THE UNIVERSITY OF SYDNEY

Due: March 22nd, 2024 (Friday, Week 5) by 11:59 PM

COMP3221
Assignment 1: Routing Algorithm

*The purpose of this assignment is to develop a routing protocol for a specified network topology by utilizing Python's Socket and Multi-threading programming. This project can be done in a group of two students, with only one team member submitting the work on behalf of the group. You need to register a group on the Canvas page: **COMP3221** → **People** → **Group - A1**.*

1 Learning Objectives

Your assignment involves developing routing protocols within a predefined network using Python. Specifically, you are tasked with creating a Python application that facilitates efficient exchange of routing information between nodes and identifies the least-cost paths within the network. Moreover, the program must be capable of managing network challenges, including changes in link costs and node failures.

Completing this assignment will equip you with comprehensive skills in:

- **Routing Protocol Design and Implementation**: Develop efficient routing protocols to find least-cost paths and optimize network data flow.
- **Socket and Multi-threading Programming in Python**: Enhance your programming skills with networking and concurrent programming for improved protocol efficiency and responsiveness.
- **Dynamic Routing Management**: Learn to adapt to network changes, managing link cost variations and failures for robust communication.

2 Assignment Guidelines

2.1 Network Architecture and Simulation

2.1.1 Creating Your Network Topology

Your task begins with constructing a network topology consisting of **10 nodes** interconnected by **at least 15 randomly generated links**. Figs. 1 provides an example network topology with the required number of nodes and connections. *You must design a unique topology and not replicate the provided sample.*

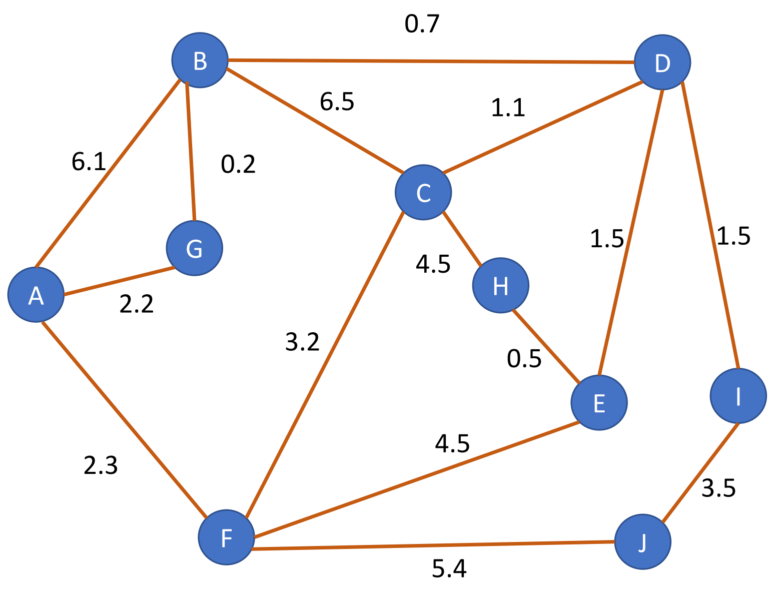

Figure 1: A sample network topology with 10 nodes and 15 connections.

2.1.2 Simulation Environment

Due to the unavailability of a physical network for deployment, you will simulate this network on a single computer for both implementation and testing purposes. This simulation involves running separate instances of your program for each node within the network topology, utilizing "localhost" for communication. Each node will correspond to a different terminal window on your machine, allowing for a comprehensive test environment.

2.2 Program Structure

Your program should be named **`COMP3221_A1_Routing.py`** and should accept command-line arguments as follows:

```
python COMP3221_A1_Routing.py <Node-ID> <Port-NO> <Node-Config-File>
```

Command-Line Arguments:

- **Node-ID**: A unique identifier assigned to each node in the network topology. In this assignment, Node-IDs are indexed using letters of the English alphabet, e.g., A, B, C, ...
- **Port-NO**: Each node in the network listens for updates from its neighbors on a specific port number. In this assignment, Port-No is an integer, starting at 6000 for the first node and incrementing by one for each subsequent node. For instance, the topology in Figs. 1 has 10 nodes: A, B, C, D, E, F, G, H, I, and J. Then, their port numbers are 6000, 6001, 6002, 6003, 6004, 6005, 6006, 6007, 6008, and 6009, respectively.
- **Node-Config-File**: This is the path to a configuration file that contains detailed information about a node's direct neighbors within the network. This file includes the cost to reach each neighbor and the port number each neighbor listens on. The file begins with

a number indicating how many neighbors the node has. Following this, each line provides details for one neighbor: the neighbor's Node-ID, the cost to reach this neighbor (a floating-point number), and the neighbor's listening port number.

For Node F, you might have a configuration file named **Fconfig.txt** with content like:

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

where 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. The second line indicates that the cost to neighbor A is 2.3 (floating-point numbers), and Node A uses port number 6000 for receiving information update packets. It is noted that all link costs are symmetric (the same in both directions, e.g., the cost from F to A is the same as that from A to F). Moreover, Node F must not have global knowledge (i.e., information about the entire network topology).

Usage example:

```
python COMP3221_A1_Routing.py F 6005 Fconfig.txt
```

This tells the program that it's operating as **Node F**, should listen on port **6005** for updates, and can find details about its neighbors in **Fconfig.txt**.

2.3 Assignment Tasks

Your node program should incorporate multi-threading and socket programming to perform multiple tasks concurrently, which is essential for the efficient operation of network protocols. For the assignment, your node program needs to manage **at least three critical functions simultaneously** using separate threads:

1. **Listening Thread:** This thread is dedicated to listening for incoming routing information from neighboring nodes. It continuously monitors the designated port for messages and processes any received data. The information typically includes updates from neighbors about their link costs and network topology changes.
2. **Sending Thread:** This thread periodically sends out the node's current routing information to its neighbors. At regular intervals, it broadcasts the latest routing table or link-cost updates to all neighboring nodes. This ensures that all nodes maintain an up-to-date view of the network topology.
3. **Routing Calculations Thread:** This thread triggers the execution of routing algorithm calculations upon detecting changes in link costs or receiving new routing information. Whenever there's a change in the network topology (e.g., link cost update, node addition/removal), this thread recalculates the least-cost paths to all other nodes in the network using the node's routing algorithm.

2.3.1 Finding the Least-Cost Paths

At the start of the process, each node in the network generates an update packet, which includes necessary information such as its neighboring nodes and the costs associated with reaching them. *You are free to define the structure of these packets*. Then, nodes should periodically broadcast these update packets to their neighbors **every 10 seconds**. This ensures that all nodes in the network continually share and receive up-to-date information about their connections and any changes in link costs.

After receiving update packets from the entire network, each node constructs a routing table, also known as a reachability matrix. This matrix provides a detailed network representation from the node's perspective, indicating which nodes are directly accessible and the associated costs for reaching them. With this information at hand, the node employs a routing algorithm (e.g., Bellman-Ford, Dijkstra, etc.) to compute the shortest paths to all other nodes within the network. To ensure that every node has gathered sufficient information for an accurate network overview, each node should wait for **60 seconds** after startup, and then execute the routing algorithm.

Upon completing the routing algorithm, each node outputs the least cost path to every other node in the network, excluding itself, to the terminal. This output includes both the path taken and the cost associated with that path. For illustration, consider the output for Node F in the example network shown in Figs. 1:

```
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
Least cost path from F to G: FAG, link cost: 4.5
Least cost path from F to H: FEH, link cost: 5
Least cost path from F to I: FCDI, link cost: 5.8
Least cost path from F to J: FJ, link cost: 5.4
```

Your node program is designed to run indefinitely in a continuous loop. This means that each node is responsible for regularly broadcasting its information value packets to all neighbors every 10 seconds, ensuring the network's routing information remains up-to-date. Additionally, the routing algorithm within each node should be re-executed whenever the link costs change, allowing the network to adapt dynamically to changes in its topology.

Note that you can choose any routing algorithm covered in the lectures for this assignment. However, all algorithms must be implemented from scratch. You are not allowed to use pre-existing libraries that provide routing protocol functionalities.

2.3.2 Dealing with Link Cost Changes and Node Failures

Your program must be capable of adapting to changes in link costs and network failures. This adaptability is crucial for ensuring the network can reconverge and update its routing paths to reflect the new costs or topology changes.

Simulate Link Cost Changes: You must provide a command-line interface (CLI) that allows users to modify the costs associated with links connecting a node to its neighbors. For instance, this could be achieved by implementing a separate thread within each node's program tasked with monitoring and applying changes to the configuration files based on the commands entered into the CLI. Any adjustments made via the CLI should also be updated to the node's configuration files, reflecting the new link costs in the network's topology.

Simulate Node Failures: You can also use the CLI to enable users to disable or enable nodes within the network. For example, this could be managed by adding commands to the CLI that mark a node as *down* or *up*. When a node is marked as *down*, it simulates a failure by stopping the broadcast of update packets and ignoring received packets, effectively removing it from the network topology. Marking it as *up* reverses this process, reconnecting the node into the network.

Adapting to Changes and Failures: Upon detecting changes in link costs or node failures, the available nodes in the network must recalculate the least cost paths to reach other nodes, then broadcast the updated routing information to their neighbors. This triggers a cascade of updates across the network, with each node adjusting its routing table based on the received information. This iterative process continues until the network achieves convergence, reflecting a consistent understanding of the network's current state among all nodes.

To validate the robustness and correctness of your implementation in handling link cost changes and node failures, it's essential to examine the output of your routing algorithm before and after simulating such failures. Below is an illustrative example focusing on Node F in Fig. 1, demonstrating the expected output changes due to the failure of Node C.

Before Node C Failure: Assuming all nodes are operational, Node F's routing table might display paths and costs to each destination in the network as follows:

```
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
Least cost path from F to G: FAG, link cost: 4.5
Least cost path from F to H: FEH, link cost: 5
Least cost path from F to I: FCDI, link cost: 5.8
Least cost path from F to J: FJ, link cost: 5.4
```

After Node C Failure: Once Node C is down, Node F's routing table needs to be updated to reflect the absence of Node C, recalculating paths that previously traversed through C:

```
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
Least cost path from F to G: FAG, link cost: 4.5
Least cost path from F to H: FEH, link cost: 5
Least cost path from F to I: FAGBDI, link cost: 6.9
Least cost path from F to J: FJ, link cost: 5.4
```

3 Submission and Report

3.1 Report

Your submission must include a report document that concisely describes your work within a strict limit of no more than 3 pages, with the exception of the references section, which may extend beyond this page limit. Your report should briefly document your net topology, routing algorithm, techniques and methodology used for implementation and the simulation results of each requirement.

1. **Network Topology**: Briefly describe how you generate the network topology, using Visual aids or diagrams for clarity.
2. **Routing Algorithm:** Outline the routing algorithm(s), explaining your selection and providing an overview of its functionality. Highlight any modifications or optimizations made to standard algorithms.
3. **Implementation Methodology:** Describe the programming approaches and tools used, with a focus on specific techniques applied to implement the routing protocols, handle link cost changes and node failures, and ensure continuous operation of the network.
4. **Simulation Results:** Summarize key findings from your simulations, stating what you have and haven't completed. Provide specific examples of how effective your system is in routing and adapting to link-cost changes and failures.

3.2 Submission Files

You are required to submit the following files to Canvas separately.

- Code (a zipped archive contains all your code and config files)
  `SSID_COMP3221_Code.zip`.
- Code Text (a single *.txt* file includes all implementation code for Plagiarism checking)
  `SSID_COMP3221_Code.txt`.
- Readme (A detailed *.txt* file that outlines the coding environment, version of packages used, instructions to launch the program, and guidelines for simulating link-cost changes and node failures)
  `SSID_COMP3221_Readme.txt`.
- Report (A *.pdf* file that includes all content required in the report section)
  `SSID_COMP3221_Report.pdf`.

Note that you must upload your submission BEFORE the deadline. The CANVAS would continue accepting submissions after the due date; however, late submissions would incur a penalty per day with a maximum of 5 days late submission allowed.

4 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.

5 Marking

This assignment contributes 15% to your final grade for this unit of study. The distribution of marks between the assignment components is as follows.

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

Please refer to the *rubric* in Canvas (**COMP3221** → **Assignment** → **Assignment 1** → **Assignment 1 - Rubric**) for detailed marking scheme. The report and the code are to be submitted in Canvas by the due date.

After Assignment 1 marks come out, please submit your inquiries about marking within the 1st week. All inquiries after that will NOT be responded.