Part I

Mobile Data and Intelligence

Chapter 1

A Survey of State-of-the-Art Routing Protocols for Mobile Ad Hoc Networks

Amitava Datta and Subbiah Soundaralakshmi

School of Computer Science and Software Engineering, University of Western Australia, Perth, WA, Australia

1.1 INTRODUCTION

Mobile ad hoc networks (MANETs) have opened up many new possibilities in using computer networks in improvised scenarios where traditional networking infrastructure is unavailable. Mobile networks are usually based on wireless communication and hence many of the established technologies developed for wired networks are not directly applicable in mobile networks. In particular, nodes forming a mobile network are not tied to any infrastructure and can form a network on the fly and for a short

Mobile Intelligence. Edited by Laurence T. Yang, Agustinus Borgy Waluyo, Jianhua Ma, Ling Tan, and Bala Srinivasan

period of time. The main difficulties in forming a mobile network are the mobility of the nodes, the nature of the wireless medium, the energy constraints of small mobile nodes, and the possibility that nodes may join or leave a network anytime during the lifetime of the network.

Since nodes are mobile, the routes in the network usually have a short life span. A route may or may not exist for the entire duration of a data communication session, unlike in wired networks, where nodes are usually present in fixed geographical positions. The wireless medium has the constraint that any communication by a node is done through broadcasting of a packet. Since the bandwidth in the wireless medium is much less compared to wired networks, this poses the problem that almost all communications take up large amount of bandwidth due to the flooding of packets. Moreover, the contention for limited bandwidth results in packet collisions and retransmissions, further wasting the available bandwidth. The nodes in a mobile ad hoc network are usually powered by batteries that may not be rechargeable during a network session. This imposes the added constraint that nodes should not waste their energy in retransmitting packets that have been lost due to collision. Since nodes spend almost equal amount of energy in transmitting and receiving packets, even overhearing packets destined for other nodes results in wastage of energy.

Routing of packets is one of the most basic activities in any computer network, wired or wireless. All applications in a MANET depend on reliable and efficient routing of packets. Hence, it is extremely important to design routing protocols that can work within the constraints of a mobile ad hoc network and provide support for all higher level applications. It is not surprising that tremendous amount of research effort has been invested in designing efficient routing protocols for MANETs during the past 10 years. The MANET working group within the Internet Engineering Task Force (IETF) [8] is considering several of these protocols for standardization. However, the task of the MANET working group is considerably difficult due to the existence of many different routing protocols in the literature. Currently, there are four protocols under consideration by the working group and the Internet drafts for these protocols are available for comments.

Our aim in this chapter is to provide a concise yet comprehensive view of many different routing protocols proposed for MANETs. We will pay special attention to the protocols under consideration by the MANET working group as naturally these protocols are some of the most efficient and reliable routing protocols proposed until now. However, we will also discuss the evolution of routing protocols from a historical point of view and discuss other protocols that have significantly different yet interesting ideas. More details about many of the protocols discussed in this chapter can be found in the paper by Belding-Royer and Toh [19] and the book edited by Perkins [15].

The rest of the chapter is organized as follows. We discuss a taxonomy of routing protocols in Section 1.2. We discuss the class of proactive routing protocols in Section 1.3 and reactive routing protocols in Section 1.4. We discuss some other classes of protocols in Section 1.5. Finally, we conclude with some comments in Section 1.6.

1.2 A TAXONOMY OF MANET ROUTING PROTOCOLS

The main aims of all routing protocols designed for MANETs are to achieve a high level of performance, in terms of high throughput, low latency, and low energy expenditure by individual nodes. However, these aims are quite often contradictory in the sense that a routing protocol might have to sacrifice one of them in order to satisfy another. For example, assume that we are trying to design a routing protocol that aims for low latency in packet delivery. This may be a quality of service (QoS) requirement in a network that delivers multimedia content from one node to another. If individual nodes want to deliver or forward packets very fast toward a destination node, they must have a very clear idea about the network topology and the routes to the destination should be as accurate as possible. However, the collection of accurate topology information requires exchange of local views of topology among the nodes. In other words, each node should inform other nodes about its neighbors frequently so that all nodes have up-to-date information about the network topology. This type of information exchange is done through sending *control messages* (this name is used to differentiate from the actual data packets) and requires the nodes to spend substantial amount of energy. Hence, a protocol may have to sacrifice battery power in order to achieve low latency.

All routing protocols implicitly assume that nodes in a MANET cooperate with each other in delivering packets. Nodes in a MANET can be classified into three categories from the point of view of a packet. A node may be a sender, receiver, or a forwarding node for the packet. A forwarding node tries its best to send a packet toward its destination. The question of security in MANETs is beyond the scope of this chapter and we will assume that all nodes in the network are trusted. We refer the reader to the paper by Pirzada et al. [18] and the references therein for more discussion on security in MANETs.

The main routing protocols for MANET can be classified into two categories, *proactive* and *reactive*, depending on how a protocol collects information about the topology of the network. Proactive protocols try to reduce latency in packet delivery by aggressively disseminating topology information throughout the network. This, however, has a detrimental effect that much of the available bandwidth in the wireless medium is used for sending control messages. Hence, a challenging problem in designing a proactive protocol is to reduce the effect of control messages in the network while still achieving an acceptable level of latency. We will discuss this issue in more detail in the next section.

Reactive protocols try to minimize the wastage of bandwidth by reducing the amount of control messages in the network. They try to find routes on-demand and do not depend on proactive collection of topology information for finding routes. However, this approach quite often increases latency in packet delivery. A challenging problem in designing reactive protocols is to reduce latency while maintaining the low volume of control messages. Mobile nodes executing reactive protocols quite often resort to indirect means such as overhearing passing wireless traffic to improve their knowledge of network topology. We will discuss reactive protocols in depth in Section 1.4.

There are other protocols that combine the advantages of both proactive and reactive protocols while eliminating their disadvantages. These protocols use a proactive protocol within a small neighborhood of each node so that the volume of control messages remain manageable and use a reactive protocol over the entire network. The zone routing protocol (ZRP) is the most notable among these protocols, and we will discuss this protocol briefly in Section 1.5.

We will also discuss an important class of protocols in Section 1.5, called link reversal protocols. These protocols are based on an elegant idea of maintaining a rooted directed acyclic graph (DAG) in a MANET. The main thrust in these protocols is to maintain the DAG (and hence routes); however, usually the overhead in these protocols is quite high.

1.3 PROACTIVE ROUTING PROTOCOLS

Proactive routing protocols try to collect as much information about the MANET as possible through proactive exchange of messages about their local topology. One of the earliest protocols for MANETs was the destination sequenced distance vector (DSDV) protocol [16], which is one of the best known proactive routing protocols. The DSDV protocol has a large overhead of control messages and hence it fell out of favor due to the emergence of more efficient reactive protocols such as the dynamic source routing (DSR) and the ad hoc on-demand distance vector (AODV) protocol. However, the low latency in packet delivery is one of the most attractive aspects of the proactive protocols. Considerable work has been done in recent years to reduce the overhead of the DSDV protocol and one of the most promising proactive protocols called the optimal link state routing (OLSR) protocol [13] is currently under consideration by the MANET working group. In this section, we will first discuss the DSDV protocol and then the OLSR protocol.

1.3.1 The Destination Sequenced Distance Vector Protocol

We should first list a few points about MANET routing protocols that are applicable for all the protocols discussed in this chapter. A routing protocol is a distributed algorithm executed by each node in a MANET. In other words, each node executes a local copy of the protocol on the data that they collect locally. Moreover, this distributed execution of a protocol aims to achieve some global performance goals such as high throughput, low latency, and minimizing the overall expenditure of energy. In the following, we will use the terms *packets* and *messages* to mean packets sent by nodes in the wireless medium.

DSDV is a table-driven protocol, in the sense that all routing decisions are taken by individual nodes based on their local routing tables. There are two parts in the DSDV protocol, namely, keeping the local routing tables as up-to-date as possible and computing routes with the help of the local routing tables. We will first discuss the second part as it will be clear how nodes find the best routes by

executing the DSDV protocol. We will then discuss how nodes update their routing tables.

First, we assume that each node has collected up-to-date information about the topology of the network in its routing table. Given a source node *S* and a destination node *D*, the purpose of a routing table is to find a best path according to some metric between *S* and *D*. In a MANET, intermediate nodes (i.e., nodes other than *S* and *D*) forward the packets that they receive from *S* toward *D*. Hence, the best path from the point of view of an intermediate node *I* is a path starting at *I* with *D* as the destination.

We will take an abstract graph theoretical view of the routing table, but this view can be applied to any other representation of the table without much modification. We consider the nodes in the MANET as nodes of our graph. There is a link or edge between two nodes if they are within the transmission range of each other. In practice, links between neighboring nodes in a MANET may not be bidirectional as the transmission ranges of both the neighbors may not be equal. We will take the simplified view that links are always bidirectional to keep the descriptions of the protocols simpler. However, our descriptions can be modified for the case when the links are unidirectional. We can now view the routing table as an adjacency matrix of the graph representing the underlying MANET. We can indicate an edge between nodes *i* and *j* by a 1 in the entry at the intersection of the *i*th row and the *j*th column. Similarly, a 0 indicates the absence of an edge.

It is now possible to find shortest paths from this adjacency matrix by running Dijkstra's shortest path algorithm. Suppose the source node (every node is potentially a source node for packets) or an intermediate node *I* has to find a best path for a packet to the destination *D*. The node runs Dijkstra's shortest path algorithm on its routing table and finds the shortest path to *D*. This shortest path must pass through one of its neighbors. The task is then to forward the packet to this neighbor. The neighbor in turn runs Dijkstra's shortest path algorithm on its own routing table to find a best path to the destination and repeats the process of forwarding the packet. We have assumed till now that hop count is the only metric for finding best paths using Dijkstra's algorithm; however, we can use several other metrics that are relevant for the wireless medium. Some other metrics could be the bandwidth as a cost on a link, the possible delay of a link, and so on. The choice of shortest path is illustrated in Figure 1.1.

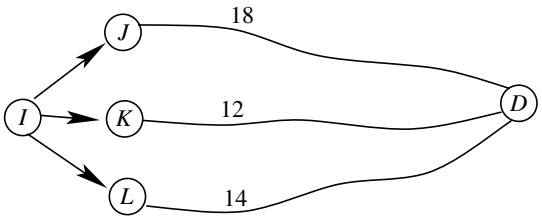

Figure 1.1 Illustration for selection of a forwarding node in the DSDV protocol. Node *I* finds three different paths to the destination *D* by running Dijkstra's shortest path algorithm on its routing table. *I* chooses neighbor *K* as the next hop of the packet since the cost of the path to *D* through *K* is least. The node IDs are shown inside the nodes and cost of the paths from *J, K*, and *L* are shown on the paths.

We now turn our attention to the other part of the DSDV protocol, namely, collection of topology information. If we expect nodes to find best paths by executing Dijkstra's shortest path algorithm, the routing table at each node should be as up-to-date as possible. Note that we can possibly never have the situation that all the routing tables in all the nodes of the MANET contain correct information. In other words, all the links that are recorded as links between neighbors may not exist at any time during the execution of the protocol due to the mobility of the nodes. Consider three nodes i, j, and k in a MANET. The node i may have noted the information that there is a link between j and k. However, this information is usually relatively old and by the time i uses this information to compute shortest paths, the link between j and k might have already broken due to the mobility of these two nodes.

This is a very important point to note as it differentiates a centralized algorithm from a distributed one; in particular, centralized algorithms usually have complete input before the execution starts. A distributed algorithm has only a partial view of the input at each node and the node executing the algorithm has to take decisions using this partial input. Moreover, routing protocols in a MANET do not even have a correct partial input due to the mobility of the nodes. Hence, any routing protocol in a MANET is not an exact algorithm in the traditional sense. It is in a sense an algorithm that tries to achieve best results using incomplete and partially incorrect inputs.

The nodes executing the DSDV protocol exchange their routing tables to update their knowledge of the network topology. Consider a node i and its routing table. Suppose i currently has three neighbors j, k, and l. If any of these three neighbors, say j, moves out of the transmission range of i, there is a change in the routing table of i. This type of change triggers a broadcast of the routing table from i to all the other nodes in the network, so that all the other nodes can update the routing tables with the changed topology. The broadcast is done by i sending its routing table to all its current neighbors and the neighbors sending it to their neighbors, and so on. If a node m receives such a broadcast from another node i, that may change the routing table of m, triggering a broadcast from m.

This is clearly an expensive process as all nodes in a MANET need to broadcast their routing tables due to changes in their local topology triggered by the mobility of the nodes. A large part of the bandwidth may be consumed by these update packets, especially in high-mobility scenarios. Several suggestions were made in the original proposal of DSDV to alleviate this overhead by sending incremental updates of routing tables instead of full updates. The idea of an incremental update is to broadcast a part of the routing table that has changed instead of sending the whole routing table. However, the fact remains that each update floods the entire network and the protocol becomes too inefficient in terms of throughput of data packets even in moderate-mobility scenarios.

We conclude the discussion of the DSDV protocol by mentioning another of its important features, assignment of sequence numbers to packets. One of the effects of broadcasting a packet, say p, from one node i to all other nodes in a MANET is that another node j gets multiple copies of p. Moreover, if i has sent two different updates of its routing table at two different instances, there is no guarantee that these two packets will arrive at j in the correct order, that is, the packet sent earlier may

reach later. Since DSDV depends on the correct topology information for routing, it is very important that nodes use most recent information for updating their routing tables. One way of ensuring that nodes use most recent information is to time-stamp each packet with the current time. That way if *j* receives two packets from *i*, it can decide which one is more recent. However, this scheme works only when all nodes maintain synchronized clocks. Clock synchronization is a difficult task in a distributed system, and in particular in a MANET, since we do not expect the nodes have access to any infrastructure. Another alternative is to use sequence numbers that work as logical clocks. Each node stamps the packets that it broadcasts with an increasing integer called a sequence number. Any node *j* receiving two packets from a node *i* can decide which packet is more recent by comparing the sequence numbers in the packets.

1.3.2 The Optimized Link State Routing Protocol

Although the DSDV protocol is attractive due its low latency in finding routes, it has a very high overhead of control packets due to the broadcasting of the routing tables. Any broadcast floods the entire network and takes up a large portion of the available bandwidth in the wireless medium. The OLSR protocol is a relatively recent attempt to reduce the control overhead of the DSDV protocol in order to increase the throughput of packet delivery. The OLSR protocol tries to improve the DSDV protocol in two ways, by reducing the size of the updates and by reducing the effect of the broadcasts. We discuss these two improvements below.

Nodes executing the OLSR protocol broadcast link states rather than routing tables. Suppose a node *i* currently has three neighbors *j*, *k*, and *l*. If at least one of the neighbors, say *k*, moves out of the transmission range of *i*, there is a change in topology and *i* should inform other nodes about it through a broadcast. Note that it is sufficient for *i* to inform that the link *i*–*k* has broken and other nodes can update their routing tables with this information. Hence, broadcasting link state information instead of routing tables is better to keep the volume of control messages low.

The main improvement in the OLSR protocol, however, comes from reducing the effect of broadcasting of packets. Note that, for a node *i*, all its neighbors receive any packet broadcast by *i* due to the fact that wireless transmission is omnidirectional. However, there is no need for all the neighbors of *i* to rebroadcast the packet again. It is desirable to choose only a subset of neighbors of *i* to rebroadcast the packet. The OLSR protocol uses a subset of neighbors called *multipoint relays* to broadcast its packets further. Strictly speaking, the concept of multipoint relays is not a part of the OLSR protocol. Rather, it is a concept used for reducing the volume of broadcast packets in wireless networks. We can illustrate the use of multipoint relays through an example. Consider again a node *i* and its three neighbors *j*, *k*, and *l*. These three neighbors are called one-hop neighbors of *i*. A two-hop neighbor can be reached from *i* in two transmissions. Suppose *i* has six two-hop neighbors *a, b, c, d, e*, and *f*. Clearly, each of these two-hop neighbors has at least one of the one-hop neighbors

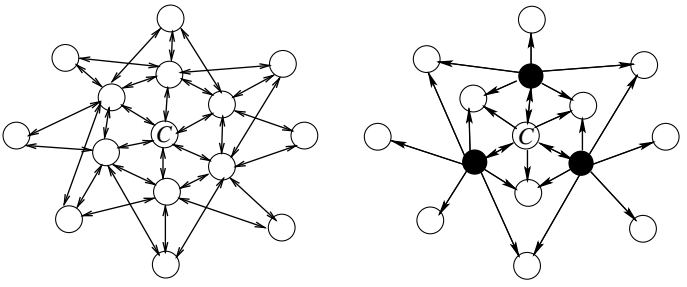

Figure 1.2 Illustration for simple flooding and flooding through multipoint relays. (a) This figure shows simple flooding. Every node that receives the packet initially sent by the central node *C* broadcasts it again. A double arrow between a pair of nodes indicates the receipt of broadcasts from each other (we have assumed symmetric communication links). (b) The number of packets has been reduced considerably by using the multipoint relay nodes (dark nodes). These nodes are one-hop neighbors of *C* and collectively neighbors of all the two-hop neighbors of *C*. The total number of packets is reduced considerably as the one-hop neighbors that are not multipoint relays do not broadcast.

of *i* as a neighbor (otherwise, they cannot be two-hop neighbors of *i*). As we have mentioned above, the multipoint relays for node *i*, denoted by *MPR(i)*, are a subset of one-hop neighbors of *i* such that all the two-hop neighbors of *i* are neighbors of the nodes in the subset *MPR(i)*. To illustrate this, suppose *k* and *l* collectively are neighbors of *a, b, c, d, e*, and *f*. In that case, we can choose *k* and *l* as the members of *MPR(i)* so that only these two nodes forward the packets broadcast by *i*. There is no need for *j* to broadcast the packets from *i*. These MPR subsets are chosen by all the nodes in the MANET by keeping track of their one-hop and two-hop neighbors. It has been shown that the broadcast overhead can be significantly reduced by using the MPR sets. The process of broadcasting through multipoint relays is illustrated in Figure 1.2.

The OLSR protocol is currently under consideration by the MANET working group of IETF and it has the desirable properties of high throughput and low latency.

1.4 REACTIVE ROUTING PROTOCOLS

The main design aim of reactive routing protocols is to reduce the control packet overhead of proactive protocols. These protocols do not maintain routing tables proactively and, as a result, cannot find routes as soon as they are required. There is usually a delay or latency in finding routes. This results in high throughput in packet delivery as the available bandwidth is utilized for delivery of data packets rather than regular flooding of control packets as in the proactive protocols. However, reactive protocols also need flooding or broadcasting of packets, which occurs on-demand. We will discuss below two of the most important reactive protocols, namely, the DSR and the AODV protocol.

1.4.1 The Dynamic Source Routing Protocol

The DSR protocol [9, 10] has two distinct phases, *route discovery* and *route maintenance*. The route discovery phase starts when a source node, say *S*, wants to find a route to a destination node *D*. Once a route to *D* has been found, the route maintenance phase starts while *S* transfers its data packets to *D* using the discovered route. We discuss these two phases in detail below.

A source node *S* starts route discovery by sending a *route request* (RREQ) packet to its neighbors. A RREQ packet has an identifier that includes a source node, a destination node, and a sequential ID that is an integer. If an intermediate node *I* receives a RREQ packet and it does not know a route to the destination node *D*, it takes one of the two actions. If it is a new RREQ packet (i.e., *I* has not seen this RREQ before), *I* broadcasts the packet to its neighbors after attaching its ID on the header of the RREQ packet. *I* also stores this packet in a list so that it can compare the identifier of this packet with future RREQ packets. If it is an old RREQ packet, that is, *I* has already received this packet in the past, *I* simply drops the packet.

If *I* knows a route to the destination *D* (we will discuss below how *I* may know a route), it initiates a *route reply* (RREP) packet by attaching the route *I* to *D* and the route from *S* to *I* in the header of the RREP packet and sending the packet back to the neighbor from whom it received the corresponding RREQ packet. Note that the RREP packet has the accumulated route from *S* to *I* in its header now. Hence, any intermediate node that receives such a RREP packet knows exactly the neighbor to whom it should send the RREP packet back. Eventually, the source node *S* receives the RREP packet and the route discovery phase ends. If no intermediate node knows a route to *D*, the RREQ packet reaches the destination *D* (provided the destination is in the same connected part of the network as *S*) and *D* sends the RREP packet. If the source node *S* does not receive a route reply within a specified period of time, it can initiate a new RREQ packet after assigning a new ID to the packet. The route discovery process in the DSR protocol is illustrated in Figure 1.3.

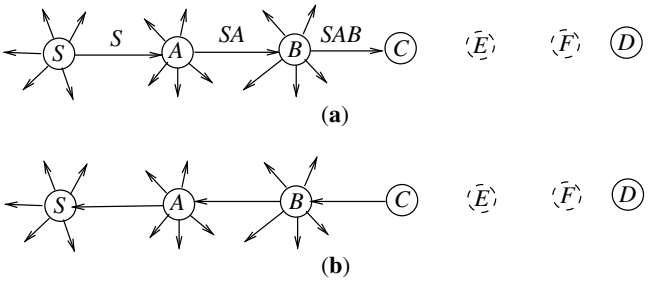

Figure 1.3 Illustration for the route discovery phase in the DSR protocol. (a) *S* starts the route discovery by broadcasting a RREQ packet. The arrows surrounding a node indicate omnidirectional broadcast. Every node appends its ID to the source route in the header of the RREQ packet. Node *C* has a route to destination *D* through the nodes *E* and *F*. (b) *C* sends a RREP packet back to *S*. Each node determines the next hop of the RREP packet by examining the source route.

One of the most important aspects of the route discovery process is the attaching of the IDs of all intermediate nodes in the header of the RREQ packet as well as the RREP packet. This list of IDs is sometimes called the *source route*. Recall that nodes executing the DSR protocol do not maintain routing tables, and hence a mechanism is needed through which a node can decide where to forward a packet when it receives a RREQ or RREP packet. The case for a RREQ packet is easier as it needs to be broadcast to all the neighbors. However, the purpose of a RREP packet is to deliver the packet to the source node *S* that initiated the route discovery. Hence, every intermediate node that receives a RREP packet should know the previous node (toward *S*) that originally sent it the corresponding RREQ packet. The purpose of attaching the source route to the header of a RREP packet is to provide this information. Moreover, RREQ packets also need to carry the source route as the node that initiates the RREP packet needs to copy this information from the corresponding RREQ packet.

Once *S* has received a RREP packet, it has a route to *D* and also knows the IDs of all the nodes along this route. *S* can now start the transfer of data packets using this route. *S* attaches the entire source route in the header of each data packet so that any intermediate node can correctly forward the packet to one of its neighbors along the source route. It is very important to maintain a route discovered in the route discovery phase. A route may break in the middle of data transmission due to the mobility of the nodes. For example, suppose a link in the route is *G–H*, where *G* and *H* are the end nodes of the link. If now *H* moves out of the transmission range of *G*, the entire route from *S* to *D* will be broken. The node *G* in this case sends a *route error* (RERR) packet to *S* by using the source route in the header. A RERR packet is almost similar to a RREP packet except that the purpose now is to inform *S* that the route it was using has broken. *S* has now two choices, either to start using an alternate route if it has one or start the route discovery phase again to find a new route.

Till now we have mentioned that nodes executing the DSR protocol do not use a routing table. However, in practice, running route discovery phase again and again is too expensive in a MANET as each route discovery is essentially a flooding of the entire network using RREQ packets. Hence, DSR uses a data structure called a *route cache* to reduce the effect of flooding. The purpose of a route cache is to store any information that a node can gather either from the packets that it receives or forwards or from the passing traffic. In particular, nodes may utilize the promiscuous mode operation allowed in IEEE 802.11 standards, where a node can overhear traffic that is not intended for it. The working of a route cache can be explained through a simple example. Consider a node *I* and a destination *D*. Suppose *I* worked as an intermediate node for forwarding data packets to *D* for a source node *P*. Assume also that the path from *I* to *D* is through some other intermediate nodes *E, F*, and *G*. Now *I* stores this route fragment *I–E–F–G–D* in its route cache. If *I* receives a RREQ packet for the destination *D* from some other source *S* in future, it can use this cached route for sending a RREP packet to *S*.

However, cached route may not always provide correct information due to the mobility of the nodes in a MANET. For example, in the example above, suppose node *G* has moved out of the transmission range of *F* and the link *F–G* has broken as a result. However, *I* is not aware of this link breakage. If *I* now sends a RREP in

reply to the RREQ from *S*, it will report a route that has already broken. This causes a problem since this will result in further route requests from *S*. A better policy is to time out old entries from the cache as most probably old entries are invalid due to the mobility in the network. The ns-2 simulator [12] adopts this policy and times out all cache entries that are older than 20 s.

The DSR protocol employs several other optimizations in order to reduce the control packet overhead and improve throughput. We discuss here only two of these optimizations due to space constraint. The first optimization is called *packet salvaging*. As we have mentioned, an intermediate node that detects link breakage in a route sends a RERR packet to the source node of that route. In addition to that, such an intermediate node tries to deliver the packet by finding an alternate route to the destination from its route cache. In other words, it tries to salvage the data packet by sending it to the destination through an alternate route. Packet salvaging reduces the possibility of loss of data packets in nodes that have experienced link failures with their neighbors while forwarding packets for other nodes. Since the source node continues to send packets until it receives a RERR packet informing the breakage of a route, the buffers in the intermediate nodes may overflow when there is a large amount of traffic in the network.

Another important optimization is related to sending RREP packets in reply to route requests. In certain situations, the route caches in the nodes may be quite up-to-date and many different nodes may be in a situation to send route replies. However, the network may get flooded due to many route replies. Hence, DSR uses a strategy where a node waits for a random period of time before sending a route reply. If it overhears that another node has already sent a route reply for the same route request, or the source node has already started using an alternate route, it does not send a route reply. This optimization reduces the overhead in the network considerably.

DSR is one of the most efficient protocols in terms of throughput even in high-mobility scenarios. Moreover, DSR almost always finds shortest paths through its route discovery mechanism. However, one of the drawbacks of the DSR protocol is the use of source routes in every packet. The number of IDs in a source route increases as the lengths of the routes increase. Since the wireless medium usually supports relatively small packet size, it is not possible to keep an entire source route in a single packet if a route is long. On the other hand, there is no guarantee of delivery of packets in the correct sequence in the wireless medium. Hence, the problem cannot be solved by splitting source routes in multiple packets. Hence, DSR is a very efficient protocol for MANETs that are relatively small and when the routes are up to about 10 hops long.

1.4.2 The Ad Hoc On-Demand Distance Vector Protocol

The AODV protocol [1, 17] tries to remove the main drawback of the DSR protocol by eliminating source routes from its control and data packets. Moreover, AODV was the first protocol to support multicasting in a MANET. Till now we have discussed

routing from the perspective of unicasting, that is, one source node sending packets to one destination node. However, this is only one of several types of communication used in a typical network. In multicasting, a source node may want to send the same packet to several destination nodes (called a *multicast group*). Broadcasting is a special case of multicasting, when a source node wants to send the same packet to all the nodes in the network. It is possible to support multicasting through finding unicast (or one-to-one) routes from the source to all the nodes in a multicast group. However, discovering and maintaining separate routes to multiple nodes is usually more expensive. It is desirable to have a common routing mechanism for all members in a multicast group. AODV solves this problem in an elegant way, as we will discuss later in this section.

AODV is in a sense a table-driven as well as a reactive protocol. Each node executing AODV maintains a routing table; however, this routing table is local and contains information only about the neighboring nodes. In contrast to DSDV, there is no need to update the routing tables through global broadcasting. AODV also has two phases, route discovery and route maintenance, like DSR. The route discovery phase is almost similar to that of DSR. The only two differences are that AODV does not use source routes and each RREQ packet is stamped with a sequence number by the source node of the packet. A source node S uses four different fields in the RREQ packet, namely, its own IP address, the IP address of the destination, its own sequence number, and the last known sequence number of the destination D. S may have obtained the sequence number of D from a packet that it had forwarded in the past; however, S does not have a route to D at present. The purpose of including the last known sequence number of D in the packet is to inform other nodes about the quality of the information that S has about D. Suppose the sequence number of D that S has is seq_1 and another node I has a routing table entry for D with a sequence number seq_2. Suppose I now receives a RREQ packet from S for destination D. Recall that sequence numbers are integers and I can compare these two sequence numbers. If $seq_1 > seq_2$, it is clear that S has more recent information about D compared to I and I need not reply to this RREQ packet as its information about D is old. On the other hand, if $seq_1 < seq_2$, I has more recent information about D compared to S. Hence, the sequence numbers are used as a logical clock in a way similar to the DSDV protocol.

AODV uses a mechanism where every node along a route sets up forward and reverse paths during the route discovery phase. We will illustrate this mechanism through an example. Suppose an intermediate node J has received a RREQ packet from another node I and the source for this RREQ packet is a node S. J sets a *reverse route* entry in its routing table with I as the destination of this entry. If J receives a RREP in reply to this RREQ in future, this reverse route entry helps J to route the packet to S through I. J also stores the current time with this reverse route entry. The reverse route entry is deleted if it is not used within a specified lifetime.

Suppose a node M receives a RREP packet from one of its neighbors N. M creates *forward path* entry in its routing table with N as the destination of the forward path. If the source node S sends data packets in future, M can send these data packets to D through N. A node J can reply to a RREQ packet with a RREP only if it has an

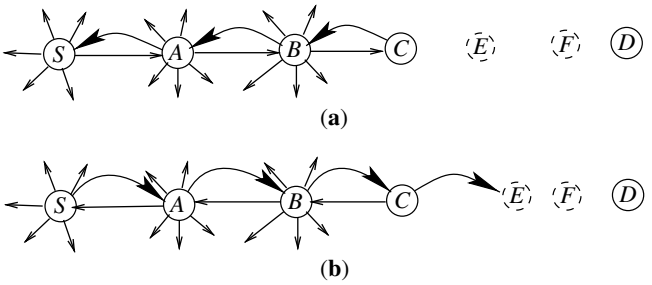

Figure 1.4 Illustration for the establishment of forward and reverse paths in the route discovery phase of the AODV protocol. (a) The propagation of the RREQ packet from the source node *S* is shown by the straight arrows. Each node makes a reverse path entry in its routing table by noting the ID of the neighbor from which it received the RREQ packet. These reverse path entries are illustrated by the bold and curved arrows. (b) The node *C* initiates a route reply by sending a RREP packet since it has a route to the destination *D*. When a node receives a RREP packet from one of its neighbors, it makes a forward path entry in its routing table by noting the ID of the neighbor from which the RREP has arrived. The propagation of the RREP packet is shown by the straight arrows and the forward path entries by the bold and curved arrows. Note that nodes such as *E* and *F* already have forward path entries as they are parts of a path to *D*.

unexpired route for *D* with a higher sequence number (compared to the sequence number in the RREQ packet) from *D*. The RREP packet follows the reverse path entries along the path through which the RREQ packet traveled to *J*. Figure 1.4 illustrates the forward and reverse path entries in the AODV protocol.

The route maintenance phase starts if there is a route breakage due to link failure during the data transfer phase along an established route. The sending of RERR packets is exactly similar to the DSR protocol, with the exception that nodes executing the AODV protocol do not try to salvage packets when a link breaks. When the source receives a RERR packet, it initiates a new route discovery phase.

Every node keeps only a single forward path entry for each route in the original AODV protocol. However, this may be inefficient because a new route discovery phase begins every time there is a route breakage, incurring large control packet overheads. There is a variant of the AODV protocol called the *multipath AODV* (AOMDV) [11] in which each node keeps multiple forward path entries for each route. For example, suppose a source node *S* has requested a route for a destination *D* and the resulting RREQ packet has passed through an intermediate node *J*. *J* may receive multiple RREP packets in future in reply to this RREQ. Assume that *J* receives RREP packets from three of its neighbors *U, V*, and *W*. In the original AODV protocol, *J* chooses only one of these three neighbors for keeping a forward path entry in its routing table and rejects the other two. In the AOMDV protocol, *J* keeps forward path entries to all the three neighbors. However, *J* marks these forward path entries with the times when the corresponding RREP packet was received and times out these entries if they become too old before they are used. Assume now that *J* chooses one of these entries, say to *U*, for forwarding data packets when *S* starts transmitting data. In case *U* moves out of the transmission range of *J* in future by breaking the *J–U* link and also the path

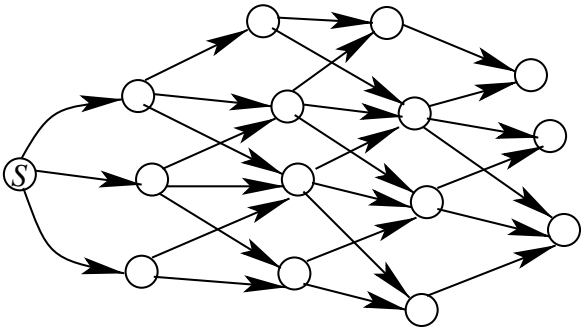

Figure 1.5 An illustration for the AOMDV protocol. In general, each node maintains multiple forward path entries in its routing table corresponding to each RREP it receives. These forward path entries are shown by bold arrows.

to the destination D, J can start using another of its forward path entries, say the link to V, for forwarding packets to D, provided the link to V is not too old. Hence, the need for sending a RERR packet to S due to the failure of the J–U link is eliminated in the AOMDV protocol. However, a RERR packet needs to be sent if the last forward path entry in J's routing table breaks. The AOMDV protocol has better performance in terms of lower packet overhead compared to the AODV protocol. We illustrate the forward path entries in the AOMDV protocol in Figure 1.5.

We now discuss how AODV supports multicasting in a MANET. The main aim is to maintain a *multicasting tree* for each *multicasting group* of nodes. A multicasting tree consists of two kinds of nodes, *multicast group members* and *multicast tree members*. All other nodes in the MANET are *non-tree nodes* from the point of view of a multicasting group. Multicast tree members are not members of the multicast group; however, they are required for maintaining connectivity of the multicasting tree. It is important to keep a multicasting tree connected as a packet received by any member of a tree can send the packet to any other member. Also, it is important to have a tree instead of a graph to avoid routing loops.

A multicasting tree is considered as a single entity for the purpose of route discovery in AODV. Recall that any RREQ packet should contain a destination node ID. A node I can send a RREQ packet to a multicasting group M in two different cases, either I wants to send packets to the nodes in M (a data request) or I wants to join M (a join request). In case of a data request, any node that has a current path to the multicasting group can send a RREP packet in reply to the RREQ. The meaning of a current path is same as in the original or unicasting version of AODV that we have discussed before. Also, a path to the multicasting group means a path to any member of the multicasting group. In case of a join request, only a member of the multicasting tree can send the RREP. The setting of reverse path and forward path entries is similar to the unicast version of AODV, with one exception. The sender of a RREQ packet with a join request may receive multiple potential branches that connect to the multicasting tree, since the RREP packets travel through different routes. The

sender should select only one of these branches as a path to the multicasting group by sending a *multicast activation* message.

It is important to maintain a multicast tree so that the tree does not get disconnected due to the mobility of the group members. Usually a special node is selected as the leader of the multicast tree and each node knows the ID of this leader node and also the path to the leader node. When a link breaks, the node that is closer to the leader initiates a route repair by sending a RREQ as a join request. The link is repaired when the corresponding RREP comes back and the node has found a new path to the multicasting group.

1.5 OTHER ROUTING PROTOCOLS

Till now we have discussed the two main categories of routing protocols for MANETs, namely, proactive and reactive protocols. However, there are other protocols that cannot be classified as purely reactive or purely proactive. We discuss two such important classes of protocols in this section, namely, the zone routing protocol (ZRP) and link reversal routing protocols.

1.5.1 The Zone Routing Protocol

The zone routing protocol [7] takes advantage of both proactive and reactive routing strategies in a MANET. Recall that the main drawback of proactive protocols is their large overhead due to proactive exchange of routing tables or link state information and that of reactive protocols is their high latency. ZRP tries to overcome both of these drawbacks while preserving the advantages of both proactive and reactive protocols.

Each node N executing the ZRP in a MANET maintains its routing zone. A routing zone is a k-hop neighborhood of N, where k is usually a small number from 2 to 4. k is also called the *radius* of a routing zone and is same for all nodes in the MANET. Each node routes proactively within its routing zone, that is, it tries to maintain a complete routing table for all the nodes in its routing zone, like the DSDV protocol. The routing outside this routing zone is done reactively, like the DSR or AODV protocol.

Suppose node S wants to start a data communication with node D. If D is within the routing zone of S, then there is no need for route discovery, as S can find a route to D by consulting its routing table. If D is outside the routing zone of S, there is a need for route discovery; however, the route discovery process is not as expensive as that of a reactive protocol. S initiates route discovery by sending a RREQ packet as in case of reactive protocols, but this RREQ packet is sent more efficiently by using *border nodes* in each routing zone. A border node is a node on the periphery of a routing zone. For example, suppose the radius of the routing zone is 2. For node S, all the nodes that are two-hop neighbors of S are the border nodes of the routing zone of S. Since S has a routing table for its routing zone, it can quickly route the RREQ packet to all these border nodes by consulting its routing table. Suppose B is one of

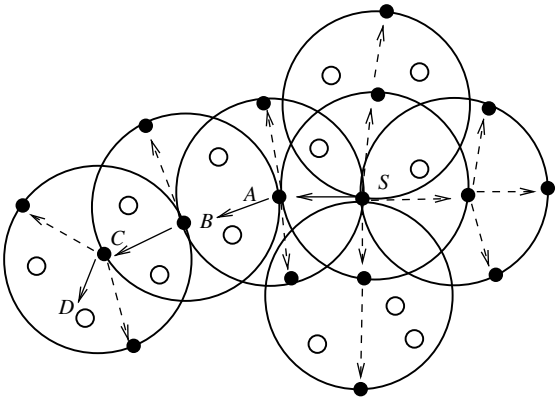

Figure 1.6 An illustration for the propagation of RREQ in the ZRP. The large circles represent the routing zones of individual nodes. The small solid circles represent border nodes and the empty circles represent other nodes. The solid arrows represent the propagation of the RREQ that succeeds in finding a route. *C* can send a RREP since the destination *D* is within its routing zone. The RREQ is propagated from one node to another on its zone boundary. The dashed arrows indicate the propagation of RREQs that are not successful.

these border nodes in the routing zone of *S*. There are two possibilities, either *D* is within the routing zone of *B* (recall that each node proactively maintains its routing zone) or *D* is not within the routing zone of *B*. In the first case, *B* can send a RREP packet back to *S* as it has a path to *D* from its routing table. *B* needs to forward the RREQ further in the second case and this is again done by using the border nodes of *B*'s routing zone. The propagation of the RREQ packet continues until either it reaches *D* or it reaches a node *C* such that *D* is within the routing zone of *C*. This process is illustrated in Figure 1.6.

It is easy to see that the propagation of a RREQ packet in the ZRP is more efficient compared to that in a pure reactive protocol such as DSR or AODV. Every node that receives the packet needs to send the packet to its border nodes and the routes to the border nodes can be found consulting the corresponding routing tables. A RREP packet is also propagated back in exactly the same way, that is, using border nodes. Hence, overall ZRP achieves a better performance in terms of lower latency compared to pure reactive protocols.

We need to ask the question whether the proactive maintenance of routing tables within routing zones incurs considerable overhead in the ZRP. The overhead is indeed high if the routing zones are large, and Haas and Perlman [7] report that the overhead is significantly lower if the radius of a routing zone is kept small, typically 2 or 3. This choice of the radius maintains both low latency and high throughput even under high-mobility scenarios.

We close our discussion on ZRP by mentioning an important optimization that needs to be done to make the protocol efficient. There is usually a significant overlap among the routing zones of different nodes in a MANET. For example, consider two

neighboring nodes *I* and *J*. The routing zones of *I* and *J* will contain almost the same nodes, except for few exceptions. Moreover, if a node *M* is a border node for the routing zone of another node *N*, *N* is also a border node for the routing zone of *M*. If the RREQ packets are sent always to the border nodes of routing zones, as we have discussed above, the RREQ packets will flood the same part of the network again and again. This is clearly undesirable. ZRP tries to remove this type of flooding by directing a RREQ packet to the parts of MANET where the packet has not yet arrived and avoids flooding the parts where the packet has already arrived. A node that has broadcast a RREQ suppresses the same RREQ in future if it gets the RREQ again. Similarly, nodes that have overheard the RREQ in the promiscuous mode know that the RREQ has arrived in their part of the network and avoid forwarding the RREQ in future. This simple optimization reduces the overhead of route discovery considerably while using the ZRP.

1.5.2 Link Reversal Routing Protocols

The main idea behind link reversal routing is to treat a MANET as a directed graph. The first such protocol was designed for static packet radio networks by Gafni and Bertsekas [6] and the MANET community later extended the idea to networks of mobile nodes. We first discuss the protocol in Ref. [6].

1.5.2.1 Gafni–Bertsekas Protocol

Consider a network of static wireless nodes with a single destination node *D* and possibly many source nodes. The source nodes want to send packets to the destination node. The link between a pair of nodes is determined by their transmission radii. We will assume for simplicity that all links between neighboring nodes are bidirectional, that is, they are within the transmission range of each other. However, the protocol works equally well for the case when the links are asymmetric.

The problem of routing to a single destination *D* can be naturally framed as a problem on a directed acyclic graph such that the nodes in the wireless network are nodes in this graph and the edges between neighboring nodes in the network are edges of the graph. Every node other than *D* has at least one outgoing edge and *D* is the only node without any outgoing edge. Hence, *D* acts as a sink for all the packets transmitted in the network. A packet transmitted over a directed edge is forwarded by the recipient of the packet over an outgoing edge and this process continues until the packet reaches the destination *D*. It cannot be forwarded again as *D* does not have any outgoing edges. Hence, the routing scheme becomes very simple. A node (other than the destination) does not need to know its position in the network or the position of the destination. If it has to route a packet to the destination, it simply needs to forward the packet over one of its outgoing edges and the packet is guaranteed to reach the destination since there is no loop in the DAG and also *D* is the only sink, or node without any outgoing edges, in the network. Hence, a node executing the Gafni–Bertsekas protocol does not need to either maintain routing tables like proactive protocols or run a route discovery

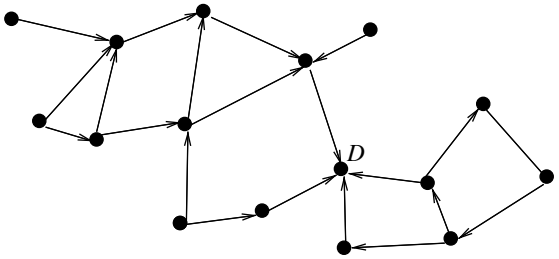

Figure 1.7 The view of a wireless network as a directed acyclic graph. The destination D is the only node that does not have any outgoing edge, every other node has at least one outgoing edge. It is easy to see that a node needs to forward a packet over one of its outgoing edges and the packet will eventually reach D.

phase like reactive protocols. A sample wireless network as a DAG is shown in Figure 1.7.

The success of the Gafni–Bertsekas protocol of course depends on route maintenance; that is, we have to ensure that always the graph is maintained as a DAG and also D is the only sink in the graph. Let us first examine how the DAG is established in the first place. Initially, the network is just a graph with links between neighbors when they are within the transmission range of each other. Only the neighbors of D have directed links to D. The initialization of the network as a DAG starts when a node S other than D needs a path to D for routing packets. S floods a QRY packet in the network in the usual manner. Any node that has a path to D replies to this QRY by sending a RPY packet. The RPY packets travel exactly in the opposite direction compared to the QRY packets. We need to explain at this point the meaning of a *path to the destination* in link reversal routing protocols. Such a path does not have any global meaning from a node's point of view in the sense that we have seen in proactive and reactive routing protocols. For a node N, a path to the destination D exists if N has at least one outgoing directed edge.

A node can send a RPY packet if it is either the destination D or one of the neighbors of D, as only the neighbors of D have a path to D due to their outgoing links to D. Suppose a node I has received a RPY packet from a node J and I and J are neighbors. I can now mark its link to J as an outgoing link as J has a route to D. After this, I also has a route to D and I can now send RPY packets to its neighbors. The initialization proceeds in this way until all the nodes in the network receive RPY packets and set their outgoing directed links. The network now is a DAG with only one sink D as we desired.

A node can lose a link in a static wireless network if nodes are allowed to switch off, that is, they stop participating in the network. Recall that the Gafni–Bertsekas protocol was designed for a packet radio network, where participation is voluntary. In general, there is no harm if a node withdraws from the network, until the situation arises that another node, say T, other than the destination becomes a sink. In other words, T loses all its outgoing links and becomes another sink in the network. This is clearly undesirable, as any packet reaching T will be stuck there and will not be

delivered to *D*. The Gafni–Bertsekas protocol rectifies this situation by two mechanisms called *full reversal* and *partial reversal*.

The main activity in a link reversal routing protocol is route maintenance. The two reversal mechanisms maintain the directed acyclicity of the network graph and hence maintain routes in the graph for a destination *D*. In the full reversal mechanism, node *I* that has lost all of its outgoing links reverses all its incoming links to make them outgoing. As a result, one or more of *I*'s neighbors may lose their outgoing links and the full reversal process continues until all the nodes have at least one outgoing link each. It can be shown that the full reversal process terminates within a finite time (usually a short time in a wireless network) if the network is connected, that is, the destination can be reached from each node. The partial reversal mechanism is more selective. Consider two neighboring nodes *I* and *J*. Suppose *I* has just reversed all its links and *J* has lost its last outgoing link as a result. *J* does not reverse its link to *I* and instead reverses its other incoming links in the partial reversal mechanism. The partial reversal mechanism is illustrated in Figure 1.8.

The Gafni–Bertsekas protocol guarantees a directed acyclic graph once the network stabilizes. However, the network may have loops and hence the acyclicity property may not hold temporarily when the partial or full reversal mechanisms are in action. However, the reversal mechanisms usually terminate within a short time in a wireless network. Although this protocol is quite elegant and efficient in static wireless networks, one of its main drawbacks is that it does not work in partitioned networks. It can be shown easily that the full or partial reversal mechanism may go into an indefinite cycle (the nodes continue reversing their links for an indefinite period of

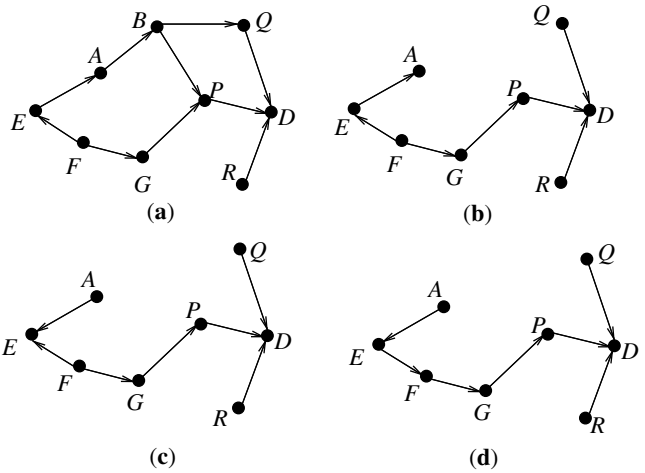

Figure 1.8 An illustration of the partial reversal mechanism in the Gafni–Bertsekas protocol. (a) Each node except the destination *D* has at least one outgoing link. (b) *B* moves away and *A* loses its last outgoing link. (c) *A* reverses its incoming links, the link to *E* in this case. *E* now loses its last outgoing link. (d) *E* does not reverse its incoming link to *A* as *A* has just reversed it. *E* instead reverses its other incoming link to *F*. The network is now again a DAG with only one sink (*D*).

time) for nodes that are partitioned from the destination D. This wastes bandwidth of the wireless network. Since mobile ad hoc networks may get partitioned from time to time due to the mobility of nodes, this protocol cannot be directly used in a MANET.

We need to mention another important point about the Gafni–Bertsekas protocol. Throughout our discussion we have assumed only one destination in the network. However, this is not realistic as usually any node in a wireless network may act as a source or destination of packets. However, the protocol can be extended easily for multiple destinations. Conceptually, we need to maintain k different DAGs for k destinations. Note that there is no global maintenance of DAG in the Gafni–Bertsekas protocol, rather individual nodes store local information to get a global DAG in a distributed fashion. A node can keep track of its links with its neighbors in a local routing table and assign status to these links as incoming and outgoing. Also, nodes need to change the status of these links during the link reversal process. It is possible to support k destinations by keeping k copies of each link with a neighbor and assigning a status (incoming or outgoing) for each copy according to the status of the DAG for a particular destination. A link between neighbors I and J could be incoming for node I for a destination D_p and outgoing for I for another destination D_q. When I receives a packet for destination D_p, it cannot forward that packet to J because the direction of the link is from J to I. On the other hand, when it receives a packet for destination D_q, it can forward the packet to J since the direction of the link is from I to J for destination D_q.

1.5.2.2 Temporally Ordered Routing Algorithm

The TORA protocol [4, 14] was designed as a modification of the Gafni–Bertsekas algorithm for mobile ad hoc networks. We cannot discuss this protocol in detail due to limitation of space and a detailed description can be found in the review article by Corson and Park [5]. We discuss only the essential components of this protocol that make it suitable for using in MANETs.

The main aim of the TORA protocol is to make it suitable for networks that may become partitioned, as is the case for MANETs. It retains the core ideas behind the Gafni–Bertsekas protocol, such as using the network as a DAG and maintaining the DAG through full and partial reversal of links. However, it has some extra mechanisms for detecting partitions in a MANET.

The direction of the links between neighbors in the TORA protocol is enforced through a height assignment to each end point of a link. Suppose I and J are two neighboring nodes and we assign integers as heights to I and J to fix the direction of the link between them. For example, if I is assigned a height of 4 and J a height of 7, then J has a greater height compared to I and the direction of the link between them will be from J to I. In other words, the link between I and J will be incoming for I and outgoing for J. If I wants to reverse this link, it has to make its own height higher than J, say 8. Link reversals in TORA are done by changing the heights of the two end nodes of a link. However, the height assignment in TORA is more complex, with each height consisting of five components. Two heights are compared according to

the lexicographic ordering of these five components. When a node wants to change its height, it usually tries to borrow the height of one of its neighbors and increases one of the components so that its height becomes higher than that neighbor. This is basically the idea of partial reversal in the Gafni–Bertsekas algorithm. There are several rules in changing heights in TORA and we will not discuss all the rules; however, we will only discuss the way TORA detects partition in the network.

One of the components in a height is the *origin ID* or *oid* and the other is time. There are situations when a node cannot borrow a neighbor's height and increase it to do partial reversal. In such cases, a node (say *I*) initiates a global new height by assigning the first component as the current time. Since current time is the global highest time, the new height becomes lexicographically highest in the entire network. The node *I* initiating a new height also sets the *oid* field of the new height with its own ID. It is then possible for *I* to detect in future if all of its neighbors have the height that it had initiated. This means all of its neighbors have tried to borrow *I*'s height and increase it to establish outgoing links to *I*. This is an indication of a partition in the network as no node is able to force a new route to the destination by increasing others' heights. Nodes executing TORA erase all their heights in such cases as there is no point in trying to find new routes since the destination is unreachable. The destination may become reachable in future again due to the mobility of nodes when there are new nodes between the partitions and these nodes connect the partitions.

Although TORA is an elegant protocol for routing in MANETs, it has quite high overhead in high-mobility scenarios. In particular, the link reversals become excessive when there are frequent link breakages in the network. It has been shown [2] that the performance of TORA becomes worse compared to reactive protocols in high-mobility scenarios. However, the performance of TORA is comparable to protocols such as DSR and AODV in low- and moderate-mobility scenarios.

1.6 CONCLUSION

We have reviewed several key protocols for routing in mobile ad hoc networks. However, this review is not comprehensive as we have left many other protocols from our discussion due to limitation of space. One such class of protocols is the class of position-based routing protocols [20], where each node knows the locations of all the other nodes through a location service such as global positioning system (GPS). It can be shown that the overheads of reactive protocols can be reduced considerably in this class of protocols. We refer the reader to the review article by Stojmenovic [20] for more details.

We conclude this chapter by briefly discussing how the MANET routing protocols are evaluated in terms of their performance, such as throughput and latency in packet delivery. Since a real evaluation of a protocol through deploying mobile nodes is expensive, most such evaluations are done through simulations in a discrete event simulator. The network simulator [12] or ns-2 has become the most popular simulator in the research community for this purpose. ns-2 has been developed for over a

decade by volunteers who are researchers and PhD students in different universities worldwide and it is still in the process of development with new protocols and features being added every year. It is written in C++, one of the most popular object-oriented programming languages. The protocols discussed in this chapter are usually evaluated under the *random waypoint* model of mobility of nodes. In this model, a node moves in a random direction with a random speed (bounded above by a maximum speed) and then pauses for a specified time before moving in another random direction again. This process continues throughout the simulation period. The range of mobilities (in terms of maximum speed) usually considered by researchers is in the range of 1 m/s (pedestrian speed) to 20 m/s (speed of a car). Much more details about performance evaluation of specific protocols can be found from the references cited in this chapter.

REFERENCES

1. E. Belding-Royer and C. E. Perkins. Evolution and future directions of the ad hoc on-demand distance-vector routing protocol. *Ad Hoc Networks*, 1(1):125–150, 2003.
2. J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. *Proceedings of the ACM MOBICOM 1998*, pp. 85–97.
3. M. S. Corson and J. Macker. Mobile ad hoc networking (MANET): routing protocol performance issues and evaluation considerations, *IETF MANET, RFC 2501*, 1999. http://www.ietf.org/html .charters/manet-charter.html.
4. M. S. Corson and A. Ephremides. A distributed routing algorithm for mobile wireless networks. *ACM/Baltzer Wireless Networks J.*, 1(1):61–82, 1995.
5. M. S. Corson and V. Park. In: C. Perkins (Ed.), Link Reversal Routing in Ad Hoc Networking Addison-Wesley, 2001, pp. 255–298, Chapter 8.
6. E. Gafni and D. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Trans. Commun.*, 29(1):11–18, 1981.
7. Z. J. Haas and M. R. Perlman. ZRP: a hybrid framework for routing in ad hoc networks. In: C. Perkins (Ed.), *Ad Hoc Networking*, Addison-Wesley, 2001, pp. 221–253, Chapter 7.
8. IETF MANET working group. http://www.ietf.org/html.charters/manet-charter.html.
9. D. B. Johnson, D. A. Maltz, and J. Broch. DSR: the dynamic source routing protocol for multi-hop wireless ad hoc networks. In: C. Perkins (Ed.), *Ad Hoc Networking*, Addison-Wesley, 2001, pp. 139–172, Chapter 5.
10. D. B. Johnson, D. A. Maltz, and Y. Hu. The dynamic source routing protocol for mobile ad hoc networks. *IETF MANET, Internet Draft*, 2003. http://www.ietf.org/html.charters/ manet-charter.html.
11. M. K. Marina and S. Das. On-demand multipath distance vector routing in ad hoc networks *Proceedings Ninth International Conference on Network Protocols (ICNP)*, pp. 14–23, 2001.
12. NS : The Network Simulator. http://www.isi.edu/nsnam/ns/.
13. Optimized link state routing protocol. *IETF MANET, RFC 3626*, 2003. http://www.ietf.org/ html.charters/manet-charter.html
14. V. Park and S. Corson. Temporally ordered routing algorithm (TORA) Version 1: functional specification. *IETF MANET, Internet Draft*, 2001. http://www.ietf.org/html.charters/manet-charter.html
15. C. Perkins (Ed.), *Ad Hoc Networking*. Addison Wesley, 2001.
16. C. Perkins and P. Bhagwat. DSDV: routing over a multihop wireless network of mobile computers. In: C. Perkins (Ed.), *Ad Hoc Networking*, Addison-Wesley, 2001, pp. 53–74, Chapter 3.

17. C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing, *IETF RFC 3591*, 2003.
18. A. A. Pirzada, C. McDonald, and A. Datta. Performance comparison of trust-based reactive routing protocols. *IEEE Trans. Mobile Comput.*, 5(6):695–710, 2006.
19. E. M. Belding-Royer and C. K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Commun. Mag.*, 6(2):46–55, 1999.
20. I. Stojmenovic. Position-based routing in ad hoc networks. *IEEE Commun. Mag.*, 40(7):128–134, 2002.