CS 460: Routing

Kevin Haroldsen

1 Introduction

Routing is necessary for any large network of interconnected computers to function efficiently. There are many protocols that can be used to coordinate costs of sending from one node to the next; however, this lab will focus on the distance-vector algorithm.

2 Distance-Vector Algorithm

The distance-vector algorithm is a decentralized algorithm for computing best-cost paths from each node to each other node. Each node starts by knowing the costs to its own neighbors. Then, at determined intervals, each node sends a *distance vector* to each of its neighbors. This distance vector is the currently known best-cost paths to each other node from that node.

Upon receiving another distance vector, a node will compare its current best-cost path with the best-cost path of its neighbor, plus the cost to get to the neighbor. If the best-cost path of its neighbor, plus the cost to get to the neighbor, is less than its current best-cost path to a destination, it will set its distance vector to this value and add an entry in the forwarding table to use that link to forward to the destination.

Eventually, the best-cost information will be distributed throughout the network and every node will know the best cost to every other node, and through which link to forward.

The distance vectors continue to be sent on a periodic basis, and when enough time passes without a node hearing from its neighbor, the link is declared dead and blacklisted.

2.1 Implementation Details

There's a bit of confusion about hostnames and addresses I've found. First, I've standardized it so that the links a node can receive on are that node's addresses.

Next, in order to be able to address nodes that may have multiple addresses, information about which link addresses correspond to which hostnames also propagates throughout the network. When sending to an host, it looks at the costs to reach all of the links, known to be associated with that hostname, to determine which link to send to.

Lastly, I've added a "quick-update", in which nodes receiving new routing information will immediately re-broadcast to their neighbors, rather than wait a period of time. This allows the initial startup time for full efficiency to be extremely low.

3 Experimentation

3.1 Five Nodes

Here is the output from processing five nodes in a row. It is verified that each node can reach each other node, and the DVR packets can be seen being transferred at the beginning.

```
All links:
Link<(1) n1 -> n2>
Link<(2) n2 -> n1>
Link<(3) n2 -> n3>
Link<(4) n3 -> n2>
Link<(5) n3 -> n4>
Link<(6) n4 -> n3>
Link<(7) n4 -> n5>
Link<(8) n5 -> n4>
0.002 dvr n1 -> n2 ({1: 1, 2: 0, 4: 1})
0.002 dvr n3 -> n2 ({1: 1, 3: 0, 4: 1, 6: 0})
0.002 dvr n3 -> n4 ({1: 1, 3: 0, 4: 1, 6: 0})
0.002 dvr n5 -> n4 ({8: 1, 5: 1, 7: 0})
0.002 dvr n4 -> n3 ({8: 0, 5: 0, 7: 1})
0.002 dvr n4 -> n5 ({8: 0, 5: 0, 7: 1})
0.002 dvr n2 -> n1 ({1: 0, 3: 1, 4: 0, 6: 1})
0.002 dvr n2 -> n3 ({1: 0, 3: 1, 4: 0, 6: 1})
0.002 dvr n3 -> n2 ({1: 1, 3: 0, 4: 1, 5: 1, 6: 0, 8: 1})
0.002 dvr n3 -> n4 ({1: 1, 3: 0, 4: 1, 5: 1, 6: 0, 8: 1})
0.002 dvr n4 -> n3 ({8: 0, 3: 1, 5: 0, 6: 1, 7: 1})
0.002 dvr n4 -> n5 ({8: 0, 3: 1, 5: 0, 6: 1, 7: 1})
0.002 dvr n2 -> n1 ({1: 0, 2: 1, 3: 1, 4: 0, 6: 1})
0.002 dvr n2 -> n3 ({1: 0, 2: 1, 3: 1, 4: 0, 6: 1})
0.003 dvr n4 -> n3 ({1: 2, 3: 1, 4: 2, 5: 0, 6: 1, 7: 1, 8: 0})
0.003 dvr n4 -> n5 ({1: 2, 3: 1, 4: 2, 5: 0, 6: 1, 7: 1, 8: 0})
0.003 dvr n3 -> n2 ({1: 1, 3: 0, 4: 1, 5: 1, 6: 0, 7: 2, 8: 1})
0.003 dvr n3 -> n4 ({1: 1, 3: 0, 4: 1, 5: 1, 6: 0, 7: 2, 8: 1})
0.003 dvr n1 -> n2 ({1: 1, 2: 0, 3: 2, 4: 1, 6: 2})
0.003 dvr n2 -> n1 ({1: 0, 2: 1, 3: 1, 4: 0, 5: 2, 6: 1, 8: 2})
0.003 dvr n2 -> n3 ({1: 0, 2: 1, 3: 1, 4: 0, 5: 2, 6: 1, 8: 2})
0.003 dvr n5 -> n4 ({8: 1, 3: 2, 5: 1, 6: 2, 7: 0})
0.003 dvr n3 -> n2 ({1: 1, 2: 2, 3: 0, 4: 1, 5: 1, 6: 0, 7: 2, 8: 1})
0.003 dvr n3 -> n4 ({1: 1, 2: 2, 3: 0, 4: 1, 5: 1, 6: 0, 7: 2, 8: 1})
0.004 dvr n5 -> n4 ({1: 3, 3: 2, 4: 3, 5: 1, 6: 2, 7: 0, 8: 1})
0.004 dvr n2 -> n1 ({1: 0, 2: 1, 3: 1, 4: 0, 5: 2, 6: 1, 7: 3, 8: 2})
0.004 dvr n2 -> n3 ({1: 0, 2: 1, 3: 1, 4: 0, 5: 2, 6: 1, 7: 3, 8: 2})
0.004 dvr n1 -> n2 ({1: 1, 2: 0, 3: 2, 4: 1, 5: 3, 6: 2, 8: 3})
0.004 dvr n4 -> n3 ({1: 2, 2: 3, 3: 1, 4: 2, 5: 0, 6: 1, 7: 1, 8: 0})
0.004 dvr n4 -> n5 ({1: 2, 2: 3, 3: 1, 4: 2, 5: 0, 6: 1, 7: 1, 8: 0})
0.005 dvr n1 -> n2 ({1: 1, 2: 0, 3: 2, 4: 1, 5: 3, 6: 2, 7: 4, 8: 3})
0.005 dvr n5 -> n4 ({1: 3, 2: 4, 3: 2, 4: 3, 5: 1, 6: 2, 7: 0, 8: 1})

== Sending packet n1 -> n2 ==
6.0 hop Link<(1) n1 -> n2>

== Sending packet n1 -> n3 ==
7.0 hop Link<(1) n1 -> n2>
7.001 hop Link<(3) n2 -> n3>

== Sending packet n1 -> n4 ==
```

```
8.0 hop Link<(1) n1 -> n2>
8.001 hop Link<(3) n2 -> n3>
8.002 hop Link<(5) n3 -> n4>

== Sending packet n1 -> n5 ==
9.0 hop Link<(1) n1 -> n2>
9.001 hop Link<(3) n2 -> n3>
9.002 hop Link<(5) n3 -> n4>
9.003 hop Link<(7) n4 -> n5>

== Sending packet n2 -> n1 ==
10.0 hop Link<(2) n2 -> n1>

== Sending packet n2 -> n3 ==
11.0 hop Link<(3) n2 -> n3>

== Sending packet n2 -> n4 ==
12.0 hop Link<(3) n2 -> n3>
12.001 hop Link<(5) n3 -> n4>

== Sending packet n2 -> n5 ==
13.0 hop Link<(3) n2 -> n3>
13.001 hop Link<(5) n3 -> n4>
13.002 hop Link<(7) n4 -> n5>

== Sending packet n3 -> n1 ==
14.0 hop Link<(4) n3 -> n2>
14.001 hop Link<(2) n2 -> n1>

== Sending packet n3 -> n2 ==
15.0 hop Link<(4) n3 -> n2>

== Sending packet n3 -> n4 ==
16.0 hop Link<(5) n3 -> n4>

== Sending packet n3 -> n5 ==
17.0 hop Link<(5) n3 -> n4>
17.001 hop Link<(7) n4 -> n5>

== Sending packet n4 -> n1 ==
18.0 hop Link<(6) n4 -> n3>
18.001 hop Link<(4) n3 -> n2>
18.002 hop Link<(2) n2 -> n1>

== Sending packet n4 -> n2 ==
19.0 hop Link<(6) n4 -> n3>
19.001 hop Link<(4) n3 -> n2>

== Sending packet n4 -> n3 ==
20.0 hop Link<(6) n4 -> n3>
```

```
== Sending packet n4 -> n5 ==
21.0 hop Link<(7) n4 -> n5>

== Sending packet n5 -> n1 ==
22.0 hop Link<(8) n5 -> n4>
22.001 hop Link<(6) n4 -> n3>
22.002 hop Link<(4) n3 -> n2>
22.003 hop Link<(2) n2 -> n1>

== Sending packet n5 -> n2 ==
23.0 hop Link<(8) n5 -> n4>
23.001 hop Link<(6) n4 -> n3>
23.002 hop Link<(4) n3 -> n2>

== Sending packet n5 -> n3 ==
24.0 hop Link<(8) n5 -> n4>
24.001 hop Link<(6) n4 -> n3>

== Sending packet n5 -> n4 ==
25.0 hop Link<(8) n5 -> n4>

Process finished with exit code 0
```

3.2 Ring of Five Nodes

Here is a snippet of the output for five nodes in a circle that highlights its effectiveness.

```
0.002 dvr n3 -> n2 ({9: 1, 4: 0, 6: 1, 7: 0})
0.002 dvr n3 -> n4 ({9: 1, 4: 0, 6: 1, 7: 0})

== Sending packet n2 -> n3 ==
11.0 hop Link<(4) n2 -> n3>

== Sending packet n2 -> n4 ==
12.0 hop Link<(4) n2 -> n3>
12.001 hop Link<(6) n3 -> n4>

== Sending packet n2 -> n5 ==
13.0 hop Link<(3) n2 -> n1>
13.001 hop Link<(2) n1 -> n5>

== Sending packet n3 -> n1 ==
14.0 hop Link<(5) n3 -> n2>
14.001 hop Link<(3) n2 -> n1>
```

3.3 Dropping packets

However, I was not able to get packet-dropping to properly have each node update their routing table. At first, there was a routing loop. Then, the nodes were notified of when a link was down, but they did not necessarily know the whole new route to find.

```
== Sending packet n1 -> n2 ==
96.0 hop Link<(1) n1 -> n2>

== Sending packet n1 -> n3 ==
97.0 hop Link<(1) n1 -> n2>
97.001 hop Link<(4) n2 -> n3>

== Sending packet n1 -> n4 ==
98.0 hop None

== Sending packet n1 -> n5 ==
99.0 hop None

== Sending packet n2 -> n1 ==
100.0 hop Link<(3) n2 -> n1>

...

== Sending packet n5 -> n1 ==
112.0 hop None

== Sending packet n5 -> n2 ==
113.0 hop None

== Sending packet n5 -> n3 ==
114.0 hop Link<(9) n5 -> n4>
114.001 hop Link<(7) n4 -> n3>

== Sending packet n5 -> n4 ==
115.0 hop Link<(9) n5 -> n4>
```

3.4 Fifteen Nodes

Like above, I could not get link loss to work properly, but lowest cost without loss is working just fine. The cycles are also being handled without any problems. Here are some example traces for routes:

```
0.004 dvr n6 -> n1 ({1: 2, 2: 2, 3: 0, 4: 2, 5: 1, 6: 3, 7: 2, 9: 2, 10: 3, 12: 1, 13: 3, 14:
↪  3, 15: 3, 16: 2, 19: 1, 20: 1, 21: 1, 22: 0, 23: 2, 24: 1, 25: 2, 26: 1, 27: 0, 28: 1,
↪  29: 1, 30: 2, 31: 3, 32: 3, 33: 2, 34: 3})
0.004 dvr n6 -> n7 ({1: 2, 2: 2, 3: 0, 4: 2, 5: 1, 6: 3, 7: 2, 9: 2, 10: 3, 12: 1, 13: 3, 14:
↪  3, 15: 3, 16: 2, 19: 1, 20: 1, 21: 1, 22: 0, 23: 2, 24: 1, 25: 2, 26: 1, 27: 0, 28: 1,
↪  29: 1, 30: 2, 31: 3, 32: 3, 33: 2, 34: 3})
0
...
== Sending packet n1 -> n15 ==
11.0 hop Link<(1) n1 -> n2>
11.001 hop Link<(8) n2 -> n14>
11.002 hop Link<(35) n14 -> n15>
...
== Sending packet n10 -> n12 ==
22.0 hop Link<(29) n10 -> n1>
```

```
22.001 hop Link<(2) n1 -> n4>
22.002 hop Link<(13) n4 -> n5>
22.003 hop Link<(17) n5 -> n12>
```

4 Conclusion

Overall, I would rate this experiment a partial success. I've managed to learn quite a bit about the challenges of Distance-Vector routing, how some information needs to be transmitted, and some needs to be left behind, and how difficult some of these edge cases are.

I've tried implementing many solutions to solve the link-loss problem, spending many hours, but none of them have been able to be ironed out well enough to be 100% effective, and so this report is given without it.

Given the difficulty I've had, I'd imagine that this area is still a topic of heavy debate in the academic community.