Lab Assignment on Unit IV

Aim: Write a program to simulate the behavior of link state routing protocol to find suitable path for transmission.

Requirements: Fedora 20 with Pentium IV and above, 1 GB RAM, 120 G.B HDD, Monitor, Keyboard, Mouse , Modelio, Eclipse, CDT, Python interpreter, Pydev, J2SE

Theory:

1. Link State Routing Protocol

Link-state routing protocols are one of the two main classes of routing protocols used in packet switching networks for computer communications, the other being distance-vector routing protocols. Examples of link-state routing protocols include Open Shortest Path First (OSPF) and intermediate system to intermediate system (IS-IS).

The link-state protocol is performed by every *switching node* in the network (i.e., nodes that are prepared to forward packets; in the Internet, these are called routers). The basic concept of link-state routing is that every node constructs 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 next best logical *path* from it to every possible destination in the network. Each collection of best paths will then form each node's routing table.

2. Calculating the shortest paths

Each node independently runs an algorithm over the map to determine the shortest path from itself to every other node in the network; generally some variant of Dijkstra's algorithmis used. This is based around a link cost across each path which includes available bandwidth among other things.

A node maintains two data structures: a tree containing nodes which are "done", and a list of *candidates*. The algorithm starts with both structures empty; it then adds to the first one the node itself. The variant of a Greedy Algorithm then repetitively does the following:

- All neighbour nodes which are directly connected to the node are just added to the tree (excepting any nodes which are already in either the tree or the candidate list). The rest are added to the second (candidate) list.

- Each node in the candidate list is compared to each of the nodes already in the tree. The candidate node which is closest to any of the nodes already in the tree is itself moved into the tree and attached to the appropriate neighbor node. When a node is moved from the candidate list into the tree, it is removed from the candidate list and is not considered in subsequent iterations of the algorithm.

The above two steps are repeated as long as there are any nodes left in the candidate list. (When there are none, all the nodes in the network will have been added to the tree.) This procedure ends with the tree containing all the nodes in the network, with the node on which the algorithm is running as the *root* of the tree. The shortest path from that node to any other node is indicated by the

list of nodes one traverses to get from the root of the tree, to the desired node in the tree.

3. Filling the routing table

With the shortest paths in hand, the next step is to fill in the routing table. For any given destination node, the best path for that destination is the node which is the first step from the root node, down the branch in the shortest-path tree which leads toward the desired destination node. To create the routing table, it is only necessary to walk the tree, remembering the identity of the node at the head of each branch, and filling in the routing table entry for each node one comes across with that identity.

Algorithm

Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a *SPT (shortest path tree)* with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has minimum distance from source.

1) Create a set *sptSet* (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3) While *sptSet* doesn't include all vertices

- **a)** Pick a vertex u which is not there in *sptSet*and has minimum distance value.
- **b)** Include u to *sptSet*.
- **c)** Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Outcome:

The shortest path is calculated to transfer the data from source to destination using the link state routing protocol.