National University of Sciences & Technology
School of Electrical Engineering and Computer Science
Department of Computing
EE353: Computer Networks, BSCS4 Fall 2016

Project Specifications (V1)	
CLO 4: Design and implement solutions to contemporary networking issues (through hands on programming)	
Maximum Marks: 10	Instructor: Dr Nadeem Ahmed / Dr Arsalan Ahmed
Date: 14th Nov 2016	Due Date: mid night 1st Jan 2017

1.0 General:

This is a group assignment. Maximum size of group is restricted to 2 members.

Please avoid plagiarism; any such case would result in award of zero marks both to the "sharer" and the "acquirer". Maximum score is 20 points that would be scaled back to 10 marks.

2.0 Learning Objectives

In this assignment your task is to implement the link state routing protocol. Your program will be running at all routers in the specified network. At each router the input to your program is a set of directly attached routers (i.e. neighbors) and the costs of these links. Each router will broadcast link-state update packets to its neighbors who would propagate this broadcast further in the network. Your implementation at each router should report the least cost path and the associated cost to all other routers in the network. Your program should be able to deal with failed routers.

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

a. Designing a routing protocol

b. Link state (Dijkstra's) algorithm

c. Socket programming in Python

d. Multi-threading in Python

e. Handling routing dynamics

3.0 Specifications:

In this assignment, you will implement the link state routing protocol using Python as the programming language. You are provided with the topology map in the form of configuration files. The program should be named LSR.py and it will accept the following command line arguments:

LSR.py A 5000 ConfigA.txt

Where A is the Router ID, 5000 is the port No and ConfigA.txt is the configuration file for Router A that has the following details:

2
B 6.5 5001
F 2.2 5005

The first line of this file indicates the number of neighbors for Router A. Note that it is not the total number of routers in the network. Following this, there is one line dedicated to each neighbor. It starts with the neighbor ID, followed by the cost to reach this neighbor and finally the port number that this neighbor is using for listening. For example, the second line in the configA.txt above indicates that the cost to neighbor B is 6.5 and this neighbor is using port number 5001 for receiving link-state packets. The router ids will be uppercase single letter alphabets. The link costs should be floating point numbers (up to the first decimal) and the port numbers should be integers. These three fields will be separated by a single white space between two successive fields in each line of the configuration file. The link costs will be static and will not change once initialized. Further, the link costs will be consistent in both directions, i.e., if the cost from A to B is 6.5, then the link from B to A will also have a cost of 6.5.

Initially each router is only aware of the costs to its direct neighbors. The routers do not have global knowledge (i.e. information about the entire network topology) at start-up. Upon initialization, each router creates a link-state update packet (containing the appropriate information – see description of link-state protocol in the textbook) and sends this packet to all direct neighbors. The exact format of the link-state packets that you will use is left for you to decide. Upon receiving this link-state update packet, each neighboring router in turn broadcasts this packet to its own neighbors (excluding the router from which it received this link-state packet). This simple flooding mechanism will ensure that each link-state packet is propagated through the entire network.

Each router should periodically broadcast the link-state update packet to its neighbors every 1 second. You are required to use **UDP** as the transport protocol for exchanging link-state packets amongst the neighbors.

On receiving link-state update packets from all other routers, a router can build up a global view of the network topology. Given a view of the entire network topology, a router should run Dijkstra's algorithm to compute least-

cost paths to all other routers within the network. Each router should wait for 30 seconds since start-up and then execute Dijkstra's algorithm.

Once a router finishes running Dijkstra's algorithm, it should print out to the terminal, the least cost path to each destination router in the topology (excluding itself) along with the cost of this path. The following is an example output for router A in some arbitrary network:

I am Router A
Least cost path to router B: ACB and the cost: 14.2
Least cost path to router C: AC and the cost: 2.5

Your program should execute forever (as a loop). In other words, each router should keep broadcasting link-state update packets every 1 second and Dijkstra's algorithm should be executed and the output printed out every 30 seconds.

3.1 Restricting Link-state Broadcasts:

Note that, a naïve broadcast strategy; wherein each router retransmits every link state packet that it receives will result in unnecessary broadcasts and thus increase the overhead. You should implement a mechanism to reduce such unnecessary broadcasts. This can be achieved in several ways. You are open to choose any method. You must describe your method in the written report.

3.2 Dealing with failure of Routers:

You must ensure that your algorithm is robust to router failures (We assume that only routers will fail while in real scenarios links may also fail). Once a router fails, its neighbors must quickly be able to detect this and the corresponding links to this failed routers must be removed. Further, the routing protocol should converge and the failed routers should be excluded from the least-cost path computations. The other routers should no longer compute least-cost paths to the failed ones. Furthermore, the failed routers should not be included in the least-cost paths to other routers.

A simple method to detect router failures is to use the link-state update message (sent after every second) as the *keep alive* message. It is recommended that you wait till at least 3 consequent link-state messages are not received from a neighbor before considering it to have failed. Once a router has detected that one of its neighbors has failed, it should update its link-state update packet accordingly to reflect the change in the local topology. Eventually, via the propagation of the updated link-state packets, other routers in the network will become aware that the failed router is unreachable and it will be excluded from the link-state computations (i.e. Dijkstra's algorithm).

Recall that each router will execute Dijkstra's algorithm periodically after every 30 seconds to compute the least-cost path to every other destination. It may

so happen that the updated link-state packets following a router failure may not have reached certain routers in the network before this interval expires (for example the router B fails at time 29 sec and Dijkstra's computation takes place at 30 sec). As a result, some of the routers will use the old topology information (prior to router failure) to compute the least-cost paths. Thus the output at these routers will be incorrect. This is not an error. It is just an artifact of the delay incurred in propagating the updated link-state information. To account for this, it is necessary to wait for at least 1 minute after the router failure is initiated. This will ensure that all routers are aware of the topology change.

You need to consider the case when a failed node again joins back the topology and starts sending link state update messages. Note that we are not considering the case when a previously unknown node joins the topology.

3.3 Multi-threading

You must use multi-threading in your implementation. We recommend that you use at least three separate threads for listening (for receiving LSA's), sending (for sending LSA's after every 1 sec) and Dijkstra's calculations (after every 30 sec). You may choose to use more than three threads as per your requirement.

3.4 Simulated network

Since we do not have access to real network routers (for implementation and testing of your programs), we will use a simulated network that will run on a single desktop machine. You will run different instances of your program on the same machine (use 'localhost'), however each instance of the routing protocol (corresponding to each router in the network) will be listening on a different port number. You can open different terminal windows (one each for every router in the network topology) to observe the behavior of your implementation. The topology size is restricted to maximum of 10 routers.

If your routing implementation executes correctly on a single desktop machine, it should also work correctly on real network routers.

3.5 Test Topology

You have been provided with configuration files for a sample topology of 6 routers. Use this topology to incrementally test your implementation. The correct output for Router A before and after failure of Router D is given. You should check your implementation for correctness at all routers with multiple router failures. Your implementation would be graded with a "Mystery" topology of maximum 10 routers with multiple router failures.

You are allowed to post new topologies on Piazza (config files) along with output for different routers so that other students can benefit from testing with different topologies.

Here is a useful link that can generate correct answers for various topologies.

https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

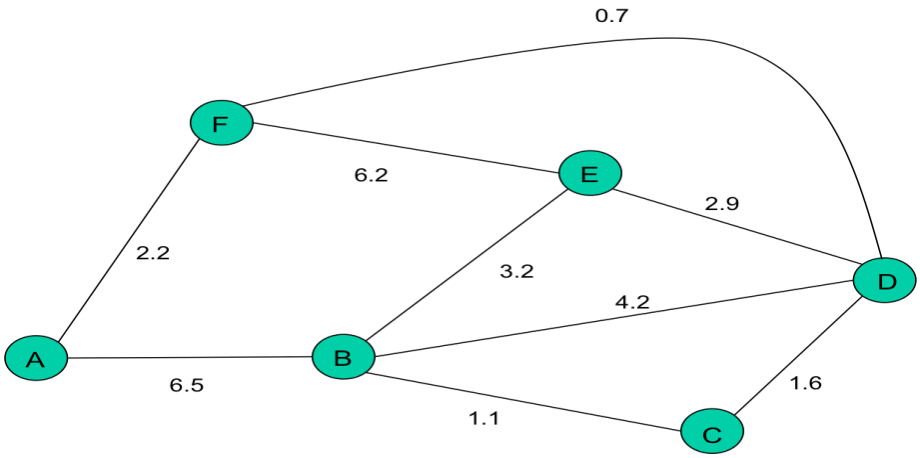

Fig:1 Sample 6 Routers topology

```
Out put with all routers working:

I am Router A
Least cost path to router C:AFDC and the cost: 4.5
Least cost path to router B:AFDCB and the cost: 5.6
Least cost path to router E:AFDE and the cost: 5.8
Least cost path to router D:AFD and the cost: 2.9
Least cost path to router F:AF and the cost: 2.2


Out put after Router D fails:

I am Router A
Least cost path to router C:ABC and the cost: 7.6
Least cost path to router B:AB and the cost: 6.5
Least cost path to router E:AFE and the cost: 8.4
Least cost path to router F:AF and the cost: 2.2
```

3.5 Submission & Report:

You are required to submit your source code and a short report to an upload link that would be made available on the LMS. Zip your source code files and your report document in a file named Project_Surname_Surname (Surname of both Group members). Nominate one of the group members to submit on behalf of the group. Please note that you must upload your submission BEFORE the deadline. The LMS would continue accepting submissions after the due date. Late submissions would carry penalty per day with maximum of 2 days late submission allowed (see Section 3.6). Students who fail to submit would not be allowed to appear in viva and those who miss the viva would not be allowed to retake the viva.

Your submitted code would be checked for similarities and any instances of plagiarism would result in award of ZERO marks for all such students, irrespective of who has shared with whom. No exceptions!!

You must write down group members' Registration No's and Names at the beginning of the report **!!** All reports will be read for marking purposes.

The size of your report **MUST be under 2 pages**. Your report should briefly document your techniques and methodology used for implementation and how you combat the relevant problems in development. Treat it as a summary document only (point form is acceptable). It should act a reference for your instructor 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. You will be asked to demonstrate your program during your viva.

3.6 Late Submission Penalty:

Late penalty (on your received marks) will be applied as follows:

1 day after deadline: 25% reduction

2 days after deadline: 50% reduction

3 days after deadline: Not accepted