COMP429 Programming Assignment 2

Distance Vector Routing Protocols

1. Problem Statement

In this assignment you will implement a simplified version of the *Distance Vector Routing Protocol*. The protocol will be run on top of four servers/laptops (behaving as routers) using TCP. Each server runs on a machine at a pre-defined port number. The servers should be able to output their forwarding tables along with the cost and should be robust to link changes. A server should send out routing packets only in the following two conditions: **a) periodic update** and **b) the user uses command asking for one**. This is a little different from the original algorithm which immediately sends out update routing information when routing table changes.

2. Getting Started

A Distance Vector Routing Algorithm (Textbook Chapter 5).

3. Protocol Specification

The various components of the protocol are explained step by step. Please strictly adhere to the specifications.

3.1 Topology Establishment

In this programming assignment, you will use **four** servers/computers/laptops. **The four servers are required to form a network topology as shown in Fig. 1.** Each server is supplied with a topology file at startup that it uses to build its initial routing table. The topology file is local and contains the link cost to the neighbors. For all other servers in the network, the initial cost would be infinity. Each server can only read the topology file for itself. The entries of a topology file are listed below:

- ***<num-servers>***
- ***<num-neighbors>***
- ***<server-ID> <server-IP> <server-port>***
- ***<server-ID1> <server-ID2> <cost>***

num-servers: total number of servers.
server-ID, server-ID1, server-ID2**:** a unique identifier for a server, which is assigned by you.
cost**:** cost of a given link between a pair of servers. Assume that cost is an integer value.
Given the network topology shown in Figure 1, the table below shows you a topology file for server 1.

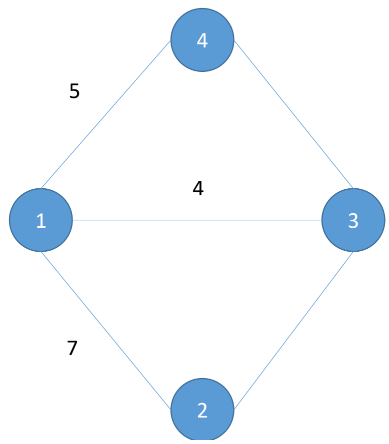

Figure 1: Example topology

Line number	Line entry	Comments
1	4	number of servers
2	3	number of edges or neighbors
3	1 128.205.36.8 4091	server-id 1 and corresponding IP, port pair
4	2 128.205.35.24 4094	server-id 2 and corresponding IP, port pair
5	3 128.205.36.24 4096	server-id 3 and corresponding IP, port pair
6	4 128.205.36.4 7091	server-id 4 and corresponding IP, port pair
8	1 2 7	server-id and neighbor id and cost
9	1 3 4	server-id and neighbor id and cost
10	1 4 5	server-id and neighbor and cost

Your topology files should only contain the Line entry part (2nd column in the pink color). In each line, every two elements (e.g., server-id and corresponding IP, corresponding IP and port number) should be separated with a **space**. For cost values, **each topology file should only contain the cost values of the host server's neighbors** (<u>The host server here is the one which will read this topology file</u>). Note that the IPs of servers may change when you are running the program in a wireless network environment. So, we need to use *ifconfig* or *ipconfig* to obtain the IP first and then set up the topology file before the demo.

IMPORTANT: In this environment, costs are bi-directional i.e. the cost of a link from A-B is the same for B-A. Whenever a new server is added to the network, it will read its topology file to determine who are its neighbors. Routing updates are exchanged periodically between neighboring servers. When this newly added server sends routing messages to its neighbors, they will add an entry in their routing tables corresponding to it. Servers can also be removed from a network. When a server has been removed from a network, it will no longer send distance vector updates to its neighbors.

When a server no longer receives distance vector updates from its neighbor for three consecutive update intervals, it assumes that the neighbor no longer exists in the network and makes the appropriate changes to its routing table (link cost to this neighbor will now be set to infinity but not remove it from the table). This information is propagated to other servers in the network with the exchange of routing updates. Please note that although a server might be specified as a neighbor with a valid link cost in the topology file, the absence of three consecutive routing updates from this server will imply that it is no longer present in the network.

3.2 Routing Update

Routing updates are exchanged periodically between neighboring servers based on a time interval specified at the startup. In addition to exchanging distance vector updates, servers must also be able to respond to user-specified events. There are 3 possible events in this system. They can be grouped into three classes: topology changes, queries and exchange commands: (1) Topology changes refer to an updating of link status (update). (2) Queries include the ability to ask a server for its current routing table (display), and to ask a server for the number of distance vectors it has received (**packets**). In the case of the packets command, the value is reset to **zero** by a server after it satisfies the query. (3) Exchange commands can cause a server to send distance vectors to its neighbors immediately.

3.3 Message Format (recommended but not required)

Routing updates are sent using the General Message format. All routing updates are TCP unreliable messages. The message format for the data part is:

0 1 2 3 (10 bits)

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 (bit)

<table>
<tr><td>Number of update fields</td><td>Server port</td></tr>
<tr><td colspan="2">Server IP</td></tr>
<tr><td colspan="2">Server IP address 1</td></tr>
<tr><td>Server port 1</td><td>0x0</td></tr>
<tr><td>Server ID 1</td><td>Cost 1</td></tr>
<tr><td colspan="2">Server IP address 2</td></tr>
<tr><td>Server port 2</td><td>0x0</td></tr>
<tr><td>Server ID 2</td><td>Cost 2</td></tr>
<tr><td colspan="2">….</td></tr>
</table>

- **Number of update fields:** (2 bytes): Indicate the number of entries that follow.
- **Server port:** (2 bytes) port of the server sending this packet.
- **Server IP:** (4 bytes) IP of the server sending this packet.
- **Server IP address n:** (4 bytes) IP of the n-th server in its routing table.
- **Server port n:** (2 bytes) port of the n-th server in its routing table.

- **Server ID n:** (2 bytes) server id of the n-th server on the network.
- **Cost n:** cost of the **path** from the server sending the update to the n-th server whose ID is given in the packet.

Note:

First, the servers listed in the packet can be any order i.e., 5, 3, 2, 1, 4. Second, the packet needs to include an entry to reach itself with cost 0 i.e. server 1 needs to have an entry of cost 0 to reach server 1.

4. Server Commands/Input Format

The server must support the following command at startup:

- **server -t <*topology-file-name*> -i <*routing-update-interval*>**

  topology-file-name: The topology file contains the initial topology configuration for the server, e.g., timberlake_init.txt. Please adhere to the format described in 3.1 for your topology files.

  routing-update-interval: It specifies the time interval between routing updates in seconds.

  port and server-id: They are written in the topology file. The server should find its port and server-id in the topology file without changing the entry format or adding any new entries.

The following commands can be specified at any point during the run of the server:

- **update <*server-ID1*> <*server-ID2*> <*Link Cost*>**

  server-ID1, server-ID2: The link for which the cost is being updated.

  Link Cost: It specifies the new link cost between the source and the destination server. Note that this command will be issued to **both** *server-ID1* and *server-ID2* and involve them to update the cost and no other server.

  For example:

  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 <u>the number of distance vector</u> (packets) this server has received since the last invocation of this information.

- **display**

  Display the current routing table as explained in our lecture, including current and neighboring nodes' distance vector.

- **disable <*server-ID*>**

  Disable the link to a given server. Doing this "closes" the connection to a given server with *server-ID*. Here you need to check if the given server is its neighbor.

- **crash**

  "Close" all connections. This is to simulate server crashes. Close all connections on all links. The neighboring servers must handle this close correctly and set the link cost to infinity.

5. Server Responses/Output Format

The following are a list of possible responses a user can receive from a server:

- On successful execution of an update, step, packets, display or disable command, the server must display the following message:

  <*command-string*> SUCCESS

  where *command-string* is the command executed. Additional output as desired (e.g., for display, packets, etc. commands) is specified in the previous section.
- Upon encountering an error during execution of one of these commands, the server must display the following response:

  <*command-string*> <*error message*>

  where *error message* is a brief description of the error encountered.
- On successfully receiving a route update message from neighbors, the server must display the following response:

  RECEIVED A MESSAGE FROM SERVER <*server-ID*>

  where the *server-ID* is the id of the server which sent a route update message to the local server.

6. Submission and Grading

6.1 What to Submit

- A README file that documents each member's contribution in details and explains how to install any prerequisites, build your program, and run your application.

- A zipped file – Name it as < your id_name>.zip, which includes all source files (.h and .c or .cc or .java files), including Makefile. Note: name your main program as distance_vector_routing.c or distance_vector_routing.cc or distance_vector_routing.java.

6.2 How to submit

Use CANVAS to submit the zipped file.

Each group is required to make a project demo video and upload it to YouTube. The length of this video should be about 10 minutes (12 minutes maximum). After uploading the demo video, please submit the video link URL in Canvas. This demo video shall show:

1. Distance Vector Algorithm part of your source code with brief explanation. Don't explain the socket program. Try to limit this explanation to 2 minutes.
2. If appliable, live compilation (if appliable) of your source code
3. Output of all commands you implemented with brief explanation. To show your algorithm works as expected, you must first show the expected routing tables and then demonstrate that your program outputs the same results. That is, you need to propose the hypothesis first and then prove it! You will receive no more than 11 points if your demo doesn't include this step.
4. You may do this demo in one laptop by opening four command windows representing four servers
5. Each group will need to play this video with me and be ready to answer any questions I may raise

6.3 Grading Criteria

- *Each group will be scheduled a demo. All the members should attend the demo, and describe their coding contribution in this project.* ***You will receive no points if you don't show up for demo.***
- Correctness of output and exceptional handling.
- Organization and documentation of your code.
- **Project is graded based on your own individual contribution.**

6.4 Important Key Points:

- There is just one program. DON'T submit separate programs for client and server.
- Error Handling is very important – Appropriate messages should be displayed when something goes bad.
- DON'T ASSUME. If you have any doubts in project description, please come to my office hour.
- Submission deadline is hard. No extension.
- Please do not submit any binaries or object files or any test files.