**Assignment 3 Specifications : Network Layer**

In this assignment, you will modify a given template (JAVA) to simulate simplified model of **“Distance Vector Routing (DVR)”.**

* A topology of routers is given as an input (topology.txt). File input/output is managed in the template. The topology contains the list of connected routers (routerId) and IP Addresses of all interfaces against each router. The first interface given in the file is always dedicated for connecting end devices.
* Open Constants.java. There are two defined constants. One is INFTY which models the infinite distance. Another one is LAMBDA which is used to define the rate of altering the state of routers (A functioning router might be switched off or a router which was switched off might be turned on). LAMBDA must be within [0, 1]. The higher the value is, the more router state changes will occur. An instance of RouterStateChanger.java does the job for you.
* A server (NetworkLayerServer.java) first reads the topology as well as lists the interfaces of routers to connect end devices (client). Each interface is a key in clientInterfaces map, where the value represents the number of end devices connected to the interface (initially 0).
* While reading the topology, the server creates instances of Router. You have to modify Router.java to complete its functionalities. More specifically, you need to implement the following:

|  |
| --- |
| /\*\*  \* Initialize the distance (hop count) for each router.  \* for itself, distance=0; for any connected router with state=true, distance=1; otherwise distance=Constants.INFTY;  \*/  public void initiateRoutingTable()  {    }    /\*\*  \* Delete all the routingTableEntry  \*/  public void clearRoutingTable()  {    }    /\*\*  \* Update the routing table for this router using the entries of Router neighbor  \* @param neighbor  \*/  public void updateRoutingTable(Router neighbor)  {    } |

* I am not going to discuss typical client-server connection establishment here as you have done it before. Go through the entire code and understand how the system works first.
* In NetworkLayerServer.java, you will implement an algorithm which can be considered similar to DVR. The pseudocode is given below

|  |
| --- |
| while(convergence)  {  //convergence means no change in any routingTable before and after executing the following for loop  for each router r //<starting from the router with routerId = startingRouterId, in any order>  {  1. T <- getRoutingTable of the router r  2. N <- find routers which are the active neighbors of the current router r  3. Update routingTable of each router t in N using the  routing table of r [Hint: Use t.updateRoutingTable(r)]  }  } |

**Caution:** In real time networking, the routers are isolated. So, the perfect modeling should have been to implement each router as a separate thread. But here, we are controlling the updates of routing table of all routers from the server from a single thread to make things simple. Please note that, this is not the exact version of DVR. It is a trade-off where you will understand how DVR works and also things are a little bit easier for you to implement.

Here the algorithm starts with the first router (routerId = startingRouterId) sending its routing table to its neighbors which are in “UP” state. The sequence of other routers does not matter. Note that, you may need to make sure router state does not change until the algorithm finishes executing.

* You will find two functions in NetworkLayerServer.java : While implementing DVR() , you are supposed to implement it with **split horizon** and **forced update** as discussed in theory class. You will find the class lecture in “*LectureNL.pdf*”; While implementing simpleDVR(), you will implement the same thing but without **split horizon** and **forced update.**
* Now, we will start connecting clients. For this, you may need to adjust or add new codes inside NetworkLayerServer.java, ServerThread.java and Client.java. **Read these files very very carefully for instructions.**
* Your task is to enable the client adjusting NetworkLayerServer.java and ServerThread.java for the tasks given as pseudo-code below. Remember, you have major works to do for the tasks in ServerThread.java

|  |
| --- |
| /\*  1. Receive EndDevice configuration from server  2. [Adjustment in NetworkLayerServer.java: Server internally handles a list of active clients.]  3. for(int i=0;i<100;i++)  4. {  5. Generate a random message  6. if(i==20)  7. {  8. Send the message to server and a special request "SHOW\_ROUTE"  9. Router assigns a random receiver from active client list [Adjustment required in ServerThread.java].  10. Display routing path, hop count and routing table of each router [You need to receive  all the required info from the server in response to "SHOW\_ROUTE" request]  11. }  12. else  13. {  14. Client sends the message to Server, which [Adjustment required in ServerThread.java] assigns a random receiver from active client list.  15. }  16. If server can successfully send the message, client will get an acknowledgement along with hop count  Otherwise, client will get a failure message [dropped packet]  17. }  18. Report average number of hops and drop rate  \*/ |

* Use instances of Packet.java for sending and receiving messages/requests. You can insert/update/delete any attribute of Packet class if required.
* Finally, write a report comparing
  + average number of hops and drop rate for LAMBDA = 0.01, 0.05, 0.10, 0.25, 0.50 and 0.80. Just show the stats in a table, nothing else.
  + Apply simpleDVR() instead of DVR() keeping LAMBDA = 0.10 and compare how drop rate changes because of implementing split horizon and forced update. Briefly put your reasoning.

**General Instructions**

* **Do Not Copy. You know what happens.**
* **Before coding, understand the entire template and plan your implementation.**
* **I would expect everyone to properly go through the template before the demonstration class. Write down your queries regarding this assignment.**
* **Keep your report as short as possible.**