THE UNIVERSITY OF SYDNEY

Due: 30 April 2021(Friday Week 8) by 11:59 PM

COMP3221
Assignment 1: Distance Vector Routing

The goal of this project is to implement a Distance Vector Routing Protocol using Socket Programming in Python.

1 Submission Details

The final version of your assignment should be submitted electronically via CANVAS by 11:59PM on the Friday of Week 8 (Hard Deadline). This is an individual project, and each student has to submit his/her own version.

1.1 Learning Objectives

In this assignment, your task is to implement the distance-vector routing protocol using python. Your program is run at all nodes in the specified network. At each node, the input of your program is a set of its neighbors and the costs of these links. Each node will learn from its neighboring nodes' perspectives and then finds the least-cost path to all nodes in the network. Your program also is able to deal with changing link costs and it should have at least three separate threads for listening (for receiving distance-vector updates), sending (for sending distance-vector updates after every 10 sec), and routing calculations (when an link-cost changes).

On completing this assignment you will gain sufficient expertise in the following skills:

- Designing a routing protocol
- Socket and Multi-threading programming using Python
- Handling routing dynamics

1.2 Network architecture and simulation

You are free to choose the net topology. The topology includes 10 nodes and the link between nodes are generated randomly. Each topology has at least 15 connections, for example, Figs. 1.

Since we do not have access to real network (for implementation and testing of your programs), we will use a simulated network that will run on a single machine. You can open

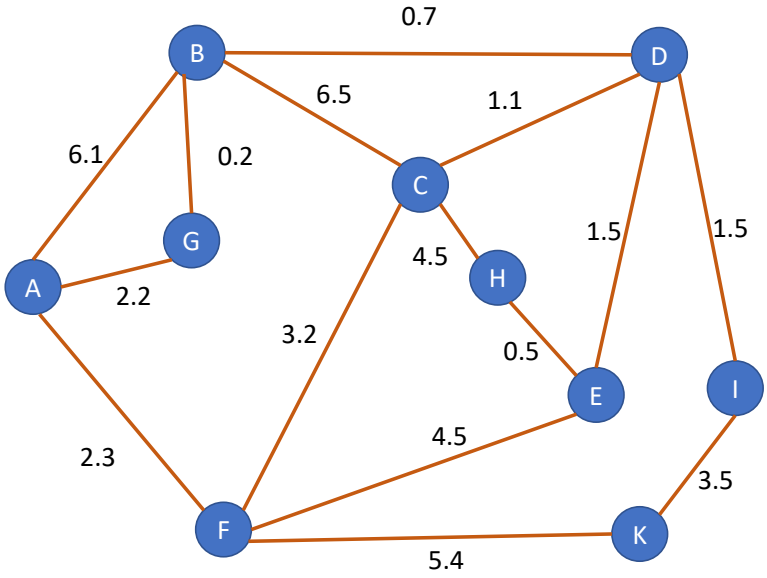

Figure 1: An example of a net topology with 10 nodes.

different terminals (one each for every node in the net topology) to run different instances of your program on the same machine (use "localhost").

1.3 Program structure

The program should be named as **`COMP3221_DiVR.py`** and accept the following command line arguments:

```
python COMP3221_DiVR.py <Node-id> <Port-no> <Node-config-file>
```

For example:

```
python COMP3221_DiVR.py F 6005 Fconfig.txt
```

- **Node-id**: is ID of a node in a net topology. In this assignment, Node-id is indexed following the alphabet, for example, A, B, C, etc..
- **Port-no**: is the port number of a node listening to the distance-vector packets. The port number is integer indexed from 6000 and increased by one for each node. For example, the net topology in Figs. 1 has 10 nodes A, B, C, D, E, F, G, H, I, and K, the port number of each node is 6000, 6001, 6002, 6003, 6004, 6005, 6006, 6007, 6008, and 6009, respectively.
- **Node-config-file**: **Fconfig.txt**, for example, is the configuration file for Node F that has the following details:

```
4
A 2.3 6000
C 3.2 6002
E 2.8 6004
K 5.4 6009
```

The first line of this file indicates the number of neighbors for Node F (it is not the total number of nodes in the network). The next four lines are to determine the connection of Node F to its neighbors. Those lines start with the neighbor ID, followed by the cost to reach this neighbor from Node F, and finally the port number that this neighbor is using for listening.

For example, the second line in the **Fconfig.txt** above indicates that the cost to neighbor A is 2.3 (floating point numbers) and Node A is using port number 6000 for receiving distance-vector packets. It is noted that the link costs will be consistent in both directions, i.e., if the cost from F to A is similar to the cost from A to F. Addionally, Node F must not have global knowledge (i.e. Information about the entire network topology).

Upon initialization, each node creates a distance-vector update packet (containing the appropriate information: its neighboring nodes and cost links between it and its neighbors) and sends this packet to all direct neighbors. You are free to define the exact format of the distance-vector packets. Upon receiving this distance-vector update packet, each neighboring node will incorporate the provided information into its routing table. Each node should periodically broadcast the distance-vector update packet to its neighbors every 10 seconds.

On receiving distance-vector update packets from all other nodes, a node can build up a reachability matrix. Given a view of the neighboring nodes and their reachability, a node should run a routing algorithm (Bellman-Ford, Dijkstra, etc..) to compute least-cost paths to all other nodes within the network. Each node should wait for 60 seconds since start-up and then execute the routing algorithm.

Once a node finishes running the routing algorithm, it will print out to the terminal the least cost path to each destination node in the topology (excluding itself) along with the cost of this path. The following is an example output for node F in some arbitrary network:

```
I am Node F
Least cost path from F to A: FA, link cost: 2.3
Least cost path from F to B: FAGB, link cost: 4.7
Least cost path from F to C: FC, link cost: 3.2
Least cost path from F to D: FCD, link cost: 4.3
Least cost path from F to E: FE, link cost:4.5
```

Your program should execute forever (as a loop). In other words, each node should keep broadcasting distance-vector value packets every 10 seconds and the routing algorithm should be executed every time a change happens.

1.4 Dealing with changes in link cost and failure:

You must ensure that your program is robust to changes in link costs and failures. Once the cost of link changes the network must be able to reconverge to accommodate the costs.

You must provide an interface for each node that can be used to display every link-cost in a network and give the ability to modify the link-cost. A simple method would be to run a separate thread that will modify the configure files. Any modification on the interface will affect the configured files.

Once the cost of a link changes, the connected nodes must recalculate the cost of reaching other nodes and must also provide an update to its neighbors, who will then notify their neighbors and so on until the network converges.

Here is an example of the correct output for Node F before and after the failure of Node C is given. You should check your implementation for correctness at all nodes with multiple node failures.

Output with all nodes working:

```
I am Node F
Least cost path from F to A: FA, link cost: 2.3
Least cost path from F to B: FAGB, link cost: 4.7
Least cost path from F to C: FC, link cost: 3.2
Least cost path from F to D: FCD, link cost: 4.3
Least cost path from F to E: FE, link cost:4.5
```

Output after Node C fails:

```
I am Node F
Least cost path from F to A: FA, link cost: 2.3
Least cost path from F to B: FAGB, link cost: 4.7
Least cost path from F to D: FAGBD, link cost: 5.4
Least cost path from F to E: FE, link cost: 4.5
```

1.5 Submission and Report:

You are required to submit your source code and a short report to Canvas.

- Code (zip file includes all implementation, readme, and config files for all nodes) **`SSID_COMP3321_Code.zip`**.
- Code (including all implementation in one file exported in a txt file for Plagiarism checking)
  `SSID_COMP3321_Code.txt`.
- Report (pdf)
  `SSID_COMP3321_Report.pdf`.

Please note that you must upload your submission BEFORE the deadline. The CANVAS would continue accepting submissions after the due date. Late submissions would carry penalty per day with maximum of 5 days late submission allowed.

The size of your report MUST be under 2 pages. Your report should briefly document your net typology, routing algorithm, techniques, and methodology used for implementation and how you combat the relevant problems in development. It should act a reference for your markers to quickly figure out what you have and haven't completed, how you did it, and it should mention anything you think is special about your system.

1.6 Academic Honesty / Plagiarism

By uploading your submission to CANVAS you implicitly agree to abide by the University policies regarding academic honesty, and in particular that all the work is original and not plagiarised from the work of others. If you believe that part of your submission is not your work you must bring this to the attention of your tutor or lecturer immediately. See the policy slides released in Week 1 for further details.

In assessing a piece of submitted work, the School of Computer Science may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program. A copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.

2 Marking

This assignment is worth 15% of your final grade for this unit of study.

- Code: Account for 80%.
- Report: Account for 20%.