COMP 416 – Computer Networks

Project #3

Due: December 17th, 2018, 11.59pm (Late submissions will not be accepted.)

Submit your project deliverables to the folder:

F:\COURSES\UGRADS\COMP416\HOMEWORK\PROJECT3\YourGroupNumberFolder

Note: This is a group-oriented project, you are encouraged to form groups of 3 students, and we suggest you a fair task distribution at the end of this project description. Working individually is allowed but not recommended.

RouteSim: Distance Vector Routing Simulator

Introduction and Motivation:

This project work is about the **network layer** of the Internet protocol stack. It involves the **Distance Vector Routing algorithm**. The objectives are to develop a discrete event routing simulator, and examine the network layer issues, the principles behind network layer services, and routing (path selection). Through this project, you are asked to develop a network simulator named RouteSim as specified in this document, perform simulations with RouteSim, and report the performance results.

Recall that routing algorithms can be classified according to the type of information kept by each router as either *Global* where all nodes have the complete topology information (Link State Algorithm) or *Decentralized* where nodes only know physically connected neighbors and link costs of neighbors, and they exchange information with neighbors (Distance Vector Algorithm).

Distance Vector Routing is based on a distributed approach that allows nodes to discover the destinations reachable in the network as well as the least cost path to reach each of these destinations. The least cost path is computed based on the link costs, and each node maintains its own distance vector and each of the neighbors' distance vector.

RouteSim Overview:

In this project, you are asked to develop and evaluate RouteSim, a Distance Vector Routing Simulator, that executes the Distance Vector Routing Algorithm on a given input topology, prints the intermediate steps and the final distance tables of the nodes. RouteSim would be based on Discrete Event Simulation (DES). DES is the process of codifying the behavior of a complex system as an ordered sequence of well-defined events. In this context, an event comprises a specific change in the system's state at a specific point in time. Note that your simulator should be offline i.e., a single process that is executed on a single computer and simulates an entire topology. There is no need for your simulator to be connected to any other process either on the same computer or remotely.

RouteSim Code Structure:

You are expected to implement RouteSim using an object-oriented language based on the following structure:

Input:

RouteSim simulator should read an input topology from an input text file. The input file should be located in the root directory of your project with the name of input.txt. The input.txt file has as many as lines as the number of nodes in the input topology. Each row is in the form of:

<Node Number>,<(Neighbor Number, Link to Neighbor's Cost)>,......, <(Neighbor Number, Link to Neighbor's Cost)>

For example, the input.txt content for the topology sample of Figure 1 is as follows.

```
0,(1,5),(2,3)
1,(0,5),(2,9),(4,1)
2,(0,3),(1,9),(3,3)
3,(2,3),(4,7)
4,(1,1),(3,7)
```

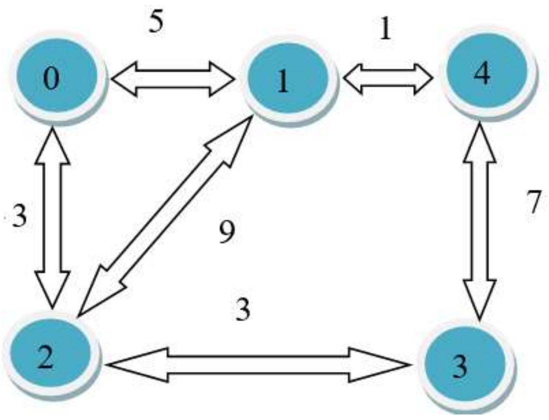

Figure 1- Sample Topology for input.txt

Node Class:

On reading the input.txt, your simulator should load the input topology by representing each node as a Node class object. You are expected to implement the Node class with the following member variables:

- An integer for the **nodeID** that represents the node's number in topology.
- A **linkCost** hash table with neighbor id as the key and the link cost to the neighbor as the value.
- A **distanceTable** with the implementation of your choice. As studied in class, the distance table inside each node is the principal data structure used by the distance vector algorithm. You may find it convenient to declare the distance table as a two-dimensional array of integers allocated dynamically, where entry [i,j] in the distance table is node's currently computed cost to node i via direct neighbor j. If this node is not directly connected to j, you can ignore this entry. We will use the convention that the integer value 999 as "infinity".

Your Node class should also implement the following functions:

- **public Node(int nodeID, HashTable<> linkCost) :** This is the constructor of the Node class, that is called upon the creation of a new Node object. This function should be invoked upon loading data from the input.txt file to the RouteSim. You need to create a new Node object to represent each node inside your topology. After initializing, the distance table and any other data structures of your Node object should be initialized upon your need. For example, the distance table should be initialized here based on your choice of data structure.

- **public void receiveUpdate(Message m):** This function is called when the node receives a message that was sent to it by one of its directly connected neighbors. The parameter m is the message object that was received. This function is the heart of the distance vector algorithm. The values it receives a message from some other node i contain i's current shortest path costs to all other network nodes. This function uses these received values to update its own distance table (as specified by the distance vector algorithm). If its minimum cost to another node changes as a result of the update, the node informs its directly connected neighbors of this change in minimum cost by sending them a message. Recall that in the distance vector algorithm, only directly connected nodes exchange messages. Thus, for example in Figure-1, nodes 0 and 1 will communicate with each other, but nodes 0 and 3 will not communicate with each other directly. This function should print out the sender and receiver of the received message, last status of the distance table of the node after each call as well as a message indicating whether the distance table has been updated or not.

- **public boolean sendUpdate():** This function is called when the node wants to notify its directly connected neighbors. This routine should generate and send the minimum cost paths between this node and all network nodes, to all directly connected neighbors. This minimum cost information is sent to neighboring nodes in a **message** object as specified in the followings. This function should decide whether or not to send the updates to the neighbors based on the convergence situation of the node. The node should decide locally without knowing any global knowledge or state from other nodes. If an update should happen, the node sends updates by calling its directly connected neighbors receiveUpdate function and returns true. Otherwise, it should return false. Upon sending an update to its neighbors, this function should print out the id of the node, and the content of the message.

- **public HashTable<> getForwardingTable():** This function is called to construct the forwarding table from the current state of the distance table of the node. The forwarding table is a HashTable with keys as String values denoting the id of the node in the topology, and values as the id of the forwarding node. For example the key j and value k in the forwarding table of node i means that node i should route the messages with destination node j by forwarding them to the neighbor node k.

Message Class:

The message class represents a message that contains the distance vector estimates from one node to another node. The implementation details of this class are left to you, however, it should contain the sender's and receiver's ID.

RouteSim Class:

The RouteSim class is the main class of the simulator, which reads the input file, creates an in memory representation of Nodes and stores the input topology inside. You should implement the distance vector algorithm inside this class. The algorithm should be executed in a round-based manner (using a for loop for example). In each round, each node's sendUpdate function is invoked where it decides on whether or not to send updates to its directly connected neighbors. The simulation terminates when none of the nodes have any new update to send i.e., all of them return false as the result of sendUpdate invocation. The RouteSim should print the number of the round, and indicate when the Distance Vector algorithm converges.

Requirements:

Static Link Cost Scenario

In this scenario, your RouteSim should perform the algorithm in a round-based manner, and at each round show the distance table of each node of the input topology. RouteSim should detect the convergence of the nodes and decide on termination. Upon termination, it should print out the distance table and forwarding table of each node. Simulate the topology of Figure-1 with your implemented RouteSim and report the number of rounds it takes for the topology to converge and terminate in your report. Also, you need to do the iterations by your own on the topology of Figure-1, type in the steps in your report, and compare with the results you collected from the simulator. Clarify any difference between your solution and the one you obtained from the simulator, in case there was a difference. For better insurance on the correctness of your implementation, you are recommended to simulate with other topologies of your choice and compare against theoretical results.

Dynamic Link Cost Scenario

In the dynamic link cost scenario, your input.txt contains the value of "x" instead of an integer cost value for at least one link. An "x" character as the cost for a link in the input file denotes that the associated link does not have a fixed-value cost. Rather, the link's cost is dynamic. You need to initialize a dynamic link with a initially randomly chosen cost between 1 to 10. You need to associate a random boolean value generator with each link to decide on the time that the cost changes. At the beginning of each round, for each dynamic link cost, you should generate a random boolean value using random generator, and if the randomly generated value is TRUE, you should assign a random value between 1 to 10 as the link cost. Otherwise, you do not need to change the cost value. Starting from a random link of your choice in Figure-1's topology, you should make the links dynamic one-by-one, and do the simulation. To make a link dynamic, simply replace its cost in the input.txt file with an "x" character. Figure-1 has 6 links and hence, you need to try this simulation 6 times; 1st time with only one dynamic link, 2nd time with two dynamic links, and the ith time with dynamic links. For each simulation, you should observe the overall number of rounds it takes for your simulator to terminate. You need to plot a graph in your report with x-axis showing the number of dynamic links (i.e., number of links with "x" cost in input.txt) in your simulation and y-axis as the number of rounds it takes for the simulator to converge and terminate.

Bonus part:

As the optional bonus part, you are asked to implement a graphical user interface (GUI) that visualize the topology graph. For example, on reading the input.txt provided, it should draw the topology graph of Figure-1 with nodes as circles, and links as connecting lines between them. The cost of each link should be displayed for each link. Your GUI should also reflect the link cost changes in real time. It should color the links selected for routing when the simulation terminates.

Report:

- The **report** is an **important part of your project** presentation, which should be submitted as both a .pdf and Word file.
- In addition to the requested graph and results, your report should show the architecture of your simulator, the class interactions, and how a node decides on termination.
- You are expected to provide the diagram of all interactions between the classes within your RouteSim. You need to provide a textual description of the diagram in the report.
- Your report also should contain a setup section to clearly direct the user on running your code i.e., how to run your code? which IDE did you use to implement your code? In the other words, the setup section should provide a step-by-step instruction for anyone who wants to execute your implemented RouteSim.
- **Report acts as a proof of work for you to assert your contributions to this project.** Everyone who reads your report should be able to reproduce the parts we asked to document without hard effort and any other external resource.
- If you need to put the code in your report, segment it as small as possible (i.e., just the parts you need to explain) and clarify each segment before heading to the next segment. For codes, you should be taking **screenshot** instead of copy/pasting the direct code. Strictly **avoid replicating the code** as whole in the report, or leaving code **unexplained**.

Project deliverables:

You should submit your source code and project report (in a single .rar or .zip file). The name of the file should be <last name of group members>. Your submission should include:

- Source Code: A .zip or .rar file that contains your implementation.
- Project report as specified in the Report section.

Demonstration:

The group selection sheet as well as demo sessions would be announced by the TA. All groups should register their group members within the specified time. Even if you are doing the project individually, you are required to register yourself as a group within the specified period that will be announced, and select a demo time slot accordingly. Attending the demo session is required for your project to be graded. All group members must attend the demo session. The on-time attendance of all group members is considered as a grading criteria.

Important Notes:

- **Please read this project document carefully BEFORE starting your design and implementation.**
- In case you use some code parts from the Internet, you must provide the references in your report (such as web links) and explicitly define the parts that you used. In general, you should not use any external third-party library for this project.
- You should **not** share your code or ideas with other project groups. Please be aware of the KU Statement on Academic Honesty.
- In your report, you need to dedicate a section describing the task division among the group members. There you are supposed to clearly state the contributions of each group member to the project.
- Your entire code should be commented in a clear manner. Everyone who reads your code should understand it without hard effort.
- Your report should present your designs aspects and act as a reference for your implementations before and after the demo session. **Please take the report part seriously and write your report as accurate and complete as possible.**

Suggested Task Distribution:

We recommend you to work in a 3-student group, and suggest the following task distribution:

Student 1: Reading and processing the inputs, implementing Node and Message classes.
Student 2: The implementation of Distance Vector Routing algorithm within RouteSim class.
Student 3: Test of RouteSim, Experiments, graph (see Requirements section) , and report.

Good Luck!