**Implementation of Unicast Routing Protocols**

Kuang-Huei Lee (klee156)

Meng Zhang (mzhang61)

**Introduction**

This report introduces the implementations of distance vector and link state routing protocols in structural programming language C++. Briefly, the non-protocol manager program provides the information of topology and message to its connection nodes respectively. The nodes which running the distance vector or link state program receive information from their manager or neighbors, set up the routing table according to its protocol and send messages. The detailed implementation steps lists below.

**Source file**

There are three source files list below: manager.cpp, distvec.cpp and linkstate.cpp. In each time processing, only one of distance vector or link state protocol is used to connect manager. In another word, the network is “pure” rather than mix of two protocols. Following is the details of working processing of each source code. Source code is rich with comments to help readers understand code blocks.

* **manager.cpp**

At very beginning, manager initializes its topology map that every node is disconnected. We use -1 to represent this situation. Manager will read the topology file to set up the topology map topology [MAX\_NODE\_NUM] [MAX\_NODE\_NUM]. Also, manager read and store message file msgs [MAX\_MSG\_NUM].

When a new node joins the network, manager will record and assign it the lowest unclaimed virtual node ID. Meanwhile, manager gets the IP address and store it in the respective ipstr[i]. Manager uses TCP to communicate with nodes to achieve a reliable information package. Therefore, the receiver could receive package in right sequence and avoid lost packages. According to the topology map, manager can find all the node’s neighbors (exsting in the map) and cost. Consequently, it builds the sending string (my\_to\_string(i+1) my\_to\_string(topology[tid][i]) ipstr[i]) ) and send the socket to the node and respective neighbors.

Manager has a listening port to get the updating topology information from stdin. Once there is a topology n1 n2 cost input, manager should rebuild its map and send this update to n1 and n2.

Manager has another listening port to the connecting nodes for the convergence notice. When all the nodes have sent the converged to manager, manager thinks all the table of nodes in the network are converged and then read msgs[MAX\_MSG\_NUM] line by line (msgs[i].src, msgs[i].dst, msgs[i].text) and store in the sending string. Manager send this socket to the source node.

After the data is sent, nothing else needs to happen until there is an update topology entered on stdin. The above procedures are repeated that sending updated information and message to node and then waiting for another new topology input.

* **distvec.cpp**

Once the node enters the network, it gets its ID and neighbor’s information from the manager. According to it, node rebuilds the routing table and sends this table to its neighbors using UDP. Meanwhile, the node could listen the routing table from its neighbors as well as the updated topology information from manager. Generally, distvec.cpp is using the Bellman-Ford Algorithm to compute its routing table. That is to compare the cost in the routing table and the new cost (whether from neighbors or manager). Choose the smaller one and update the next hop accordingly.

There is a “timer” in distvec.cpp. Once there is an updating information comes, reset the timer. The converged notice will be sent to manager if timeout.

If the node receives message from manager, firstly it will decide whether there is a route to the destination. If not, it will figure out what the next hop is and send the socket of message to it. Also, the node can listen the message information from its neighbor, receiving the message and so-far path. If the node is not the final destination, it will forward to the next hop.

* **linkstate.cpp**

Link state node gets the same information as the distance vector from the manager and does the similar procedures with distance vector. However there are some differences. Link state uses Dijkstra’s Algorithm and heap to compute its routing table, which core idea is if the path cost to node w is bigger that the sum of the cost to the neighbor and the neighbor’s path cost, then we update the smaller cost and corresponding next hop. Meanwhile, the node also stores the string track and print sit out.

**Processing graph**

The processing graph of the manager and the distvec node is attached after to help understand the implementation. Link state shares the similar idea except the routing protocol.

**Contributions of each team member:**

1. Kuang-Huei Lee:

He built the network connection in manager.cpp, distvec.cpp and linkstate.cpp;

He also worked on link state protocol including Dijkstra's algorithm with heap.

He worked on distance vector protocol in distvec.cpp

He worked on updating routing table, convergence, and sending messages code in distvec.cpp and linkstate.cpp

2. Meng Zhang:

She worked on parsing topology and messages in manager.cpp

She also worked on distance vector protocol in distvec.cpp;

She worked on this report and readme.

Manager

IP table: <ID><IP>

Assign lowest unclaimed ID

Connection, get IP address

Distvec

Initial routing table:

<dist><nexthop><cost>

if dist is itself, cost=0;

else, cost=1000 (considered as disconnected

additional topology information entered on stdin

N

Keep waiting for all converged

Routing table

Routing table

Notice of convergence

All converged?

Y

Keep waiting update and checking timer

N

Timer>1?

N

Reset timer

Y

Update?

Neighbors

print out: disconnection

N

Y

Cost=-1?

Send to relevant node

Topology[ ][ ]

Update topology map

Set up topology map

Topology.txt

Has route?

Send topology info to node and its neighbor: (neighborID1+cost1+IP1)+ (neighborID2+cost2+IP2)+…

Search topology[ID][], if cost >0, means there exist a link to its neighbor.

Send msg to source node

Message []:

<src><dist><test>

Y

Update routing table:

<dist><nexthop><cost>

*Bellman-Ford Algorithm*

when dist=neighborID:

if cost > newcost, update to new cost and nexthop;

else, remain the routing table

print out: new connection

Topology.txt

Neighbor

Neighbor

Keep waiting for information

Y

Message infomation

According to routing table and dist, figure out which is the next hop

N

Message infomation

Is the destination?

print out: src, dist, hops, text

Y

N

print out: no route