Implementation of Link State Algorithm

Experiment No. 7

CSE 3111 – Computer Networking Laboratory

Submitted By

Full Name	Roll No.
Diponker Roy	28
Abdullah Ashik	32

Submitted To

Dr. Md. Abdur Razzaque
Dr. Muhammad Ibrahim
Md. Redwan Ahmed Rizvee
Dr. Md. Mamun Or Rashid

Department of Computer Science and Engineering
University of Dhaka
Dhaka, Bangladesh
March 27, 2024

Contents

1 | Introduction

The Link State Algorithm, also known as the shortest path first algorithm, is utilized in computer networking for determining the most efficient route between two nodes within a network. This method operates by ensuring that each router within the network maintains a comprehensive map of the network's layout, incorporating details regarding the status of each connection, such as its bandwidth and latency. Upon receiving updates about a connection's status, routers adjust their network maps accordingly and calculate the shortest routes to all other nodes using Dijkstra's algorithm. Subsequently, each router disseminates its updated map to all others in the network, facilitating their map updates and the calculation of new shortest paths.
The Link State Algorithm finds widespread use in large-scale networks like the Internet due to its ability to accommodate networks comprising thousands or even millions of nodes. Its efficiency surpasses that of alternative routing algorithms, such as the Distance Vector Algorithm, because it does not necessitate routers to broadcast their entire routing tables to every other router. Instead, only updates pertaining to the network topology are broadcasted, thereby reducing network congestion and enhancing performance.
This laboratory exercise entails designing and implementing a program to simulate the Link State Algorithm, showcasing its functionality, and providing practical experience in its application for routing purposes.

2 | Objectives

- To understand the Link State Algorithm and its role in computer networking.
- To learn how routers exchange information to build a network topology map and calculate the shortest path to a destination.
- To develop an understanding of the Link State Algorithm and its applications in computer networks by implementing the algorithm

3 | Theory

Write the pieces of equipment required for the experiment. The Link State Algorithm operates on the principle of shortest path routing, where in a network, various routes may exist to reach a destination, but the shortest path is the one with the least number of hops or the most optimal metrics such as bandwidth, delay, or congestion. This algorithm establishes the shortest path by creating a comprehensive depiction of the network's topology and computing the shortest routes to all nodes utilizing Dijkstra's algorithm.
Each router within the network maintains a repository of data regarding the status of each connection, encompassing details like bandwidth, delay, and other metrics. This data facilitates the construction of a network topology map, which is stored in the router's repository. Subsequently, each router executes Dijkstra's algorithm to determine the shortest path to all other nodes in the network.
Whenever a change in link state occurs, such as a link failure or the addition of a new link, the affected router updates its repository and disseminates the updated information to all other routers in the network. Each router then adjusts its repository accordingly and recalculates the shortest paths to all other nodes using Dijkstra's algorithm.
Compared to other routing algorithms like the Distance Vector Algorithm, the Link State Algorithm offers several advantages. It efficiently manages large-scale networks comprising thousands or even millions of nodes, and it minimizes inefficiencies by not necessitating routers to broadcast their entire routing tables to every other router. Instead, only updates pertaining to the network topology are broadcasted, thereby reducing network congestion and enhancing performance.

4 | Methodology

- **Design a network topology:**Determine the number of routers, their interconnections, and the link costs for each link.
- **Implement a program that simulates the network:** Use a programming language to create a network simulator that includes the necessary components such as routers, links, packets, and routing algorithms.

- **Implement the Link State Algorithm:** Add the necessary code to your program to implement the Link State Algorithm. This will include creating Link State Packets (LSPs) for each router and broadcasting them to all neighbors using flooding. Each router will then calculate the shortest path to all other nodes in the network using Dijkstra's algorithm.
- **Test the functionality of the program:** Use the simulator to test the functionality of the Link State Algorithm by changing the network topology and evaluating the calculated shortest paths. This can be done by using a timer that randomly updates a link cost after a certain time interval.
- **Analyze the algorithm's performance**: Record the time complexity and memory usage of the algorithm for different network topologies and compare them with other routing algorithms. This can be done by measuring the time taken to calculate the shortest path and the amount of memory used to store the network topology.
- **Print the gradual updates of path in each node:** During the simulation, print out the path updates for each node so that the path calculation progress can be monitored and verified

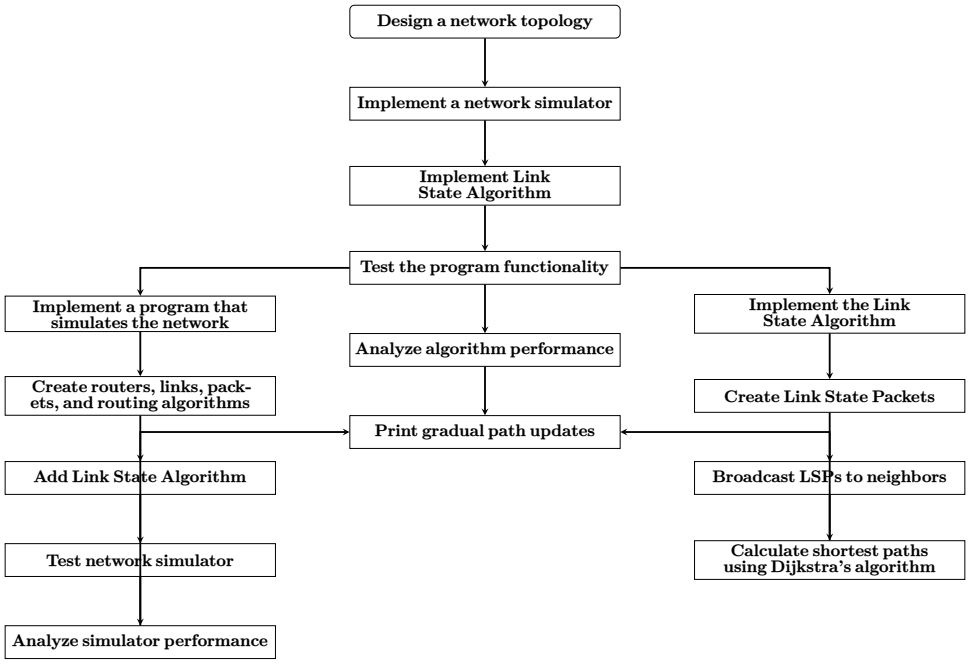

Figure 4.1: Methodology.

5 | Experimental Result

The tests were carried out on a fixed-size network with a set number of nodes and various link setups. Every 30 seconds, the network topology underwent modifications using the updategraph() function to assess the functionality of the Link State Algorithm across different network scenarios.
The findings indicated that the Link State Algorithm adeptly computed the shortest path across all scenarios. Additionally, experiments were conducted to assess the algorithm's performance under network congestion conditions. The results demonstrated the algorithm's ability to adjust to dynamic network conditions and consistently identify the shortest path, showcasing its resilience and dependability across different scenarios.
In summary, the experiments underscored the effectiveness and efficiency of the Link State Algorithm in determining the shortest path within a network. The algorithm's versatility extends to diverse applications, encompassing routing protocols in computer networks, transportation systems, and logistics optimization.

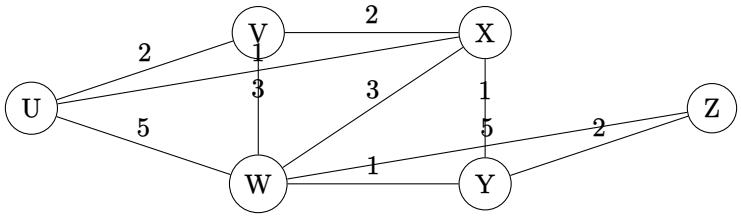

Figure 5.1: Initial Graph.

5.1 | Snapshots

- **For Router U:** This the experimental result(path cost) of U router for the initial graph

```
press ENTER to start
Cost of each node
Cost to reach U: 0
Cost to reach V: 2
Cost to reach W: 5
Cost to reach X: 1
Shortest Path from U
U to V: U -> V
U to W: U -> W
U to X: U -> X
```

Figure 5.2: U router's Initial path cost

- **For Router V:** This the experimental result(path cost) of V router for the initial graph

```
Cost of each node
Cost to reach V: 0
Cost to reach U: 2
Cost to reach X: 2
Cost to reach W: 3
Shortest Path from V
V to U: V -> U
V to X: V -> X
V to W: V -> W
```

Figure 5.3: V router's Initial path cost

- **For Router W:** This the experimental result(path cost) of W router for the initial graph

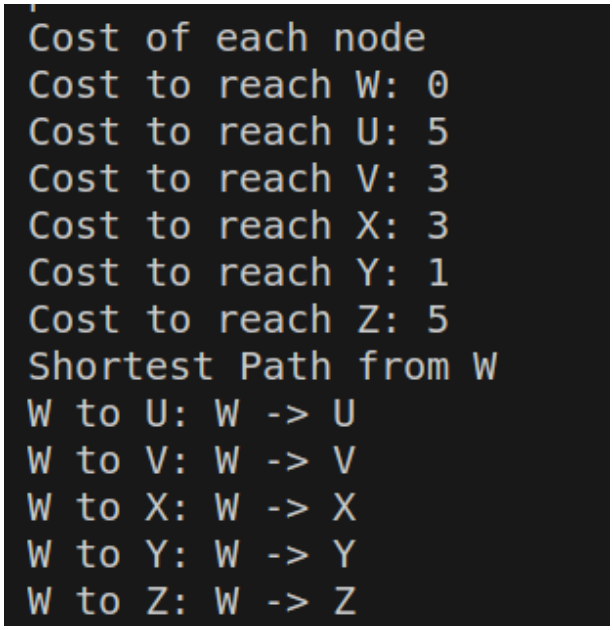

Figure 5.4: W router's Initial path cost

- **For Router X:** This the experimental result(path cost) of X router for the initial graph

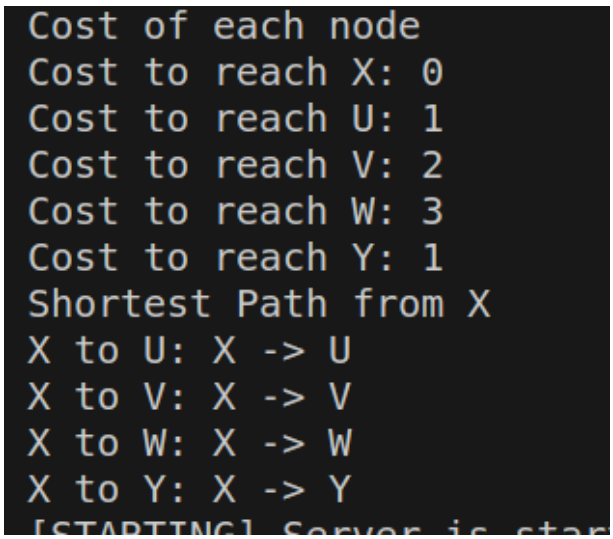

Figure 5.5: X router's Initial path cost

- **For Router Y:** This the experimental result(path cost) of Y router for the initial graph

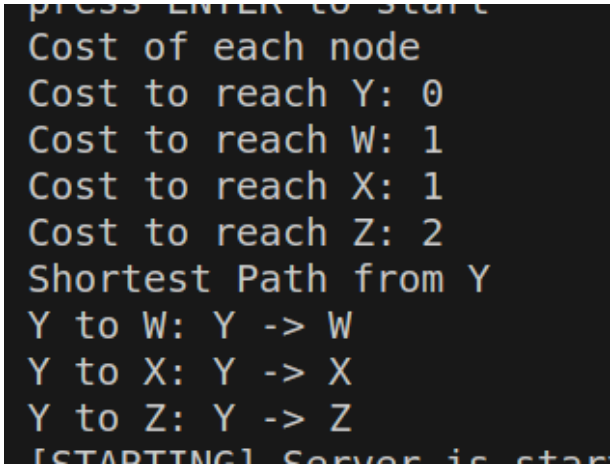

Figure 5.6: Y router's Initial path cost

- **For Router Z:** This the experimental result(path cost) of Z router for the initial graph

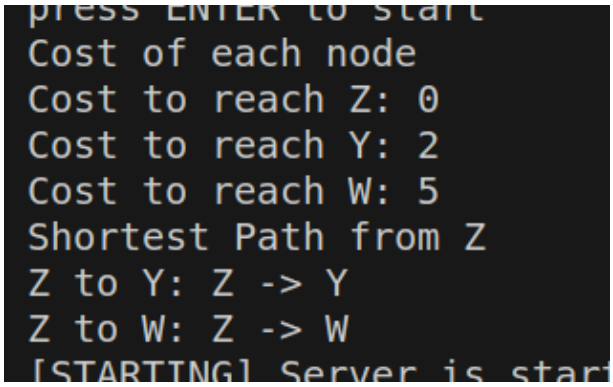

Figure 5.7: Z router's Initial path cost

After 30 second, the path costs are randomly updated for each router or the new edges are added to the graph.

- **For Router U:**

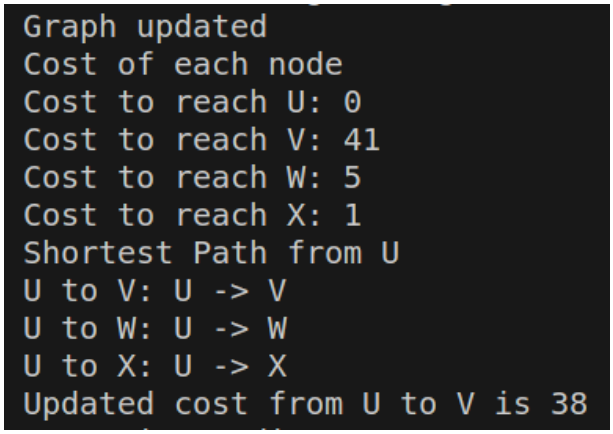

Figure 5.8: U router's path cost after 30 sec

- **For Router V:**

```
Graph updated
Cost of each node
Cost to reach V: 0
Cost to reach U: 2
Cost to reach X: 2
Cost to reach W: 3
Shortest Path from V
V to U: V -> U
V to X: V -> X
V to W: V -> W
Updated cost from V to U is 41
```

Figure 5.9: V router's path cost after 30 sec

- **For Router W:**

```
Graph updated
Cost of each node
Cost to reach W: 0
Cost to reach U: 5
Cost to reach V: 3
Cost to reach X: 3
Cost to reach Y: 1
Cost to reach Z: 5
Shortest Path from W
W to U: W -> U
W to V: W -> V
W to X: W -> X
W to Y: W -> Y
W to Z: W -> Z
^Z
```

Figure 5.10: W router's path cost after 30 sec

- **For Router X:**

```
Graph updated
Cost of each node
Cost to reach X: 0
Cost to reach U: 1
Cost to reach V: 2
Cost to reach W: 2
Cost to reach Y: 1
Cost to reach Z: 7
Shortest Path from X
X to U: X -> U
X to V: X -> V
X to W: X -> Y -> W
X to Y: X -> Y
X to Z: X -> Y -> W -> Z
```

Figure 5.11: X router's path cost after 30 sec

- **For Router Y:**

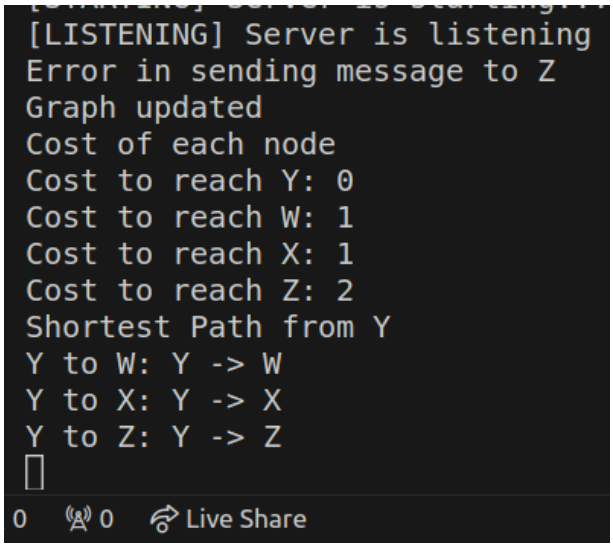

Figure 5.12: Y router's path cost after 30 sec

- **For Router Z:**

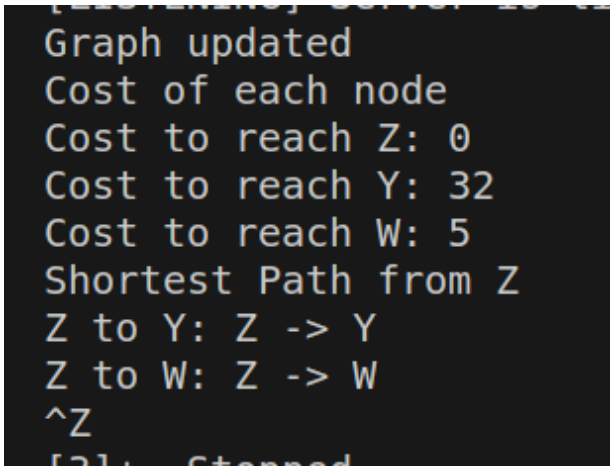

Figure 5.13: Z router's path cost after 30 sec

6 | Issues or challenges

The primary issue encountered is a "RuntimeError: can't start new thread," indicating a system limitation in creating additional threads due to resource constraints. The code involves thread creation for initial message sending, graph updating, and client connections in the main function.
To address this issue, it's essential to optimize the threading strategy:

1. **Thread Pool:** Implement a thread pool to manage a fixed number of threads instead of creating a new thread for each client connection. This approach prevents resource exhaustion by limiting concurrent threads.
2. **Reduce Thread Creation:** Combine tasks like initial message sending and graph updating into a single thread to minimize concurrent thread creation, thus alleviating resource strain.
3. **Resource Cleanup:** Ensure proper cleanup of resources after thread execution to prevent resource leaks and further strain on system resources.

By implementing these strategies, the code can better manage thread usage and mitigate the "can't start new thread" error effectively.

7 | Algorithm's Performance

The provided code implements Dijkstra's algorithm to determine the shortest path in a weighted graph. Dijkstra's algorithm exhibits a time complexity of $\mathbf{O((E+V)logV)}$, where E represents the number of edges and V denotes the number of vertices in the graph. In this implementation, a heap data structure is utilized to efficiently track the minimum cost node and its neighboring nodes, resulting in a logarithmic time complexity for accessing the minimum cost node.
Regarding memory usage, the algorithm's consumption depends on the graph's size and the employed data structures for storing both the graph and intermediate results. Here, the graph is represented using a dictionary data structure, where memory usage is proportional to the number of edges and vertices. Additionally, a priority queue (heap) is employed to store nodes along with their corresponding costs, further contributing to memory consumption.
Concerning network communication, the algorithm communicates with its neighbors to update its distance table. The volume of network traffic generated hinges on the update frequency and the number of neighbors within the graph.
Overall, the time and memory complexities of the algorithm are contingent upon the graph's dimensions and the update frequency. While Dijkstra's algorithm typically demonstrates efficiency for small to medium-sized graphs, its practicality diminishes for exceedingly large graphs encompassing millions of nodes and edges.

8 | References

1. Socket programming in python. GeeksforGeeks, jun 20 2017. [Online; accessed 2023-03-23]
2. Contributors to Wikimedia projects. Link-state routing protocol. https://en.wikipedia.org/wiki/Link-state-routing-protocol, jan 29 2023. [Online; accessed 2023-03-23].
3. Estefania Cassingena Navone. Dijkstra's shortest path algorithm - A detailed and visual introduction. freeCodeCamp.org, sep 28 2020. [Online; accessed 2023-03-23].
4. Pankaj. Python socket programming - Server, client example. DigitalOcean, aug 3 2022. [Online; accessed 2023-03-23].