CS 542 – Computer Networks

Project Report

Submitted by,

Guru Prasad Natarajan,

A20344932

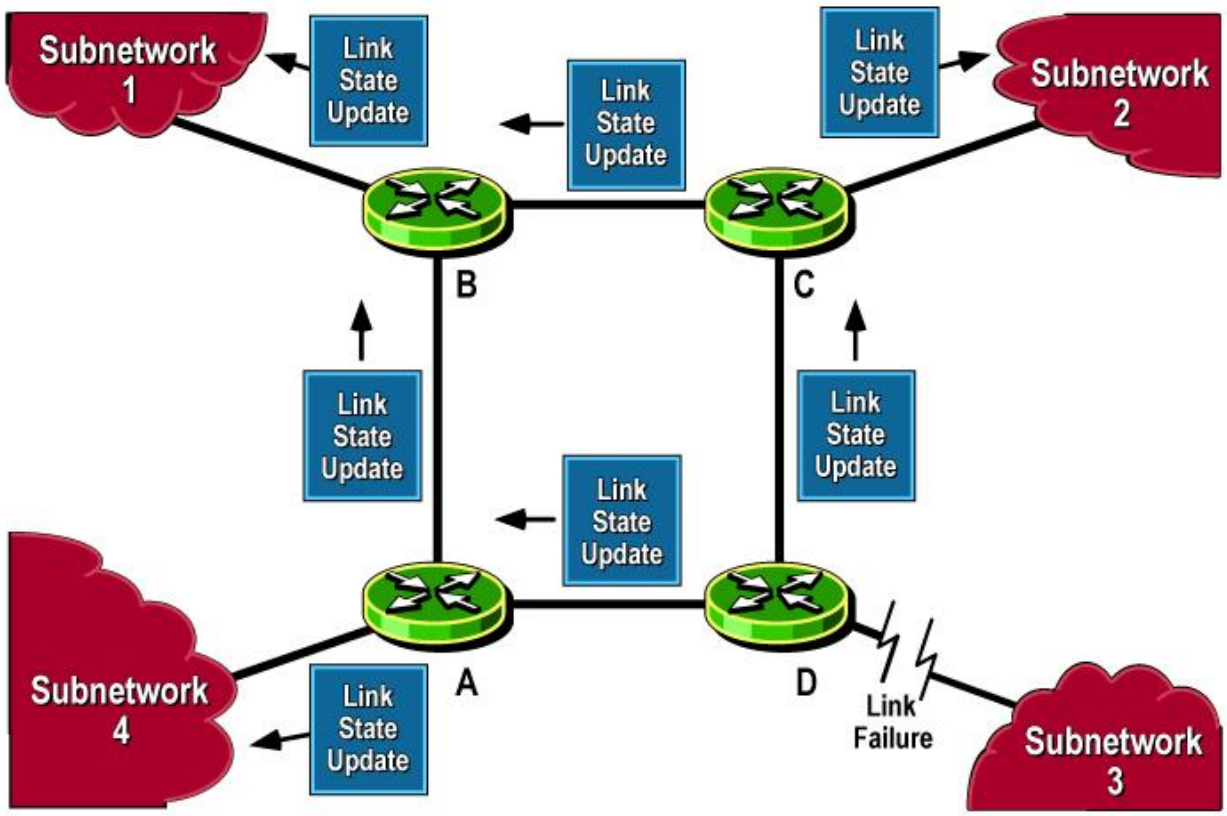
Section #03

Introduction to Link State Routing

Link state routing protocols operate by flooding link state information into the network. The link state information is stored in a link state database, sometimes called a topological database, and each router has a copy of this database.

The link state database includes information such as the number of ports on each router, the Network Layer address and current state of those ports, and the subnetworks to which those ports connect. In a very real sense, there is sufficient information in the database for the router to generate a complete map of the internetwork, to label each router with its appropriate addresses, and to label each port with its current state and cost. Then, using a shortest path first (SPF) algorithm (e.g., Dijkstra's Algorithm), each router can calculate the shortest path from itself to every known subnetwork and use the information to populate a routing table.

Link State Routing Protocols and Routing updates



The above picture depicts what happens in an internetwork implementing a link state routing protocol when a link failure occurs. The router experiencing the link failure (or any other change in the status of one of its ports) generates a link status update on each of its active ports. That link status message is flooded throughout the internetwork, resulting in an altered link state database at each router. Each router then recalculates its routes and updates its routing table. In the network depicted on the visual, all of the routers discover that they no longer have a viable route to Subnetwork 3. This subnetwork has been isolated from the rest of the internetwork and will remain so until the link failure is fixed.

This description of link state protocol operation is highly simplified. Link state routing protocols are complex and must deal with potentially complex issues. How these issues are addressed differs depending on the specific link state routing protocol being discussed. Some of the issues that must be addressed follow.

- How a newly installed router initiates itself. In order for the router to correctly populate its link state database, it must obtain a copy of the link state database from one of the active routers. At the same time, it must distribute information about itself and its ports to the rest of the routers in the network.

- Detecting the complete failure of a router. A router may be able to send a link status update on its active ports when one of its ports fails, but it is unlikely to notify the rest of the network that it is about to crash, nor can it send the updates posthumously. Somehow, the routing protocol must define a strategy by which the other routers can detect the absence of one of their peers.

- How routers ensure that their database is correct. If any router's database contains incorrect information, the routing table it builds will also be incorrect. The routers must have a strategy for ensuring, perhaps even periodically checking, the database correctness.

- How to reconnect a network that has been divided by a failure. When a network failure causes the internetwork to split into two unconnected internetworks, the link state databases in the two separated internetworks eventually lose all information related to the subnetworks they can no longer reach. When the failure is resolved and the internetwork is made whole, the two portions of the network must synchronize their databases.

- Minimizing bandwidth consumed by flooding updates in large internetworks. Although link status messages are small, as the network grows large, the number of updates that might be transmitted at any one moment grows quickly. And flooding is still an inefficient (though highly reliable) method of sending information.

- Minimizing the size of the link state database in large internetworks. A large internetwork (i.e., many routers and subnetworks) means a large link state database. A large link state database requires a significant amount of memory to store, and a significant amount of processing power to run, the calculations necessary to generate a routing table. This is especially a problem when the router is integrated into a server, which needs to focus its memory and processing resources on servicing user applications.

Advantages:

. The fact that the routers flood link state information throughout the internetwork reduces the convergence time of the network. Each router maintains its own database, which should be identical to all other databases in the network. Convergence time is now bound by the time it takes a message to travel across the network (even extremely large networks can have transmission time under a few seconds), and the interval between routing table generations. This means that the size of the internetwork contributes a negligible amount to the convergence time, making link state routing protocols suitable for very large internetworks.

The fact that the information flooded into the network usually consists of small packets with a few items of link state information, instead of full routing tables, means that a link state protocol generally uses less bandwidth than a distance vector protocol. This is true even though flooding typically implies a significant load of traffic. Most link state routing protocols, however, define a number of mechanisms to control (and minimize) the flood of information.

Disadvantages:

Link state protocols have the disadvantage of being fairly complex to implement. All routers in the network require redundant databases. Routing table calculation is also a more complex process than simply comparing routing table entries. In short, link state protocols tend to be more memory-intensive and processor-intensive than distance vector routing protocols.

While these disadvantages seem trivial in an era of high-powered microprocessors and inexpensive random access memory (RAM), they should not be underestimated. Today's networks are increasingly large, multiprotocol, and complex. In addition, servers commonly provide the routing function. In a multiprotocol router (which could be concurrently running several different routing protocols) or a server providing routing function (which also has to support services to clients), the processor and memory demands of a link state routing protocol may have a significant impact.

Another serious drawback is the complexity of many link state routing protocols. They are significantly more complex than are distance vector routing protocols. While some network administrators might be willing to wade through the complexity in order to achieve better network reliability and performance, many are not. The ones who are willing tend to be administrators of relatively large or mission critical internetworks. Smaller internetworks, however, will probably operate fine using distance vector routing protocols.

Problem Description

The objective of this project is to develop a Link State Routing simulator to find the shortest path of a network. In order to find the shortest path Dijkstra's algorithm is used.

The project is coded using python which accepts the input file which contains the n x n array to generate routing tables for the given network and finding the most optimal path between a given set of nodes.

Design

The project is designed in a menu-driven approach, so that users can read through the menu items and select his/her choice. Following are the menu items of the project:

(1) Create a Network Topology

(2) Build a Connection Table

(3) Shortest Path to Destination Router

(4) Print all connection tables

(5) Modify network topology

(6) Exit

Menu item 4 is introduced as part of the additional enhancements.

Implementation

Description of each and every menu is as follows:

Menu 1:

- Get the input file from the user. Read the contents of the file and store it in a matrix format. From the topology matrix formed, distances information of each node and its neighbors are obtained.
- If the file not a valid file, then appropriate error message is thrown to the user.

Menu 2:

- Get the source router from the user. If the source is valid, then for the corresponding router, connection table is formed.
- Connection table is formed using Dijkstra's algorithm on the previously formed distances information.

Menu 3:

- This menu gets the destination router from the user and finds the shortest distance between the previously supplied source and the destination and prints it.

Menu 4:

- Prints the connection tables of all the nodes. This is implemented as part of the additional enhancements.

Menu 5:

- Input the router to be removed. If the router is valid, the topology matrix is updated such that the connections for the particular is set to -1, which means the particular node does not have any link with the other nodes.
- With the newly formed topology matrix, form the connection table and if the destination is provided find the shortest distance between the given source and the destination in the absence of the node removed.

Menu 6:

- Exit from the program.

Implementation of the program

Sample Input:

```
 0  -1   5   1  -1
-1   0  -1   7   9
 5  -1   0  -1   4
 1   7  -1   0   2
-1   9   4   2   0
```

Step 1:

- Print the menu items using the "printmenu()" function.
- Read the topology matrix line by line and store it in a numpy array.
- Store the distance information in a separate dictionary. This dictionary would contain only the information of the neighbors distances and an arraylist is used to store the node values.
- The distances dictionary looks like below for the above input,

  {1: {3: 5, 4: 1}, 2: {4: 7, 5: 9}, 3: {1: 5, 5: 4}, 4: {1: 1, 2: 7, 5: 2}, 5: {2: 9, 3: 4, 4: 2}}

- Once the above steps are done, the topology matrix is printed.
- Proper error handling mechanisms are used to capture error scenarios such as file not found or invalid file name.
-

Step 2:

- Get the source router from the user. If the source valid, that is if the input supplied is with the range of the total number of routers proceed with the program. Else appropriate error message is thrown to the user to re-input the source router.
- With the source as the root node, Dijkstras algorithm is used to build the connection table.
- The logic to build the connection table is as follows:

  - Source node is selected as the root, and then it finds the shortest path of its neighbors adding it to the seen dictionary.

  - Again with the new node with the shortest path as the root, find its neighbors having the shortest distance.

  - Then it searches for the nodes which are unseen and selects the one with the shortest and add it to the current distance.

  - It keeps on updating the shortest distance until all the nodes are seen.

  - During the whole process, the program keeps track of two important information, one is the intermediate node which is used to go to the next router stored in intermediate dictionary and predecessors node which keeps track of the last added node.

Step 3:

- Once the destination is provided, and since the source router information is available from the previous step, shortest distance between the source and the destination can be calculated.

- The process begins with back tracking from the destination to the source. If the predecessor information is not available then the destination cannot be reached from the given source.
- Else track the path and add it to an arraylist and reverse it to get the actual path from the source to the destination.
- The cost of all nodes can calculated by adding up all the costs in the visited dictionary.

Step 4:

- Node details such as node id is obtained in Step 2. In order to print all the connection table, iterate through each node id stored in nodes arraylist.
- With each node as the source, call the Dijkstra function to build the connection table.

Step 5:

- Once the node to be removed is obtained, repeat step 1 to obtain the modified topology matrix.
- The above step calls “modifytopology()” function with the router to be removed and copy of the original topology matrix as parameters.
- The router to be removed is removed from the distances dictionary as well as the nodes arraylist so that it doesn’t interfere with the optimal path finding process.
- With the updated information, and source and destination supplied in the previous steps call dijkstra() function to build the connection table and “shortest_path()” function to calculate the new shortest path.

Step 6:

- Terminate the program.

Pseudocode for Dijkstra’s

1 Initialization:

2 N' = {u} (permanent node)

3 for all nodes v

4 if v adjacent to u

5 then D(v) = c(u,v) 6 else D(v) = ∞

7

8 Loop

9 find w not in N' such that D(w) is a minimum

10 add w to N'

11 update D(v) for all v adjacent to w and not in N' :

12 D(v) = min(D(v), D(w) + c(w,v))

13 /* new cost to v is either old cost to v or known

14 shortest path cost to w plus cost from w to v */

15 until all nodes in N'

Running the program

Experimental Results

Input:

```
 0  -1   5   1  -1
-1   0  -1   7   9
 5  -1   0  -1   4
 1   7  -1   0   2
-1   9   4   2   0
```

CS542 Link State Routing Simulator

---

(1) Create a Network Topology

(2) Build a Connection Table

(3) Shortest Path to Destination Router

(4) Print all connection tables

(5) Modify topology

(6) Exit

Master Command : 1

Input original network topology matrix data file: input.txt

Review original topology matrix:

0 -1 5 1 -1

-1 0 -1 7 9

5 -1 0 -1 4

1 7 -1 0 2

-1 9 4 2 0

CS542 Link State Routing Simulator
---

(1) Create a Network Topology

(2) Build a Connection Table

(3) Shortest Path to Destination Router

(4) Print all connection tables

(5) Modify topology

(6) Exit

Master Command : 2

Select a source router : 1

Router 1 connection table

Destination	Interface
1	--
2	4
3	3
4	4
5	4

CS542 Link State Routing Simulator
---

(1) Create a Network Topology

(2) Build a Connection Table

(3) Shortest Path to Destination Router

(4) Print all connection tables

(5) Modify topology

(6) Exit

Master Command : 3

Select a destination router : 2

The shortest path from Router 1 to Router 2 : 1-> 4-> 2

The total cost for reaching the destination : 8

CS542 Link State Routing Simulator

---

(1) Create a Network Topology

(2) Build a Connection Table

(3) Shortest Path to Destination Router

(4) Print all connection tables

(5) Modify topology

(6) Exit

Master Command : 4

Connection table of all the nodes

Router 1 connection table

Destination	Interface
1	--

2	4
3	3
4	4
5	4

Router 2 connection table

Destination	Interface
1	4
2	--
3	4
4	4
5	5

Router 3 connection table

Destination	Interface
1	1
2	5
3	--
4	5
5	5

Router 4 connection table

Destination	Interface

1	1
2	2
3	1
4	--
5	5

Router 5 connection table

Destination	Interface
1	4
2	2
3	3
4	4
5	--

CS542 Link State Routing Simulator

---

(1) Create a Network Topology

(2) Build a Connection Table

(3) Shortest Path to Destination Router

(4) Print all connection tables

(5) Modify topology

(6) Exit

Master Command : 5

Select a router to modify: 4

Review original topology matrix:

0 -1 5 -1 -1

-1 0 -1 -1 9

5 -1 0 -1 4

-1 -1 -1 0 -1

-1 9 4 -1 0

Router 1 connection table

Destination	Interface
1	--
2	3
3	3
5	3

The shortest path from Router 1 to Router 2 : 1-> 3-> 5-> 2

The total cost is : 18

Instance 2:

Master Command : 1

Input original network topology matrix data file: input1.txt

Review original topology matrix:

0 2 -1 -1 -1 -1 6 -1

2 0 7 -1 2 -1 -1 -1

-1 7 0 3 -1 3 -1 -1

-1 -1 3 0 -1 -1 -1 2

-1 2 -1 -1 0 2 1 -1

-1 -1 3 -1 2 0 -1 2

-1 -1 -1 -1 1 -1 0 4

-1 -1 -1 2 -1 2 4 0

Master Command : 2

Select a source router : 5

Router 5 connection table

Destination	Interface
1	2
2	2
3	6
4	6
5	--
6	6

7	7
8	6

Master Command : 3

Select a destination router : 8

The shortest path from Router 5 to Router 8 : 5-> 6-> 8

The total cost for reaching the destination : 4

Instance 3:

Master Command : 1

Input original network topology matrix data file: input2.txt

Review original topology matrix:

0 4 5 1 3

4 0 -1 2 5

5 -1 0 -1 4

1 2 -1 0 2

3 5 4 2 0

Master Command : 2

Select a source router : 4

Router 4 connection table

Destination	Interface
1	1
2	2
3	1
4	--
5	5

Master Command : 3

Select a destination router : 2

The shortest path from Router 4 to Router 2 : 4-> 2

The total cost for reaching the destination : 2

Instance 4:

Master Command : 1

Input original network topology matrix data file: input3.txt

Review original topology matrix:

0 1 2 3 4

1 0 6 7 8

2 6 0 -1 -1

3 7 -1 0 2

4 8 -1 2 0

Master Command : 5

Select a router to modify: 3

Source not provided. Please select a source:1

Destination not provided. Please select a destination:2

Review original topology matrix:

0 1 -1 3 4

1 0 -1 7 8

-1 -1 0 -1 -1

3 7 -1 0 2

4 8 -1 2 0

Router 1 connection table

Destination	Interface
1	--
2	2
4	4
5	5

The shortest path from Router 1 to Router 2 : 1-> 2

The total cost is: 1

Instance 5:

Master Command : 1

Input original network topology matrix data file: input.txt

Review original topology matrix:

0 -1 5 1 -1

-1 0 -1 7 9

5 -1 0 -1 4

1 7 -1 0 2

-1 9 4 2 0

Master Command : 4

Connection table of all the nodes

Router 1 connection table

Destination	Interface
1	--
2	4
3	3
4	4
5	4

Router 2 connection table

Destination	Interface
1	4
2	--
3	4

4	4
5	5

Router 3 connection table

Destination	Interface
1	1
2	5
3	--
4	5
5	5

Router 4 connection table

Destination	Interface
1	1
2	2
3	1
4	--
5	5

Router 5 connection table

Destination	Interface
1	4
2	2

3	3
4	4
5	--

Conclusion:

Thus the implemented Link State Simulator works good for any kind of topology regardless of its size. It was able to derive the shortest path information with partial information from its neighbors. The program has several error trapping mechanisms to capture invalid scenarios and guide users with the next step.