Report - CS3205 Assignment 3

Aniswar Srivatsa Krishnan
Computer Science and Engineering
Indian Institute of Technology Madras
CS18B050

***Abstract*—The objective of this assignment is to implement a simplified version of the Open Shortest Path First (OSPF) routing protocol.**

***Index Terms*—OSPF, BGP, IP, networks, Dijkstra, algorithm**

I. Introduction

The objective of this assignment is to implement a simplified version of the Open Shortest Path First (OSPF) routing protocol. In a simplified view of the network, it is seen as a collection of interconnected routers. One router is indistinguishable from another in the sense that all routers executed the same routing algorithm to compute routing paths through the entire network. In practice, this model and its view of a homogenous set of routers all executing the same routing algorithm is simplistic for two important reasons:

- *Scale*. As the number of routers becomes large, the overhead involved in communicating, computing, and storing routing information becomes prohibitive. Today's Internet consists of hundreds of millions of routers. Storing routing information for possible destinations at each of these routers would clearly require enormous amounts of memory. The overhead required to broadcast connectivity and link cost updates among all of the routers would be huge! A distance-vector algorithm that iterated among such a large number of routers would surely never converge. Clearly, something must be done to reduce the complexity of route computation in a network as large as the Internet.
- *Administrative autonomy*. The Internet is a network of ISPs, with each ISP consisting of its own network of routers. An ISP generally desires to operate its network as it pleases (for example, to run whatever routing algorithm it chooses within its network) or to hide aspects of its network's internal organization from the outside. Ideally, an organization should be able to operate and administer its network as it wishes, while still being able to connect its network to other outside networks.

Both of these problems can be solved by organizing routers into autonomous systems (ASs), with each AS consisting of a group of routers that are under the same administrative control. Often the routers in an ISP, and the links that interconnect them, constitute a single AS. Some ISPs, however, partition their network into multiple ASs. In particular, some tier-1 ISPs use one gigantic AS for their entire network, whereas others break up their ISP into tens of interconnected ASs. An autonomous system is identified by its globally unique autonomous system number (ASN). AS numbers, like IP addresses, are assigned by ICANN regional registries. Routers within the same AS all run the same routing algorithm and have information about each other. The routing algorithm running within an autonomous system is called an intra-autonomous system routing protocol.

II. Experimental Details

A. *Open Shortest Path First (OSPF) Routing :*

OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra's least-cost path algorithm. With OSPF, each router constructs a complete topological map (that is, a graph) of the entire autonomous system. Each router then locally runs Dijkstra's shortest-path algorithm to determine a shortest-path tree to all subnets, with itself as the root node. Individual link costs are configured by the network administrator. The administrator might choose to set all link costs to 1, thus achieving minimum-hop routing, or might choose to set the link weights to be inversely proportional to link capacity in order to discourage traffic from using low-bandwidth links. OSPF does not mandate a policy for how link weights are set (that is the job of the network administrator), but instead provides the mechanisms (protocol) for determining least-cost path routing for the given set of link weights.
With OSPF, a router broadcasts routing information to all other routers in the autonomous system, not just to its neighboring routers. A router broadcasts link-state information whenever there is a change in a link's state (for example, a change in cost or a change in up/down status). It also broadcasts a link's state periodically (at least once every 30 minutes), even if the link's state has not changed. This periodic updating of link state advertisements adds robustness to the link state algorithm. OSPF advertisements are contained in OSPF messages that are carried directly by IP, with an upper-layer protocol of 89 for OSPF. Thus, the OSPF protocol must itself implement functionality such as reliable message transfer and link-state broadcast. The OSPF protocol also checks that links are operational (via a HELLO message that is sent to an attached neighbor) and allows an OSPF router to obtain a neighboring router's database of network-wide link state.

B. *Assumptions and Variables*

The following are the assumptions and variables involved in our implementation of a simplified version of the Open Shortest Path First (OSPF) routing protocol for this assignment.

- Given a set of N routers, the goal is for EACH router to:
  1) exchange HELLO packets with neighbours
  2) create Link State Advertisement (LSA) packets based on neighboring nodes' info
  3) broadcast the LSA packets to all other routers in the network
  4) construct the network topology based on the LSA packets received from other routers
  5) determining the routing table entries based on this topology, by using Dijkstra' algorithm (single source - all nodes shortest paths). If multiple equal-cost paths exist, any one of them can be reported.
- The HELLO packets will be exchanged every hi seconds; the LSA updates will be sent every $lsai$ seconds; the routing table computation will be done every $spfi$ seconds.
- The input format is as follows:

TABLE I

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

  The first entry on the first line specifies the number of routers (N) The node indices go from 0 to (N - 1). The second entry on the first line specifies the number of links. Each subsequent row contains the tuple $(i, j, MinC_{ij}, MaxC_{ij})$. This implies a bidirectional link between nodes i and j . The use of minimum and maximum will be defined later.
- OSPF Processing: Each OSPF router (running as a Linux process) will perform the following actions:
  - Obtain necessary parameters including its node identifier (i)
  - Read the input file and find out its neighboring node identifiers.
  - Establish a UDP socket on port number (10000 + i) for all OSPF communications.
  - Send a HELLO message to its neighbors, once every HELLO INTERVAL seconds. This value is specified in the command line; Default: 1 second. You can have one thread implement this part. Packet Format:

HELLO	srcid

    where srcid is replaced with i.
  - When a router receives a HELLO message on an interface, it will reply with the HELLOREPLY message along with the cost. The cost reported for link ij by node j for a packet received from node i is a RANDOM number between $MinC_{ij}$ and $MaxC_{ij}$. Node i, on receiving this message, will store this value as the cost for $link_{ij}$ .The message format is:

HELLOREPLY	j	i	value for link ij

  - Send a Link State Advertisement (LSA) message to its neighbors, once every LSA INTERVAL seconds. This value is specified in the command line; Default: 5 seconds. You can have one thread implement this part. Packet Format:

LSA	srcid	Seq. No	No. Entries	Neigh1	Cost1	...

    The sequence number is incremented by the sender for each LSA message that it sends.
  - When a node receives an LSA message from a neighbor, it will store the LSA information and forward the LSA to all interfaces other than the interface that the packet arrived on, *if and only if* the newly received LSA's sequence number is strictly greater than the last known sequence number from the sender.
  - Determine the topology using all recent Link State Advertisement (LSA) messages received from all other routers; and then run shortest-cost path computation algorithm every SPF INTERVAL seconds. This value is specified in the command line; Default: 20 seconds.
  - The output will be stored in the routing table output file along with the time stamp. The output file name for Node i will be outfile-i.txt, where outfile is specified in the command line.
  - Output Format:

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
...		

C. *Implementation Details*

1) *Execution:*

- The program is invoked with the following command-line parameters:

```
% ./ospf -i id -f infile
-o outfile -h hi
-a lsai -s spfi
```

- The values specified in the command line are:
  - id: Node identifier value(i)
  - infile: Input file
  - outfile: Output file
  - hi: HELLO INTERVAL (in seconds)
  - lsai: LSA INTERVAL (in seconds)
  - spfi: SPF INTERVAL (in seconds)

The program is a multithreaded program involving 4 threads: 1 each for Sending Hello Packets, Sending LSA Packets, Determing Shortest Path and Receiving Packets

2) ***`argument_handler()`***:

- This function takes care of processing the command line arguments and initializing the corresponding variables in the program.

3) ***initialize_graph()****:*

- This function takes care of processing the input file and initializing the graph data structure in the program.
- The graph is stored as a vector of vector of triples (min, max, weight), in the adjacency matrix format. For each edge the min and max value are received from the input file and setup.
- The graph also has an associated mutex to help in synchronization when multiple threads access the graph.

4) ***get_sock_id()****:*

- This function takes care of creating the socket and binding it to the port (10000+i).
- This socket is used by the other routines for sending and receiving messages.

5) ***hello()****:*

- This function is executed in a separate thread and takes care of sending hello message to its neighbours once every hi seconds. The function sends the new round of hello messages only if all the neighbours have acknowledged the hello message in the previous round. This is accomplished by a shared counter which is updated every time an acknowledgement is received for the hello message.

6) ***lsa()****:*

- This function is executed in a separate thread and takes care of sending Link State Advertisement (LSA) message to its neighbors, once every $lsai$ seconds.
- We iterate through the graph, find the cost of the neighbours, form the message string and send it to the socket to all the neighbours.
- It also increments the sequence number.

7) ***ospf()****:*

- This function is executed in a separate thread and takes care of computing the shortest-path route from that router to all other routers in the network using Dijsktra's Single Source Shortest Path Algorithm, once every $spfi$ seconds.
- It prints the details of the paths to the outfile in the format mentioned above.

8) ***receive()****:*

- This function is executed in a separate thread and takes care of receiving messages and taking appropriate action, which may fall in the below three categories.
    1) **Hello Message**: It replies with the HELLOREPLY message along with the cost. The cost reported for link ij by node j for a packet received from node i is a RANDOM number between $MinC_{ij}$ and $MaxC_{ij}$. Node i, on receiving this message, will store this value as the cost for $link_{ij}$.
    2) **LSA Message**: Tt stores the LSA information and forwards the LSA to all interfaces other than the interface that the packet arrived on, *if and only if* the newly received LSA's sequence number is strictly greater than the last known sequence number from the sender. It also updates the last known sequence number
    3) **Hello Reply Message**: It updates the counter which checks whether all neighbours have acknowledged the previous round of Hello messages. It also updates the link weight according to the value given in the message.

9) ***main()****:*

- This fucntion calls `argument_handler()`, `initialize_graph()`, `get_sock_id()` and creates 4 threads for executing, `hello()`, `lsa()`, `ospf`, `receive()`.

III. Results and Observation

The outputs are obtained for the following input files and parameter combinations. Input 1 is run for a period of 40 seconds whereas Input 2 is run for a period of 20 seconds.

Input 1

8		22	
3	4	2	45
0	2	55	57
4	6	22	36
6	7	19	23
5	7	42	60
3	6	19	23
1	3	32	52
4	7	24	39
4	5	10	56
2	7	17	33
1	6	27	60
0	4	6	34
3	5	4	52
1	7	40	43
2	6	38	51
1	2	43	49
2	3	14	35
2	4	19	57
1	4	20	28
0	1	1	18
5	6	34	56
3	7	9	14

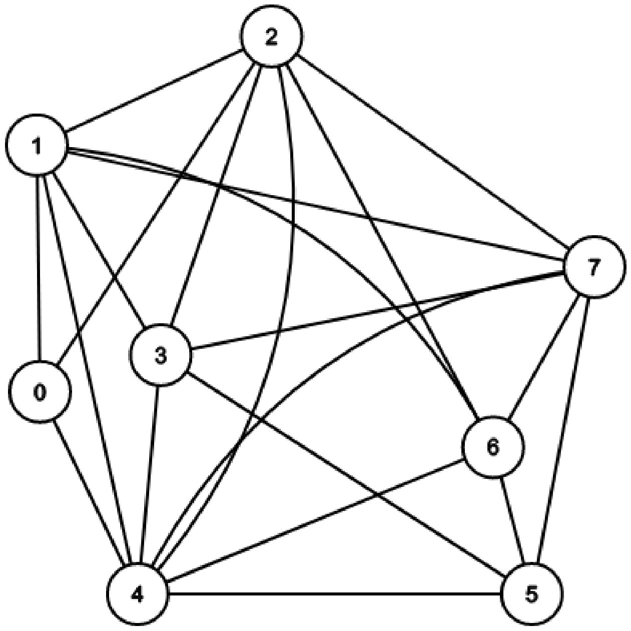

Fig. 1: Input 1

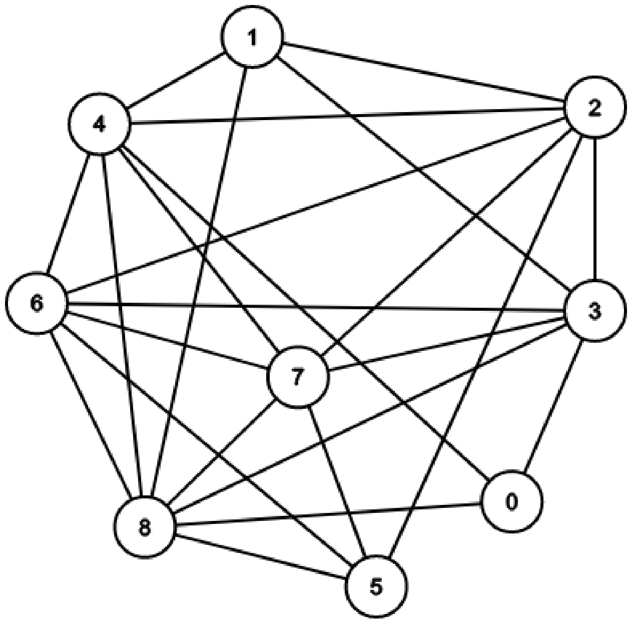

Fig. 2: Input 2

Input 2

9		24	
4	7	5	23
0	8	23	24
5	7	9	14
3	6	10	28
7	8	44	48
0	4	2	5
6	8	1	30
1	2	29	47
4	6	15	56
5	8	23	38
1	8	7	14
3	8	31	50
1	3	31	43
2	5	4	44
0	3	3	60
2	4	55	58
6	7	11	23
5	6	27	32
2	3	20	44
2	6	2	24
4	8	38	59
3	7	9	50
1	4	13	30
2	7	41	60

$$hi = 1$$
$$lsai = 5$$
$$spfi = 20$$

- We observe that the shortest paths have been computed by each node to all the other nodes at regular intervals and the path costs vary with each interval as every hello message leads to the nodes generating a new cost for the corresponding links.
- The hello message frequency is directly proportional to the dynamicity of the network. If the hello message is being sent at a high frequency then the weights of the links keep changing more frequently. On the other hand if the frequency of the hello message is low then the weights of the links are relatively stable. We can also stop the hello messages for a certain period to allow the network weights to stabilize.
- The density of the graph is related to the average speed of propagation of LSA information. A dense graph leads to LSA information propagating faster as the maximum number of hops between any two nodes remain relatively less. For example, our graph in Input 1 is quite dense and the maximum number of hops in any path is 3. On the other hand, in a sparse graph, the maximum number of hops can be high and hence LSA information propagation can be slow.
- We observe that in certain scenarios the path to node B computed by node A is different from the path to node A computed by node B even though the graph is taken to be

undirected in nature (i.e., the link is cost is same in both direction for travel between two nodes). This might be because of the delay in propagation of the LSA messages to all the packets before shortest path is computed. For example if the HELLO interval is 1 second, LSA interval is 5 seconds, and OSPF interval is 20 seconds, then at the 19th second certain new costs will be generated by hello messages. However, since the LSA interval is 5 seconds the new link state would not propagate to the other nodes in the network before the shortest path is computed, and hence this may lead to interesting consequences like the one mentioned above.

A. *Output Tables - Input 1*

Routing Table for Node No. 0 at Time 20

Destination	Path	Cost
1	0-1	17
2	0-2	56
3	0-4-3	48
4	0-4	29
5	0-4-5	45
6	0-4-6	51
7	0-1-7	57

Routing Table for Node No. 0 at Time 40

Destination	Path	Cost
1	0-1	14
2	0-2	57
3	0-4-3	48
4	0-4	29
5	0-4-5	64
6	0-1-6	55
7	0-1-7	55

Routing Table for Node No. 1 at Time 20

Destination	Path	Cost
0	1-0	17
2	1-2	43
3	1-3	33
4	1-4	20
5	1-4-5	36
6	1-4-6	42
7	1-7	41

Routing Table for Node No. 1 at Time 40

Destination	Path	Cost
0	1-0	14
2	1-2	43
3	1-4-3	41
4	1-4	22
5	1-4-5	57
6	1-6	32
7	1-7	42

Routing Table for Node No. 2 at Time 20

Destination	Path	Cost
0	2-0	56
1	2-1	43
3	2-3	22
4	2-3-4	41
5	2-3-5	55
6	2-6	42
7	2-7	27

Routing Table for Node No. 2 at Time 40

Destination	Path	Cost
0	2-0	57
1	2-1	43
3	2-3	16
4	2-3-4	35
5	2-3-5	60
6	2-3-6	35
7	2-3-7	27

Routing Table for Node No. 3 at Time 20

Destination	Path	Cost
0	3-1-0	50
1	3-1	33
2	3-2	22
4	3-5-4	28
5	3-5	12
6	3-6	20
7	3-7	13

Routing Table for Node No. 3 at Time 40

Destination	Path	Cost
0	3-1-0	57
1	3-1	43
2	3-2	16
4	3-4	39
5	3-5	36
6	3-6	19
7	3-7	12

Routing Table for Node No. 4 at Time 20

Destination	Path	Cost
0	4-0	16
1	4-1	20
2	4-2	48
3	4-5-3	31
5	4-5	19
6	4-6	25
7	4-7	28

Routing Table for Node No. 4 at Time 40

Destination	Path	Cost
0	4-0	29
1	4-1	22
2	4-2	39
3	4-7-3	37
5	4-5	49
6	4-6	35
7	4-7	26

Routing Table for Node No. 5 at Time 20

Destination	Path	Cost
0	5-4-0	35
1	5-4-1	44
2	5-3-2	34
3	5-3	12
4	5-4	19
6	5-3-6	32
7	5-3-7	24

Routing Table for Node No. 5 at Time 40

Destination	Path	Cost
0	5-4-0	78
1	5-4-1	71
2	5-3-2	52
3	5-3	36
4	5-4	49
6	5-6	45
7	5-7	43

Routing Table for Node No. 6 at Time 20

Destination	Path	Cost
0	6-1-0	44
1	6-1	27
2	6-2	42
3	6-3	20
4	6-4	31
5	6-3-5	33
7	6-7	19

Routing Table for Node No. 6 at Time 40

Destination	Path	Cost
0	6-1-0	46
1	6-1	32
2	6-3-2	35
3	6-3	19
4	6-4	35
5	6-5	45
7	6-7	23

Routing Table for Node No. 7 at Time 20

Destination	Path	Cost
0	7-4-0	49
1	7-1	42
2	7-2	17
3	7-3	12
4	7-4	33
5	7-3-5	25
6	7-6	19

Routing Table for Node No. 7 at Time 40

Destination	Path	Cost
0	7-4-0	55
1	7-1	41
2	7-2	22
3	7-3	12
4	7-4	26
5	7-5	43
6	7-6	23

*B. **Output Tables - Input 2***

Routing Table for Node No. 0 at Time 20

Destination	Path	Cost
1	0-4-1	26
2	0-4-7-5-2	41
3	0-4-7-3	29
4	0-4	2
5	0-4-7-5	29
6	0-4-7-6	31
7	0-4-7	17
8	0-8	24

Routing Table for Node No. 1 at Time 20

Destination	Path	Cost
0	1-4-0	24
2	1-2	34
3	1-3	34
4	1-4	22
5	1-8-5	43
6	1-8-6	39
7	1-4-7	37
8	1-8	10

Routing Table for Node No. 2 at Time 20

Destination	Path	Cost
0	2-5-7-4-0	36
1	2-1	34
3	2-5-7-3	31
4	2-5-7-4	34
5	2-5	7
6	2-6	24
7	2-5-7	19
8	2-5-8	40

Routing Table for Node No. 3 at Time 20

Destination	Path	Cost
0	3-0	3
1	3-0-4-1	27
2	3-2	32
4	3-0-4	5
5	3-0-4-7-5	32
6	3-6	16
7	3-0-4-7	20
8	3-0-8	26

Routing Table for Node No. 4 at Time 20

Destination	Path	Cost
0	4-0	4
1	4-1	17
2	4-7-5-2	28
3	4-0-3	7
5	4-7-5	21
6	4-0-3-6	23
7	4-7	9
8	4-0-8	27

Routing Table for Node No. 5 at Time 20

Destination	Path	Cost
0	5-7-4-0	22
1	5-7-4-1	35
2	5-2	17
3	5-7-4-0-3	25
4	5-7-4	18
6	5-7-6	23
7	5-7	9
8	5-8	30

Routing Table for Node No. 6 at Time 20

Destination	Path	Cost
0	6-3-0	14
1	6-8-1	27
2	6-2	9
3	6-3	11
4	6-3-0-4	18
5	6-2-5	26
7	6-7	18
8	6-8	17

Routing Table for Node No. 7 at Time 20

Destination	Path	Cost
0	7-4-0	14
1	7-4-1	27
2	7-5-2	26
3	7-3	13
4	7-4	10
5	7-5	9
6	7-6	18
8	7-4-0-8	37

Routing Table for Node No. 8 at Time 20

Destination	Path	Cost
0	8-0	24
1	8-1	9
2	8-6-2	39
3	8-0-3	27
4	8-1-4	26
5	8-5	34
6	8-6	30
7	8-1-4-7	36

IV. Learnings

This assignment helped us understand the celebrated the Open Shortest Path First (OSPF) routing protocol in great detail. We understood the process by which the nods are able to construct the entire network topology with the help of link state advertisement packets. The assignment also familiarized me with multithreaded programs in `C++`. Writing multithreaded programs brings with it many challenges like synchronization, which we need to handle with the help of structures like mutexes.

V. Additional Thoughts

- When multiple paths to a destination have the same cost, OSPF allows multiple paths to be used (that is, a single path need not be chosen for carrying all traffic when multiple equal-cost paths exist). Finding all the paths can lead to exponential complexity. We can try to limit the number of paths to say a fixed small number like 5 and find them.
- We can implement authentication with the use of an algorithm like MD5. MD5 authentication is based on shared secret keys that are configured in all the routers. For each OSPF packet that it sends, the router computes the MD5 hash of the content of the OSPF packet appended with the secret key. Then the router includes the resulting hash value in the OSPF packet. The receiving router, using the preconfigured secret key, will compute an MD5 hash of the packet and compare it with the hash value that the packet carries, thus verifying the packet's authenticity. Sequence numbers are also used with MD5 authentication to protect against replay attacks.
- The Border Gateway Protocol (BGP) which is the inter-autonomous system protocol, can be implemented for getting an idea of how the internet works at the large scale.

VI. Conclusion

Thus the simulation of the the Open Shortest Path First (OSPF) routing protocol has been completed successfully. It's performance under various network topologies have been observed. We observe that the results of the experiment correlate well with the desired goal of the protocol.

References

[1] Thread - C++ standard library reference documentation

[2] Kurose, J. F., Ross, K. W. (2016). Computer Networking: A Top-Down Approach. Boston, MA: Pearson. ISBN: 978-0-13-359414-0