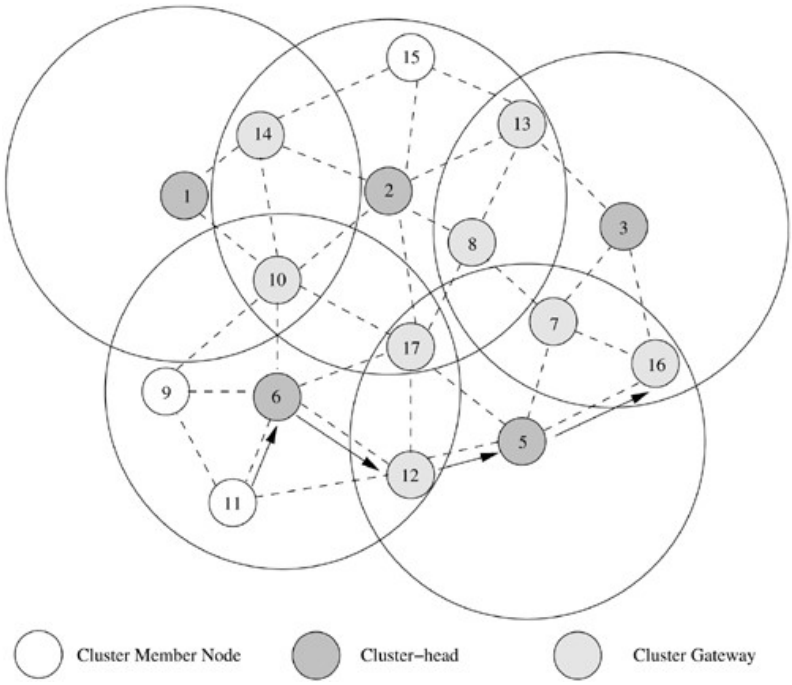
7.4.3 Cluster-Head Gateway Switch Routing Protocol

The cluster-head gateway switch routing protocol (CGSR) [8] uses a hierarchical network topology, unlike other table-driven routing approaches that employ flat topologies. CGSR organizes nodes into clusters, with coordination among the members of each cluster entrusted to a special node named *cluster-head*. This cluster-head is elected dynamically by employing a *least cluster change (LCC)* algorithm [8]. According to this algorithm, a node ceases to be a cluster-head only if it comes under the range of another cluster-head, where the tie is broken either using the lowest ID or highest connectivity algorithm. Clustering provides a mechanism to allocate bandwidth, which is a limited resource, among different clusters, thereby improving reuse. For example, different cluster-heads could operate on different spreading codes on a CDMA system. Inside a cluster, the cluster-head can coordinate the channel access based on a token-based polling protocol. All member nodes of a cluster can be reached by the cluster-head within a single hop, thereby enabling the cluster-head to provide improved coordination among nodes that fall under its cluster. A token-based scheduling (assigning access token to the nodes in a cluster) is used within a cluster for sharing the bandwidth among the members

of the cluster. CGSR assumes that all communication passes through the cluster-head. Communication between two clusters takes place through the common member nodes that are members of both the clusters. These nodes which are members of more than one cluster are called *gateways.* A gateway is expected to be able to listen to multiple spreading codes that are currently in operation in the clusters in which the node exists as a member. A gateway conflict is said to occur when a cluster-head issues a token to a gateway over a spreading code while the gateway is tuned to another code. Gateways that are capable of simultaneously communicating over two interfaces can avoid gateway conflicts.

The performance of routing is influenced by token scheduling and code scheduling (assigning appropriate spreading codes to two different clusters) that are handled at cluster-heads and gateways, respectively. The routing protocol used in CGSR is an extension of DSDV. Every member node maintains a routing table containing the destination cluster-head for every node in the network. In addition to the cluster member table, each node maintains a routing table which keeps the list of next-hop nodes for reaching every destination cluster. The *cluster (hierarchical) routing protocol* is used here. As per this protocol, when a node with packets to be transmitted to a destination gets the token from its cluster-head, it obtains the destination cluster-head and the next-hop node from the cluster member table and the routing table, respectively. CGSR improves the routing performance by routing packets through the cluster-heads and gateways. A path from any node *a* to any node *b* will be similar to $a - C_1 - G_1 - C_2 - G_2 - \ldots C_i - G_j \ldots G_n - b$, where G_i and C_j are the i^{th} gateway and the j^{th} cluster-head, respectively, in the path. Figure 7.9 shows the cluster-heads, *cluster gateways,* and normal cluster member nodes in an ad hoc wireless network. A path between node 11 and node 16 would follow 11 - 6 - 12 - 5 - 16. Since the cluster-heads gain more opportunities for transmission, the cluster-heads, by means of a dynamic scheduling mechanism, can make CGSR obtain better delay performance for real-time flows. Route reconfiguration is necessitated by mainly two factors: firstly, the change in cluster-head and secondly, the stale entries in the cluster member table and routing table. CGSR depends on the table update mechanism to handle the latter problem, while the least cluster change algorithm [8] handles the former.

Figure 7.9. Route establishment in CGSR.



Advantages and Disadvantages

CGSR is a hierarchical routing scheme which enables partial coordination between nodes by electing cluster-heads. Hence, better bandwidth utilization is possible. It is easy to implement priority scheduling schemes with token scheduling and gateway code scheduling. The main disadvantages ofCGSR are increase in path length and instability in the system at high mobility when the rate of change of cluster-heads is high. In order to avoid gateway conflicts, more resources (such as additional interfaces) are required. The power consumption at the cluster-head node is also a matter of concern because the battery-draining rate at the cluster-head is higher than that at a normal node. This could lead to frequent changes in the cluster-head, which may result in multiple path breaks.

7.5 ON-DEMAND ROUTING PROTOCOLS

Unlike the table-driven routing protocols, on-demand routing protocols execute the path-finding process and exchange routing information only when a path is required by a node to communicate with a destination. This section explores some of the existing on-demand routing protocols in detail.

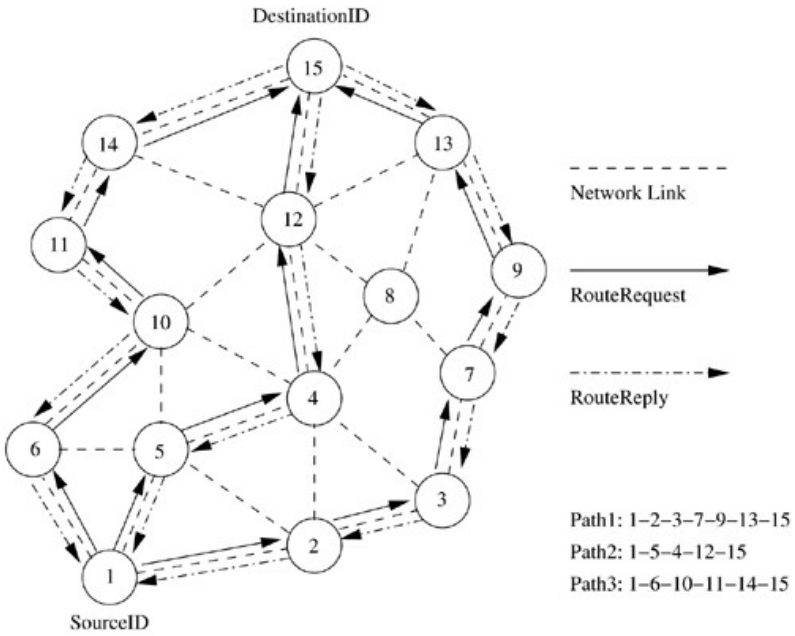
7.5.1 Dynamic Source Routing Protocol

Dynamic source routing protocol (DSR) [10] is an on-demand protocol designed to restrict the bandwidth consumed by control packets in ad hoc wireless networks by eliminating the periodic table-update messages required in the table-driven approach. The major difference between this and the other on-demand routing protocols is that it is *beacon-less* and hence does not require periodic *hello*packet (*beacon*) transmissions, which are used by a node to inform its neighbors of its presence. The basic approach of this protocol (and all other

on-demand routing protocols) during the route construction phase is to establish a route by flooding *RouteRequest* packets in the network. The destination node, on receiving a *RouteRequest* packet, responds by sending a *RouteReply* packet back to the source, which carries the route traversed by the *RouteRequest* packet received.

Consider a source node that does not have a route to the destination. When it has data packets to be sent to that destination, it initiates a *RouteRequest* packet. This *RouteRequest* is flooded throughout the network. Each node, upon receiving a *RouteRequest* packet, rebroadcasts the packet to its neighbors if it has not forwarded already or if the node is not the destination node, provided the packet's time to live (TTL) counter has not exceeded.
Each *RouteRequest* carries a sequence number generated by the source node and the path it has traversed. A node, upon receiving a *RouteRequest*packet, checks the sequence number on the packet before forwarding it. The packet is forwarded only if it is not a duplicate *RouteRequest.* The sequence number on the packet is used to prevent loop formations and to avoid multiple transmissions of the same *RouteRequest* by an intermediate node that receives it through multiple paths. Thus, all nodes except the destination forward a *RouteRequest*packet during the route construction phase. A destination node, after receiving the first *RouteRequest*packet, replies to the source node through the reverse path the *RouteRequest* packet had traversed. InFigure 7.10, source node 1 initiates a *RouteRequest* packet to obtain a path for destination node 15. This protocol uses a route cache that stores all possible information extracted from the source route contained in a data packet. Nodes can also learn about the neighboring routes traversed by data packets if operated in the promiscuous mode (the mode of operation in which a node can receive the packets that are neither broadcast nor addressed to itself). This route cache is also used during the route construction phase. If an intermediate node receiving a *RouteRequest* has a route to the destination node in its route cache, then it replies to the source node by sending a *RouteReply* with the entire route information from the source node to the destination node.

Figure 7.10. Route establishment in DSR.

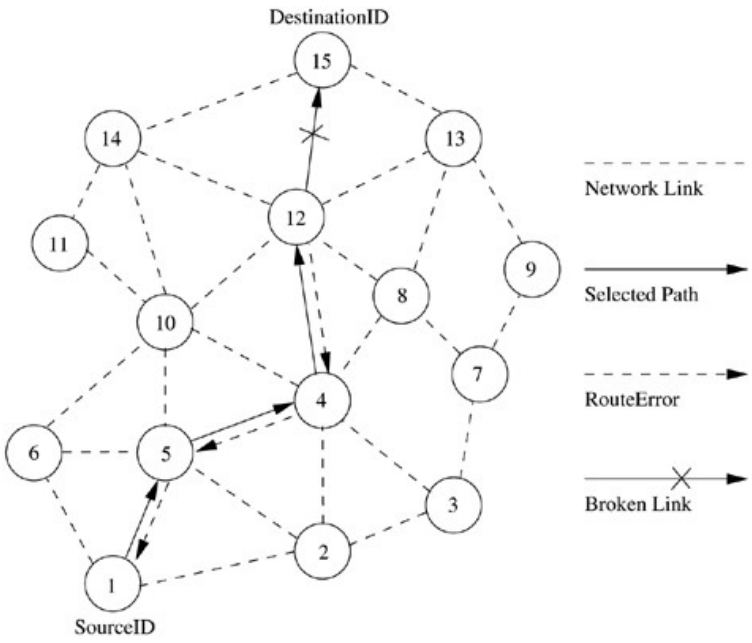


Optimizations

Several optimization techniques have been incorporated into the basic DSR protocol to improve the performance of the protocol. DSR uses the route cache at intermediate nodes. The route cache is populated with routes that can be extracted from the information contained in data packets that get forwarded. This cache information is used by the intermediate nodes to reply to the source when they receive a *RouteRequest* packet and if they have a route to the corresponding destination. By operating in the promiscuous mode, an intermediate node learns about route breaks. Information thus gained is used to update the route cache so that the active routes maintained in the route cache do not use such broken links. During network partitions, the affected nodes initiate *RouteRequest* packets. An exponential backoff algorithm is used to avoid frequent *RouteRequest* flooding in the network when the destination is in another disjoint set. DSR also allows piggy-backing of a data packet on the*RouteRequest* so that a data packet can be sent along with the *RouteRequest.*

If optimization is not allowed in the DSR protocol, the route construction phase is very simple. All the intermediate nodes flood the *RouteRequest* packet if it is not redundant. For example, after receiving the *RouteRequest* packet from node 1 (refer to Figure 7.10), all its neighboring nodes, that is, nodes 2, 5, and 6, forward it. Node 4 receives the *RouteRequest* from both nodes 2 and 5. Node 4 forwards the first *RouteRequest* it receives from any one of the nodes 2 and 5 and discards the other redundant/duplicate *RouteRequest* packets.
The *RouteRequest* is propagated till it reaches the destination which initiates the *RouteReply.* As part of optimizations, if the intermediate nodes are also allowed to originate *RouteReply* packets, then a source node may receive multiple replies from intermediate nodes. For example, in Figure 7.11, if the intermediate node 10 has a route to the destination via node 14, it also sends the *RouteReply* to the source node. The source node selects the latest and best route, and uses that for sending data packets. Each data packet carries the complete path to its destination.

Figure 7.11. Route maintenance in DSR.



When an intermediate node in the path moves away, causing a wireless link to break, for example, the link between nodes 12 and 15 in Figure 7.11, a *RouteError* message is generated from the node adjacent to the broken link to inform the source node. The source node reinitiates the route establishment procedure. The cached entries at the intermediate nodes and the source node are removed when a*RouteError* packet is received. If a link breaks due to the movement of edge nodes (nodes 1 and 15), the source node again initiates the route discovery process.

Advantages and Disadvantages

This protocol uses a reactive approach which eliminates the need to periodically flood the network with table update messages which are required in a table-driven approach. In a reactive (on-demand) approach such as this, a route is established only when it is required and hence the need to find routes to all other nodes in the network as required by the table-driven approach is eliminated. The intermediate nodes also utilize the route cache information efficiently to reduce the control overhead. The disadvantage of this protocol is that the route maintenance mechanism does not locally repair a broken link. Stale route cache information could also result in inconsistencies during the route reconstruction phase. The connection setup delay is higher than in table-driven protocols. Even though the protocol performs well in static and low-mobility environments, the performance degrades rapidly with increasing mobility. Also, considerable routing overhead is involved due to the source-routing mechanism employed in DSR. This routing overhead is directly proportional to the path length.

7.5.2 Ad Hoc On-Demand Distance-Vector Routing Protocol

Ad hoc on-demand distance vector (AODV) [11] routing protocol uses an on-demand approach for finding routes, that is, a route is established only when it is required by a source node for transmitting data packets. It employs destination

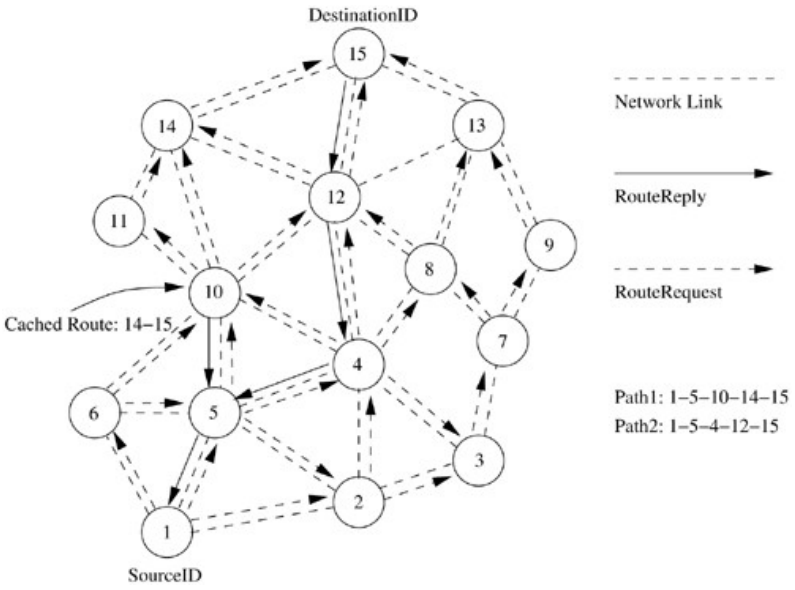
sequence numbers to identify the most recent path. The major difference between AODV and DSR stems out from the fact that DSR uses source routing in which a data packet carries the complete path to be traversed. However, in AODV, the source node and the intermediate nodes store the next-hop information corresponding to each flow for data packet transmission. In an on-demand routing protocol, the source node floods the *RouteRequest* packet in the network when a route is not available for the desired destination. It may obtain multiple routes to different destinations from a single *RouteRequest.* The major difference between AODV and other on-demand routing protocols is that it uses a destination sequence number (DestSeqNum) to determine an up-to-date path to the destination. A node updates its path information only if the DestSeqNum of the current packet received is greater than the last DestSeqNum stored at the node.

A *RouteRequest* carries the source identifier (SrcID), the destination identifier (DestID), the source sequence number (SrcSeqNum), the destination sequence number (DestSeqNum), the broadcast identifier (BcastID), and the time to live (TTL) field. DestSeqNum indicates the freshness of the route that is accepted by the source. When an intermediate node receives a *RouteRequest,* it either forwards it or prepares a *RouteReply* if it has a valid route to the destination. The validity of a route at the intermediate node is determined by comparing the sequence number at the intermediate node with the destination sequence number in the *RouteRequest* packet. If a *RouteRequest* is received multiple times, which is indicated by the BcastID-SrcID pair, the duplicate copies are discarded. All intermediate nodes having valid routes to the destination, or the destination node itself, are allowed to send*RouteReply* packets to the source. Every intermediate node, while forwarding a *RouteRequest,* enters the previous node address and its BcastID. A timer is used to delete this entry in case a *RouteReply* is not received before the timer expires. This helps in storing an active path at the intermediate node asAODV does not employ source routing of data packets. When a node receives a *RouteReply* packet, information about the previous node from which the packet was received is also stored in order to forward the data packet to this next node as the next hop toward the destination.

Consider the example depicted in Figure 7.12. In this figure, source node 1 initiates a path-finding process by originating a *RouteRequest* to be flooded in the network for destination node 15, assuming that the *RouteRequest* contains the destination sequence number as 3 and the source sequence number as 1. When nodes 2, 5, and 6 receive the *RouteRequest* packet, they check their routes to the destination. In case a route to the destination is not available, they further forward it to their neighbors. Here nodes 3, 4, and 10 are the neighbors of nodes 2, 5, and 6. This is with the assumption that intermediate nodes 3 and 10 already have routes to the destination node, that is, node 15 through paths 10-14-15 and 3-7-9-13-15, respectively. If the destination sequence number at intermediate node 10 is 4 and is 1 at intermediate node 3, then only node 10 is allowed to reply along the cached route to the source. This is because node 3 has an older route to node 15 compared to the route available at the source node (the destination sequence number at node 3 is 1, but the destination sequence number is 3 at the source node), while node 10 has a more recent route (the destination sequence number is 4) to the destination. If the *RouteRequest* reaches the destination (node 15) through path 4-12-15 or any other alternative route, the destination also sends a *RouteReply* to the source. In

this case, multiple *RouteReply* packets reach the source. All the intermediate nodes receiving a *RouteReply* update their route tables with the latest destination sequence number. They also update the routing information if it leads to a shorter path between source and destination.

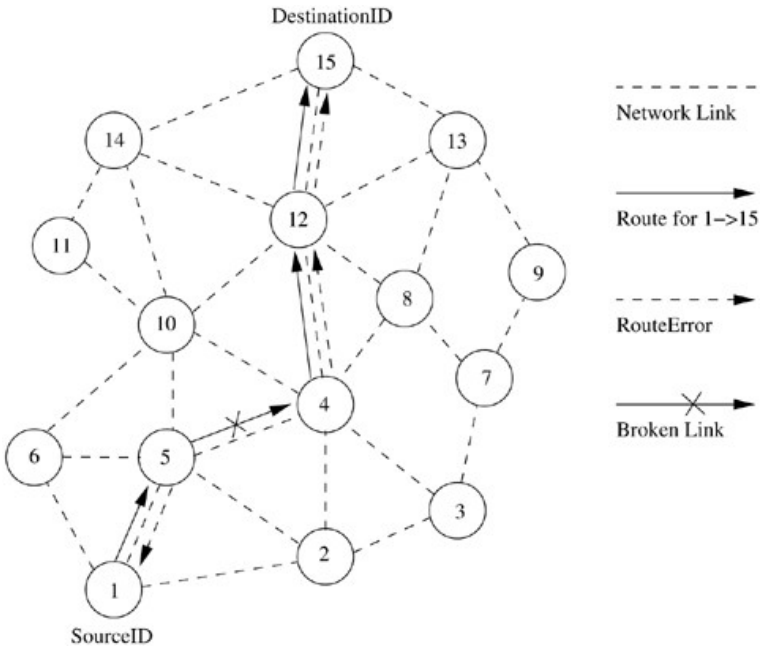
Figure 7.12. Route establishment in AODV.



AODV does not repair a broken path locally. When a link breaks, which is determined by observing the periodical *beacon*s or through link-level acknowledgments, the end nodes (*i.e.,* source and destination nodes) are notified. When a source node learns about the path break, it reestablishes the route to the destination if required by the higher layers. If a path break is detected at an intermediate node, the node informs the end nodes by sending an unsolicited *RouteReply* with the hop count set as ∞.

Consider the example illustrated in Figure 7.13. When a path breaks, for example, between nodes 4 and 5, both the nodes initiate *RouteError* messages to inform their end nodes about the link break. The end nodes delete the corresponding entries from their tables. The source node reinitiates the path-finding process with the new BcastID and the previous destination sequence number.

Figure 7.13. Route maintenance in AODV.



Advantages and Disadvantages

The main advantage of this protocol is that routes are established on demand and destination sequence numbers are used to find the latest route to the destination. The connection setup delay is less. One of the disadvantages of this protocol is that intermediate nodes can lead to inconsistent routes if the source sequence number is very old and the intermediate nodes have a higher but not the latest destination sequence number, thereby having stale entries. Also multiple *RouteReply* packets in response to a single *RouteRequest* packet can lead to heavy control overhead. Another disadvantage ofAODV is that the periodic *beacon*ing leads to unnecessary bandwidth consumption.

7.6 HYBRID ROUTING PROTOCOLS

In this section, we discuss the working of routing protocols termed as hybrid routing protocols. Here, each node maintains the network topology information up to *m* hops. The different existing hybrid protocols are presented below.

7.6.1 Core Extraction Distributed Ad Hoc Routing Protocol

Core extraction distributed ad hoc routing (CEDAR) [17] integrates routing and support for QoS. It is based on extracting core nodes (also called as dominator nodes) in the network, which together approximate the minimum dominating set. A dominating set (DS) of a graph is defined as a set of nodes in the graph such that every node in the graph is either present in the DS or is a neighbor of some node present in the DS. There exists at least one core node within three hops. The DS of the least cardinality in a graph is called the minimum dominating set. Nodes that choose a core node as their dominating node are called core member nodes of the core node concerned. The path between two core nodes is termed a virtual link. CEDAR employs a distributed algorithm to select core nodes. The selection of core nodes represents the core extraction phase.

CEDAR uses the core broadcast mechanism to transmit any packet throughout the network in the unicast mode, involving as minimum number of nodes as possible. These nodes that take part in the core broadcast process are called core

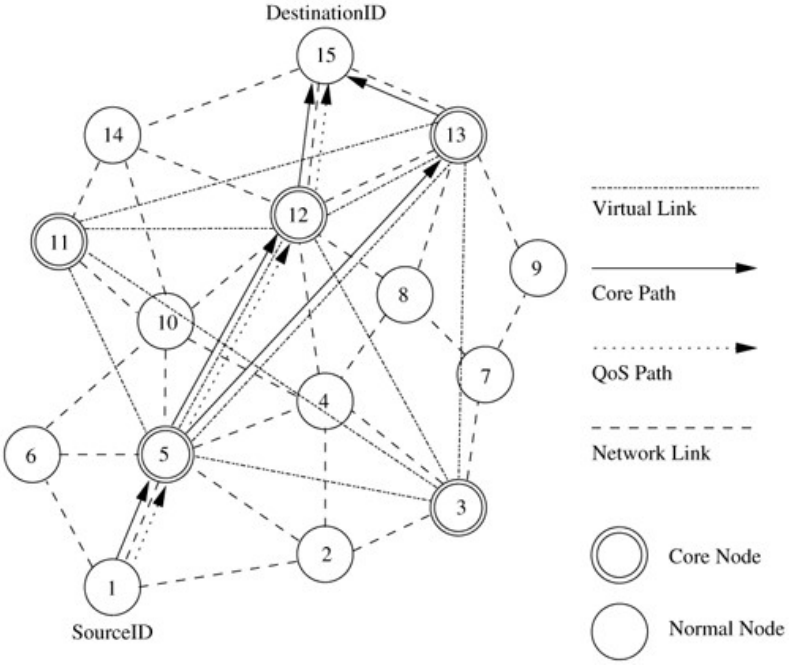
nodes. In order to carry out a core broadcast efficiently, each core node must know about its neighboring core nodes. The transfer of information about neighboring core nodes results in significant control overhead at high mobility. When a core node to which many nodes are attached moves away from them, each node has to reselect a new core node. The selection of core nodes, which is similar to the distributed leader election process, involves substantial control overhead.

Each core node maintains its neighborhood local topology information. CEDAR employs an efficient link state propagation mechanism in which information regarding the presence of high bandwidth and stable links is propagated to several more nodes, compared to the propagation of information regarding low bandwidth and unstable links, which is suppressed and kept local. To propagate link information, slow-moving increase-waves and fast-moving decrease-waves are used. An increase-wave carrying update information is originated when the bandwidth on the link concerned increases above a certain threshold. The fast-moving decrease-waves are propagated in order to quickly notify the nearby nodes (core nodes which are at most separated by three hops) about the reduction in available bandwidth. As bandwidth increase information moves slowly, only stable high-bandwidth link state information traverses long distances. If the high-bandwidth link is unstable, then the corresponding increase-wave is overtaken by fast-moving decrease-waves which represent the decrease in available bandwidth on the corresponding link. These waves are very adaptive to the dynamic topology of ad hoc wireless networks. Increase- and decrease-waves are initiated only when changes in link capacity cross certain thresholds, that is, only when there is a significant change in link capacity. Fast-moving decrease-waves are prevented from moving across the entire network, thereby suppressing low bandwidth unstable information to the local nodes only. Route establishment in CEDAR is carried out in two phases. The first phase finds a core path from the source to the destination. The core path is defined as a path from the dominator of the source node (source core) to the dominator of the destination node (destination core). In the second phase, a QoS feasible path is found over the core path. A node initiates a *RouteRequest* if the destination is not in the local topology table of its core node; otherwise, the path is immediately established. For establishing a route, the source core initiates a core broadcast in which the *RouteRequest* is sent to all neighboring core nodes as unicast data. Each of these recipient core nodes in turn forwards the *RouteRequest* to its neighboring core nodes if the destination is not its core member. A core node which has the destination node as its core member replies to the source core. Once the core path is established, a path with the required QoS support is then chosen.

To find a path that can provide the required QoS, the source core first finds a path to the domain of the farthest possible core node in the core path, which can provide the bandwidth required. Among the available paths to this domain, the source core chooses the shortest-widest path (shortest path with highest bandwidth). Assume *MidCore* is the farthest possible core node found by the source core. In the next iteration, *MidCore* becomes the new source core and finds another *MidCore* node that satisfies the QoS support requirements. This iterative process repeats until a path to the destination with the required bandwidth is found. If no feasible path is available, the source node is informed about the non-availability of a QoS path.

Consider Figure 7.24 where the source is node 1 and the destination is node 15. The core nodes in the network are nodes 3, 5, 11, 12, and 13. In this figure, node 5 is the dominator of nodes 1 and 6. Similarly, node 12 is the dominator of node 15. When node 1 initiates a *RouteRequest* to be flooded throughout the network, it intimates its core node the <source id, destination id> pair information. If the core node 5 does not have any information about the dominator of node 15, which is the destination node, it initiates a core broadcast. Due to this, all nearby core nodes receive the request in the unicast transmission mode. This unicast transmission is done on the virtual links. For core node 5, the virtual link with core node 3 comprises of the links 5-2 and 2-3, while the virtual link between core nodes 5 and 13 is represented by path 5-4-8-13. When a core node receives the core broadcast message, it checks whether the destination is its core member. A core node having the destination as one of its core members replies to the source core node. In our case, core node 12 replies to core node 5. The path between core nodes 12 and 5 constitutes a core path. Once a core path is established, the feasibility of the path in terms of the availability of the required bandwidth is checked. If the required bandwidth is available on the path, the connection is established; otherwise, the core path is rejected.

Figure 7.24. Route establishment in CEDAR.

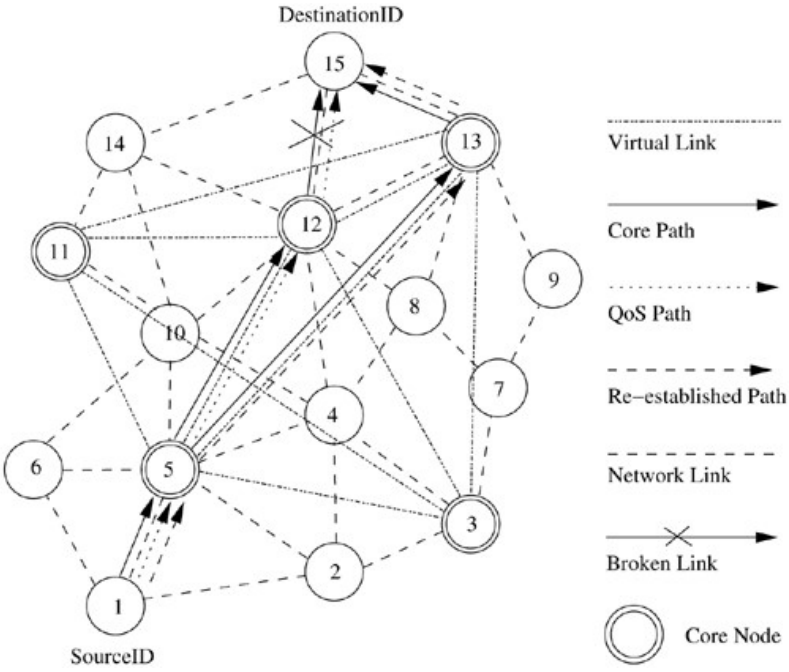


CEDAR attempts to repair a broken route locally when a path break occurs. When a link u-v on the path from source to the destination fails, node u sends back a notification to the source and starts recomputation of a route from itself to the destination node. Until the route recomputation gets completed, node u drops every subsequent packet it receives. Once the source node receives the notification sent by node u, it immediately stops transmitting packets belonging to the corresponding flow, and starts computing a new route to the destination. If the link break occurs near the source, route recomputation at node u may take a long time, but the notification sent by node u reaches the source node very rapidly and prevents large packet losses. If the broken link is very close to the destination, the notification sent by node u might take a longer time to reach the

source, but the route recomputation time at node u is small and hence large packet losses are again prevented. If the link break occurs somewhere near the middle of the path, then both the local route recomputation mechanism and the route break notification mechanism are not fast enough, and hence a considerable amount of packet loss occurs in this case.

Consider the network topology shown in Figure 7.25. When the link between nodes 12 and 15 breaks, node 12 tries to reconnect to the destination using an alternate path that satisfies the bandwidth requirement. It also notifies the source node about the link break. The source node tries to reconnect to the destination by reinitiating the route establishment process. In case node 12 does not have any other feasible path, then the alternate path 1-5-4-8-13-15 found by the source node is used for the further routing of packets.

Figure 7.25. Route maintenance in CEDAR.



Advantages and Disadvantages

The main advantage of CEDAR is that it performs both routing and QoS path computation very efficiently with the help of core nodes. The increase- and decrease-waves help in appropriate propagation of the stable high-bandwidth link information and the unstable low-bandwidth link information, respectively. Core broadcasts provide a reliable mechanism for establishing paths with QoS support. A disadvantage of this protocol is that since route computation is carried out at the core nodes only, the movement of the core nodes adversely affects the performance of the protocol. Also, the core node update information could cause a significant amount of control overhead.

7.7 ROUTING PROTOCOLS WITH EFFICIENT FLOODING MECHANISMS

Many of the existing on-demand routing protocols employ flooding of *RouteRequest* packets in order to obtain a feasible path with the required

packet-forwarding constraints. Flooding of control packets results in a significant amount of redundancy, wastage of bandwidth, increase in number of collisions, and broadcast storms[1] at times of frequent topological changes. Existing routing protocols that employ efficient flooding mechanisms to counter the requirement of flooding include preferred link-based routing (PLBR) protocols [20] and optimized link state routing (OLSR) protocol [21]. The former belongs to the on-demand routing protocols category and the latter belongs to the table-driven routing protocols category. These protocols utilize algorithms that require a minimum number of transmissions in order to flood the entire network.

[1] Broadcast storm refers to the presence/origination of a large number of broadcast control packets for routing due to the high topological instability occurring in the network as a result of mobility.

7.7.1 Preferred Link-Based Routing Protocols

SSA [15] uses the preferred link approach in an implicit manner by processing a *RouteRequest* packet only if it is received through a strong link. Wired networks also employ preferred links mechanisms [22], but restrict themselves by selecting a single preferred link, based on heuristics that satisfy multiple constraints, for example, minimum cost and least delay required by the route. In ad hoc networks, the single preferred link model is not suitable due to reasons such as lack of topology information, continuously changing link characteristics, broadcast nature of the radio channel, and mobility of nodes.

Sisodia *et al.* proposed two algorithms in [20] known as preferred link-based routing (PLBR) protocols that employ a different approach in which a node selects a subset of nodes from its neighbors list (*NL*). This subset is referred to as the *preferred list* (*PL*). Selection of this subset may be based on link or node characteristics. Every *RouteRequest* packet carries the list of a selected subset of neighbors. All neighbors receive *RouteRequest* packets because of the broadcast radio channel, but only neighbors present in the PL forward them further. The packet is forwarded by *K* neighbors, where *K* is the maximum number of neighbors allowed in the PL. PLBR aims to minimize control overhead in the ad hoc wireless network. All nodes operate in the promiscuous mode. Each node maintains information about its neighbors and their neighbors in a table called neighbor's neighbor table (*NNT*). It periodically transmits a *beacon* containing the changed neighbor's information. PLBR has three main phases: route establishment, route selection, and route maintenance.

The route establishment phase starts when a source node (Src) receives packets from the application layer, meant for a destination node (Dest) for which no route is currently available. If Dest is in Src's *NNT*, the route is established directly. Otherwise, Src transmits a *RouteRequest* packet containing the source node's address (SrcID), destination node's address (DestID), a unique sequence number (SeqNum) (which prevents formation of loops and forwarding of multiple copies of the same *RouteRequest* packet received from different neighbors), a traversed path (*TP*) (containing the list of nodes through which the packet has traversed so far and the weight assigned to the associated links), and a PL. It also contains a time to live (*TTL*) field that is used to avoid packets being present in the network forever, and a *NoDelay* flag, the use of which will be described later in this section. Before forwarding a *RouteRequest* packet,

each eligible node recomputes the preferred list table (*PLT*) that contains the list of neighbors in the order of preference. The node inserts the first *K* entries of the *PLT*onto the *PL* field of the *RouteRequest* packet (*K* is a global parameter that indicates the maximum size of *PL*). The old *PL* of a received packet is replaced every time with a new *PL* by the forwarding node. A node is eligible for forwarding a *RouteRequest* only if it satisfies all the following criteria: the node ID must be present in the received *RouteRequest* packet's *PL*, the *RouteRequest* packet must not have been already forwarded by the node, and the *TTL* on the packet must be greater than zero. If Dest is in the eligible node's *NNT*, the *RouteRequest* is forwarded as a unicast packet to the neighbor, which might either be Dest or whose *NL* contains Dest. If there are multiple neighbors whose *NL* have Dest, the *RouteRequest* is forwarded to only one randomly selected neighbor. Otherwise, the packet is broadcast with a new *PL* computed from the node's *NNT*. *PLT* is computed by means of one of the two algorithms discussed later in this section. If the computed *PLT* is empty, that is, there are no eligible neighbors, the *RouteRequest* packet is discarded and marked as *sent*. If the *RouteRequest* packet reaches the destination, the route is selected by the route selection procedure given below.

When multiple *RouteRequest* packets reach Dest, the route selection procedure selects the best route among them. The criterion for selecting the best route can be the shortest path, or the least delay path, or the most stable path. Dest starts a timer after receiving the first *RouteRequest* packet. The timer expires after a certain *RouteSelectWait* period, after which no more *RouteRequest* packets would be accepted. From the received *RouteRequest* packets, a route is selected as follows.

For every *RouteRequest i* that reached Dest during the *RouteSelectWait* period, Max(W_{min}^{i}) is selected, where W_{min}^{i} is the minimum weight of a link in the path followed by *i*. If two or more paths have the same value for *Max* (W_{min}^{i}), the shortest path is selected. After selecting a route, all subsequent*RouteRequest* packets from the same Src with a SeqNum less than or equal to the SeqNum of the selected *RouteRequest* are discarded. If the *NoDelay* flag is set, the route selection procedure is omitted and *TP* of the first *RouteRequest* reaching the Dest is selected as the route. The *NoDelay* flag can be set if a fast connection setup is needed.

Mobility of nodes causes frequent path breaks that should be locally repaired to reduce broadcast of the*RouteRequest*. The local route repair broadcast mechanism used in ABR [14] has a high failure rate due to the use of restricted *TTL*, which increases the average delay in route reestablishment. PLBR uses a quick route repair mechanism which bypasses the down link (Dest side) node from the broken path, using information about the next two hops from *NNT*.

Algorithms for Preferred Links Computation

Two different algorithms have been proposed by Sisodia *et al.* in [20], for finding preferred links. The first algorithm selects the route based on degree of neighbor nodes (degree of a node is the number of neighbors). Preference is given to nodes whose neighbor degree is higher. As higher degree neighbors cover more nodes, only a few of them are required to cover all the nodes of

the *NNT*. This reduces the number of broadcasts. The second algorithm gives preference to stable links. Links are not explicitly classified as stable or unstable. The notion of stability is based on the weight given to links.

Neighbor Degree-Based Preferred Link Algorithm (NDPL)

Let d be the node that calculates the preferred list table *PLT*. *TP* is the traversed path and OLD_{PL} is the preferred list of the received *RouteRequest* packet. The *NNT* of node d is denoted by NNT_d. $N(i)$ denotes the neighbors of node i and itself. Include list (*INL*) is a set containing all neighbors reachable by transmitting the *RouteRequest* packet after execution of the algorithm, and the exclude list (*EXL*) is a set containing all neighbors that are unreachable by transmitting the *RouteRequest* packet after execution of the algorithm.

1. In this step, node d marks the nodes that are not eligible for further forwarding the *RouteRequest*packet.
    1. If a node i of *TP* is a neighbor of node d, mark all neighbors of i as reachable, that is, add $N(i)$ to*INL*.
    2. If a node i of OLD_{PL} is a neighbor of node d, and $i < d$, mark all neighbors of node i as reachable, that is, include $N(i)$ in *INL*.
    3. If neighbor i of node d has a neighbor n present in *TP*, mark all neighbors of i as reachable by adding $N(i)$ to *INL*.
    4. If neighbor i of node d has a neighbor n present in OLD_{PL}, and $n < d$, here again add $N(i)$ to *INL*, thereby marking all neighbors of node i as reachable.
2. If neighbor i of node d is not in *INL*, put i in preferred list table *PLT* and mark all neighbors of i as reachable. If i is present in *INL*, mark the neighbors of i as unreachable by adding them to *EXL*, as$N(i)$ may not be included in this step. Here neighbors i of d are processed in decreasing order of their degrees. After execution of this step, the *RouteRequest* is guaranteed to reach all neighbors of$d$. If *EXL* is not empty, some neighbor's neighbors n of node d are currently unreachable, and they are included in the next step.
3. If neighbor i of d has a neighbor n present in *EXL*, put i in *PLT* and mark all neighbors of i as reachable. Delete all neighbors of i from *EXL*. Neighbors are processed in decreasing order of their degrees. After execution of this step, all the nodes in NNT_d are reachable. Apply reduction steps to remove overlapping neighbors from *PLT* without compromising on reachability.
4. Reduction steps are applied here in order to remove overlapping neighbors from *PLT* without compromising on reachability.
    1. Remove each neighbor i from *PLT* if $N(i)$ is covered by remaining neighbors of *PLT*. Here the minimum degree neighbor is selected every time.
    2. Remove neighbor i from *PLT* whose $N(i)$ is covered by node d itself.

Weight-Based Preferred Link Algorithm (WBPL)

In this algorithm, a node finds the preferred links based on stability, which is indicated by a weight, which in turn is based on its neighbors' temporal and spatial stability.

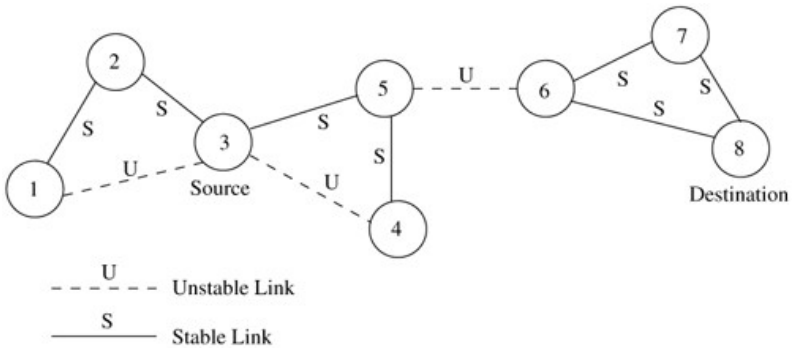
1. Let $BCnt_i$ be the count of *beacon*s received from a neighbor i and TH_{bcon} is the number of beacons generated during a time period equal to that required to cover twice the transmission range ($TH_{bcon} = \frac{2 \times transmission\ range}{maximum\ velocity \times period\ of\ beacon}$). Weight given to i based on time stability (WT^i_{time}) is

$$WT_{time} = \begin{cases} 1 & \text{if } BCnt_i > TH_{bcon} \\ BCnt_i / TH_{bcon} & \text{otherwise.} \end{cases}$$

2. Estimate the distance (D^i_{Est}) to i from the received power of the last few packets using appropriate propagation models. The weight based on spatial stability is $WT^i_{spatial} = \frac{R - D_{Est}}{R}$.
3. The weight assigned to the link i is the combined weight given to time stability and spatial stability. $W_i = WT^i_{time} + WT^i_{spatial}$.
4. Arrange the neighbors in a non-increasing order of their weights. The nodes are put into the *PLT* in this order.
5. If a link is overloaded, delete the associated neighbor from *PLT*. Execute *Step* 1 of NDPL and delete $\forall i,\ i \in PLT \cap i \in INL$. Also, delete those neighbors from *PLT* that satisfy *Step* 4 of NDPL.

Consider, for example, Figure 7.29, where the node 3 is the source and node 8 is the destination. *S* and *U* denote stable and unstable links. In WBPL and NDPL, the source that initiates that *RouteRequest* as Dest is not present in *NNT* and computes the preferred link table (*PLT*). Let $K = 2$ be the preferred list'ssize. In NDPL, after *Step* 2 the *PLT* becomes {5, 1}, and after *Step* 3 also the *PLT* remains {5, 1}. In reduction *Step* 4*b*, neighbor 1 is deleted from *PLT* and hence node 3 sends the *RouteRequest* only to neighbor 5. In WBPL, the weights are assigned to all neighbors according to *Steps* 1, 2, and 3, and all neighbors are in *PLT*. In *Step* 5, neighbors 1, 4, and 2 are deleted from *PLT* due to *Step* 4*a* and 4*b* ofNDPL and hence only node 5 remains. Now the *RouteRequest* can be sent as a unicast packet to avoid broadcast. If it is broadcast, all the nodes receive the packet, but only node 5 can further forward it. As Dest 8 is present in node 5's *NNT*, it directly sends it to node 6 which forwards it to Dest. Here only three packets are transmitted for finding the route and the path length is 3. Now consider SSA. After broadcasts by nodes 3 and 5, the *RouteRequest* packet reaches node 6, where it is rejected and hence the *RouteRequest* fails to find a route. After timeout, it sets a flag indicating processed by *all* and hence finds the same route as WBPL and NDPL.

Figure 7.29. Example network. ***Reproduced with permission from*** **[20], © Korean Institute of Communication Sciences, 2002.**



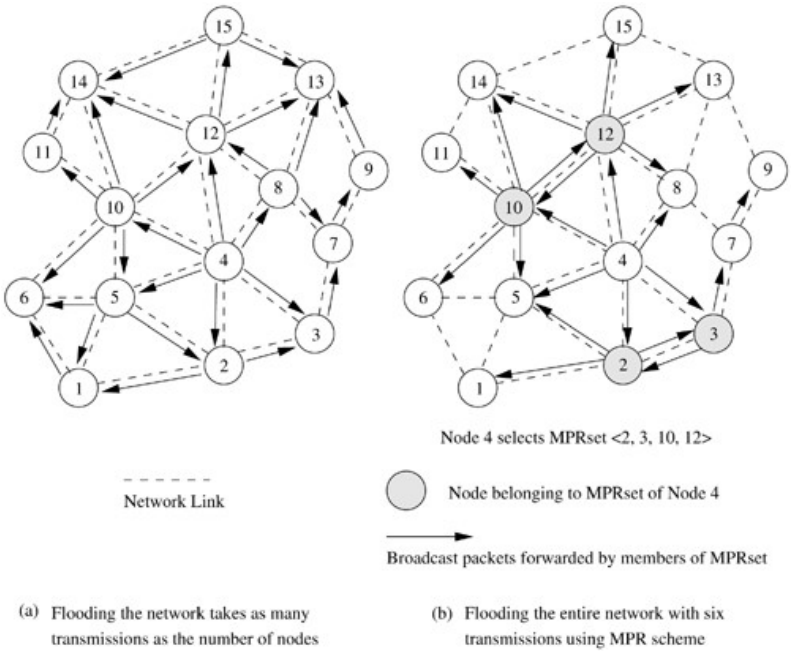
The efficient flooding mechanism employed in this protocol minimizes the broadcast storm problem prevalent in on-demand routing protocols. Hence this protocol has higher scalability compared to other on-demand routing protocols. The reduction in control overhead results in a decrease in the number of collisions and improvement in the efficiency of the protocol. PLBR achieves bandwidth efficiency at the cost of increased computation.
Both NDPL and WBPL are computationally more complex than other *RouteRequest* forwarding schemes.

7.7.2 Optimized Link State Routing

The optimized link state routing (OLSR) protocol [21] is a proactive routing protocol that employs an efficient link state packet forwarding mechanism called *multipoint relaying*. This protocol optimizes the pure link state routing protocol. Optimizations are done in two ways: by reducing the size of the control packets and by reducing the number of links that are used for forwarding the link state packets. The reduction in the size of link state packets is made by declaring only a subset of the links in the link state updates. This subset of links or neighbors that are designated for link state updates and are assigned the responsibility of packet forwarding are called *multipoint relays.* The optimization by the use of *multipoint relaying* facilitates periodic link state updates. The link state update mechanism does not generate any other control packet when a link breaks or when a link is newly added. The link state update optimization achieves higher efficiency when operating in highly dense networks. Figure 7.30 (a) shows the number of message transmissions required when the typical flooding-based approach is employed. In this case, the number of message transmissions is approximately equal to the number of nodes that constitute the network. The set consisting of nodes that are *multipoint relays* is referred to as *MPRset.* Each node (say, *P*) in the network selects an *MPRset* that processes and forwards every link state packet that node *P* originates. The neighbor nodes that do not belong to the *MPRset* process the link state packets originated by node *P* but do not forward them. Similarly, each node maintains a subset of neighbors called *MPR selectors,* which is nothing but the set of neighbors that have selected the node as a *multipoint relay.* A node forwards packets that are received from nodes belonging to its*MPRSelector* set. The members of both *MPRset* and *MPRSelectors* keep changing over time. The members of the *MPRset* of a node are selected in such a manner that every node in the node's two-hop neighborhood has a bidirectional link with the node. The selection of nodes that constitute the *MPRset*significantly affects the performance of OLSR because a node calculates routes to all destinations only through the members of its *MPRset.* Every node periodically broadcasts its *MPRSelector* set to nodes in its immediate neighborhood. In order to decide on the membership of the nodes in the *MPRset,* a node periodically sends *Hello* messages that contain the list of neighbors with which the node has bidirectional links and the list of neighbors whose transmissions were received in the recent past but with whom bidirectional links have not yet been confirmed. The nodes that receive this *Hello* packet update their own two-hop topology tables. The selection of *multipoint relays* is also indicated in the*Hello* packet. A data structure called *neighbor table* is used to store the list of neighbors, the two-hop neighbors, and the status of neighbor nodes. The

neighbor nodes can be in one of the three possible link status states, that is, uni-directional, bidirectional, and *multipoint relay.* In order to remove the stale entries from the *neighbor table,* every entry has an associated timeout value, which, when expired, removes the table entry. Similarly a sequence number is attached with the *MPRset* which gets incremented with every new *MPRset.*

Figure 7.30. An example selection of MPRset in OLSR.



The *MPRset* need not be optimal, and during initialization of the network it may be same as the neighbor set. The smaller the number of nodes in the *MPRset,* the higher the efficiency of protocol compared to link state routing. Every node periodically originates *topology control* (TC) packets that contain topology information with which the routing table is updated. These TC packets contain the*MPRSelector* set of every node and are flooded throughout the network using the *multipoint relaying*mechanism. Every node in the network receives several such TC packets from different nodes, and by using the information contained in the TC packets, the *topology table* is built.
A TC message may be originated by a node earlier than its regular period if there is a change in the *MPRSelector* set after the previous transmission and a minimal time has elapsed after that. An entry in the *topology table*contains a destination node which is the *MPRSelector* and a last-hop node to that destination, which is the node that originates the TC packet. Hence, the routing table maintains routes for all other nodes in the network.

Selection of Multipoint Relay Nodes

Figure 7.30 (b) shows the forwarding of TC packets using the *MPRset* of node 4. In this example, node 4 selects the nodes 2, 3, 10, and 12 as members of its *MPRset.* Forwarding by these nodes makes the TCpackets reach all nodes within the transmitting node's two-hop local topology. The selection of the optimal *MPRset* is NP-complete [21]. In [21], a heuristic has been proposed for selecting the *MPRset.*The notations used in this heuristic are as follows: $N_i(x)$

refers to the i^{th} hop neighbor set of node x and $MPR(x)$ refers to the *MPRset* of node x.

1. $MPR(x) \leftarrow \emptyset$/* Initializing empty *MPRset* */
2. $MPR(x) \leftarrow$ { Those nodes that belong to $N_1(x)$ and which are the only neighbors of nodes in $N_2(x)$ }
3. While there exists some node in $N_2(x)$ which is not covered by $MPR(x)$
    1. For each node in $N_1(x)$, which is not in $MPR(x)$, compute the maximum number of nodes that it covers among the uncovered nodes in the set $N_2(x)$.
    2. Add to $MPR(x)$ the node belonging to $N_1(x)$, for which this number is maximum.

A node updates its *MPRset* whenever it detects a new bidirectional link in its neighborhood or in its two-hop topology, or a bidirectional link gets broken in its neighborhood.

Advantages and Disadvantages

OLSR has several advantages that make it a better choice over other table-driven protocols. It reduces the routing overhead associated with table-driven routing, in addition to reducing the number of broadcasts done.
Hence OLSR has the advantages of low connection setup time and reduced control overhead.