Assignment 3: Implementing distance vector routing protocol

Due Date: Friday April 1st, 11:59 p.m.

Objectives

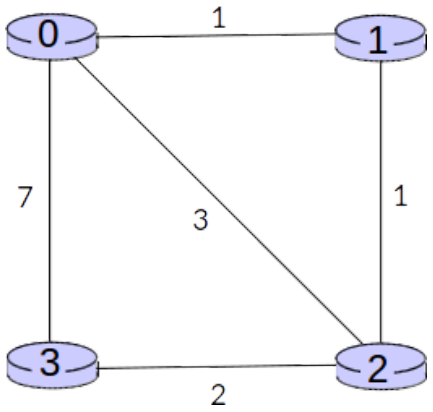
- Implementing a distributed asynchronous distance vector routing protocol

General Instructions:

- Make sure to use python 3.5 or higher.
- Download **start.zip**. It contains the files you need for this assignment
- You should not change any other file. All the changes must be done within **`Node.py`**.
- Make sure to watch the supporting video.
- You run the code as shown in the following:

  > **python `NetworkSimulator.py`**

  The simulator is asking for the number of nodes and trace values. If the number of nodes is four the network will have the following structure:



  you will see that the simulator is displaying the distance table for each router, as well as the next node to be taken to reach a destination. Initially everything is zero. Once you develop the code for `Node.py` they will change as shown in the sample outputs provided.

- Your output should match exactly the given output. They are printed by the simulator. Make sure you do not have extra printing in your code when submitting. When developing and debugging your code, you may need to put lots

of print statements. Remember to comment them out or remove them before submitting.

The functions you will develop

In this programming assignment, you will write the code within a router implementing distance vector routing protocol. For a router you implement the following functions:

- **`__init__(self, ID, networksimulator, costs)`:** where **`ID`** is the router identifier (for the example network it is 0, 1, 2 or 3), **`costs`** is a vector consisting the cost of the links connecting the node to the neighboring nodes, the value is 999 (assumed to be infinity) if there is not a link to a node.

  The function initializes a router (Node object). The constructor is called by the simulator and you should not worry about calling it. When you run the NetworkSimulator.py If you pass 4 as the number of nodes, then it initializes 4 nodes. This function also initializes the distance table for each node. and sends the initial distance table to neighbors of a node.

- **`recvUpdate(pkt)`:** When a router receives an update from its neighbors, it uses that to update its distance table. The value a node receives is in **`pkt`** which is an object of the class of `RTPacket` defined in `common.py`. If there is any change in the distance table of the node, then it informs its directly connected neighbors. The communication is done by sending a packet of type `RTPacket`.

The above two are the main functions you need to implement. It is okay to define helper functions. In this assignment you are implementing the code at the network layer. You are using the interfaces provided by the underlying layer (link layer or layer 2) and you are assuming the link layer is sending the messages reliably and in order to other routers.

Software Interfaces

The functions described above are the ones that you will write. The following functions can be called by the above functions.

- **`tolayer2(pkt)`**: where pkt is an object of class `RTPacket` defined in common.py:

  ```
  class RTPacket:
      def __init__(self, sid, did, mincosts):
          self.sourceid = sid
          self.destid = did
          self.mincosts = deepcopy(mincosts)
  ```

  A router (Node object) calls this function when it has an update in its distance table and wants to send it to its directly connected neighbors.

- **`printdt()`:** prints the distance table for a router. It is already called from within NetworkSimulator. You can call this function for debugging purposes in your code, but make sure to comment it out or remove it before submitting the code. I do not want extra print in your code.

The simulated network environment

When you run the simulator, you will be asked to specify values regarding the simulated network environment:

- **Number of nodes: possible values are 3, 4, and 9.** 4 assumes the network structure in the above figures. 3 and 9 assumes the network structure used in the lecture for providing examples of distance vector algorithm.
- **Trace** Setting a tracing value of 1 or 2 will print out useful information about what is going on inside the simulation (e.g., it shows the changes in distance tables and messages that are communicated between routers). A tracing value of 0 will turn this off. A tracing value of 2 will display all sorts of odd messages that are for the simulator. A tracing value of 2 may be helpful to you in debugging your code. A tracing value of 1 will display the distance table in each router at the end of the simulation as well as the next hop router to a destination.

Sample outputs:

In addition to the simulator code, three sample outputs are provided.

- output-3: Number of nodes is three and the network structure is the same as the first example in the lecture on DV algorithm.
- output-4: Number of nodes is four and the network structure is the same as the above figure.
- output-5: Number of nodes is three and the network structure is the same as the second example in the lecture on DV algorithm.

Helpful Hints

Debugging. We'd recommend that you put LOTS of print statements in your code while you're debugging your functions. Don't forget to disable them before you submit your code. **I do not want any print statement by your code**

Events in simulator. There are three events happening on the sender side:

- FROM_LAYER2
- LINK_CHANGE

In the distance vector algorithm, a router updates its distance table if it receives any update from its neighbors (FROM_LAYER2 event) or if there is any change in the links connected to the router (LINK_CHANGE event). In this assignment you only implement the updates due to FROM_LAYER2 event (when a router receives updates in distance tables of its neighbors).

A description of files in the supplied code

`Common.py`: include the definitions of classes: RT`Packet`, `Event`, `EventType` and `EventList`. You need to know about the RT`Packet` class. The rest are used by the simulator.

`NetworkSimulator.py`: The main code for simulator. It asks the user to enter the parameters needed to initialize the simulator. You can hardcode some values when testing your code. It initializes an object of type `NetworkSimulator`.

It includes the implementations for all the interfaces needed to communicate with link layer (layer 2). You should not change this code. When a `NetworkSimulator` is initialized, it creates the routers (Node objects) and the links between them. When the

routers are initialized, they are passed a reference to the current `NetworkSimulator` object.

Node.py: It includes the constructor for a Node as well as `recvUpdate` method. Each `Node` has an attribute named `ns` that points to the `NetworkSimulator` object. So each router has access to the simulator methods through this member variable. For example, if you want to call `tolayer2` function from one of the methods in `Node.py`, you should call it like the following:

```
self.ns.tolayer2
```

Remember: to access an attribute of a class, you need to precede that with the `self` keyword in Python.

Each node also has two attributes: `distanceTable` which stores the distance table of a router and `routes` which stores the next hop to be taken to reach a destination.

submission instruction:

- Submit only the following file: Node`.py`.
- Do not submit any other files. **Do not zip the files.**
- Only one submission per group: Make sure to put both names at the beginning of the file. Put both names on the same line.
- Your program should not print anything. Make sure to remove or comment out any debugging message.

Grading (tentative, subject to change)

It is very important for your code to run. Otherwise you lose a big portion of the mark.
[5] Following submission instruction
[-25] code does not run
[30] Distance Tables correct
[15] Route Tabless correct