CSC358 Assignment 2 Report

Gongzhi Wang, Kangzhi Gao, Yi Gao

Objectives

The main objective of this report is to 1. Describe the implementations of a multi network router with two routing algorithms, namely, Routing Information Protocol (RIP) and Open Shortest Path First (OSPF). 2. Discuss the design choices we made when choosing data structures and how they affect the efficiency of our code. 3. Showcase the topologies used and our methods of testing. 4. Analyze the performances of each algorithm under different networks.

Implementation

Host/Client

-

Common Router Functionalities (For Both RIP and OSPF)

Socket Creation:

- For each interface, the router will initialize two types of sockets, a UDP socket that is receiving a UDP broadcast message from its respective subnet and a TCP socket that is waiting for connection from other routers or end systems.

Connect Host:

- When receiving the UDP broadcast message from an end system (let's call it a host for the rest of the section), the router will send back its inet ip address to the host and wait to accept the host to connect with a TCP socket.

Forwarding Message:

- When the router receives a TCP packet from one of its TCP sockets (either forwarded by other routers or sent by an end system), the router will extract the packet's destination and TTL. If the destination ip/port is in the router's forwarding table and the TTL is greater than 0 after decremented by 1, then the router will forward the packet to the corresponding interface by using the interface's TCP socket with TTL decremented by 1. If the destination ip/port is not in the router's forwarding table or the TTL after decremented by 1 equal to 0, the router will drop the packet.

RIP

Advertise

- For each router, the router constantly broadcasts its forwarding table to neighbor routers every 30 seconds.

Get Advertised

- For each router, it will constantly check if other routers send broadcast messages in each subnet. If an interface catches a forwarding table sent by other routers, the router will add the host destination ip in the forwarding table as a key into its forwarding table with the interface that gets the forwarding table and the distance to the host as the value . For example, r2 sends its forwarding table to r1 with the information {h1: [xxx.xxx.xxx.xxx, 1]}, then r1 will add h1 to its forwarding table as {h1: [yyy.yyy.yyy.yyy, 2]} where yyy.yyy.yyy.yyy is the interface ip that receives r2's forwarding table.

OSPF

There are two components when routing with OSPF: a router and a monitor node.

Router

get_neighbour()

- In OSPF, routers have full topology knowledge, therefore each router will broadcast to and listen from its neighbours and record their information. Each router will also keep a record of its hosts, so other routers would know which router to send to when they get a host ip address.

send_forwarding_table()

- After preparing the forwarding table, the router will broadcast to the monitor and once it becomes writable, the router sends forwarding tables to the monitor.

get_forwarding_table()

- The router receives updated forwarding tables from the monitor, as soon as it becomes readable. Router will only update its forwarding table when it is different from its own forwarding table.

In the main function, given an ip, the router will use its forwarding table to either forward it to other routers, or send it to its switch.

Monitor

send_and_receive()

- Similar to send and get forwarding tables in OSPF router, send_and_receive() receives forwarding tables from all routers and passes them into set_routing_table, which calls appropriate functions to process them.

process_forwarding_table()

- This function maps incoming ip address to a string representing a router, i.e. “r1”. Then, the string is added to the routing table, at which time, this string will be checked against each router in the routing table to find if there is a “connection”. A connection is found when the other router is in this router’s list of neighbours (from forwarding table), and this router is in theirs. If there is a connection, these two routers will add each other in the routing table. For example: the existing routing table is:
  {'r1': ['r2'], 'r2': ['r1']}, if we added “r3” to this routing table, and r3 is connected to r1, the new routing table would be:
  {'r1': ['r2', ‘r3’], 'r2': ['r1'], ‘r3’:[‘r1’]}

dijkstra()

- This function runs dijkstra’s algorithm to find the shortest path to each node given the routing table. The result table is computed_routing_table. This table follows the format: {node: {target: neighbour}}, where target is the destination node, and neighbour is the node you need to go next to reach target node. The computed_routing_table for example above is:
  {‘r2': {'r1': 'r1', 'r3': 'r1'}, ‘r1’: {‘r2’: ‘r2’, ‘r3’: ‘r3’}, ‘r3’: {‘r1’: ‘r2’, ‘r2’:’r1’}}

routing_table_to_send()

- Finally, we unpackage the router string into receiving ip address, and map host ips to the next routes we need to send. We know the next router because we have destination ip of the host from the forwarding table and routing table, computed_routing_table[current_router][destination_ip] gives us the ip of the next router. Below is one example of routing table sent:

```
Routing table to send: {'11.2.11.2': {'h2': '10.104.0.2', 'h3': '10.104.0.2', 'h1': '10.104.0.2'},
```

Design Choice

RIP

In RIP, routers' forwarding table follows the format of {host_ip:(interface_ip, distance)}. The distance for the hosts that the router owns is always 0 initially. And for each time the router gets an advertisement from other routers, the distances of the corresponding hosts increase by one.

OSPF

The time complexity of Dijkstra's algorithm is approximately O(V^2), V being the number of routers. Ideally, we would like that to be the upper bound complexity of OSPF monitor. Since a lot of conversions happen in the monitor, i.e. forwarding_table -> router string, router string -> source ip, router -> hosts, connections between each router… If we were to loop through all forward tables every time we want to find something, the complexity would be through the roof. Instead, we use numerous dictionaries to map every conversion we need. We trade space for time because we believe time is more valuable for a router.

Testing

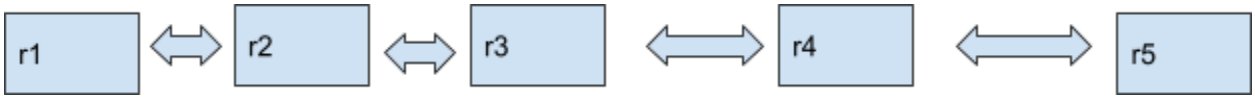
For every topology, we have two versions, one for RIP and OSPF. The OSPF version has an additional host called m0 as the monitor node.

What we are testing:
To test out the time it takes to generate forwarding tables for every router, we record the difference in the time of the last modified forwarding table and the time of the next update for each router.

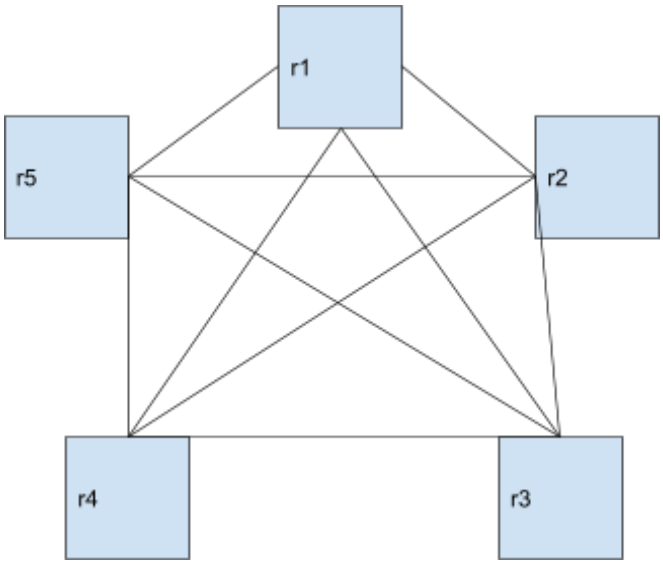
How we are testing:
For testing in OSPF, we first add all routers but one, after they are online, we add the last router. Reasoning: in OSPF, forwarding tables will be sent as soon as there is a change in topology, this way, all routers will have the same starting time.
For testing in RIP, we first add all routers, and have a host from one router to send a message to a router that is the farthest from it (in terms of hops). For example, in topology one, host from r5 will send a message to host in r1.

Topologies:

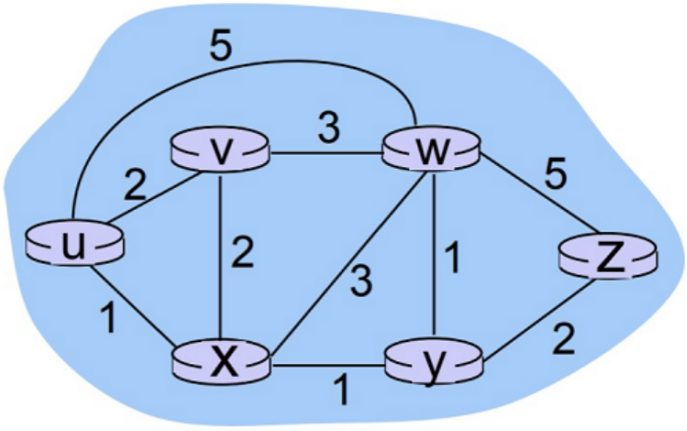
1) Straight Lines



2) Star Topology



3) Random Topology From Lecture



Performance (in seconds)

Topology One

RIP:	OSPF:
From r5's host	Adding r5's host
r1: 139.6	r1: 31.8
r2: 100.5	r2: 31.6
r3: 66.9	r3: 31.7
r4: 53.06	r4: 26.8

In topology one, OSPF completely outperforms RIP. From the chart, in RIP, we can see that r1 takes the most time receiving a forwarding table and r4 takes the least. This is expected, as topology one is designed to target RIP's slow convergence, in other words, if there are multiple hops between source and destination, routers in RIP would need to wait for its neighbour to propagate the forwarding table, whereas in OSPF, each router will only receive forwarding table from the monitor node. Therefore it makes little difference in time when it comes to different topologies.

Topology Two

RIP:	OSPF:
From r5's host	Adding r1
r1: 44.0	r2: 37.3
r2: 40.0	r3: 30.3
r3: 40.0	r4: 30.2
r4: 40.0	r5: 30.2

As opposed to topology one, we designed topology two as the ideal network for RIP: each node in the network is connected to every other node, therefore it takes only one hop to reach any node. As we see from the data, each router in OSPF has similar time in delay for the reason discussed above, but also, every router in RIP not only has similar time but also has a significant improvement in performance. It is also worth noting that as we grow the network bigger (under the same structure, where every node is connected), RIP will outperform OSPF, because the time it takes for a hop remains the same, but the monitor node in OSPF will need more time to calculate the forwarding table, because the network is bigger.

Topology Three (Common)

r1 = u
r2 = v
r3 = w
r4 = x
r5 = y
r6 = z

RIP:	OSPF:
From r6's host	Adding r1
r1: 70.1	r2: 25.2
r2: 41.6	r3: 35.3
r3: 25.6	r4: 30.3
r4: 41.5	r5: 30.7
r5: 14.0	r6: 26.7

We get topology three from the lecture, this topology is not designed in either one's favour, we use it to test the average performance of the two algorithms. The pattern in time is the same as above. The average for RIP is around 38 seconds, and the average in OSPF is around 30 seconds. OSPF is slightly better than RIP.

Conclusion
Under small networks:
RIP will be better under a connected network where each node, or almost every node, is connected. And it scales well (given it is still a small network) compared to OSPF. However, OSPF will outperform RIP if the network is not well connected, as OSPF does not really care about network topology and RIP does.
For big networks:
OSPF will be better almost every time. RIP has a hop limit of 15, and it uses bandwidth every 30 seconds whereas OSPF has no hop limits and only updates when there is a change in topology.