Project 3 s17

Programming Assignment 3:
Simulating the distance vector algorithm
Due May 7th 23:55

This project should be done in pairs.

Overview

In this laboratory programming assignment, you will simulate a network of routers implementing the distance-vector routing algorithm.

There are two main goals for this project:

1. To gain a deeper understanding of the distance-vector routing algorithm and examine how it works in a distributed setting.
2. Gain experience in programming with datagram sockets (UDP)
3. Gain experience in programming with threads

Requirements:

Each router will act similarly to what a real router in using the RIP routing protocol. That is:

1. A router should update its distance vector and forwarding table every time it receives a new distance vector from his neighbors or the weight to its neighbor changes.
2. A router will send an update every *n* seconds or every time its own distance vector changes.
3. If a router does not receive an update after 3 time periods of *n* seconds, it should drop the neighbor.
4. A router will forward a message with a specific destination according to its forwarding table.
5. A router will accept changes in its weights to its neighbors and advertise it to them
6. A router will support poisoned reverse as an option.

Implementation:

You will write your implementation using the following instructions:

1. The code must be written in java
2. Each router you will implement will be a process with it's own IP address and port on which it is listening on (you can test your code using localhost with different port numbers similar to project 1).
3. Each router should accept 2 parameters as follows:

   java router [-reverse] neighbors.txt

   a. Reverse is an option to activate poisoned reverse

   b. a filename as a parameter. The first row should specify the current node's ip and port number, and then each row should have the neighbor's ip and port number with the weight of the link. You can assume that the files are correct and represent an undirected graph (the weight from A to B is the same as the weight from B to A). An example can be shown here:

```
127.0.0.1 9876
127.0.0.1 9877 1
127.0.0.1 9878 1
127.0.0.1 9879 1
```

4. Each process will need multiple threads. Specifically:
    a. One thread for sending a DV update (that should happen every *n* seconds). You can use the Timer class to implement that.
    b. One thread to accept the incoming messages from the neighbors and perform the necessary operations.
    c. One thread to read commands from the command line and perform the necessary operations.
5. When reading from the command line, the following commands should be allowed:
    a. **PRINT** - print the current node's distance vector, and the distance vectors received from the neighbors.
    b. **MSG <dst-ip> <dst-port> <msg>** - send message msg to a destination with the specified address.
    c. **CHANGE <dst-ip> <dst-port> <new-weight>** - change the weight between the current node and the specified node to new-weight and update the specified node about the change.
6. When receiving a message from the neighbor, the following types of messages should be allowed:
    a. A new distance vector
    b. A new weight update
    c. A message being sent
7. When a message is sent, each forwarding router should append its ip address and port number to the end of the message (thus the final message should look like: **msg ip-port ip-port ip-port …**).
8. If a message arrives without any entry in the forwarding table it should simply be dropped.
9. The router should print out all the actions it performs for easy debugging. The messages should be of the following format:
    a. When a distance vector is received from a neighbor print:
    new dv received from ipAddress:port with the following distances:
    ipAddress:port distance
    ipAddress:port distance
    ipAddress:port distance
    b. When a new weight to a neighbor is received print:
    new weight to neighbor IpAddress:port of newWeight
    c. When a neighbor is dropped because of a timeout:
    neighbor ipAddr:port dropped
    d. When your own distance vector is updated:
    new dv calculated:
    ipAddress:port distance nextHopIpAddress:nextHopPort
    ipAddress:port distance nextHopIpAddress:nextHopPort
    ipAddress:port distance nextHopIpAddress:nextHopPort
    e. When a DV update is sent out as a timed event:
    Update sent to all neighbors at time t(in seconds)
    ipAddress:port distance
    ipAddress:port distance
    ipAddress:port distance
    f. When a message is forwarded to the next router:
    Message msg from ipAddr:port to ipAddr:port forwarded to ip:port
    msg(the actual message with the concatenated ip addresses and port numbers)

Resources:

1. Use the book to understand the distance vector algorithm.

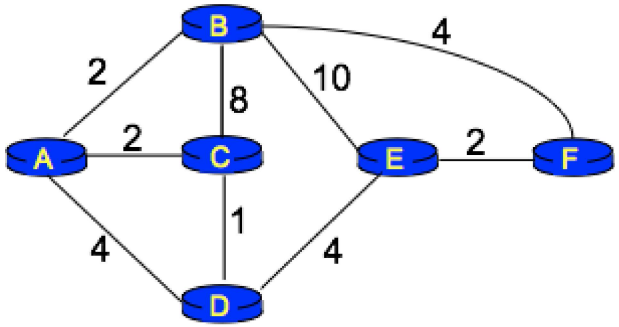
2. To understand how to use sockets for a UDP server/client see section 2.8 of the older version of the book here:
http://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_6/prevEd/2-7_2-8.pdf
3. Here is an example of the Timer class. Although in this example they use an inner class, you can use a regular class as well.
http://bioportal.weizmann.ac.il/course/prog2/tutorial/essential/threads/timer.html
4. There are many tutorials online for thread usage. Here is a simple example that I think shows the basics well:
http://www.careerbless.com/java/Threads/ImplementingRunnable.php

Submission:

Besides your code (which should be well commented), please submit the following report:

1. A short introduction to what the program is trying to achieve.
2. A description of the distance vector algorithm
3. A description of your code design and choice of data structures.
4. Correctness Results - In this section you should present the results for a few simple scenarios. For each one manually calculate the results and then show that one node on the network produces the desired output:
   a. The following simple network at node A (each link has weight 1):

   b. The same network as A, but after convergence then change the weight between B and C to 10… (no poisoned reverse)
   c. The same situation as B with poisoned reverse.
   d. The following more complicated network at E:



   e. The same as in d, but remove node D after convergence.
5. Conclusion - Mention any observations you have made about the distance vector protocol.
6. References - If necessary.
7. A section on what each member of the team did as part of the project.