**Name: Gautam Othayoth Ganapathiyadan**

**Course: CSE589**

**Project: Routing Protocols**

**Overview:**

This Project implements Distance Vector Protocol by running on top of the various CSE Servers viz. Timberlake, Underground, Euston, Embankment and Highgate.

It can also take commands from the user while listening to messages from other servers.

**Design**: The code will read the topology file that is passed in as input and then find the server id of the server on which the program is running. It then identifies the number of server and the number of neighbors it has in the network. The cost is also stored for each neighbor. Once this is obtained, the Distance Vector Protocol is calculated by using the Bellman Ford Algorithm and this table is maintained in each server. Also if a server stops sending requests for 3 successive intervals, the cost gets updated to infinity.

**Build:**

The topology file is read using fopen and the required details are stored in the respective variables (line number: 108, comment: reading topology file starts). numserv stores the number of servers in the networks. numedge stores the number of neighbors.

**SERVER START UP –**

Command to be used for startup - ./gautamot\_proj2 -t <topology file name> -i <interval>

Eg : ./gautamot\_proj2 -t input.txt -i 2

**DATA TYPES/CLASSES USED –**

**File used** – gautamot\_proj2.cpp

The topology class object tpf stores the serverids and their respective ips and port numbers of all servers.(ln : 55)

The topology class object tpf stores the servers and their respective ips and port numbers of the server running the program.

The retdet class object rtdd functions like a dictionary where it returns the IP address and port when server id is passed as index. (ln : 78)

The costs class object ct functions like a dictionary where it returns the cost when server id is passed as index.(ln : 92)

The costnd class object tpf stores the serverids, neighborids and their respective costs.(ln : 85)

**ROUTING TABLE STRUCTURE** –

Between Line Numbers 766 and 805 in the program.

Search for comment: ROUTING TABLE STRUCTURE starts

|  |  |  |
| --- | --- | --- |
| Data to be sent | Variable which stores the data in the program | Bytes |
| Number of updates | num\_updates | 2 |
| Port Number of the server sending the packet | crs.ipd | 2 |
| IP of the server sending the packet | crs.port | 4 |
| IP of the neighbor | tpf[server id of the neighbor].ipd | 4 |
| Port of the neighbor | tpf[server id of the neighbor].port | 2 |
| Serverid of the neighbor | Cost[iterator].neighid | 2 |
| Cost of the link between server sending the packet and the neighbor | Ct[server id of the neighbor].cta | 2 |

Note that an entry for the cost to the current server is present as 0.

**BELLMAN FORD ALGORITHM**

Between lines 308 and 354 in the program.

Search for comment: BELLMAN FORD ALGORITHM starts here

Bellman Ford algorithm is implemented by checking f[l-1](u) < f[l](v) +w(u,v) for all u, v belonging to edges. Here f[l](v) is the shortest path from node v to a particular destination. w(u,v) is the initial weight of the edge/link between u and v.

**COMMANDS THAT CAN BE USED**

**update 1 2 inf**

The link between the servers with IDs 1 and 2 is assigned to infinity.

**update 1 2 8**

Change the cost of the link to 8.

**step**

Send routing update to neighbors right away. Note that except this, routing updates only happen

periodically.

**packets**

Display the number of distance vector packets this server has received since the last instance when this

information was requested.

**disable server-id**

Disable the link to given server. Here you need to check if the given server is its neighbor.

**crash**

Emulate a server crash. Close all connections on all links. The neighboring servers must handle this

close correctly and set the link cost to infinity.

**display**

Display the current routing table. The display should be formatted as a sequence of lines, with each line

indicating: <destination-server-ID> <next-hop-server-ID> <cost-of-path>

**TIMEOUT LOGIC**

If no update is received from a neighbor for 3 successive intervals, the cost to the server is set to infinity.

However, to account for packets losses that could happen if there are a lot of neighbors, a buffer time is also included along with the 3 interval time for recording a timeout and assigning the cost to be infinity.

**IMPORTANT NOTE**: As there could be packet losses in UDP and if the interval passed in the command line is large, it will take time for the server to form the correct Routing Table. Hence, it is advised to wait for some time before invoking “display” function to view the correct Routing Table after initial startup and update commands.