AODV Project

Instructions:

To run the project execute the command ***make aodv.sim.*** To send a request message click the button on the source node.

Approach:

Our implementation follows the approach described in [1].

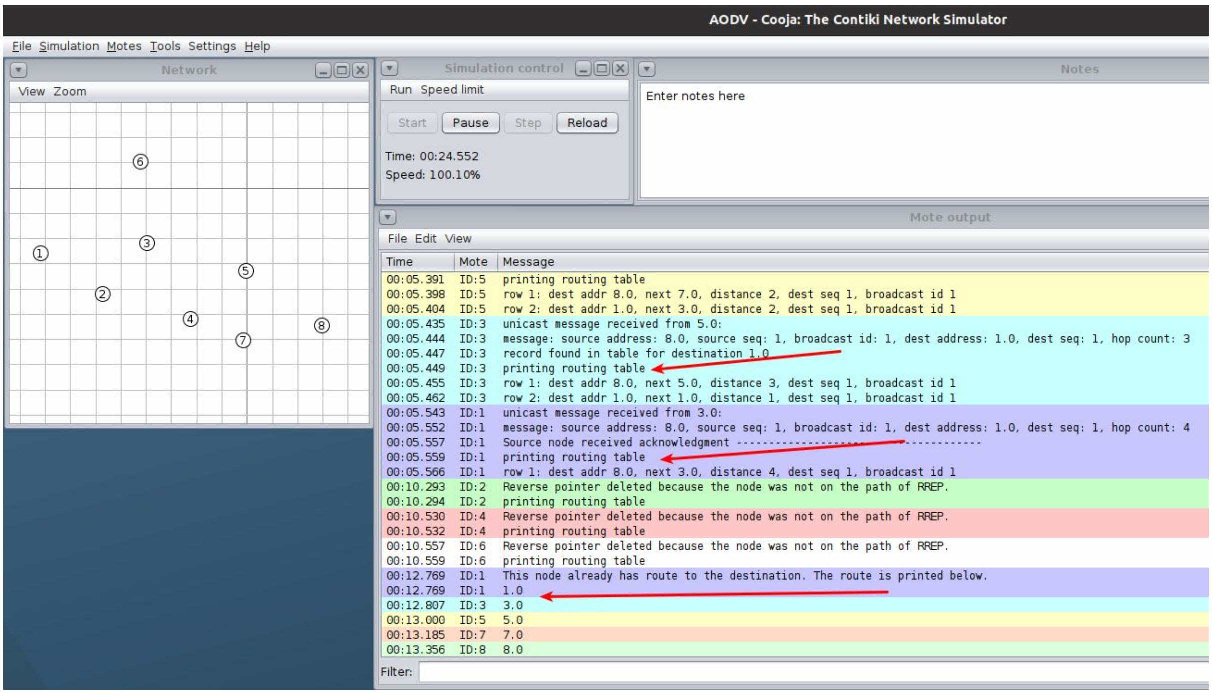
Note:

For all the cases below, the destination node is hardcoded to node8 from any source node.

Testing scenarios

Case01: Source node wants to communicate with destination

Source node will broadcast a message to its neighbour, if the neighbour node contains the route of the destination node in its routing table, it returns a route in the unicast message to the source node. If the route is not present in its table, it will broadcast to its neighbouring nodes until the destination nodes have reached. Once the destination node has reached, the destination node prepares the return request, and instead of broadcasting the return route, it sends the unicast response to its neighbour until it reaches the source node.



```
Time: 00:24.552
Speed: 100.10%
```

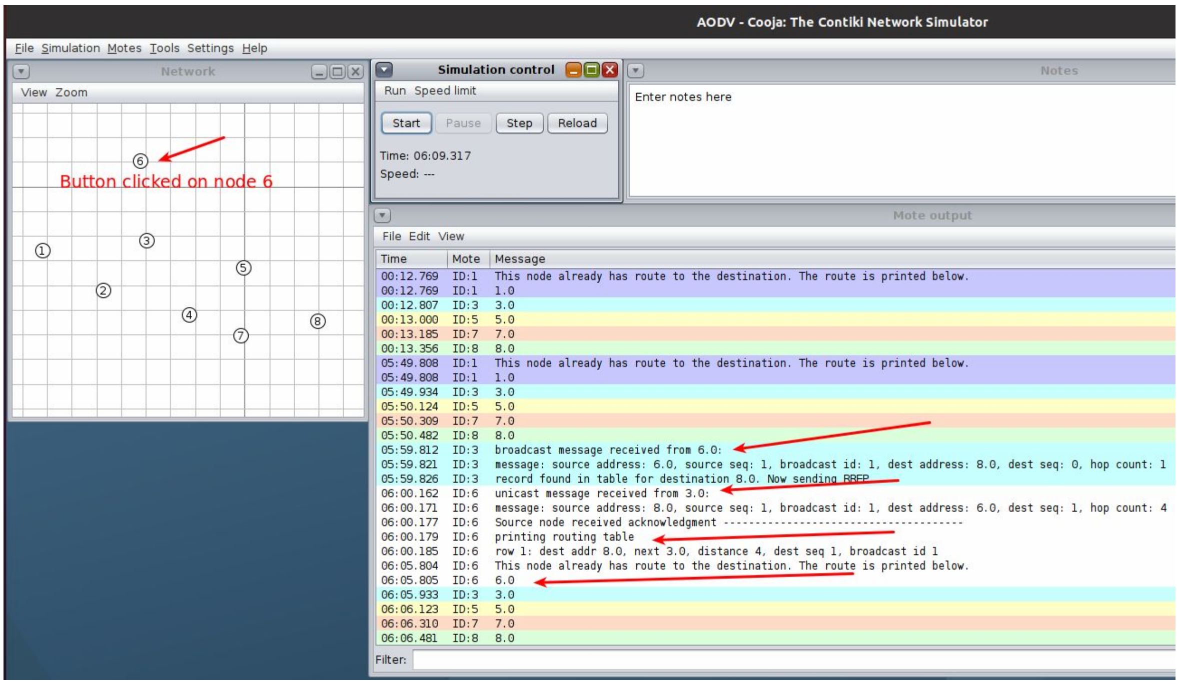
```
Time       Mote   Message
00:05.391  ID:5   printing routing table
00:05.398  ID:5   row 1: dest addr 8.0, next 7.0, distance 2, dest seq 1, broadcast id 1
00:05.404  ID:5   row 2: dest addr 1.0, next 3.0, distance 2, dest seq 1, broadcast id 1
00:05.435  ID:3   unicast message received from 5.0:
00:05.444  ID:3   message: source address: 8.0, source seq: 1, broadcast id: 1, dest address: 1.0, dest seq: 1, hop count: 3
00:05.447  ID:3   record found in table for destination 1.0
00:05.449  ID:3   printing routing table
00:05.455  ID:3   row 1: dest addr 8.0, next 5.0, distance 3, dest seq 1, broadcast id 1
00:05.462  ID:3   row 2: dest addr 1.0, next 1.0, distance 1, dest seq 1, broadcast id 1
00:05.543  ID:1   unicast message received from 3.0:
00:05.552  ID:1   message: source address: 8.0, source seq: 1, broadcast id: 1, dest address: 1.0, dest seq: 1, hop count: 4
00:05.557  ID:1   Source node received acknowledgment --------------------------------
00:05.559  ID:1   printing routing table
00:05.566  ID:1   row 1: dest addr 8.0, next 3.0, distance 4, dest seq 1, broadcast id 1
00:10.293  ID:2   Reverse pointer deleted because the node was not on the path of RREP.
00:10.294  ID:2   printing routing table
00:10.530  ID:4   Reverse pointer deleted because the node was not on the path of RREP.
00:10.532  ID:4   printing routing table
00:10.557  ID:6   Reverse pointer deleted because the node was not on the path of RREP.
00:10.559  ID:6   printing routing table
00:12.769  ID:1   This node already has route to the destination. The route is printed below.
00:12.769  ID:1   1.0
00:12.807  ID:3   3.0
00:13.000  ID:5   5.0
00:13.185  ID:7   7.0
00:13.356  ID:8   8.0
```

Figure 01

In the Figure1, node1 wants to communicate with node8, node1 will broadcast request (RReq) to its neighbouring nodes (node2 and node3). The neighbouring nodes will broadcast the request to their neighbouring nodes till the request reaches the node8. Upon receiving the request, node8 will prepare the reply packet (RRep) and unicast it backward to the nodes till it reaches the source node1. The node1 will add the entry to its routing table and print the path to the destination in the end.

Case02: Route to destination return by intermediate node

In this case, if another node asks for the route of the destination node. It will ask for the route from its neighbouring nodes by broadcasting the request. If its neighbouring nodes contain the destination route, it will return it to the source node; instead of rebroadcasting the request to the whole network.



Time: 06:09.317
Speed: ---

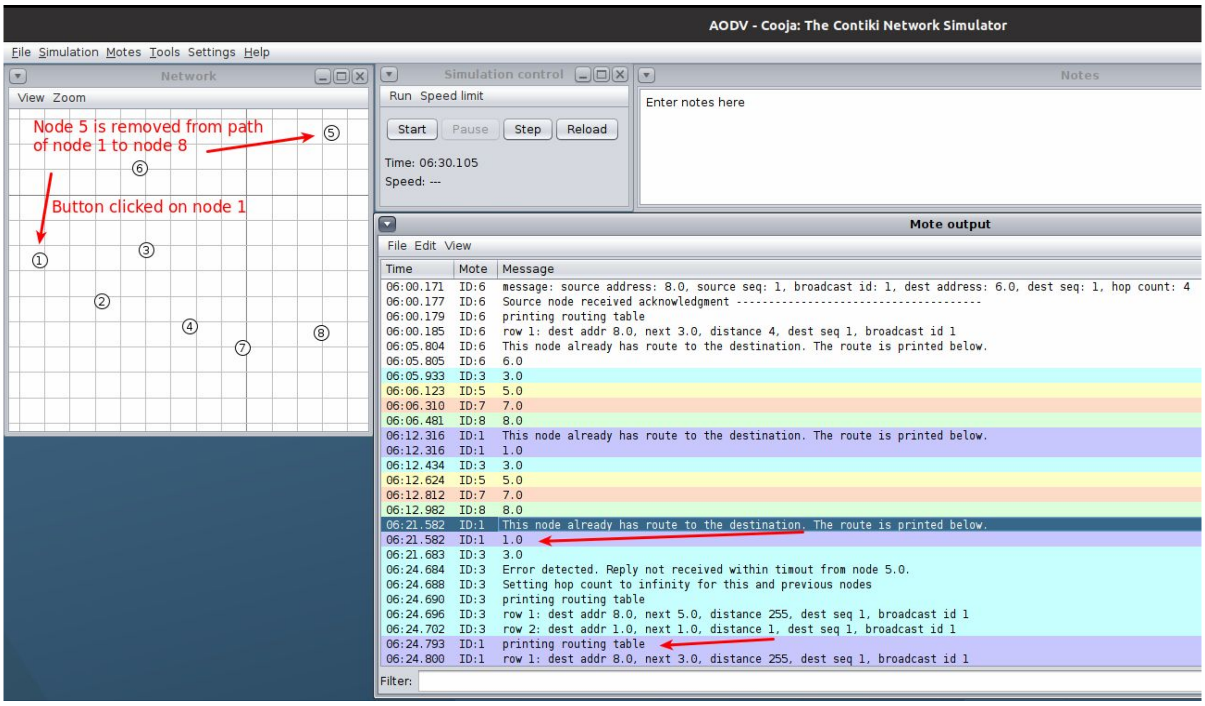
```
Time       Mote  Message
00:12.769  ID:1  This node already has route to the destination. The route is printed below.
00:12.769  ID:1  1.0
00:12.807  ID:3  3.0
00:13.000  ID:5  5.0
00:13.185  ID:7  7.0
00:13.356  ID:8  8.0
05:49.808  ID:1  This node already has route to the destination. The route is printed below.
05:49.808  ID:1  1.0
05:49.934  ID:3  3.0
05:50.124  ID:5  5.0
05:50.309  ID:7  7.0
05:50.482  ID:8  8.0
05:59.812  ID:3  broadcast message received from 6.0:
05:59.821  ID:3  message: source address: 6.0, source seq: 1, broadcast id: 1, dest address: 8.0, dest seq: 0, hop count: 1
05:59.826  ID:3  record found in table for destination 8.0. Now sending RREP
06:00.162  ID:6  unicast message received from 3.0:
06:00.171  ID:6  message: source address: 8.0, source seq: 1, broadcast id: 1, dest address: 6.0, dest seq: 1, hop count: 4
06:00.177  ID:6  Source node received acknowledgment -------------------------------------
06:00.179  ID:6  printing routing table
06:00.185  ID:6  row 1: dest addr 8.0, next 3.0, distance 4, dest seq 1, broadcast id 1
06:05.804  ID:6  This node already has route to the destination. The route is printed below.
06:05.805  ID:6  6.0
06:05.933  ID:3  3.0
06:06.123  ID:5  5.0
06:06.310  ID:7  7.0
06:06.481  ID:8  8.0
```

Figure 2

As illustrated in Figure 2, the node6 asks for the route for the node8. It will broadcast the request to its neighbour node3. As node3 already contains the route of node8 in its route table, it will return the route to source node (node6).

Case03: Remove the intermediate node

If one of the intermediate nodes is removed from the network, while the source node communicates with the destination node, the node that finds the broken link will propagate the error message up to the source node and sets the *hop count* to infinity for all nodes in that path. If the source node still wants to send the request to the destination node, it will again send a broadcast request for the new route to its destination with an incremented destination's *sequence number*.

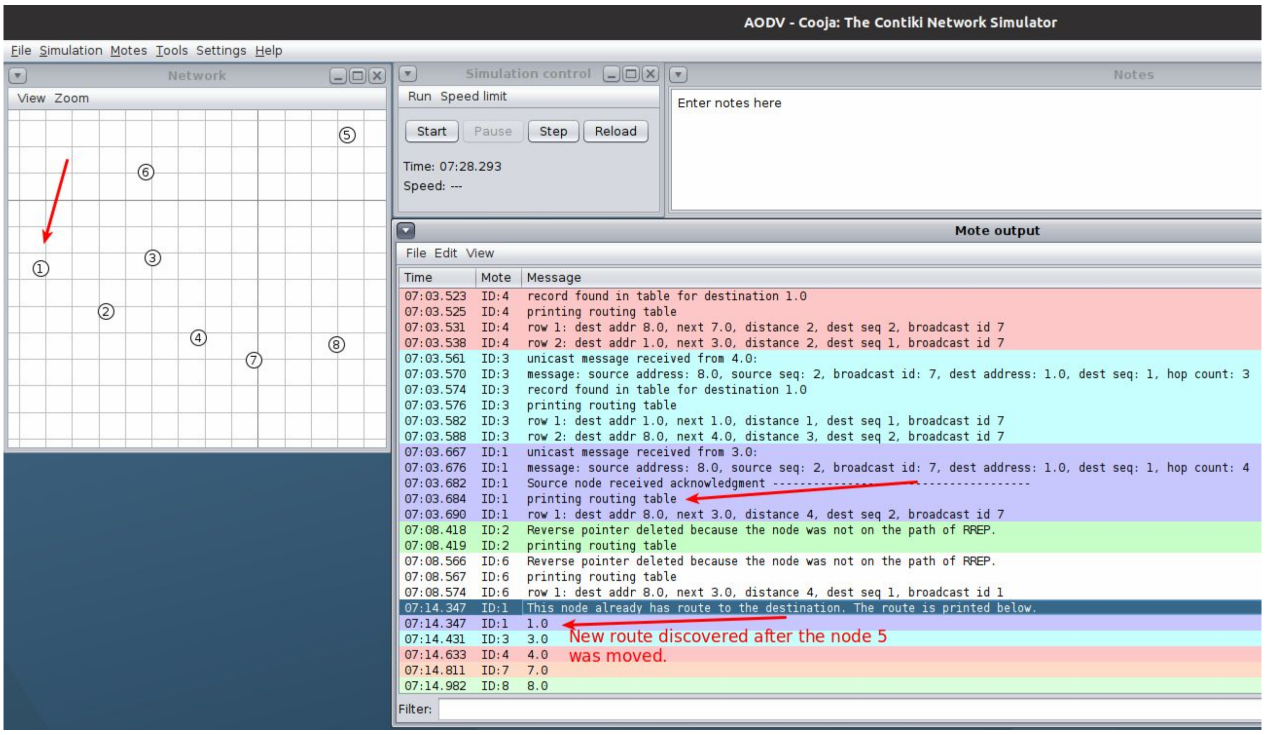


```
Time: 06:30.105
Speed: ---
```

```
Time       Mote   Message
06:00.171  ID:6   message: source address: 8.0, source seq: 1, broadcast id: 1, dest address: 6.0, dest seq: 1, hop count: 4
06:00.177  ID:6   Source node received acknowledgment -------------------------------------
06:00.179  ID:6   printing routing table
06:00.185  ID:6   row 1: dest addr 8.0, next 3.0, distance 4, dest seq 1, broadcast id 1
06:05.804  ID:6   This node already has route to the destination. The route is printed below.
06:05.805  ID:6   6.0
06:05.933  ID:3   3.0
06:06.123  ID:5   5.0
06:06.310  ID:7   7.0
06:06.481  ID:8   8.0
06:12.316  ID:1   This node already has route to the destination. The route is printed below.
06:12.316  ID:1   1.0
06:12.434  ID:3   3.0
06:12.624  ID:5   5.0
06:12.812  ID:7   7.0
06:12.982  ID:8   8.0
06:21.582  ID:1   This node already has route to the destination. The route is printed below.
06:21.582  ID:1   1.0
06:21.683  ID:3   3.0
06:24.684  ID:3   Error detected. Reply not received within timout from node 5.0.
06:24.688  ID:3   Setting hop count to infinity for this and previous nodes
06:24.690  ID:3   printing routing table
06:24.696  ID:3   row 1: dest addr 8.0, next 5.0, distance 255, dest seq 1, broadcast id 1
06:24.702  ID:3   row 2: dest addr 1.0, next 1.0, distance 1, dest seq 1, broadcast id 1
06:24.793  ID:1   printing routing table
06:24.800  ID:1   row 1: dest addr 8.0, next 3.0, distance 255, dest seq 1, broadcast id 1
```

Figure 03

The intermediate node (node5) is removed from the network, so the path between the node1 and node8 is broken. It's shown in Figure3, as the node (node3) discovers the broken link in next-hop, it will generate the error(RERR) and propagate it backwards until it reaches the source node (node1) and sets the *hop count* to infinity (maximum integer value, i.e. 255) of all the nodes in the path.



```
Time       Mote  Message
07:03.523  ID:4  record found in table for destination 1.0
07:03.525  ID:4  printing routing table
07:03.531  ID:4  row 1: dest addr 8.0, next 7.0, distance 2, dest seq 2, broadcast id 7
07:03.538  ID:4  row 2: dest addr 1.0, next 3.0, distance 2, dest seq 1, broadcast id 7
07:03.561  ID:3  unicast message received from 4.0:
07:03.570  ID:3  message: source address: 8.0, source seq: 2, broadcast id: 7, dest address: 1.0, dest seq: 1, hop count: 3
07:03.574  ID:3  record found in table for destination 1.0
07:03.576  ID:3  printing routing table
07:03.582  ID:3  row 1: dest addr 1.0, next 1.0, distance 1, dest seq 1, broadcast id 7
07:03.588  ID:3  row 2: dest addr 8.0, next 4.0, distance 3, dest seq 2, broadcast id 7
07:03.667  ID:1  unicast message received from 3.0:
07:03.676  ID:1  message: source address: 8.0, source seq: 2, broadcast id: 7, dest address: 1.0, dest seq: 1, hop count: 4
07:03.682  ID:1  Source node received acknowledgment ----------------------------------
07:03.684  ID:1  printing routing table
07:03.690  ID:1  row 1: dest addr 8.0, next 3.0, distance 4, dest seq 2, broadcast id 7
07:08.418  ID:2  Reverse pointer deleted because the node was not on the path of RREP.
07:08.419  ID:2  printing routing table
07:08.566  ID:6  Reverse pointer deleted because the node was not on the path of RREP.
07:08.567  ID:6  printing routing table
07:08.574  ID:6  row 1: dest addr 8.0, next 3.0, distance 4, dest seq 1, broadcast id 1
07:14.347  ID:1  This node already has route to the destination. The route is printed below.
07:14.347  ID:1  1.0
07:14.431  ID:3  3.0
07:14.633  ID:4  4.0
07:14.811  ID:7  7.0
07:14.982  ID:8  8.0
```

Figure 04

If the source node (node1) still wants to access the destination node (node8) it will restart the route's discovery (on clicking the button again) and increment the destination's sequence number. Then the new route is discovered to reach the destination.

References

1. Perkins, C. E., & Royer, E. M. (1999). Ad-hoc on-demand distance vector routing. *Proceedings - WMCSA'99: 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 90–100. https://doi.org/10.1109/MCSA.1999.749281