Routing

Brandt Elison
Joe Eklund

April 12, 2016

1 Introduction

For this project, we implemented a distance vector routing (DVR) protocol for nodes in a network to discover the best routes to other nodes. This protocol works by each node broadcasting a distance vector to its neighbors. This distance vector tells neighbors what it would cost to get to other nodes in the network from the current node. Nodes use their neighbors distance vectors to build efficient routing tables for forwarding packets.

2 Experiments

To demonstrate that our DVR implementation works as it should, we ran a series of routing tests. We built three different networks shown below in snippets 1-3.

Snippet 1: 5 nodes in a line

```
Topology:
  n1--n2--n3--n4--n5

Link Addresses:
  n1->n2 = 0
  n2->n1 = 1
```

Snippet 2: 5 nodes in a ring

```
      n3
    /    \
  n2      n4
   |       |
  n1 ---- n5
```

Snippet 3: 5 node mesh

```
     ---- n7 ---- n8
    /     |        |
  n9      |        |
    \---- n6      n2 -- n14 -- n15
           \     /  \    /
            \   /    \  /
   n10 ---- n1      n3
             |       |
             |       |
                n4 ——— n5 —— n13
                 |      |
                 |      |
                n11    n12
```

We ran our DVR implementation on each of these networks and verified that routes were correctly established by outputting the state of the nodes both during and after identifying the entire network.

2.1 Five Node Line

These experiments relate to the network in Snippet 1. After the network was setup, the nodes were given 10 simulator minutes to build their routing tables, then a packet was sent from n1 to n5. The results are shown in snippets 4-6.

Snippet 4: The path taken when transmitting a packet from n1 to n5

```
========================
600.001064 n1 forwarding packet to 8
600.002864 n2 forwarding packet to 8
600.004664 n3 forwarding packet to 8
600.006464 n4 forwarding packet to 8
600.008264 n5 received packet
```

Snippet 5: Example output of nodes updating their distance vectors based on neighbor broadcasts

```
===================================
n2 receiving a new dv from address 1
Vector received:  {1: 0}
DV before update: {2: 0, 3: 0}
DV after update:  {1: 1, 2: 0, 3: 0}
===================================
===================================
n4 receiving a new dv from address 8
Vector received:  {8: 0}
DV before update: {6: 0, 7: 0}
DV after update:  {8: 1, 6: 0, 7: 0}
===================================
===================================
n1 receiving a new dv from address 2
Vector received:  {2: 0, 3: 0}
DV before update: {1: 0}
DV after update:  {1: 0, 2: 1, 3: 1}
===================================
===================================
n3 receiving a new dv from address 3
Vector received:  {2: 0, 3: 0}
DV before update: {4: 0, 5: 0}
DV after update:  {2: 1, 3: 1, 4: 0, 5: 0}
===================================
===================================
n2 receiving a new dv from address 4
Vector received:  {4: 0, 5: 0}
DV before update: {1: 1, 2: 0, 3: 0}
DV after update:  {1: 1, 2: 0, 3: 0, 4: 1, 5: 1}
===================================
```

Snippet 6: The final distance vectors and forwarding tables for each node in the network

```
==================================
Final DV for n1: {1: 0, 2: 1, 3: 1, 4: 2, 5: 2, 6: 3, 7: 3, 8: 4}
Final DV for n2: {1: 1, 2: 0, 3: 0, 4: 1, 5: 1, 6: 2, 7: 2, 8: 3}
Final DV for n3: {1: 2, 2: 1, 3: 1, 4: 0, 5: 0, 6: 1, 7: 1, 8: 2}
Final DV for n4: {1: 3, 2: 2, 3: 2, 4: 1, 5: 1, 6: 0, 7: 0, 8: 1}
Final DV for n5: {1: 4, 2: 3, 3: 3, 4: 2, 5: 2, 6: 1, 7: 1, 8: 0}
==================================
Final forwarding table for n1: {2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1}
Final forwarding table for n2: {1: 2, 4: 3, 5: 3, 6: 3, 7: 3, 8: 3}
Final forwarding table for n3: {1: 4, 2: 4, 3: 4, 6: 5, 7: 5, 8: 5}
Final forwarding table for n4: {1: 6, 2: 6, 3: 6, 4: 6, 5: 6, 8: 7}
Final forwarding table for n5: {1: 8, 2: 8, 3: 8, 4: 8, 5: 8, 6: 8, 7: 8}
```

Snippet 4 shows that we were able to successfully send a packet from n1 to n5. The output says they are "forwarding...to 8", but this is address 8 which belongs to node 5. The nodes were able to correctly establish forwarding tables.

Snippet 5 shows a few examples of nodes receiving a neighbor's distance vector and updating their own appropriately. They key/value pairs map addresses to distances rather than nodes to distances, which is why the keys go up to 8 rather than 4. These outputs show that the nodes are updating as expected.

Snippet 6 shows the final distance vector and forwarding table for each node in the network. The forwarding table key/value pairs map target address to outgoing link address. By comparing Snippet 6 to Snippet 1, we can see that the distance vectors all describe the appropriate distances and the forwarding tables give the correct routes.

2.2 Five Node Ring

These experiments relate to the network in Snippet 2. After the network was setup, the following steps took place:

1. The nodes were given 5 simulator minutes to build their routing tables.
2. A packet was sent from n1 to n5.
3. Ten seconds later, the link between nodes n1 and n5 was dropped.
4. The network was given 5 more minutes to rebuild routing tables.
5. Another packet was sent from n1 to n5.

After the link is dropped, we expect the network to adjust its routes and distance vectors automatically. This is shown below in snippets 7-9.

Snippet 7: The path taken when transmitting a packet from n1 to n5 before and after dropping a link

```
300.00108 n1 forwarding packet to 10
300.00288 n5 received packet
310.00108 Deactivating link from n1 to n5
310.00108 Deactivating link from n5 to n1
610.00216 n1 forwarding packet to 10
610.00396 n2 forwarding packet to 10
610.00576 n3 forwarding packet to 10
610.00756 n4 forwarding packet to 10
610.00936 n5 received packet
```

Snippet 8: The final distance vectors and forwarding tables for each node before dropping a link

```
==============================
Final DV for n1: {1: 0, 2: 0, 3: 1, 4: 1, 5: 2, 6: 2, 7: 2, 8: 2, 9: 1, 10: 1}
Final DV for n2: {1: 1, 2: 1, 3: 0, 4: 0, 5: 1, 6: 1, 7: 2, 8: 2, 9: 2, 10: 2}
Final DV for n3: {1: 2, 2: 2, 3: 1, 4: 1, 5: 0, 6: 0, 7: 1, 8: 1, 9: 2, 10: 2}
Final DV for n4: {1: 2, 2: 2, 3: 2, 4: 2, 5: 1, 6: 1, 7: 0, 8: 0, 9: 1, 10: 1}
Final DV for n5: {1: 1, 2: 1, 3: 2, 4: 2, 5: 2, 6: 2, 7: 1, 8: 1, 9: 0, 10: 0}
==============================
Final forwarding table for n1: {3: 1, 4: 1, 5: 1, 6: 1, 7: 2, 8: 2, 9: 2, 10: 2}
Final forwarding table for n2: {1: 3, 2: 3, 5: 4, 6: 4, 7: 4, 8: 4, 9: 3, 10: 3}
Final forwarding table for n3: {1: 5, 2: 5, 3: 5, 4: 5, 7: 6, 8: 6, 9: 6, 10: 6}
Final forwarding table for n4: {1: 8, 2: 8, 3: 7, 4: 7, 5: 7, 6: 7, 9: 8, 10: 8}
Final forwarding table for n5: {1: 10, 2: 10, 3: 10, 4: 10, 5: 9, 6: 9, 7: 9, 8: 9}
```

Snippet 9: The final distance vectors and forwarding tables for each node after dropping a link

```
==============================
Final DV for n1: {1: 0, 2: 0, 3: 1, 4: 1, 5: 2, 6: 2, 7: 3, 8: 3, 9: 4, 10: 4}
Final DV for n2: {1: 1, 2: 1, 3: 0, 4: 0, 5: 1, 6: 1, 7: 2, 8: 2, 9: 3, 10: 3}
Final DV for n3: {1: 2, 2: 2, 3: 1, 4: 1, 5: 0, 6: 0, 7: 1, 8: 1, 9: 2, 10: 2}
Final DV for n4: {1: 3, 2: 3, 3: 2, 4: 2, 5: 1, 6: 1, 7: 0, 8: 0, 9: 1, 10: 1}
Final DV for n5: {1: 4, 2: 4, 3: 3, 4: 3, 5: 2, 6: 2, 7: 1, 8: 1, 9: 0, 10: 0}
==============================
Final forwarding table for n1: {3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1}
Final forwarding table for n2: {1: 3, 2: 3, 5: 4, 6: 4, 7: 4, 8: 4, 9: 4, 10: 4}
Final forwarding table for n3: {1: 5, 2: 5, 3: 5, 4: 5, 7: 6, 8: 6, 9: 6, 10: 6}
Final forwarding table for n4: {1: 7, 2: 7, 3: 7, 4: 7, 5: 7, 6: 7, 9: 8, 10: 8}
Final forwarding table for n5: {1: 9, 2: 9, 3: 9, 4: 9, 5: 9, 6: 9, 7: 9, 8: 9}
```

Snippet 7 shows that initially the packet is transmitted directly from n1 to n5, but after the link between those nodes is dropped, the network learns the new route to n5 and is able to forward the packet from n1 to n5 by going through each of the other nodes.

Comparing snippets 8 and 9 show how each node individually adjusted its understanding of the network when the link was dropped. For example, the distance vector for n1 gets updated so that the distance to address 10 (at n5) gets changed from 1 to 4, and the forwarding table gets changed to transmit on link 1 instead of link 2.

2.3 Fifteen Node

These experiments relate to the network in Snippet 3. After the network was setup, the following steps took place:

1. The nodes were given 5 simulator minutes to build their routing tables.
2. A packet was sent from n1 to n12.
3. Ten seconds later, the link between nodes n4 and n5 was dropped.
4. The network was given 5 more minutes to rebuild routing tables.
5. Another packet was sent from n1 to n12.
6. Ten seconds later, the link between nodes n4 and n5 was readded.
7. The network was given 5 more minutes to rebuild routing tables.

8. Another packet was sent from n1 to n12.

After the link is both dropped and readded, we expect the network to adjust its routes and distance vectors automatically. This is shown below in snippets 10-11.

Snippet 10: The path taken when transmitting packets from n1 to n5 before and after each link change

```
300.001288 n1 forwarding packet to 31
300.003088 n4 forwarding packet to 31
300.004888 n5 forwarding packet to 31
300.006688 n12 received packet
310.001288 Deactivating link from n5 to n4
310.001288 Deactivating link from n4 to n5
610.002576 n1 forwarding packet to 31
610.004376 n2 forwarding packet to 31
610.006176 n3 forwarding packet to 31
610.007976 n5 forwarding packet to 31
610.009776 n12 received packet
620.002576 Activating link from n5 to n4
620.002576 Activating link from n4 to n5
920.003864 n1 forwarding packet to 31
920.005664 n4 forwarding packet to 31
920.007464 n5 forwarding packet to 31
920.009264 n12 received packet
```

```
==================================
n5 receiving a new dv from address 31
Vector received:  {31: 0}
DV before update: {16: 0, 17: 0, 18: 0, 15: 0}
DV after update:  {16: 0, 17: 0, 18: 0, 31: 1, 15: 0}
==================================
==================================
n5 receiving a new dv from address 32
Vector received:  {32: 0}
DV before update: {16: 0, 17: 0, 18: 0, 31: 1, 15: 0}
DV after update:  {32: 1, 15: 0, 16: 0, 17: 0, 18: 0, 31: 1}
==================================
==================================
n5 receiving a new dv from address 10
Vector received:  {9: 0, 10: 0, 11: 0}
DV before update: {32: 1, 15: 0, 16: 0, 17: 0, 18: 0, 31: 1}
DV after update:  {32: 1, 9: 1, 10: 1, 11: 1, 15: 0, 16: 0, 17: 0, 18: 0, 31: 1}
==================================

...
...

==================================
n5 receiving a new dv from address 10
Vector received:  {33: 1, 34: 1, 35: 1, 36: 2, 5: 1, 6: 1, 7: 1, 8: 1, 9: 0, 10: 0,
    11: 0, 15: 1, 16: 1, 17: 1, 18: 1}
DV before update: {1: 2, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 1, 10: 1, 11:
    1, 12: 1, 13: 1, 14: 1, 15: 0, 16: 0, 17: 0, 18: 0, 29: 3, 30: 2, 31: 1, 32: 1,
    33: 2, 34: 2, 35: 2}
DV after update:  {1: 2, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 1, 10: 1, 11:
    1, 12: 1, 13: 1, 14: 1, 15: 0, 16: 0, 17: 0, 18: 0, 29: 3, 30: 2, 31: 1, 32: 1,
    33: 2, 34: 2, 35: 2, 36: 3}
==================================
==============================
n5 receiving a new dv from address 31
Vector received:  {32: 2, 9: 2, 10: 2, 11: 2, 15: 1, 16: 1, 17: 1, 18: 1, 31: 0}
DV before update: {1: 2, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 1, 10: 1, 11:
    1, 12: 1, 13: 1, 14: 1, 15: 0, 16: 0, 17: 0, 18: 0, 29: 3, 30: 2, 31: 1, 32: 1,
    33: 2, 34: 2, 35: 2, 36: 3}
DV after update:  {1: 2, 2: 2, 3: 2, 4: 2, 5: 2, 6: 2, 7: 2, 8: 2, 9: 1, 10: 1, 11:
    1, 12: 1, 13: 1, 14: 1, 15: 0, 16: 0, 17: 0, 18: 0, 29: 3, 30: 2, 31: 1, 32: 1,
    33: 2, 34: 2, 35: 2, 36: 3}
==============================
```

Snippet 10 shows that the first packet correctly takes the shortest path from n1 to n12, and that the second packet correctly reroutes to the next shortest path after the link is dropped. We also see that the final packet, which is sent after the link is restored, was able to take the original shortest path from n1 to n12.