MONASH UNIVERSITY
DEPARTMENT OF ELECTRICAL & COMPUTER SYSTEMS ENGINEERING

Performance of Telecommunication Networks

Experiment VIII: Introductory Network Layer Protocols: Ad-hoc Routing

Alexander Senior Y. Ahmet Şekercioğlu

Contents

1 Ad-hoc Routing

Ad-hoc routing is a form of routing used when all the normal assumptions in routing are thrown out the window:

- when nodes can move around,
- when *routers* can move around,
- when nodes can pop in and out of the network randomly;

or more generally, when the topology of the network is not fixed. Such networks are called ad-hoc networks, and special routing algorithms need to be used with them. One such algorithm is Ad-hoc On-Demand Distance Vector (AODV) routing; as its name suggests, it is related to the distance vector routing algorithm that you have learned about. However, AODV was specially developed to work in mobile networks, and is notable in that nodes discover routes to other nodes only when they need to send a packet. In this lab you will learn about AODV, and implement it in OMNeT++.

2 AODV

2.1 Route Creation

Note that the following is adapted from Chapter 5.2.10 of *Computer Networks* [1], which is available in electronic form from the library (see the Unit Website for the link).

In AODV, all nodes maintain a routing table, with the fields as shown in Figure 1.

Destination	Next Hop	Distance	Sequence #	Active Neighbours
⋮	⋮	⋮	⋮	⋮

Figure 1: Structure of the routing tables in AODV.

Destination is the address of a node within the network; **next hop** is the address of the node by which the destination node can (eventually) be reached; **distance** is the number of hops to the destination; **sequence number** holds the most recent sequence number that the node has seen for the destination (more information below); and finally a list of **active neighbours**, i.e. neighbours that use this node to reach the destination (this is used in the maintenance phase of the algorithm, described in Section 2.2).

When a node wishes to send data to another node, it first consults its routing table to see if it has an entry for it; if it does, then it will forward the message appropriately. If not, it will construct a special packet, called a ROUTE REQUEST. This packet has the format shown in Figure 2.

Source Address	Request ID	Destination Address	Source Sequence #	Destination Sequence #	TTL

Figure 2: Format of a ROUTE REQUEST packet.

The fields are as follows:

Source Address is the address of the node sending the packet.

Request ID is formed from a local counter that each node maintains, and is incremented with every ROUTE REQUEST. Specifying both a source address and a request ID will uniquely identify one ROUTE REQUEST in the network.

Destination address is the address of the node for which a route is being sought.

Source Sequence # is maintained from a local counter at the source node (i.e. the sender of the ROUTE REQUEST), and is incremented everytime this node sends another ROUTE REQUEST or replies to another node's ROUTE REPLY (detailed later).

Destination Sequence # is maintained in the routing table, and is the most recent value of the destination's sequence number that this node has seen (or 0 if it hasn't seen any).

This ROUTE REQUEST is broadcast to all local nodes. Upon receiving the ROUTE REQUEST, they will first check that the received packet is not a duplicate. Each node maintains a table containing source addresses, and the last request ID seen for those addresses; if the request ID in the table is the same or larger than the ID in the packet, then the packet is a duplicate and is discarded.

Otherwise, if the packet is not a duplicate *and* the receiving node is the requested destination, it will create a ROUTE REPLY packet as shown in Figure 3. The source address and destination address are copied directly from the ROUTE REQUEST, but the destination sequence number is copied in from the local counter (which is then incremented). The hop count is set to 0, and the TTL is set to an appropriate value. This reply is then unicast back to the sender.

Source Address	Destination Address	Destination Sequence #	Hop Count	TTL

Figure 3: Format of a ROUTE REPLY packet.

If the packet is not a duplicate and the receiving node is not the destination, then it will consult its routing table; if it *a*) has an entry for the destination, and *b*) the destination sequence number listed in the entry is *greater than or equal to* the destination sequence number listed in the packet[1], it will create a ROUTE REPLY packet as detailed previously, which it will unicast back to the sender. Otherwise, it will forward the ROUTE REQUEST packet to its neighbours. When it does so, it creates an entry is its *reverse route table*, which has the format shown in Figure 4. If/when the ROUTE REPLY to the relevant ROUTE REQUEST is received, the reverse route table will be used to unicast the reply back to the requester. Each entry in the reverse route table has a timer associated with it; when the timer expires, the entry is deleted.

Source Address	Request ID	Source Sequence #	Next Hop
⋮	⋮	⋮	⋮

Figure 4: Structure of the reverse route tables in AODV.

When a ROUTE REQUEST is created, it will be propagated throughout the network (with all the nodes forwarding the request creating entries in their reverse route tables) until the request encounters a node that either knows a fresh route to the destination, or is the destination itself. Then, a ROUTE REPLY will be created, and will follow a path

[1] This is to ensure that old (and possibly invalid) routes are not spread throughout the network.

back to the source of the request via the reverse routing tables[2]. When each node along the path receives the reply, it will update its routing table with the information in the reply if:

- it does not know a route to the destination,
- the destination sequence number in the packet is greater than that in the node's routing table, or
- the sequence numbers are equal but the hop count in the packet is less than the distance field in the routing table.

They will then forward the ROUTE REPLY according to the information in their reverse routing table, which will eventually reach the source node. Hence all nodes along the reverse path can be guaranteed to know an update-to-date route to the destination, all thanks to the original requester of the route! Note that nodes that are *not* on the reverse route will not receive the ROUTE REPLY, and will eventually delete the relevant entries in their reverse route tables when the timers expire.

2.1.1 Exercises

1. Implement ad-hoc routing using the MiXiM extension to OMNeT++. You will find the flooding network layers in the previous lab to be very helpful as a starting point, but you will need to add a lot of functionality.

   As a reminder, nodes will be required to maintain (at least):

   - their routing table, the format of which is shown in Figure 1 (you do not have to include the active neighbours field yet)
   - their reverse routing table, the format of which is shown in Figure 4; note that the node will also need to implement timers (i.e. self-messages) on the entries in these tables
   - their 'history table', which contains the most recent request ID seen for a given node
   - a local counter which they will use for recording their (source) sequence number
   - a local counter which they will use for recording their request IDs

   The packet types required (for routing purposes at least) are:

   - ROUTE REQUEST (format shown in Figure 2)
   - ROUTE REPLY (format shown in Figure 3)

[2]This path will start at the destination, proceed to the *first* node that forwarded the ROUTE REQUEST to the destination, and then continue on via the information contained in the reverse route tables back to the source.

2. As an extension to the algorithm, you may wish to consider and implement the following: the initial broadcast of the ROUTE REQUEST can lead to many packets been flooded in the network. To limit this, the TTL of the request can initially be set to 1; if no reply is heard within a set amount of time, then the request can be re-sent (with an incremented request ID!) with a TTL of 2, etc. This will effectively conduct the search for a route to the destination locally, and then at ever-growing 'rings' around the source node. How does this impact the performance of the network, especially the number of packets sent and the average latency of packets?

2.2 Route Maintenance

It was stated in the introduction that AODV can deal with a changing network topology; it does this by *a)* having nodes keep track of their neighbours, and *b)* having nodes note which neighbours forward messages through them. Each node in the network periodically broadcasts a HELLO packet to its neighbours, who send a reply straight back. In this way, nodes can learn and keep track of their neighbours; if a node does not get an expected HELLO message from a neighbour, or it does not get a response to its HELLO from a neighbour, then the node knows that neighbour has left the network.

As stated previously, nodes keep a list of active neighbours for each destination in their routing tables (see Figure 1). When a node forwards a data packet, it notes down the neighbour that sent that packet in the 'active neighbour' field for that destination. If, in the course of neighbour updates, the node discovers one of its neighbours has gone down, it will check its routing table for any entries that have the downed node as the next hop for any destination and note the active neighbours. The node will send a ROUTE UNAVAILABLE packet to the active neighbours, notifying them that their route to the destination(s) is now invalid and should be purged (it will then delete those entry/entries). These nodes will likewise forward the ROUTE UNAVAILABLE message to their active neighbours for that destination, and purge their routing tables. Hence the ROUTE UNAVAILABLE packets will propagate throughout the network until the bad route is deleted from every node.

2.2.1 Exercises

Implement the route maintenance part of the AODV algorithm. Obviously, it will not be used if the nodes themselves do not fail, hence attempt to simulate node failure in the network. One way to do this would be to have an module external to the network randomly decide on a node to deactivate (perhaps by emitting a signal); the network layer of the receiving node would then no longer respond to any messages[3].

The extra packet formats you will need are:

- HELLO (trivial format)

[3]Ideally, you would also create your own custom subclass of the `csma` MAC layer, so that the MAC layer also receives the same signal and does not respond either; otherwise you would have the slightly odd situation of a node whose radio works fine, but does not actively talk to anyone.

- HELLO REPLY (trivial format)
- ROUTE UNAVAILABLE (this will only require the destination address(es) of the nodes that can now not be reached via the existing route)

References

[1] Andrew S. Tanenbaum, *Computer Networks*, Prentice Hall, 4th edition, 2002.