Final Project: Implementing a Simple Router

In the previous lab you implemented a simple firewall for allowing transport level packets through but disallowing network only traffic using the OpenFlow protocol. Now you will need to implement a simple router, that is to be able to route and forward packets through the network.

Introduction to Routing:

Routing is the mechanism for moving packets from one host to another through the network. Routing protocols take a graph and prune edges to make a tree. A graph is a collection of edges which connects vertices. A tree removes surreptitious edges from the graph. The mechanism of pruning edges from the graph depends on the tree's purpose. A common purpose, especially in computer science is to guarantee that every node has a minimum distance to a core node as done with a spanning tree. Algorithms discussed in lecture will cover Dijkstra and Bellman-Ford algorithms which use this concept to base their shortest paths. These algorithms are used to develop either Distance-Vector (RIP) or Link State routing protocols (OSPF). Further material is covered in cmpe151 and cmps102 where you will learn additional information about how these protocols. However, in this lab we will be implementing simple and static routes. Static routes are not dynamic, and do not have to follow any constraints such as shortest paths, they can be determined for any reason.

For static routes, the network administer implements routes for packets to be forwarded through, they do not change unless the network administrator decides to change the rules. This is opposite to more dynamic routing protocols which modify routes based on changes in link utilization, routers going down, or other factors. This enables dynamic routing protocols to adjust to network events. Unlike static routes, which in the case of errors in the network, maintain their predetermined forwarding.

Assignment:

Your job will be to write code using the OpenFlow protocol and using the mininet environment to implement simple static routing between the hosts in the environment. Special attention should be paid to what IP addresses reside in the private network address space and should not be routable through the network. However, for this lab, we are going to allow certain private address routes based on the topologies connections.

Your code will need to allow some private addresses to communicate across switches. For example in Figure 1, A and C should be able to communicate

because they have the same private address space connecting their local router (reminder that in mininet they are really switches, but we will logically think of them as routers). A and B, and C and B should not be able to communicate because they have separate private address spaces that span across routers. Private address hosts should only be allowed to communicate across routers if both routers have a host in the same private address. However, that does not mean that if another Host X was attached to 1, that was not in the 10.0.0.0/8 addresses space should be able to communicate because on the return path it would have a mismatch request for a private address behind another router.

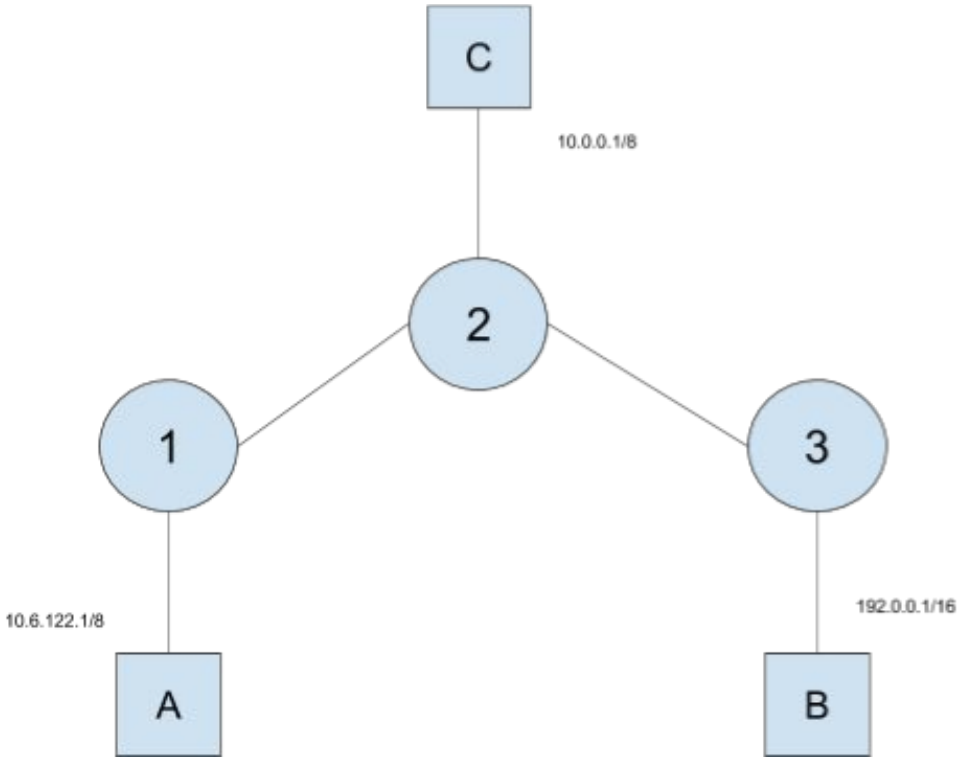

Figure 1: Simple Private Addressing Topology

In Figure 2, we have zoomed in from our previous figure. Even though there are multiple IP addresses that are both public and private IP addresses, because they are all connected to the same router there will be routes to the other nodes. So D, E, F, G, and H are all able to communicate. Combining Figures 1 and 2, we would say that nodes E and G in Figure 2 are able to communicate with B. However, other nodes such as D, F, and H would be unable to communicate with B. This is an arbitrary decision to allow private addresses to communicate with the same private address across routers, but to explicitly disallow public addresses to communicate with private addresses across router links.

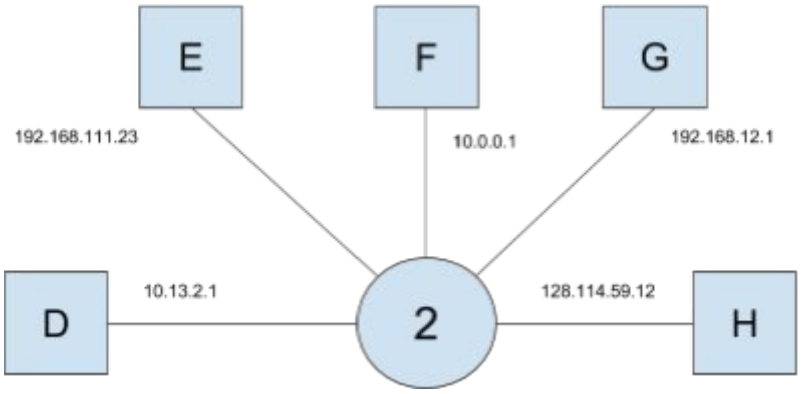

Figure 2: Single Router Topology

Another way to put this is that communication that transits connections between routers is only able to communicate with a private address if the requesting node is also in the same private address space.

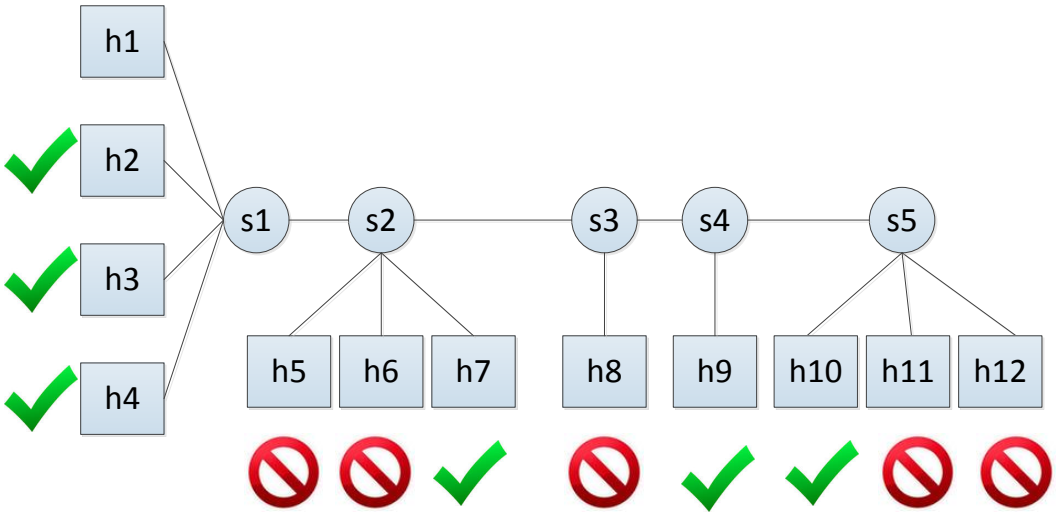

Figure 3: Project Topology

Based on the rules described above, we will now provide examples based on the topology that will be given to you in the final project. There will be a total of 12 hosts, with 5 switch/routers. Figure 3 shows all the hosts able to communicate with host 1. Host 1 is able to communicate with all of its local hosts (h2, h3, h4)

attached to the same switch router. It is also able to communicate with h7,h9 and h10 which shares the same associate private IP address. Because s1 and s2 both have local nodes with the same private IP address space, they are able to communicate. h1 is unable to communicate with h5 because they have mismatched private IP addresses. Host 6 is unable to communicate with host 1 because h6 is a public IP address and should not be able to communicate with the private IP address.
The full communication matrix is shown below using pingall.

Expected Output:

final_topo.py:

```
<Host h1: h1-eth0:10.0.0.2 pid=X>
<Host h2: h2-eth0:192.168.0.20 pid=X>
<Host h3: h3-eth0:192.168.0.30 pid=X>
<Host h4: h4-eth0:192.168.0.40 pid=X>
<Host h5: h5-eth0:192.168.0.50 pid=X>
<Host h6: h6-eth0:172.16.0.2 pid=X>
<Host h7: h7-eth0:10.10.10.10 pid=X>
<Host h8: h8-eth0:172.16.5.2 pid=X>
<Host h9: h9-eth0:10.20.30.40 pid=X>
<Host h10: h10-eth0:10.11.12.13 pid=X>
<Host h11: h11-eth0:172.16.32.64 pid=X>
<Host h12: h12-eth0:192.168.0.60 pid=X>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None,s1-
eth4:None,s1-eth5:None pid=X>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None,s2-
eth4:None,s2-eth5:None pid=X>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None
pid=X>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None
pid=X>
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None,s5-eth3:None,s5-
eth4:None pid=X>
<RemoteController c0: 127.0.0.1:6633 pid=X>
```

pingall:

```
h1->   h2  h3 h4  X  X  h7  X  h9  h10  X  X
h2->h1     h3 h4 h5 X  X   X   X   X    X  h12
h3->h1 h2     h4 h5 X  X   X   X   X    X  h12
h4->h1 h2 h3     h5 X  X   X   X   X    X  h12
h5->X  h2 h3 h4     h6 h7  X  X   X    X  X
h6->X  X  X  X  h5     h7 h8 X   X    h11 X
h7->h1 X  X  X  h5 h6     X  h9  h10 X   X
h8->X  X  X  X  X  h6  X     X   X   h11 X
h9->h1 X  X  X  X  X   h7 X      h10 X   X
h10->h1 X  X  X  X  X  h7 X  h9      h11 h12
h11->X  X  X  X  X  h6 X  h8  X  h10     h12
h12->X  h2 h3 h4 h5 X  X  X   X  h10 h11
```

For your Demo with the TA you will need to verify your results with the above when using the **final_topo.py**. You should verify that you code does not cause duplicate acknowledgements and that rules are being adding dynamically (that is a route is only created when needed). These routes should be based on the physical port values. It is fine to have static routes that are based on these port values as long as they are also added dynamically. It will be a requirement that you are able to forward packets across routers appropriately.

If you are using POX, we provide you with some starter code: router.py. All you need to do is specify the logic for the 5 switches to determine how they should be routing.

Demo/Grading Rubric:

For this assignment you will be implementing and demonstrating static routes using OpenFlow. Unlike previous labs you will need to demonstrate your working code to the TA before final weeks. I f you finish your code early, you may presented to your TA before the final due date to be checked off. We will post times 2 additional lab sections specifically for check offs on the Wednesday and Thursday during the final weeks. It is recommended that you try and check off before then so as avoid 45 over students also trying to check off.

Grading Rubric:

❑ 40: Checked off to the TA

- ❑ No partial credit will be awarded

❑ 20: Routes are implemented based on ingress/egress ports not on OFPP_FLOOD

❑ 20: Switches routes should be added dynamically

❑ 20: No duplicate packets should be created or received

- ❑ 10 pts: <20 are created and/or handled
- ❑ 0 pts: > 20 duplicate packets

❑ 40: You are able to forward traffic across multiple switches

- ❑ No partial credit will be awarded

❑ 40: The correct behavior works on **final_topo.py**

- ❑ -4 for each incorrect entry

❑ 20: Code and README submitted with the project.

----- 200 POINTS ----

❑ Late submissions to eCommons will be subject to a -20 point deduction and a -3 points per hour after deduction of top of the static penalty. No exceptions will be made.

Deliverables:

router.*: Your router file should be router.py. You should make sure that your code has your name. Make sure that you demo your code to your TA as well as submitting your code to eCommons.

README: The README should again contain your name. If there were any discrepancies or notes to make please state them in the README. If you used a non-default controller, please state instructions for installing and using the controller as well as running your code.