# MNC Project 2 Report

## Distance Vector Routing

Elroy Alva

50168107

### Introduction

DVRP is a routing protocol that sends distance vector updates to all its neighbors. It finds the shortest path to a node using the packets received from its neighbors. It does not need to communicate with all routers to find the shortest route. It uses the Bellman Ford algorithm to do this.

For this project I have implemented the DVR protocol that operates over the UDP protocol. It read a topology file that sets it initial routes and distance matrix. It then proceeds to send routing packets to the other servers. If a link fails, it can set its distance to infinity and stop sending it packets.

For doing this project I have referred to the Beej Sockets guide and implemented many of the lessons there.

The program can be started using two parameters. Namely the topology file name and the timeout interval in seconds.

*./server –t topologyFile.txt –i 10*

I have implemented all the functions in the project description.

For storing '**infinity'** as a distance in my matrix I have used **'-1'**

**Update:**

To perform the update command, we set our own distance vector to the specified value, then we a send an update packet over UDP to the client who's link needs to be updated and it will do the needful.

**Display:**

The display command will look into the cost table structure and display the shortest distance to all nodes on the network as well as the next hop to get to it.

**Crash:**

When a crash command is entered, it will terminate the node process. This will result in it not sending anymore packets to its neighbors. The neighbors after not receiving an update for 3 intervals will set the cost to infinity and look for a shorter path to it. This will suffer from the count to infinity problem

**Disable:**

The disable command will disable a link to a certain neighbor. The command will not work if a node tries to disable itself or a non-neighbor. It will also send a command to neighbor to disable its link.

**Step:**

The step command will force a node to send a DV packet to all its neighbors immediately.

**Packets:**

This command will display a packet count of the total received DV packets. After which it will set the counter again to 0.

I used an integer matrix to store the initial distances. I only used the indices from one onwards so as to make it easier to reference the correct node.

To store *costTable* I used a 2-dimensional cost matrix as well as custom struct to store the next hop. When the costs were updated I changed the *costMatrix* as well as the struct.

Whenever a timeout occurs we send DV packets to our neighbors so that they update their tables correspondingly. Whenever we receive a DV packet we update our cost tables and calculate our shortest paths.

**Update Cost table is implemented on line 604:**

This method will calculate the shortest path using the bellman ford algorithm

**Display Cost table is implemented on line 444:**

This method will look in the cost matrix as well as a structure that stores the next hop.

**Send Packets to neighbor is implanted on line 566:**

This method will read the DV and send the data to its neighbors. If a server times out, we will stop sending them any packets. The packets are constructed in a char buffer and not serialized. I did this because I kept running into errors while serializing the update data. I have sent the messages in the very same structure but as a string separated by a whitespace.

**Disable link is implanted on line 557:**

This method will take a parameter as an integer and set cost to that server id as infinity. It will then send an update packet (updt) to the corresponding recipient to set cost to infinity. This neighbor cannot be updated after this command. Will throw an error if we try to update itself or a non-neighbor.

Every time an interval passes we print a

‘.....~~~.....’

This was done to better understand how packets are sent and received.

This program was tested on 5 VM’s on Ubuntu as well as the cse servers.

Most of the code is commented and self-explanatory.

I have used the Beej guide to sockets extensively and implemented this using those guidelines and methods.

**Conclusion**

Thus we have successfully implemented DVR protocol to run on different servers and handle crashes and route updates automatically.