CommNet Routing Project Group 87

Marco Hassan Lukas Meier Laurin Goeller

2020-04-9

Contents

1 Part 1

1.1 Question 1.1

1.1.1 Configuration

We got the subnet 87.200.0.0/23 and divided it into two subnets:\ 87.200.0.0/24 = VLAN 10 and 87.200.1.0/24 = VLAN 20. There we choose the following addresses:

```
IP address
ZURI:
ZURI-L2.10      up      default         87.200.0.11/28
ZURI-L2.20      up      default         87.200.1.21/28
GENE:
GENE-L2.10      up      default         87.200.0.10/28
GENE-L2.20      up      default         87.200.1.20/28
```

Where,

```
## CERN ##
# staff     # student
87.200.0.1  87.200.1.1

## ETH ##
# staff     # student
87.200.0.2  87.200.1.2

## EPFL ##
# staff     # student
87.200.0.3  87.200.1.3
```

Traceroute from EPFL-student to EPFL-staff:

```
root@student_3:~# traceroute 87.200.0.3
traceroute to 87.200.0.3 (87.200.0.3), 30 hops max, 60 byte packets
 1  87.200.1.20 (87.200.1.20)  6.856 ms  6.650 ms  7.934 ms
 2  * * 87.200.0.3 (87.200.0.3)  23.982 ms
```

We can see it takes the route via the router ZURI (87.200.1.20) which is correct as they are in different VLANs.

Traceroute from ETHZ-staff to EPFL-student:

```
traceroute to 87.200.1.3 (87.200.1.3), 30 hops max, 60 byte packets
 1  87.200.0.11 (87.200.0.11)  7.536 ms  7.434 ms  7.414 ms
 2  * * 87.200.1.3 (87.200.1.3)  37.190 ms
```

We can see it chooses the route via the router ZURI (87.200.1.11). We see another IP address as before because we are now in VLAN 10 which chooses 87.200.0.11 as gateway.

Traceroute from EPFL-student to ETHZ-staff:

```
root@student_3:~# traceroute 87.200.0.2
traceroute to 87.200.0.2 (87.200.0.2), 30 hops max, 60 byte packets
 1  87.200.1.20 (87.200.1.20)  14.054 ms  12.372 ms  12.175 ms
 2  87.200.0.2 (87.200.0.2)  30.420 ms  30.451 ms  30.393 ms
```

1.2 Question 1.2

1.2.1 Router IP Matching

After mapping all the IPs to the interfaces, when showing the connection on a particular router (say NEWY router) you get

```
NEWY_router# show ip route connected
C>* 87.0.5.0/24 is directly connected, port_PARI, 15:26:14
C>* 87.0.8.0/24 is directly connected, port_LOND, 00:07:12
```

```
C>* 87.0.10.0/24 is directly connected, port_BOST, 00:08:03
C>* 87.0.11.0/24 is directly connected, port_ATLA, 00:08:57
C>* 87.0.12.0/24 is directly connected, port_MIAM, 15:27:05
C>* 87.105.0.0/24 is directly connected, host, 14:05:41
C>* 87.155.0.0/24 is directly connected, lo, 15:04:43
C>* 158.87.0.0/16 is directly connected, ssh, 6d01h55m
```

1.2.2 OSPF configuration

Before configuring it you should check that the host can ping directly to the connected router.

You can check this at the NEWY host for instance running

```
root@NEWY_host:~# ping 87.105.0.1
PING 87.105.0.1 (87.105.0.1) 56(84) bytes of data.
64 bytes from 87.105.0.1: icmp_seq=1 ttl=64 time=0.027 ms
64 bytes from 87.105.0.1: icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from 87.105.0.1: icmp_seq=3 ttl=64 time=0.033 ms
64 bytes from 87.105.0.1: icmp_seq=4 ttl=64 time=0.046 ms
64 bytes from 87.105.0.1: icmp_seq=5 ttl=64 time=0.049 ms
64 bytes from 87.105.0.1: icmp_seq=6 ttl=64 time=0.041 ms
```

OSPF routers flood IP routes over OSPF adjacencies. At its heart, OSPF is a link-state protocol that uses flooding of link-state information and a Dijkstra least-cost path algorithm.

After configuring all of the ospf on the entire US part you can see from instance that the Atlanta router cannot just simply reach it's directly connected router but rather all of the routers in the US. See it with the following command

```
ATLA_router# show ip route ospf
O>* 87.0.5.0/24 [110/20] via 87.0.11.1, port_NEWY, 00:32:43
O>* 87.0.6.0/24 [110/20] via 87.0.13.2, port_MIAM, 00:40:22
O>* 87.0.7.0/24 [110/30] via 87.0.11.1, port_NEWY, 00:30:22
O>* 87.0.8.0/24 [110/20] via 87.0.11.1, port_NEWY, 00:33:02
O>* 87.0.9.0/24 [110/20] via 87.0.13.2, port_MIAM, 00:40:47
O>* 87.0.10.0/24 [110/20] via 87.0.11.1, port_NEWY, 00:33:12
O   87.0.11.0/24 [110/10] is directly connected, port_NEWY, 00:43:45
O>* 87.0.12.0/24 [110/20] via 87.0.11.1, port_NEWY, 00:32:37
  *                       via 87.0.13.2, port_MIAM, 00:32:37
O   87.0.13.0/24 [110/10] is directly connected, port_MIAM, 00:43:53
O>* 87.105.0.0/24 [110/20] via 87.0.11.1, port_NEWY, 00:06:11
O>* 87.106.0.0/24 [110/30] via 87.0.11.1, port_NEWY, 00:07:13
O   87.107.0.0/24 [110/10] is directly connected, host, 00:03:21
O>* 87.108.0.0/24 [110/20] via 87.0.13.2, port_MIAM, 00:04:42
```

Finally we display the traceroute as requested.

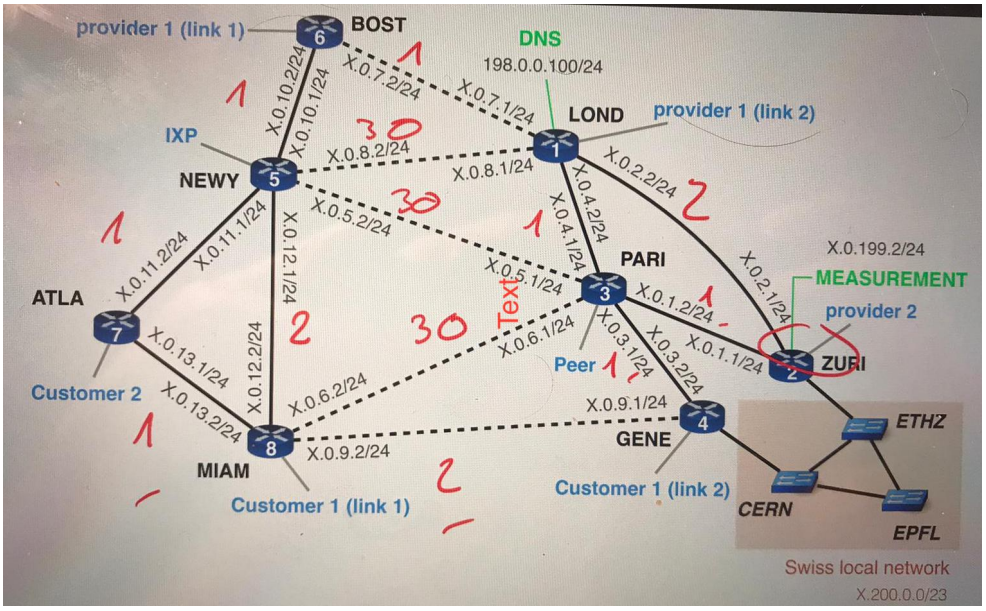
```
root@PARI_host:~# traceroute 87.107.0.1
traceroute to 87.107.0.1 (87.107.0.1), 30 hops max, 60 byte packets
 1  PARI-host.group87 (87.103.0.2)  2.023 ms  2.342 ms  2.327 ms
 2  GENE-PARI.group87 (87.0.3.2)  3.507 ms LOND-PARI.group87 (87.0.4.2)  3.493 ms  3.481 ms *
 3  MIAM-GENE.group87 (87.0.9.2)  4.098 ms  4.295 ms
 4  ATLA-NEWY.group87 (87.0.11.2)  14.481 ms NEWY-BOST.group87 (87.0.10.1)  22.876 ms
    ATLA-NEWY.group87 (87.0.11.2)  14.449 ms
 5  ATLA-MIAM.group87 (87.0.13.1)  12.898 ms  12.903 ms host-ATLA.group87 (87.107.0.1)  14.430 ms
```

Notice that we did not save a traceroute before configuring 1.3 and therefore you observe some load-balancing between the GENE-MIAM-ATLA and LOND-BOST-NEWY-ATLA as a by product of the weights set in the next section.

1.3 Question 1.3

1.3.1 Bandwidth understanding

After setting the weights such that the two continents stays separated and never cross the atlantic, you have to



h!0.0

Figure 1: Chosen OSPF Weights

configure the OSPF weights such that submarine paths with higher bandwidth are preferred whenever it is possible.

This is done through `iperf3` command. This measures the throughput between a client and a server. To do that you have first to instantiate an `iperf3` server on a region and then an `iperf3` client on the other. The two will send each other ICMP messages recording the available bandwidth.

You will then get for the network the following measures:

```
BOST - LOND ~ 95 Mbit/sec
NEWY - LOND ~ 11.3 Mbit/sec
NEWY - PARI ~ 11.3 Mbit/sec
PARI - MIAM ~ 2.6 Mbit/sec
GENE - MIAM ~ 11.3 Mbit/sec
```

The GENE - MIAM is slow as the Swiss local Network is slow. So you can safely assume config 4 mentioned in the routing project questions as this is the only configuration matching our bandwidth measurements listed above.

1.3.2 Further Requirements

Load Balancing:

In addition, you need to make sure that all traffic from MIAM to NEWY is loadbalanced on the two paths MIAM-NEWY, and MIAM-ATLA-NEWY and the traffic from ZURI to LOND is loadbalanced on the two paths ZURI-LOND, and ZURI-PARI-LOND.

Weights to avoid atlantic crossing:

Your top priority is to minimize latency, and to do so you must configure OSPF weights such that the traffic never traverses two submarine links

In order to do that we decided to set the following OSPF weights1

Finally the requested traceroute between the ATLA host to the ZURI loopback interface looks as follows

```
root@ATLA_host:~# traceroute 87.152.0.1
traceroute to 87.152.0.1 (87.152.0.1), 30 hops max, 60 byte packets
 1  ATLA-host.group87 (87.107.0.2)  0.084 ms  0.020 ms  0.017 ms
 2  MIAM-ATLA.group87 (87.0.13.2)  0.325 ms NEWY-ATLA.group87 (87.0.11.1)  0.316 ms
    MIAM-ATLA.group87 (87.0.13.2)  0.284 ms
 3  BOST-NEWY.group87 (87.0.10.2)  0.357 ms GENE-MIAM.group87 (87.0.9.1)  0.574 ms  0.549 ms
 4  PARI-LOND.group87 (87.0.4.1)  10.987 ms  10.941 ms  10.884 ms
```

```
 5  87.152.0.1 (87.152.0.1)  11.779 ms  12.289 ms  11.631 ms
```

As expected from our weights you can see that the road is balanced among MIAM-ATLA-GENE-PARI-ZURI and ZURI-(PARI)-BOST-NEWY-ATLA.

Obviously, due to the multiple load balancing each message might take a different road, however performing multiple `traceroute` you will eventually end up observing the required load-balance (the one among ZURI-PARI-LOND) visible in the 4th line below.

```
root@ATLA_host:~# traceroute 87.152.0.1
traceroute to 87.152.0.1 (87.152.0.1), 30 hops max, 60 byte packets
 1  ATLA-host.group87 (87.107.0.2)  0.127 ms  0.034 ms  0.017 ms
 2  NEWY-ATLA.group87 (87.0.11.1)  0.318 ms  0.282 ms  0.258 ms
 3  BOST-NEWY.group87 (87.0.10.2)  0.387 ms  0.319 ms  0.301 ms
 4  LOND-BOST.group87 (87.0.7.1)  20.464 ms PARI-LOND.group87 (87.0.4.1)
    10.609 ms  10.590 ms
 5  87.152.0.1 (87.152.0.1)  12.313 ms  12.292 ms  12.274 ms
```

1.4 Question 1.4

> The Swiss local network is not able to handle large amounts of traffic. Therefore, make sure that no transit traffic goes from ZURI to GENE (or vice versa) through the Swiss local network. Only traffic destined to one of the hosts in the Swiss local network should go through these interfaces.
>
> However, there is one exception for the traffic from ZURI to the host connected to PARI. This traffic should take the following path: ZURI-GENE-PARI.

To achieve the result we decided to set up a static route, therefore
we configured the ZURI router as follows:

```
ZURI_router# conf t
ZURI_router(config)# ip route 87.103.0.1/24 87.200.0.10
```

This effectively redirects all of the traffic directed to PARI in the ZURI router to the GENE interface.

Notice however as a potential drawback that should the network topology change the static router will not automatically adapt to the changes as do the OSPF dynamic routes given the OSPF weights.

Then, show the result of a traceroute from ETHZ-staff to the host connected to PARI.

```
traceroute to 87.103.0.1 (87.103.0.1), 30 hops max, 60 byte packets
 1  87.200.0.11 (87.200.0.11)  7.952 ms  7.671 ms  7.648 ms
 2  87.200.0.10 (87.200.0.10)  20.137 ms  20.111 ms  20.091 ms
 3  PARI-GENE.group87 (87.0.3.1)  19.564 ms PARI-ZURI.group87 (87.0.1.2)  12.942 ms  12.939 ms
 4  * host-PARI.group87 (87.103.0.1)  17.163 ms  17.170 ms
```

You can see now that the traceroute goes through the GENE gateway.

1.5 Question 1.5

In this question we configured the full-mash iBGP connectivity such that the external BGP messages will be propagated correctly across our 87.0.0.0/8 subnet.

At the end the internal BGP configuration for the Atlanta interface should look as follows:

```
ATLA_router# show ip bgp summary
IPv4 Unicast Summary:
BGP router identifier 87.157.0.1, local AS number 87 vrf-id 0
BGP table version 11950
RIB entries 448, using 81 KiB of memory
Peers 8, using 163 KiB of memory
Neighbor        V         AS MsgRcvd MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd
```

```
87.151.0.1      4          87    5685    3608        0    0    0 2d11h46m            1
87.152.0.1      4          87    6307    3608        0    0    0 2d11h46m            1
87.153.0.1      4          87    3825    3608        0    0    0 2d11h46m            2
87.154.0.1      4          87    3872    3600        0    0    0 2d11h46m            1
87.155.0.1      4          87   15449   14285        0    0    0 01w2d08h          273
87.156.0.1      4          87    5672    3609        0    0    0 2d11h46m            1
87.158.0.1      4          87   14888   14283        0    0    0 01w2d08h            8
179.1.98.2      4          90    5689    9731        0    0    0 3d06h47m            3
```

Regarding the `update-source lo` command what this does is effectively change the IP address of the iBGP connection, which is by default set to the interface closest to the to the iBGP peer as mentioned in the provided tutorial. Such a default might be troublesome as if this specific interface is disrupted the BGP messages will not be able to forward the messages to the desired peer albeit the router might be yet up and running.

To avoid such a problem it is common practice to update the source and change the default by specifying the destination of the iBGP as the loopback interface. This is always up and allows Border Gateway Protocol (BGP) neighborship between two routers to stay up even if one of the outbound physical interface connected between the routers is down.[1] This is what was actually done with the command above.

2 Part 2

2.1 Question 2.1

> To include in your report: Explain what next-hop-self does and why you have to use it using an example in your own network. Also, explain on which BGP sessions next-hop-self is required.

The next-hop state the IP address the message should reach next to go to the desired destination. Now, the problem with eBGP is that when eBGP announcements are distributed over iBGP sessions the next-hop IP address does not change by default with respect to the one received by externally connected router when AS are crossed. So keeping the default, an router working internally in our subnet would not be able to determine the next-hop to reach the desired destination as that would be mapped to the external AS interface the message came from and this is unknown to the internal router.

To deal with the issue above, a `next-hop-self` command is issued such that when a router forwards an announcement via iBGP it updates the next-hop IP address to itself so that the routers in the subnet always knows who to reach next.

> Then, show us the results of a show ip bgp for the router PARI. You should see the prefixes advertised by your neighboring ASes, which would indicate that your eBGP sessions are correctly configured and that the advertisements are correctly propagated through your iBGP sessions.

```
*>i1.0.0.0/8         87.155.0.1               100      0 104 102 1 i
*>i2.0.0.0/8         87.155.0.1               100      0 104 102 2 i
*>i3.0.0.0/8         87.155.0.1               100      0 104 102 1 3 i
*>i3.108.0.0/25      87.155.0.1               100      0 104 102 42 44 41 i
*>i3.108.0.128/25    87.155.0.1               100      0 104 102 42 44 41 i
*>i4.0.0.0/8         87.155.0.1               100      0 104 102 1 4 i
*>i4.108.0.0/25      87.155.0.1               100      0 104 102 41 44 42 i
*>i4.108.0.128/25    87.155.0.1               100      0 104 102 41 44 42 i
*>i5.0.0.0/8         87.155.0.1               100      0 104 102 1 4 5 i
*>i5.108.0.0/25      87.155.0.1               100      0 104 102 42 44 41 i
*>i5.108.0.0/26      87.155.0.1               100      0 104 102 2 4 5 i
*>i5.108.0.64/26     87.155.0.1               100      0 104 102 1 4 5 i
*>i5.108.0.128/25    87.155.0.1               100      0 104 102 42 44 41 i
*>i5.108.0.128/26    87.155.0.1               100      0 104 102 1 4 5 i
*>i5.108.0.192/26    87.155.0.1               100      0 104 102 1 4 5 i
*>i5.108.128.0/26    87.155.0.1               100      0 104 102 1 4 5 i
*>i5.108.128.64/26   87.155.0.1               100      0 104 102 1 4 5 i
*>i6.0.0.0/8         87.155.0.1               100      0 104 102 1 4 6 i
*>i6.108.0.0/25      87.155.0.1               100      0 104 102 41 44 42 i
*>i6.108.0.128/25    87.155.0.1               100      0 104 102 41 44 42 i
 ....
```

[1] https://www.omnisecu.com/cisco-certified-network-associate-ccna/what-is-loopback-interface-in-a-router.php

Then, show us that your neighboring ASes do receive the advertisement for your /8 prefix. To do that, show in your report the result of the looking glass for one router located in a neighboring AS. You should see your prefix in the looking glass.

You can see the correct advertising at this link. There for instance for our provider AS85, you can see the following advertisement:

```
*>i87.0.0.0/8       85.154.0.1               0    100      0 87 i
```

Finally, show us that you have data-plane connectivity with your neighbors by showing the result of a traceroute from your PARI-host to the PARI-host of one of your neighboring ASes

```
root@PARI_host:~# traceroute 90.103.0.1
traceroute to 90.103.0.1 (90.103.0.1), 30 hops max, 60 byte packets
 1  PARI-host.group87 (87.103.0.2)  0.920 ms  0.084 ms  0.042 ms
 2  GENE-PARI.group87 (87.0.3.2)  0.190 ms LOND-PARI.group87 (87.0.4.2)  0.151 ms GENE-PARI.group87 (87.0.3
 3  * MIAM-GENE.group87 (87.0.9.2)  0.395 ms  0.340 ms
 4  NEWY-BOST.group87 (87.0.10.1)  20.477 ms ATLA-NEWY.group87 (87.0.11.2)  11.245 ms  11.190 ms
 5  ATLA-MIAM.group87 (87.0.13.1)  11.823 ms ZURI-PARI.group90 (90.0.1.1)  20.321 ms ATLA-MIAM.group87 (87.
 6  ZURI-LOND.group90 (90.0.2.1)  21.703 ms  20.524 ms  20.483 ms
 7  PARI-LOND.group90 (90.0.4.1)  17.806 ms GENE-PARI.group90 (90.0.3.2)  21.309 ms  21.294 ms
 8  PARI-LOND.group90 (90.0.4.1)  22.193 ms  22.135 ms  22.132 ms
 9  host-PARI.group90 (90.103.0.1)  21.310 ms  16.509 ms  20.813 ms
```

2.2 Question 2.2

In this question we had to use the community values to send BGP advertisements to the peers connected to you through an IXP.

This was done as followed according to the mentioned configurations of the routing paper.

To include in your report: Take a screenshot of the relevant parts of the out route-map you configured on the session from the router in NEWY to the IXP. In a few sentences explain what all the lines in the route-map mean and do. Then, show a looking glass entry of another AS which proves that your prefix has been advertised through the IXP. Finally, use the measurement container to perform a traceroute from another AS (in another region) to your AS for a destination where the traffic should go through the IXP. Show the result in your report.

To properly set up the community tags we used:

```
NEWY_router# conf t
NEWY_router(config)# route-map MAP_OUT permit 8
NEWY_router(config-route-map)# set community 125:102 125:104 125:106 125:108 125:110 125:112
NEWY_router(config-route-map)# exit
NEWY_router(config)# router bgp 87
NEWY_router(config-router)# neighbor 180.125.0.125 route-map MAP_OUT out
```

We add a community value to every packet. The IXP learns what AS can send Traffic to us. ASes in our Cluster are excluded as of Question 3.2. Where they are not allowed to send traffic to us.

Looking glass Group 106:

```
*>i87.0.0.0/8       106.155.0.1              0     50      0 87 i
*>i87.108.0.0/25    106.155.0.1                    50      0 87 85 83 81 21 i
 *                   179.2.12.1                     10      0 103 101 21 i
*>i87.108.0.0/26    106.155.0.1                    50      0 87 i
*>i87.108.0.64/26   106.155.0.1                    50      0 87 i
*>i87.108.0.128/25  106.155.0.1                    50      0 87 85 83 81 21 i
 *                   179.2.12.1                     10      0 103 101 21 i
*>i87.108.0.128/26  106.155.0.1                    50      0 87 i
*>i87.108.0.192/26  106.155.0.1                    50      0 87 i
```

You can see that the only possible route directly to our AS is via the IXP.

Launch traceroute form AS89 to 87.105.0.1 with the measurement container:

```
root@9a1c443ab88a:~# ./launch_traceroute.sh 106 87.105.0.1
Hop 1:  106.0.199.1 TTL=0 during transit
Hop 2:  106.0.1.2 TTL=0 during transit
Hop 3:  106.0.2.2 TTL=0 during transit
Hop 4:  106.0.8.2 TTL=0 during transit
Hop 5:  180.125.0.87 TTL=0 during transit
Hop 6:  87.105.0.1 Echo reply (type=0/code=0)
...
```

Where 180.125.0.87 is the IXP.

3 Part 3

3.1 Question 3.1

First we have to respect the customer-peer-client relation. To do that you have to set for instance the following weights:

```
provider : 50
peer : 100
customer: 200
```

In order for you to set the preferences above on the external interfaces via eBGP, you need on the one hand to tag the customers, peers and providers as such. Moreover, you will have to allow all of the traffic coming from your customers to the peers and providers.

In order to do that you have to specify tags identifying the three categories above. Here we decided to work even in a more specific way by setting the following tags:

```
Definiere: community-lists:
87:1 - BOST-inter-AS85
87:2 - LOND-inter-AS85
87:3 - ZURI-inter-AS86

87:11 - PARI-inter-AS88
87:12 - NEWY-inter-IXP

87:21 - GENE-inter-AS89
87:22 - MIAM-inter-AS89
87:23 - ATLA-inter-AS90
```

Then for instance for the BOST interface where you interact with a provider you would enter the following commands:

```
conf t
route-map MAP_IN permit 10
set community 87:1
set local-preference 50
bgp community-list 1 permit 87:21
bgp community-list 1 permit 87:22
bgp community-list 1 permit 87:23
route-map MAP_OUT permit 10
match community 1
ip prefix-list 87 permit 87.0.0.0/8
route-map MAP_OUT permit 11
match ip address prefix-list 87
router bgp 87
neighbor ip-of-neighbor route-map MAP_IN in
neighbor ip-of-neighbor route-map MAP_OUT out
```

For the peer-peer connections- for instance in the PARI interface - you would have a similar command with different preferences.

For the customer connections for instance in the GENE router

```
conf t
route-map MAP_IN permit 10
set community 87:21
set local-preference 200
exit
router bgp 87
route-map MAP_OUT permit 9
neighbor ip-of-neighbor route-map MAP_IN in
```

You can see above that on the one hand you have to set the tags to mark the different customers, peers and providers in the route-maps `in` part, as well as specifying the local-preference associated with such tag. In contrast to that w.r.t. the routes-map `out` part you will have to specify a community list with the tags of your customers you want to let through. On top of that as there is an implicitly appended `else deny` clause in route maps you have to advertise your /8 subnet through the route-map command above such that the traffic is still advertised.

> Then, show that your configuration works properly by adding the result of the looking glass of one of your peers, which is supposed to show that this peer does receive the prefixes of your customers, but does not receive the prefixes of your other peers.

For instance the looking glass of AS88:

```
*>i87.0.0.0/8       88.153.0.1               0   1000      0 87 i
* 90.0.0.0/8        179.1.91.1                    500      0 87 90 i
*> 102.0.0.0/8      179.1.91.1                    500      0 85 102 i
*> 104.0.0.0/8      179.1.91.1                    500      0 85 104 i
*> 106.0.0.0/8      179.1.91.1                    500      0 85 106 i
*> 108.0.0.0/8      179.1.91.1                    500      0 85 108 i
...
```

> Finally, launch a traceroute from one of your customers towards one of your peers using the measurement container. Verify that your AS forwards the packet directly to your peer and not to your provider. Include the result of the traceroute in your report.

```
root@9a1c443ab88a:~# ./launch_traceroute.sh 89 88.151.0.1
Hop 1:  89.0.199.1 TTL=0 during transit
Hop 2:  179.2.2.1 TTL=0 during transit
Hop 3:  88.0.11.1 TTL=0 during transit
Hop 4:  88.0.10.2 TTL=0 during transit
Hop 5:  88.151.0.1 Echo reply (type=0/code=0)
...
```

From the above you can see that the traceroute does not go through our AS. This is however possible following the BGP decision as both our AS and AS88 is a provider of AS89. Both routes (the direct one and the one over you) should get the same local preference but the direct one will normally have a shorter AS path and is therefore preferred.

3.2 Question 3.2

3.2.1 Q 3.2 (i)

Question: do not advertise any prefix to ASs in the same cluster.

This is done by not adding them to the community-list in 2.2.

3.2.2 Q 3.2 (ii)

Question: Do not allow any advertisements from neighbouring ASs.

This is done with the following command on NEWY router:

```
bgp as-path access-list 1 permit ^(102|104|106|108|110|112).*$
show bgp as-path-access-list 1

conf t
route-map MAP_IN permit 10
```

```
match as-path 1
exit
route-map MAP_IN deny 11
exit
```

the new NEWY route map MAP_IN looks like the following (the new/important lines are commented)

```
BGP:
route-map: MAP_IN Invoked: 12406
 permit, sequence 10 Invoked 12116
   #IN_MAP matches all routes coming from 102, 104, 108, 110, 112 (exercise 3.2 part (ii))
  Match clauses:
    as-path 1
    # All incoming routes from peers which are matched are tagged with
    # the according community value as well as a local preference of 100.
  Set clauses:
    community 87:12
    local-preference 100
  Call clause:
  Action:
    Exit routemap
 deny, sequence 11 Invoked 4
  Match clauses:
  Set clauses:
  Call clause:
  Action:
    Exit routemap
```

and the new NEWY route map MAP_OUT

```
route-map: MAP_OUT Invoked: 39931
 permit, sequence 8 Invoked 6457
  Match clauses:
  # Outgoing routes are tagged with the community values so the IXP # passes them on only to AS 102, 104, 1
  Set clauses:
    community 125:102 125:104 125:106 125:108 125:110 125:112
  Call clause:
  Action:
    Exit routemap
 permit, sequence 9 Invoked 17360
  Match clauses:
    ip address prefix-list 88
  Set clauses:
  Call clause:
  Action:
    Exit routemap
```

The following routes towards the Stub AS'es in our cluster are obtained when running sh ip bgp on NEWY router.

```
* i91.0.0.0/8       87.158.0.1              50    200      0 89 91 i
*>i                 87.157.0.1                    200      0 90 91 i
* i92.0.0.0/8       87.158.0.1              50    200      0 89 92 i
*                  180.125.0.104                  100      0 104 105 107 106 89 92 i
*>i                 87.157.0.1                    200      0 90 92 i
```

Only direct routes (hence the local preference of 200) and no routes via the IXP can be seen (except the one over AS 104, but it is possible 104 wrongly advertises this route to our AS)

This is the output of the looking glass of G84 NEWY (same region, same IXP)

```
* i87.0.0.0/8       83.158.0.1              69   3000      0 85 87 i
*>i                 83.157.0.1                   3000      0 86 87 i
```

This is the output of the looking glass of G104 NEWY (same region, same IXP)

```
*> 87.0.0.0/8       180.125.0.87             0     50      0 87 i
*> 88.0.0.0/8       180.125.0.85                   50      0 85 88 i
```

3.3 Question 3.3

The Idea is to append our own AS number multiple times to the as path in all advertisements coming from router LOND and BOST. The path over ZURI will therefore be recognized as "shorter" and taken as the preferred route.

potential drawback is based on that we can try to influence our provider but we cannot decide the path to take. E.g if the next hop is us (as a customer), then the path length wont even be considered for choosing the best route.

The following commands were used on router LOND + BOST

```
conf t
route-map MAP_OUT permit 10
set as-path prepend 87 87 87 87 87 87
exit
```

Resulting looking glass

```
*> 87.0.0.0/8       179.1.89.2               0    100      0 87 i
 * i                       85.154.0.1               0    100      0 87 i
```

and for the other route:

```
 * i87.0.0.0/8       85.158.0.1               0    100      0 87 i
*>                          179.1.90.2               0    100      0 87 i
```

No appended as-path (e.g. 87 87 87 87 87 87) is shown so both routes take the gateway through ZURI which is correct.

3.4 Question 3.4

The BGP MED attribute can be used to inform our provider about preferred routes. By setting the BGP MED of 20 to all outgoing routes in LOND and BOST to 10 we expect the provider to send us data via BOST. Problem: Provider still knows both routes and might ignore our preference.

Commands on the specific router (ZURI =20, BOST = 10):

```
conf t
route-map MAP_OUT permit 10
set metric 10/20
exit
```

Here are the two routes and their different BGP MED Value (10 and 20). This is the output of the looking glass of Group 85 from router GENE.

```
*>i87.108.0.0/26    85.158.0.1               10    100      0 87 87 87 87 87 87 87 i
* 179.1.90.2               20    100      0 87 87 87 87 87 87 87 i
```

85.158.0.1 is the route via BOST and 179.1.90.2 is the route via ZURI. They know both routes with different metric.

3.5 Question 3.5

Prefix Hijacking. Looking at the looking glasses

```
*>i87.0.0.0/8       9.155.0.1                     200      0 22 23 42 44 41 68 69 86 87 i
*>i87.108.0.0/25    9.155.0.1                     200      0 22 23 21 i
*>i87.108.128.0/25  9.155.0.1                     200      0 22 23 21 i
```

We see that tier Level 1 AS 21 is advertising our prefix at `/25` precision.

This was furthermore sustained by checking at a `traceroute` for instance from the AS9 to our MIAM host at `87.108.0.1`. As expected the traffic was redirected in direction AS 21.

As visible from such traceroute displayed in the Appendix 4, once the traffic reaches the haijacking AS this does not consume it but rather forwards it to the AS24 via its ZURI router. An loop occurs until the message TTL eventually reaches 0 and is being dropped.

We solved the Hijiacking issue also by advertising the prefix on a higher precision level. In such a way we will gain a share of the traffic that was taken away back.

This would practically mean for the specific case to add on top of the previously defined ip subnet in the prefix-list 87 the following on the MIAMI router:

```
conf t
ip prefix-list 88 permit 87.108.0.0/26
ip prefix-list 88 permit 87.108.0.64/26
ip prefix-list 88 permit 87.108.0.128/26
ip prefix-list 88 permit 87.108.0.192/26
match ip address prefix-list 88
exit
```

Finally notice that we had to advertise the more specific routes entries such that the MIAM router will know how to deal with the `/26` entries.

Would your current solution to mitigate the hijack work for every possible hijack?

No; that does not hold true. the hijacker can continue the race to the bottom advertising an even more specific ip.

3.6 Question 3.6

Connection via OPENVPNGUI for Windows(OpenVpn in Ubuntu didn't work):

```
on Ubuntu for Windows:
lukas@DESKTOP-DK7RGUL:~$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
[...]
255.255.255.255 0.0.0.0         255.255.255.255 U         0 0          0 eth1
224.0.0.0       0.0.0.0         240.0.0.0       U         0 0          0 eth1
87.200.30.255   0.0.0.0         255.255.255.255 U         0 0          0 eth1
87.200.30.0     0.0.0.0         255.255.255.0   U         0 0          0 eth1
87.200.30.53    0.0.0.0         255.255.255.255 U         0 0          0 eth1
```

Now we set up a default route on my Computer (currently only for our AS):

```
ip address add 87.0.0.0/8 dev eth1
```

New netstat -rn, we can see that we should have access to the mini-Internet:

```
255.255.255.255 0.0.0.0         255.255.255.255 U         0 0          0 eth1
224.0.0.0       0.0.0.0         240.0.0.0       U         0 0          0 eth1
87.200.30.255   0.0.0.0         255.255.255.255 U         0 0          0 eth1
87.200.30.0     0.0.0.0         255.255.255.0   U         0 0          0 eth1
87.200.30.53    0.0.0.0         255.255.255.255 U         0 0          0 eth1
87.255.255.255  0.0.0.0         255.255.255.255 U         0 0          0 eth1
87.0.0.0        0.0.0.0         255.0.0.0       U         0 0          0 eth1
87.0.0.0        0.0.0.0         255.255.255.255 U         0 0          0 eth1
```

But unfortunately the traceroute didn't work... So we ended 3.6 there.

```
traceroute -I eth1 87.200.X.X
```

4 Appendix

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Apr  6 14:05:39 2020 from 157.0.0.250
root@9a1c443ab88a:~# ./launch_traceroute.sh 9 87.108.0.1
Hop 1:  9.0.199.1 TTL=0 during transit
Hop 2:  9.0.4.2 TTL=0 during transit
Hop 3:  9.0.8.2 TTL=0 during transit
Hop 4:  179.0.16.1 TTL=0 during transit
Hop 5:  23.200.0.11 TTL=0 during transit
Hop 6:  23.0.4.2 TTL=0 during transit
Hop 7:  179.0.13.1 TTL=0 during transit
Hop 8:  23.0.4.2 TTL=0 during transit
Hop 9:  179.0.13.1 TTL=0 during transit
Hop 10:  23.0.4.2 TTL=0 during transit
Hop 11:  179.0.13.1 TTL=0 during transit
Hop 12:  23.0.4.2 TTL=0 during transit
Hop 13:  179.0.13.1 TTL=0 during transit
Hop 14:  23.0.4.2 TTL=0 during transit
Hop 15:  179.0.13.1 TTL=0 during transit
Hop 16:  23.0.4.2 TTL=0 during transit
Hop 17:  179.0.13.1 TTL=0 during transit
Hop 18:  23.0.4.2 TTL=0 during transit
Hop 19:  179.0.13.1 TTL=0 during transit
Hop 20:  23.0.4.2 TTL=0 during transit
Hop 21:  179.0.13.1 TTL=0 during transit
Hop 22:  23.0.4.2 TTL=0 during transit
Hop 23:  179.0.13.1 TTL=0 during transit
Hop 24:  23.0.4.2 TTL=0 during transit
Hop 25:  179.0.13.1 TTL=0 during transit
Hop 26:  23.0.4.2 TTL=0 during transit
Hop 27:  179.0.13.1 TTL=0 during transit
Hop 28:  23.0.4.2 TTL=0 during transit
Hop 29:  179.0.13.1 TTL=0 during transit
Hop 30:  23.0.4.2 TTL=0 during transit
```

Figure 2: Traceroute from AS9 towards MIAM host before fixing the Hjiack