INTERNET PROTOCOLS PROJECT 3

Link-State and Distance-Vector Routing Algorithms

BHARATH VENKATESH
Student ID: 001037220
SRIDEVI JANTLI
Student ID: 200019504

Objective:

- Implementing Link-State and Distance-Vector routing algorithms.
- Computing the time required by each source using the link-state routing algorithm to compute paths to every other node in the network.
- Computing the maximum number of iterations it takes for any node to update its routing table before it converges.
- Implementing Distance-Vector routing algorithm over UDP sockets.

Design:

Link-State Routing Algorithm: This is a centralized routing algorithm which uses the Dijkstra's single source shortest path computation algorithm over all the nodes in the network as follows:

- Initializing the routing table with the neighbor nodes and the cost to reach them.
- Run through the Dijkstra's algorithm once for each node.
- For each node, the Dijkstra's algorithm iterates through each of its neighbor's routing table to find the path to other nodes that are reachable through this neighbor node using Dijkstra's algorithm.
- For each node, the cost will be updated if the existing cost to reach a particular node is greater than the one which is computed to route through a different set of nodes.
- The algorithm finally maintains a centralized table of shortest path to reach from every node to every other node.

Distance-Vector Routing Algorithm: This is a distributed routing algorithm which uses Bellman Ford single source shortest path computation algorithm over all the nodes as follows:

- Initialize the routing tables at every node with its neighbor nodes and the costs to reach them.
- Update the routing table to all the neighbor nodes of some initial node.
- Each of these neighbor nodes when receives the updated routing table, computes the shortest path to reach the nodes that are now reachable through this neighbor nodes using Bellman Ford equation.
- This new updated routing table is sent to all the nodes except from the one we received the update.
- The nodes that receives this update, now re-computes its routing table and the process of updating the routing table and sending updated table to neighbors repeats until all the nodes converge i.e. until there is no change in the routing tables of any of the nodes in the network.

Implementation:

Programming Language: Java 1.6 [Using open-jdk]

Link-State:

- Four different classes are used to implement Link-State algorithm namely Nodes, Links, Cost, and linkstate.

- The class Nodes maintains the Node number, minimum distance to the source, parent node to reach the source and the linked list of all the links to reach its neighboring nodes.
- The class Links maintains the neighbor node and the weight to reach that node.
- The class Cost maintains the parent node to reach a host and the minimum distance to reach that source.
- The class linkstate is used to run the Dijkstra's algorithm over each node to compute the shortest paths between all pairs.
- The Dijkstra function computes the shortest path from a source to every other node.
- Each node is sent as a source to the Dijkstra function to compute shortest paths between all pairs and routing tables are printed for the required nodes.

Distance-Vector Centralized:

- Two classes are used to implement Distance-Vector algorithm namely Nodes and DistanceVector.
- The Nodes class maintains the neighbor node number and the cost to reach a particular destination node.
- The DistanceVector class is used to run the Bellman Ford algorithm over all the nodes to compute the shortest path between all pairs.
- The ReadMatrix function is used to read input graph into required data-structure which is used for computing paths.
- The routing tables of all the nodes are initialized using the DistanceVector constructor.
- The entire network topology is known to every node in this centralized approach, hence the routing table updation is handled using queues where each node that needs to update its routing table is added into the queue.
- The shortest paths between all pairs is computed using Bellman function.
- The PrintMatrix function is used to display the routing tables of the required nodes

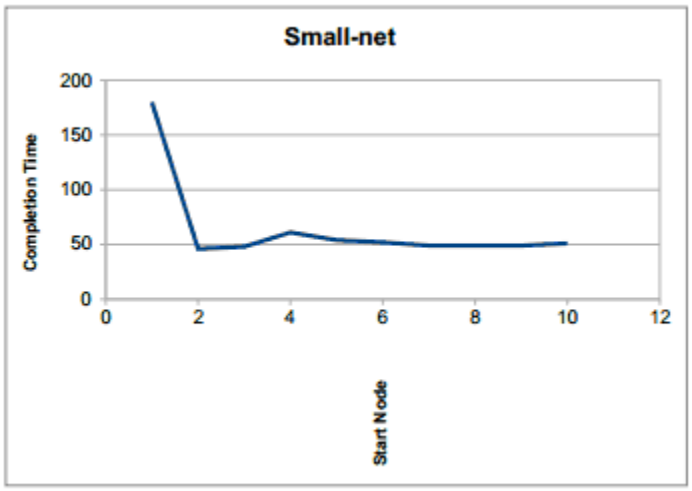
Distance-Vector Distributed (Using UDP):

- This is a multi-threaded approach with one thread handling the sending of updates to other nodes and another thread waiting to receive the updates from other nodes.
- Four classes are used namely Nodes, Server, Client, DistanceVector_UDP.
- The Nodes class maintains the neighbor node number and the cost to reach a particular destination node. This class implements Serializable for maintaining the states of the routing table.
- The Client class extends DistanceVector_UDP which send its initial routing table to all the neighbor nodes over the UDP sockets.
- The Server class extends DistanceVector_UDP. It listens to the updates sent by other nodes and calls the Bellman function to update the existing routing table. If there is any change, then it sends the updates to the neighbor nodes using the send_neighbor function over the UDP sockets.
- The class DistanceVector_UDP is the main class which initializes the states of the routing tables at each node. The ReadMatrix function reads the input file into the required data-structure used for computing the paths in Bellman function.
- The PrintTable function is used to display the routing table of the node.

Task 1: Running Time of the LS

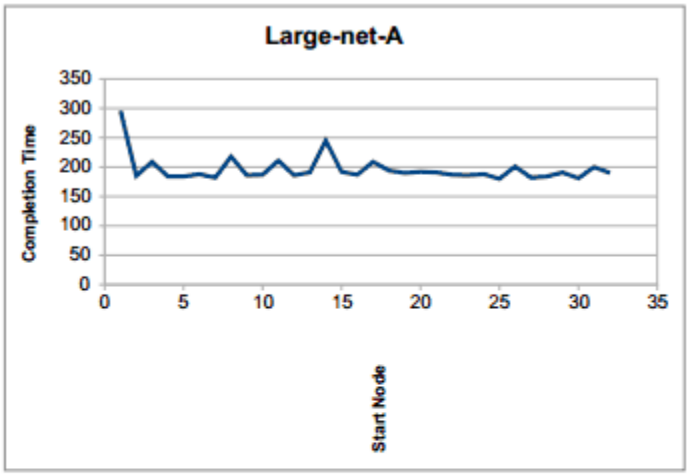
1. small-net

Start Node	Completion Time
1	180
2	46
3	48
4	61
5	54
6	52
7	49
8	49
9	49
10	51



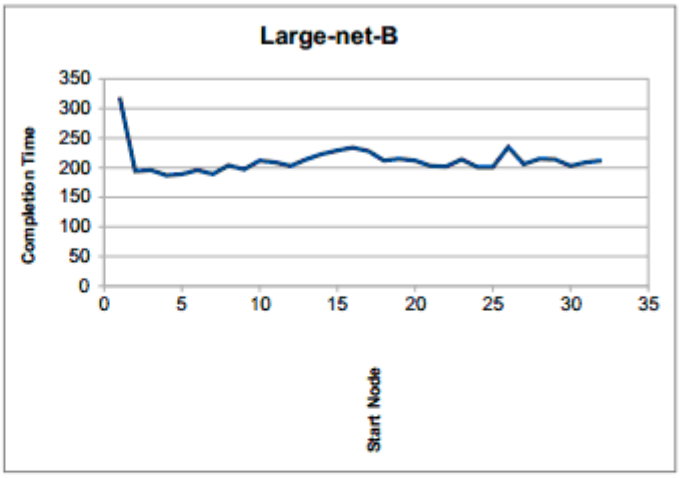
2. large-net-a

Start Node	Completion Time
1	296
2	185
3	209
4	184
5	184
6	188
7	182
8	218
9	186
10	187
11	211
12	186
13	191
14	245
15	192
16	187
17	209
18	194
19	190
20	192
21	191
22	187
23	186
24	188
25	180
26	201
27	182
28	184
29	191
30	181
31	200
32	190



3. large-net-b

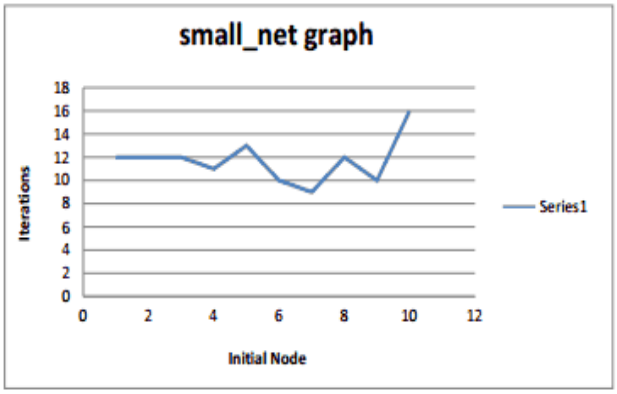
Start Node	Completion Time
1	319
2	194
3	196
4	187
5	189
6	196
7	189
8	204
9	197
10	212
11	209
12	203
13	214
14	223
15	229
16	234
17	228
18	212
19	215
20	212
21	203
22	202
23	214
24	201
25	201
26	235
27	206
28	215
29	214
30	203
31	209
32	212



Task 2: Iterations of the DV

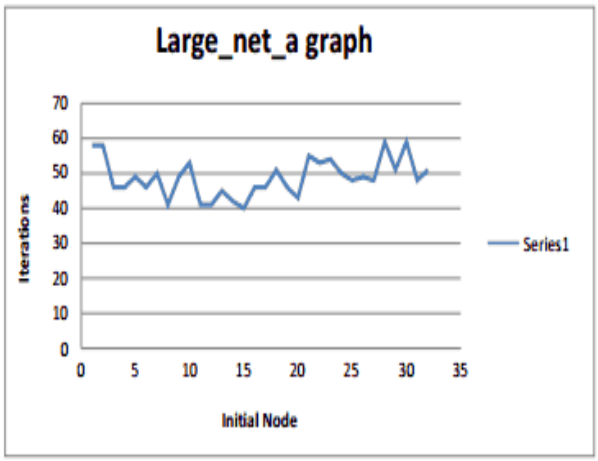
A. Small-Net

Initial Node	Iterations
1	12
2	12
3	12
4	11
5	13
6	10
7	9
8	12
9	10
10	16



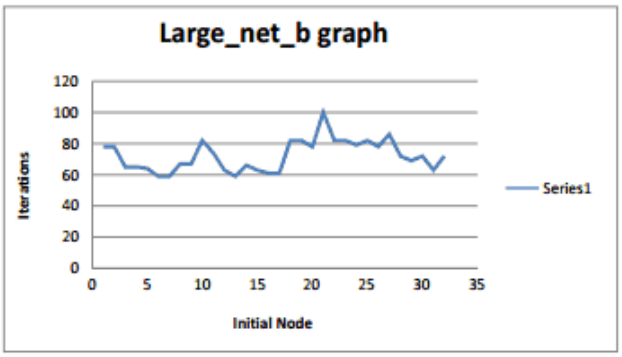
B. Large-Net-A

Initial Node	Iterations
1	58
2	58
3	46
4	46
5	49
6	46
7	50
8	41
9	49
10	53
11	41
12	41
13	45
14	42
15	40
16	46
17	46
18	51
19	46
20	43
21	55
22	53
23	54
24	50
25	48
26	49
27	48
28	59
29	51
30	59
31	48
32	51



C. Large-net-B

Node	Iterations
1	78
2	78
3	65
4	65
5	64
6	59
7	59
8	67
9	67
10	82
11	74
12	63
13	59
14	66
15	63
16	61
17	61
18	82
19	82
20	78
21	100
22	82
23	82
24	79
25	82
26	78
27	86
28	72
29	69
30	72
31	63
32	72



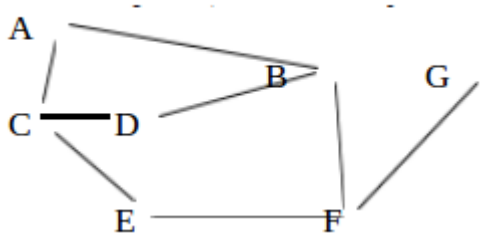
Analysis of Task1:

We see from the graphs for task1 that the running time for Link State Algorithm remains in the same range for most of the nodes barring a few outliers. This can be accounted for the fact that Djkstra is a single source shortest path algorithm wherein each node contacts every other node and based on the contacted nodes routing table, it updates its routing table. Thus, irrespective of from which node we start the algorithm, the algorithm performs the same number of runs. We can also observe that the time taken to compute the routing table increases with the increase in the number of nodes.

Analysis of Task2:

We see from the graphs above that the convergence time **does get affected by the starting node**. The reason for this is because the number of hops required to reach the farthest nodes depends on from where we start converging and the connectivity of its neighbors to other nodes in the network.

Consider a small example to demonstrate the cause.



Here if we start from Node C for instance, it takes 3 iterations to converge as it takes 3 iterations for Node 3 to reach the farthest node G.

However if we start with Node B, then in only 2 iterations Node B would have known all its neighbors thus the network converges in two iterations.

This demonstrates that the initial node does impact the number of iterations required to converge the network.