**CAS CS 655 Introduction to Computer Networks**

**Programming Assignment 3: Routing (Link State vs. Distance Vector)**

Tian Zhang, Wenda Qin

1. Overview:

This report contains the following parts:

* 2 Distance Vector Routing (the Split Horizon with Poison Reverse)
* 3 Link State Routing
* 4 Comparison
* 5 Effect of topology on routing performance

We implemented both link state and distance vector, and compare them in terms of routing message overhead and convergence speed. Meanwhile, for the Link State Routing, we utilized the Dijkstra algorithm to find the update cost table forward by taken the adjacency metric graph and source node as input. For Distance Vector Routing, we tried to implement the Split Horizon with Poison Reverse for preventing reverse routes between two routers.

1.1 Code Usage:

There are three folder containing our code: DVR, LSR and InputFile. You can “cd” under each folder, and try to type the following command to run the corresponding code:

javac Project3.java

java Project3

After you compiling the code, you need to type two number Trace number and Seed number. Additionally, for “InputFile” folder, after compiling the code, the terminal will ask you to provide Trace number and Seed number and an out-sourced topology file path. For example:

Enter trace level (>= 0): [0] 2

Enter seed (>= 0): [3322] 3322

Enter a file path of adjacency matric../testcase/input.txt

And the format of topology, the file should look similar like the following example:

0 2 5 1

2 0 9999 7

5 9999 0 10

1 7 10 0

(0-9998 presents the weight of edge among directed linked two node, 9999 present the two node is not directed connected.)

2 Distance Vector:

The following topology is our template for routing for both DVR and LSR.



Figure 1 Example Topology

2.1 Design:

**void rtinit(int nodename, int[] initial\_lkcost):** To initialize each node, the initial link costs array is supplied to each node as well as the **nodename**. The node name is saved for use later on because each node needs to know what itself is. Then using the initial **lk\_costs**, we can fill the costs vector array with the supplied values. Every node that we cannot reach, or do not yet know we can reach, is initialized to INFINITY. Then the node sets its **lkcosts** to the **initial\_lkcosts** and sends the costs to all of its direct neighbors.

**void rtupdate(Packet rcvdpkt):** When a node receives and update packet, it goes through each entry in **costs[nodename][pktNode]** and updates each entry with the supplied information in the packet's array. In each spot, we put the cost to the sender + the cost the sender **advertises(costs[pktNode][i] = rcvdpkt.mincost[i]+baseDistance**). We do not update the costs if it was already INFINITY and is still INFINITY. After updating all of this information, each node will then reconsider it's **lkcost** array.

**void updateMinCosts(int sender):** To update, we call updateMinCosts(int sender) and supply the sender of the packet we have just received. Then, for each cost in **lkcost**, we look also at what the cost is to go through the sender for that node. If we can go through the sender at a lower cost, then we update lkcost to reflect that. We also set this entry in the poison array to be INFINITY (This is the poison reverse part). If the sender is not cheaper, then the entry in the poison array stays the same. If the sender did not send any cheaper costs, then we do not need to do anything. If any of our lkcost entries have changed, then we will send the poison packet to the sender. The sender will get the **poison packet** and update costs for **'nodename'** to ensure that he is never picked as a next hop for the node that 'nodename' is using sender as a next hop for.

**void sendPoison(int[] poisonArray, int dest):** To send poison packets, we call sendPoison and give the poison packet and the destination.

**void linkhandler(int linkid, int newcost)** : The Link handler works pretty similarly to update. First, the node updates the cost to the linkid in the costs table. Then, we need to check to see if we have a cheaper path to 'linkid', so we set the cost to 'linkid' to INFINITY and the nextHop node to INFINITY. Then we can look through the costs table and find the cheapest next hop node. Then, we can leverage the updateMinCosts function by setting lkcosts[linkid] to infinity and passing nextHop in as a parameter. This ensures that the lkcosts array gets updated with nextHop as the next hop. Then a poison packet will be sent to the next hop to ensure the next hop does not choose 'nodename' as a path to 'linkid'. A drawback of the implementation is that 'nodename' will send a poison packet to 'linkid' if 'linkid' is still the cheapest path. This does not really affect anything, but it is a waste of a packet.

3 Link State Routing

**rtinit(int nodename, int [] initial\_lkcost).** This routine will be called once at the beginning of the emulation. It should initialize *nodename* and the node’s distance to its immediate neighbors *lkcost.* You will also need to update the adjacency metric *graph* and forwarding table *costs*. These data structures are explained in detail later. In Figure 1, all links are bi-directional and the costs in both directions are identical. After initializing the data structures needed by your node, the node should then send its link state (cost) information to all the nodes in the network using only its directly connected neighbors. This link-cost information is sent to neighboring nodes in a routing (link state) *packet* by calling the routine NetworkSimulator.tolayer2(*packet*), as described below. The format of the routing packet is also described below.

**rtupdate(Packet rcvdpkt).** This routine will be called when a node receives a routing (link state) packet that was sent to it by one of its directly connected neighbors. rtupdate() is the ``heart'' of the link state algorithm. The values it receives in a routing packet from some other node *i* contain *i*'s current link costs to its neighbors. rtupdate() uses these received values to update its adjacency metric *graph*. It then runs Dijkstra’s ***shortest path algorithm*** on the *graph* to update its forwarding table *costs*. Since the received packet is a broadcast (link state) packet, the node forwards the received link-state packet to other neighboring nodes in the network. Note that you need to make sure that flooding stops when all the nodes in the network receives the link-state packet.

**int[][] dijkstra(int graph[][], int src).** The routine takes the adjacency metric *graph* and source node *src* as input, and returns a 2D array *output*. *output* is a 4-by-2 array, where destination *i* can be reached from source node *src* with a minimum cost of *output*[*i*][0] via node *output*[*i*][1]. Note that *output*[*i*][1] is the immediate node before the destination node *i* on the shortest path from *src* to *i*. Note that *output* is different from forwarding table *costs*, and you will need information in *output* to update the forwarding table *costs*.

**int[][] findNextHop(int[][] costs,int src).** After the **Dijkstra algorithm,** we **use** findNextHop to find the next hop and return the corresponding cost value and the nextHop.

4 Comparison

We set the trace to 2, and set five seed values for experiments(12,128,512,1234,3322) and try to predefined five topologies which is store at the folder of “/testcase”.

Table 1 LSR vs DVR without Link Change

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Seed=12 | Seed=128 | Seed=512 | Seed=1234 | Seed=3322 |
| Convergence Time for LSR | 11.93 | 9.91 | 10.76 | 8.78 | 8.14 |
| Packet Number for LSR | 30 | 30 | 30 | 30 | 30 |
| Convergence Time for DVR | 5.13 | 3.64 | 3.34 | 2.60 | 2.23 |
| Packet Number for DVR | 10 | 10 | 10 | 10 | 10 |

Table 2 LSR vs DVR with Link Change

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Seed=12 | Seed=128 | Seed=512 | Seed=1234 | Seed=3322 |
| Convergence Time for LSR | 20001.58 | 20001.15 | 20002.56 | 20001.83 | 20001.00 |
| Packet Number for LSR | 54 | 54 | 54 | 54 | 54 |
| Convergence Time for DVR | 20000 | 20000 | 20000 | 20000 | 20000 |
| Packet Number for DVR | 10 | 10 | 10 | 10 | 10 |

Table 3 Convergence Time for 5 predefined topologies(DVR)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Seed=12 | Seed=128 | Seed=512 | Seed=1234 | Seed=3322 |
| Input | 3.15 | 4.46 | 3.61 | 3.88 | 3.07 |
| Input2 | 5.50 | 2.53 | 3.90 | 4.08 | 4.02 |
| Input3 | 2.73 | 3.07 | 3.47 | 2.75 | 2.28 |
| Input4 | 2.68 | 3.02 | 2.09 | 1.97 | 1.93 |
| Input5 | 4.52 | 2.92 | 4.97 | 4.80 | 3.43 |

Table 4 Convergence Time for 5 predefined topologies(LSR)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Seed=12 | Seed=128 | Seed=512 | Seed=1234 | Seed=3322 |
| Input | 8.25 | 9.26 | 8.71 | 8.28 | 8.17 |
| Input2 | 9.53 | 9.33 | 9.19 | 10.08 | 9.13 |
| Input3 | 10.01 | 9.17 | 8.27 | 7.15 | 7.24 |
| Input4 | 8.68 | 8.42 | 7.02 | 7.92 | 6.62 |
| Input5 | 7.64 | 7.52 | 9.57 | 9.62 | 8.15 |

As we can see from our textbook:

• Message complexity. We have seen that LS requires each node to know the cost of each link in the network. This requires O(|N| |E|) messages to be sent. Also, whenever a link cost changes, the new link cost must be sent to all nodes. The DV algorithm requires message exchanges between directly connected neighbors at each iteration. We have seen that the time needed for the algorithm to converge can depend on many factors. When link costs change, the DV algorithm will propagate the results of the changed link cost only if the new link cost results in a changed least-cost path for one of the nodes attached to that link.

• Speed of convergence. We have seen that our implementation of LS is an O(|N|2) algorithm requiring O(|N| |E|)) messages. The DV algorithm can converge slowly and can have routing loops while the algorithm is converging. DV also suffers from the count-to-infinity problem.

Our experiment also find the same tendency.

6 Sample Trace

6.1 DVR:

Network Simulator v1.0

Enter trace level (>= 0): [0] 2

Enter seed (>= 0): [3322] 3322

Initializing Node 0

===============================

Sending information to Node 1 from Node 0

Link Costs: 0 1 1 9999

===============================

Sending information to Node 2 from Node 0

Link Costs: 0 1 1 9999

===============================

Initializing Node 1

===============================

Sending information to Node 0 from Node 1

Link Costs: 1 0 10 7

===============================

Sending information to Node 2 from Node 1

Link Costs: 1 0 10 7

===============================

Sending information to Node 3 from Node 1

Link Costs: 1 0 10 7

===============================

Initializing Node 2

===============================

Sending information to Node 0 from Node 2

Link Costs: 1 10 0 2

===============================

Sending information to Node 1 from Node 2

Link Costs: 1 10 0 2

===============================

Sending information to Node 3 from Node 2

Link Costs: 1 10 0 2

===============================

Initializing Node 3

===============================

Sending information to Node 1 from Node 3

Link Costs: 9999 7 2 0

===============================

Sending information to Node 2 from Node 3

Link Costs: 9999 7 2 0

===============================

MAIN: rcv event, t=0.092, at 2 src: 0, dest: 2, contents: 0 1 1 9999

Updating Node 2 From sender 0

Sent Link Costs: 0 1 1 9999

===============================

via

D2 | 0 1 3

----|-----------------

0| 1 2 9999

dest 1| 9999 9999 9999

3| 9999 9999 9999

MAIN: rcv event, t=0.104, at 0 src: 1, dest: 0, contents: 1 0 10 7

Updating Node 0 From sender 1

Sent Link Costs: 1 0 10 7

===============================

via

D0 | 1 2

----|-----------------

1| 1 11

dest 2| 9999 9999

3| 9999 9999

MAIN: rcv event, t=0.115, at 1 src: 0, dest: 1, contents: 0 1 1 9999

Updating Node 1 From sender 0

Sent Link Costs: 0 1 1 9999

===============================

via

D1 | 0 2 3

----|-----------------

0| 1 2 9999

dest 2| 9999 9999 9999

3| 9999 9999 9999

MAIN: rcv event, t=0.447, at 2 src: 1, dest: 2, contents: 1 0 10 7

Updating Node 2 From sender 1

Sent Link Costs: 1 0 10 7

===============================

via

D2 | 0 1 3

----|-----------------

0| 1 2 9999

dest 1| 11 10 17

3| 9999 9999 9999

MAIN: rcv event, t=0.884, at 3 src: 1, dest: 3, contents: 1 0 10 7

Updating Node 3 From sender 1

Sent Link Costs: 1 0 10 7

===============================

via

D3 | 1 2

----|-----------------

0| 9999 9999

dest 1| 7 17

2| 9999 9999

MAIN: rcv event, t=1.375, at 1 src: 2, dest: 1, contents: 1 10 0 2

Updating Node 1 From sender 2

Sent Link Costs: 1 10 0 2

===============================

via

D1 | 0 2 3

----|-----------------

0| 1 2 9999

dest 2| 11 10 12

3| 9999 9999 9999

MAIN: rcv event, t=1.410, at 1 src: 3, dest: 1, contents: 9999 7 2 0

Updating Node 1 From sender 3

Sent Link Costs: 9999 7 2 0

===============================

via

D1 | 0 2 3

----|-----------------

0| 1 2 9999

dest 2| 11 10 12

3| 9999 9 7

MAIN: rcv event, t=1.669, at 3 src: 2, dest: 3, contents: 1 10 0 2

Updating Node 3 From sender 2

Sent Link Costs: 1 10 0 2

===============================

via

D3 | 1 2

----|-----------------

0| 9999 9999

dest 1| 7 17

2| 12 2

MAIN: rcv event, t=2.032, at 0 src: 2, dest: 0, contents: 1 10 0 2

Updating Node 0 From sender 2

Sent Link Costs: 1 10 0 2

===============================

via

D0 | 1 2

----|-----------------

1| 1 11

dest 2| 11 1

3| 9999 9999

MAIN: rcv event, t=2.306, at 2 src: 3, dest: 2, contents: 9999 7 2 0

Updating Node 2 From sender 3

Sent Link Costs: 9999 7 2 0

===============================

via

D2 | 0 1 3

----|-----------------

0| 1 2 9999

dest 1| 11 10 17

3| 9999 9 2

MAIN: rcv event, t=10000.000, at -1

Cost from 0 to 1 changed to 20

via

D0 | 1 2

----|-----------------

1| 20 11

dest 2| 11 1

3| 9999 9999

Cost from 1 to 0 changed to 20

via

D1 | 0 2 3

----|-----------------

0| 20 2 9999

dest 2| 11 10 12

3| 9999 9 7

MAIN: rcv event, t=20000.000, at -1

Cost from 0 to 1 changed to 1

via

D0 | 1 2

----|-----------------

1| 1 11

dest 2| 11 1

3| 9999 9999

Cost from 1 to 0 changed to 1

via

D1 | 0 2 3

----|-----------------

0| 1 2 9999

dest 2| 11 10 12

3| 9999 9 7

Simulator terminated at t=20000.000000, no packets in medium

Simulator terminated at time 20000.0

The quantity of packet sent to layer2 10

6.2 LSR

Network Simulator v1.0

Enter trace level (>= 0): [0] 2

Enter seed (>= 0): [3322] 3322

MAIN: rcv event, t=0.092, at 2 src: 0, dest: 2, contents: 0 1 1 9999

D2 | cost next-hop

----|-----------------------

0| 1 2

dest 1| 2 0

2| 0 2

3| 2 2

MAIN: rcv event, t=0.104, at 0 src: 1, dest: 0, contents: 1 0 10 7

D0 | cost next-hop

----|-----------------------

0| 0 0

dest 1| 1 0

2| 1 0

3| 8 1

MAIN: rcv event, t=0.115, at 1 src: 0, dest: 1, contents: 0 1 1 9999

D1 | cost next-hop

----|-----------------------

0| 1 1

dest 1| 0 1

2| 2 0

3| 7 1

MAIN: rcv event, t=0.447, at 2 src: 1, dest: 2, contents: 1 0 10 7

D2 | cost next-hop

----|-----------------------

0| 1 2

dest 1| 2 0

2| 0 2

3| 2 2

MAIN: rcv event, t=0.884, at 3 src: 1, dest: 3, contents: 1 0 10 7

D3 | cost next-hop

----|-----------------------

0| 8 1

dest 1| 7 3

2| 2 3

3| 0 3

MAIN: rcv event, t=1.375, at 1 src: 2, dest: 1, contents: 1 10 0 2

D1 | cost next-hop

----|-----------------------

0| 1 1

dest 1| 0 1

2| 2 0

3| 4 0

MAIN: rcv event, t=1.410, at 1 src: 3, dest: 1, contents: 9999 7 2 0

D1 | cost next-hop

----|-----------------------

0| 1 1

dest 1| 0 1

2| 2 0

3| 4 0

MAIN: rcv event, t=1.515, at 1 src: 2, dest: 1, contents: 0 1 1 9999

D1 | cost next-hop

----|-----------------------

0| 1 1

dest 1| 0 1

2| 2 0

3| 4 0

MAIN: rcv event, t=1.669, at 3 src: 2, dest: 3, contents: 1 10 0 2

D3 | cost next-hop

----|-----------------------

0| 3 2

dest 1| 7 3

2| 2 3

3| 0 3

MAIN: rcv event, t=2.032, at 0 src: 2, dest: 0, contents: 1 10 0 2

D0 | cost next-hop

----|-----------------------

0| 0 0

dest 1| 1 0

2| 1 0

3| 3 2

MAIN: rcv event, t=2.112, at 1 src: 3, dest: 1, contents: 1 10 0 2

D1 | cost next-hop

----|-----------------------

0| 1 1

dest 1| 0 1

2| 2 0

3| 4 0

MAIN: rcv event, t=2.306, at 2 src: 3, dest: 2, contents: 9999 7 2 0

D2 | cost next-hop

----|-----------------------

0| 1 2

dest 1| 2 0

2| 0 2

3| 2 2

MAIN: rcv event, t=2.329, at 0 src: 2, dest: 0, contents: 1 0 10 7

D0 | cost next-hop

----|-----------------------

0| 0 0

dest 1| 1 0

2| 1 0

3| 3 2

MAIN: rcv event, t=2.949, at 3 src: 2, dest: 3, contents: 0 1 1 9999

D3 | cost next-hop

----|-----------------------

0| 3 2

dest 1| 4 2

2| 2 3

3| 0 3

MAIN: rcv event, t=2.988, at 2 src: 0, dest: 2, contents: 1 0 10 7

D2 | cost next-hop

----|-----------------------

0| 1 2

dest 1| 2 0

2| 0 2

3| 2 2

MAIN: rcv event, t=3.322, at 1 src: 0, dest: 1, contents: 1 10 0 2

D1 | cost next-hop

----|-----------------------

0| 1 1

dest 1| 0 1

2| 2 0

3| 4 0

MAIN: rcv event, t=3.603, at 0 src: 1, dest: 0, contents: 1 10 0 2

D0 | cost next-hop

----|-----------------------

0| 0 0

dest 1| 1 0

2| 1 0

3| 3 2

MAIN: rcv event, t=3.867, at 1 src: 2, dest: 1, contents: 9999 7 2 0

D1 | cost next-hop

----|-----------------------

0| 1 1

dest 1| 0 1

2| 2 0

3| 4 0

MAIN: rcv event, t=4.384, at 2 src: 1, dest: 2, contents: 0 1 1 9999

D2 | cost next-hop

----|-----------------------

0| 1 2

dest 1| 2 0

2| 0 2

3| 2 2

MAIN: rcv event, t=4.422, at 1 src: 3, dest: 1, contents: 0 1 1 9999

D1 | cost next-hop

----|-----------------------

0| 1 1

dest 1| 0 1

2| 2 0

3| 4 0

MAIN: rcv event, t=4.575, at 3 src: 1, dest: 3, contents: 0 1 1 9999

D3 | cost next-hop

----|-----------------------

0| 3 2

dest 1| 4 2

2| 2 3

3| 0 3

MAIN: rcv event, t=4.723, at 3 src: 2, dest: 3, contents: 1 0 10 7

D3 | cost next-hop

----|-----------------------

0| 3 2

dest 1| 4 2

2| 2 3

3| 0 3

MAIN: rcv event, t=4.768, at 0 src: 1, dest: 0, contents: 9999 7 2 0

D0 | cost next-hop

----|-----------------------

0| 0 0

dest 1| 1 0

2| 1 0

3| 3 2

MAIN: rcv event, t=5.623, at 0 src: 2, dest: 0, contents: 9999 7 2 0

D0 | cost next-hop

----|-----------------------

0| 0 0

dest 1| 1 0

2| 1 0

3| 3 2

MAIN: rcv event, t=6.160, at 2 src: 3, dest: 2, contents: 1 0 10 7

D2 | cost next-hop

----|-----------------------

0| 1 2

dest 1| 2 0

2| 0 2

3| 2 2

MAIN: rcv event, t=6.243, at 1 src: 0, dest: 1, contents: 9999 7 2 0

D1 | cost next-hop

----|-----------------------

0| 1 1

dest 1| 0 1

2| 2 0

3| 4 0

MAIN: rcv event, t=6.588, at 3 src: 1, dest: 3, contents: 1 10 0 2

D3 | cost next-hop

----|-----------------------

0| 3 2

dest 1| 4 2

2| 2 3

3| 0 3

MAIN: rcv event, t=7.152, at 2 src: 1, dest: 2, contents: 9999 7 2 0

D2 | cost next-hop

----|-----------------------

0| 1 2

dest 1| 2 0

2| 0 2

3| 2 2

MAIN: rcv event, t=7.217, at 2 src: 3, dest: 2, contents: 0 1 1 9999

D2 | cost next-hop

----|-----------------------

0| 1 2

dest 1| 2 0

2| 0 2

3| 2 2

MAIN: rcv event, t=8.139, at 2 src: 0, dest: 2, contents: 9999 7 2 0

D2 | cost next-hop

----|-----------------------

0| 1 2

dest 1| 2 0

2| 0 2

3| 2 2

Simulator terminated at t=8.139409, no packets in medium

Simulator terminated at time 8.139408551286357

The quantity of packet sent to layer2 30