



UNIVERSITY OF DHAKA

CSE:3111 COMPUTER NETWORKING LAB

Lab report 7

Implementation of the link state routing algorithm

Shariful Islam Rayhan

Roll: 41

Md Sakib Ur Rahman

Roll: 37

Submitted to:

Dr. Md. Abdur Razzaque

Dr. Muhammad Ibrahim

Dr. Md. Redwan Ahmed Rizvee

Dr. Md. Mamun Or Rashid

Submitted on: 2024-03-28

1 Introduction

In the world of computer networks, getting data from one place to another efficiently is key. An important tool for this is the Link State Routing algorithm. It is like a map for routers, helping them find the best paths through a network. Unlike some other methods, Link State Routing shares a lot of detailed information about the network, allowing routers to make smart decisions. This report will break down what Link State Routing is all about, how it works, and why it is so useful for keeping our networks running smoothly.

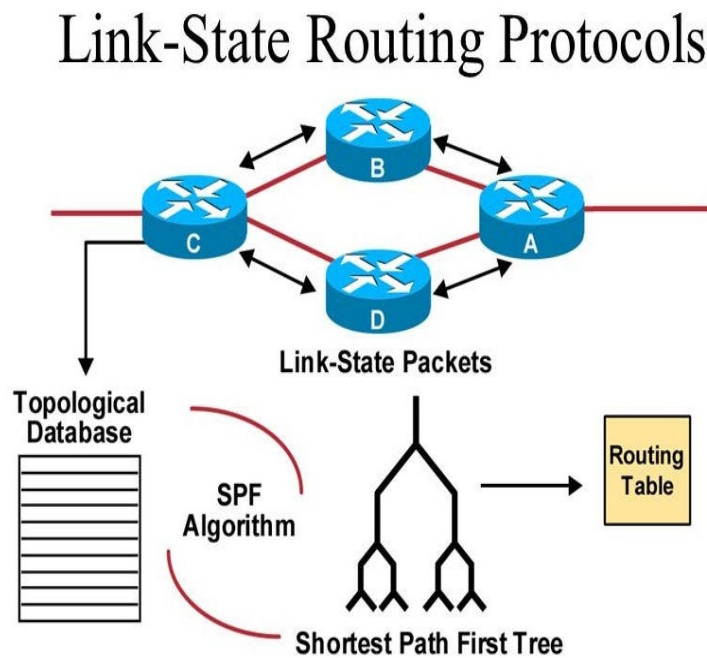


Figure 1: Link State Routing Protocols by APSP

1.1 Objectives

- Implementing the link state routing protocol.
- See the application of dijkstras algorithm
- Analyze the performance for different matrices.

2 Theory

2.1 Overview

One of the two primary types of routing protocols used in packet switching networks for computer communications is link-state routing protocol; the other type is distance-vector routing protocol. All switching nodes in the network—that is, nodes ready to forward packets; on the Internet, these are referred to as routers—perform the link-state protocol. Link-state routing works on the fundamental principle that each node creates a graph that illustrates its connection to the network, indicating which nodes are connected to which other nodes. The next optimal logical path between each node and every potential destination in the network is then separately determined by each node. The routing table for each node is then composed of each group of optimal paths.

With link-state routing protocols, every router has access to the whole network topology. The best next hop from each router is then determined separately using local topology information for each potential destination in the network. The routing table is made up of all of the best-next-hops.

2.2 Distributing the Maps to all the nodes

The first main stage in the link-state algorithm is to give a map of the network to every node. This is done with several subsidiary steps.

2.2.1 Determining the neighbours of each node

First, each node needs to determine what other ports it is connected to, over fully working links; it does this using reachability protocol it runs periodically and separately with each of its directly connected neighbours.

2.2.2 Distributing the information for the map

Next, each node periodically (and in case of connectivity changes) sends a short message, the link-state advertisement, which:

1. Identifies the node that produces it.
2. Identifies all the other nodes (either routers or networks) to which it is directly connected.
3. Includes a 'sequence number', which increases every time the source node makes up a new version of the message.

This message is sent to all nodes in a network. As a necessary precursor, each node in the network remembers, for every one of its neighbors, the sequence number of the last link state message it received from that node. When a link state advertisement is received at a node, the node looks up the sequence number it has stored for the source of that link state message: If this message is newer (that is, it has a higher sequence number), it is saved, the sequence number is updated, and a copy is sent in turn to each of the neighbors of that node. This procedure quickly receives a copy of the latest version of the link state ad for each node to every node on the network.

2.2.3 Creating the map

Finally, with the complete set of link state advertisements (one from each node in the network) in hand, each node produces the graph for the network map. The algorithm iterates over the collection of link state advertisements; for each one, it makes links on the map of the network, from the node which sent that message, to all the nodes which that message indicates are neighbors of the sending node. No link is considered to have been correctly reported unless the two ends agree; i.e., if one node reports that it is connected to another, but the other node does not report that it is connected to the first, there is a problem, and the link is not included on the map.

2.3 Calculating the routing table

The second main stage of the link state algorithm is to produce routing tables, inspecting the maps. This is again done with several steps.

2.3.1 Calculating the shortest paths

Each node autonomously executes an algorithm on the map to find the most efficient route from its location to all other nodes in the network, typically employing a version of Dijkstra's algorithm. This process revolves around assessing the link cost along each path, considering factors such as available bandwidth. A node keeps track of two data structures: a tree that includes nodes marked as "done," and a list of potential candidates. Initially, both structures are void, with the node being the first entry in the former. The modified greedy algorithm iteratively performs the following steps:

1. All neighbor nodes that are directly connected to the node are just added to the tree (except any nodes that are already in either the tree or the candidate list). The rest are added to the second (candidate) list.
2. 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 nodes left in the candidate list. (When there are nodes, all nodes in the network will have been added to the tree.) This procedure ends with the tree that contains all nodes in the network and 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.

2.3.2 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, which comes across that identity.

2.4 Dijkstra's Algorithm

Algorithm 1: Dijkstra's Algorithm

Input: Graph $G = (V, E)$, source vertex s
Result: Shortest path from s to all other vertices

```
for each vertex  $v \in V$  do
     $dist[v] \leftarrow \infty$ ;
     $visited[v] \leftarrow false$ ;
 $dist[s] \leftarrow 0$ ;
while there are unvisited vertices do
     $u \leftarrow$  vertex in  $V$  with smallest  $dist[u]$  value;
     $visited[u] \leftarrow true$ ;
    for each neighbor  $v$  of  $u$  do
         $alt \leftarrow dist[u] + \text{weight of edge } (u, v)$ ;
        if  $alt < dist[v]$  then
             $dist[v] \leftarrow alt$ ;
```

Figure 2: Pseudocode of Dijkstra's algorithm

3 Methodology

3.1 Design a network topology

Describe the network topology, including the number of routers, their inter-connections, and the link costs for each link.

3.2 Implement a program that simulates the network

Create a network simulator using a programming language such as Python. Include necessary components such as routers, links, packets, and routing algorithms.

3.3 Implement the Link State Algorithm

Enhance the program to implement the Link State Algorithm. This involves creating link state packets (LSP) for each router, broadcasting them to all neighbors via flooding, and then using the Dijkstra algorithm to calculate the shortest paths to all other nodes in the network.

3.4 Test the functionality of the program

Use the simulator to test the functionality of the link state Algorithm by altering the network topology and assessing the computed shortest paths. Introduce a timer to randomly update link costs at specific intervals.

3.5 Analyze the algorithm's performance

Record the time complexity and memory usage for various network topologies. Compare these metrics with other routing algorithms by measuring the time it takes to compute the shortest paths and the amount of memory used to store the network topology.

3.6 Print the gradual updates of paths in each node

During simulation, print the path updates for each node to monitor and verify the progress of the path calculation.

4 Experimental result

The experiments were conducted on a network of constant size with a specific number of nodes and different link configurations. Changes to the network topology were made every 30 seconds using the `updategraph()` function to evaluate the effectiveness of the Link State Algorithm in various network settings. The results showed that the algorithm successfully calculated the shortest path in all tested scenarios. Furthermore, tests were carried out to evaluate how the algorithm performed under conditions of network congestion. The outcomes illustrated the algorithm's capacity to adapt to changing network conditions and reliably determine the shortest path, highlighting its robustness and consistency across diverse scenarios.

In conclusion, the experiments highlighted the efficiency and effectiveness of the Link State Algorithm in finding the shortest path in a network. The algorithm's flexibility applies to various uses, including routing protocols in computer networks, transportation systems, and optimizing logistics.

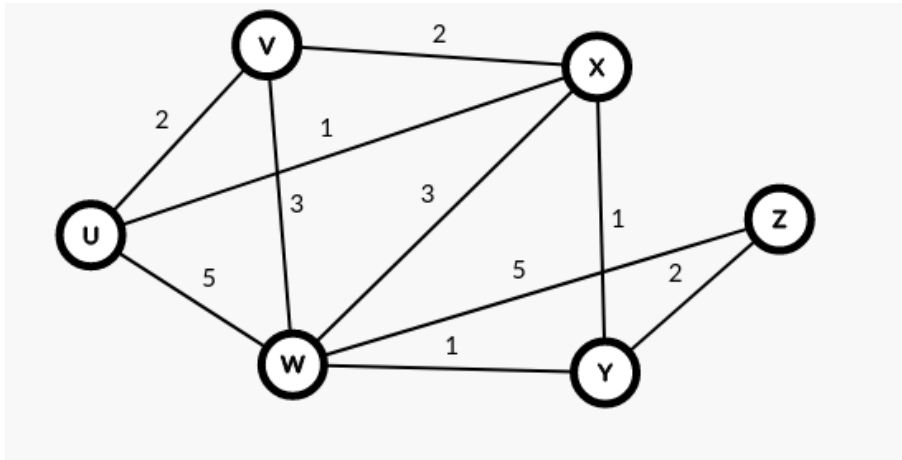


Figure 3: Initial Graph

4.1 Snapshots

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

```
(base) rayhan@rayhan-B560M-GAMING-HD:~/networking/lab 7$ python3 x.py
press ENTER to start
Cost of each node
Cost to reach X: 0
Cost to reach U: 1
Cost to reach V: 2
Cost to reach W: 3
Cost to reach Y: 1
Shortest Path from X
X to U: X -> U
X to V: X -> V
X to W: X -> W
X to Y: X -> Y
```

Figure 4: U router's Initial Path Cost

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

```
(base) rayhan@rayhan-B560M-GAMING-HD:~/networking/lab 7$ python3 v.py
press ENTER to start
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
[STARTING] Server is starting
```

Figure 5: V router's Initial Path Cost

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

```
(base) rayhan@rayhan-B560M-GAMING-HD:~/networking/lab 7$ python3 x.py
press ENTER to start
Cost of each node
Cost to reach X: 0
Cost to reach U: 1
Cost to reach V: 2
Cost to reach W: 3
Cost to reach Y: 1
Shortest Path from X
X to U: X -> U
X to V: X -> V
X to W: X -> W
X to Y: X -> Y
```

Figure 6: W router's Initial Path Cost

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

```
(base) rayhan@rayhan-B560M-GAMING-HD:~/networking/lab 7$ python3 x.py
press ENTER to start
Cost of each node
Cost to reach X: 0
Cost to reach U: 1
Cost to reach V: 2
Cost to reach W: 3
Cost to reach Y: 1
Shortest Path from X
X to U: X -> U
X to V: X -> V
X to W: X -> W
X to Y: X -> Y
```

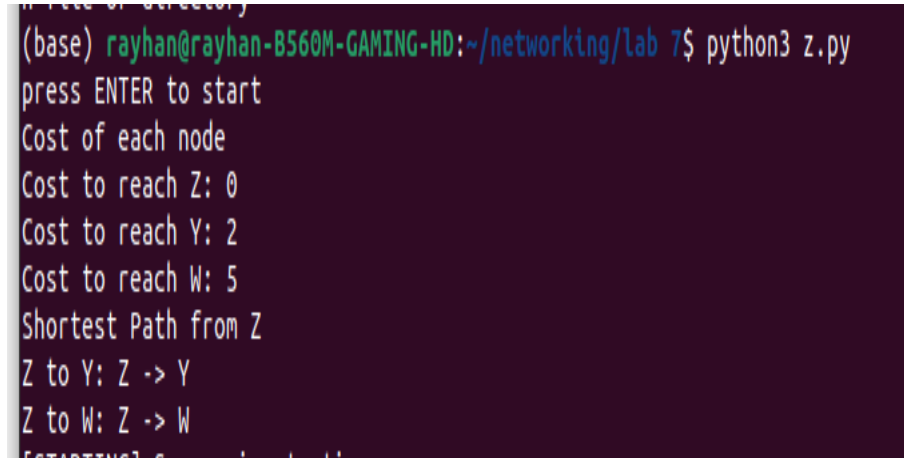
Figure 7: X router's Initial Path Cost

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

```
USERError: [Errno 98] Address already in use
(base) rayhan@rayhan-B560M-GAMING-HD:~/networking/lab 7$ python3 y.py
press ENTER to start
Cost of each node
Cost to reach Y: 0
Cost to reach W: 1
Cost to reach X: 1
Cost to reach Z: 2
Shortest Path from Y
Y to W: Y -> W
Y to X: Y -> X
Y to Z: Y -> Z
```

Figure 8: Y router's Initial Path Cost

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



```
(base) rayhan@rayhan-B560M-GAMING-HD:~/networking/lab 7$ python3 z.py
press ENTER to start
Cost of each node
Cost to reach Z: 0
Cost to reach Y: 2
Cost to reach W: 5
Shortest Path from Z
Z to Y: Z -> Y
Z to W: Z -> W
```

Figure 9: Z router's Initial Path Cost

5 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.

6 Algorithm Performance

The provided code implements Dijkstra's algorithm to find the shortest path in a weighted graph. Dijkstra's algorithm has a time complexity of $O((E+V)\log V)$, where E is the number of edges and V is the number of vertices in the graph. In this implementation, a heap data structure is used to efficiently keep track of the node with the minimum cost and its adjacent nodes, resulting in a logarithmic time complexity for accessing the node with the minimum cost. In terms of memory usage, the algorithm's consumption depends on the size of the graph and the data structures used to store both the graph and intermediate results. Here, the graph is represented using a dictionary data structure, where memory usage scales with the number of edges and vertices. Moreover, a priority queue (heap) is utilized to store nodes and their associated costs, adding to the memory consumption. Regarding network communication, the algorithm communicates with its neighbors to update its distance table. The amount of network traffic generated depends on the update frequency and the number of neighbors in the graph. The time and memory complexities of the algorithm are influenced by the dimensions of the graph and the update frequency. While Dijkstra's algorithm is generally efficient for small to medium-sized graphs, its practicality decreases for very large graphs with millions of nodes and edges.

References

- [1] Computer networking : a top-down approach 6th ed.
- [2] Link State Routing Algorithm: <https://www.upgrad.com/tutorials/software-engineering/software-key-tutorial/what-is-link-state-routing/>
- [3] Dijkstra's Algorithm: <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>