



# UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 7 : Implementation of Link State Algorithm

**Submitted By:**

Afser Adil Olin

Roll No : AE-47

Anika Tabassum

Roll No : Rk-61

**Submitted On :**

March 23, 2023

**Submitted To :**

Dr. Md. Abdur Razzaque

Md Mahmudur Rahman

Md. Ashraful Islam

Md. Fahim Arefin

# Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>                               | <b>2</b> |
| 1.1      | Objectives . . . . .                              | 2        |
| <b>2</b> | <b>Theory</b>                                     | <b>2</b> |
| 2.1      | Routing Algorithm . . . . .                       | 2        |
| 2.2      | Link State Routing Algorithm . . . . .            | 2        |
| 2.3      | The Link-State (LS) Routing Algorithm . . . . .   | 3        |
| <b>3</b> | <b>Finding APSP</b>                               | <b>4</b> |
| 3.1      | Graph . . . . .                                   | 4        |
| 3.2      | APSP . . . . .                                    | 5        |
| 3.3      | Forwarding table . . . . .                        | 6        |
| <b>4</b> | <b>Methodology</b>                                | <b>7</b> |
| <b>5</b> | <b>Implementation</b>                             | <b>7</b> |
| 5.1      | Procedure . . . . .                               | 7        |
| 5.1.1    | Extracting information at the beginning . . . . . | 7        |
| 5.1.2    | Flooding . . . . .                                | 8        |
| 5.1.3    | Dijkstra . . . . .                                | 8        |
| 5.2      | Results . . . . .                                 | 8        |
| 5.2.1    | Experimental Result . . . . .                     | 8        |
| 5.3      | Conclusion . . . . .                              | 9        |
| <b>6</b> | <b>Experience</b>                                 | <b>9</b> |

# 1 Introduction

The Link State Algorithm is a widely used routing algorithm in computer networks. It enables routers to create a map of the entire network topology and compute the shortest path to each destination. In this lab, you will design and implement a program that simulates the Link State Algorithm, demonstrating its functionality and gaining practical experience in using the algorithm for routing purposes.

## 1.1 Objectives

- To develop an understanding of the Link State Algorithm and its applications in computer networks by implementing the algorithm in a simulated network environment.
- To accurately represent the network topology by maintaining a complete and up-to-date database of network topology information, congestion, throughput, and delay.
- To quickly converge to a stable and efficient routing configuration in response to changes in the network topology.

# 2 Theory

## 2.1 Routing Algorithm

A routing algorithm is a process or set of rules that determines the best path for data packets to travel from one network node to another in a computer network. In other words, it is a method used by routers to determine the most efficient way to forward data packets between networks. A routing algorithm is a process or set of rules that determines the best path for data packets to travel from one network node to another in a computer network. In other words, it is a method used by routers to determine the most efficient way to forward data packets between networks. Routing algorithms are essential in computer networks because they help to ensure that data packets are delivered quickly and efficiently. There are many different types of routing algorithms, each with its own strengths and weaknesses. We will be discussing Link State Algorithm (LSA).

## 2.2 Link State Routing Algorithm

Link state routing (LSR) is a routing protocol used in packet-switched networks that uses a link state database to store information about the network topology. It is one of the two main types of routing protocols, the other being distance-vector routing.

The LSR process can be divided into several phases:

1. **Initialization Phase:** The first phase is the initialization phase, where each router in the network learns about its own directly connected links. This information is then stored in the router's link state database.
2. **Flooding Phase:** The second phase is the flooding phase, where each router floods its link state information to all other routers in the network. This allows each router to learn about the entire network topology.
3. **Path calculation phase:** The third phase is the shortest path calculation phase, where each router uses the link state information to calculate the shortest path to every other router in the network. This is typically done using Dijkstra's algorithm.

4. **Route Installation Phase:** The fourth and final phase is the route installation phase, where each router installs the calculated shortest paths in its routing table. This allows the router to forward packets along the optimal path to their destination.

One of the main benefits of LSR is that it only requires routers to have knowledge of their directly connected links, as opposed to the entire network topology. This makes it well-suited for large and complex networks. Additionally, LSR is less prone to routing loops and can quickly adapt to changes in the network topology.

A graph is used to formulate routing problems. Recall that a graph  $G = (N, E)$  is a set  $N$  of nodes and a collection  $E$  of edges, where each edge is a pair of nodes from  $N$ . In the context of network-layer routing, the nodes in the graph represent routers—the points at which packet-forwarding decisions are made—and the edges connecting these nodes represent the physical links between these routers.

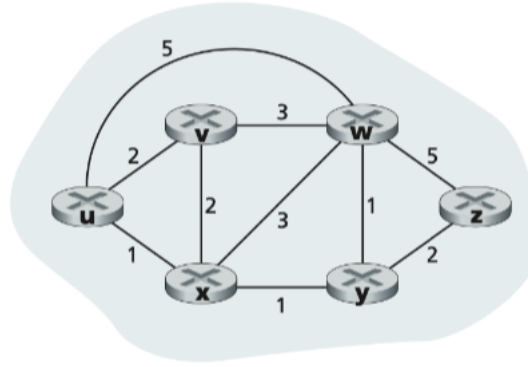


Figure 1: Initial Abstract graph model of a computer network

As shown in Figure, an edge also has a value representing its cost. Typically, an edge's cost may reflect the physical length of the corresponding link, the link speed, or the monetary cost associated with a link.

### 2.3 The Link-State (LS) Routing Algorithm

The link-state routing algorithm is known as Dijkstra's algorithm, which computes the least-cost path from one node (the source, which we will refer to as  $u$ ) to all other nodes in the network. Dijkstra's algorithm is iterative and has the property that after the  $k$ th iteration of the algorithm, the least-cost paths are known to  $k$  destination nodes, and among the least-cost paths to all destination nodes, these  $k$  paths will have the  $k$  smallest costs. Let us define the following notation:

- **$D(v)$ :** cost of the least-cost path from the source node to destination  $v$  as of this iteration of the algorithm.
- **$p(v)$ :** previous node (neighbor of  $v$ ) along the current least-cost path from the source to  $v$ .
- **$N'$ :** subset of nodes;  $v$  is in  $N'$  if the least-cost path from the source to  $v$  is definitively known.

The centralized routing algorithm consists of an initialization step followed by a loop. The number of times the loop is executed is equal to the number of nodes in the network. Upon termination, the algorithm will have calculated the shortest paths from the source node  $u$  to every other node in the network.

Listing 1: Link-State (LS) Algorithm for Source Node  $u$

```

1 Initialization:
2  $N' = \{u\}$ 
3 for all nodes  $v$ 
4     if  $v$  is a neighbor of  $u$ 
5         then  $D(v) = c(u, v)$ 
6     else  $D(v) = \infty$ 
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 each neighbor  $v$  of  $w$  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       least path cost to  $w$  plus cost from  $w$  to  $v$  */
15 until  $N' = N$ 

```

### 3 Finding APSP

#### 3.1 Graph

Graph for the Link-State (LS) Routing Algorithm:

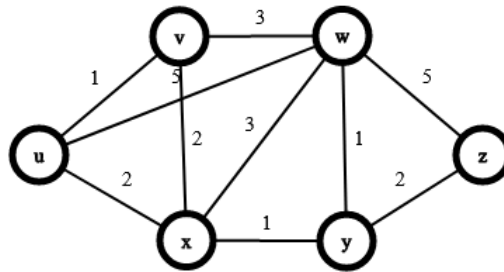


Figure 2: Initial Network Topology

### 3.2 APSP

These are the results for after running the algorithm on the given graph as shown above.

| Dijkstra's Algorithm Results for all nodes |             |               |             |
|--|-------------|---------------|-------------|
| Source                                     | Destination | Shortest Path | Path Length |
| u  | u           | {u}           | 0           |
|  | v           | {u,v}         | 2           |
|  | w           | {u,w}         | 5           |
|  | x           | {u,x}         | 1           |
|  | y           | {u,x,y}       | 2           |
|  | z           | {u,x,y,z}     | 4           |
| v  | u           | {u,v}         | 2           |
|  | v           | {v}           | 0           |
|  | w           | {v,w}         | 3           |
|  | x           | {v,x}         | 2           |
|  | y           | {v,x,y}       | 3           |
|  | z           | {v,x,y,z}     | 5           |
| w  | u           | {w,x,u}       | 4           |
|  | v           | {w,v}         | 3           |
|  | w           | {w}           | 0           |
|  | x           | {w,y,x}       | 2           |
|  | y           | {w,y}         | 1           |
|  | z           | {w,y,z}       | 3           |
| x  | u           | {x,u}         | 1           |
|  | v           | {x,v}         | 2           |
|  | w           | {x,y,w}       | 2           |
|  | x           | {x}           | 0           |
|  | y           | {x,y}         | 1           |
|  | z           | {x,y,z}       | 3           |
| y  | u           | {y,x,u}       | 2           |
|  | v           | {y,x,v}       | 3           |
|  | w           | {y,w}         | 1           |
|  | x           | {y,x}         | 1           |
|  | y           | {y}           | 0           |
|  | z           | {y,z}         | 2           |
| z  | u           | {z,y,x,u}     | 4           |
|  | v           | {z,y,x,v}     | 5           |
|  | w           | {z,y,w}       | 3           |
|  | x           | {z,y,x}       | 3           |
|  | y           | {z,y}         | 2           |
|  | z           | {z}           | 0           |

### 3.3 Forwarding table

The forwarding table simulated for this graph by our algorithm was found to be

| Forwarding table for all nodes |             |          |             |
|--------------------------------|-------------|----------|-------------|
| Source                         | Destination | Next Hop | Path Length |
| u                              | u           | none     | 0           |
|                                | v           | v        | 2           |
|                                | w           | w        | 5           |
|                                | x           | x        | 1           |
|                                | y           | x        | 2           |
|                                | z           | x        | 4           |
| v                              | u           | v        | 2           |
|                                | v           | none     | 0           |
|                                | w           | w        | 3           |
|                                | x           | x        | 2           |
|                                | y           | x        | 3           |
|                                | z           | x        | 5           |
| w                              | u           | x        | 4           |
|                                | v           | v        | 3           |
|                                | w           | none     | 0           |
|                                | x           | y        | 2           |
|                                | y           | none     | 1           |
|                                | z           | y        | 3           |
| x                              | u           | u        | 1           |
|                                | v           | v        | 2           |
|                                | w           | y        | 2           |
|                                | x           | none     | 0           |
|                                | y           | y        | 1           |
|                                | z           | y        | 3           |
| y                              | u           | x        | 2           |
|                                | v           | x        | 3           |
|                                | w           | w        | 1           |
|                                | x           | x        | 1           |
|                                | y           | none     | 0           |
|                                | z           | z        | 2           |
| z                              | u           | y        | 4           |
|                                | v           | y        | 5           |
|                                | w           | y        | 3           |
|                                | x           | y        | 3           |
|                                | y           | y        | 2           |
|                                | z           | none     | 0           |

## 4 Methodology

- Each router in the network only needs to know about its directly connected links. This allows for a more scalable solution, as routers do not need to maintain a full view of the entire network topology.
- LSR uses a link state database to store information about the network topology. This database is updated by flooding link state information between routers.
- LSR uses the shortest path algorithm, such as Dijkstra's algorithm, to calculate the shortest path to every other router in the network. This ensures that packets are always forwarded along the most efficient path.
- LSR can quickly adapt to changes in the network topology. When a link goes down or a new link is added, the link state information is updated and the shortest path is recalculated.
- Once the simulations have been run, the results must be analyzed to evaluate the performance of the implementation.
- LSR is less prone to routing loops, as each router only installs the shortest path in its routing table.
- LSR supports multiple equal-cost paths, which allows for load balancing and redundancy.

## 5 Implementation

### 5.1 Procedure

In our implementation we first take the information of their neighbours from a txt file named Path.txt. Then we start a threading for handling incoming messages from other routers called `handle_send()`. In our implementation, we updated an edge randomly and send the updated information about the graph to its neighbours after a certain time which is 30 sec. We sent the messages in different thread as sending messages is independent to receiving messages. Our implementation mainly consists of 3 part.

#### 5.1.1 Extracting information at the beginning

In `make_top()` function, we extracted the information from `file.txt`. In this file the first column consists of IP addresses of the one node (u), the second column consists of Port number of the that node(u), the third column consists of the IP addresses of the second node(v) and the fourth column consists of Port number of the second node (v). The fifth column consists of the weight between node u and v. We made sure extracting only the edge for that server by including a if statement that take and store the result if the IP address and Port number matches of the u node.

**Memory Complexity :** For storing the information of edges, we need  $O(E)$  memory .

**Time Complexity :** For extracting the information we need  $O(E)$  time complexity.



### 5.1.2 Flooding

At the beginning , we send the information of a node and it's neighbours to it's neighbour which forward the message if the ttl value is not 0 to their respective neighbours. This is called flooding. The sending message is handled by "send\_msg()" function. In the receiving part, we used "recv\_msg(conn, addr)" function. This function extracted the incoming information, then update the adjacency list according to the message. Then it run the Dijkstra algorithm and print the updated path and the parents of the nodes that can be reached from the current node u.

**Memory Complexity :** It just updated the previous list, so Auxiliary space complexity is  $O(1)$  given we already made a list of the adjacency list.

**Time Complexity :** It checks all the edges came from the message. So the time complexity will be  $O(E)$ .

### 5.1.3 Dijkstra

In this part, We just run the Dijkstra algorithm. We update the list of distance and their parent as we get a minimum distance from the current node(u).

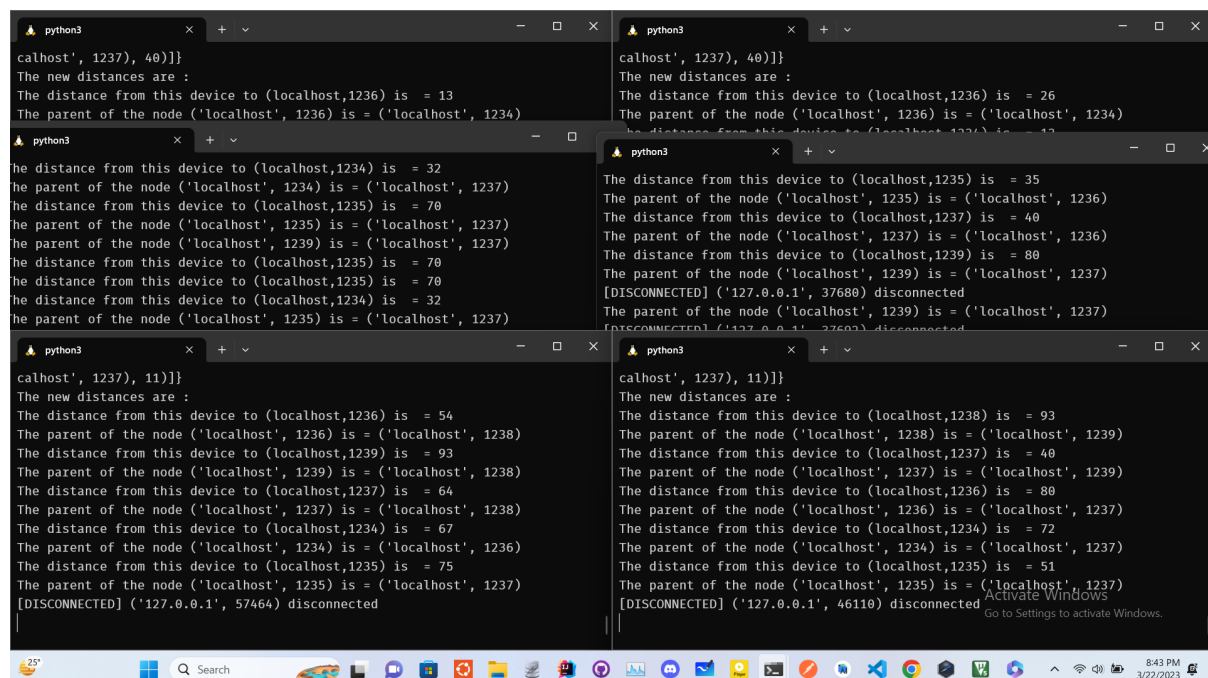
**Memory Complexity :** Auxiliary space Complexity  $O(V^2)$

**Time Complexity :** By using priority queue , we get  $O(E \log(V))$

## 5.2 Results

After updating the graph , we printed the current updated adjacency list of the graph and the distance and the parent of each node from the current node u.

### 5.2.1 Experimental Result



```
python3
calhost', 1237), 40]]
The new distances are :
The distance from this device to (localhost,1236) is = 13
The parent of the node ('localhost', 1236) is = ('localhost', 1234)

python3
The distance from this device to (localhost,1234) is = 32
The parent of the node ('localhost', 1234) is = ('localhost', 1237)
The distance from this device to (localhost,1235) is = 70
The parent of the node ('localhost', 1235) is = ('localhost', 1237)
The parent of the node ('localhost', 1239) is = ('localhost', 1237)
The distance from this device to (localhost,1235) is = 70
The distance from this device to (localhost,1235) is = 70
The distance from this device to (localhost,1234) is = 32
The parent of the node ('localhost', 1235) is = ('localhost', 1237)

python3
calhost', 1237), 11]]
The new distances are :
The distance from this device to (localhost,1236) is = 54
The parent of the node ('localhost', 1236) is = ('localhost', 1238)
The distance from this device to (localhost,1239) is = 93
The parent of the node ('localhost', 1239) is = ('localhost', 1238)
The distance from this device to (localhost,1237) is = 64
The parent of the node ('localhost', 1237) is = ('localhost', 1238)
The distance from this device to (localhost,1234) is = 67
The parent of the node ('localhost', 1234) is = ('localhost', 1236)
The distance from this device to (localhost,1235) is = 75
The parent of the node ('localhost', 1235) is = ('localhost', 1237)
[DISCONNECTED] ('127.0.0.1', 57464) disconnected

python3
calhost', 1237), 11]]
The new distances are :
The distance from this device to (localhost,1238) is = 93
The parent of the node ('localhost', 1238) is = ('localhost', 1239)
The distance from this device to (localhost,1237) is = 40
The parent of the node ('localhost', 1237) is = ('localhost', 1239)
The distance from this device to (localhost,1236) is = 80
The parent of the node ('localhost', 1236) is = ('localhost', 1237)
The distance from this device to (localhost,1234) is = 72
The parent of the node ('localhost', 1234) is = ('localhost', 1237)
The distance from this device to (localhost,1235) is = 51
The parent of the node ('localhost', 1235) is = ('localhost', 1237)
[DISCONNECTED] ('127.0.0.1', 46110) disconnected
```

Figure 3: Least cost path and Parents for nodes

### 5.3 Conclusion

In conclusion, Link State Routing (LSR) is a powerful and efficient routing algorithm used in computer networks. It uses a link state nodes to store information about the state of all the links in the network and uses Dijkstra's shortest path algorithm to determine the best path for data to travel. LSR is particularly useful in large networks where the topology changes frequently and it does not suffer from the count-to-infinity problem. However, it requires more memory and processing power than DVR and is not as scalable for very large networks.

## 6 Experience

1. We had to collect and analyze network topology information, as this requires a deep understanding of the network's routers, connections, and associated costs.
2. We had to optimize the algorithm's performance and monitoring network stability
3. We had to evaluate the data consistency mechanism of the distributed database by intentionally introducing inconsistencies into the graph.

## References

- [1] Shortest path problem between routing terminals implementation in Python. *GeeksforGeeks*, feb 27 2020. [Online; accessed 2023-03-22].
- [2] Prepbytes. Link state routing algorithm. <https://www.prepbytes.com/blog/miscellaneous/link-state-routing-algorithm/>, jan 31 2023. [Online; accessed 2023-03-22].