Lab 5 Report

Distance Vector Routing

Kyle Pontius

1 Implementation Explanation

My implementation of Distance Vector Routing closely followed the spec that we were given. First, I'm going to describe my nodes, then next I'll walk through the general execution chronologically.

The application I wrote to facilitate the broadcast protocol worked by keeping a routing table as well as a last-contacted list. The routing table itself was a separate class which contained two lists: first, its neighbor's routing tables and second, its own master routing table.

The chronological execution is fairly straightforward. When the network is starting, each node has an empty routing table, since it doesn't know how many neighbors it has. As each neighbor broadcasts itself, the node then records the link it received the packet on and sets the distance to 0 in its routing table. Next, as each adds itself to its routing table, it then broadcasts itself as a destination address. The neighbors then record that at a cost of 1, then forward on those entries to their neighbors. The app that runs the protocol for each node has a list of neighbors with the last-contact-time recorded for each. Each time the node hears from one its neighbors, it updates the last-contact-time for that neighbor, then checks all the neighbors to ensure that none have gone over the allowed limit of 90 seconds, in my case.

If the node has gone over that amount, the app asks the routing table to remove that link from its own routing table, then rebuilds the forwarding table. Once that entry has been removed and rebuilt, with a certain cost for that node, it broadcasts that destination and cost out to its neighbors. As a result, the neighbors will see that cost, raise it by 1, then send that out. This mechanism cascades through the network, effectively removing that link from operation once an active link, requiring less cost, is found.

Since it's difficult to recognize when the network has stopped updating their routing tables, I use a time significantly in the future of the time that the node is taken down or put up, before sending my example packets. Also, in the output that you'll see listed below, the routing table output is listed as {destination_address: cost}

2 5-node Row

Graph:

n1 – n2 – n3 – n4 – n5

Output (Routing Tables Updating):

0, n2, Updated Routing Table Values:{2: 0}
0, n1, Updated Routing Table Values:{1: 0}
0, n3, Updated Routing Table Values:{4: 0}
0, n2, Updated Routing Table Values:{2: 0, 3: 0}
0, n4, Updated Routing Table Values:{6: 0}
0, n3, Updated Routing Table Values:{4: 0, 5: 0}

0, n5, Updated Routing Table Values:{8: 0}
0, n4, Updated Routing Table Values:{6: 0, 7: 0}
30, n2, Updated Routing Table Values:{1: 1, 2: 0, 3: 0}
30, n1, Updated Routing Table Values:{1: 0, 2: 1, 3: 1}
30, n3, Updated Routing Table Values:{2: 1, 3: 1, 4: 0, 5: 0}
30, n2, Updated Routing Table Values:{1: 1, 2: 0, 3: 0, 4: 1, 5: 1}
30, n4, Updated Routing Table Values:{4: 1, 5: 1, 6: 0, 7: 0}
30, n3, Updated Routing Table Values:{2: 1, 3: 1, 4: 0, 5: 0, 6: 1, 7: 1}
30, n5, Updated Routing Table Values:{8: 0, 6: 1, 7: 1}
30, n4, Updated Routing Table Values:{8: 1, 4: 1, 5: 1, 6: 0, 7: 0}
60, n2, Updated Routing Table Values:{1: 1, 2: 0, 3: 0, 4: 1, 5: 1}
60, n1, Updated Routing Table Values:{1: 0, 2: 1, 3: 1, 4: 2, 5: 2}
60, n3, Updated Routing Table Values:{1: 2, 2: 1, 3: 1, 4: 0, 5: 0, 6: 1, 7: 1}
60, n2, Updated Routing Table Values:{1: 1, 2: 0, 3: 0, 4: 1, 5: 1, 6: 2, 7: 2}
60, n4, Updated Routing Table Values:{2: 2, 3: 2, 4: 1, 5: 1, 6: 0, 7: 0, 8: 1}
60, n3, Updated Routing Table Values:{1: 2, 2: 1, 3: 1, 4: 0, 5: 0, 6: 1, 7: 1, 8: 2}
60, n5, Updated Routing Table Values:{8: 0, 4: 2, 5: 2, 6: 1, 7: 1}
60, n4, Updated Routing Table Values:{2: 2, 3: 2, 4: 1, 5: 1, 6: 0, 7: 0, 8: 1}
90, n2, Updated Routing Table Values:{1: 1, 2: 0, 3: 0, 4: 1, 5: 1, 6: 2, 7: 2}
90, n1, Updated Routing Table Values:{1: 0, 2: 1, 3: 1, 4: 2, 5: 2, 6: 3, 7: 3}
90, n3, Updated Routing Table Values:{1: 2, 2: 1, 3: 1, 4: 0, 5: 0, 6: 1, 7: 1, 8: 2}
90, n2, Updated Routing Table Values:{1: 1, 2: 0, 3: 0, 4: 1, 5: 1, 6: 2, 7: 2, 8: 3}
90, n4, Updated Routing Table Values:{1: 3, 2: 2, 3: 2, 4: 1, 5: 1, 6: 0, 7: 0, 8: 1}
90, n3, Updated Routing Table Values:{1: 2, 2: 1, 3: 1, 4: 0, 5: 0, 6: 1, 7: 1, 8: 2}
90, n5, Updated Routing Table Values:{2: 3, 3: 3, 4: 2, 5: 2, 6: 1, 7: 1, 8: 0}
90, n4, Updated Routing Table Values:{1: 3, 2: 2, 3: 2, 4: 1, 5: 1, 6: 0, 7: 0, 8: 1}
120, n2, Updated Routing Table Values:{1: 1, 2: 0, 3: 0, 4: 1, 5: 1, 6: 2, 7: 2, 8: 3}
…

Explanation:

This output is a print out of each nodes' new routing table after it receives a broadcast packet. The network is nearly complete at this point. To illustrate that behavior of the routing tables, I highlighted *n1*'s updates to shows its growth as it receives new addresses from its neighbors.

Output (Sending Packet):

…
900.0 - (n1) Packet Forwarded - Data: Hello world!; Source_Address: 1; Destination_Address: 8
900.001096 - (n2) Packet Forwarded - Data: Hello world!; Source_Address: 1; Destination_Address: 8
900.002192 - (n3) Packet Forwarded - Data: Hello world!; Source_Address: 1; Destination_Address: 8
900.003288 - (n4) Packet Forwarded - Data: Hello world!; Source_Address: 1; Destination_Address: 8
900.004384 - (n5) Packet ARRIVED - Data: Hello world!; Source_Address: 1; Destination_Address: 8
…

Explanation:

After the network was set up, there was a packet sent from *n1* to *n5*. This packet starts in *n1* passed through *n2, n3, n4,* and finally arrives in *n5*.

3 5-node Ring

Graph:

```
n1 – n3 – n5
  \       /
   n2 – n4
```

Output (Routing Tables Updating):

…
0, n2, Updated Routing Table Values:{10: 0}
0, n3, Updated Routing Table Values:{3: 0}
0, n4, Updated Routing Table Values:{8: 0}
0, n1, Updated Routing Table Values:{1: 0}
0, n1, Updated Routing Table Values:{1: 0, 2: 0}
0, n5, Updated Routing Table Values:{5: 0}
0, n5, Updated Routing Table Values:{5: 0, 6: 0}
0, n2, Updated Routing Table Values:{9: 0, 10: 0}
0, n3, Updated Routing Table Values:{3: 0, 4: 0}
0, n4, Updated Routing Table Values:{8: 0, 7: 0}
30, n2, Updated Routing Table Values:{9: 0, 10: 0, 2: 1, 1: 1}
30, n3, Updated Routing Table Values:{1: 1, 2: 1, 3: 0, 4: 0}
30, n4, Updated Routing Table Values:{8: 0, 9: 1, 10: 1, 7: 0}
30, n1, Updated Routing Table Values:{1: 0, 2: 0, 10: 1, 9: 1}
30, n1, Updated Routing Table Values:{1: 0, 2: 0, 3: 1, 4: 1, 9: 1, 10: 1}
30, n5, Updated Routing Table Values:{3: 1, 4: 1, 5: 0, 6: 0}
30, n5, Updated Routing Table Values:{3: 1, 4: 1, 5: 0, 6: 0, 7: 1, 8: 1}
30, n2, Updated Routing Table Values:{1: 1, 2: 1, 7: 1, 8: 1, 9: 0, 10: 0}
30, n3, Updated Routing Table Values:{1: 1, 2: 1, 3: 0, 4: 0, 5: 1, 6: 1}
30, n4, Updated Routing Table Values:{5: 1, 6: 1, 7: 0, 8: 0, 9: 1, 10: 1}
…

Explanation:

This output shows the network starting with no routing tables set up. You can see each node beginning with just an entry for itself, then slowly updating with several more nodes as the broadcasts starting coming in from their neighbors. I've highlighted ***n1***'s progress in yellow and ***n2***'s in green as just a couple examples of the protocol functioning properly.

Output (Sending Packet):

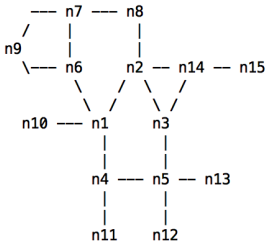
…
900.0 - (n1) Packet Forwarded - Data: Hello world!; Source_Address: 1; Destination_Address: 10
900.001096 - (n2) Packet ARRIVED - Data: Hello world!; Source_Address: 1; Destination_Address: 10
1500.0 - ----> DISABLED LINKS <----
1560.001016 - Removing last_contact_list entry: n2
1590.001016 - Removing last_contact_list entry: n1
2800.0 - (n1) Packet Forwarded - Data: Hello world!; Source_Address: 1; Destination_Address: 10
2800.001096 - (n3) Packet Forwarded - Data: Hello world!; Source_Address: 1; Destination_Address: 10
2800.002192 - (n5) Packet Forwarded - Data: Hello world!; Source_Address: 1; Destination_Address: 10
2800.003288 - (n4) Packet Forwarded - Data: Hello world!; Source_Address: 1; Destination_Address: 10
2800.004384 - (n2) Packet ARRIVED - Data: Hello world!; Source_Address: 1; Destination_Address: 10
…

Explanation:

This packet is generated on *n1* and sent to *n2*. The packet is first sent before the link has been taken down between *n1* and *n2*, and the network has been rearranged. Next, another packet is sent about 1000 seconds later. As you can see from the output, on the first attempt the packet goes immediately to *n2* from *n1*. On the second attempt, the packet starts at *n1* then goes to *n3, n5, n4,* and finally *n2*.

4 15-node Mesh

Graph:



Output (Routing Tables Updating):

0, n4, Updated Routing Table Values:{12: 0}
0, n4, Updated Routing Table Values:{12: 0, 13: 0}
0, n4, Updated Routing Table Values:{12: 0, 13: 0, 14: 0}
30, n4, Updated Routing Table Values:{1: 1, 2: 1, 3: 1, 4: 1, 12: 0, 13: 0, 14: 0}
30, n4, Updated Routing Table Values:
{1: 1, 2: 1, 3: 1, 4: 1, 12: 0, 13: 0, 14: 0, 15: 1, 16: 1, 17: 1, 18: 1}
30, n4, Updated Routing Table Values:
{1: 1, 2: 1, 3: 1, 4: 1, 12: 0, 13: 0, 14: 0, 15: 1, 16: 1, 17: 1, 18: 1, 30: 1}
60, n4, Updated Routing Table Values:
{1: 1, 2: 1, 3: 1, 4: 1, 5: 2, 6: 2, 7: 2, 8: 2, 12: 0, 13: 0, 14: 0, 15: 1, 16: 1, 17: 1, 18: 1, 19: 2, 20: 2, 21: 2, 29: 2, 30: 1}
…

Explanation:

Since this network has significantly more output than the other two networks, I opted to extract just *n4*'s output. Above, you can see how *n4*'s routing table gets updated each time it receives a broadcast packet. It's interesting to note than several packets hit at the same simulated time and the routing table grows several addresses in just a single step.

Output (Sending Packet):

900.0 - (n11) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
900.001096 - (n4) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
900.002192 - (n1) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
900.003288 - (n10) Packet ARRIVED - Data: Hello world!; Source_Address: 30; Destination_Address: 29
2250.0 - ----> DISABLED LINKS <----
2310.001016 - Removing last_contact_list entry: n4
2340.001016 - Removing last_contact_list entry: n1
2800.0 - (n11) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
2800.001096 - (n4) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
2800.002192 - (n5) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29

2800.003288 - (n3) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
2800.004384 - (n2) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
2800.00548 - (n1) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
2800.006576 - (n10) Packet ARRIVED - Data: Hello world!; Source_Address: 30; Destination_Address: 29
4500.0 - ----> ENABLED LINKS <----
5500.0 - (n11) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
5500.001096 - (n4) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
5500.002192 - (n1) Packet Forwarded - Data: Hello world!; Source_Address: 30; Destination_Address: 29
5500.003288 - (n10) Packet ARRIVED - Data: Hello world!; Source_Address: 30; Destination_Address: 29

Explanation:

Once the network has been optimized, the first packet is sent. The initial path is *n11, n4, n1,* then arriving at *n10*. Next, the link between *n1* and *n4* is taken down and the network readjusts itself. Following that process, a second packet is sent and its path is now optimized to *n11, n4, n5, n3, n2, n1,* then *n10*. Finally, the link is re-enabled ("→ ENABLED LINKS ←"), and the packet is sent after the network readjusts again. The path has returned to the originally optimized path of *n11, n4, n1,* then *n10*.