OSPF Routing Algorithm

Report for Assignment 3 as part of the *CS3205 : Introduction to Computer Networks* course

Shreesha G Bhat (CS18B103)
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

I. Aim

The aim of the assignment is to simulate a simplified version of the *Open Shortest Path First* (OSPF) Routing Algorithm.

II. Introduction

Network Routing i.e. choosing the best path among all the available paths can be viewed as a policy decision in intra-domain networks. The Open Shortest Path First (OSPF) routing protocol is one such protocol which is based on Dijkstra's algorithm. Each router periodically *floods* local reachability information to it's neighbours which is then circulated throughout the network. Eventually, each router gets to know the complete network topology and can then locally run Dijsktra's algorithm to find the shortest path to every other router and fill it's local routing table entries correspondingly.
In this assignment, we represents routers as nodes of a graph and network links as the undirected edges between the nodes. Assume the node IDs are numbered from $\{0, 1, 2, ...n\}$, each node i performs the following tasks.

- Periodically send `HELLO` messages to neighbours with IDs $j > i$.
- Send `HELLOREPLY` messages to the node j from which the corresponding `HELLO` message was received. This message contains information about the edge weight for the edge (i, j).
- Periodically send `LSA` advertisement messages to all it's neighbours.
- Store information from received `LSA` messages and forward the message to it's neighbours.
- Periodically run the Dijsktra's shortest path algorithm using the network topology constructed from recently received `LSA` messages and fill the local routing table.

III. Experimental Details

A. Experimental Setup

1) Program Invocation: Each router is implemented as a separate Linux process. The executable is run with the following command line arguments

```
./ospf -i <node_id> -f <infile> -o <outfile> -h
           <hi> -a <lsai> -s <spfi>
```

where the values specified in the command line arguments are

- `-i <node_id>` : Node identifier for the current process.
- `-f <infile>` : Input file
- `-o <outfile>` : Output file
- `-h <hi>` : Interval between consecutive `HELLO` messages sent from the current node.
- `-a <lsai>` : Interval between consecutive `LSA` messages advertised by the current node.
- `-s <spfi>` : Interval between consecutive runs of the Shortest Path algorithm.

2) Input File Format: The input file format is as follows

TABLE I: Example Input File Format

5		10	
0	2	2	8
2	3	5	10
3	4	6	20
4	7	4	10
...			

The first row contains the number of nodes N and number of edges E respectively. The node indices go from 0 to $N - 1$. Each subsequent row represents one edge. The first two entries are the two end points of the edge and the next two entries are the minimum and maximum possible edge weights for the corresponding edge.

3) Process Description: Once the process corresponding to the router i is started, it does the following tasks

- Obtain necessary parameters such as `node_id`, output file, input file and other intervals from the command line arguments.
- Read the input file and find out it's neighbouring nodes.
- Establish a UDP socket on port number `10000 + i` for all further OSPF communications.
- Branch into 3 threads
  - **The `hello_msg_sender` thread** :
    Sends a `HELLO` message to neighbours $j > i$ every `hi` seconds. This value is specified in the command line argument with a default value of 1 second. The packet format is as shown below

HELLO	srcid

  - **The `lsa_advertiser` thread** :
    Sends a *Link State Advertisement* (LSA) message to it's neighbours every `lsai` seconds. This value is specified in the command line argument with a default value of 5 seconds. The packet format is as shown below

LSA	srcid	sqnum	No. entries	Neigh1	Cost1	...

    The sequence number is incremented by the sender for every LSA message it sends.
  - **The `shortest_path` thread** :
    Determines the topology using all the recent LSA messages received from all other routers and then runs Dijsktra's shortest path algorithm once every `spfi` intervals. This value is specified in the command line argument with a default value of 20 seconds.
    The output of each shortest path computation is stored in the output file. The output file for node i will be `outfile-i.txt`, where `outfile` is specified in the command line. The output file format is as follows

TABLE II: Example Output File format for Node 1 at Time 30

Routing Table for Node No. 1 at Time 30		
Destination	Path	Cost
2	1-3-2	5
3	1-3	2
4	1-3-2-4	10
...		

- Apart from the above 3 threads we also have a receiver thread (the main thread). This thread receives messages on the established UDP port and performs the following actions
  - On receiving a `HELLO` message on an interface, replies with a `HELLOREPLY` message along with the corresponding edge cost. The cost reported by node j on receiving a `HELLO` message from node i is a random number between `min` $c_{(i,j)}$ and `max` $c_{(i,j)}$ as defined by the input file as shown in Table I. The node i on receiving the `HELLOREPLY` message updates it's internal data structures with the cost as sent in the message. The message format is as follows

`HELLOREPLY`	`j`	`i`	`cost of link (i, j)`

    Note that node j also updates it's internal tables with the corresponding link cost.
  - On receiving a `LSA` message from a neighbour j, node i will store the LSA information and forward the message to all neighbouring nodes other than j *if and only if* the newly received LSA's sequence number is strictly greater than the last known sequence number from the originator of the `LSA` message.

B. Components Involved

The code section for the `ospf.cpp` contains the `main` function followed by 3 other functions for the 3 threads; `hello_msg_sender`, `lsa_advertiser` and `shortest_path`.
A `graph` and `parameters` class has been defined to host the functions and the state variables for the graph and the 3 parameters `lsai`, `spfi`, `hi` respectively.
The main function takes in the command line arguments and creates a graph object after reading the input file. The edge weights of the graph are initialized to their maximum possible value according to the input file. Then a UDP socket is setup at port number `10000 + i` for process with node ID i for all further communication.
Further on, 3 threads are invoked to perform their respective tasks as defined above. The main thread acts as a receiver thread.
The `shortest_path` thread writes the routing table information to the file `output-i.txt` for the process with node ID i at intervals as specified by the parameter `spfi`.
A *mutex* is defined for the edge weight data structures in the `graph` class inorder to handle concurrency issues when different threads read or write to this data structure simultaneously. Locks have been defined and used at appropriate places to define critical sections so that concurrency issues are resolved.

C. Additional Details

A `Makefile` and bash script have been provided to invoke the required number of processes with the corresponding command line arguments. The bash script also kills the processes created after 100 seconds.

IV. Results

The algorithm was run on the following two different input graphs with the default set of parameters i.e. `hi` = 1, `spfi` = 20, `lsai` = 5.

TABLE III: Input File : `example-input1`

8		20	
0	1	2	5
0	7	10	18
0	3	1	10
0	2	11	17
1	2	4	8
1	6	5	10
1	5	5	7
1	4	15	20
1	7	6	15
1	3	6	10
2	5	8	10
2	3	4	10
2	6	5	9
2	4	7	21
2	7	9	16
3	7	2	6
3	5	3	12
3	4	13	15
4	5	10	21
5	7	11	18

Figure 1 is a representational figure for the graph as given in Table III

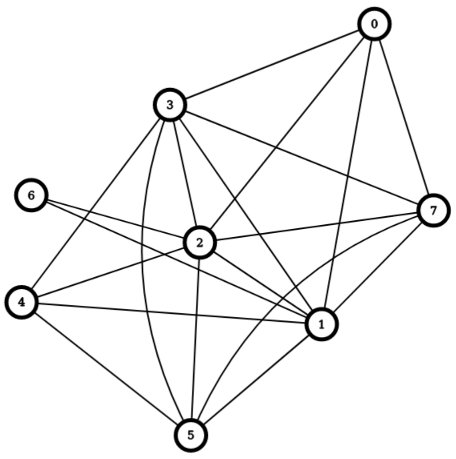

Fig. 1: Representational Figure for the graph as given by `example-input1`

TABLE IV: Input File : `example-input2`

8		20	
0	1	1	8
0	5	4	9
0	7	5	15
0	4	3	11
1	3	7	17
1	6	8	10
1	5	10	17
1	2	5	10
1	4	12	14
2	7	9	21
2	6	8	17
2	3	5	6
2	5	12	17
3	7	3	17
3	4	8	13
3	5	10	20
4	7	11	18
4	5	8	19
5	7	20	24
5	6	21	27

Figure 2 is a representational figure for the graph as given in Table IV

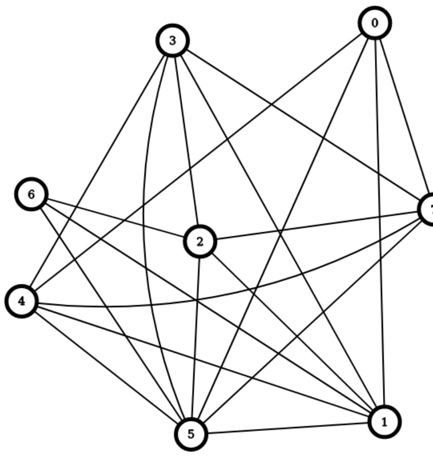

Fig. 2: Representational Figure for the graph as given by `example-input2`

For each input file, the 8 processes were made to run for 100 seconds and then killed.
The following are the routing tables for each input file and corresponding nodes at time 100

A. *Routing Tables at time 100 for* `example-input1`

Routing Table for Node No. 0 at time 100		
Destination	Path	Cost
1	0-1	4
2	0-1-2	9
3	0-3	1
4	0-3-4	16
5	0-1-5	11
6	0-1-6	9
7	0-3-7	3

Routing Table for Node No. 1 at time 100		
Destination	Path	Cost
0	1-0	4
2	1-2	5
3	1-0-3	6
4	1-4	17
5	1-5	7
6	1-6	8
7	1-7	6

Routing Table for Node No. 2 at time 100		
Destination	Path	Cost
0	2-1-0	7
1	2-1	5
3	2-3	6
4	2-4	17
5	2-5	9
6	2-6	8
7	2-3-7	8

Routing Table for Node No. 3 at time 100		
Destination	Path	Cost
0	3-0	1
1	3-0-1	3
2	3-2	6
4	3-4	15
5	3-5	5
6	3-0-1-6	8
7	3-7	6

Routing Table for Node No. 4 at time 100		
Destination	Path	Cost
0	4-3-0	17
1	4-1	17
2	4-2	17
3	4-3	15
5	4-5	14
6	4-1-6	22
7	4-3-7	17

Routing Table for Node No. 5 at time 100		
Destination	Path	Cost
0	5-3-0	6
1	5-1	7
2	5-2	9
3	5-3	5
4	5-4	14
6	5-1-6	12
7	5-3-7	7

Routing Table for Node No. 6 at time 100		
Destination	Path	Cost
0	6-1-0	12
1	6-1	8
2	6-2	8
3	6-1-0-3	13
4	6-1-4	25
5	6-1-5	15
7	6-1-7	14

Routing Table for Node No. 7 at time 100		
Destination	Path	Cost
0	7-3-0	7
1	7-1	6
2	7-2	11
3	7-3	6
4	7-3-4	21
5	7-3-5	11
6	7-1-6	14

B. *Routing Tables at time 100 for* `example-input2`

Routing Table for Node No. 0 at time 100		
Destination	Path	Cost
1	0-1	7
2	0-1-2	12
3	0-1-2-3	18
4	0-4	10
5	0-5	9
6	0-1-6	15
7	0-7	6

Routing Table for Node No. 1 at time 100		
Destination	Path	Cost
0	1-0	7
2	1-2	5
3	1-3	11
4	1-0-4	11
5	1-0-5	12
6	1-6	10
7	1-2-7	17

Routing Table for Node No. 2 at time 100		
Destination	Path	Cost
0	2-1-0	6
1	2-1	5
3	2-3	6
4	2-1-0-4	10
5	2-1-0-5	11
6	2-6	13
7	2-7	16

Routing Table for Node No. 3 at time 100		
Destination	Path	Cost
0	3-1-0	12
1	3-1	11
2	3-2	6
4	3-4	10
5	3-5	11
6	3-2-6	14
7	3-7	11

Routing Table for Node No. 4 at time 100		
Destination	Path	Cost
0	4-0	10
1	4-1	14
2	4-3-2	16
3	4-3	10
5	4-5	8
6	4-1-6	22
7	4-7	11

Routing Table for Node No. 5 at time 100		
Destination	Path	Cost
0	5-0	9
1	5-1	15
2	5-2	16
3	5-3	11
4	5-4	8
6	5-6	25
7	5-0-7	15

Routing Table for Node No. 6 at time 100		
Destination	Path	Cost
0	6-1-0	17
1	6-1	10
2	6-2	13
3	6-2-3	19
4	6-1-4	24
5	6-5	25
7	6-1-0-7	23

Routing Table for Node No. 7 at time 100		
Destination	Path	Cost
0	7-0	6
1	7-0-1	13
2	7-2	16
3	7-3	11
4	7-4	11
5	7-0-5	15
6	7-0-1-6	23

V. Some Observations on the effect of the parameters

On running the program on an input file with two sets of parameter values (`hi`, `lsai`, `spfi`) = $(1, 5, 20)$ and $(5, 1, 20)$, the following observations can be made.

- The `HELLO` message exchange acts as a simulation of varying link costs. For our example simulations above, the interval between consecutive `HELLO` messages i.e. `hi` was set to the default value of 1. This means that for a node i, the local data structure holding the values of the costs for the various edges refreshes it's values for the edge weights between i and it's neighbours approximately every 1 second. But note that every other node receives information about the edge weights between i and it's neighbours once every $\texttt{lsai} = 5$ seconds (roughly). This being a less frequent event implies that nodes other than i and have a different view / older view of the graph (as given by the `LSA` messages) as compared to the node i at steady state. Therefore, the shortest path computation (which happens once every 20 seconds) is performed at different nodes using different views of the graph.
- If we instead set $\texttt{hi} > \texttt{lsai}$ (provided $\texttt{spfi} > \texttt{hi}$), that would mean that the link costs vary much more slowly as compared to the frequency with which `LSA` are sent and hence in this case, the entire network has a consistent view of the graph at the point of shortest path computation.

VI. Learning Outcomes & Conclusion

- Learnt and understood the OSPF routing protocol
- Coded up a simplified version of the OSPF protocol.
- Observed the effect of the parameters involved.

References

[1] Kurose, J. F. Ross, K. W. (2016), Computer Networking: A Top-Down Approach , Pearson , Boston, MA .
[2] OSPF slides provided by the TAs during class hours.
[3] CS3205 Lecture Slides