CS 1652 – Data Communication and Computer Networks – Project#3[1]

Due: Monday July 20th @ 11:59pm

Late submission: Wednesday July 22th @11:59pm with 10% penalty per late day

OVERVIEW

Purpose: To implement a distributed asynchronous distance vector routing protocol

Goal 1: To implement the protocol over a static network topology

Goal 2: To allow the link costs to change

More specific details follow below.

OVERVIEW

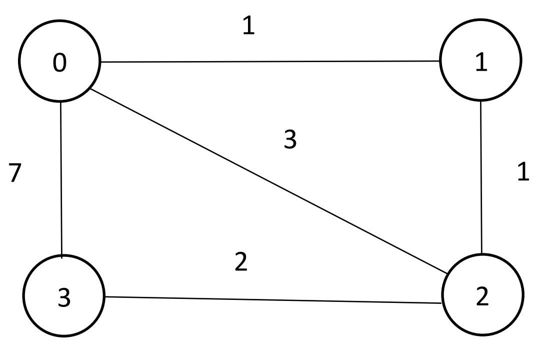

Figure 1. Network topology and link costs for Project 3

In this project, you will be writing a "distributed" set of procedures that implement a distributed asynchronous distance vector routing for the network shown in Figure 1, which shows a small four-node network topology and the link costs.

THE ROUTINES YOU WILL WRITE

You are to write the following routines which will ``execute'' asynchronously within the emulated environment that we have written for this project.

For node 0 (Entity0.java), you will write the following methods:

- Entity0(). The constructor will be called once at the beginning of the emulation and has no arguments. It should initialize the distance table in node 0 to reflect the direct costs of 1, 3, and 7 to nodes 1, 2, and 3, respectively. In Figure 1, all links are bi-directional and the costs in both

[1] Adapted from Computer Networking: a Top-Down Approach, 7th Edition.

directions are identical. After initializing the distance table, and any other data structures needed by your node 0 routines, it should then send its directly-connected neighbors (in this case, 1, 2 and 3) the cost of its minimum cost paths to all other network nodes. This minimum cost information is sent to neighboring nodes in a routing packet by calling the routine NetworkSimulator.toLayer2() as described below. The format of the routing packet is also described below.

- update(Packet p). This method will be called when node 0 receives a routing packet that was sent to it by one if its directly connected neighbors. The parameter p is a reference to the packet that was received.

update() is the "heart" of the distance vector algorithm. The values it receives in a routing packet from some other node *i* contain *i*'s current shortest path costs to all other network nodes. update() uses these received values to update its own distance table (as specified by the distance vector algorithm). If its own minimum cost to another node changes as a result of the update, node 0 informs its directly connected neighbors of this change by sending them a routing packet. Recall that in the distance vector algorithm, only directly connected nodes will exchange routing packets. Thus nodes 1 and 2 will communicate with each other, but nodes 1 and 3 will not communicate with each other.

As we saw in class, the distance table inside each node is the principal data structure used by the distance vector algorithm. It is declared as a 4-by-4 array of int's inside Entity.java, where entry [i,j] in the distance table in node 0 is node 0's currently computed cost to node *i* via direct neighbor *j*. If 0 is not directly connected to j, you can ignore this entry. We will use the convention that the integer value 999 is ``infinity.''

Figure 2 provides a conceptual view of the relationship of the methods inside node 0.

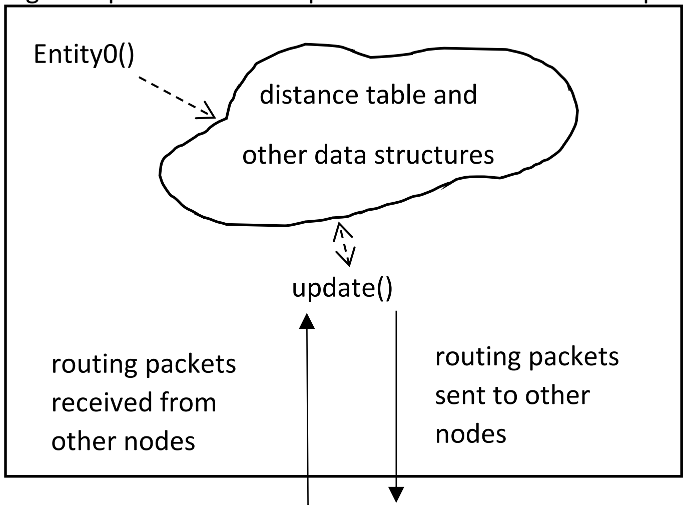
Entity0()
distance table and
other data structures
update()
routing packets received from other nodes
routing packets sent to other nodes

Figure 2 Relationship between methods inside node 0

Similar methods are defined for nodes 1, 2 and 3. Thus, you will write 8 methods in all: Entity0(), Entity1(), Entity2(), Entity3(), Entity0.update(), Entity1.update(), Entity2.update(), and Entity3.update().

Software Interfaces

The methods described above are the ones that you will write. We have written the following methods that can be called by your routines:

- NetworkSimulator.toLayer2(Packet p) where Packet, which is already declared for you in Packet.java, has the following data fields.

  private int source;

  private int dest;

  private int[] mincost;

  The method toLayer2() is defined in the file NetworkSimulator.java.
- Entity0.printDT() will pretty print the distance table for node 0. Similar pretty-print routines are defined for you in the files Entity1.java, Entity2.java, and Entity3.java.

The Simulated Network Environment

The methods Entity0(), Entity1(), Entity2(), Entity3(), Entity0.update(), Entity1.update(), Entity2.update(), and Entity3.update() send routing packets (whose format is described above) into the medium. The medium will deliver packets in-order, and without loss to the specified destination. Only directly-connected nodes should be able to communicate. The delay between the sender and the receiver is variable (and unknown).

When you compile your methods and the provided methods together and run the class in Project.java, you will be asked to specify only one value regarding the simulated network environment:

Tracing. Setting a tracing value of 1 or 2 will print out useful information about what is going on inside the emulation (e.g., what's happening to packets and timers). A tracing value of 0 will turn this off. A tracing value greater than 2 will display all sorts of odd messages that are for emulator-debugging purposes. A tracing value of 2 may be helpful to you in debugging your code. You should keep in mind that real implementors do not have underlying networks that provide such nice information about what is going to happen to their packets!

Part 1

You are to write the methods Entity0(), Entity1(), Entity2(), Entity3(), Entity0.update(), Entity1.update(), Entity2.update(), and Entity3.update() which together will implement a distributed, asynchronous computation of the distance tables for the topology and costs shown in Figure 1.

You should put your methods for nodes 0 through 3 in the files called Entity0.java, ..., Entity3.java. You are NOT allowed to access any global variables that are visible outside of a given file (e.g., any variables you define in Entity0.java may only be accessed inside Entity0.java). This is to force you to abide by the coding conventions that you would have to adopt if you were really running the methods in four distinct nodes. To compile your routines run: javac Entity0.java Entity1.java Entity2.java Entity3.java.

PART 2

You are to write two methods, Entity0.linkCostChangeHandler(int whichLink, int newCost) and Entity1.linkCostChangeHandler(int whichLink, int newCost), which will be called if (and when) the cost of the link between 0 and 1 changes. These methods should be defined in the files Entity0.java and Entity1.java, respectively. The methods will be passed the id of the neighboring node on the other side of the link whose cost has changed, and the new cost of the link. Note that when a link cost changes, these methods will have to update the distance table and may (or may not) have to send updated routing packets to neighboring nodes. FYI, the cost of the link will change from 1 to 20 at time 10000 and then change back to 1 at time 20000. Your methods will be invoked at these times.

SUBMISSION REQUIREMENTS

You should upload the **source files and sample output**. For the sample output, your methods should print out a message whenever they are called, giving the time (You will have to expose the time variable defined inside the NetworkSimulator class). For the update() methods, you should print the identity of the sender of the routing packet that is being passed to your method, whether or not the distance table is updated, the contents of the distance table (you can use the provided pretty-print methods), and a description of any messages sent to neighboring nodes as a result of any distance table updates.

The sample output should be an output listing with a TRACE value of 2. Highlight the final distance table produced in each node. Your program will run until there are no more routing packets in-transit in the network, at which point the emulator will terminate.

GRADE BREAKDOWN

Item	**Number of Points**
The four constructors	32
The four update methods	32
The two linkCostChangeHandler methods	26
Sample output	10