BGP Route Hijacking and Anycast

In this project you are to demonstrate route hijacking with BGP, and also how to ha e two nodes with the same IP address without conflict. Specifically, you are to create a line of **five** routers, each representing a separate AS. Each router should also ha e a host connected. Host h0 is there so each r_i (including r1) has r_i-eth1 to the left and (except for r5) r_i-eth2 to the right. The interfaces pointing *downwards* are r_i-eth0.

```
h0----r1----r2----r3----r4----r5
      |     |     |     |     |
      h1    h2    h3    h4    h5
```

You will start BGP on each of the routers. The link from r1 to r2 will be subnet 10.0.1.0/24, on through the link from r4 to r5 as 10.0.4.0/24. The left host on each link will have host byte 1, and the right host will have host byte 2. For example, r3 will have its right-hand interface be r3-eth2, with IP address 10.0.3.1, and its lefthand interface will be r3-eth1, with IP address 10.0.2.2. Here is a table of the r_i interfaces:

	left interface	left IP address	right interface	right IP address
r1	r1-eth1	None	r1-eth2	10.0.1.1
r2	r2-eth1	10.0.1.2	r2-eth2	10.0.2.1
r3	r3-eth1	10.0.2.2	r3-eth2	10.0.3.1
r4	r4-eth1	10.0.3.2	r4-eth2	10.0.4.1
r5	r5-eth1	10.0.4.2	None	

The h_i--r_i subnet will be 10.0.(10*i).0/24 (so the h2-r2 subnet is 10.0.**20**.0/24), with h_i having IP address 10.0.(10*i).10 and r_i-eth0 having IP address 10.0.(10*i).1.

Getting the eth1 and eth2 stuff set up for the r_i is rather frustrating, so I'm giving you that part. Note that it is a loop. Add what you need to create the h_i nodes, the r_i-h_i links, and give the r_i-eth0 interfaces and the h_i-eth0 interfaces (the latter in a separate loop) their IP addresses.

You will have to write code to create h0-h5, and create the appropriate links. These can either be done with appropriate loops, or via individual statements.

You will also have to write code further down to assign IP addresses to h1-h5 (see below about h0).

You will have to install Quagga on your virtual machine. As root, do this with the command **apt-get install quagga**. You may also want to install traceroute: **apt-get install traceroute**.

The python file bgpbase.py is your starting point. You should create a folder to work in, perhaps /home/mininet/loyola/bgp.

Each router needs a directory of configuration files; you should name these r1 through r5 (the bgpbase.py code assumes these names). They will all be subdirectories of your mininet/loyola/bgp folder. Each folder will also need the following configuration files, which you will create:

- zebra.conf

- bgpd.conf

Each folder will also have the following files created dynamically: zebra.pid, zebra.log, bgpd.pid, bpgd.log. In order for all this to work, the r_i directories and the conf files will all have to be owned by "quagga". You can do this in a single command (as root) from the bgp directory with: **chown -R quagga r?**. (The -R is for "recursive", changing the directories and their contents).

Most file references are *relative* to the current directory, *eg* to "r1/bgpd.conf". However, in bgpbase.py I did use an absolute file reference, DIRPREFIX. Change it to reflect your configuration as necessary. I had trouble making this relative, but didn't investigate hard; feel free to look into this (or not).

The zebra.conf files are all very similar; the parts in bold have to change. Here is the file for r2:

```
hostname r2
password zebra
enable password zebra

interface r2-eth0
  multicast
interface r2-eth1
  multicast

log file r2/zebra.log
```

The bgpd.conf file is a bit more complicated. Here is the file for r2 again; we are assuming that r_i has AS number 100*i:

```
hostname r2
password zebra
enable password zebra

router bgp 200

bgp router-id 10.0.1.2             # r1 is an exception: 10.0.0.2. r3 is 10.0.2.2; r4
is 10.0.3.2
neighbor 10.0.1.1 remote-as 100    # left neighbor
neighbor 10.0.2.2 remote-as 300    # right neighbor

! prefix we announce
network 10.0.20.0/24            # r2's h2 link
network 10.0.1.0/24             # r2's left subnet to r1
network 10.0.2.0/24             # r2's right subnet to r3

log file r2/bgpd.log
```

You will have to keep track of the lefthand and righthand networks for each router; r1 does *not* announce its lefthand network (yet), and r5 does not *have* a righthand network. If things are not working, check the bgpd.log files for error messages.

Your basic layout is probably good when "h1 ping h5" works at the mininet> prompt. Note that BGP might take 10-30 seconds to get the routes figured out.

Part 1: Hijacking

The first step here is to have h0 have IP address 10.0.50.10, and r1-eth1 have IP address 10.0.50.1. The next step is to have r1 **announce** this route. To do this, start up an xterm on r1, and enter the following commands:

```
telnet 10.0.1.1 bgpd          # this connects to r1's bgpd daemon
Password: zebra
r1> enable
Password: zebra
r1# configure terminal (or just "conf t")
r1(config)# router bgp 100
r1(config-router)# network 10.0.50.0/24       # the actual announcement that r1 can reach this network
```

Give everyone a few seconds, and run **ip route list** on each router (*eg* with mininet> **r2 ip route list**). This should show that r1 and r2 route to 10.0.50.0/24 via their lefthand interfaces r1-eth1 and r2-eth1, and r4 and r5 via their righthand interfaces r4-eth2 and r5-eth2. What do you think r3 will do? Send me the routing output for each of r1-r5 (remember if you do this at the mininet> prompt you can copy with CNTL-SHIFT-C, and then paste it into a document on your host laptop).

R1 has in effect made an announcement about 10.0.50.0/24 that has "stolen" traffic from the real 10.0.50.0/24. At least for part of the "Internet" of r1 through r5.

Part 2: BGP Anycast

The idea here is that we can have multiple nodes with IP address 10.0.50.10, without conflict. BGP will figure out which one is which. You can do this with the above, *or* modify your Python file and r1/bgpd.conf so that:

- h0 has address 10.0.50.10
- r1-eth0 has address 10.0.50.1
- r1 announces 10.0.50.0/24

At this point each of h1-h4 should be able to ping 10.0.50.10, *but some hi will get h0 and some will get h5*. One way to tell is to look at the routing tables for r1-r5: which way do they route to 10.0.50.0/24? Another way is to use **traceroute 10.0.50.10**, and look at the sequence of IP addresses traversed. Submit whichever form of output you choose.

Note that there is **no way** h0 and h5 (with the same IP address) can ever talk to one another, because there is no route between them. So all is good.