Chapter 2

Routing in mobile ad hoc networks

In this chapter the concept of routing in mobile ad hoc networks (MANETs) is presented. In Section 2.1 the concept of routing is presented. This section explains what routing is as well as when and why it is needed. Section 2.2 describes two basic types of routing algorithms called *link state* and *distance vector* that are assumed to be known in the following sections. After introducing the basic routing protocols the focus shifts to the area of MANET routing. In Section 2.3 a number of routing protocols specifically tailored for the MANET setting are presented.

2.1 Introduction

Routing is the process of passing some data, henceforth referred to as a *message*, along in a network. The message originates from a source host, travels through some intermediate hosts and finally ends up at the destination host. Routing is as such only needed when the communicating parties are not in direct contact and therefore some intermediate parties, called *routers*, are needed to pass along the message between the source and the destination.

Consider a simple network such as the one shown in Figure 2.1. Here

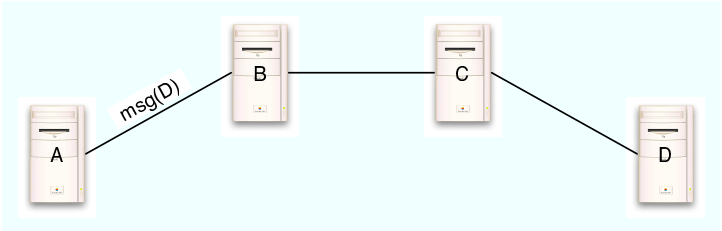


Figure 2.1: Simple routing scenario.

host A wants to send a message to host D but D is not within direct *range* of A, which in this setting means that host A has no direct link to host D. To facilitate the message delivery the hosts B and C are thus used as routers of the message.

In this simple setting every host that acts as a router has only two network interfaces and therefore the act of routing the message becomes very simple; an incoming message is simply retransmitted on the opposite link as it was received on until the destination is reached.

Routing becomes more complicated when the size of the network grows. In a larger network, such as the one shown in Figure 2.2, a lot of questions need to be answered when a message is to be routed:

- How do the hosts acting as routers know which way to send the message?
- What should be done if multiple paths connect the sender and receiver?
- Does an answer to the message *have* to follow the same path as the original message?

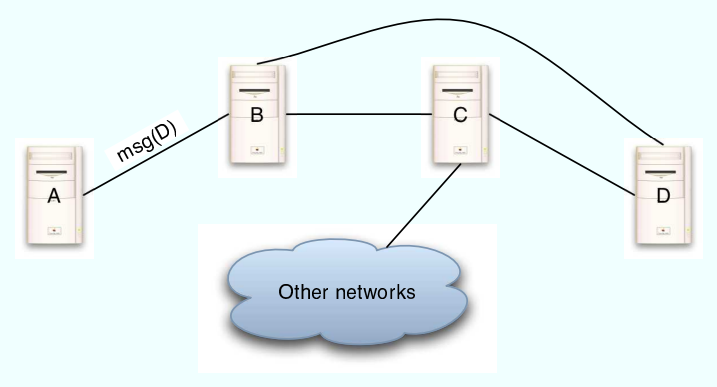


Figure 2.2: A more advanced routing example.

A simple solution would be to *broadcast* every single message. This would mean that any message that a router receives is simply broadcasted on every outgoing link. This way all messages will reach their destination at some point. The problem with this approach is that it will create an obscene amount of unnecessary traffic in the network, and it is thus not very scalable. This is where routing protocols enter the scene. The responsibility of routing protocols is exactly answering the questions posed above.

2.2 Basic routing protocols

In this section the basics of routing protocols are presented along with examples of the two dominating routing algorithms: the link state (LS) approach and the distance vector (DV) approach.

A routing protocol must enable the computers and routers to find a *path* or *route* through the network. Since these protocols must be executed on machines it is necessary to find a way of representing the network inside a machine. An obvious choice is to represent a network as a graph where the routers are the nodes of the graph and the links between them are the edges. Furthermore, the edges can be weighted so that some sort of cost metric can be used in the path determination. This cost metric may be physical distance, so that a link across the Atlantic has a higher cost than a local area link. The link cost may be variable, so that the cost of individual links change to reflect how much bandwidth is used.

Using this graph representation with weighted edges a network can be represented as seen in Figure 2.3. The small example shown in Figure 2.3

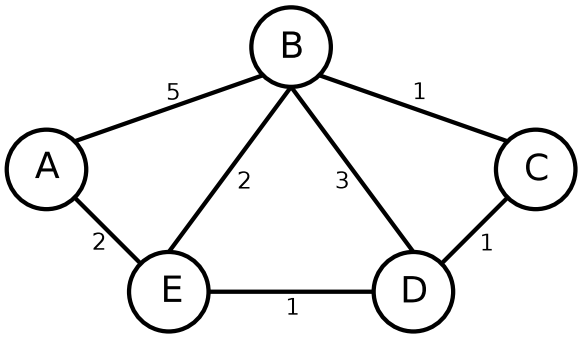


Figure 2.3: An network represented as a weighted graph. Routers are displayed as circles with their names written inside, and adjacent to the edges their weight (cost) is displayed.

will be used extensively in the following sections where the two main classes of routing protocols are discussed.

The main difference between the two classes of routing protocols is whether or not they use *global information*, meaning that every router has complete knowledge of the network. The algorithms using global information are often referred to as link state (LS) algorithms because all nodes need to maintain state information about all the links in the network. An example of such an algorithm is shown in Section 2.2.1. An example of an algorithm that does not rely on global information is the distance vector (DV) algorithm which is described in Section 2.2.2.

2.2.1 Link State

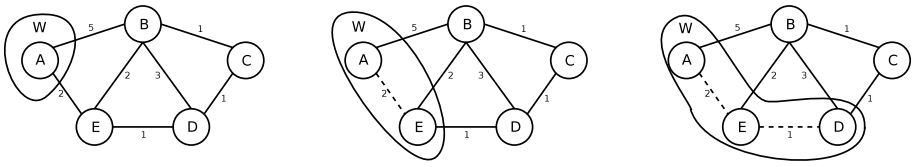
In a LS algorithm all nodes and links with associated weights are known to every single node. This means that the task of finding routes through the network becomes a simple case of finding the *single-source shortest path*. A well known algorithm for finding this single-source shortest path is Dijkstra's algorithm.

Dijkstra's algorithm

Before going into details with the inner workings of Dijkstra's algorithm an informal description is given that should give an intuition as to how it works.

When the algorithm starts a set W is initialised to contain only the source node v. In every step of the algorithm the edge e with the lowest cost connecting W with a node $u \notin W$ is chosen and u is added to the set. This edge e is then the last edge in the shortest path from v to u. The algorithm loops for $n-1$ iterations, where n is the number of nodes in the graph, and then the shortest paths to all destinations have been found. The correctness proof of this algorithm will not be given here, but the interested reader may take a look at the description of Dijkstra's algorithm by Cormen et al. [4, pages 597-598].

An example of the algorithm in action is shown in Figure 2.4 where the first two iterations are depicted.



(a) Initial state. (b) 1st iteration. (c) 2nd iteration.

Figure 2.4: An example run of Dijkstra's algorithm on a small network.

To get an idea of how Dijkstra's algorithm may be implemented a pseudocode implementation is given in Pseudocode 2.1. This pseudocode is based upon the implementation given by Cormen et al. [4, page 595].

In the pseudocode W is the set of nodes to which the shortest path is already known, s is the source node, $D(v)$ is the shortest known distance between s and v, $P(v)$ is the predecessor of v on the shortest path from s to v and $c(u, v)$ is the cost of the edge from u to v. The implementation uses a min-queue Q for storing the nodes to which the shortest path has not yet been found. Inside Q the nodes are ordered by the currently best known

distance to s.

```
1  for all nodes v ≠ s do
2    D(v) ← ∞
3    P(v) ← nil
4  D(s) ← 0
5  W ← ∅
6  for all nodes v do
7    Insert(Q, v)
8  while Q ≠ ∅ do
9    v ← Extract-Min(Q)
10   W ← W ∪ v
11   for each node u adjacent to v do
12     if D(u) > D(v) + c(v, u) then
13       D(u) ← D(v) + c(v, u)
14       Update-Key(Q, u)
```

Pseudocode 2.1: Dijkstra's single-source shortest path algorithm.

Evaluation

To use a LS algorithm like the one described by Dijkstra, every router needs to have complete knowledge of the network. This is accomplished in practice by letting each router broadcast the identities and costs of *all* of its outgoing links to *all* other routers in the network. Every time a link cost changes the information is again broadcasted by the affected routers.

It is obvious that for everything other than very small networks this will lead to a massive amount of control communication just to maintain the routing tables. Furthermore, every router has to recalculate its entire routing table whenever a link cost has changed somewhere in the network. Dijkstra's single-source shortest path algorithm is known to have a running time of $O(m + n \log n)$, where n is the number of nodes and m is the number of edges. This means that classical LS routing will probably not be feasible in networks with mobile nodes, where the topology is constantly changing.

LS has its force in small *stable* networks, where stable means that changes in link costs or link availability are rare. In a stable network the LS algorithm will quickly enter a quiescent state where no calculation of routes is done until the next change in link cost or availability occurs.

2.2.2 Distance Vector

In DV routing no global knowledge is needed. DV protocols are distributed algorithms where the shortest distance to any given node in the network is

calculated in cooperation between the nodes. DV algorithms are based on the Bellman-Ford algorithm that is described by Cormen et al. [4, page 588].

Bellman-Ford's algorithm

The algorithm described here is in fact not the original Bellman-Ford algorithm which is described by Cormen et al. since that version does in fact use global knowledge of the network. This is *not* the case with the version presented here but it does build on the same principles as the original algorithm.

The algorithm is decentralised. This means that no global knowledge is needed before the calculation of the shortest paths can begin. The only information needed by each node in the network is the identities of its neighbours and the costs of its outgoing links.

Each node stores a *distance table* of distances to known nodes. This table contains a column for each neighbour of the node and a row for each known destination. When a new destination is encountered a new row is simply added to this table.

The algorithm works by having nodes send *updates* to their neighbours. An update is a message from a node v that says what the cost of v's shortest path to another node u is. When a node receives such an update it updates its distance table accordingly. If this update means that the node calculates a new shortest path to a destination it sends an update to all of *its* neighbours notifying them of this change. This way the information about shortest paths spread through the network, and after a number of iterations all nodes will know of the best paths to all other nodes. The route calculation is bootstrapped by having all nodes send updates about all of their links to their neighbours at startup.

The algorithm is formally described in Pseudocode 2.2 which is based on the description by Kurose and Ross [13, page 288]. The code shown here is executed at all nodes in the network. In the pseudocode $D^x(v, u)$ is the distance to node v over u in the distance table of node x, and $c(u, v)$ is the cost of the link from u to v. Updates are written $U^x(v, c)$ which means that the update comes from x and says that v can be reached with cost c.

Lines 1–4 of the pseudocode initialises the distance table of the node x and in lines 5–7 the initial updates are sent to all of x's neighbours. After these initial steps the algorithm enters an infinite loop where the node waits for updates or link cost changes of directly connected links.

In the case where an update message has arrived the single destination that the update refers to is updated in the distance table. If the cost of a directly connected link has changed the costs of *all* affected table entries are updated.

When an event has been handled, regardless of whether it was an update or a link cost change, it is checked whether the change resulted in a new

```
1  for all neighbours v do
2    D^x(v,v) = c(x,v)
3    for all neighbours u ≠ v do
4      D^x(u,v) = ∞
5  for all neighbours v do
6    for all neighbours u ≠ v do
7      Send update U^x  (u, D^x(u,u)) to v
8  loop forever
9    Wait for 1) an update from a neighbour, or
10            2) a change in link cost to a neighbour.
11   if an update U^v(u,c) has arrived then
12     D^x(u,v) = c
13   else if link cost c(x,v) has changed by c.
14     for all destinations u
15       D^x(u,v) = D^x(u,v) + c
16   if a new shortest path D^x(u,v) has been found then
17     for all neighbours z do
18       Send update U^x (u, D^x(u,v)) to z
```

Pseudocode 2.2: Bellman-Ford's single-source shortest path algorithm.

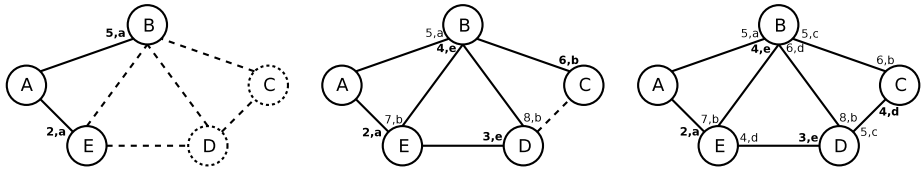
shortest path to some destination. If such new minimums are found updates are sent to all neighbours.

An example run of the algorithm on the network from Figure 2.3 can be seen in Figure 2.5 In this example node A's view of the network together with its distance table is shown. In (a) the initial state before any updates are received is depicted. In (b) node A has received the initial updates from both B and E which has added C and D to its distance table. Finally in (c) A has received an update from E saying that it can reach C with cost 2 and the shortest path to C is thus updated. At (c) node A has entered into the quiescent state and now has complete knowledge of the network until a link cost changes somewhere.

Evaluation

In a stable network, where changes in link cost and availability are few and far between, the DV algorithm will quickly enter into a quiescent state where no control messages are sent, just like it was the case for the LS algorithm. But, in contrast to the LS algorithm, DV also handles mobile networks very well.

DV protocols do not need global knowledge which is an advantage in mobile environments. Furthermore, since updates are only sent to the neighbours of a node, the amount of control traffic generated in DV routing is lower than in LS routing. Also DV does not recompute the *entire* routing



(a) Initial state.

D^A	B	E
B	***5***	∞
E	∞	***2***

(b) 1st iteration.

D^A	B	E
B	5	*4*
C	***6***	∞
D	8	*3*
E	7	**2**

(c) 2nd iteration

D^A	B	E
B	5	**4**
C	6	*4*
D	8	**3**
E	7	**2**

Figure 2.5: An example of distance vector routing. Node A has been chosen as source and the figure thus shows A's view of the network. Under each figure the distance table of node A at that time is displayed.

table every time a link cost changes, and this is a big advantage on small, resource constrained devices like the mobile nodes.

There is a problem with the DV algorithm as it is described here, and that is the *count-to-infinity* problem. This problem occurs when a link cost increases. The problem is that, unless some action is taken to prevent it, a route may pass twice through the same node for a while. This can be prevented by using techniques like poisoned reverse or split-horizon.

2.3 MANET routing protocols

The routing protocols described in Section 2.2 were designed to operate in network environments such as the wired Internet or local area networks. Such networks are fairly stable, meaning that changes in link availability and cost are rare. This is *not* the case in MANETs where possibly all nodes, including the sender and receiver, are mobile and may be moving during communication. This high level of mobility leads to a need for special purpose routing protocols that take the continually changing topology of the network into consideration.

In order to know what to focus on when defining protocols for MANETs it is important to identify the characteristics of these networks. MANETs typically consist of resource-poor, energy constrained devices with limited bandwidth and high error rates. These properties combined with the missing infrastructure and the high mobility of the nodes mean that, according to Belding-Royer [2], the focus should be on the following properties:

- **Minimal control overhead.** Because of the limited energy and bandwidth of the mobile devices it is *very* important to minimise the amount of unnecessary traffic that flows through the network. Control messages used for routing information consume bandwidth, processing power and energy and should therefore be kept at a minimum.
- **Minimal processing overhead.** The mobile devices in MANETs are as mentioned typically resource-poor and as such do not have very large processors. This means, that it is not feasible to use routing algorithms that are computationally complex on these small devices. Complex algorithms require large amounts of CPU cycles to execute which exhausts the energy of the devices.
- **Multihop routing capability.** Because of the missing infrastructure nodes in MANETs must act as routers of messages when the sender and recipient are not within direct transmission range of each other.
- **Dynamic topology maintenance.** The high mobility of the nodes in a MANET makes it essential that the routing protocols are able to quickly adapt to the rapidly changing topology of the network. Furthermore, this topology maintenance must be done with a minimum of overhead, i.e. using a minimum of control messages.
- **Loop prevention.** As it is the case for all routing protocols, loops must be prevented at all costs. Loops are especially destructive in a MANET setting, since bandwidth is scarce and packet processing and forwarding are expensive. Loops in MANETs are thus extremely wasteful of already scarce resources.

With these design goals in mind a number of routing protocols specifically tailored for MANETs have been created. These MANET routing protocols are typically divided up into *pro-active* and *re-active* protocols.

In a pro-active routing scheme every node maintains a routing table of routes to all other nodes in the network. This routing table is updated whenever a change occurs in the network. This means, that when a node needs to send a message to another node, it already has a route to that node in its routing table. Two examples of pro-active routing protocols are described here. These have been chosen because they are representative of the area of pro-active MANET routing and because most other pro-active approaches are based on these two protocols. The first is a distance vector protocol called the destination-sequenced distance vector (DSDV) protocol. DSDV is described in Section 2.3.2. The second pro-active protocol described here is a LS protocol which is called the optimised link state routing (OLSR) protocol. OLSR is presented in Section 2.3.3.

Re-active routing protocols, also called *on-demand* routing protocols, do not maintain routing tables at all times. In a re-active routing scheme a route

to a destination node is discovered when it is needed, i.e. when the source has some data to send. Two examples of re-active routing protocols are presented here beginning with a description of a distance vector protocol called the ad hoc on demand distance vector (AODV) protocol in Section 2.3.4. After AODV comes a description of the dynamic source routing (DSR) protocol in Section 2.3.5. Again these have been chosen because they are representative of the area of re-active MANET routing.

Some protocols utilise both pro- and re-active routing. One of these *hybrid* protocols is the zone routing protocol (ZRP) which is briefly described in Section 2.3.6.

Before going into details about the different MANET routing protocols one of the basic concepts of MANET routing is presented in Section 2.3.1. What is presented here is the neighbourhood discovery mechanism that is used in most MANET routing protocols.

2.3.1 Local connectivity management

One thing that most MANET protocols have in common is that they need to have some sort of mechanism that will allow the discovery of *neighbours* in the network. In a MANET setting a neighbour is a node which is within broadcast range, i.e. a node that can be reached in a single hop.

Neighbour discovery is normally done by having the nodes periodically broadcast control messages; sometimes called `HELLO` messages. These `HELLO` messages are not relayed by the receiving nodes, and therefore only the nodes within transmission range of the sending node will receive these messages.

A `HELLO` message contains information about the neighbours known to the sending node. For each neighbour listed in the message it is also listed whether the link to that neighbour is known to be uni- or bi-directional.

When a node x receives a `HELLO` message from another node y it adds y to its neighbour table (if it does not already exist). Then, based on the information found in the `HELLO` message, the link to y is marked as either uni- of bi-directional:

- If the `HELLO` message contains an entry for x the link between x and y must be bi-directional. x thus updates its neighbour table with that information.

- Otherwise x marks the entry for y as uni-directional.

In any subsequent `HELLO` message sent by x this new information that has been recorded about node y is included.

Besides facilitating the neighbourhood discovery procedure the `HELLO` messages often also contain additional information needed for the routing protocol to function. This is protocol specific and can therefore not be described here.

2.3.2 Destination-Sequenced Distance Vector

The following description of the destination-sequenced distance vector DSDV protocol is based on the original article by Perkins and Bhagwat [15].

To make distance vector routing more attractive in a MANET setting DSDV adds some optimisations to the original distributed Bellman-Ford algorithm that was described in Section 2.2.2.

The main optimisation is, as the name hints at, the addition of sequence numbers in the routing information. Using these sequence numbers DSDV makes sure that no routing loops can occur in the network. This was one of the important design goals of MANET protocols that were listed in Section 2.3.

Using sequence numbers

The are a number of rules as to how the sequence numbers are handled in DSDV.

Each node maintains a counter that represents its current sequence number. This counter starts at zero and is incremented by two whenever it is updated. This means that a sequence number set by the node itself will always be an even number. The sequence number of a node is propagated through the network in the update messages that are sent to the node's neighbours. Every time an update message is sent the sending node increments its sequence number and prefixes this to the message.

When a node receives an update message it thus receives the current sequence number of the sending node. This information is stored in the receiving nodes route table and is passed further along in the network in any subsequent update messages sent regarding routes to that destination. This way the sequence number set by the destination node is stamped on every route to that node. Hence the name *destination-sequenced* distance vector protocol.

Using sequence numbers an update message contains the following elements for each route: A destination node, a cost of the route, a next-hop node, and the latest known destination sequence number. When an update to a route to destination node x arrives the following rules apply:

- If the sequence number of the updated route is higher (i.e. more recent) than the one currently stored in the route table the updated route is chosen.

- If the sequence numbers are the same the route with the lowest cost is chosen.

Neighbours monitor each other to see if any links break (e.g. when neighbouring nodes move out of transmission range). When such a link break

happens the node that discovers the breakage sends an update to its neighbours stating that the cost to the node that has disappeared is now ∞. The destination sequence number in this update is set to be the last known sequence number plus one. The sequence number is thus an odd number whenever a node discovers a link breakage. That only one is added to the sequence number means that any following updates sent by the disappeared node will automatically supersede this sequence number.

Following these simple rules is enough to make the DSDV protocol loop free. The complete proof will not be presented here but the interested reader is referred to the article by Perkins and Bhagwat [15]. To gain some intuition as to why the sequence numbers make DSDV loop free consider how the sequence numbers are distributed throughout the network. Sequence numbers are changed in the following two ways:

1. **When a link breaks.** In this case the sequence number is changed by a neighbouring node. Link breakage can not form a loop though for obvious reasons so this will not be discussed further.

2. **When a node sends an update message.** Here the node changes its own sequence number and sends information about it to its neighbours. These neighbours pass this information on to their neighbours and so on.

Looking at the second case it can be seen that the updates of sequence numbers propagates through the network in a way such that the closer you get to a given node to more recent the last known sequence number of that node is. This means that, on a path from a node y to a destination node x, the nodes in the shortest path route will form an increasing sequence of sequence numbers. When routes are updated the route with the most recent sequence number will always be selected which in turn means that a next-hop node that is closer to the destination is always chosen. When the next-hop is always closer to the destination node no loop can be formed.

Sending updates

Some further optimisations to improve the performance of the protocol in a MANET setting are done in the way that update messages are sent when routes change. Remember from Section 2.2.2 that a DV protocol builds its routing tables by exchanging update messages between neighbours. There are two types of updates that are sent in DSDV; full and incremental updates. Full updates, which contain information about all routes known by the sending node, are sent relatively infrequently. Incremental updates, which contain only those routes that have changed since the last full update, are sent regularly at some set interval.

By splitting the updates up into full and incremental instead of just sending the entire routing table in each message a lot of control traffic is

avoided. DSDV contains a number of rules as to when updates are sent. These rules are presented here:

- Full updates are sent in some relatively large interval.
- Incremental updates are sent frequently.
- Full updates are allowed to use multiple network protocol data units (NPDUs) whereas incremental updates may only use a single NPDU. When there are too many incremental changes to fit inside a single NPDU a full update is sent instead.
- When an update to a route is received different actions are taken depending on what *kind* of information it contains.
  - If a route to a previously unknown node is received this information is scheduled for *immediate* update - meaning that an incremental update is sent as soon as possible.
  - If a route to a known node is improved (i.e. has a lower cost) this information is scheduled to be sent in the next incremental update.
  - If a route to a known host has a more recent sequence number but an unaltered cost this update is scheduled to be sent in the next incremental update *if there is enough room in the single NPDU that the incremental update must stay within.*

By prioritising the information that goes into updates in this way the DSDV protocol tries to postpone the sending of full updates.

The protocol as described so far has one big problem and that is *routing fluctuations*. Because routes are selected based on both their cost and their sequence number a scenario where a node repeatedly switches between a couple of routes could easily occur. Consider for example a scenario as the one seen in Figure 2.6.

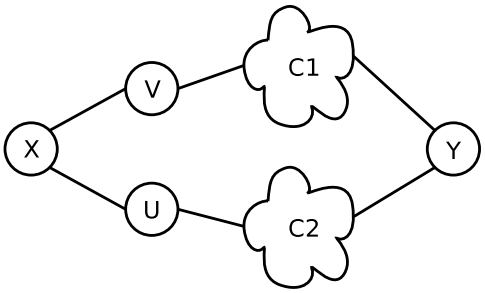


Figure 2.6: Routing fluctuations in DSDV.

The clouds in the figure symbolise a cluster of nodes. Node x has two routes to y that goes through either u or v. When y sends updates, and therefore changes its sequence number, these updates have to propagate through $c1$ and $c2$ respectively before they hit v and u. Assume that the path from x to y through v is more expensive than the path through u. Furthermore assume that x always receives information about the updated sequence number of y from node v first. Now x selects the route through v and schedules this update to be sent in the next incremental update. Then the update message from u arrives with the same sequence number and a better cost and this new route is chosen and advertised. This can go on forever creating an excessive amount of control traffic.

The fluctuation problem is solved in DSDV by introducing delays in the propagation of routing information. If the cost to a destination changes this information is scheduled for advertisement at a time depending on the average settling time for that destination.

If this approach was used in Figure 2.6 the update received from v would never be advertised by x because the update from u would have invalidated it.

2.3.3 Optimised Link State Routing

Optimised link state routing (OLSR) was first described in the article by Jacquet et al. [10].

OLSR is, as the name implies, a link state routing protocol. It has been specifically designed to be effective in an environment with a dense population of mobile devices that communicate frequently with each other.

The optimisation introduced by OLSR is the introduction of the multi point relay (MPR) set. The MPR set is a subset of one-hop neighbours of a node that is used for routing the messages of that node. The nodes in the MPR set are called *MPR selectors*.

The Multipoint Relay Set

The MPR set is selected independently by each node as a subset of its neighbouring nodes. The MPR set is selected among the neighbour nodes in such a manner that the set covers all the nodes that are two hops away. This is illustrated in Figure 2.7.

The MPR set does not have to be optimal; i.e. the set does not have to contain the minimum number of nodes needed to cover all two-hop neighbours.

Each node stores a list of its one-hop neighbours and a list of its two-hop neighbours. This information is collected from the `HELLO` messages that all nodes periodically broadcast to their one-hop neighbours. A `HELLO` message contains information about the one-hop neighbours of the sending node and

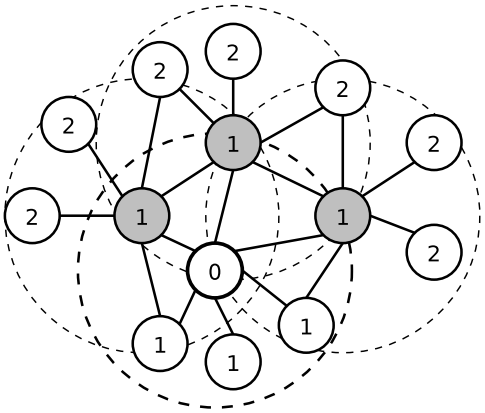


Figure 2.7: An example of a multipoint relay set. Each node in the network is labelled with its distance from the source node. The dashed circles show the range of the source node and of the nodes in the MPR set.

based on this information every node can gain information about all of its one- and two-hop neighbours. With this information in hand calculating the MPR set can be done by executing the following simple algorithm:

1. Insert the one-hop neighbours into a max-heap ordered by the number of two-hop neighbours that they cover.
2. Sort the list of two-hop neighbours ordered by some unique identifier (e.g address).
3. Repeatedly do:
   (a) Extract the max element m from the heap and insert it into the MPR set.
   (b) Mark the set s of two-hop nodes that are covered by m as covered.
   (c) Decrease the keys of all nodes in the heap that have neighbours among the nodes in s.

   until all two-hop nodes are covered.

Routing with the MPR set

The `HELLO` messages are used to build the list of one and two-hop neighbours which are used to compute the MPR set. These message are only broadcasted to one-hop neighbours, and as such they are not enough to create a routing table for the entire network. A different kind of control message is needed to send routing information through the network. This is the topology control (TC) message.

TC messages are sent periodically to all nodes in the network. Each such message contains a list of neighbours that have selected the sending node as a MPR selector. When the TC message is sent the sending node broadcasts it to its one-hop neighbours but *only* the nodes that are in the MPR set of the sending node rebroadcast the message. This process goes on until the entire network has been flooded. Since the MPR set covers the two-hop nodes of the sending node all two-hop nodes will be reached; and by recursion it is trivial to see that all nodes will receive the TC message.

When a node receives these TC messages it calculates its routing table in a way similar to the LS algorithm that is described in Section 2.2.1.

What remains to be shown is that even though a node only sends information about those neighbours that have selected it as an MPR selector the entire network is still covered. Again, as it was the case for the TC messages, it is trivial to see that this is the case because of the way that the MPR set has been selected. The MPR selectors cover the entire network and since information about these nodes is used in the routing tables routes to all destinations are available.

Evaluation

Using the MPR sets to route information through the network saves a lot of unnecessary traffic. Consider a TC message: When such a message is broadcasted throughout the network only the MPR selectors of the sending node rebroadcast the message. Since the MPR nodes cover the network the message will reach all nodes. That only MPR nodes rebroadcast the message saves a lot of duplicate traffic that would be generated in a regular network flooding.

Furthermore the control messages are smaller than in the regular link state protocol, since only information about MPR selectors is included in the TC messages.

OLSR is computationally demanding for the individual nodes especially in an environment with a high level of mobility. Each node has to compute both its MPR set and routing table upon changes in the topology. Both of these operations are demanding; the MPR set calculation is a $O\left(n_1^2 \log n_1\right)$ operation where n_1 is the number of one-hop neighbours. The calculation of the routing table is a $O\left(m + n \log n\right)$ where n is the number of MPR selectors and m is the number of edges connecting these MPR selectors.

Because of the high cost of maintaining the routing information in OLSR this protocol is only usable in environments with a dense population of nodes that communicate frequently. In such an environment the high maintenance cost will be outweighed by the advantage of having routing information immediately available.

2.3.4 Ad Hoc On-Demand Distance Vector

The ad hoc on demand distance vector (AODV) protocol is a re-active routing protocol, meaning that routes are acquired when they are needed. The following description of AODV is based on the original article by Perkins and Royer [16].

The following description of AODV assumes that only symmetrical links are used. To ensure that this is the case a neighbourhood discovery mechanism like the one described in Section 2.3.1 is used.

AODV uses, like DSDV, destination sequence numbers to avoid routing loops.

Path discovery

When a node needs to send some data to another node in a re-active routing protocol a *path discovery* mechanism is triggered. How this path discovery is done in AODV is described here.

When a node x needs to send some data to another node y, that it does not already have a route for, it sends a route request (RREQ) message to its neighbours. The RREQ contains the following information:

- *Source address*: The address of the node that initiated the RREQ.
- *Source seq. no.*: The current sequence number of the source node.
- *Broadcast id*: A unique id of the current RREQ. This counter is incremented every time a RREQ is sent.
- *Destination addr.*: The address of the node that the source wishes to establish a route to.
- *Destination seq. no.*: The last known sequence number of the destination node.
- *Hop count*: The number of hops traversed so far. This counter is incremented by each intermediate node when forwarding the RREQ.

The pair *<source address, broadcast id>* uniquely identifies a RREQ. Whenever a node receives a RREQ, it can thus check to see if it has already seen it before and in that case the duplicate RREQ is simply discarded.

When a node receives a RREQ message from one of its neighbours it can take one of two actions. If the receiving node has a path to the destination node in its routing table, and if that path has a sequence number equal to or greater than the destination sequence number of the RREQ, it responds to the RREQ by sending a route reply (RREP) message back to the source. If, on the other hand, it does not have a recent route to the destination, it broadcasts the RREQ to its neighbours with the hop count increased by one.

This is illustrated in Figure 2.8. In this figure, the source node S wants to discover a route to the destination node D. Three pieces of information are shown for each RREQ in the figure; the source address, the destination sequence number and the hop count. It can be seen that only the hop count is altered as the message is forwarded through the network.

Eventually the RREQ arrives at node X who has a recent route to D (in fact D is one of its neighbours). X will then send a RREP back to S to inform it of the discovered route.

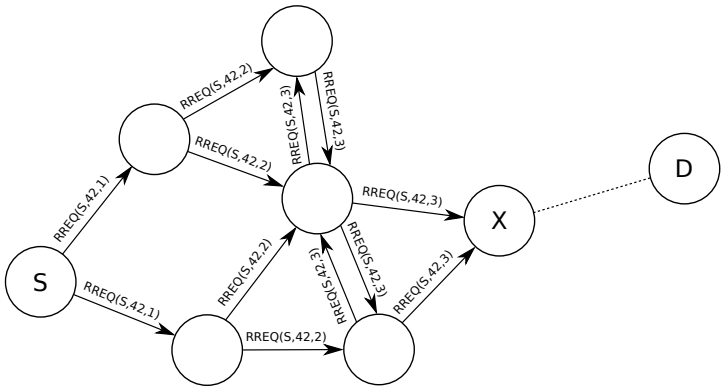


Figure 2.8: Propagation of route request (RREQ) messages through a network.

When the intermediate nodes receive the RREQ message they record the address of the neighbour from whom they received the message. In this way the intermediate nodes form a *reverse path* that can be used to route a potential RREP back to the source node. This propagation of the RREP message is shown in Figure 2.9.

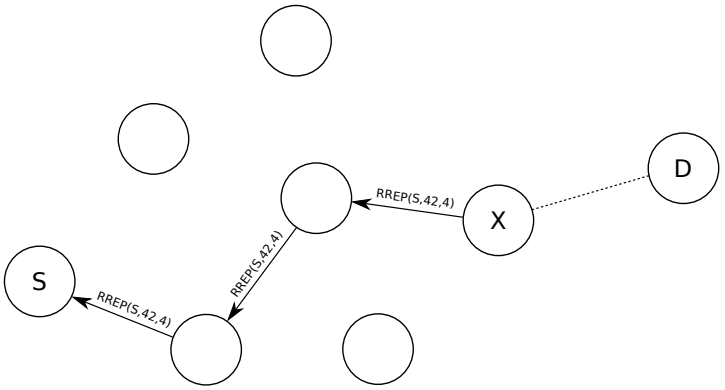


Figure 2.9: Propagation of a route reply (RREP) message through a network.

A RREP message contains the source and destination address, the destination sequence number, the total number of hops from source to destination,

and a lifetime value for the route.

In the example presented here only a single RREP message is sent back to the source. It is possible though, that multiple replies are sent. In this case an intermediate node forwards the first RREP it receives, and any following RREP messages that it receives are only forwarded if: 1) they have a higher destination sequence number, or 2) they have the same sequence number but a lower hop count.

When the RREP is send back to the source the intermediate nodes record which node they receive it from and in this way they build a *forward path* from the source to the destination. This path is then used to route the data along.

Evaluation

AODV is designed with all of the goals mentioned in Section 2.3 in mind. It tries to minimise the control traffic flowing through the network by having nodes only maintain active routes. The processing overhead is also very small since the only actions ever done by a node are simple table lookups and updates. And, finally, loops are prevented by using the same technique as in DSDV.

The amount of control traffic flowing through the network is minimised by avoiding the system-wide broadcasts of entire routing tables. In DSDV system-wide broadcasts are sent when a change occurs, to enable all nodes to compute routes to all other nodes. This amounts to a control message overhead that grows as $O\left(n^2\right)$ where n is the amount of nodes in the network. This overhead is avoided in AODV by using the discovery mechanism that was described, and by only maintaining *active* routes in the network. In this respect every forward route entry has an associated timeout, and if no data has flowed through the route within the timeout period the route is discarded.

Avoiding the system-wide broadcasts when maintaining the route table seems like a good idea, but it comes at the cost of having to do system-wide broadcasts, when a new route must be discovered. As could be seen in Figure 2.8, the RREQ message is heavily replicated in the network before a proper route is found. These RREQ messages are a lot smaller than the routing tables that are broadcasted in DSDV, but in some usage scenarios, the path finding mechanism may also be very costly.

It is possible to reduce the overhead involved when sending RREQ messages by using an *expanding ring* technique. Using the expanding ring technique a RREQ message is initially only forwarded a small number of hops by setting a low time-to-live (TTL) value on the message. If no route is found in this way, the TTL is increased and the RREQ is sent again. If the destination is relatively close to the source, this will save a large amount of control traffic. If, on the other hand, the destination node is far away, the

expanding ring technique is even more costly than the standard system-wide broadcast.

2.3.5 Dynamic Source Routing

This description of the dynamic source routing (DSR) protocol is based on the article by Johnson and Maltz in [11].

DSR is an on-demand routing protocol just like AODV. It uses a path discovery mechanism for locating routes through the network much like the one that is used in AODV. The main difference between AODV and DSR is the fact that DSR is a *source routing* protocol.

Source routing is a routing technique, where every message contains a header describing the entire path that the message must follow. When a node receives such a message, it checks whether it is the destination node, and if not, forwards the message to the next node in the given path.

Using this technique, there is no need for intermediate nodes to keep any state about an active route, as it was the case in the AODV protocol.

This protocol does not assume symmetrical links and can actually utilise uni-directional links. This means, as will be shown shortly, that the protocol may choose one route for sending messages from A to B and a different route for sending from B to A.

Path discovery

The discovery mechanism of DSR is much the same as for AODV. The source node sends a route request (RREQ) message to all of its neighbours and these neighbours forward the message until a route is found.

The RREQ initially contains only the source and destination address and a *request id*. The pair <*source address, request id*> uniquely defines a RREQ and this is used by intermediate nodes to avoid forwarding RREQ messages, that have already been handled. When an intermediate node receives a RREQ, it does the following: If it has no route to the destination it appends itself to the list of nodes in the RREQ, and then forwards the RREQ to its neighbours. If it does have a route to the destination, it appends this route to the list of nodes in the RREQ and sends a route reply (RREP) back to the source. The propagation of RREQ messages throughout the network is shown in Figure 2.10.

Remembering how the propagation of RREQ messages looked in AODV (Figure 2.8) it can be seen that the exact same number of messages are sent when using DSR. And furthermore these messages flow along the same paths.

When a node is ready to send a RREP message back to the source node it can do one of three things: 1) If it already has a route to the source node it can send the RREP along this path, 2) It can reverse the route in the RREP

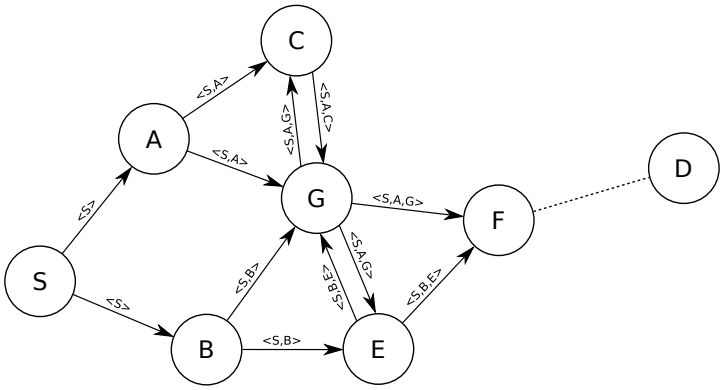


Figure 2.10: Propagation of route request (RREQ) messages throughout the network.

and try to send it along this path or 3) It may initiate a new RREQ to find a route to the source. Only the second of these approaches assume that links are symmetrical and if this operation fails the third approach can be used as a fallback. Looking closely at the third approach a loop can be identified; an endless sequence of RREQ messages will be sent between the source and destination hosts unless something is done. That something is simply that the RREP message is piggy-backed on the second RREQ message. This way the receiver of this RREQ message will be given a path to use when returning the reply.

The route cache

There is no actual route table in DSR. Instead, DSR uses a *route cache* of currently known routes. This route cache of a node is in effect a tree rooted at the node.

The route cache can contain multiple routes to a single destination. This means that when an error occurs that invalidates a link in the tree, there may still exist a valid route *around* this link for the destinations that were previously reached through the lost link.

Because of the tree structure of the route cache it can be represented using $O\left(n^2\right)$ storage in the worst case, where n is the number of nodes in the network. It is not necessary through to use $O\left(n^2\right)$ storage for the route cache, and in all practical implementations some upper limit on the size of the cache will be set. This upper limit must allow for the possibility, that a route may use all nodes in the network, which gives a storage complexity of $O\left(n\right)$.

Promiscuous mode operation

DSR is the first of the protocols presented here that take advantage of the fact that wireless devices can overhear messages that are not addressed to them. Consider for example Figure 2.11. In this figure A sends a message to C, but because of the broadcast nature of the medium, B is able to overhear the message.

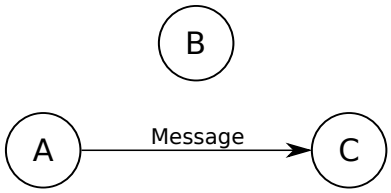


Figure 2.11: Promiscuous mode operation.

Having nodes overhear messages that are not addressed to them is called promiscuous mode operation.

Promiscuous mode operation is not required for DSR to work, but it does improve the protocol quite a bit. There are a number of optimisations to the protocol that use promiscuous mode operation and some of these are presented here.

One thing that promiscuous mode operation can be used for is to do *passive acknowledgements*. To discover link failures, e.g. when two nodes on a path moves out of transmission range, some sort of acknowledgement mechanism must be used. This is usually done by using link-layer acknowledgements but if such a functionality is not available, other means must be used. A passive acknowledgement is when a host, after sending a message to the next hop host in a path, overhears that the receiving host is transmitting the message again. This can be taken a sign, that the host has in fact received the message and is now in the process of forwarding it towards the next hop.

Another optimisation that promiscuous mode offers is the possibility for a host to overhear the messages sent to its neighbours. This can be used in a number of ways. For example, a host that overhears a message may add the route of the message to its route cache. The overheard message may also be an error message in which case the host can correct its route cache to reflect the topology changes.

Overhearing messages may also be used for route shortening. Take the following example: A route contains the nodes A, B and C in that order. When A forwards a message to B, C overhears this message. C thus finds out that the route can be shortened by skipping node B (i.e. sending the message directly from A to C) and it notifies the source of the message by sending it an unsolicited RREP message with this new path.

Evaluation

Like AODV the DSR protocol only maintains *active* routes, i.e. routes that have been used within some predefined timeout period. The number of control messages used to maintain these routes are kept low by using some of the same optimisations as in AODV; duplicate RREQ messages are discarded and intermediate nodes can respond to route requests, if they have a route to the destination. The route discovery mechanism can also easily be extended by using some variant of the expanding ring algorithm. Furthermore, using promiscuous mode operation, the route cache can be kept up to date to try and minimise the length that a RREQ message must travel.

The storage overhead of DSR is $O(n)$. The only thing, that a node has to store is the route cache and information about the recently received RREQ messages. It has already been argued that the route cache is kept within $O(n)$. The list of recently seen RREQ messages can be kept at a constant size $O(1)$ by simply discarding the information in FIFO order.

The processing overhead of DSR is also fairly small since the size of the route cache is only $O(n)$. This means that the maintenance of the route cache can be done cheaply.

Loops are avoided easily in source routing by having nodes check whether they themselves are a part of the current path before forwarding a RREQ message. If the node is already a part of the path the RREQ is discarded since forwarding it would create a routing loop.

2.3.6 Zone Routing Protocol

The zone routing protocol (ZRP) is a hybrid protocol; it uses both pro-active and re-active routing schemes. This protocol is described in detail in the article by Pearlman and Haas [14].

In ZRP each node defines a *zone* consisting of all of its n-hop neighbours where n may be varied. Within the zone the node pro-actively maintains a routing table of routes to all other nodes in the zone. This is done using the *intrazone* routing protocol, which is a LS based protocol.

When sending messages between nodes in the zone the zone routing table may thus be utilised. But when a node needs to send a message to a node outside of the zone a re-active *interzone* routing scheme is used. The re-active routing done in ZRP uses a concept called *bordercasting*. Bordercasting works in the following way: The source node sends a route request (corresponding to the RREQ messages used in the previous sections) to all of the nodes on the border of its zone. These nodes are the ones in the zone whose minimum distance to the source node is the zone radius. Each border node then checks to see if the destination resides within its zone; if it does it sends a route reply back to the source node and otherwise it forwards the request to its border nodes thus propagating the query throughout the network. An example of

this is shown in Figure 2.12.

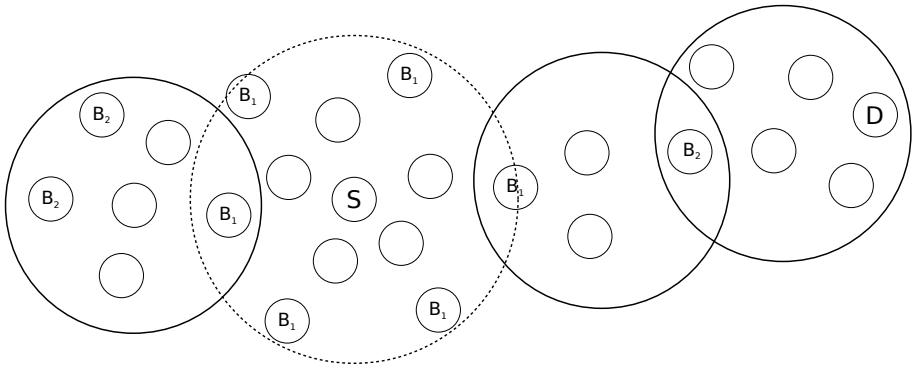


Figure 2.12: Route discovery in ZRP. This example shows how bordercasting is used to re-actively find a route from source node S to destination node D.

In this example the zone radius is set to two hops. This is probably unrealistic in a real world setting, but the low radius has been chosen to make the figure more simple. Node S initially sends a route request message to the nodes marked B_1. These nodes forward the message to the nodes marked B_2 and in this level the node D is reached.

Evaluation

Using bordercasting saves a lot of control traffic when doing route discovery compared to the other re-active protocols. This is because the RREQ messages are sent only to the border nodes of every zone that it reaches. Using this bordercast strategy comes at the cost of having to use more control messages within the limited range of the zones.

ZRP has a storage complexity of $O(n^2)$, where n is the number of neighbours within the zone, and because the intrazone routing protocol is LS based it has the $O(m + n \log n)$ running time where m is the number of edges connecting the n nodes in the zone. In dense MANET scenarios this will lead to a very large route computation overhead and ZRP will thus not be feasible in such networks.

Chapter 3

Energy efficient MANET routing

The previous chapter was concerned with the general concept of routing traffic through a mobile ad hoc network (MANET), and a number of routing techniques and protocols were presented. The protocols presented in the previous chapter were chosen because they are representative of the field of routing in MANETs, and because they collectively represent the three common approaches towards routing in MANETs: *pro-active routing*, *re-active routing* and *hybrid approaches*.

None of the protocols presented in Chapter 2 were specifically designed to be *energy efficient* though. They all try to minimise the amount of control traffic that flows throughout the network, which of course saves energy since transmitting messages consume energy at both the sending and the receiving hosts, but this is done primarily to avoid wasting bandwidth and the protocols as such are not optimised towards energy efficiency.

Energy efficiency, on the other hand, is the focus of this chapter. Here the general concept of energy efficient MANET routing is presented and some protocol proposals are discussed.

3.1 Introduction to energy efficient routing

There are two main approaches towards energy efficiency in MANET routing and they are the *power-save approach* and the *power-control approach*.

The power-save approach is concerned with *sleep states*. In a power-save protocol the mobile nodes utilise that their network interfaces can enter into a sleep state where less energy is consumed. The power-save approach is described in Section 3.3.

In the power-control approach no sleep states are utilised. Instead the power used when *transmitting* data is varied; which also varies the transmission range of the nodes. There is some energy to be saved by using the

power-control approach but the real source of energy waste in most mobile ad hoc networks (MANETs) is idle time energy usage, i.e. the energy spent when a node is idle but not sleeping. Because of this observation only the power-save approach will be considered in this thesis. The power-control approach is still interesting though, and could be used as a complement to the power-save approach, and it is therefore described shortly in Section 3.2.

When talking about *energy efficiency* one must also define what the goal of this energy efficiency is, i.e. which lifetime is it that the protocol tries to maximise? One approach is to maximise the lifetime of the entire network. In such a scheme stronger nodes, i.e. nodes that have a longer battery lifetime, may be asked to do a lot of the heavy lifting. This means that these nodes deplete their energy resources faster than they would normally, but this is done to increase the overall lifetime of the network. Another approach is to use minimum energy routing where the protocol tries to always use the route that uses the minimum amount of energy. This approach does not favour any specific nodes and the lifetime of the network as a whole is not maximised. In this thesis the weight is on protocols that try to optimise the overall network lifetime.

Some energy efficient protocols, power-save and power-control alike, use the physical position of the nodes to make their routing decisions. These protocols generally assume that there is some sort of positioning mechanism available, such as for example a global positioning system (GPS) unit. In this thesis it will not be assumed that nodes have positioning devices. Work is being done on some positioning schemes that use only information from the IEEE 802.11 (wi-fi) interface. One example is the positioning scheme developed by Place Lab (http://placelab.org). This would make it possible to use the position aware routing algorithms on devices with no GPS unit. There seems to be some problems with these positioning schemes though. For example the approach taken by Place Lab assumes that there are statically positioned devices, such as wireless access points, in the network at all times. This is not the case in the general MANET setting and therefore the Place Lab approach is useless in this setting. Because of these uncertainties routing that uses position information are not considered in this thesis.

There is in fact a third class of energy efficient routing protocols, that do not fall into either the power-control or the power-save class. These protocols neither use sleep states nor transmission power control, and as such it could be said that they do not exactly *save* any power at all. What these protocols try to do is in fact *load balancing* instead of power saving; they try to make sure that the load of routing traffic is balanced in a way such as to maximise the overall lifetime of the network. An example of such an algorithm is DEAR [8], that is described by Gil et al. These pure load balancing algorithms are not described any further in this thesis.

3.2 The power-control approach

Power-control protocols cut down on energy consumption by controlling the transmission power of the wireless interfaces. Turning down the transmission power when communicating with nearby neighbours has a number of beneficial effects: firstly, it consumes less energy at the sending host to transmit the message; and secondly, since the transmission range of the message is lowered a smaller number of nodes have to *overhear* the message which also saves energy. It thus seems obvious that controlling the transmission power for this kind of short range communication is beneficial for the overall energy consumption of the network.

Less obvious though is that, even disregarding the energy preserving effects of having less interference, it may still be beneficial to use a shorter transmission range when doing multi-hop routing. This may lead to longer routes in terms of hops but, because of the non-linear relation between the transmission range and the energy used by the wireless network interface, it may still lead to a lower overall energy consumption. An example of this can be seen in Figure 3.1. In this figure the cost of transmitting data for one second over a wireless link is described by the cost function c. Here it can be seen that using the intermediate node B when sending from A to C will save $0.3J/s$.

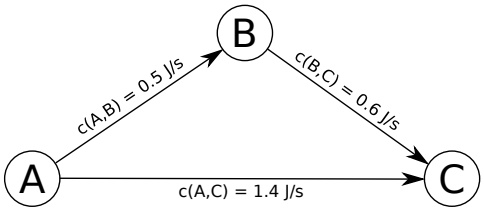


Figure 3.1: A scenario where it would be beneficial to use intermediate nodes instead of using the direct link.

Considering this observation some protocol proposals, such as for example PARO by Gomez et al. [9], work after the principle that additional forwarders (intermediate nodes) should be added in a route between a source destination pair, if this lowers the cost of the path. This approach lowers the overall energy cost of the route as well as the amount of interference in the network.

The power control approach may be interesting to investigate when developing energy efficient routing protocols, but, as mentioned earlier, it will not be discussed further in this thesis.

3.3 The power-save approach

Protocols that use the power-save approach cut down on energy consumption by utilising the sleep states of the network interfaces. While a node is sleeping it can not participate in the network, which means, that protocols utilising sleep states need to either 1) use retransmissions of messages to make sure that a message is received, or 2) make sure that all of the nodes do not sleep at the same time, and thus delegate the work of routing data to the nodes that are awake.

While in this sleep state the node can not send or receive data and any messages that would normally be sent *to* or *through* this node will consequently be lost. This is of course not acceptable since lost messages leads to retransmissions which leads to a higher energy consumption. Therefore the power-save protocols define ways in which the nodes can *take turns* sleeping and being awake so that none, or at least a very small percentage, of the messages sent in the network are lost due to nodes being in the sleep state.

Power-save protocols are specifications of *how* it is possible to maximise the amount of time that nodes are sleeping while still retaining the same connectivity and loss rates comparable to a network where no nodes are sleeping.

One approach towards utilising sleep states efficiently is the IEEE 802.11 ad hoc power saving mode that is a part of the IEEE standard. This low-level power saving mode does power saving on the link layer and is as such independent of which routing protocol is used on the network layer. The IEEE 802.11 ad hoc power saving mode is described in Section 3.3.1.

An example of a power-save approach that works on a higher level is BECA/AFECA that is described by Xu et al. [20]. This approach, which is based on retransmissions, is described in Section 3.3.2.

Another power-save approach that operates on a higher level is Span described by Chen et al. [3]. Span works by keeping some nodes awake at all times, and the routing responsibility is thus delegated to this routing backbone. This approach is described in Section 3.3.3.

3.3.1 IEEE 802.11 ad hoc power saving mode

The following description is based in part on the article by Feeney [7] and in part on the article by Röhl et al. [17].

The IEEE 802.11 ad hoc power saving mode (AHPSM) works by defining a synchronised *beacon interval* within which each node can take a number of actions. The beacon interval is maintained in a distributed fashion by all the nodes in the network by the transmission of beacons. In the end of each beacon interval the nodes compete for transmission of the next beacon, using a random back-off algorithm to avoid collisions. The node that first transmits the beacon wins, and all others have to adjust their timers to this

beacon.

Once the beacon interval has been established the power saving is done in the following way. In the beginning of each beacon interval all nodes must be awake. In this part of the interval all senders announce to their receivers of unicast data that they have some data for them by sending them an *ad hoc traffic indication map* (*ATIM*). These ATIMs are acknowledged by the receiver, and after this both nodes must remain awake to be able to send and receive the data respectively. This first part of the beacon interval is called the *ATIM window* and here only ATIMs are allowed, i.e. no data may be sent. Broadcast traffic is also announced within the ATIM window but these ATIMs do not need to be acknowledged by the receivers.

Nodes that do not have any data to send, and did not receive any ATIMs within the ATIM window, can then safely sleep until just before the end of the beacon interval. The senders and receivers on the other hand stay awake and use the remainder of the beacon interval to transmit data.

It is evident, that the performance of the IEEE 802.11 power save mode is very dependent on the relative sizes of the beacon interval and the ATIM window. If the ATIM window is too short there is not enough time to announce enough traffic to utilise the entire beacon interval. If, on the other hand, the ATIM window is too long not only will the nodes have to stay awake for a longer time, it will also mean that more traffic can be announced than it is possible to send within the beacon interval. Furthermore, if the beacon interval is too short the cost of beaconing and the overhead of entering and leaving the sleep mode becomes too high. To complicate matters even more the "correct" size of beacon interval and ATIM window is relative to the traffic load in the network, so an adaptive beacon interval and ATIM window size may be needed to make the AHPSM effective.

Apart from the problem of selecting good sizes for the beacon interval and ATIM window the AHPSM also suffers from other problems. The main problem is that the delivery latency can be very high when multiple hops are used. This is because, for each hop that the message traverses, an entire beacon interval must pass before the message can be passed on to the next hop in the route.

As mentioned the AHPSM works on the link layer and is therefore independent of the overlying network layer routing protocol. This is of course not the whole truth. The overlying protocol must be able to efficiently make use of the AHPSM and to do that the two layers must be tightly coupled. If, for example, the overlying protocol uses periodically broadcasted `HELLO` messages it becomes important to synchronise the transmission of these messages so that all nodes exchange `HELLO` messages at the same time. If this is not done it is easy to imagine a scenario, where each node sends its periodic `HELLO` message in a different beacon interval. And with the `HELLO` messages being broadcasted this means that no nodes will be able to enter into the sleep state, since there will almost always be some broadcast traffic in the

network.

One such higher level routing approach that works in close cooperation with the IEEE 802.11 ad hoc power saving mode is Span that is described in the Section 3.3.3. Span even remedies some of its shortcomings such as the high delivery latency.

3.3.2 BECA/AFECA

In the article by Xu et al. [20] two power-save approaches are described; the simple Basic Energy-Conserving Algorithm (BECA) and the extended version of BECA called Adaptive Fidelity Energy-Conserving Algorithm (AFECA). The difference between BECA and AFECA is that AFECA takes node density into consideration when determining the period of time that a node may sleep.

Both approaches are only power saving algorithms and *not* routing protocols. This means that they need to work together with some existing MANET routing protocol. It makes sense to choose an on-demand routing protocol for this purpose since the control messages sent in a pro-active protocol would keep the nodes awake even in low traffic scenarios. For the simulations done in the article BECA and AFECA is build on top of AODV.

In the following both algorithms are described naturally starting with BECA before going on to AFECA.

Basic energy-conserving algorithm (BECA)

As mentioned, BECA is based on retransmissions to avoid loosing data. This means that the algorithm consists of some timing information that defines the periods that nodes spend in the different states defined by the algorithm, and a specification of how many retransmissions are needed to ensure that at least one packet reaches its destination. BECA operates with three different states; *sleeping*, where the node has its network interface in the sleep state, *listening*, where the network interface is active and the node is listening for incoming traffic, and *active*, where the node is actively participating in the routing of messages. The states and the transitions between them can be seen in Figure 3.2.

In the state diagram in Figure 3.2 some time intervals are used. These are described in detail here along with some other important BECA parameters:

T_l The time that a node spends in the *listening* state.

T_s The time that a node spends sleeping.

T_a The period of time that a node remains active *when no messages are being processed*, i.e. when the node is in fact not active anymore.

T_o The interval that messages are retransmitted in.

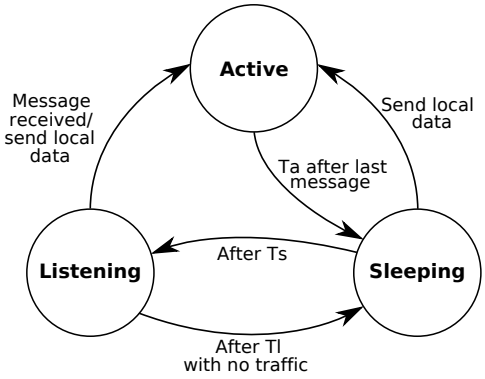


Figure 3.2: BECA state diagram.

R The number of retransmissions that are needed to maintain stability.

To make sure that no messages are lost due to nodes being asleep, some relations between the individual time intervals must be defined. These are defined in Equation (3.1).

$$\begin{aligned} T_l &= T_o \\ T_s &= kT_o, \text{ for some constant } k \\ R &\geq k+1 \\ T_a &\geq T_o \end{aligned} \tag{3.1}$$

Using these rules for the time intervals T_l, T_s, T_a, and T_o together with the R retransmissions ensures that no messages are lost in the network due to sleeping nodes. As can be seen, the sleeping and listening intervals are both defined from T_o, which is the key to why a message is eventually received by the destination. If a message is sent from a host A to a host B while B is sleeping the message will be retransmitted $R \geq k+1$ times with interval T_o until the message has been received. Since the sleep interval T_s is defined as kT_o at least one of these transmissions will be received, even in the case where B enters the sleep state just before A starts transmitting the message. This is illustrated in Figure 3.3.

So what is gained energy-wise by using BECA? And at what cost? Of course the sleeping and retransmissions incurs a higher latency, in the worst case of kT_o and on average $\frac{kT_o}{2}$. This latency is added for each hop in a route and for multi hop routing this adds a delay on average of $\frac{n \times kT_o}{2}$, where n is the length of the route. To keep this latency low one needs to select a small value for k.

This counteracts the wish for low energy usage. When considering energy efficiency a higher value of k is attractive because this means that the sleep

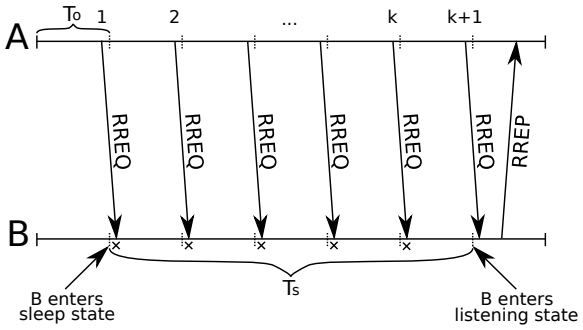


Figure 3.3: Example of retransmissions in BECA.

to listening ratio is high, and consequently the energy savings are big. The maximum possible power saving, which is only reached in a network with no traffic, is $\frac{k}{k+1}$.

Choosing the right value for k is a weighting of what is most important in the current usage scenario. In the simulations done by Xu et al. they find that a value of $k = 1$, which theoretically leads to a 50% saving on energy, gives a good balance between energy savings and transmission latency. Larger values of k does not considerably improve energy efficiency and it is therefore not worth it, when the larger delays are taken into consideration.

A nice feature of BECA, which also applies to AFECA, is that in high traffic scenarios, where all nodes are on at all times, nodes are simply kept in the active state. In this way the power saving mechanism is disabled and the performance of the protocol is thus as good as the underlying protocol.

Adaptive fidelity energy-conserving algorithm (AFECA)

AFECA uses the same power-save model as BECA with a single modification. Instead of having nodes sleep for T_s seconds when they enter the sleeping state a new, variable sleep interval T_{sa} is introduced.

T_{sa} is varied according to the number of neighbours surrounding a node. Each node tries to estimate its current number of neighbours by recording the id of each node that it overhears while in the listening state. Nodes in this neighbour list are removed if they have not been heard from with a certain time interval T_e. The only information that is used from this approximative neighbour list is its length N. Using this information the sleep interval is selected by Equation (3.2).

$$T_{sa} = \text{Random}\,(1, N) \times T_s \tag{3.2}$$

Equation (3.2) results in a sleep time of $\frac{N \times T_s}{2}$ on average for nodes, which clearly favours nodes in a dense population in a way such that these nodes may save a lot of power. It must be noted that the neighbour count N is

only an approximation and in most cases the node will have more than N neighbours since it only counts the ones that it have overheard.

There is a problem with this approach, and that is the fact that the longer sleep periods potentially leads to a greater packet loss. In BECA the sleep period T_s is adjusted so that at least one or the R retransmissions will reach the sleeping node. But when the sleep period becomes longer, and there is no accompanying modification of R, this guarantee can no longer be given. One possible approach to alleviate this problem would be to make R variable as well. A conservative approach would the be to chose $R = (N \times k) + 1$ whenever a message was sent.

According to Xu et al. AFECA is able to double the overall lifetime of the network as the network density rises.

3.3.3 Span

Span, as it is described by Chen et al. [3], is a power-save approach based on the notion of connected dominating sets (CDSs). A CDS is a connected subgraph S of a graph G such that every vertex u in G is either in S or adjacent to some vertex v in S. In layman terms the CDS is a set of nodes from which all other nodes in the network can be reached.

A CDS is ideal for routing purposes since the definition of a CDS means that all nodes of the network can be reached from it. It is therefore possible to use the nodes in the CDS as the only routers in the network. In Span the nodes that are a part of the CDS are called *coordinators*. These coordinators define a routing *backbone* that is used for routing all traffic in the network. Non-coordinator nodes are thus not used for routing purposes and they may therefore spend some of their time sleeping.

Using the coordinators for routing of course saves energy at the non-coordinator nodes that may sleep while the coordinators does all the heavy lifting. But, if the coordinator role is not distributed amongst the participating nodes in the network the coordinators will quickly deplete their energy resources, which will lead to a very small overall network lifetime. To counter this Span defines a coordinator selection scheme that tries to distribute the coordinator responsibility evenly amongst the nodes. The coordinator selection scheme takes such factors as the remaining battery capacity of the node and the *utility* of the node into consideration. The utility of a node is a measurement of how much more connected the network would be if that node was chosen as a coordinator, and it is measured in how many more pairs of neighbour nodes that would be connected if the node was chosen as a coordinator. The coordinator selection scheme is described in detail in the next section.

Coordinator selection

The coordinator selection mechanism is perhaps the most important part of Span. This mechanism is what ensures that the burden of being a coordinator, and therefore a router of messages, is divided amongst the nodes in a way such as to maximise the network lifetime. The coordinator selection algorithm is invoked periodically at every non-coordinator node, and similarly the coordinator nodes periodically run a coordinator withdrawal algorithm.

When the coordinator selection algorithm is run the node needs information about its one and two-hop neighbours, and for each neighbour also whether that neighbour is a coordinator. This information is maintained pro-actively by using a standard `HELLO` message approach, as the one described in Section 2.3.1, where each `HELLO` message contains information about neighbours and coordinators of the sending node.

As mentioned both the utility of the node and the remaining energy is taken into consideration when finding new coordinators. The way that it is implemented is by using a randomised back-off delay that the node uses before announcing itself as a new coordinator. The utility and remaining energy are factors in this delay. The entire delay computation is shown in Equation (3.3).

$$delay = \left(\left(1 - \frac{E_r}{E_m}\right) + \left(1 - \frac{C_i}{\binom{N_i}{2}}\right) + R\right) \times N_i \times T \tag{3.3}$$

The back-off delay calculation in Equation (3.3) randomly chooses a delay over an interval proportional to $N_i \times T$, where N_i is the number of neighbours for node i and T is the round trip delay for a small packet. The random part of the expression is build of three important pieces. The energy aspect is reflected in the subexpression $\left(1 - \frac{E_r}{E_m}\right)$, where E_r is the remaining capacity and E_m is the maximum amount of energy that the node can have. This subexpression makes sure that there is a linear relation between energy capacity and willingness to become a coordinator.

The utility of the node, i.e. the current value of having the node as a coordinator, is reflected in the subexpression $\left(1 - \frac{C_i}{\binom{N_i}{2}}\right)$, where N_i is the number of neighbours for node i, and C_i is the number of additional pairs of nodes that will be connected if i is selected as a coordinator. This subexpression makes sure that nodes that offer a good connectivity of the routing backbone are preferred, which in turn means that fewer coordinator nodes are required.

Finally, the third piece of the subexpression is the random part R. Here R is a randomly selected number from the interval $(0, 1[$. This random part is important to make sure that the coordinator announcements are distributed evenly so that all nodes will not announce their willingness at the same time,

which would result in a large number of packet collisions.

To make sure that nodes do not remain coordinators indefinitely a coordinator withdrawal algorithm is run periodically by the coordinator nodes. This withdrawal algorithm checks a number of things. Firstly, a coordinator should withdraw if all of its neighbours can connect to each other either directly or through one or two other coordinators. Secondly, when a node has been a coordinator for a certain period of time it checks if every pair of neighbour nodes can reach each other via one or two other neighbours, even if those neighbours are not coordinators at the time. If this is the case the node marks itself as a tentative coordinator. A tentative coordinator still does coordinator work but the coordinator selection algorithm does not regard tentative coordinators as coordinators. Therefore new coordinators will be selected in an area that has tentative coordinators and these can then withdraw as coordinators.

According to the simulations done by Chen et al., this coordinator selection mechanism does a very good job of distributing the job being coordinator amongst the nodes.

Cooperation with other protocols

Span in itself is not a routing protocol; it only defines a way to fairly select the coordinators for the routing backbone in a distributed fashion using only local knowledge. To make Span work it must cooperate with a routing protocol such as for example AODV, DSR or DSDV. In the article by Chen et al. a *geographic forwarding* protocol is used for the simulations. Such a protocol uses the physical location of the nodes in its routing considerations, and it therefore depends on some sort of location device such as a GPS unit.

Geographic routing is done using a greedy strategy where a node always forwards a given message to the one of its neighbours that will bring it closest to its ultimate destination. Nodes pro-actively maintain a neighbour table by sending out periodic `HELLO` messages. These `HELLO` messages can be combined with Spans advertisements of neighbours and coordinators that are also sent out periodically.

When a route to a node, that is not within two hops of the sender, is needed this route is re-actively searched for using a broadcasted route request packet. As it was the case in the other re-active protocols, AODV and DSR that were described in Section 2.3.4 and 2.3.5 respectively, a route reply is returned in response to the route request. This route reply contains the physical location of the destination node, which is all that is needed to start the greedy routing algorithm.

When a protocol, such as the geographical routing approach described here, is combined with Span there are some special considerations that must be taken. First off, nodes in a Span network need to periodically send and receive `HELLO` messages to build the table of two-hop neighbours that is needed

to run the coordinator selection algorithm. This fits nicely together with some of the routing protocols that we have studied thus far, like for example OLSR, AODV and geographic forwarding, since these use a neighbourhood discovery mechanism that involves periodic `HELLO` messages.

Secondly, for Span to work it is important that *only* coordinator nodes are included in the route calculations of the routing protocol. Remember that the coordinator nodes form the routing backbone that must forward all traffic in the network, and therefore all routes computed by the routing protocol must use only these nodes. This could pose quite a challenge for many routing protocols, since the coordinator role is switched between nodes regularly. This means that the protocol must be tuned towards a *very* dynamic neighbourhood where neighbours regularly disappear just to reappear a little later. This rapid switching of neighbours is no problem in the geographic routing approach used in the article since no routes are ever recorded in this routing scheme. It may, on the other hand, be a problem in other protocols. For example, in ad hoc on demand distance vector (AODV) routes are stored by having intermediate nodes record the next-hop node in route. When the next-hop repeatedly disappears this becomes a problem and an large amount of route repair operations will be needed. The same problem would occur in a source routing protocol such as DSR, since the nodes used in the route calculation would repeatedly change with route breakage as result.

How protocols such as AODV and dynamic source routing (DSR) would perform when working in co-operation with Span is hard to predict. A simulation study will be needed here. It would be possible though, to make some changes to protocols such as AODV and DSR that would make the co-operation with Span work better. One such change could be to let the protocols store multiple routes to a destination, even when the intermediate nodes used in the inactive routes do not exist at the current time. Then, when a node disappears, some of the alternative routes could be tried before a new route request was initiated.

Span and the IEEE 802.11 AHPSM

Span, as it is described in the article by Chen et al., depends on the IEEE 802.11 ad hoc power saving mode for the actual power saving operations such as putting the non-coordinator nodes to sleep and buffering packets for sleeping nodes.

Span works together with the AHPSM in a number of ways. Only non-coordinators have the possibility of entering into the sleep state, if they have no pending traffic. Since coordinator nodes do not operate in power saving mode packets routed between coordinators do not need to be announced within the ATIM window. This means that packets that traverse the routing backbone do not suffer from the delays that can occur in multi hop routing when AHPSM is used.

Span also defines some changes to the AHPSM to optimise it for usage in a network using Span. Firstly, because both Span and the overlying geographic forwarding algorithm, that Chen et al. base the simulations on, uses broadcast traffic for local connectivity management, an optimisation of the handling of broadcast traffic has been implemented. In the original AHPSM only a single broadcast notification was needed since nodes would stay awake for the entire beacon interval anyway. In the simulations every broadcast message must be advertised so that a node may go back to sleep when it has received all its pending messages. Another optimisation is the introduction of a new *advertised traffic window* (ATM). The ATM is placed directly after the ATIM window and within this window of time all advertised traffic must be transmitted. The rest of the beacon period is then used for messages sent between coordinators, i.e. routing traffic.

3.4 Span on BECA/AFECA

As mentioned in Section 3.3.3 Span in itself is just a distributed algorithm for calculating a CDS of coordinator nodes. To save energy Span needs to work together with another power saving algorithm that actually puts the nodes to sleep. In the article by Chen et al. [3] Span works together with the AHPSM to achieve power saving. But the article mentions that this lower layer can easily be exchanged with other power saving approaches.

In this section it will thus be discussed how Span can be placed on top of AODV and BECA/AFECA. In the following, when BECA is mentioned this applies to both BECA and AFECA.

It is in fact quite trivial to place Span on top of AODV. Span uses some periodic exchange of neighbourhood information between neighbouring nodes to build the CDS of coordinator nodes. This neighbourhood information can easily be piggy-backed on the `HELLO` messages that are already sent in the AODV protocol. The AODV `HELLO` messages are thus extended to include information about the coordinators and neighbours of the sending node.

This small extension is enough to build Span's coordinator backbone. The next thing to do is to make sure that only coordinators are used for routing purposes. This is done by simply letting coordinators be the only ones that are allowed to forward route requests (RREQs). In AODV route replys (RREPs) follow the reverse path of the received RREQ and following this reverse path creates the forward path towards the destination. This means that if only coordinators forward RREQ packets only coordinators will be used in the resulting route.

The two extension described thus far are enough to make Span function on top of AODV. There are some further optimisations that can be made though. For one it is possible to avoid some of the retransmissions that are normally used in BECA. All packets are normally retransmitted R times,

as described in Section 3.3.2, but since the coordinator nodes are always awake there is no need to retransmit the packets that flow between these nodes. This saves a lot of retransmissions since coordinator nodes are the only routers of traffic.

Another "optimisation" is that the ratio between T_l and T_s can be larger when Span is used. This is related to the low usage of retransmissions that were just mentioned; if the only places where retransmissions are done is in the beginning and end of a route, it does not mean that much if more retransmissions are used. Furthermore, a larger ratio usually means much larger latencies in packet delivery. This is not the case when Span is used since the coordinator backbone is always on. A larger ratio between T_l and T_s will in the end mean lower energy consumption.