The University of Calgary

Department of Electrical & Computer Engineering

ENSF 462 Networked Systems

(Fall 2023)
Due: Nov. 28th, 2023

Lab 4 Routing Algorithm Implementation

Lab Section	Section Date	Location
B01	November 23rd, 2023	ENA 305
B02	November 28th, 2023	ENG 24
B03	November 22nd, 2023	ICT 319

Objectives

The objective of this lab is to implement Dijkstra's link state routing algorithm. By completing this lab, you will learn how routers broadcast link state message among other nodes in the network, find the least cost distance to any router in the network, and create forwarding tables.

Note: The terms "router" and "node" are used interchangeably in this manual.

Definitions

- Node label and id: Each node in the network has a unique label and id. The node label is denoted using alphabets in English language (upper case), starting from *A*. The node id is represented by integers greater than or equal to 0. There is a one-to-one mapping between node id and label. For example, *A* is mapped to 0, *B* is mapped to 1, *C* is mapped to 2 and so on.
- Link state: A link state vector of a router provides information about a router's neighbors, non-neighbors and corresponding link costs. Link cost from a router to itself is set as 0. Link cost to those routers which are not neighbors is set to infinity. Use 999 as infinity in this lab. We suppose that for each edge in the network, link cost is a number less than 999. Furthers, link costs to each node are arranged in increasing order of node id. For example, link state vector of node A in the sample network topology (Figure 1) is [0 2 1 5 999 999]. Similarly, for B it is [2 0 2 3 999 999], for C it is [1 2 0 3 1 999], for D it is [5 3 3 0 1 5], for E it is [999 999 1 1 0 2] and for F it is [999 999 999 5 2 0].

- Previous node: The last node before the destination node on the least cost path from the router to that destination node (see definition of previous node in Dijkstra Algorithm in slides or textbook).

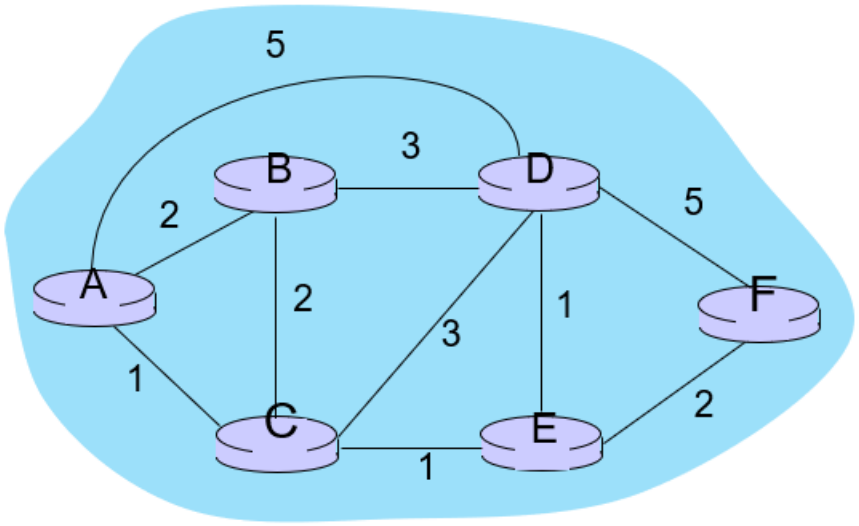

Figure 1: Example Network Topology

Implementation Requirements

You need to write a program called `Router.py` which implements the router behaviour which is composed of the following components:

1. Send Link State information: Every second, each router sends its link state info to all its neighbors.
2. Receive and Broadcast Link State information: Upon receiving a link state info from one of its neighbors, the router records this info and forwards a copy of received info to all its neighbors based on a broadcast algorithm of your choice.
3. Path and Forwarding Table Computation:
    - Compute least cost paths based on Dijkstra algorithm using the whole network topology. This is done every 10 seconds and if whole network topology is already known to the router. If the router does not have all link state information from all nodes yet to have whole network view, it must defer its computation of least cost path for the next 10 seconds.
    - Each time after execution of Dijkstra algorithm, build the forwarding table for the router, which has two columns, the first column is the id of the destination node (note in actual forwarding tables it is the IP address of destination), and the second column is the label of the neighbor that is the first hop on the least cost path to the destination (note that in actual forwarding tables this in the port number of the first link on the path). The forwarding table in each router has an entry for all destination nodes except the router itself.

The routers use UDP protocol to send or receive Link State Information.
You should use **multi-threading** with separate thread for each of three components above: i.e., "Sending Link State Information every second", "Receiving Link State Information from neighbour", and "Computing least cost path and forwarding tables."

You are free in implementing broadcasting messages. Here are some ideas:

- Simple implementation of broadcasting is sending a copy of link state information we receive from any neighbour to all neighbors. This results in many copies of messages to circulate in the network for ever.
- Another way is attaching a counter to each message, which is decremented by one each time it is received in a router. As soon as the counter is 0, the message is not forwarded any more.

You are free to choose one of the methods above or implement more efficient broadcasting algorithm of your choice.

Program running Instructions and Input Description

Your router program **should get three command line arguments** as the following:

Python Router.py <routerid> <routerport> <configfile>

<routerid>: is the router id
<routerport>: is the socket port number used by the router to send and receive link state messages over UDP and
<configfile>: is the filename containing information about total number of routers, and neighboring routers and corresponding link costs **for this router**.

The format of the configuration is as follows:

<Total number of nodes> (first line)
<neighbor node label> <space> <neighbor node id> <space> <link cost> <space> <neighbor node port number> (a line per neighbor)

configuration file for each node WOULD NOT contain entry for nodes which are not neighbors.

The configuration file for node F is shown below as an example (please see uploaded configuration files of all nodes on D2L, for the given sample network):

6
D 3 5 7003
E 4 2 7004

More examples of configuration files (for the network topology represented in Figure 1) have been provided on D2L along with this manual. For the given configuration files, you

can run 6 instances of your Router.py program, one for each router as following (assuming all configuration files are stored in the same location as the Router.py):

```
python Router.py 0 7000 config-A.txt
python Router.py 1 7001 config-B.txt
python Router.py 2 7002 config-C.txt
python Router.py 3 7003 config-D.txt
python Router.py 4 7004 config-E.txt
python Router.py 5 7005 config-F.txt
```

Output

Each instance of router program should print the result of Dijkstra algorithm and the forwarding table every time this algorithm is executed in that router. For example, for node B (with routerid 1) in Figure 1, it should print:

Destination_Routerid	Distance	Previous_node_id
0	2	1
1	0	1
2	2	1
3	3	1
4	3	2
5	5	4

The forwarding table in B is printed as follows:

Destination_Routerid	Next_hop_routerlabel
0	A
2	C
3	D
4	C
5	C

Assumptions

- The underlying graph is connected, i.e., there at least one path from any router to any other router in the network.
- Network topology does not change during a single run of the program.
- All instances of the router program are run in a single machine.
- For simplicity, we will ONLY consider networks with nodes less than or equal to 10.
- Routers' id starts from 0 and goes up one by one (0, 1, 2, 3, …)

Lab 4 Routing Algorithm Implementation

Submit a lab report that includes the following:

- **Your name and UCID #**
- **Your Python file Router.py**
- **Screenshots demonstrating the required outputs for all routers instances.**