COMP/ELEC 416 : Computer Networks

Project #3 : Part 2

Due: May 27, 2020, 11.59pm (Late submissions will not be accepted.)

This is a group (3-student) project. We suggest a fair task distribution among group members at the end of this project description.

ModivSim: Modified Distance Vector Routing Simulator

Introduction and Motivation:

This project work is about the **routing algorithms** as part of the **network layer** of the Internet protocol stack. It involves design and implementation of a modified version of the **Distance Vector Routing algorithm**. The objectives of the project 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 **ModivSim** as specified in this document, perform simulations with ModivSim, and report the performance results. Moreover, you are asked to use the simulator to test the flow of network packets and observe how the path selection is performed.

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. For this project, the following modifications in the original distance vector routing algorithm are required.

1. The neighbors would share the distance vector after every p seconds (where p is an input), instead of sharing it when there is an update.
2. The neighbors would also share the bandwidth of each link it has. More details are discussed later in this document.
3. In distance vector routing, each node generates forwarding table that gives next hop of the shortest path to the destination (forwarding table is explained in function getForwardingTable()). In this project, each node would keep 2 best hops instead of 1 for each other node in the topology. This approach is useful to handle the situation when there is congestion in the network. An example is provided in the section "Flow Routing Simulation".

ModivSim Overview:

In this project, you are asked to develop and evaluate ModivSim, a modified Distance Vector Routing Simulator, that executes the specified Distance Vector Routing Algorithm on a given input topology, prints the intermediate steps and the final distance tables of the nodes. ModivSim 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. In your simulator, each node should be represented as a separate process/thread. You can run all the nodes locally. Each node (process) should communicate with the other through socket programming. If you use threads as nodes, you can use thread communication technique to run the simulation. As the nodes send messages to the neighbors, there might be synchronization problems, but the routing algorithm is designed to converge after some iterations. In real world, the nodes are supposed to be distributed in the network.

ModivSim Code Structure:

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

Input:

ModivSim simulator should read an input topology from an node text files. Each node text file has the input topology of respective node. Each row is in the form of:

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

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

Node0.txt: 0,(1,5,10),(2,3,15)
Node1.txt: 1,(0,5,10),(2,9,5),(4,1,15)
Node2.txt: 2,(0,3,15),(1,9,5),(3,3,10)
Node3.txt: 3,(2,3,10),(4,7,5)
Node4.txt: 4,(1,1,15),(3,7,5)

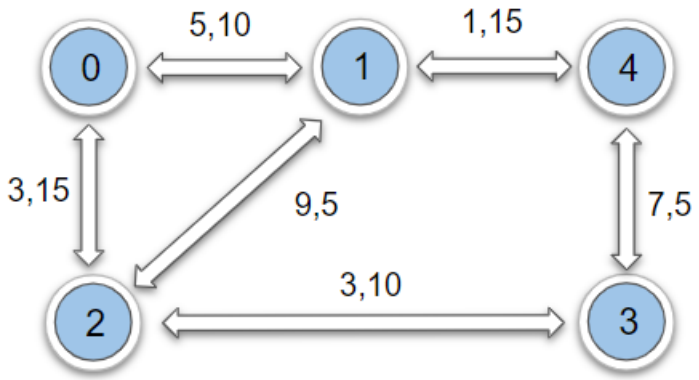

Figure 1- Sample Topology for input.txt

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 **linkBandwith** hash table with neighbor id as the key and the link bandwidth to the neighbor as the value.
- A **distanceTable** with the implementation of your choice. Recall that, 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".
- A **bottleneckBandwidthTable** (to be used in "flow routing simulation" part of the project) that contains information about the bottleneck link bandwidth from the current node to the target node. You can declare as one dimensional array, where entry-i in the table gives bottleneck (the minimum bandwidth seen in the link) from current node to the node 'i'. As an example, node '0' in figure 1 should keep '10' as bottleneck link bandwidth for node '1' and also for node '4'.

Your Node class should also implement the following functions:

- **public Node(int nodeID, HashTable<> linkCost, HashTable<> linkBandwidth)**: 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 ModivSim. 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 nodes. For example, the key j and value (k1, k2) 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 k1 or k2. Make sure that k1 and k2 are arranged in ascending order of the cost of the respective paths. As can be seen in the example forwarding table, if you want to reach node 3 from node 0, you can take the hop to 2 or 1, but 2 gives the shorter path than forwarding to 1.

Key	Value
1	(1, 2)
2	(2,1)
4	(1, 2)
3	(2, 1)

Example Forwarding table for node 0.

Message Class:

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

ModivSim Class:

The ModivSim class is the main class of the simulator that starts by initializing each router. A number of windows appear on the screen, one for the simulator output and one for each router node that is initialized, see Figure 2 (the content of the terminal will be different depending on your own implementation). In your implementation, in the window of each of the router nodes, the minimum distance vector and forwarding table should be displayed. Optionally, the physical cost to each of the neighbors as well as minimum distance vectors received from neighboring nodes can also be shown. You should implement the modified distance vector algorithm inside this class. The algorithm should be executed in a periodic manner. After every p second (where p is the period), each node's sendUpdate function is invoked to send updates to its directly

connected neighbors. The ModivSim should be able to detect the convergence and print the number of the rounds taken before the Distance Vector algorithm converges.

As an alternative, you can start several terminals manually (instead of main class creating windows) and then detect the convergence of the algorithm.

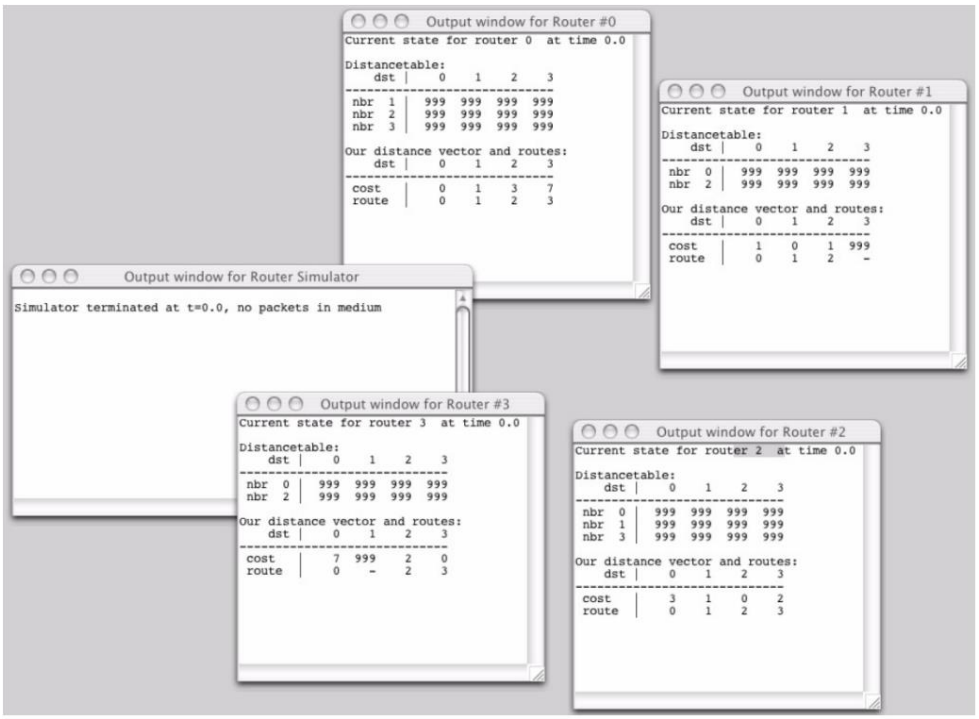

Figure 2 – Sample output of the simulation.

Requirements:

Static Link Cost Scenario

In this scenario, your ModivSim should perform the algorithm in a periodic manner, and at each round show the distance table of each node of the input topology. ModivSim should detect the convergence of the nodes but never terminates. Upon convergence, it should print out the distance table and forwarding table of each node. Simulate the topology of Figure-1 with your implemented ModivSim and report the number of rounds it takes for the topology to converge 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 I dynamic links. For each simulation, you should observe the overall number of rounds it takes for your simulator to converge. 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.

Flow Routing Simulation:

Once you have the implementation of the specified distance vector routing algorithm and the information about the shortest paths, the task is to test this by sending different packet flows through the topology. For this part, you will be using the static link cost implementation. In the network, there are different network flows going through the routers. Usually there is packet switching happening i.e. each flow is divided in packets and all the flows reach the destination packet by packet. The following example shows how the flows are allocated to the links:

Flow A: 0 -> 3, 100Mbits

This means there is a flow from node 0 to 3 that sends 100 Mbits of data in between. If we take the above-mentioned example topology, the shortest path from 0 to 3 is 0->2->3 and the total cost is 6. When we calculate the bandwidth of this path (which is the bottleneck link bandwidth) that is 10 Mbps that means this path would be occupied by the flow for 10s.

Now if we have another flow:

Flow B: 0->3, 200Mbits

We can not use the path 0->2->3 as it is being used by Flow A. There are several algorithms that are used, but for our simplicity we would select the next available shortest path. So now our algorithm would select the path 0 -> 1 -> 4 -> 3 for 10 seconds as the shortest path is not available. This was an example when the flows from the same end points overlap with each other. As an example where flows from different end points collide, consider Flow A and the following Flow C:

Flow C: 1->2, 100Mbits

The shortest path from 1->2 is 1->0>2, but the link from 0->2 is already being used by Flow A, so that cannot be used by Flow C. In this case, your simulator should select a different path for the flow. If there is no path available, then the flow should go in the queue (from where comes the queueing delay) and then assigned a path when available.

You are asked to simulate this behavior. You should write a program that takes flows as an input and outputs the best available path during for each. Test would be performed with different input flows and the outputs would be observed.

Requirements for Flow Routing:

By this time, your algorithm is converged; meaning that each node has the information of the shortest distance from itself to every other node in the topology and can generate forwarding table. You can collect that information and write a code for this part of the project.

Your code should take the flow information as an input and assign a path to each flow. You can keep a finite amount of flows in the flow.txt file and sequentially send them through the topology. The flow.txt would keep information for each flow as follows.

Flow label,<source, destination, flow size>, …., Flow label,<source, destination, flow size>

Example flow.txt:

A, 0, 3, 100
B, 0, 3, 200
C, 1, 2, 100

Your program should

1. Output the flows and their assigned paths.
2. Output the flows that are in the queue.

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 ModivSim. 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 ModivSim.
- **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:

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

Good Luck!