Comparison of OSPF and RIP Routing Algorithms

CSC358 Assignment 2

Danyal Khan, Anna Liang, Haidder Sajjad

April 4th, 2022

Table of Contents

Table of Contents **2**

Introduction **3**

Multinet - OSPF 4

Multinet - RIP 8

Summary of Design Decisions & Assumptions: 9

Methodology **10**

Results **11**

Analysis **12**

Expectations 13

Conclusion **14**

References **15**

Introduction

Routers are nodes in the network that can connect to multiple end systems, and allow those end systems to send messages to one another by appropriately forwarding data. When a router connects to another router, it must first initialize in a manner similar to an end system. However, once the initial connection is made, the router now needs to know which packets should be sent to the new router. This is where routing protocols come into play. A routing protocol describes the rules that must be followed by a router while it interacts with neighboring routers to learn the path and to maintain the network in the routing tables. This report compares the performance between the two routing protocols, Routing information Protocol (RIP) and Open Shortest Path First (OSPF).

Simple End System and Simple Router:

Read the README on instructions on how to run the simple end system and simple router.

Given any router on the network, r<n> (e.g. r1), run the following commands on the mininet terminal to get the following functionalities:

- Get forwarding table: r<n> route
  - Or on the router terminal:
    - xterm r<n>
    - route
- Set forwarding table: r<n> ip route add <ip address to route> via <router interface ip to send to> dev <interface>
  - e.g: ip route add 10.2.0.0/24 via 10.0.6.2 dev r1-eth2
    - Routes all ip's destined for 10.2.0.0 to adress 10.0.6.2 on the interface r1-eth2
  - On the router terminal
    - xterm r<n>
    - ip route add <ip address to route> via <router interface ip to send to> dev <interface>
- Print forwarding table: r<n> route
  - Or on the router terminal:
    - xterm r<n>
    - route

Note: The advertise and update functionalities were not implemented for RIP since our RIP algorithm is based on a centralized monitor node, similar to what was used in the OSPF implementation.

While this is not an authentic RIP implementation, the goal was to implement as many RIP features given the time constraints and tools used

Multinet - OSPF

Running OSPF:

To run the Multinet OSPF algorithm, first cd into the *multi_net_router* directory and run "*sudo python multi_net_OSPF.py*". Instead of using something like a monitor node that periodically gets the topology from the routers and updates all their routing tables, we emulate this 'monitor node' in the *runOSPF* function in multi_net_OSPF.py.

Implementation:

When running *multi_net_OSPF.py*, it first creates the topology (using NetworkTopo class) in the run function. Then runs the script for each router, *scripts_for_router<n>.sh* where <n> is a router number. The script sets up the router by configuring each interface with a MAC address and ip address and enabling broadcasting and ipv4 forwarding. Then a new process is forked which runs the *runOSPF* function every 30 minutes.

The *runOSPF* function is a dynamic function in the sense that it can be run on any topology and set the correct forwarding tables for each router given the conditions that: all routers start with a lowercase r, all hosts start with a lowercase h, all switches start with a lowercase s, and there are maximum 255 hosts per router. The *runOSPF* function maintains a set of data structures which holds the topology of the network. Using these various data structures the function runs a BFS on every router pair, and for the first router on the path to the destination it adds the routing information in the source router's forwarding table.

Algorithms Used:

Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. The reason this was done is because all the links or paths have the same weight. And if all link costs are equal, **Dijkstra = BFS**. Hence there is no need to do a weighted search.That's part of our assumptions. If they're uniform, that means we don't have to account for the weight of the link, because they're all the same. If we're doing nonuniform, then we'd use Dijkstra's algorithm, but in our case and assumption, that is not needed.

Data Structures Used:

We used several data structures in *multi_net_OSPF.py*, most of these were maps, and were used in setting up a representation of the network topology. These include: map_of_routers_and_links, cached_router_paths, switches_under_router, hosts_under_routers, all_hosts, all_routers, etc.

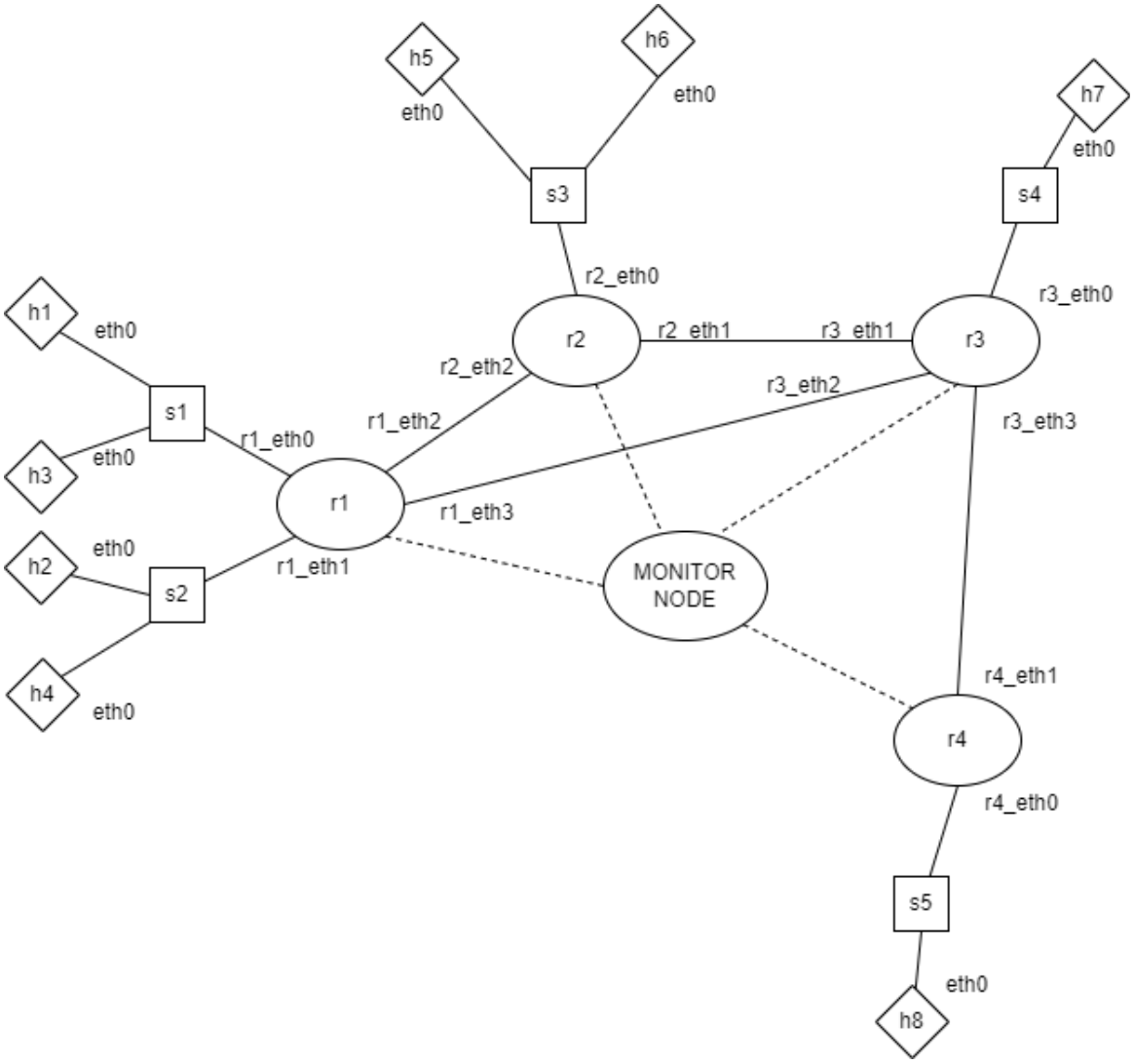
Hash maps are indexed data structures that were used in the BFS. A hash map makes use of a hash function to compute an index with a key into an array of buckets or slots. Its value is mapped to the bucket with the corresponding index. The first hash map was used to store the actual graph, so like a hashmap of the overall topology. Routers were the key and the value was the list of routers it's connected to. Hashmaps were used again to store the cached router paths. In this hashmap, the key is two routers, router1 and router2 and the values are the path from router 1 to router 2.

A **Queue** is a linear data structure that stores items in First In First Out (FIFO) manner that was also used in the BFS. With a queue, the least recently added item is removed first. The use of it in the BFS algorithm was used to keep track of the locations remaining to be searched in the traversal of the graph in the BFS.

Testing:

To test the multinet OSPF program run *sudo python multi_net_OSPF.py* in the multi_net_router directory, then run *xterm h1* and *xterm h8*. On both host terminals run ifconfig, and on the h1 terminal run *nc -l 4567*. Then on the h8 terminal run *nc 10.0.1.10 4567 < test_snd*. This sends a message from host 8 (h8) to host 1 (h1) on different routers.

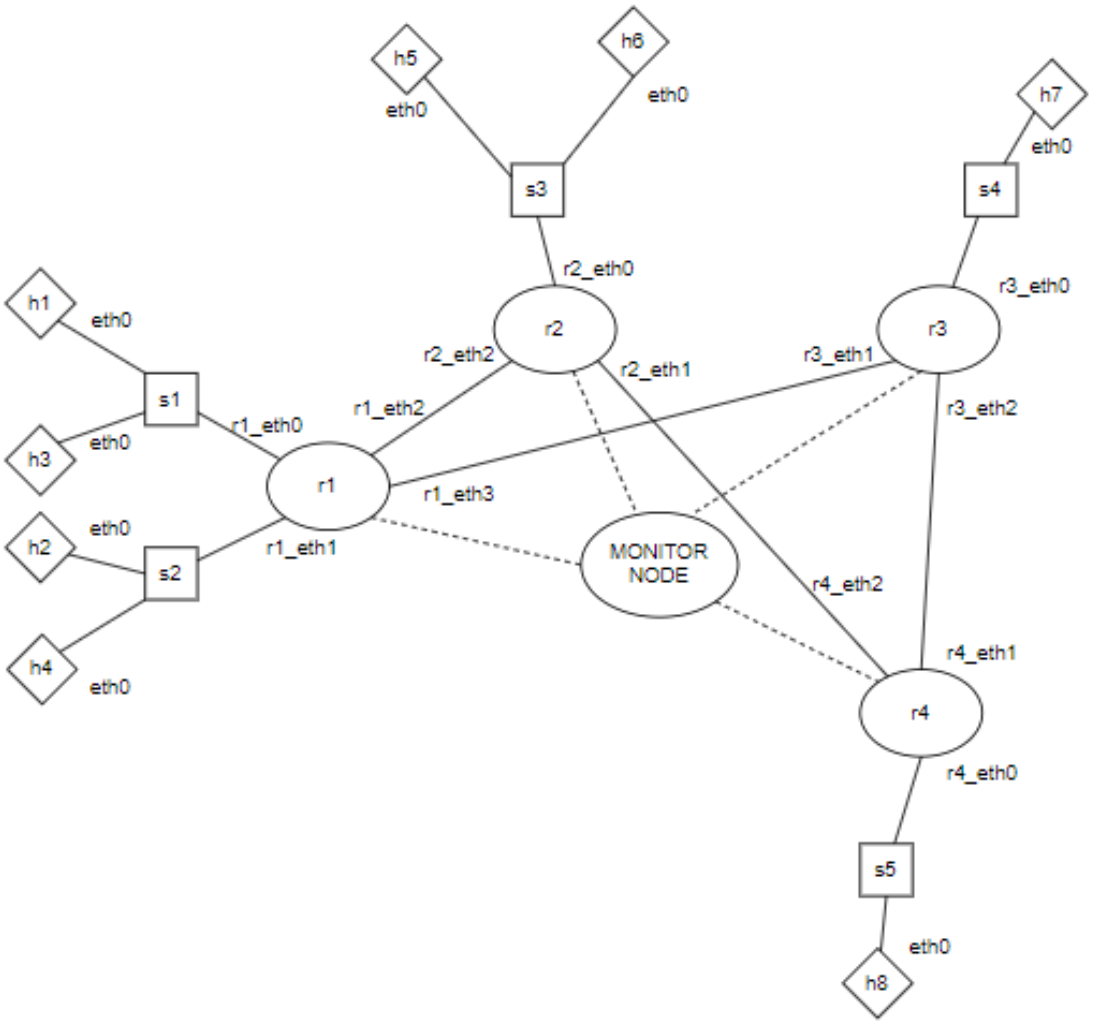
Here is a full network topology used in multi_net_OSPF.py. The program also prints out all the router's forwarding tables initially and after running the algorithm to see how they changed:



And the ip-addresses at each interface:

Routers	eth0	eth1	eth2	eth3
r1	10.0.1.1/24	10.0.2.1/24	10.0.6.1/24	10.0.3.1/24
r2	10.2.0.1/24	10.0.4.1/24	10.0.6.2/24	N/A
r3	10.3.0.1/24	10.0.4.2/24	10.0.3.2/24	10.0.5.1/24
r4	10.4.0.1/24	10.0.5.2/24	N/A	N/A

We added a second topology to test with, in the multi_net_router2 directory named multi_net_OSPF2.py.
To run, do: *sudo python multi_net_OSPF2.py*.
Here is it's graph and ip addresses for each interface:



Routers	eth0	eth1	eth2	eth3
r1	10.0.1.1/24	10.0.2.1/24	10.0.6.1/24	10.0.3.1/24
r2	10.2.0.1/24	10.0.4.1/24	10.0.6.2/24	N/A
r3	10.3.0.1/24	10.0.3.2/24	10.0.5.1/24	N/A
r4	10.4.0.1/24	10.0.5.2/24	10.0.4.2/24	N/A

Multinet - RIP

Running OSPF:

To run the Multinet RIP algorithm, first cd into the *multi_net_router* directory and run "*sudo python multi_net_RIP.py*". Instead of having routers advertise updates to their routing tables and then update their own tables based on advertisements received, we implemented RIP using a centralized controller. This 'monitor node' was emulated in the *runRIP* function in multi_net_RIP.py.

Algorithms Used:

Bellman-Ford algorithm, is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. This may sound similar to BFS/Dijkstra's algorithm that was used in OSPF routing protocol to achieve the same goal, however Bellman-Ford using dynamic programming and also works when there is a negative weight edge, it also detects the negative weight cycle.

Data Structures Used:

Hash maps are indexed data structures that were used in the BFS. A hash map makes use of a hash function to compute an index with a key into an array of buckets or slots. Its value is mapped to the bucket with the corresponding index. The first hash map was used to store the actual graph, so like a hashmap of the overall topology. Routers were the key and the value was the list of routers it's connected to. Hashmaps were used again to store the cached router paths. In this hashmap, the key is two routers, router1 and router2 and the values are the path from router 1 to router 2.

Implementation:

When running *multi_net_RIP.py*, it first creates the topology (using NetworkTopo class) in the *run* function. Then runs the script for each router in the network, scripts_for_router<n>.sh where <n> is a router number. These scripts are the same ones used for OSPF. Then a new process is forked which runs the runRIP function every 30 seconds.

The runRIP function is can be run on any topology and can set the correct forwarding tables for each router given the following conditions: all routers start with a lowercase r, all hosts start with a lowercase h, all switches start with a lowercase s, and there are maximum 255 hosts per router. The function runs the Bellman-Ford algorithm on every router, and for the first router on the path to all other routers in the network, it adds the routing information in the source router's forwarding table. However, if the distance (i.e. hop count) exceeds 15, the path is not added and the destination router is deemed unreachable.

Testing:

To test the multinet RIP program, run *sudo python multi_net_RIP.py* in the multi_net_router directory, then run *xterm h1* and *xterm h8*. On both host terminals run ifconfig, and on the h1 terminal run *nc -l 4567*. Then on the h8 terminal run *nc 10.0.1.10 4567 < test_snd*. This sends a message from host 8 (h8) to host 1 (h1) on different routers.

The topologies used are the same as the ones for OSPF, so that time testing will be on the same topologies. The program also prints out all the router's forwarding tables initially and after running the algorithm to see how they changed. To run the second topology, cd into the multi_net_router2 directory and run: sudo python multi_net_RIP2.py

Summary of Design Decisions & Assumptions:

OSPF:

Open Shortest Path First routing algorithm is implemented using Dijkstra's shortest path algorithm. All links or paths have the same weight, and if all link costs are equal, **Dijkstra = BFS.** As a weighted BFS is essentially Dijkstra's, but because we're assuming all link costs have the same weight, we implemented OSPF using BFS.

RIP:

Routing Information Protocol is normally a decentralized routing algorithm, meaning the calculation of the least-cost path is carried out in an iterative manner by the routers. No node has complete information about the costs of all network links. Instead, each node begins with only the knowledge of the costs of its own directly attached links. Only then, through an iterative process of calculation and exchange of information with its neighboring nodes, a node gradually calculates the least-cost path to a destination. This is called distance-vector algorithm, because each node maintains a vector of estimates of the costs to all other nodes in the network. However, we took a more centralized approach. So our RIP algorithm is based on a centralized monitor node..

Methodology

There are two aspects that can be analyzed when comparing the performance between the two routing protocols, 1) the time required to populate the routing tables and 2) the time required to send the messages through the topology. The time required to send messages through your topology is not going to be analyzed as an assumption is that all link costs are uniform meaning there will not be a difference in bandwidth and transmission rate between the two routing protocols. As mentioned on Piazza, because we did not attempt the bonus component of non-uniform costs, this analysis is simply going to demonstrate that the forwarding tables are identical between the two protocols, and thus the only difference is the time needed for the table of computation. Hence, the primary form of analysis between becoming performance will be observing and examining the time to compute the forwarding tables between OSPF and RIP. The python `time` module allows us to work with time in Python and includes functionality like getting the current time and pausing the Program from executing.

To minimize external variables, the tests were all performed on a WSL-Ubuntu-20.04 terminal on MobaXTerm, using an ASUS laptop that has the following specifications:

- Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.21 GHz
- 16.0 GB RAM
- 64-bit operating system, x64-based processor

The performance statistics collected:

- **OSPF Forwarding Table Computation Time:** (seconds) The amount of time required in seconds to compute the forwarding table using Open Shortest Path First routing protocol
- **RIP Forwarding Table Computation Time:** (seconds) The amount of time required in seconds to compute the forwarding table using Routing Information Protocol.

Results

Each test described in the methodology section was executed 10 times . Table 1 shows the mean of the forwarding table computation times collected for each routing algorithm and topology.

	OSPF Forwarding Table Computation Time (seconds)	RIP Forwarding Table Computation Time (seconds)
multi_net_topo1	0.07072174549102783	0.09241776466369629
multi_net_topo2	0.07570197582244872	0.13939297199249268

Table 1. Comparing forwarding table computation times

Analysis

The two protocols that were implemented were Routing information Protocol (RIP) and Open Shortest Path First (OSPF).

RIP, is an example of distance vector routing for local networks. RIP works to deliver the whole routing table to all active interfaces every 30 seconds. In RIP protocol, hop count is the only metric to decide the best path to a remote network. RIP is implemented using the bellman ford algorithm.

OSPF (Open Shortest Path First), a link-state routing protocol. OSPF routing protocol collects link state information from routers in the network and determines the routing table information to forward packets. This occurs by creating a topology map for the network. OSPF is implemented using dijkstra's algorithm.

From table 1, it can be observed that on average, through both multi network topologies, OSPF computed its routing tables faster then RIP. For the first topology, OSPF ran on average ~0.02 seconds faster than RIP and on average ~ 0.06 seconds for the second topology. It can can also be observed that time needed to compute the forwarding tables using the OSPF algorithm stayed close to the same whereas with the RIP, there was ~50% increase in time from topology #1, to topology #2. A potential reason as to why OSPF takes quicker to compute the forwarding tables is that the underlying algorithm used in its implementation, Dijkstras, has a better runtime as the time complexity is O(E logV) where E is the number of edges and V is the number of vertices in the graph. The Bellman-Ford algorithm, the underlying algorithm used in the implementation of RIP, has a slower runtime at O(VE).

Overall, the underlying algorithm used in OSPF and RIP may accomplish the same goal as both are single-source shortest path algorithms, i.e. both determine the shortest distance of each vertex of a graph from a single source vertex. However, there were some differences that were discovered that made cases as to why one was better than the other. To begin with, unlike RIP, OSPF only exchanges routing information when there's a change in network topology or at longer intervals such as 30 minutes. It effectively calculates the shortest path with minimum network traffic when the change occurs. This is a disadvantage for RIP as it checks with its neighboring routers every 30 seconds, which increases network traffic. Another disadvantage is that RIP has a maximum hop count of 15, which means that on large networks, routers may not be able to be reached. This is why OSPF protocol best fits complex networks that comprise multiple subnets working to ease network administration and optimize traffic whereas RIP protocol is a great fit for small networks as it's easy to understand and configure.

Conclusion

Results indicate that Open Shortest Path First has performed better than the Routing Information Protocol in that it is able to compute the forwarding tables faster. As mentioned before, OSPF performs better than RIP in terms of cost of transmission and is suitable for larger networks as there is no hop count. Although RIP is said to work better in smaller networks, since the sample size of the topologies was relatively small, it cannot be concluded with certainty. This also implies that our implementation of RIP was suboptimal as we did take a centralized approach as opposed to a decentralized approach that it's actually supposed to take. Most importantly, the underlying algorithms that the routing protocols are based on, favour OSPF as Dijkstra's shortest path algorithm has a faster runtime then Bellman-Ford algorithm that RIP uses. Finally, the performance tests were all conducted on one machine. While this was done to limit the effects of external variables affecting the data, perhaps testing on different machines with different system configurations would yield different results. For instance, OSPF is more intensive in processing power and RAM consumption.

Overall, there are many factors to consider when selecting a routing protocol, and when doing so, it is especially important to consider the consequences of a bad decision as things scale. When the network is small, RIP is simple and sufficiently fast, but as the size and complexity of a network scales up, RIP performs worse than OSPF. If there are more random, quick changes to topology than in our testing, OSPF may be better at responding to the change. Moreover, bad news travels slowly in distance vector algorithms like RIP. However, in practice, implementations of OSPF are more intensive in processing power and RAM consumption (Muaz and Kuruvikulam, 2020), so processing capacity is important to consider. We only tested relatively small topologies with little traffic. Finally, it may be important to consider how much memory a router can hold since OSPF may require more memory to store various copies of the routing table.

References

Cordeiro, Edwin. “Mininet Network with OSPF and BGP.” GitHub, 1 July 2016, https://github.com/edwinsc/mininet_ospf_bgp/blob/master/I2RS%20Hackathon%20-%20Mininet%20Network%20with%20OSPF%20and%20BGP.pdf

edoardesd. “(Mininet) How to Create a Topology with Two Routers and Their Respective Hosts.” Edited by Kohányi Róbert, Stack Overflow, 29 Apr. 2020, https://stackoverflow.com/questions/46595423/mininet-how-to-create-a-topology-with-two-routers-and-their-respective-hosts

Muaz, Abdulla, and Kuruvikulam, Chandrasekaran Arun. Performance Evaluation of Wireless Routing Protocols: RIP and OSPF in WLAN. Journal of Applied Technology and Innovation, Nov. 2020, https://dif7uuh3zqcps.cloudfront.net/wp-content/uploads/sites/11/2020/11/02172457/Performance-Evaluation-of-Wireless-Routing-Protocols-RIP-and-OSPF-in-WLAN.pdf.

RIP Disadvantages, csc.columbusstate.edu/summers/NOTES/cs458/chap04/tsld049.htm.