CS 455: Computer Communications and Networking
Distributed Distance Vector Routing
Due date: 11/25, 11:59pm

Project description

In this assignment, you will implement distributed distance vector routing (DVR) algorithm. The idea is that you are provided with a sample network (nodes, their connectivity and link weights), and your job is to find the shortest paths from each node to all other nodes using DVR algorithm that we studied in class. Your implementation of DVR should have three important features:

(1) Distributed: Nodes do not have complete network topology at the start of the algorithm. Each node will send/receive information to their neighbors and the network as a whole will converge after a few rounds on message sharing.

(2) Concurrent: Because you are implementing all nodes on the same computer, you have to implement each node as a thread. Each node thread will open a TCP socket with its neighboring node thread and exchange the DV messages.

(3) Asynchronous: For simplicity, we will assume that we know the order in which nodes send out their DV to their neighbors. This means that at any point of time, you will only have one node sending out DV messages to its neighbors.

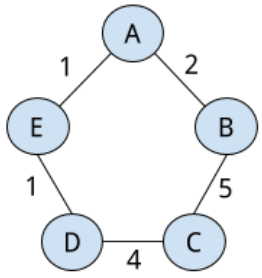
We will now look at how your program should be designed to have all three features.

Reading network graph

The first step towards implementing DVR is to know the network topology. We know that each node does not know the entire network topology in DVR, so you will first write a network_init() function in your code. The network_init() function performs the following tasks:

1. It will read the network topology from network.txt input file. The network.txt will have an adjacency matrix as shown below:

0	2	0	0	1
2	0	5	0	0
0	5	0	4	0
0	0	4	0	1
1	0	0	1	0



Adjacency matrix is a NxN matrix where N is the number of nodes and every element (i, j) indicates the weight of the link between node i and j. If the weight is 0, nodes and i and j are not directly connected to each other. This matrix is symmetric meaning that weight from node i to j is the same weight from node j to i.

The network.txt will be a plain txt file with nothing more than the adjacency matrix. You are free to hardcode the filename in your code.

We will assume that N = 5, i.e., your adjacency matrix is going to be exactly 5x5. However, the nodes can be connected in any way (not necessarily in a ring topology as shown above).

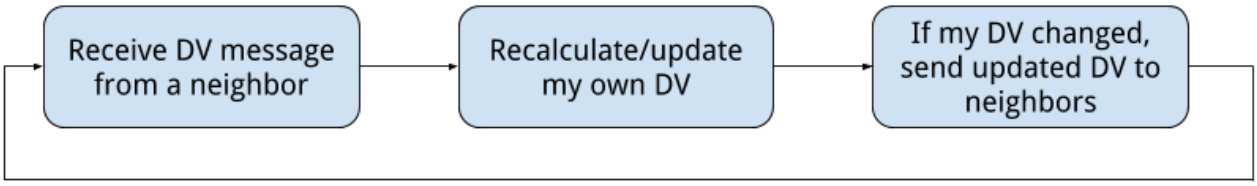
2. Once you have read the network topology, your network_init() function will create N threads (one for each node). Note that your implementation can use more than one thread for each node based on your conceived design.

   At the time of creating individual threads, network_init() will pass each node thread (i) who are the neighbors of the node and (ii) the link weights to the neighbors. For example, for node A above, this would be {(E,1), (B,2)}. The node threads are not allowed to look at the entire network topology as DVR is a distributed algorithm.

After this point, the network_init() function relinquishes control to the node threads.

Operations at a each node

Each node will carry out fixed set of operations within its thread. At a high level, each node is performing the actions shown below.



DV Matrix and DV Message

Before understanding these operations in details, let's see how we can create our DV message structure. Each node maintains a NxN matrix which we refer as DV matrix. As we discussed in the class, DV matrix is what the node uses to run the DVR algorithm. In a DV matrix at node k, an element (i, j) indicates node k's currently computed cost to node i via direct neighbor j. If k is not connected to j, you can ignore this entry. We will use the convention that the integer 999 indicates "infinity". Note that DV matrix is not the same as the adjacency matrix we discussed above.

When a node sends DV message to its neighbor, it does not send its DV matrix in the message. Instead, it sends its own DV estimate (a row in the DV matrix) to its neighbors. Refer to our lecture slides/book to understand the difference.

Socket Connections

Each node will require multiple socket connections in order to send and receive DV messages from its neighbors. Note that there can be many possible ways you can choose to implement this part, but we will discuss one possible way.

Each node can create a TCP server socket. This is the socket that its neighboring nodes will use to send DV messages to it. This socket can also be used as the identity of the node. You are allowed to implement a global data structure that is essentially a list of server socket of each node. When node A wants to send message to node B, it can look up the list and find the server socket for node B.

While the server socket can be used receive messages, you can use the client socket to send DV messages. If node A wants to send its DV message to its neighbor node B, it first creates a client TCP socket, connects to node A's server socket and sends the message. You can choose to maintain that connection or close it and open it again next time.

Now the three operations shown above in the figure can be summarized as below:

1. Receive DV message from a neighbor: receive DV message on node's TCP server socket from a neighbor's client socket.
2. Update my DV estimates: Based on DV message received from my neighbor, update my DV estimates (DV matrix data structure) based on bellman-ford equation.
3. Send DV message to neighbors: If my DV estimates are different from before, open a client TCP socket to a neighbor's server TCP socket and send the my updated DV. Repeat this for every neighbor.

DV Message Sending Order and Stopping Condition

We will assume that nodes send out that DV messages in a strict order in each round. For N=5, this order will be A, B, C, D and E. This order will keep repeating round after round (i.e., A, B, C, D, E, A, B, C, D, E, …). Let us take an example based on the ring topology shown above. First, node A sends out DV message to its neighbors B and E. Both B and E will update their DV based on message from A. However, if their DV has changes, they will not immediately send out message to their neighbors, but instead wait until it is their turn in the round to send the message. If B and E both have updated DVs, B will send out the messages to its neighbors next, but E will wait until A, B, C and D have completely sending out their messages. Until this point, node E can keep on updating it DV and then send out the

latest update to its neighbors. Because you are sending messages one by one, at any point of time, there will be only two nodes communicating (exchanging DV messages).

The message sending will stop when the shortest path from all nodes to all nodes are found. One way to check that would be to see if DV messages are sent but no changes occur any more.

Printing Debug Messages and Final Output

Your code should output the following messages upon running -

```
-------
Round 1: A
Current DV matrix = …
Last DV matrix = …
Updated from last DV matrix or the same? Updated

Sending DV to node B
Node B received DV from A
Updating DV matrix at node B
New DV matrix at node B = …

Sending DV to node E
Node E received DV from A
Updating DV matrix at node E
New DV matrix at node E = …
-------
Round 2: B
Current DV matrix = …
Last DV matrix = …
Updated from last DV matrix or the same? Updated

Sending DV to node C
Node C received DV from B
Updating DV matrix at node C
New DV matrix at node C = …

Sending DV to node A
Node A received DV from B
No change in DV at node A
-------
…..
```

.....

Final output:
Node A DV = ...
Node B DV = ...
Node C DV = ...
Node D DV = ...
Node E DV = ...
Number of rounds till convergence (Round # when one of the nodes last updated its DV) = ...

Policies and submission

Programming language

Implement your code in Python.

What to submit? [Read this before you start working on the project]

- Submit the following
    1. Your code in a zip archive file. If you simply include a pre-compiled executable binary and do not include your code files in the archive, no points will be given.
    2. A README.txt which explains how to compile your program. A standard way in which your code will be ran and tested is through "./my-dvr" command. Include the command using which your compiled code should be ran and tested.
    3. An OUTPUT.txt file that shows the output of your code for following two networks.