

Communication Networks

Group 85

Dominik Schindler

Student ID: 09-916-123

Yannick Huber

Student ID: 16-917-619

Dario Bolli

Student ID: 16-832-685

FS 2021

227-0120-00L: Communication Networks

Eidgenössische Technische Hochschule Zürich



Due Date: 07-05-2021

1 Intra-domain Routing

Question 1.1

We partitioned the subnet 85.200.0.0/23 into the two subnets 85.200.0.0/24 and 85.200.1.0/24. The partition allows to easily group the hosts into two VLANs. For the NASA hosts on the VLAN 20, we used addresses from 85.200.1.0/24, and for the SpaceX hosts on VLAN 10 we used addresses from 85.200.0.0/24. See Table 1 for the detailed assignment of the IP addresses.

The traceroute from SpaceX_1 to SpaceX_3 (Figure 1) shows a single hop. We therefore conclude, that packets from SpaceX_1 are directly sent to SpaceX_3 on the layer-2 connection (using switches) between the hosts.

In contrary, the traceroute from SpaceX_1 to NASA_3 (Figure 2) includes the IP address 85.200.0.10. That is the address of NEWY-L2.10, the NEWY router interface for the switch S2 on the VLAN 10. VLAN 10 is SpaceX' VLAN. We therefore conclude that SpaceX_1 cannot directly send a message to NASA_3. Instead the packets must travel for a first hop on the senders VLAN 10 to the layer-3 router (the default gateway) and for a second hop on the receivers VLAN 20.

Table 1: IPv4 addresses for the hosts in the data center DC1

Host(-interface)	IP address
BROO-L2.10	85.200.0.110/24
BROO-L2.20	85.200.1.120/24
NEWY-L2.10	85.200.0.10/24
NEWY-L2.20	85.200.1.20/24
NASA_1	85.200.1.21/24
NASA_2	85.200.1.22/24
NASA_3	85.200.1.23/24
SpaceX_1	85.200.0.11/24
SpaceX_2	85.200.0.12/24
SpaceX_3	85.200.0.13/24

```
root@SpaceX_1:~# traceroute -n 85.200.0.13
traceroute to 85.200.0.13 (85.200.0.13), 30 hops max, 60 byte packets
 1 85.200.0.13 16.163 ms 15.926 ms 15.928 ms
```

Figure 1: Traceroute from SpaceX_1 to SpaceX_3 (85.200.0.13). Packets are directly routed between hosts on the same VLAN.

```
root@SpaceX_1:~# traceroute -n 85.200.1.23
traceroute to 85.200.1.23 (85.200.1.23), 30 hops max, 60 byte packets
 1 85.200.0.10 5.702 ms 5.668 ms 5.634 ms
 2 85.200.1.23 22.295 ms 22.320 ms 22.000 ms
```

Figure 2: Traceroute from SpaceX_1 to NASA_3 (85.200.1.23). The packets are routed via NEWY-L2.10 (85.200.0.10) between the hosts in different VLANs.

Question 1.2

See Figure 3 for the traceroute from CHAR-host to DETR-host.

```

traceroute to DETR-host.group85 (85.105.0.2), 30 hops max, 60 byte packets
1  CHAR-host.group85 (85.103.0.2)  0.089 ms  0.049 ms  0.045 ms
2  NEWY-CHAR.group85 (85.0.4.1)  0.158 ms  0.116 ms  0.118 ms
3  BROO-NEWY.group85 (85.0.1.1)  0.243 ms  0.219 ms  0.201 ms
4  DETR-host.group85 (85.105.0.2)  0.294 ms  0.279 ms  0.273 ms

```

Figure 3: Traceroute from CHAR-host to DETR-host.

Question 1.3

We do not expect the NEWY-STLO traceroute (Figure 5) to pass through PITT since we set the weights STLO-PITT and PITT-NEWY to 27 and 36, such that the links are not used, and we haven't set up the static route yet.

The output for the CHIC-NASH traceroute (Figure 4) is different. The traceroute shows the case of load-balancing. We see that CHIC is sending to DETR and STLO at the same time since the different routes have equal OSPF weights. Then traceroute follows the path to NASH-STLO and BROO-DETR. Then it stops since it reaches its destination via NASH first (on page 3, traceroute and TTL decrementing are explained).

```

traceroute to NASH-host.group85 (85.108.0.2), 30 hops max, 60 byte packets
1  CHIC-host.group85 (85.106.0.2)  0.048 ms  0.014 ms  0.013 ms
2  STLO-CHIC.group85 (85.0.11.2)  0.113 ms  0.115 ms  DETR-CHIC.group85 (85.0.10.1)  0.090 ms
3  NASH-host.group85 (85.108.0.2)  0.157 ms  0.138 ms  BROO-DETR.group85 (85.0.2.1)  0.166 ms

```

Figure 4: Traceroute from CHIC-host to NASH-host

```

root@NEWY_host:~# traceroute STLO-host.group85
traceroute to STLO-host.group85 (85.107.0.2), 30 hops max, 60 byte packets
1  NEWY-host.group85 (85.102.0.2)  0.051 ms  0.020 ms  0.017 ms
2  CHAR-NEWY.group85 (85.0.4.2)  0.127 ms  0.111 ms  0.081 ms
3  NASH-CHAR.group85 (85.0.6.2)  0.189 ms  0.167 ms  0.142 ms
4  STLO-host.group85 (85.107.0.2)  0.242 ms  0.242 ms  0.249ms

```

Figure 5: Traceroute from NEWY-host to STLO-host

We configured the weights such that the load-balanced paths all have equal weights sum of 30 (Least Common Multiple of 2,3, and 5, which are the numbers of links for each of the three paths that should be load-balanced). The links STLO-PITT and PITT-NEWY get higher weights such that the links are not used at all by traffic, except as a backup if one link fails.

For simplicity, each link has the same weight in both directions (see Table 2).

Table 2: OSPF link weights.

router	interface	OSPF weight	router	interface	OSPF weight
BROO	BROO-L2.10	7	DETR	port_BROO	6
BROO	BROO-L2.20	7	DETR	port_CHIC	6
BROO	port_DETR	6	DETR	port_PITT	14
BROO	port_NEWY	6	NASH	port_CHAR	6
NEWY	NEWY-L2.10	7	NASH	port_PITT	10
NEWY	NEWY-L2.20	7	NASH	port_STLO	15
NEWY	port_BROO	6	PITT	port_CHAR	11
NEWY	port_CHAR	6	PITT	port_DETR	14
NEWY	port_PITT	27	PITT	port_NASH	10
CHAR	port_NASH	6	PITT	port_NEWY	27
CHAR	port_NEWY	6	PITT	port_STLO	36
CHAR	port_PITT	11	STLO	port_CHIC	15
CHIC	port_DETR	6	STLO	port_NASH	15
CHIC	port_STLO	15	STLO	port_PITT	36

Question 1.4

Traceroute uses the TTL entry to measure the path route and length. It starts with a TTL of 1. The first router which transmits the packet to the target reduces the TTL by one unit and responds with an error "Time exceeded". Now the source knows the first hop and time to get there. The source increments now for every round the TTL by one. Each layer-3 router reduces TTL, and eventually responds back when TTL gets 0. This is repeated till the destination has been first reached. This allows the number of hops and time to be determined.

For the IPv6 tunnel, each IPv6 packet is encapsulated and given an IPv4 header (with different TTL) including the address of the egress of the tunnel. It is then transmitted over the IPv4 infrastructure.

As expected, traceroute between SpaceX_2 and NASA_4 shows only the hops where the IPv6 packet is outside the tunnel and the TTL (IPv6) can be decremented (Figure 6).

```
root@SpaceX_2:~# traceroute 85:201:1:24::
traceroute to 85:201:1:24:: (85:201:1:24::), 30 hops max, 80 byte packets
 1 85:200:0:1:: (85:200:0:1::) 7.174 ms 7.058 ms 7.445 ms
 2 85:201:1:2:: (85:201:1:2::) 7.891 ms 8.327 ms 8.253 ms
 3 85:201:1:24:: (85:201:1:24::) 15.556 ms 15.455 ms 15.433 ms
```

Figure 6: IPv6 traceroute between SpaceX_2 and NASA_4

With tcpdump (in NEWY container) we can display the traffic that is routed via a defined interface. When the static route is not activated, the traffic flows over port_CHAR (see Figure 7). The port_PITT sees little traffic since the OSPF weights are unfavourable (see Figure 8). Later on, we see that the ping from SpaceX_3 in DC1 to NASA_4 in DC2 goes completely over the tunnel NEWY_to_STLO. Note: tcpdump in Figure 8, 7 and 9 have been shortened.

```
root@NEWY_router:~# tcpdump -i port_CHAR
listening on port_CHAR, link-type EN10MB (Ethernet), capture size 262144 bytes
16:40:... IP 85.152.0.1 > 85.157.0.1: IP6 85:200:0:13:: > 85:201:1:24::: ICMP6, echo request,...
16:40:... IP 85.157.0.1 > 85.152.0.1: IP6 85:201:1:24:: > 85:200:0:13::: ICMP6, echo reply,...
...
24 packets captured; 24 packets received by filter; 0 packets dropped by kernel
```

Figure 7: tcpdump on port_CHAR while pinging NASA_4 from SpaceX_3, without static route

```
root@NEWY_router:~# tcpdump -i port_PITT
listening on port_PITT, link-type EN10MB (Ethernet), capture size 262144 bytes
16:39:34.909687 IP PITT-NEWY.group85 > 224.0.0.5: OSPFv2, Hello, length 48
16:39:34.910034 IP NEWY-PITT.group85 > 224.0.0.5: OSPFv2, Hello, length 48
...
2 packets captured; 2 packets received by filter; 0 packets dropped by kernel
```

Figure 8: tcpdump on port_PITT while pinging NASA_4 from SpaceX_3, without static route

```
root@NEWY_router:~# tcpdump -i NEWY_to_STLO
listening on NEWY_to_STLO, link-type RAW (Raw IP), capture size 262144 bytes
16:35:01.569491 IP6 85:200:0:13:: > 85:201:1:24::: ICMP6, echo request, seq 17, length 64
16:35:01.571948 IP6 85:201:1:24:: > 85:200:0:13::: ICMP6, echo reply, seq 17, length 64
...
16 packets captured; 16 packets received by filter; 0 packets dropped by kernel
```

Figure 9: tcpdump on NEWY_to_STLO while pinging NASA_4 from SpaceX_3 with static route enabled

Question 1.5

We do not want the IPv6 tunnel to flow over arbitrary paths. To use the NEWY-PITT-STLO route for this, we define static routes in both directions on the three routers. This has the advantage that it is not

bound to OSPF weights. The disadvantages are that the route is fixed. If a link on the route is broken, an alternative is not dynamically switched on and the connection remains offline.

To actually see if the traffic between DC1 and DC2 flows over the intended path, a ping call can be transmitted over the IPv6 tunnel. With tpcdump we can see that the traffic flows over the PITT-NEWY connection and thus works (see Figure 9).

In Figure 10, traceroute shows again the path over the smallest OSPF weights. As expected, we do not see the routing going over PITT using the static route.

```
traceroute to 85.107.0.1 (85.107.0.1), 30 hops max, 60 byte packets
 1  85.102.0.2  0.257 ms  0.043 ms  0.024 ms
 2  85.0.4.2    0.134 ms  0.087 ms  0.101 ms
 3  85.0.6.2    1.583 ms  1.393 ms  1.362 ms
 4  85.0.12.1   2.696 ms  2.660 ms  2.659 ms
 5  85.107.0.1  3.331 ms  3.579 ms  3.544 ms
```

Figure 10: Traceroute from NEWY-host to STLO-host

Traceroute from NASA_1 in DC1 to NASA_4 in DC2 (using IPv6) shows once again just the IPv6 hops outside the tunnel (see Figure 11).

```
root@NASA_1:~# traceroute -n 85:201:1:24::
traceroute to 85:201:1:24:: (85:201:1:24::), 30 hops max, 80 byte packets
 1  85:200:1:1:: 11.431 ms 11.038 ms 11.079 ms
 2  85:201:1:2:: 11.211 ms 11.243 ms 11.227 ms
 3  85:201:1:24:: 19.377 ms 19.173 ms 19.469 ms
```

Figure 11: Traceroute NASA_1 to NASA_4

The traffic flowing on NEWY-PITT-STLO path is not just DC-to-DC traffic. There are also OSPF messages transmitted.

2 Inter-Domain Routing

Question 2.1

The command `update-source` tries to establish a iBGP session with a user-specified non-default address with its iBGP peer. By default, iBGP uses the address of the physical interface to connect the iBGP peers. In our use case, we avoid the direct use of the physical interfaces for the iBGP peers as the binding on the loopback address brings better resiliency for link failures. By using the loopback addresses of the routers, the iBGP sessions become independent on the actual port or link failures in the AS. With a functioning router, the loopback interface never goes offline .

See Figure 12 for the BGP summary of the NASH router.

```

NASH_router# show ip bgp summary
IPv4 Unicast Summary:
BGP router identifier 85.158.0.1, local AS number 85 vrf-id 0
BGP table version 2517
RIB entries 17, using 3264 bytes of memory
Peers 8, using 170 KiB of memory

Neighbor      V      AS  MsgRcvd  MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd  PfxSnt
85.151.0.1    4       85    14507    14833     0     0     0  01w2d19h      1         1
85.152.0.1    4       85    14157    14830     0     0     0  01w2d19h      1         1
85.153.0.1    4       85    14625    14834     0     0     0  01w2d19h      1         1
85.154.0.1    4       85    14086    14759     0     0     0  01w2d18h      1         1
85.155.0.1    4       85    14206    14756     0     0     0  01w2d18h      8         1
85.156.0.1    4       85    14585    14761     0     0     0  01w2d18h      1         1
85.157.0.1    4       85    14496    14827     0     0     0  01w2d19h      2         1
179.1.22.2    4       87    13303    13352     0     0     0  01w0d23h      4         8

Total number of neighbors 8

```

Figure 12: BGP summary of the NASH router.

Question 2.2

The next hop attribute of a BGP route is used to specify the next address to be able to reach the source address. By default, when sending out an eBGP route, the router changes the next hop address to its own address. When receiving an external BGP route, the next hop address remains unchanged by default (so it is still the address of the external router which sends us the eBGP route).

Once inside our network, we do not know how to reach this external router. So we would like to change the next-hop to an address that we are able to reach from everywhere inside our network, and which knows how to reach the external router sending the eBGP route.

This is why we use the `next-hop-self` command, which replaces the next-hop attribute of a BGP route, by the actual address of the router (where we apply the command). See Figure 13 for the ip bgp output of our CHAR router.

```

CHAR_router# show ip bgp
BGP table version is 385, local router ID is 85.153.0.1, vrf id 0
Default local pref 100, local AS 85
...
Network          Next Hop          Metric LocPrf Weight Path
*>i1.0.0.0/8      85.156.0.1        100      0 83 82 1 i
...
*>i72.0.0.0/8     85.156.0.1        100      0 83 82 41 43 72 i
*>i81.0.0.0/8     85.156.0.1        100      0 83 81 i
*>i82.0.0.0/8     85.156.0.1        100      0 83 82 i
*>i83.0.0.0/8     85.156.0.1        0        100      0 83 i
*>i84.0.0.0/8     85.155.0.1        0        100      0 84 i
* i85.0.0.0/8     85.151.0.1        0        100      0 i
* i               85.152.0.1        0        100      0 i
* i               85.154.0.1        0        100      0 i
*>               0.0.0.0          0        32768 i
* i               85.158.0.1        0        100      0 i
* i               85.155.0.1        0        100      0 i
* i               85.156.0.1        0        100      0 i
* i               85.157.0.1        0        100      0 i
* 86.0.0.0/8      179.1.23.2        0        100      0 88 86 i
*>i               85.157.0.1        0        100      0 86 i
*> 86.0.2.0/24    179.1.23.2        0        100      0 88 86 i
*> 88.0.0.0/8     179.1.23.2        0        100      0 88 i
*> 89.0.0.0/8     179.1.23.2        0        100      0 88 89 i
*> 90.0.0.0/8     179.1.23.2        0        100      0 88 90 i
*>i91.0.0.0/8     85.151.0.1        0        100      0 91 i
...
Displayed 53 routes and 62 total paths

```

Figure 13: BGP output CHAR router, group 85.

- We can observe a direct connection with AS 88 in CHAR through the interface with the address 179.1.23.2.
- The BGP route to AS 84 is advertised through 85.155.0.1 which is our router in DETR.
- The BGP route to AS 83 is advertised through 85.156.0.1 which is our router in CHIC.
- The BGP route to AS 86 is advertised through 85.157.0.1 which is our router in STLO, and also through AS 88 (179.1.23.2), because specific BGP policies are not implemented yet.
- We do not see the prefix of AS 87, as it was not advertising at the time we did this task. We however got the 87 prefix later.
- We can also observe that our neighbors advertise their learned routes as well (e.g. the route to AS 1).

To verify that the prefix 85.0.0.0/8 is being advertised to the CHIC router in AS 83, we can have a look at the result of the looking glass in CHIC router of AS 83 in Figure 14.

```
BGP table version is 275, local router ID is 83.156.0.1, vrf id 0
Default local pref 100, local AS 83
...
Network          Next Hop          Metric LocPrf Weight Path
*> 1.0.0.0/8      179.0.27.1        0      100      0 81 1 i
* i               83.155.0.1        0      100      0 82 1 i
...
* i83.0.0.0/8     83.153.0.1        0      100      0 i
* i               83.158.0.1        0      100      0 i
*>               0.0.0.0           0          32768 i
* 84.0.0.0/8      179.0.27.1        0      100      0 81 84 i
*>i               83.157.0.1        0      100      0 84 i
*>i85.0.0.0/8     83.158.0.1        0      100      0 85 i
*>i86.0.0.0/8     83.153.0.1        0      100      0 86 i
*>i86.0.2.0/24    83.158.0.1        0      100      0 85 88 86 i
...
*> 180.123.0.0/24 179.0.27.1        0      100      0 81 41 44 45 i
* i               83.155.0.1        0      100      0 82 41 44 45 i
Displayed 52 routes and 92 total paths
```

Figure 14: Result of the looking glass for the neighboring AS 83

By looking at the result of a traceroute from our PITT-host to AS 84's PITT-host, we can see that we have data-plane connectivity with our neighboring AS's (Figure 15).

```
root@PITT_host:~# traceroute 84.104.0.1
traceroute to 84.104.0.1 (84.104.0.1), 30 hops max, 60 byte packets
 1 PITT-host.group85 (85.104.0.2) 0.262 ms 0.046 ms 0.023 ms
 2 DETR-PITT.group85 (85.0.7.2) 0.134 ms 0.089 ms 0.100 ms
 3 179.1.21.1 (179.1.21.1) 2.661 ms 2.506 ms 2.477 ms
 4 NASH-CHAR.group84 (84.0.6.2) 2.525 ms 2.506 ms 2.581 ms
 5 PITT-CHAR.group84 (84.0.5.2) 2.821 ms 2.783 ms 2.761 ms
 6 host-PITT.group84 (84.104.0.1) 3.218 ms 2.370 ms 2.406 ms
```

Figure 15: Traceroute from the PITT-host in our AS 85 to the PITT-host in our neighboring AS 84.

3 Policy Routing

Question 3.1

We use two different BGP communities: 85:100 for customers, and 85:200 for providers and peers. These two communities are enough to export our routes. To customers, we advertise the prefix of our AS, the routes that we learn from customers, peers, and providers. To providers and peers only the prefix of our

```
route-map 86_MAP_IN permit 10
set community 85:200
```

Figure 16: Excerpt from the STLO router configuration that shows the *in route-map*. As STLO is a peer, we set the community value 85:200 to all incoming routes. We use 85:200 to tag routes from peers and providers.

```
ip prefix-list 4 seq 5 permit 85.0.0.0/8
!
bgp community-list 1 seq 5 permit 85:100
```

```
route-map 86_MAP_OUT permit 10
match community 1
!
route-map 86_MAP_OUT permit 20
match ip address prefix-list 4
```

Figure 17: Excerpt from the STLO router configuration that shows the *out route-map*. We export routes learned from customers (`match community 1`, where community-list 1 contains the community value 85:100 that we assign to customer routes) and our own prefix (`match ip address prefix-list 4`, where prefix-list 4 contains our prefix 85.0.0.0/8).

own AS and the routes that we learn from customers are exported. See Figures 16 and 17 for the details in the configuration.

The looking glass result of our direct peer AS 86 in Figure 18 shows that the routes to our customers AS 87 and AS 88 are correctly advertised as *85 87 i* and *85 88 i*, respectively. The routes to our providers AS 83 and AS 84 only show *0 83 i* and *0 84 i*. Thus, the route via our AS 85 is not advertised to our peer. Finally, the routes to our peers AS 102, AS 104, AS 106, AS 108, AS 110, and AS 112 all use our AS 86's provider AS 83. Again, the routes via our AS 85 are not advertised to our direct peer.

```
BGP table version is 4150, local router ID is 86.157.0.1, vrf id 0
Default local pref 100, local AS 86
```

Network	Next Hop	Metric	LocPrf	Weight	Path
*>11.0.0.0/8	86.155.0.1		10	0	83 82 1 i
...					
*>183.0.0.0/8	86.155.0.1	0	10	0	83 i
*>184.0.0.0/8	86.0.11.1	0	10	0	84 i
* i	86.156.0.1	0	10	0	84 i
*> 85.0.0.0/8	179.1.24.1	0	100	0	85 i
* i86.0.0.0/8	86.0.11.1	0	100	0	i
* i	86.156.0.1	0	100	0	i
* i	86.0.12.2	0	100	0	i
* i	86.158.0.1	0	100	0	i
* i	86.153.0.1	0	100	0	i
*>	0.0.0.0	0		32768	i
* i	86.151.0.1	0	100	0	i
* i	86.155.0.1	0	100	0	i
* 87.0.0.0/8	179.1.24.1		100	0	85 87 i
*>i	86.153.0.1	0	1000	0	87 87 87 87 i
...					
*>112.0.0.0/8	86.155.0.1		10	0	83 112 i

```
Displayed 81 routes and 103 total paths
```

Figure 18: The result of the looking glass for the host STLO in our direct peer AS 86.

Figure 19 shows the traceroute from our provider AS 83 to our customer AS 87.


```

root@04f92618e1cf:~# ./launch_traceroute.sh 83 87.104.0.1
Hop 1: 83.0.199.1 TTL=0 during transit
Hop 2: 179.1.17.2 TTL=0 during transit
Hop 3: 85.0.10.1 TTL=0 during transit
Hop 4: 85.0.2.1 TTL=0 during transit
Hop 5: 85.0.1.2 TTL=0 during transit
Hop 6: 85.0.4.2 TTL=0 during transit
Hop 7: 85.0.9.2 TTL=0 during transit
Hop 8: 179.1.22.2 TTL=0 during transit
Hop 9: 179.1.26.2 TTL=0 during transit
Hop 10: 87.0.7.1 TTL=0 during transit
Hop 11: 87.104.0.1 Echo reply (type=0/code=0)
Hop 12: 87.104.0.1 Echo reply (type=0/code=0)
...

```

Figure 19: Traceroute from our provider AS 83 to our customer AS 87.

Question 3.2

```

!
bgp as-path access-list 120 deny ^(81|83|87|89|91)_
bgp as-path access-list 125 permit ^(102|104|106|108|110|112)_
!
bgp community-list 1 seq 5 permit 85:100
bgp community-list 2 seq 5 permit 85:200
!
route-map 125_MAP_OUT permit 10
  match community 1
  set community 125:102 125:104 125:106 125:108 125:110 125:112
!
route-map 125_MAP_OUT permit 20
  match ip address prefix-list 4
  set community 125:102 125:104 125:106 125:108 125:110 125:112
!
route-map 125_MAP_IN permit 10
  match as-path 120
!
route-map 125_MAP_IN permit 20
  match as-path 125
  set community 85:200
  set local-preference 100
!

```

Figure 20: route map definition at BROO router

First, let's have a look at Figure 20 for the route map set-up in the BROO router.

In the first line:

The incoming BGP routes whose attribute as-path is ending with one of the AS of list 120 (*81 83 87 89 or 91*) are refused at the MAP_IN.

Those are the AS's located in the same region of the mini-internet, and we don't want them to reach us through the IXP 125.

For the second line:

The incoming BGP routes whose attribute as-path is ending with one of the AS of list 125 (*102 104 106 108 110 or 112*) are permitted at the MAP_IN.

Those are the AS's located in another region of the mini-internet, and we want them to reach us through the IXP 125.

Doing both the deny and the permit may be redundant, but we thought that it would be more robust in case of one malicious AS advertising us some route that we do not want.

The community list 1 (with community tag 85:100) is associated to the BGP routes we received from our customers.

The community list 2 (with community tag 85:200) is associated to the BGP routes we received from our peers and customers.

Line 5-10:

We define the policy for the BGP routes we advertise.

First we do a match, to select only the BGP routes learned from our customers (community list 1). Then we set community tags on the outgoing BGP routes, such that the IXP relays them to our peer AS's connected through IXP 125.

We also need to advertise our own prefix. To do that, we have to define a chained route-map (with the same name, but another sequence number, here 20, which is bigger, and will be executed after the first route-map out).

This chained route map, matches a prefix list (4). This prefix list is composed only of 85.0.0.0/8 for now, but we can add here other prefixes that we would like to advertise (e.g. 3.4).

Line 11-end:

We define the policy for the incoming BGP routes.

The first route map is matching the BGP routes from list 120. This list is denying the BGP routes which have an as-path terminating with one of the AS listed.

Then, the second route map in is also a chained route map, which maps the BGP routes from list 125 (allowing routes terminating with one of the listed AS's).

On those incoming routes, we set a community tag of 85:200, which is associated to BGP routes received from peers or providers (peers in that case).

Then we set a local preference of 100. This is to ensure that our network prefers the path learned from customers over the routes learned from peers over the routes learned from providers (local pref set to 1000 for routes learned from customers, and to 1 for routes learned from providers).

We can verify that we deny the BGP routes learned through peer connection (through IXP) with the AS's located in the same region from the bgp output in Figure 21.

```
BR00_router# show ip bgp
BGP table version is 3313, local router ID is 85.151.0.1, vrf id 0
Default local pref 100, local AS 85
```

Network	Next Hop	Metric	LocPrf	Weight	Path
...					
*>181.0.0.0/8	85.155.0.1		1	0	84 81 i
...					
*>183.0.0.0/8	85.155.0.1		1	0	84 83 i
...					
* 185.0.0.0/8	85.157.0.1	0	100	0	i
...					
*>187.0.0.0/8	85.158.0.1	0	1000	0	87 i
...					
*>189.0.0.0/8	85.153.0.1		1000	0	88 89 i
* i	85.158.0.1		1000	0	87 89 i
...					
*>191.0.0.0/8	85.153.0.1		1000	0	88 89 91 i
* i	85.158.0.1		1000	0	87 89 91 i
...					

Displayed 79 routes and 89 total paths

Figure 21: The direct route (through IXP125) from the stub AS 81, 83, 87, 89 and 91, which are in the same region is denied. Instead, we receive the routes traffic via our neighboring AS's

We can also verify that we do not advertise our prefix through peer connection (through IXP) to the AS's located in the same region as our AS, by looking at the output of the looking glass of BROO router in AS 81 in Figure 22.

We verify that we advertised our prefix through peer connection (through IXP) to the AS's located in another region, by looking at the output of the looking glass of BROO router in AS 106 in Figure 23.

Finally, we do a traceroute from the measurement container of AS 111 to verify that the route taken goes indeed through IXP125 (Figure 24).

```

BGP table version is 4081, local router ID is 81.151.0.1, vrf id 0
Default local pref 100, local AS 81
...
  Network          Next Hop          Metric LocPrf Weight Path
...
*> 85.0.0.0/8      179.0.27.2          100      0 83 85 i
*                  179.0.28.2          100      0 84 85 85 85 85 i
*                  179.0.25.2          50       0 82 83 85 i
...
Displayed 162 routes and 222 total paths

```

Figure 22: The looking glass for router BROO of AS 81 shows that we do not connect with AS 81 through the IXP. Otherwise we would observe shorter entries of the form 0 85 i in the AS-path field of the BGP advertisements. Our prefix is advertised to AS 81 through our providers.

```

BGP table version is 7928, local router ID is 106.151.0.1, vrf id 0
Default local pref 100, local AS 106
...
  Network          Next Hop          Metric LocPrf Weight Path
...
*> 85.0.0.0/8      180.125.0.85         0          0 85 i
...
Displayed 161 routes and 249 total paths

```

Figure 23: The looking glass output for the router BROO of AS 106 in our neighboring region reveals that our prefix is advertised through the IXP 125. Advertisements via other paths would have additional third-party AS's inserted in the AS-path field of the BGP advertisements.

```

root@04f92618e1cf:~# ./launch_traceroute.sh 111 BROO-host.group85
Hop 1: 111.0.199.1 TTL=0 during transit
Hop 2: 111.0.9.1 TTL=0 during transit
Hop 3: 111.0.7.2 TTL=0 during transit
Hop 4: 179.1.56.1 TTL=0 during transit
Hop 5: 110.0.4.1 TTL=0 during transit
Hop 6: 110.0.1.1 TTL=0 during transit
Hop 7: 85.101.0.2 Echo reply (type=0/code=0)
Hop 8: 85.101.0.2 Echo reply (type=0/code=0)
...

```

Figure 24: The traceroute from the STLO host in AS 111 to the BROO host in our AS 85 shows that the traffic crosses IXP125 (there is a direct change from the routers of AS 110 with IP addresses 110.x.x.x to the BROO host of our AS with the IP address 85.101.0.2).

Question 3.3

In order to influence other ASs in the network to route traffic into our subnet preferably via our router CHIC (connected to our provider AS 83) instead of DETR (connected to our provider AS 84), we configured our eBGP advertisements such that the default route selection rules of BGP make our intentions happen. We could influence the inbound control via the *AS path length* or the *MED value*. Because the *AS path length* has higher priority in the BGP route selection process – and because we want other ASs to route via AS 83 even if the path is longer – we focus on the former. See Figure 25 for the unmodified BGP advertisements to AS 83 and Figure 26 for the artificially increased *AS path* that we advertise to AS 84 in order to make the routed via AS 84 unattractive.

There are drawbacks to that approach. It is not difficult to detect that we manipulate the path length by network operators of other ASs. That's a problem for us if we consider ourselves sneaky and don't follow contracts which those ASs (e.g., with AS 84 who as a provider has an interest to route traffic to us). Additionally, the effect of our manipulation can be reversed by other ASs. They can simply subtract the additional paths again.

```

BGP table version is 4561, local router ID is 83.151.0.1, vrf id 0
Default local pref 100, local AS 83
...
  Network          Next Hop          Metric LocPrf Weight Path
...
*>184.108.0.0/25    83.155.0.1          100          0 82 22 i
*>185.0.0.0/8       83.158.0.1          0          1000 0 85 i
*>185.108.0.0/25    83.155.0.1          100          0 82 21 i
*>185.108.0.0/26    83.158.0.1          1000         0 85 i
*>185.108.0.64/26   83.158.0.1          1000         0 85 i
*>186.0.0.0/8       83.157.0.1          100          0 84 86 i
...

```

Figure 25: The looking glass output for the router BROO of our first provider AS 83 shows the normal BGP advertisement to our AS. See the looking glass output of AS 84 in Figure 26 on how we control inbound traffic to our AS 85.

```

BGP table version is 7897, local router ID is 84.151.0.1, vrf id 0
Default local pref 100, local AS 84
...
  Network          Next Hop          Metric LocPrf Weight Path
...
  84.108.0.64/26    0.0.0.0           0           32768 i
*>185.0.0.0/8       84.153.0.1         0          110 0 85 85 85 85 i
*>185.108.0.0/25    84.155.0.1         20          90 0 81 21 i
* 1                84.156.0.1         10          90 0 82 21 i
*>185.108.0.0/26    84.153.0.1         110         0 85 85 85 85 i
*>185.108.0.64/26   84.153.0.1         110         0 85 85 85 85 i
*>186.0.0.0/8       84.158.0.1         0          110 0 86 i
...

```

Figure 26: The looking glass output for the router BROO of our second provider AS 84 shows our BGP advertisements with artificially increased *AS path length* that makes the routes to our prefix via AS 85 unattractive.

Question 3.4

When checking the looking glass of other groups, we found routes to 85.108.0.0/25 with a path originating from AS 21. As a quick counter measure, we advertised the two more specific prefixes in the /26 subnet: 85.108.0.0/26 and 85.108.0.64/26. Our fix can easily be circumvented by the attackers. They could still attract the traffic for other prefixes that we did not fight back, e.g., 85.108.0.128/25. They could also attack the same prefix again by advertising even more specific prefixes, e.g., 85.108.0.0/27, 85.108.0.32/27, 85.108.0.64/27, and 85.108.0.96/27. In addition, we have to add a static route in the NASH router to the interface "host" to guide traffic to NASH host correctly since the NASH router does not know what to do with /26 prefix. In the long run, only the filtering of the rogue route advertisements by neighboring ASs would help (we did not attempt this solution as – hopefully in contrary to the real internet – the direct neighbors include tier-1s that are controlled by the attacking TAs).

Bonus question 3.5

At the beginning, after establishing the VPN connection, only the 85.200.30.0/24 prefixes in the VLAN 30 are reachable (Figure 27).

```

dbolli@DESKTOP-E9PHKGU:~$ netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags    MSS Window  irtt Iface
0.0.0.0        172.17.64.1     0.0.0.0         UG        0 0          0 eth0
85.200.30.0    0.0.0.0         255.255.255.0   U          0 0          0 tap0
172.17.64.0    0.0.0.0         255.255.240.0   U          0 0          0 eth0

```

Figure 27: Netstat of local device before any changes

We define the default gateway in our local device (Figure 28) to the router interface (e.g. NEWY-L2.30),

with an exception for the address of the snowball VPN tunnel, so that we can still connect over the internet. The disadvantage is that we lose connection with everything that is outside our mini-internet (e.g. Google 8.8.8.8). From Figure 29 it is visible that we are now able to reach external AS's, such as AS 87.

```
dbolli@DESKTOP-E9PHKGU:~$ netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask          Flags      MSS Window  irtt Iface
0.0.0.0          85.200.30.2     0.0.0.0          UG         0 0      0 tap0
82.130.102.244   172.17.64.1     255.255.255.255 UGH        0 0      0 eth0
85.200.30.0      0.0.0.0         255.255.255.0    U         0 0      0 tap0
172.17.64.0      0.0.0.0         255.255.240.0    U         0 0      0 eth0
```

Figure 28: Netstat of local device with default gateway modifications

```
dbolli@DESKTOP-E9PHKGU:~$ traceroute -n 87.155.0.1
traceroute to 87.155.0.1 (87.155.0.1), 30 hops max, 60 byte packets
 1 85.200.30.2 22.534 ms 24.439 ms 25.033 ms
 2 85.0.4.2 24.793 ms 24.729 ms 24.554 ms
 3 85.0.6.2 24.509 ms 24.390 ms 24.309 ms
 4 179.1.22.2 27.920 ms 28.187 ms 29.623 ms
 5 87.155.0.1 29.680 ms 29.688 ms 29.749 ms
```

Figure 29: Traceroute from one local device to DETR's router of the AS 87, using the VPN

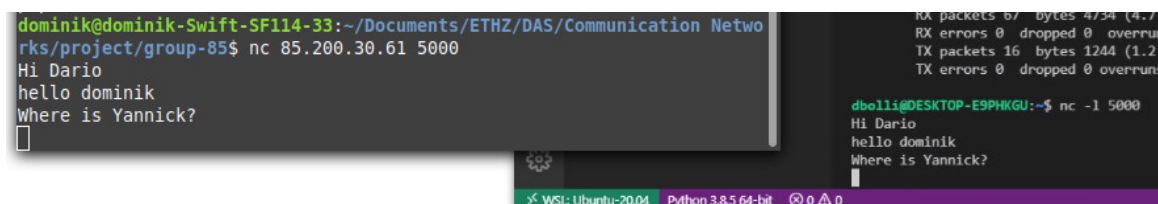
First, our configuration restricted the access to the global internet (e.g. Google 8.8.8.8 not reachable). This means that it is not possible to ping an actual server in the real internet. In addition, the ping command can be specified to go over a certain network device (here tap0 for the OpenVPN), which gives the same ping-times as before. Secondly, we can use traceroute and resolve the addresses within our mini-net to see if it matches as we expect. Generally, this means that the traceroute will not show the first hop of the local internet connection since it goes through the VPN tunnel.

We perform a traceroute between two VPN endpoints within AS 85 (Figure 30). As expected, the traceroute shows that the traffic goes directly over layer-2 switches from one VPN client to another. If the connection had been established over the real internet, we could see a lot more hops including the first hop of every local internet connection.

```
dbolli@DESKTOP-E9PHKGU:~$ traceroute -n 85.200.30.55
traceroute to 85.200.30.55 (85.200.30.55), 30 hops max, 60 byte packets
 1 85.200.30.55 25.527 ms 27.490 ms 27.792 ms
```

Figure 30: Traceroute from one local device to another local device, using the VPN

Finally, to test if we can transmit traffic over AS 85 from one group member to another, we used Netcat (nc). This way we started a discussion between the two local devices as seen below (Figure 31).



The image shows two terminal windows side-by-side. The left window is titled 'dominik@dominik-Swift-SF114-33' and shows a Netcat listener on 85.200.30.61. It receives a connection from 85.200.30.61 and the user 'Dario' logs in. The user sends the messages 'hello dominik' and 'Where is Yannick?'. The right window is titled 'dbolli@DESKTOP-E9PHKGU' and shows a Netcat listener on -l 5000. It receives a connection from 85.200.30.61 and the user 'Dario' logs in. The user sends the same messages: 'hello dominik' and 'Where is Yannick?'. The status bar at the bottom indicates 'WSL: Ubuntu-20.04' and 'Python 3.8.5 64-bit'.

Figure 31: Discussion between Dominik's local device (terminal on the left) and Dario's local device (Zoom teleconferencing window on the right) using TCP connection, and the VPN.