GENI Lab 8

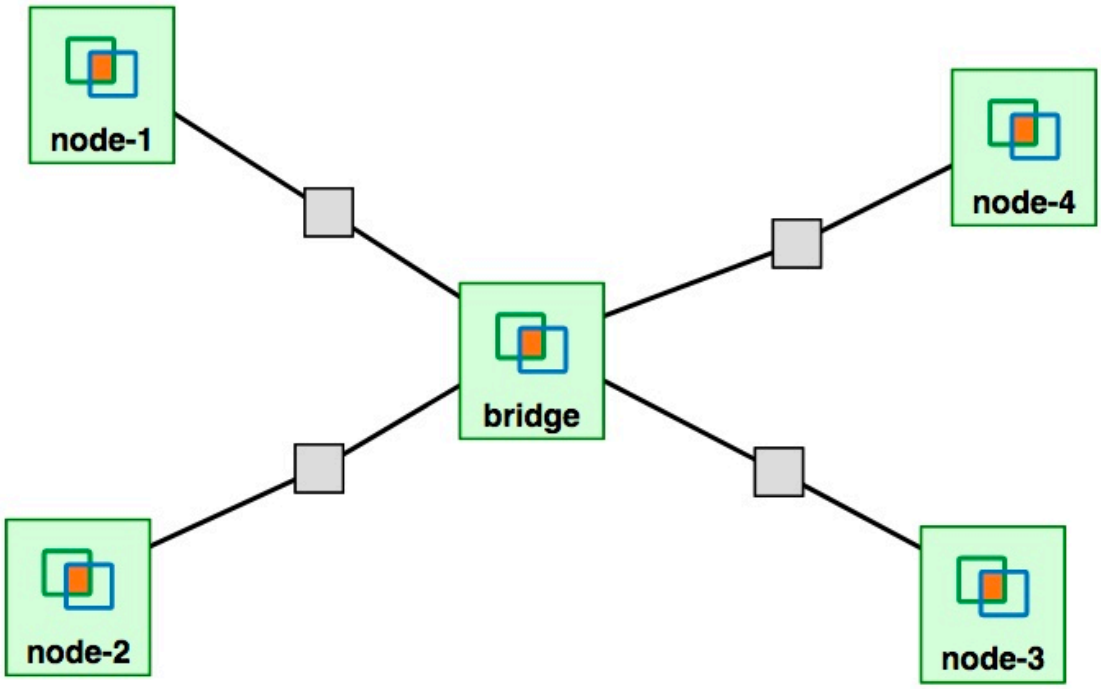
Due: 3:30pm Dec 10 Tuesday, 2019

0. Introduction

See Pre-lab8 for details.

1. Pre-Lab Information

See Pre-lab8 for details. The following is the topology of the network for this lab.



2. Lab Details, Part I: Learning MAC addresses

Now, we will take a **close-up** watch on how MAC addresses are added to the forwarding table on the bridge.

On the **bridge**, start by verifying that the forwarding table contains only local addresses:

```
brctl showmacs br0
```

and if not, wait for the non-local addresses to age out.

Then, run

```
bridge monitor fdb
```

on the bridge. This will show us new entries live, as they are added to or removed from the forwarding table.

On **node-2, node-3, and node-4**, we will use `tcpdump` to watch traffic on the interface connected to the bridge. Run

```
sudo tcpdump -n -e -i eth1
```

This will run `tcpdump` on the "eth1" interface (`-i eth1`), with the Ethernet headers showing (`-e`) and with IP addresses showing as numeric values, rather than hostnames (`-n`).

Then, on **node-1**, we will send some traffic to node-2:

```
ping -c 5 10.0.0.2
```

On the **bridge**, stop (i.e. Ctrl-c) the `bridge monitor fdb` command.

> Here, you are required to watch my video where I elaborate in detail what is going on among all the nodes including the bridge.

In summary, in the output, we can see:

- (not shown) Through a protocol called ARP, node-1 gets the MAC address of IP address 10.0.0.2 (node-2).
- Then as the first frame from node-1 arrives at the bridge, an entry for that host is added to the forwarding table on the bridge, indicated which bridge port that MAC address is connected to like this:

  ```
  02:8b:ac:42:7b:49 dev eth1 vlan 0
  ```

- On the bridge, the first frame from node-1 is then sent out of all of the other bridge ports, because it has

not yet known which bridge port the destination MAC address is connected to. In the tcpdump output, we see an entry similar to the following on all three nodes

```
22:28:54.194318 02:8b:ac:42:7b:49 > 02:11:7b:13:6e:4e, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.1 > 10.0.0.2: ICMP echo request, id 8495, seq 1, length 64
```

- node-2 (and only node-2) sends a reply to the ping request. When this reply reaches the bridge, an entry is added to the forwarding table for node-2, indicated which switch port that MAC address is connected to:

```
02:11:7b:13:6e:4e dev eth2 vlan 0
```

- Since the location of node-1 is already known, the reply from node-2 (shown below) is only sent out the bridge port where node-1 is located. Therefore, the reply does not appear in the tcpdump output on node-3 or node-4.

```
22:28:54.198137 02:11:7b:13:6e:4e > 02:8b:ac:42:7b:49, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 8495, seq 1, length 64
```

- Since the location of node-2 is known, further ping requests (shown below) are only sent out the bridge port where node-2 is located. They therefore appear in the tcpdump output on node-2, but not on the other nodes:

```
...
22:28:54.198096 02:8b:ac:42:7b:49 > 02:11:7b:13:6e:4e, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.1 > 10.0.0.2: ICMP echo request, id 8495, seq 1, length 64
22:28:54.198137 02:11:7b:13:6e:4e > 02:8b:ac:42:7b:49, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 8495, seq 1, length 64
22:28:55.199303 02:8b:ac:42:7b:49 > 02:11:7b:13:6e:4e, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.1 > 10.0.0.2: ICMP echo request, id 8495, seq 2, length 64
22:28:55.199330 02:11:7b:13:6e:4e > 02:8b:ac:42:7b:49, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 8495, seq 2, length 64
22:28:56.200553 02:8b:ac:42:7b:49 > 02:11:7b:13:6e:4e, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.1 > 10.0.0.2: ICMP echo request, id 8495, seq 3, length 64
22:28:56.200578 02:11:7b:13:6e:4e > 02:8b:ac:42:7b:49, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 8495, seq 3, length 64
22:28:57.201829 02:8b:ac:42:7b:49 > 02:11:7b:13:6e:4e, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.1 > 10.0.0.2: ICMP echo request, id 8495, seq 4, length 64
22:28:57.201865 02:11:7b:13:6e:4e > 02:8b:ac:42:7b:49, ethertype IPv4 (0x0800), l
ength 98: 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 8495, seq 4, length 64
...
```

(ACTION) Do the following.

- On bridge, **wait till the non-local addresses to age out**. Then run `bridge monitor fdb`.
- Between node-1, node-2, node3, and node-4, run **M-I-N-I-M-U-M** number of `ping` commands such that the bridge learns the MAC addresses of all these four nodes.

  > Hint, **you definitely don't have to run the `ping` commands at all nodes to all other nodes.**

- On bridge, stop `bridge monitor fdb`. Run `brctl showmacs br0` to confirm.
- Take a screenshot of the bridge and and save it in `learn_mac_addresses_yourinitial.jpg`.
  - Make sure the screenshot contains both `bridge monitor fdb` and `brctl showmacs br0` commands and their outputs above.

3. Lab Details, Part II: Effect of a Smaller Collision Domain

3.1. MAC Address Learning Is Turned On (Default)

Finally, we will observe how a switch or a bridge improves network performance by reducing the number of hosts in each collision domain. Only one host in a collision domain may transmit at any one time, and the other hosts listen to the network in order to avoid collisions. Thus, the total network capacity is shared among all hosts on the collision domain. Therefore, the ideal case is that each host is in its own collision domain to utilize the full network capacity/bandwidth!

To measure this effect, we will use `iperf`, a tool that estimates network capacity by trying to send data as quickly as possible over a link, and measuring the total data rate.

To install iperf, on **each of node-1, node-2, node-3, and node-4**, run

```
sudo apt-get update
sudo apt-get -y install iperf
```

Then, run the `iperf` receiver/server application on **node-3 and node-4**:

```
iperf -s
```

Finally, we will run the `iperf` transmitter to send traffic to each of those. Try to start the two transmitters quickly, one after the other. Flex your fingers if needed 🙂

On **node-1**, run

```
iperf -c 10.0.0.3 -t 90
```

and on **node-2**, quickly run

```
iperf -c 10.0.0.4 -t 90
```

After a couple of minutes, you should see messages on the two receiver/server instances, indicating the total data rate, e.g.

```
[  4]  0.0-96.5 sec  10.9 MBytes   945 Kbits/sec
```

on one, and

```
[  4]  0.0-96.6 sec  10.9 MBytes   945 Kbits/sec
```

on the other.

The bandwidth of all the links is 1000 Kbits/sec, therefore, each of of two traffic flows (node-1 to node-3 and node-2 to node-4) was transmitted at roughly full speed if you check out the last number in the output above (945 Kbits/sec or something like that). **This is because, by default, each port on a bridge becomes its own collision domain, and so each network segment can separately support the full network capacity.**

3.2. MAC Address Learning Is Turned Off

Now, we will **turn off** MAC address learning on the **bridge** (by setting the ageing timeout to 0, so that all forwarding table entries expire immediately. Basically, the bridge forgets as soon as it learns -- how sad😩):

```
sudo brctl setageing br0 0
```

Without MAC address learning, all frames will be flooded on all ports, and all four nodes will be in a single collision domain. The 1000 Kbps capacity will be shared between them.

The `iperf` receivers/servers should still be running on **node-3 and node-4**. We will start the transmitters again, one after the other. On **node-1**, run

```
iperf -c 10.0.0.3 -t 90
```

and on **node-2**, quickly run

```
iperf -c 10.0.0.4 -t 90
```

Now we can see from the measured data rate on each of the two receivers/servers, that the 1000 Kbps link capacity is shared between the two traffic flows. E.g.

```
[  5]  0.0-96.9 sec  5.50 MBytes   476 Kbits/sec
```

on one, and

```
[  5]  0.0-98.9 sec  5.88 MBytes   498 Kbits/sec
```

on the other.

To restore MAC address learning, on the bridge run

```
sudo brctl setageing br0 300
```

(ACTION) Do the following.

- With MAC learning turned **ON**, run an `iperf` receiver/server on **node-4** and on **node-1, node-2, and node-3**, simultaneously run `iperf` transmitters to send traffic to the same target **node-4** (`iperf -c 10.0.0.4 -t 90`).
- Take a screenshot of each of the hosts node-1, node2, node-3, and node-4, combine them into one image file named `collision_domain_yourinitial.jpg`.
    - Make sure the screenshot for each node contains the `iperf -c 10.0.0.4 -t 90` command and its output above.

4. What to Turn in?

Submit the following files:

- (3 points) learn_mac_addresses_yourinitial.jpg
- (3 points) collision_domain_yourinitial.jpg

On BlazeVIEW, when submitting your work, in the comment section, answer the questions below **in your own words**:

- (4 points) Describe how many `ping` commands you ran in the last experiment at the end of **section 2** (as shown in your figure `learn_mac_addresses_yourinitial.jpg`) and what they were (i.e from which node to which node).
- (4 points) Describe the network capacity/bandwidth (i.e. data rate) seen by each of the three transmitters in the last experiment at the end of **section 3.2** (as shown in your figure `collision_domain_yourinitial.jpg`), and **very importantly**, explain why it was different from what we did in Section 3.1, where MAC learning was also turned on.