



UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 7: Implementation of Link State Algorithm

Submitted By :

Name: Md. Emon Khan

Roll No : 30

Name: Mahmudul Hasan

Roll No : 60

Submitted To :

Dr. Md. Abdur Razzaque

Dr. Md. Mamun Or Rashid

Dr. Muhammad Ibrahim

Md. Redwan Ahmed Rizvee

Submitted On : March 28, 2024

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 2 |
| 1.1 | Objectives | 2 |
| 2 | Theory | 2 |
| 2.1 | Router Mapping | 2 |
| 2.2 | Creation, Maintenance, and Updating of a Topology | 3 |
| 3 | Methodology | 3 |
| 4 | Experimental results | 6 |
| 4.1 | Snapshots of the Program Outputs | 6 |
| 4.2 | Performance Analysis | 12 |
| 5 | Experience | 16 |

1 Introduction

Link State is one of the most widely used routing protocols. It holds significant importance in computer networking. In this routing protocol, the routers construct a graph of all interconnected routers in its network. Using this graph, routers can calculate the shortest path to a given destination using a shortest path algorithm. The primary objective of this lab is to introduce participants with the fundamentals of the Link State protocol to develop and deepen the understanding of its intricacies.

1.1 Objectives

Here are the main goals of this lab:

- To learn how the Link State routing protocol helps distant computers, clients, routers, servers or any other machines talk to each other efficiently.
- To get introduced to the concept of finding the best routes by making a basic routing system for testing.
- Get a good overview of how the Link State algorithm actually operates by trying to implement it

2 Theory

Link State routing algorithm treats all the routers/devices within a network as nodes, and the links among them as edges of a graph that represents a network topology. Each edge of the topology has a cost associated with it. Then each router runs a shortest path algorithm on the graph to find the minimum cost path from itself to all other nodes.

2.1 Router Mapping

Each router in the network has a mapping of all the routers in that network. The mapping consists of the addresses of the routers. Typically the address refers to the IP of the routers. In our lab implementation, as all the routers are represented by different processes, the addresses here refer the ports mapped with the names of each of the routers.

2.2 Creation, Maintenance, and Updating of a Topology

Each router initially knows some of the routers (but not necessarily all) and the cost of communicating with each. Then they send their own data to their respective neighbours as an LSP or Link State Packet. Upon receiving an LSP, a router runs a shortest path algorithm on the updated topology to determine the minimum costs to all other routers or nodes.

3 Methodology

The implementation of Link State algorithm involves the creation of a network topology, creation of servers/routers or nodes using socket programming, implementing the logic and maintaining data for each router inside file systems.

- **Configuration File for Maintenance**

First we create a file that systematically defines the mapping of the routers, and the definition of the topology. We choose to save the structure into a JSON file. The structure looks like this:

```
{
  "routers": {
    "router1": 8001,
    "router2": 8002,
    "router3": 8003,
    "router4": 8004,
    "router5": 8005,
    "router6": 8006
  },
  "graph": [
    ["router1", "router2", 2],
    ["router1", "router3", 5],
    ["router1", "router4", 1],
    ["router2", "router3", 3],
    ["router2", "router4", 3],
    ["router3", "router4", 3],
    ["router3", "router5", 1],
    ["router3", "router6", 5],
    ["router4", "router5", 1],
    ["router5", "router6", 2]
  ]
}
```

}

- **Create Servers for each Router**

Then we proceed to create the servers for each router. We first read the information of the routers from the file we just created. Then we create separate threads for the routers and host their servers on their respective ports

- **Implement Router logic**

Each router handles two cases

- **Listen for incoming LSPs** Each router keeps listening for any incoming change. Upon receiving an LSP the router does the following operations sequentially:
 - The router saves the LSP data into a dedicated file
 - The router extracts the new links and updates the topology it saved in its dedicated server file
 - Then it runs a Dijkstra (a shortest path algorithm) on the updated topology and saves the new shortest path information to file
- **Send LSP to others** Each router keeps an eye on the server dedicated file. On the occasion of any change in the shortest file information, it creates an LSP of its shortest paths and sends to neighbouring nodes or routers. Records in the server dedicated files are saved as below:

```
[
  {
    "id": "router1",
    "savetime": 1711505324.804621,
    "ttl": 60,
    "links": {
      "router1-router2": 2,
      "router1-router3": 4,
      "router1-router4": 1,
      "router1-router5": 2
    }
  },
  {
    "id": "router2",
```

```

        "savetime": 1711505324.804621,
        "ttl": 60,
        "links": {
            "router1-router2": 2,
            "router2-router3": 3,
            "router2-router4": 3
        }
    },
    {
        "id": "router4",
        "savetime": 1711505324.8077211,
        "ttl": 60,
        "links": {
            "router1-router4": 1,
            "router2-router4": 3,
            "router3-router4": 3,
            "router4-router5": 1
        }
    }
]

```

- **Manage the Configuration File for Testing**

The configuration file is being updated every 30 seconds automatically by the script. This way the router mapping and network graph definition can be changed, and performance of the implementation can be analysed

- **Analysing the Implementation** To analyze the performance of the implementation, we have done the following:

- The main function of the code runs a process to test each test case
- In each case, the code generates a graph with n nodes and m links connecting them
- The loop that runs the tests increments n by itself, ie, doubles the number of nodes n in each loop until n is greater 256.
- In that loop runs another loop that calculates m starting from $m = n$. And in each loop, m is doubled until it reaches $n*(n-1)/2$
- Then the function generates a random graph of the calculated size and starts the process that runs the test

- The process is awaited for 40s to finish. Another 10s delay is added inside the process to await the routers to start. So, in average a test takes 50-60 seconds
- After terminating the process, the analysis and reports are printed in the console

But it turns out, the output is very large. So, output has been printed into a file named "ExperimentResults.txt". And finally the reports for all test results have been printed in another file "Reports.txt"

4 Experimental results

4.1 Snapshots of the Program Outputs

```
D:\Codes\py\Networking-Lab\Lab 7>python main.py
Test 1: Running new test...

Router router1 shortest paths: [['router2', 7], ['router3', 69], ['router4', 58]]
Router router2 shortest paths: [['router1', 7], ['router3', 76], ['router4', 65]]
Router router3 shortest paths: [['router1', 69], ['router2', 169], ['router4', 11]]
Router router4 shortest paths: [['router1', 58], ['router2', 65], ['router3', 11]]
Total nodes: 4          Total links: 4          Time elapsed: 0.2293243408203125 s          Total Memory used: 2,4443359375 kB

Test 2: Running new test...
█
```

Figure 1: Output of a Sample Topology Generated by the Script

```

D:\Codes\py\Networking-Lab\Lab 7>python main.py
Test 1: Running new test...

Router router1 shortest paths: [['router2', 7], ['router3', 69], ['router4', 58]]
Router router2 shortest paths: [['router1', 7], ['router3', 76], ['router4', 65]]
Router router3 shortest paths: [['router1', 69], ['router2', 109], ['router4', 11]]
Router router4 shortest paths: [['router1', 58], ['router2', 65], ['router3', 11]]
Total nodes: 4      Total links: 4      Time elapsed: 0.2293243408203125 s      Total Memory used: 2.4443359375 kB

Test 2: Running new test...

Router router1 shortest paths: [['router4', 14], ['router5', 12], ['router6', 18], ['router7', 100], ['router8', 23]]
Router router2 shortest paths: [['router5', 15], ['router8', 4]]
Router router3 shortest paths: [['router6', 158], ['router7', 76]]
Router router4 shortest paths: [['router1', 14], ['router5', 26], ['router6', 4], ['router7', 86]]
Router router5 shortest paths: [['router1', 12], ['router2', 15], ['router4', 26], ['router6', 98], ['router8', 11]]
Router router6 shortest paths: [['router1', 18], ['router3', 158], ['router4', 4], ['router5', 30], ['router7', 82]]
Router router7 shortest paths: [['router1', 168], ['router3', 76], ['router4', 86], ['router6', 82]]
Router router8 shortest paths: [['router1', 23], ['router2', 4], ['router5', 11]]
Total nodes: 8      Total links: 8      Time elapsed: 0.341646671295166 s      Total Memory used: 5.830078125 kB

Test 3: Running new test...

```

Figure 2: The Script Keeps Generating and Running the Test Cases

```

240', 78], ['router66', 57], ['router79', 60]]
router95 shortest paths: [['router113', 39], ['router175', 119], ['router195', 92], ['router243', 92], ['router248', 81], ['router250', 48], ['router31', 120], ['router42', 94], ['router57', 66]]
router96 shortest paths: [['router101', 138], ['router163', 117], ['router193', 153], ['router26', 76], ['router97', 161]]
router97 shortest paths: [['router141', 129], ['router48', 83], ['router91', 161]]
router98 shortest paths: [['router140', 112], ['router151', 108], ['router153', 67], ['router186', 145], ['router218', 113], ['router254', 45], ['router38', 56], ['router49', 111]]
router99 shortest paths: [['router124', 46], ['router127', 53], ['router138', 29], ['router163', 6], ['router218', 113], ['router254', 45], ['router38', 56], ['router49', 111]]
Total nodes: 256      Total links: 256      Time elapsed: 2.67047023773195
5 kB

All samples finished testing.
Here's an analysis of the Link State algorithm implementation:

Test Case 1
Number of nodes: 4
Number of edges: 4
Total Time Taken by the Process: 0.3128232955932617
Total Memory used by the Process: 2.4482421875

Test Case 2
Number of nodes: 8
Number of edges: 8
Total Time Taken by the Process: 0.2801685333251953
Total Memory used by the Process: 6.78125

Test Case 3
Number of nodes: 8
Number of edges: 16

```

Figure 3: The last case ends with a Message and the Report Starts


```
Test Case 1
Number of nodes: 4
Number of edges: 4
Total Time Taken by the Process: 0.3128232955932617
Total Memory used by the Process: 2.4482421875

Test Case 2
Number of nodes: 8
Number of edges: 8
Total Time Taken by the Process: 0.2801685333251953
Total Memory used by the Process: 6.78125

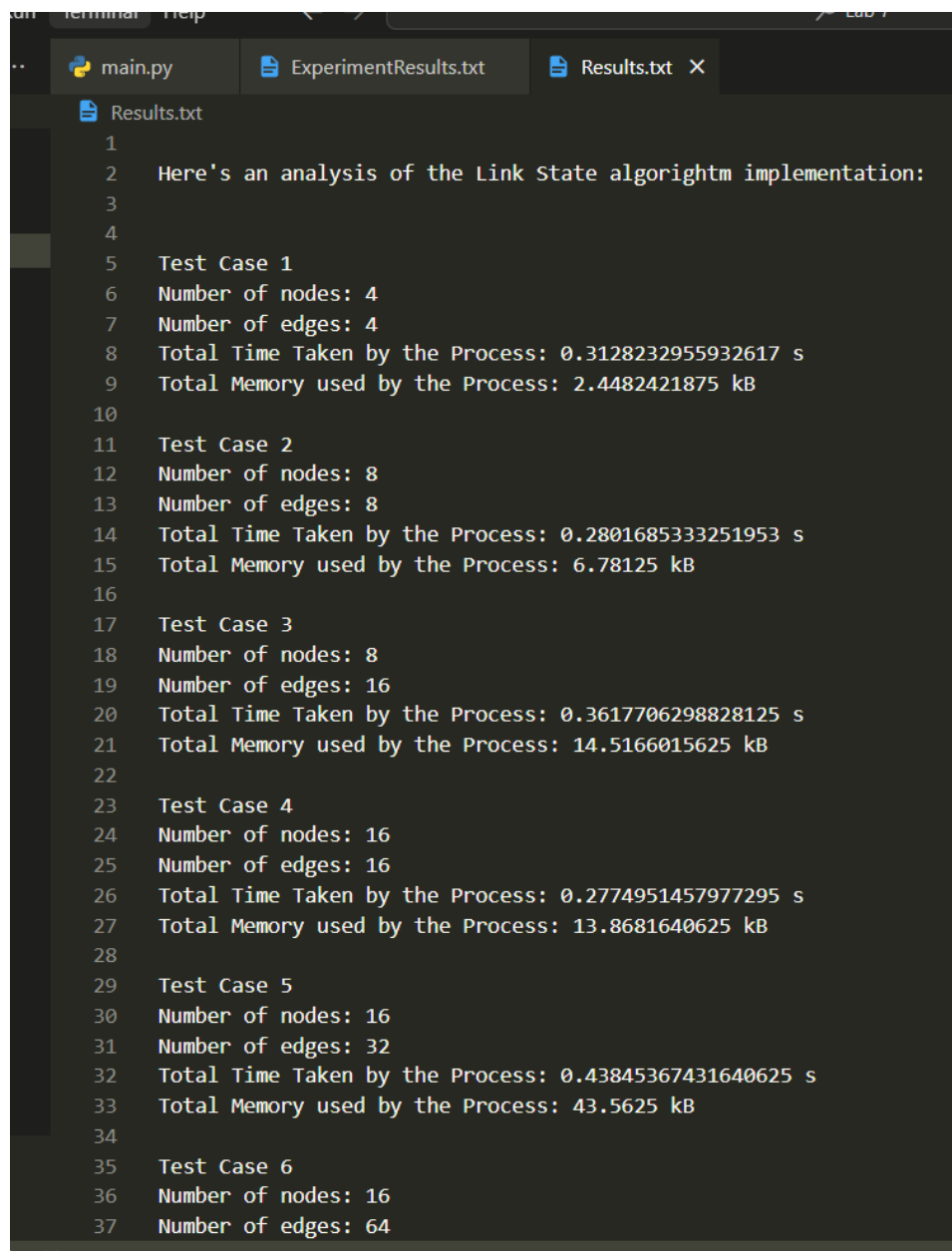
Test Case 3
Number of nodes: 8
Number of edges: 16
Total Time Taken by the Process: 0.3617706298828125
Total Memory used by the Process: 14.5166015625

Test Case 4
Number of nodes: 16
Number of edges: 16
Total Time Taken by the Process: 0.2774951457977295
Total Memory used by the Process: 13.8681640625

Test Case 5
Number of nodes: 16
Number of edges: 32
Total Time Taken by the Process: 0.43845367431640625
Total Memory used by the Process: 43.5625

Test Case 6
Number of nodes: 16
```

Figure 4: Reports of each Case is Finally Printed Separately

A screenshot of a code editor window with a dark theme. The editor has three tabs at the top: 'main.py', 'ExperimentResults.txt', and 'Results.txt'. The 'Results.txt' tab is active and shows a text file with line numbers on the left. The text content is as follows:

```
1  
2 Here's an analysis of the Link State algorithm implementation:  
3  
4  
5 Test Case 1  
6 Number of nodes: 4  
7 Number of edges: 4  
8 Total Time Taken by the Process: 0.3128232955932617 s  
9 Total Memory used by the Process: 2.4482421875 kB  
10  
11 Test Case 2  
12 Number of nodes: 8  
13 Number of edges: 8  
14 Total Time Taken by the Process: 0.2801685333251953 s  
15 Total Memory used by the Process: 6.78125 kB  
16  
17 Test Case 3  
18 Number of nodes: 8  
19 Number of edges: 16  
20 Total Time Taken by the Process: 0.3617706298828125 s  
21 Total Memory used by the Process: 14.5166015625 kB  
22  
23 Test Case 4  
24 Number of nodes: 16  
25 Number of edges: 16  
26 Total Time Taken by the Process: 0.2774951457977295 s  
27 Total Memory used by the Process: 13.8681640625 kB  
28  
29 Test Case 5  
30 Number of nodes: 16  
31 Number of edges: 32  
32 Total Time Taken by the Process: 0.43845367431640625 s  
33 Total Memory used by the Process: 43.5625 kB  
34  
35 Test Case 6  
36 Number of nodes: 16  
37 Number of edges: 64
```

Figure 5: Reports are saved in "Results.txt"

```
terminal Help  Lab 7
main.py x ExperimentResults.txt x Results.txt
ExperimentResults.txt
1 Test 1: Running new test...
2
3 Sample:
4 {
5   "routers": {
6     "router1": 8001,
7     "router2": 8002,
8     "router3": 8003,
9     "router4": 8004
10  },
11  "graph": [
12    [
13      "router1",
14      "router4",
15      44
16    ],
17    [
18      "router2",
19      "router3",
20      46
21    ],
22    [
23      "router3",
24      "router4",
25      36
26    ],
27    [
28      "router2",
29      "router4",
30      60
31    ]
32  ]
33 }
34
35 Output:
36 router1 shortest paths: [['router2', 104], ['router3', 80], ['router4', 44]]
37 router2 shortest paths: [['router1', 104], ['router3', 46], ['router4', 60]]
```

Figure 6: All Test Results are Saved in "ExperimentResults.txt". This also Shows the Sample Graph used in that Case

```
[
  [
    "router2",
    "router3",
    46
  ],
  [
    "router3",
    "router4",
    36
  ],
  [
    "router2",
    "router4",
    60
  ]
]
}

Output:
router1 shortest paths: [['router2', 104], ['router3', 80], ['router4', 44]]
router2 shortest paths: [['router1', 104], ['router3', 46], ['router4', 60]]
router3 shortest paths: [['router1', 80], ['router2', 46], ['router4', 36]]
router4 shortest paths: [['router1', 44], ['router2', 60], ['router3', 36]]
Total nodes: 4          Total links: 4          Time elapsed: 0.3128232955932617 s          Total Memory used: 2.4482421875 kB

Test 2: Running new test...

Sample:
{
  "routers": {
    "router1": 8001,
    "router2": 8002,
    "router3": 8003,
```

Figure 7: The Outputs are Shown Below their Respective Samples

```
main.py ExperimentResults.txt Results.txt
ExperimentResults.txt
57849 router86 shortest paths: [['router156', 141], ['router16', 61], ['router166', 105], ['router179', 163], ['router193', 58], ['router212',
57850 router87 shortest paths: [['router103', 38], ['router139', 45], ['router14', 127], ['router220', 54], ['router239', 125]]
57851 router9 shortest paths: [['router11', 162], ['router111', 98], ['router116', 27], ['router155', 54], ['router191', 144], ['router218', 10
57852 router91 shortest paths: [['router112', 98], ['router137', 174], ['router141', 124], ['router169', 138], ['router208', 34], ['router229', 10
57853 router92 shortest paths: [['router123', 87], ['router191', 83], ['router213', 137], ['router22', 82]]
57854 router93 shortest paths: [['router149', 23], ['router59', 5]]
57855 router94 shortest paths: [['router100', 45], ['router11', 72], ['router119', 102], ['router152', 69], ['router157', 91], ['router187', 7
57856 router95 shortest paths: [['router113', 39], ['router175', 119], ['router195', 92], ['router243', 92], ['router28', 68]]
57857 router96 shortest paths: [['router101', 138], ['router163', 117], ['router193', 153], ['router26', 76], ['router38', 67]]
57858 router97 shortest paths: [['router141', 129], ['router48', 83], ['router91', 161]]
57859 router98 shortest paths: [['router140', 112], ['router151', 108], ['router153', 67], ['router186', 145], ['router200', 72], ['router218', 10
57860 router99 shortest paths: [['router124', 46], ['router127', 53], ['router138', 29], ['router163', 6], ['router195', 100], ['router206', 10
57861 Total nodes: 256          Total links: 256          Time elapsed: 2.6704702377319336 s          Total Memory used: 306.115234375 kB
57862
57863
57864
57865
57866 All samples finished testing!! See Reports in Reports.txt
57867
```

Figure 8: The Output of all Test Results Combined is Huge

4.2 Performance Analysis

Here's what we have in the "Reports.txt" output file after running the program once:

Here's an analysis of the Link State algorithm implementation:

Test Case 1

Number of nodes: 4

Number of edges: 4

Total Time Taken by the Process: 0.22468233108520508 s

Total Memory used by the Process: 2.4423828125 kB

Test Case 2

Number of nodes: 8

Number of edges: 8

Total Time Taken by the Process: 0.25572800636291504 s

Total Memory used by the Process: 7.205078125 kB

Test Case 3

Number of nodes: 8

Number of edges: 16

Total Time Taken by the Process: 0.303241491317749 s

Total Memory used by the Process: 15.14453125 kB

Test Case 4

Number of nodes: 16

Number of edges: 16

Total Time Taken by the Process: 0.28476834297180176 s

Total Memory used by the Process: 17.9951171875 kB

Test Case 5

Number of nodes: 16

Number of edges: 32

Total Time Taken by the Process: 0.40393900871276855 s

Total Memory used by the Process: 45.3134765625 kB

Test Case 6

Number of nodes: 16

Number of edges: 64

Total Time Taken by the Process: 0.6700005531311035 s

Total Memory used by the Process: 99.1875 kB

Test Case 7

Number of nodes: 32

Number of edges: 32

Total Time Taken by the Process: 0.3645634651184082 s

Total Memory used by the Process: 35.2548828125 kB

Number of nodes: 32

Number of edges: 64

Total Time Taken by the Process: 0.5485620498657227 s

Total Memory used by the Process: 104.619140625 kB

Test Case 8

Number of nodes: 32

Number of edges: 128

Total Time Taken by the Process: 1.0149662494659424 s

Total Memory used by the Process: 281.35546875 kB

Test Case 9

Number of nodes: 32

Number of edges: 256

Total Time Taken by the Process: 2.472524881362915 s

Total Memory used by the Process: 733.0146484375 kB

Test Case 10

Number of nodes: 64

Number of edges: 64

Total Time Taken by the Process: 0.5164077281951904 s

Total Memory used by the Process: 71.5146484375 kB

Test Case 11

Number of nodes: 64

Number of edges: 128

Total Time Taken by the Process: 0.8324041366577148 s

Total Memory used by the Process: 230.4697265625 kB

Test Case 12

Number of nodes: 64

Number of edges: 256

Total Time Taken by the Process: 1.9751358032226562 s
Total Memory used by the Process: 711.54296875 kB

Test Case 13

Number of nodes: 64
Number of edges: 512
Total Time Taken by the Process: 4.9778618812561035 s
Total Memory used by the Process: 1924.3017578125 kB

Test Case 14

Number of nodes: 64
Number of edges: 1024
Total Time Taken by the Process: 16.11722445487976 s
Total Memory used by the Process: 5551.0029296875 kB

Test Case 15

Number of nodes: 128
Number of edges: 128
Total Time Taken by the Process: 0.8868918418884277 s
Total Memory used by the Process: 149.98828125 kB

Test Case 16

Number of nodes: 128
Number of edges: 256
Total Time Taken by the Process: 1.7217929363250732 s
Total Memory used by the Process: 471.498046875 kB

Test Case 17

Number of nodes: 128
Number of edges: 512
Total Time Taken by the Process: 3.747854709625244 s
Total Memory used by the Process: 1611.43359375 kB

Test Case 18

Number of nodes: 128
Number of edges: 1024
Total Time Taken by the Process: 10.702005386352539 s
Total Memory used by the Process: 4834.69140625 kB

Test Case 19

Number of nodes: 128
Number of edges: 2048
Total Time Taken by the Process: 30.002763032913208 s
Total Memory used by the Process: 11210.8466796875 kB

Test Case 20
Number of nodes: 128
Number of edges: 4096
Total Time Taken by the Process: 30.007826805114746 s
Total Memory used by the Process: 12079.98828125 kB

Test Case 21
Number of nodes: 256
Number of edges: 256
Total Time Taken by the Process: 2.683258533477783 s
Total Memory used by the Process: 284.9521484375 kB

The report clearly demonstrates the impact of graph size on the performance of Dijkstra's algorithm:

- **Increased Nodes and Edges Lead to Higher Requirements:** Both time and memory usage increase as the number of nodes (routers) and edges (links) in the graph grow. This is evident from the significant difference in execution time and memory consumption between test cases:
 - Test Case 20 (largest graph): 30.008 seconds and 12080kB
 - Test Case 1 (smallest graph): 0.22 seconds and 2.4kB
- **Edges Significantly Impact Performance:** Even with the same number of nodes, graphs with more edges exhibit a substantial increase in time complexity and memory usage. Test cases 20 and 21 illustrate this well:
 - Test Case 20 (128 nodes, 4096 edges): 12x slower and 42x higher memory usage compared to Test Case 21 (256 nodes, 256 edges)
- **Theoretical Time Complexity:** When implemented with a priority queue, Dijkstra's algorithm has a time complexity of $O((V+E) \log V)$, where V is the number of nodes and E is the number of edges. Considering all routers, the combined complexity might approach $O(V(V+E) \log V)$.

- **Small Graph Complexity:** For a graph with 32 nodes and 64 edges, the time complexity translates to approximately 4640 (times two or three at most) operations: $(32(32 + 64) \log 32) \approx 4640$. This suggests the Dijkstra computations themselves take minimal time (like a few milliseconds).
- **Dominant Factors for Delays:** The observed delays likely arise from factors beyond Dijkstra's algorithm itself, such as:
 - Network communication (socket communication)
 - File I/O operations (read/write)
 - Waiting for responses from other network entities

In essence, while the number of nodes and edges influences the overall complexity of Dijkstra's algorithm, the report suggests that for smaller graphs, other factors likely dominate the observed execution time.

5 Experience

- Designed the JSON structures to store the data for each server
- We used multiprocessing to handle the router configurations
- Used system methods to detect and handle file change
- Used system methods to delete or create new files
- Learned more about Link State algorithm and its efficiency through the analysis

References

- [1] <https://www.geeksforgeeks.org/>
- [2] <https://www.javatpoint.com/>
- [3] <https://www.programiz.com/>