Programming assignment 2: Network Layer

CSEE4119 Computer Networks: Tuesday, April 17

Academic Honesty Policy

You are permitted and encouraged to help each other through Piazza's web board, in order to discuss and understand concepts learned in class or go deeper in a particular subject.
HOWEVER, you may *NOT* share source code or hardcopies of source code, or answers to written assignments. Refrain from activities or the sharing materials that could cause your source code to APPEAR TO BE similar to another student's enrolled in this or previous years. Cheating will be dealt with severely. Cheaters will be punished. Source code and written answers should be yours and yours only. Do not cheat.

1 Introduction

In this assignment, you will work on the network layer. The goal is to implement a distance vector routing protocol.

This time, you won't start your program from scratch, but add your code in a class of a program we coded. The goal of this program is to simulate a network with a given topology. **This assignment is shorter, but you will be less guided**. This time, all your assignment is **in one single project** and you don't create the main function as we did it for you.

2 Due Dates and Program Submission

This programming assignment is due on **April 17th**. **No lateness is accepted**. This time, you will all submit using CourseWorks.

Before submitting, make sure your program compiles and runs properly on a CS or EE machine, because you will be graded on those machine, and code that doesn't compile will obviously be considered as incorrect and will obtain the grade of 0.

3 Get started

Eclipse:

We strongly advise you to use Eclispe (you can download eclipse with java here: http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/indigosr1).

To start the assignment, create a new project from existing source, and chose the Folder called PA2 from the .zip we gave you. You can also create a new Java project and copy the files in src.

The program:

All the code you need to write will be in Node.java (or classes you created). You are not allowed to change any code contained in the other files that are provided (we will check that you did not before grading).

Inputs and outputs:

The program takes 2 input files, you have examples in the .zip you downloaded (Event Sand Graph). Those files are given in argument, with the name of the output file (everything is commented in the main). They behave as follows:

- Graph describes the topology:
    - First line in the number of nodes
    - then you have one line per node, with numbers separated by a space for: node_ID neighbor1_ID costToReachneighbor_1 neighbor2_ID costToReachneighbor_2 ...
    - NodeIDs have to be 1, 2, 3, 4...
- Event describes the simulation. First line is the timeslots («turns» for the duration of your simulation), second is the number of events, and then you have one line per event (each characteristics are separated by a space) :
    - the turn at which the event is taken into account (it obviously has to be less than the total number of turns).
    - the type of event: R for remove a link between two nodes ; M to modify the cost between two nodes.
    - ID of the source node for the link that is changed/suppressed.
    - ID of the destination node for the link that is changed/suppressed.

- for M events only: the new cost.

The program outputs a file with the topology to see if the algorithm works well. You have one line per link in your network, with this syntax: id_node id_other_node cost.

Your work:

You only have to edit Node.java (you can create new classes, but not change ours except Node.java), but we invite you to look at the rest too.

You are free to use whatever message format you want for nodes to communicate, and whatever data structure you want to be stored in the nodes (e.g., to store a distance vector).

According to the Graph and Event files, the program will initiate your Nodes. At each turn, the function turn() is called for every Node, to get the messages it wants to send to its neighbors.
Then for each message received by a Node, its ReceiveMessage(Message) function is called.

Remember that a Node can only send messages to its neighbors!

Don't forget to implement the add/removeNeighbor and changeCost functions, or you will never find the right result!

4 First step: Bellman-Ford Algorithm (70%)

In the first step, you have to implement a basic Distance Vector protocol, with Bellman-Ford Algorithm. It will be seen in class, and it is described in the book (Section 4.5.2 p. 318-389). Edit the Node class to get the proper behavior.

As a reminder, Bellman-Ford algorithm has the following behavior:
- Wait until the cost of one of your link change or until you receive a message
- Compute the distance vector knowing that
- Notify all your neighbors of the changes

You can use the data structure you want for the Distance Vector, and you have to figure out the messages format.

5 Second step: The Count to Infinity Problem (30%)

Now you need to improve your algorithm to prevent some cases of count to infinity. To do that, you will add a poison reverse feature to your Distance Vector Protocol.

Conclusion

Feel free to ask all the questions you have :-). And good luck!