**CHAPTER-4**

The basic concept of link-state routing is that every node creates a map of the connectivity to the network in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the best logical path from it to every possible destination in the network. The collection of best paths will create a shortest path tree, which then forms the node’s routing table. Link state routing is based on the assumption that each node has partial knowledge: it knows the state (cost) of its links. In other words, the whole topology can be compiled from the partial knowledge of each node. In this way, it allows the topology to be dynamic. In link state routing, four different tasks are required to ensure that each node has the routing table showing the least-cost node to every other node.

1. Creation of the states of the links by each node, called the link state packet or LSP.

2. Dissemination of LSP data to every other router, called ﬂooding.

3. Formation of a shortest path tree for each node.

4. Calculation of a routing table based on the shortest path tree.

* **Creation of Link State Packet :** A link state packet (LSP) can carry a large amount of information. For this project, we consider that it carries a minimum amount of data: the node identity and the cost of links. Both of these details are needed to make the

topology. LSPs are generated either on a periodic basis or when there is a change in the topology of the domain.Also called shortest path first (SPF) forwarding.Named after Dijkstra’s algorithm (1959) which it uses to compute routes.All routers have tables which contain a representation of the entire network topology.In the form of lists of routers and information about each router’s neighbours and the connection between the two

* Each router creates a *link state packet* (LSP) which contains names (e.g. network addresses) and cost to each of its neighbours
* The LSP is transmitted to *all* other routers, who each update their own records
* When a routers receives LSPs from all routers, it can use (collectively) that information to make topology-level decisions
* **Flooding of LSPs:**After a node has prepared an LSP, it must be sent to all other nodes. This process is called ﬂooding. The steps are as the following:

1. The creating node sends a copy of the LSP to each neighbour.

2. A node that receives an LSP compares it with the data it may already have. If the newly arrived LSP is same as the old one it has, it discards the LSP. If it is newer, then it updates the data ,and sends a copy of it to each neighbour except the one it came from.

* **Formation of Shortest Path Tree** : After receiving all LSPs, each node will have a copy of the whole topology. Dijkstra’s Algorithm ﬁnds the shortest path to every node using this topology data. The algorithm is explained in detail in next section.
* **Calculation of Routing Table from Shortest Path:**Tree Each node uses the shortest path tree found by the Dijkstra Algorithm to construct its routing table. The routing table contains the cost of reaching each node from the root.

4

**CHAPTER-5**

**ALGORITHM**

**DIJKSTRA ALGORITHM:**

The Dijkstra algorithm is used to create a shortest path tree from a given graph. A shortest path tree is a tree in which the path between 2

the root and every other node is the shortest. The algorithm uses the following steps:

**1. Initialization** : Select the node as the root of the tree and add it to the path. Set the shortest distances for all the rootâ˘ A´Zs neighbors to the cost between the root and those neighbors. Set the shortest distance of the root to zero.

**2. Iteration :** Repeat the following two steps until all nodes are added to the path:

(a) Adding the next node to the path: Search the nodes not in the path. Select the one with minimum shortest distance and add it to the path.

(b) Updating: Update the shortest distance for all remaining nodes using the shortest distance of the node just moved to the path .

**ALGORITHM IMPLEMENTATION:**

The Dijkstra Algorithm to ﬁnd the shortest path to from a source router to a destination router is implemented in Python for this project. The program follows below steps : • The program ﬁrst asks for a network topology ﬁle. It validates the data and store in matrix format. • The next step is to create the connection table. The program takes the source router as input, and performs the Dijkstra Algorithm on it as explained in previous section. At every step, it keeps track of two type of nodes :

1. The interface used to go to next router. (For connection table.) 2. The parent node of last added node. (To create the ﬁnal path.) • Once both connection table and parent table are ready, the shortest path can be found from given source to destination by one of the two ways :

1. Starting from the source node, follow the interface from the connection table to reach to the destination.

2. Starting from the destination node, follow the parent node from the parent table to reach to the source, and provide the reverse path.

• In both the cases, the path and total cost is found and returned to the user.

• If there is no path from given source and destination, the program returns with such message.

5

**CHAPTER-6**

**SOURCE CODE:**

**#!/usr/bin/env python**

**import csv**

**import sys**

**import os**

**router\_matrix = []**

**matrix\_set = 0**

**nodes = []**

**distances = {}**

**unvisited = {}**

**previous = {}**

**visited = {}**

**interface = {}**

**path = []**

**start = 0**

**end = 0**

**# Function to print the choices when program starts.**

**def print\_choices():**

**print ("######################################################")**

**print (" Link State Routing Simulator\n")**

**print ("(1) Input Network Topology File")**

**print ("(2) Build a Connection Table")**

**print ("(3) Shortest Path to Destination Router")**

**print ("(4) Exit")**

**print ("\n######################################################\n")**

**pass**

**# Function to check if entered command is valid or not - i.e. :**

**# 1: Should be a digit.**

**# 2: Should be from the range of given choices.**

**def check\_choices(command):**

**if not command.isdigit():**

**print ("Please enter a number as command from given choices..")**

**return -1**

**else:**

**command = int(command) 6**

**if command > 4 or command < 1 :**

**print ("Please enter a valid command from given choices..")**

**return -1**

**else:**

**return command**

**# Function to process the given input file.**

**def process\_file(fname):**

**global matrix\_set**

**global router\_matrix**

**matrix\_set = 0**

**router\_matrix = []**

**with open(fname) as f:**

**router\_matrix=[list(map(int,x.split(" "))) for x in f] # Data from input file is stored in a two dimensional list(array).**

**matrix\_set = 1**

**print ("\nReview original topology matrix:\n")**

**for line in router\_matrix :**

**for item in line :**

**print (item),**

**print**

**print**

**set\_distances(router\_matrix) # Distances are stored in a dictionary - key,value pair - with source router as key and distances in form of a dictionary as value.**

**# Function to store the distances in dictionary format.**

**def set\_distances(router\_matrix):**

**global distances**

**global nodes**

**distances = {}**

**nodes = []**

**7**

**num\_nodes = len(router\_matrix)**

**for i in range(num\_nodes):**

**tempdict = {}**

**for j in range(num\_nodes):**

**if i!=j and router\_matrix[i][j]!=-1:**

**tempdict[j+1] = router\_matrix[i][j]**

**distances[i+1] = tempdict**

**nodes.append(i+1)**

**def dijkstra(start): global distances**

**global nodes**

**global unvisited**

**global previous**

**global visited**

**global interface**

**# set the values to none for initialization.**

**unvisited = {node: None for node in nodes}**

**previous = {node: None for node in nodes}**

**interface = {node: None for node in nodes}**

**visited = {node: None for node in nodes}**

**current = int(start)**

**currentDist = 0**

**unvisited[current] = currentDist**

**while True:**

**for next, distance in distances[current].items():**

**if next not in unvisited: continue**

**newDist = currentDist + distance**

**if not unvisited[next] or unvisited[next] > newDist:**

**unvisited[next] = newDist**

**previous[next] = current**

**if not interface[current]:**

**interface[next] = next**

**else:**

**interface[next] = interface[current]**

**visited[current] = currentDist**

**del unvisited[current] 8**

**done = 1**

**for x in unvisited:**

**if unvisited[x]:**

**done = 0**

**break**

**if not unvisited or done:**

**break**

**elements = [node for node in unvisited.items() if node[1]]**

**current, currentDist = sorted(elements, key = lambda x: x[1])[0]**

**# Function to generate the shortest path using the parent table generated by function dijkstra.**

**def shortest\_path(start, end):**

**global path**

**path = []**

**dest = int(end)**

**src = int(start)**

**path.append(dest)**

**while dest != src:**

**path.append(previous[dest])**

**dest = previous[dest]**

**path.reverse()**

**print\_choices()**

**command = 0**

**# Run till user wants to exit.**

**while command !=4 :**

**command = check\_choices(input("\nCommand : "))**

**# Accept the topology file.**

**9**

**if command == 1:**

**if matrix\_set == 1:**

**answer = raw\_input("\nThe network topology is already uploaded. Do you want to overwrite? (Y/N) :")**

**if matrix\_set == 0 or answer == 'Y' or answer == 'y':**

**filename = input("\nInput original network topology matrix data file[ NxN distance matrix. (value : -1 for no link, 0 for self loop) : ")**

**if os.path.isfile(filename):**

**process\_file(filename)**

**start = 0**

**end = 0**

**else:**

**print ("\nThe file does not exist. Please try again..")**

**# Accept the source router and display the connection table.**

**elif command == 2:**

**if matrix\_set == 1 :**

**start = input("\nSelect a source router : ")**

**if start.isdigit() and int(start) > 0 and int(start) <= len(router\_matrix):**

**dijkstra(start)**

**print ("\nDestination\tInterface")**

**for key in interface:**

**print (key,"\t\t", interface[key])**

**else:**

**start = 0**

**print ("\nPlease enter a valid source router.")**

**else:**

**print ("\nNo network topology matrix exist. Please upload the data file first.. ")**

**# Accept the destination router and display the shortest path and cost.**

**elif command == 3:**

**if matrix\_set == 1 :**

**end = input("\nSelect a destination router : ")**

**10**

**if end.isdigit() and int(end) > 0 and int(end) <= len(router\_matrix):**

**if int(start) == 0:**

**print ("\nNo source router selected yet. Please select a source router using choice : 2.")**

**elif int(start) == int(end):**

**print ("\nSource and Destination routers are same. Please select a different destination router.")**

**elif not previous[int(end)] :**

**print ("\nThere does not exist any route from Source : %s to Destination : %s. \nPlease select a different destination router. " %(start, end))**

**else:**

**shortest\_path(start,end)**

**print ("\nThe shortest path from router %s to router %s : " %(start, end)),**

**for item in path:**

**print (str(item) + ' '),**

**print ('')**

**cost = 0**

**if visited[int(end)]:**

**cost = visited[int(end)]**

**print ("\nThe total cost is : ", cost)**

**else:**

**print ("\nPlease enter a valid destination router.")**

**pass**

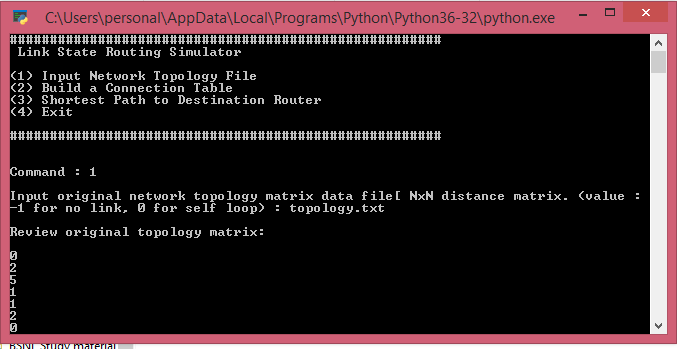
**else :**

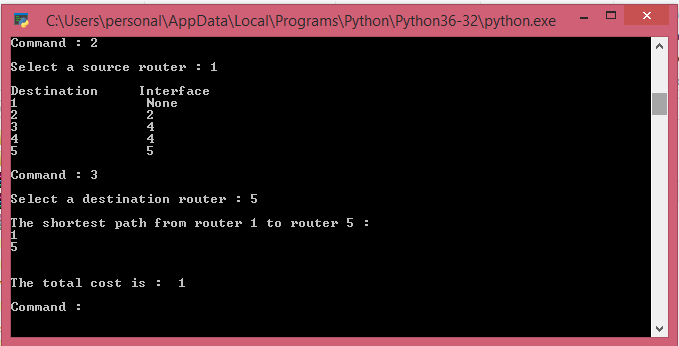
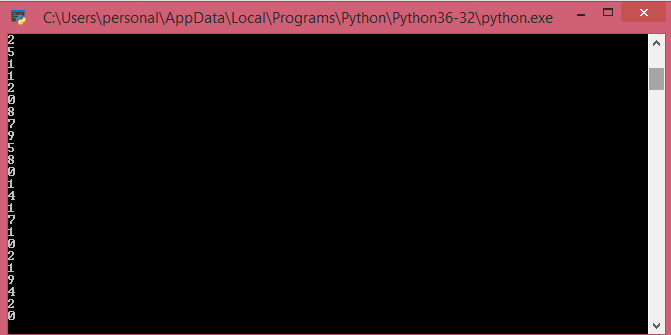
**print ("\nNo network topology matrix exist. Please upload the data file first.. ")**

**#Exit if command is 4.**

**print ("\nExit CN project. Good Bye!\n")**

**11**

**OUTPUTS:** ****

****

**CONCLUSION:**

The implented program for Link State Algorithm works for any network topology regardless of the size of network. With every node having partial information about the network topology, it can create shortest path tree for the network. Given valid topology data, it will provide you with the shortest path between source router and destination router.