CS 455: Computer Communications and Networking
PA-3: Distributed Distance Vector Routing

Project description

In this assignment, you will implement a 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 the 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 as in distance vector routing. Each node will send/receive information to its neighbors and the shortest paths in the network as a whole will converge after a few rounds of message sharing.

(2) Concurrent: Because you are implementing all nodes on the same computer, you have to implement each node as a thread. Similar to BGP, 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 in 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.

(1) 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:

It will read the network topology from the network.txt input file. The network.txt will contain 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

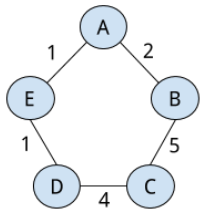


Fig. 1: Adjacency matrix

The adjacency matrix is an 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 the weight from node i to j is the same weight from node j to i.

The network.txt will be a plain text 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) Creating node threads

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 their neighbor connectivity: (i) who are the neighbors of the node and (ii) the link weights to the neighbors. For example, for node A above in Fig. 1, this would be {(E,1), (B,2)}.

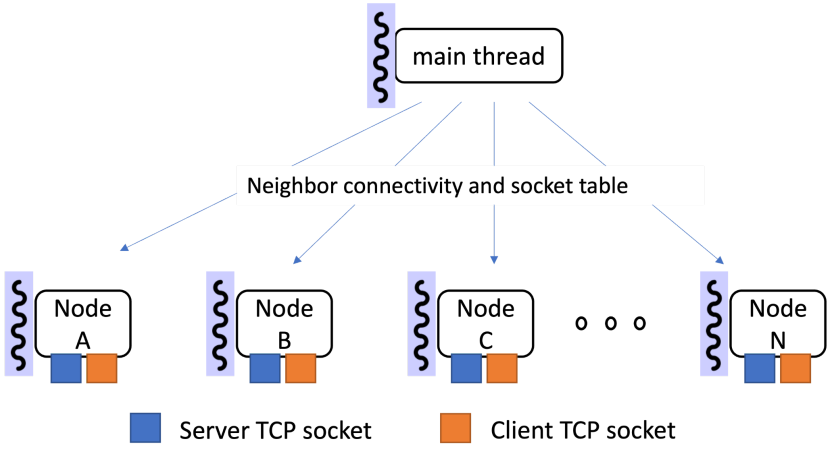


Fig. 2: Threads and socket connections

(3) Distance vectors and DV message

Let's see how we can create our DV message structure. Each node maintains (M+1) distance vectors - one of itself and M others received from its M neighbors. The size of a distance vector is N where N is the number of nodes in the network. Node x's distance vector, that is Dx = [Dx(y): y in N], contains x's estimate of its cost to all destinations y in N. The nodes will send their own DV to their neighbors in the form of a DV message. The order in which the nodes will send out DV messages is discussed below.

(4) Socket connections

Each node will require socket connections in order to send and receive DV messages from its neighbors. Here, we will discuss one possible way to implement this.

Each node will have two types of sockets: TCP server socket and TCP client socket (see Fig. 2 above). (1) TCP server socket: This is the socket that its neighboring nodes will use to send DV messages to a node. When node A wants to send a message to node B, it can look up the socket table and find the server socket for node B.

(2) TCP client socket: While the server socket can be used to 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 uses its client socket to connect to Node B's server socket and sends the message. It can disconnect it from Node B and then connect to another node's server socket to send message to that and so on.

These sockets can be the identity of the nodes. You are allowed to implement a global data structure (socket table) that is shared by the main thread with all threads so that nodes can find the socket of other nodes.

(5) DV message sending order and stopping condition

We will assume that nodes send out 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 messages to its neighbors B and E. Both B and E will update their DV based on the message from A. *However, if their DV has changed, they will not immediately send out a 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 completed sending out their messages. Until this point, node E can keep on updating its 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 is found. One way to check that would be to see if DV messages are sent but no changes occur anymore.

(6) Printing Debug Messages and Final Output

Your code should output the following messages upon running -

---

Round 1: A

Current DV = …
Last DV = ...
Updated from last DV or the same? Updated

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

Sending DV to node E
Node E received DV from A
Updating DV at node E
New DV at node E = ...

Round 1: B
Current DV = …
Last DV = ...
Updated from last DV or the same? Updated

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

Sending DV to node A
Node A received DV from B
No change in DV at node A

…..
…..

Round 2: A
Current DV = …
Last DV = ...
Updated from last DV or the same? Updated

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

Sending DV to node A
Node A received DV from B
No change in DV at node A

Round 2: B

Current DV = …
Last DV = ...
Updated from last DV or the same? Updated

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

You can implement your code in C or Python.

Working with a partner

You are required to do this project with your project partner of PA1 and PA2. You can choose to do it alone but cannot change the partner from PA1 and PA2 without my permission.

Note on plagiarism

In this class, it is absolutely mandatory that you adhere to GMU and Computer Science Department honor code rules. This also means (1) do not copy code from online resources and (2) your implementation should be purely based on your own thought process and conceived design, and not inspired by any other students or online resources, (3) you can copy your code or design from other students in the class.

We reserve the right to check your assignment with other existing assignments (from other students or online resources) for cheating using code matching programs. Any violation of the honor code will be reported to the GMU honor committee, with a failing grade (F) in this course.

*** Do not put your code online on Github or other public repositories ***
Violation of this policy would be considered an honor code violation.

Grading, submission, and late policy

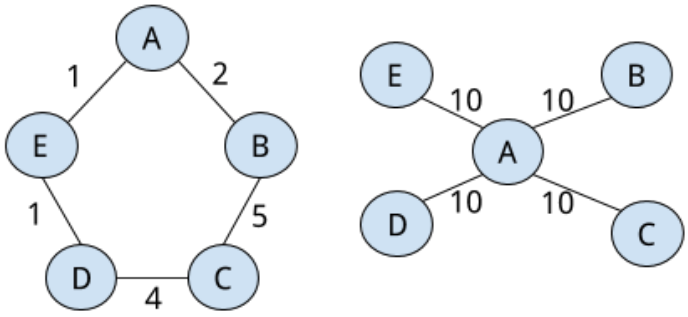
- The standard late policy applies - Late penalty for this lab will be 15% for each day. Submissions that are late by more than 3 days will not be accepted
- This lab accounts for **11%** of your final grade
- You will submit your solution via Blackboard

Grading rubric

network_init() function (read and parse the txt file) Create N Threads (one for each node)	5 points 10
DV message format (what information is being sent between nodes) Send and Receive DV messages (sockets' connection logic) Recalculate/update the DV	5 35 20
Sending order (A,B,C,D,E,A,B,...) Stop condition (if the code stops in a reasonable # of steps)	10 5
Print debug messages and final output	10

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 run and tested is through "./my-dvr" command. Include the command using which your compiled code should be run and tested.
    3. An OUTPUT.txt file that shows the output of your code for the following two networks.



4. A PARTNERS.txt file mentioning the name and GMU IDs of two students who worked on the project as a team. Both team members should submit these four pieces separately on Blackboard before the deadline.
