

ITCS 6166: PROGRAMMING PROJECT 2B

Implementing Distance Vector Routing Protocol

Project Team:

Name: Anagha Sarmalkar

800#: 801077504

1. INTRODUCTION

Distance vector routing protocol is used to determine the shortest route on a given network topology for sending data packets based on distance. The routers exchange information with one another using routing tables.

2. PROGRAM STRUCTURE

This particular implementation uses UDP sockets to send routing information across the network. Router.py file is executed on every router and the correct UDP port number and .dat file are passed as command line arguments. The program includes the following methods.

a. **print_output():**

- This function is used to print the outputs in the desired formats whenever the router sends out routing tables to its neighbours.

b. **initialize_routing():**

- This function is used to initialize the routing table(vector_table) with the link_cost table (neighbours)

c. **sender(sock, nports):**

- This function takes the socket object and the port number of the router neighbour and initializes it using **initialize_routing()**. It then sends routing information(vector_table) to its neighbours after every 15 seconds. The vector_table is serialized using pickle and sent.

d. **receiver(sock, port):**

- This function takes the socket object and the port number to start receiving routing information from the neighbours. It first initializes the

routing table using **initialize_routing()**. This serialized data is unserialized and sent to **vector_routing(data, port)** to calculate the new routing table.

e. get_port_details():

- This function is used to load the **port_info.pickle** file and get the port details of the routers which are currently on the network.

f. read_dat_file():

- This function is used to read the .dat input file and extract and return the neighbours and their link costs.

g. get_router_details():

- This function is used to check the validity of the command line arguments provided and adds them to the **port_info.pickle** file to keep a track of the number of routers in the network along with the information of their port numbers.
- It checks if the .dat file is present in the current working directory.
- It checks if the port number provided is already in use by some other router and tells the user accordingly.
- It checks if the same .dat file is provided again for a different port.

h. vector_routing(data, port):

- This function takes the routing table received from the neighbours (data) and compares it with its link cost table to assign infinite distance values to the non neighbour nodes.
- It reads the input .dat file for this router again to ensure no link change has occurred. If it has taken place, the routing table is reset with its initial link_cost values and sent to the neighbours. The new routing table is thus calculated again.
- If the link cost has not changed, the function updates its own routing table (vector_table) according to the Bellman Ford equation for finding the least cost path.
- The router achieving the minimum is the next hop in the shortest path which is used in the routing table.

i. main():

- This is the main function which takes the UDP port number and the .dat file provided as command line arguments. The router will be identified with

the name of the .dat file provided to it. **get_router_details()** is called to check the validity of the arguments.

- Upon ensuring correctness of the parameters, the port information of this router is added to a **port_info.pickle** file which keeps track of the number of routers in the network. This file is updated whenever a new router is added in the network. This file needs to be deleted every time a new network topology is being tested.
- The UDP socket for the router then binds on the localhost with its UDP port number and starts a thread for receiving the routing tables from its neighbours by calling **receiver(sock, port)**.
- The while loop keeps on checking if the neighbours of the router have started by checking the port_info.pickle file. It starts the sending thread for the neighbour only when it has started by calling **sender(sock, nports)**. The while loop breaks once all the neighbours have started
- Another thread for printing the outputs is started by calling **print_output()**.

3. EXECUTION STEPS

Please ensure that the working directory **does not include port_info.pickle** before execution of any router. This configuration file is created dynamically to keep a track of the number of routers connected on the network and their corresponding port numbers. This file needs to be deleted before testing every new network configuration.

- a. Run the Router.py on the required number of terminals as per the network topology and UDP port number and respective .dat file as command line arguments. Make sure that new UDP port numbers and correct .dat files are provided for every terminal.

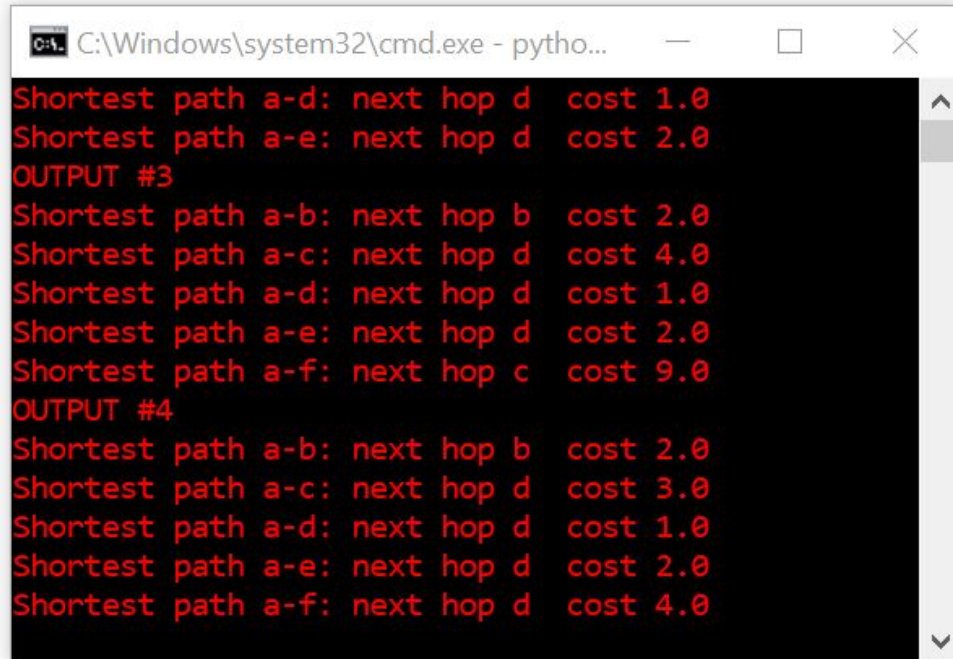
Eg: python Router.py 5000 a.dat

- b. Alternatively, you can open(double-click) the 6 .bat files which are batch scripts to open the command line with predefined parameters.

4. OUTPUT SCREENSHOTS

The following are the screenshots for the network configuration in the assignment.

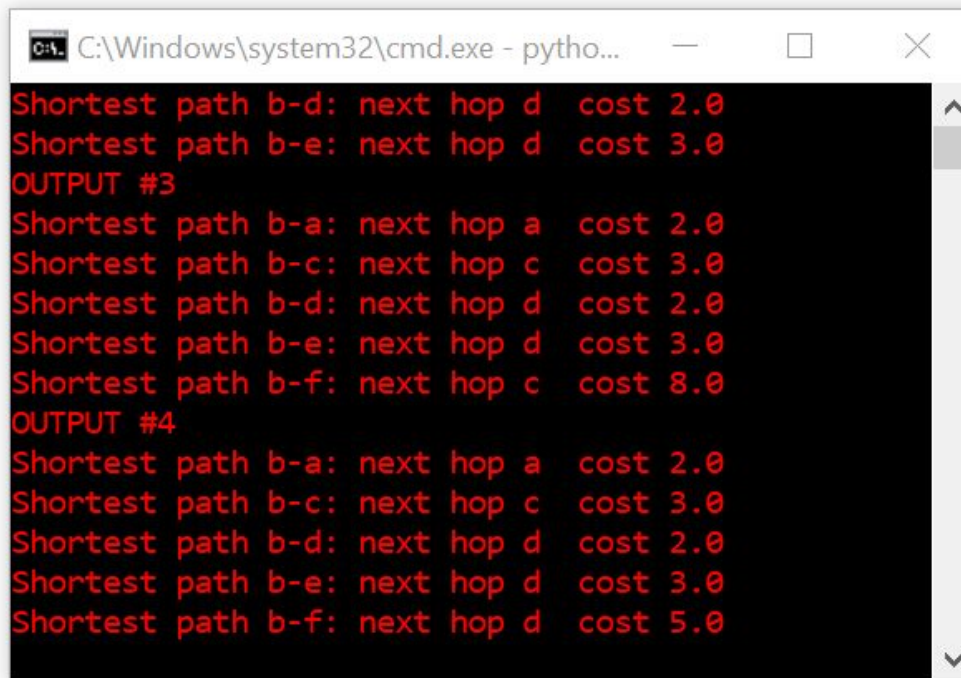
a. Router A



```
C:\Windows\system32\cmd.exe - pytho...

Shortest path a-d: next hop d cost 1.0
Shortest path a-e: next hop d cost 2.0
OUTPUT #3
Shortest path a-b: next hop b cost 2.0
Shortest path a-c: next hop d cost 4.0
Shortest path a-d: next hop d cost 1.0
Shortest path a-e: next hop d cost 2.0
Shortest path a-f: next hop c cost 9.0
OUTPUT #4
Shortest path a-b: next hop b cost 2.0
Shortest path a-c: next hop d cost 3.0
Shortest path a-d: next hop d cost 1.0
Shortest path a-e: next hop d cost 2.0
Shortest path a-f: next hop d cost 4.0
```

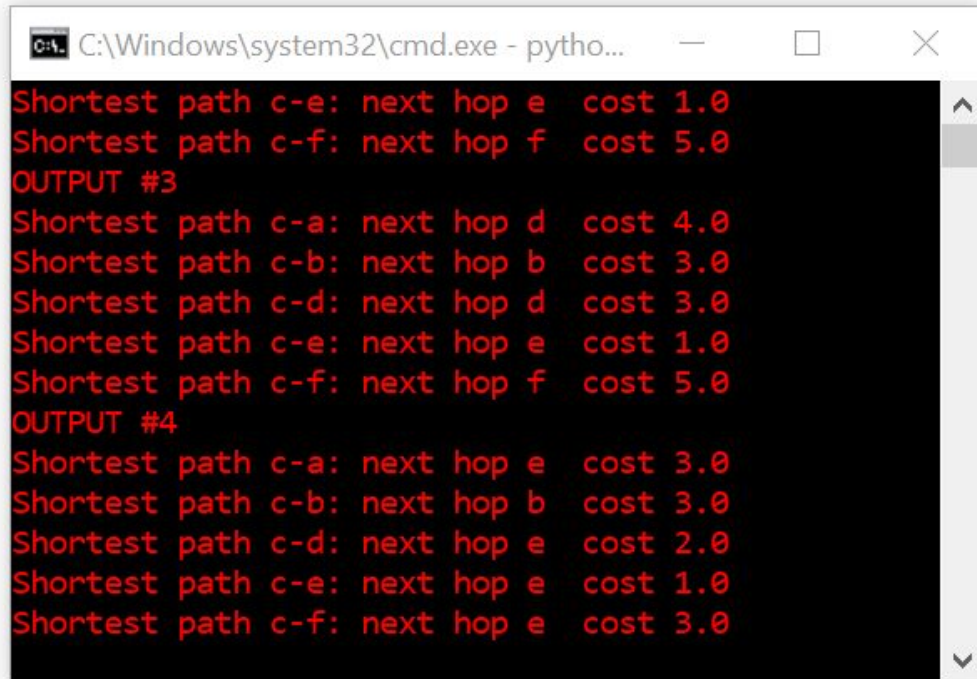
b. Router B



```
C:\Windows\system32\cmd.exe - pytho...

Shortest path b-d: next hop d cost 2.0
Shortest path b-e: next hop d cost 3.0
OUTPUT #3
Shortest path b-a: next hop a cost 2.0
Shortest path b-c: next hop c cost 3.0
Shortest path b-d: next hop d cost 2.0
Shortest path b-e: next hop d cost 3.0
Shortest path b-f: next hop c cost 8.0
OUTPUT #4
Shortest path b-a: next hop a cost 2.0
Shortest path b-c: next hop c cost 3.0
Shortest path b-d: next hop d cost 2.0
Shortest path b-e: next hop d cost 3.0
Shortest path b-f: next hop d cost 5.0
```

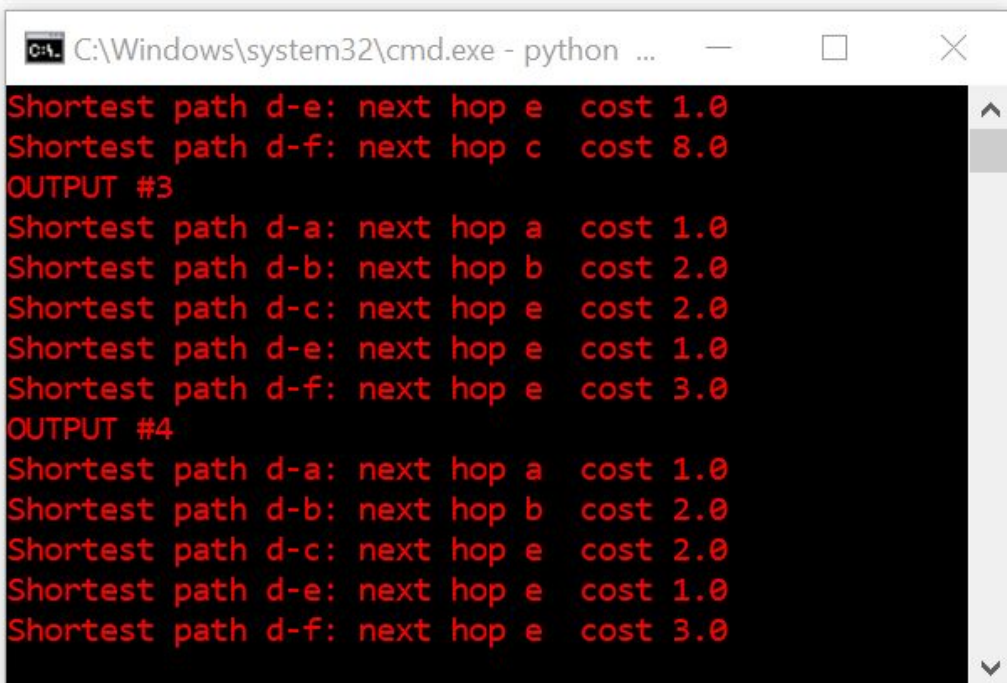
c. Router C



```
C:\Windows\system32\cmd.exe - pytho...

Shortest path c-e: next hop e cost 1.0
Shortest path c-f: next hop f cost 5.0
OUTPUT #3
Shortest path c-a: next hop d cost 4.0
Shortest path c-b: next hop b cost 3.0
Shortest path c-d: next hop d cost 3.0
Shortest path c-e: next hop e cost 1.0
Shortest path c-f: next hop f cost 5.0
OUTPUT #4
Shortest path c-a: next hop e cost 3.0
Shortest path c-b: next hop b cost 3.0
Shortest path c-d: next hop e cost 2.0
Shortest path c-e: next hop e cost 1.0
Shortest path c-f: next hop e cost 3.0
```

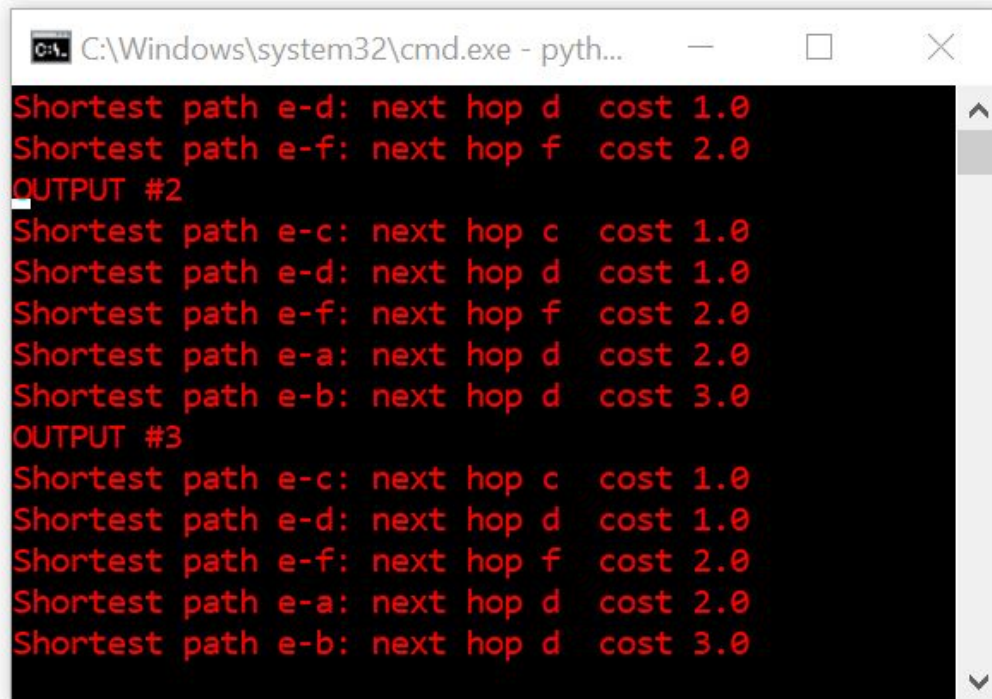
d. Router D



```
C:\Windows\system32\cmd.exe - python ...

Shortest path d-e: next hop e cost 1.0
Shortest path d-f: next hop c cost 8.0
OUTPUT #3
Shortest path d-a: next hop a cost 1.0
Shortest path d-b: next hop b cost 2.0
Shortest path d-c: next hop e cost 2.0
Shortest path d-e: next hop e cost 1.0
Shortest path d-f: next hop e cost 3.0
OUTPUT #4
Shortest path d-a: next hop a cost 1.0
Shortest path d-b: next hop b cost 2.0
Shortest path d-c: next hop e cost 2.0
Shortest path d-e: next hop e cost 1.0
Shortest path d-f: next hop e cost 3.0
```

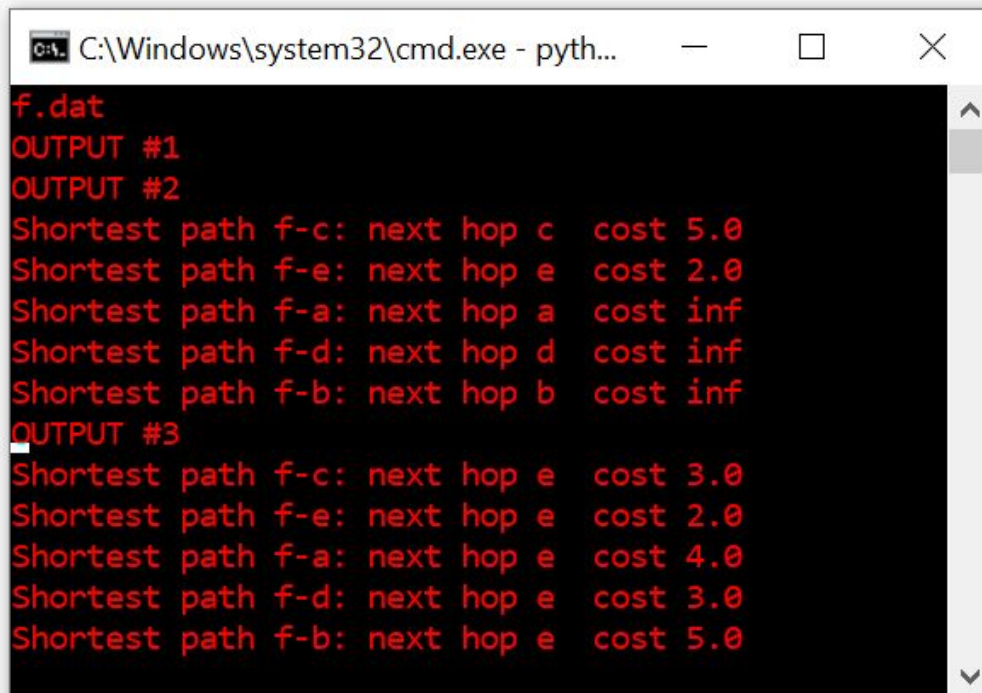
e. Router E



```
C:\Windows\system32\cmd.exe - pyth...

Shortest path e-d: next hop d cost 1.0
Shortest path e-f: next hop f cost 2.0
OUTPUT #2
Shortest path e-c: next hop c cost 1.0
Shortest path e-d: next hop d cost 1.0
Shortest path e-f: next hop f cost 2.0
Shortest path e-a: next hop d cost 2.0
Shortest path e-b: next hop d cost 3.0
OUTPUT #3
Shortest path e-c: next hop c cost 1.0
Shortest path e-d: next hop d cost 1.0
Shortest path e-f: next hop f cost 2.0
Shortest path e-a: next hop d cost 2.0
Shortest path e-b: next hop d cost 3.0
```

f. Router F



```
C:\Windows\system32\cmd.exe - pyth...

f.dat
OUTPUT #1
OUTPUT #2
Shortest path f-c: next hop c cost 5.0
Shortest path f-e: next hop e cost 2.0
Shortest path f-a: next hop a cost inf
Shortest path f-d: next hop d cost inf
Shortest path f-b: next hop b cost inf
OUTPUT #3
Shortest path f-c: next hop e cost 3.0
Shortest path f-e: next hop e cost 2.0
Shortest path f-a: next hop e cost 4.0
Shortest path f-d: next hop e cost 3.0
Shortest path f-b: next hop e cost 5.0
```

5. REFERENCES

- a. <https://en.wikipedia.org/>
- b. <https://wiki.python.org/moin/UdpCommunication>
- c. <https://stackoverflow.com/>
- d. <https://pymotw.com/2/socket/udp.html>
- e. <https://www.studytonight.com/network-programming-in-python/working-with-udp-sockets>

6. CHALLENGES:

Tried to implement link cost changes by resetting the vector tables of the neighbours, however could not succeed. I also tried implementing Poison reverse but could not get the outputs correct.