zone from a node belonging to the uncovered region in c's extended

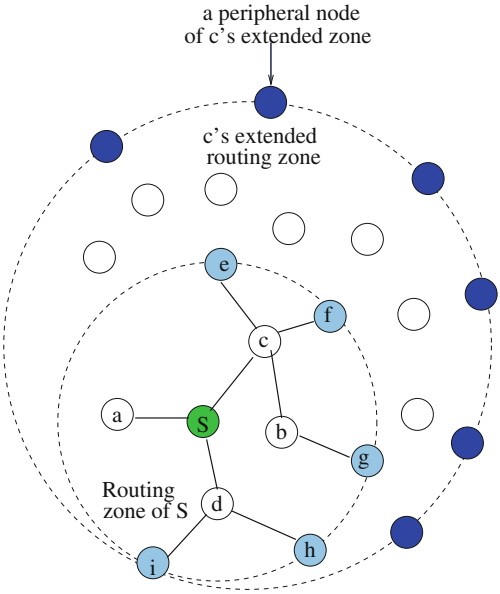


Fig. 7.18 Extended routing zone of node *c*

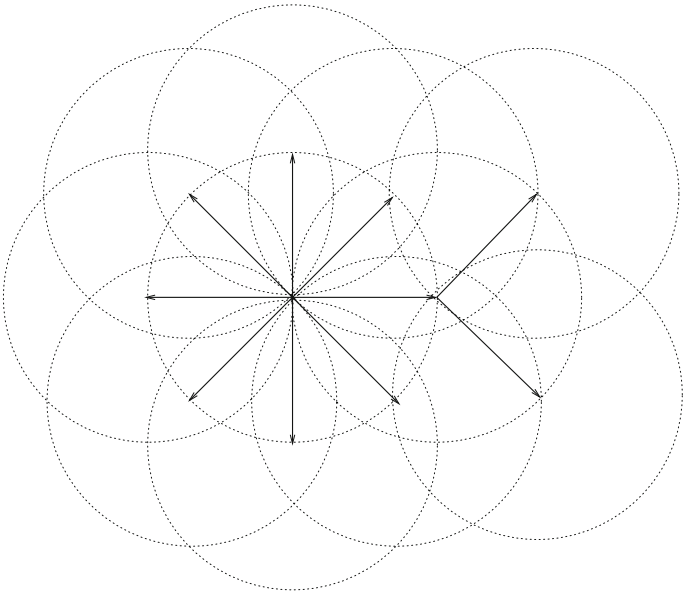


Fig. 7.19 Directing route discovery query from a source

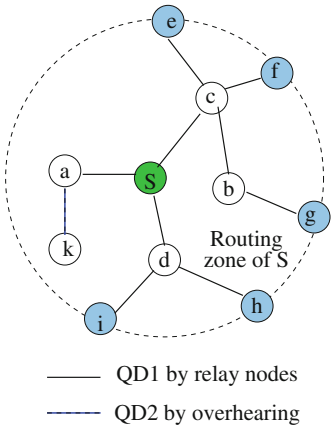


Fig. 7.20 Detection of route discovery query

zone. The query detection technique QD2 is used by the nodes of a routing zone which have overheard a route discovery query. These nodes can terminate the query if a re-bordercast is received directly from any node in the uncovered region. In a single channel transmission, QD2 can be detected by any node in transmission range of a relaying node. The capability for QD2 can be implemented through IP and MAC layer broadcasts. For example, as illustrated in Fig. 7.20 node *k* in *S*'s routing zone overhears query from *a*, and participates in preventing the same query from re-entering *S*'s region. The other nodes in *S* use QD1 as all of these node participate in forwarding the query.

7.6.4 Random Delay in Query Processing

Random query processing delay introduces asynchronicity in forwarding queries. The idea behind the approach is to ensure that the same query does not reach a node simultaneously by different paths. Each node waits for a random time before it constructs the bordercast tree. As the nodes waits to send the query, it could also detect queries from other bordercasting nodes and prune the overlapping branches in the bordercast tree. For example, as shown in Fig. 7.21 nodes *a* and *b* both receive query at the same time. If they both rebroadcast the query simultaneously, they will later come to know that both were spreading the same query in overlapping regions and wasting bandwidth. But using the random delay approach, *b* withholds the query, while *a* schedules its query much in advance. Furthermore, *a* uses QD1 to know about the same query from *b* when it launches its own query. Therefore, *a* can prune its downstream branches belonging to *b*'s zone.

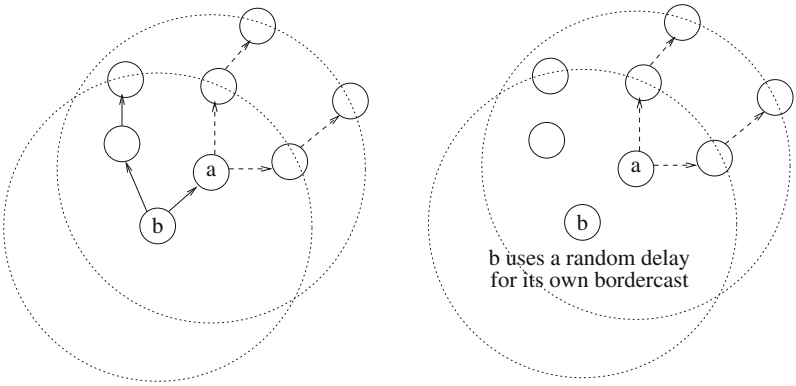


Fig. 7.21 Random delay in query processing

7.6.5 Route Caching

Active routes are cached at intermediate nodes. The route cache can be used to reduce frequency of route discovery. When the nodes on an active route move, that route becames invalid. If the link breaks occur upstream towards the destination then instead of initiating a fresh discovery of route, it will be less expensive to repair the existing route locally with the help of route cache. A local repair works by patching the invalid route to the destination from the nearest upstream node from the first link break. Thus the local repair constructs a bypass to avoid the first broken link on the existing path. In theory a route may be patched up many times. But in practice, the route may deviate appreciably from a shortest route after one or two such patchings. Therefore, after a few local repairs, the source may decide to initiate a fresh route discovery. This scheme follows the approach similar to that we found in use of forward pointer for location update schemes described earlier.

References

1. T.W. Chen, M. Gerla, Global state routing: a new routing scheme for ad-hoc wireless networks, *1998 IEEE International Conference on Communication* (1998), pp. 171–175
2. C.-C. Chiang, W. Hsiao-Kuang, W. Liu, M. Gerla, Routing in clustered multihop, mobile wireless networks with fading channel. IEEE SICON **97**, 197–211 (1997)
3. T. Corman, C. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms* (MIT Press, Cambridge MA, USA, 2001)
4. S. Corson, J. Macker, Mobile ad hoc networking (MANET): routing protocol performance issues and evaluation considerations, http://www.ietf.org/rfc2501. January 1999. RFC-2501
5. R. Dube, C. Rais, W. Kuang-Yeh, S. Tripathi, Signal stability-based adaptive routing (SSA) for ad hoc mobile networks. IEEE Pers. Commun. **4**(1), 36–45 (1997)
6. E. Gafni, D. Bertsekas, Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Trans. Commun. **29**(1), 11–18 (1981)

7. L. Ji, M. Corson, A lightweight adaptive multicast algorithm, *Globecom'98* (1998), pp. 1036–1042
8. D.B. Johnson, D.A. Maltz, DSR the dynamic source routing protocol for multihop wireless ad hoc networks, ed. by C.E. Perkins. *Ad Hoc Networking*, Chap. 5 (Addison-Wesley, 2001), pp. 139–172
9. Y.B. Ko, N.H. Vaidya, Location-aided routing (LAR) in mobile ad hoc networks. Wireless Netw. **6**(4), 307–321 (2000)
10. S. Murthy, J.J. Garcia-Luna-Aceves, An efficient routing protocol for wireless networks. ACM Mobile New. Appl. J. 183–197 (1996)
11. V.D. Park, M.S. Corson, A highly adaptive distributed routing algorithm for mobile wireless networks, *IEEE INFOCOM'97* (1997)
12. G. Pei, M. Gerla, T.W. Chen, Fisheye state routing in mobile ad hoc networks, *ICDCS Workshop on Wireless Networks and Mobile, Computing* (2000), pp. D71–D78
13. C.E. Perkins (ed.), *Ad Hoc Networking* (Addison-Wesley, 2001)
14. C.E. Perkins, P. Bhagawat, Highly dynamic destination-sequenced distance-vector (DSDV) algorithm for mobile computers. Comput. Commun. Rev. 234–244 (1994)
15. C.E. Perkins, E.M. Royer, Ad-hoc on-demand distance vector routing, *The 2nd IEEE Workshop on Mobile Computing Systems and Applications* (1994), pp. 90–100
16. M.R. Perlman, Z.J. Haas, Determining the optimal configuration of the zone routing protocol. IEEE J. Sel. Areas Commun. **17**(8), 61–81 (1999)
17. E.M. Royer, C.K. Toh, A review of current routing protocols for ad-hoc mobile wireless networks. IEEE Mag. Pers. Commun. **17**(8), 46–55 (1999)
18. C.K. Toh, Associativity-based routing for ad-hoc mobile networks. Wireless Pers. Commun. **4**, 103–139 (1997)