LINGI2142 - Computer networks: configuration and management

Nikita Tyunyayev - Soukéina Bojabza - Arnaud Lahousse

I. Introduction

As a part of the course LINGI2142 : "Computer networks: configuration and management", we have been asked to create a complete network. We used as a template the OVH network, and we implemented it on ipmininet. In the network we had to implement all the routers and configure the protocols : the border gateway protocol, anycast, some communities and the security. Below, there are the details of the network implementation.

We will first describe the architecture of the network. Then, we will present our IP addressing plan. We will then show you an example of a router configuration on ipmininet. Next, we will discuss our BGP configuration choices and then the justification of the IGP costs we set in the network.

After that, we will present you what we did in order to set communities in our network, add an implementation of anycast and address the security question.

II. Presentation of the network

The network implemented is the Asia part of OVH, with some of the Europe and North America parts to buckle the loop. An illustration can be found on the figure 1. On this schema, the boxes represent the routers while the clouds are the clients of our network, which are Vodafone, Equinix and NTT.

In practice, the clients are modeled by routers outside the network. So, in the schema, each cloud figure is in fact a router to which a host is connected. The routers of a same client are in the same autonomous system (AS). We did not connect these routers between them because the AS configuration must be managed by the client (we are only in charge of OVH).

In order to have redundancy, we have put two routers by city. We also connected them in a way that we can always find another route to reach the destination. More precisely, there is always a way to reach the Asia part since this is where Equinix and NTT are located and there will therefore be an important traffic towards them.

Note: In the following, we will often refer to the routers as $< City > 1$ or $< City > 2$. The suffix 1 is the router located on the left, and then 2 is the one on the right. Except for San José (SJO) where 1 is above and 2 below.

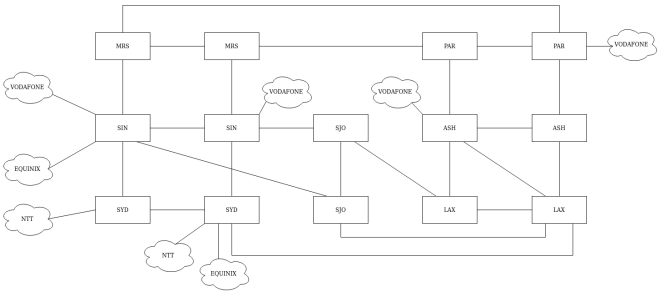


Fig. 1: Implemented network

III. Addressing

A. IPv6

For the IPv6 addressing we have set the first 48 bits common to the whole network : `1627:6000:0000::`. The next 4 bits are used to differentiate the continents: `0` for Europe, `1` for North America and `2` for Asia. The network mask of each router is `/64`.

For the addresses of the links, they have a `/64` mask.

For the peers attached to the network, we just changed the first 32 bits : `1627:6100:0000::` for Vodafone, `1627:6200:0000::` for Equinix and `1627:6300:0000::` for NTT.

The figure 2 shows an illustration of our IPv6 addressing plan.

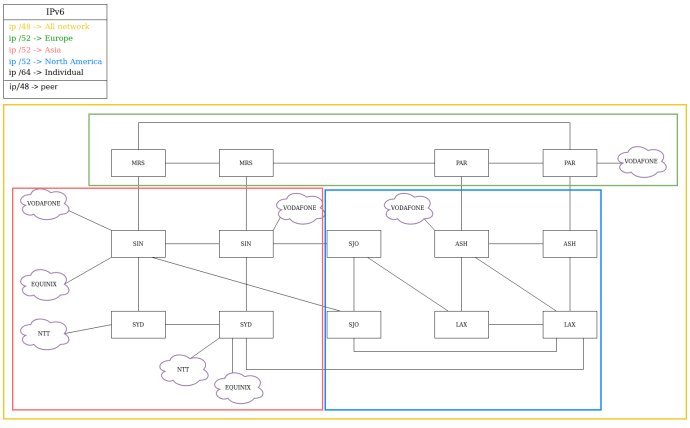


Fig. 2: IPv6 addressing plan

B. IPv4

For the IPv4 addressing, the first 20 bits are the same in the whole network: `162.76.240..` and a network mask of `/20`. Instead of taking the next 8 bits by continent like we

did in IPv6, we decided to assign the next 4 bits to the cities: 1 for Marseille, 2 for Paris, 3 for Singapore, 4 for Sydney, 5 for Los Angeles, 6 for San José and 7 for Ashburn. Finally, each router has a network mask of `/32`.

About the addresses of the links, we had the choice between `/31` and `/30` masks. We have chosen `/30` because it's generally better supported by protocols such as OSPF, it is the most used in point to point links and because we have enough IPv4 addresses to do that.

For the peers attached to the network, we assigned the address `160.76./24` and the next 8 bits are used to differentiate the clients: 7, 8 and 9 for Vodafone, Equinix and NTT respectively.

The figure 3 shows an illustration of our IPv4 addressing plan.

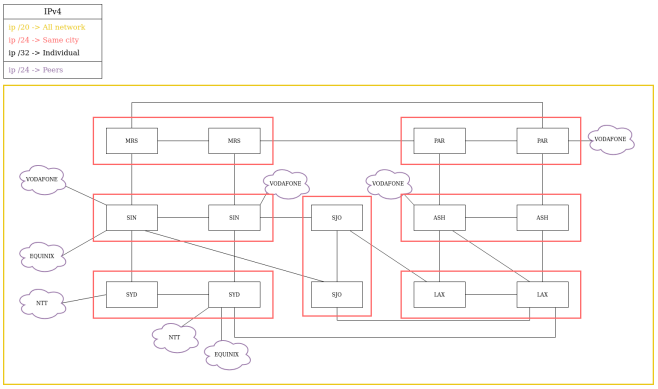


Fig. 3: IPv4 addressing plan

IV. Configuration of the routers and peers

On each router of the network and on the routers representing the peers, we have set the configuration with `RouterConfig`. It allows to cancel the OSPF and OSPF6 on each router and so to manage how we want to configure the routers properly. After that, we have added daemons OSPF6, OSPF and BGP. You will find an example of implementation on the router MRS1 on the figure 4.

```
MRS1 = self.addRouter("MRS1",config=RouterConfig, lo_addresses=[europe_ipv6 + "000::/64", MRS_ipv4 + "253/32"])
MRS1.addDaemon(OSPF6)
MRS1.addDaemon(OSPF)
MRS1.addDaemon(BGP,debug=("neighbor",))
```

Fig. 4: Implementation of the router MRS1

V. iBGP, eBGP and route reflectors

The BGP configuration is shown on figure 5.

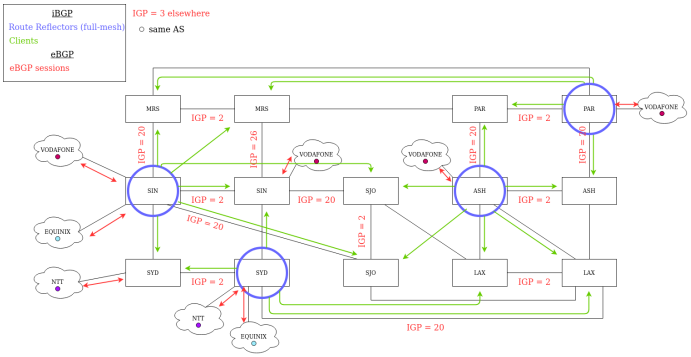


Fig. 5: BGP configuration

A. iBGP

There are four route reflectors (RR) in our network (circled in blue), all connected in full-mesh. The clients of these RRs are indicated by the green arrows. All the clients are connected to two different RRs in order to have redundancy. Furthermore, we always connected them to the closest RRs to avoid any issue such as loops in the network.

B. Choice of the route reflectors

The choice of the route reflectors was made in the following way : since we have 3 continents, we wanted at least a route reflector in each of them in order to orient the internal traffic. Indeed, since Vodafone is located in the three continents, we wanted a way to tell the routers to send the packets destined to a prefix belonging to Vodafone to the one located in their continent; because in practice, we would not want to reach it in North America if we are in Europe for example.

For the route reflectors themselves, we tried to put them the closest to the continents borders. As said before, each BGP client is connected to two RRs for redundancy. To be more precise, the first route reflector is one located in the same continent as him and the other is on another one. This is only true for the routers in Europe and America because there is only Vodafone so we found it necessary to have a link with other continents (again to ensure redundancy).

For the RRs of Asia, we decided to put two RRs because the three clients are present there and on several routers. Therefore, with two RRs, the traffic would be more distributed between the routers. The RRs are put on the routers that are connected to two clients, to make sure that the packets will go through them.

C. Hierarchical route reflectors ?

We chose to not use hierarchical route reflectors because we found it useless in our case. Indeed, we only have four route reflectors so the number of sessions ($\frac{n\times(n-1)}{2} = 6$) between them is acceptable. Using hierarchy would complexify something that was not too complicated from the beginning.

Furthermore, the choice of using full-mesh was more obvious given the configuration of our network : as said previously, we want at least one route reflector in each continent, which already makes 3 route reflectors. We just added an another

one to distribute the asian traffic more efficiently. Also, we want the three continents to communicate. Hierarchical route reflectors would not have met these requirements.

D. eBGP

The eBGP sessions are represented by the red arrows. There is a session between the clients (Vodafone, Equinix and NTT) and the router to which they are directly linked. Indeed, the clients are located in other ASes so the only place where the eBGP sessions can be placed is on the paths between those ASes and OVH.

VI. IGP COST

As you can see in the figure 5, we have set IGP costs between all the routers inside the network. At the beginning, we had set the IGP cost between the routers of the same city (ex: between the two `SIN` routers) to 1, of the same continent but not the same city to 3 and between continents to 11. These choices were made in order to keep the traffic local: since the cost is high to go in another continent, the routers are less likely to choose these routes. With the same logic, same cities are more proned to communicate with each other since the cost is minimal. However, there is a problem with ipmininet: if we do that ipmininet crash. The problem is on the side of ipmininet rather than ours.

Therefore, we have set IGP cost values to 2 between the same city, 3 for the cities in the same continent and 20 between the continents. There is one exception: we set a cost of 26 between MRS2 and SIN2 in order to avoid a route deflection.

VII. COMMUNITIES

We were asked to define some BGP communities in our network. By looking at existing examples, like [2] for NTT, we found interesting to use the following ones:

- Set the local-pref to 200
- AS-prepending x1
- AS-prepending x2

Let's describe each of them.

A. Set local-pref to 200

This is a pretty basic feature that we provide in our network. Our peers can set community `200:200` on their prefix to set the local-pref to 200.

B. AS prepending x1

AS-prepending is useful in many cases. For instance, it enables our peers to set up back-up routes. The AS-prepend x1 community is `1:100`.

C. AS prepending x2

Same idea as before we had a level of control by providing a "prepend two times" community which is `2:100`.

D. Implementation

In practice, the communities are implemented by using Route Maps. You can see the definition of our Route Maps on figure 8.

```
bgp community-list standard local_pref_200 permit 200:200
bgp community-list standard prepend_x1 permit 1:100
bgp community-list standard prepend_x2 permit 2:100
!
route-map general_route_map permit 30
 match community local_pref_200
 set local-preference 200
 on-match next
!
route-map general_route_map permit 100
!
route-map general_route_map2 permit 10
 match community prepend_x1
 set as-path prepend 1
 on-match next
!
route-map general_route_map2 permit 20
 match community prepend_x2
 set as-path prepend 1 1
 on-match next
!
route-map general_route_map2 permit 100
!
line vty
!
```

Fig. 6: route-map configuration

As you can see two route-maps are used one on input and the other one on output. general_route_map handles the input and the set local-pref community. On the other hand general_route_map_2 handles the output and is used for AS-prepending. The last permits are used to counter the implicit deny at the end of every route-maps. We use on-match next in our conditions to enable the use of multiple communities on the same prefix.

In the script folder, you can find the following scripts to deploy the communities:

- `BGP_ccl_COMM_NAME.py` to create the community-lists.
- `BGP_PX1_COMML_RMNAME_SEQ.py` to add "append x1" action to a route-map.
- `BGP_PX2_COMML_RMNAME_SEQ.py` to add "append x2" action to a route-map.
- `BGP_COMML_LPREF_RMNAME_SEQ.py` to add set local-pref action to a route-map.
- `BGP_NEIGHBOR_RMAP_INOUT.py` to apply a route-map to a peer.

A test is provided at the end of the script which set the local-pref and the prepend X2 communities for routes from the "VODASH1" router. You can use the script BGP_SET_ANY_COMM_RMNAME_SEQ.py to add the action set community to a route-map.

```
*> 1627:6000:0:1ffa::/64
                    fe80::4068:12ff:feca:aacb
                                                0 1 1 1 3 ?
```

Fig. 7: AS prepending

```
*> 1627:6000:0:1ffa::/64
                    fe80::7cd4:feff:fe5e:1ea3
                                          0    200      0 3 ?
```

Fig. 8: local-pref

As you can see the communities are working well.

VIII. Anycast

Inside the network we have implemented anycast. We have modeled it by adding two servers: one connected to the router of SJO2, another one to the router of PAR2. An illustration can be found on the figure 9, where the anycast servers are in orange.

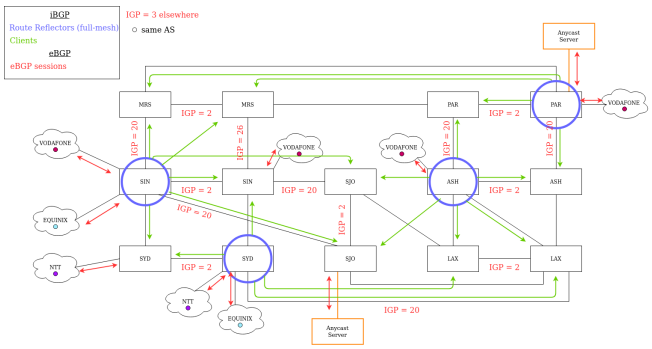


Fig. 9: Anycast servers

We would have implemented a DNS server on it but we are limited by ipmininet so we needed to find something else. For the implementation, we had to choose between two pertinent daemons to put on the router: we hesitated between OSFP and BGP.

At first, we wanted to use OSPF because the ”hello request” which allows to verify if a router is still alive is more frequently sent than with BGP, every 1 second against 180 seconds for BGP. However, in terms of security, OSPF is not secure. In fact, if the router which represents the server has a problem, it could send OSPF packets in the network and therefore break the configuration, which is unacceptable. With BGP, we can use route-maps to filter imports.

Therefore, we decided to implement BGP on the router and decrease the time of fault detection to address the timing problem: we have set the rate of ”keep alive packet” to 4 seconds. Now, if a client wants to reach the server, he chooses the one with the ”shortest” path, but if the link between this server fails he will be redirected to the other server.

To show you an example of how the anycast works, let’s take the following situation: the client Vodafone in paris connected to the router wants to reach the server. The network directly connects Vodafone to the closest server, which is the server S2 here. The figure 10 below shows the output of the `traceroute` command to reach the server.

```
traceroute to S2 (1627:6000:0:3::) from 1627:6100:0:aaa::1, 30 hops max, 24 byte
 packets
 1  VDFPAR2 (1627:6100:0:aaa::2)  0.198 ms  0.074 ms  0.294 ms
 2  PAR2 (1627:6000:0:ffa::2)  0.223 ms  0.175 ms  0.167 ms
 3  S2 (1627:6000:0:3::)  0.509 ms  0.215 ms  0.08 ms
root@vagrant:~/shared# traceroute6 S1
traceroute to S1 (1627:6000:0:3::) from 1627:6100:0:aaa::1, 30 hops max, 24 byte packets
 1  VDFPAR2 (1627:6100:0:aaa::2)  0.094 ms  0.066 ms  0.058 ms
 2  PAR2 (1627:6000:0:ffa::2)  0.09 ms  0.067 ms  0.063 ms
 3  S2 (1627:6000:0:3::)  0.225 ms  0.067 ms  0.064 ms
```

Fig. 10: Traceroute6 between Vodafone Paris and the servers before the link failure

The servers S1 and S2 have the same loopback address, this is why both addresses have the same destination S2.

If the link between S2 and PAR2 fails, the network adapts automatically the route towards the other server, here S1. The figure 11 below shows the output of `tracertoute` after the link failure.

```
traceroute to S2 (1627:6000:0:3::) from 1627:6100:0:aaa::1, 30 hops max, 24 byte packets
 1  VDFPAR2 (1627:6100:0:aaa::2)  0.228 ms  0.077 ms  0.068 ms
 2  PAR2 (1627:6000:0:ffa::2)  0.191 ms  0.18 ms  0.173 ms
 3  ASH2 (1627:6000:0:220::2)  0.186 ms  0.177 ms  0.411 ms
 4  LAX2 (1627:6000:0:1022::2)  0.322 ms  0.38 ms  0.253 ms
 5  SJO2 (1627:6000:0:1220::1)  0.329 ms  0.223 ms  0.176 ms
 6  S2 (1627:6000:0:3::)  0.104 ms  0.183 ms  0.301 ms
root@vagrant:~/shared# traceroute6 S1
traceroute to S1 (1627:6000:0:3::) from 1627:6100:0:aaa::1, 30 hops max, 24 byte packets
 1  VDFPAR2 (1627:6100:0:aaa::2)  0.212 ms  0.171 ms  0.201 ms
 2  PAR2 (1627:6000:0:ffa::2)  0.185 ms  0.193 ms  0.08 ms
 3  ASH2 (1627:6000:0:220::2)  0.176 ms  0.313 ms  0.199 ms
 4  LAX2 (1627:6000:0:1012::2)  0.196 ms  0.687 ms  0.301 ms
 5  SJO2 (1627:6000:0:1220::1)  0.199 ms  0.205 ms  0.189 ms
 6  S2 (1627:6000:0:3::)  0.236 ms  1.013 ms  0.187 ms
```

Fig. 11: Traceroute6 between Vodafone Paris and the servers after the link failure

To implement this service we have created two routers with the configuration `RouterConfig` and added a daemon BGP. Both routers have the same loopback address in IPv4 and IPv6 and they are both in the same autonomous system. As we have chosen to use BGP on the router, we have initialized an eBGP session between the routers simulating both the servers and routers of SJO2 and PAR2.

The implementation is shown in the figure 14.

```
#servers

S1 = self.addRouter("S1", config=RouterConfig, lo_addresses=[server_ipv6 + "::/64", server_ipv4 + "1/32"])
S2 = self.addRouter("S2", config=RouterConfig, lo_addresses=[server_ipv6 + "::/64", server_ipv4 + "1/32"])

#Adding BGP daemons to manage failures

S1.addDaemon(BGP,address_families=(AF_INET6(redistribute=['connected']),AF_INET(redistribute=['connected'])))
S2.addDaemon(BGP,address_families=(AF_INET6(redistribute=['connected']),AF_INET(redistribute=['connected'])))

# S1.addDaemon(OSPF6)
# S2.addDaemon(OSPF6)

self.addAS(64512, (S1,))
self.addAS(64512, (S2,))

l_S1_SJO2 = self.addLink(S1,SJO2, igp_metric=3)
l_S1_SJO2[S1].addParams(ip=(server_ipv6 + "a1a::1/64",server_ipv4 + "5/30"))
l_S1_SJO2[SJO2].addParams(ip=(server_ipv6 + "a1a::2/64",server_ipv4 + "6/30"))

l_S2_PAR2 = self.addLink(S2,PAR2, igp_metric=3)
l_S2_PAR2[S2].addParams(ip=(server_ipv6 + "a1a::3/64",server_ipv4 + "9/30"))
l_S2_PAR2[PAR2].addParams(ip=(server_ipv6 + "a1a::4/64",server_ipv4 + "10/30"))

ebgp_session(self,S1, SJO2)
ebgp_session(self,S2, PAR2)
```

Fig. 12: Implementation of anycast

To Secure the link between the routers and the servers, we have implemented a password to the link. We would use a SHA256 but ipmininet don’t allow that, so we use MD5.

Of course the AS containing the servers is a private AS and is therefore never announced outside OVH. We have hidden the AS number to the other network. To do this we have implemented route map on the routers connected to the server (This part is not in the code above because to do that we use "pexpect" to launch the command directly in FRRouting with the script BGP_rm_aspath_ASN_RM_SEQ.py). The figure below show you the situation where we don't hide the AS number :

```
*> 1627:6000:0:3::/64
                    fe80::ec9b:47ff:fe40:5c5b
                                                0 1 64512 ?
```

Fig. 13: Not hidden number of AS

And our situation :

```
1627:6000:0:3a1a::/64
                    fe80::46b:98ff:fee7:ea6a
                                                0 1 ?
```

Fig. 14: hidden number of AS

IX. Security

When designing our network, we had to think about some ways to make it secure, in order to avoid attacks coming from outside. In this section, we will present you the technique we used to achieve this goal.

A. Password on BGP

For the security, we have implemented a password between the peers connected to the network and the routers at the edge of it. For the moment, we have used MD5 hash because FRRouting can only support that. We are aware that this hash is not the most secure, which is why we wanted to implement a SHA256 hash at the beginning.

There are two interesting things to note about this password:

- the hash is a mix of a secret password and the name of the peer.
- Why would we use a password on BGP since it is in a higher layer than TCP ? Answer: because it can prevent the TCP *reset attack*. *Reset attack* is an attack deployed by a "hacker" or a firewall who sends reset packets to the router to interrupt the connection. With the password, we are protected against the attackers.

B. Password on OSPF

To avoid the same problems encountered with BGP, we wanted to implement a password between all routers of the network. However, FRRouting cannot support secured passwords. So, the only security we have implemented is *plain text authentication*. This method is not the most secured but it is the one supported by FRRouting. Like with BGP, we wanted to use SHA256 hash but FRRouting doesn't support it. Furthermore, we implemented the plain text authentification only for OSPFv2 as it's not supported on OSPFv3. We are using one password per continent.

To use ospf password simply add the option password="yourPassword" to a link the same way as you will set igp_metric for instance.

C. TTL check

To prevent BGP connexions from unwanted peers, we decided to use the neighbor PEER ttl-security hops NUMBER FRROUTING command. It prevents connexion from peers which are more than NUMBER hops away. The base TTL for ipv6 packets is 64, we increased it to 255. By doing this we are sure that we effectively don't connect to peers further than NUMBER hops. NUMBER is set to 2 in our network.

X. Limitations of our design

A. IPv4 and eBGP on ipmininet

At the beginning of the project, when BGP wasn't implemented, the ipv4 worked, but now that we have implemented BGP, the ipv4 works partially, for instance ping4all doesn't work totally, we don't understand why.

B. Convergence time

The network takes a little while to converge totally (Maximum one minute) so when we want to use command or test our network we need to wait that the network converges. In sight of the number of routers, protocol and scripts to launch, it seems to take a reasonable time.

C. Ping6all

If we launch a ping6all, there are only 40 pings which match and 16 drop which is normal since hosts from the the same peer are not connected in their own AS in our configuration. To fix this problem we can link all the routers of the same peer together but we are not in charge of the peer, so we let this task to network engineers of the peer.

XI. How to launch the code

To respond to all the specifications, we have done some modifications to ipmininet, so in the file of the project we have added our version of ipmininet. Furthermore, to configure router with command "FFRouting" we use a library python : `pexpect`. So before launching our code, you need to install our ipmininet like this :

```
sudo pip install -e MyIpmininet/
```

and install `pexpect` :

```
sudo pip install pexpect
```

XII. Conclusion

With this project, we are now able to design a network: from the routers organisation to the BGP configuration, passing by the addressing plan. We also took some measures to ensure the security of the network. We then upgraded the network to use the BGP communities. And we finally made it support anycast and designed it to detect failures efficiently.

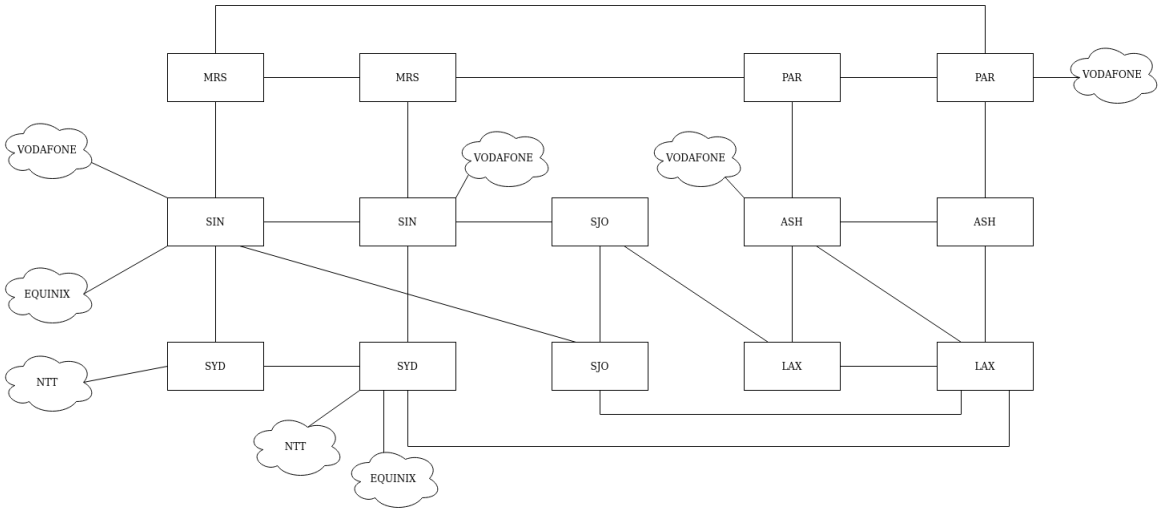
All of that was implemented on ipmininet in order to see if our design choices were pertinent.

REFERENCES

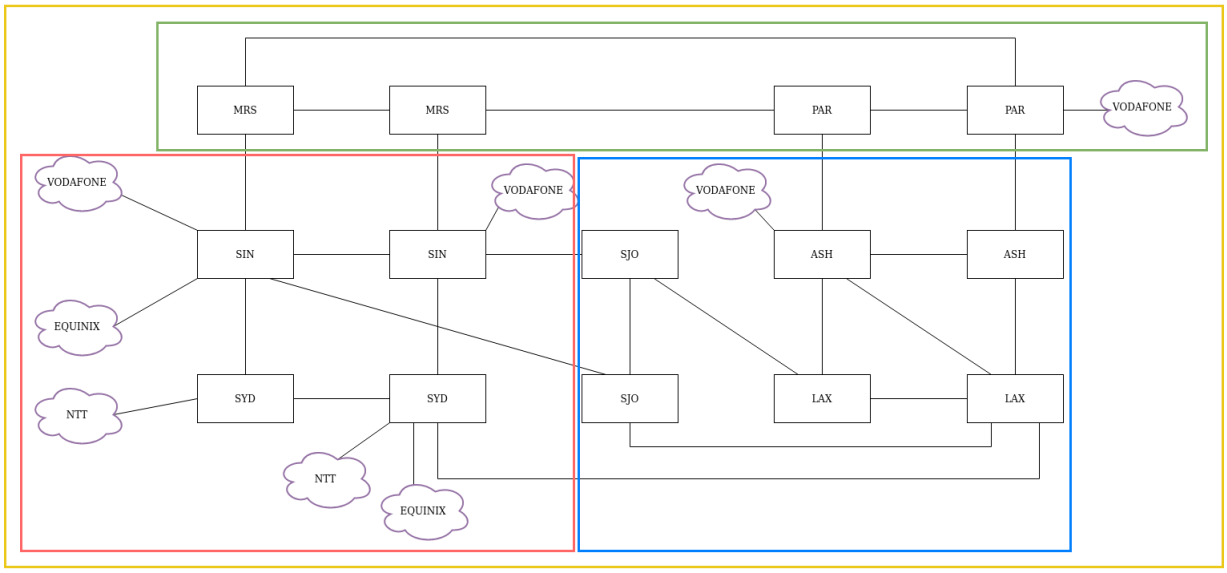
[1] https://en.wikipedia.org/network
[2] ttp://docs.frrouting.org/en/latest/
[3] https://www.gin.ntt.net/support-center/policies-procedures/routing/
[4] https://networklessons.com/cisco/ccnp-route/introduction-to-route-maps
[5] https://community.cisco.com/t5/technology-and-support/ct-p/technology-support
[6] https://ipmininet.readthedocs.io/en/latest/
[7] https://stackoverflow.com/questions/27941036/telnet-to-login-with-username-and-password-to-mail-server?fbclid=IwAR1bq8O-2AUTw7naOGe0UXn_GNh5mGxEDE7sOR_I93oO4RB_8PI6Hl1AV4Q
[8] https://pexpect.readthedocs.io/en/stable/
[9] https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.116.6367&rep=rep1&type=pdf
[10] https://stat.ripe.net/0
[11] https://www.peeringdb.com
[12] https://bgpview.io/
[13] https://docs.ovh.com
[14] http://weathermap.ovh.net/

1 Annexes

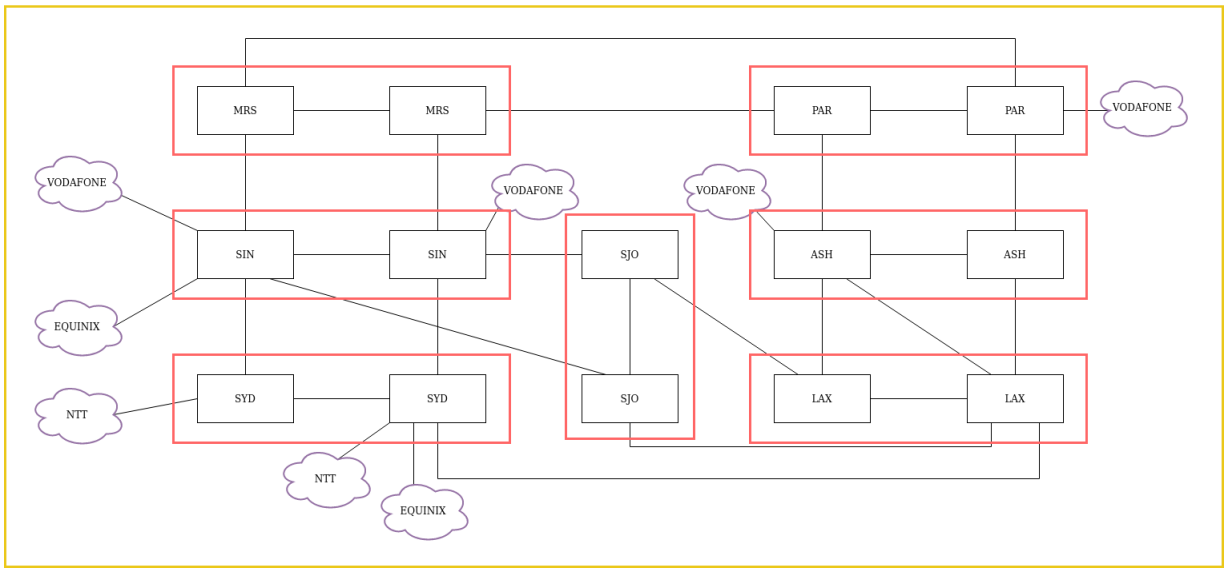
Here are the extended images :



IPv6
ip /48 -> All network
ip /52 -> Europe
ip /52 -> Asia
ip /52 -> North America
ip /64 -> Individual
ip/48 -> peer

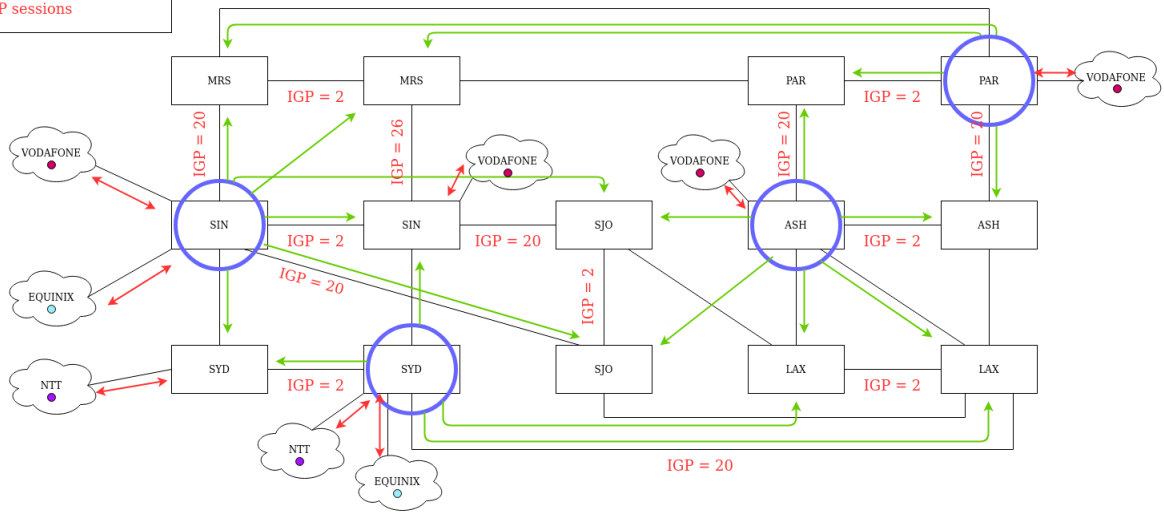


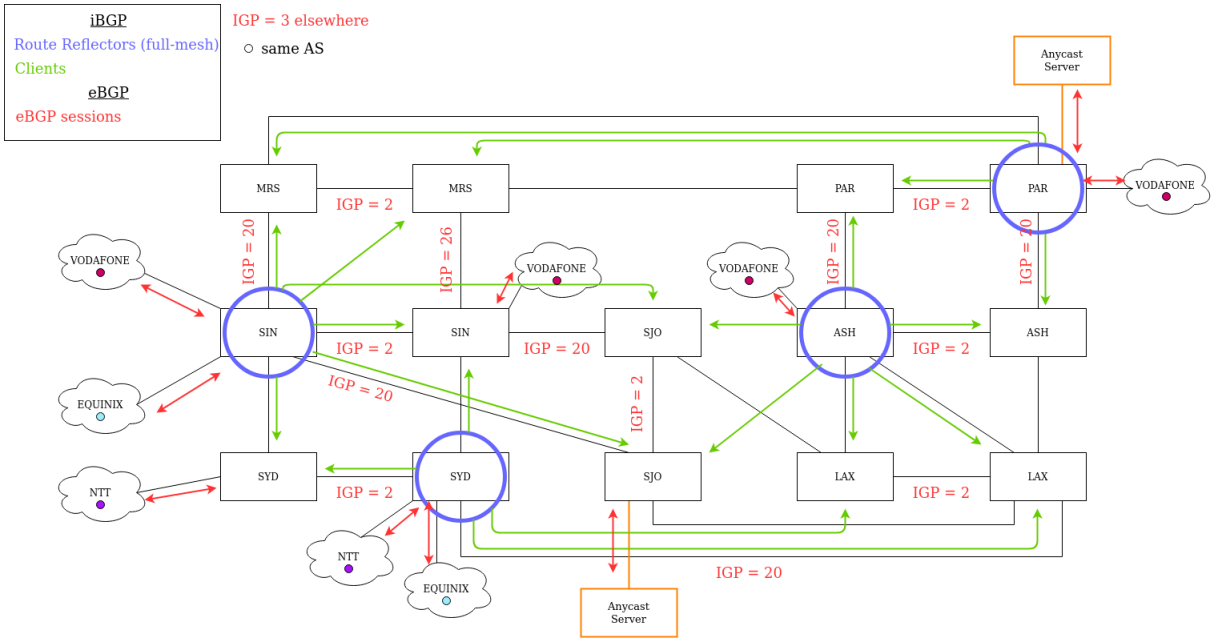
IPv4
ip /20 -> All network
ip /24 -> Same city
ip /32 -> Individual
ip /24 -> Peers



iBGP
Route Reflectors (full-mesh)
Clients
eBGP
eBGP sessions

IGP = 3 elsewhere
○ same AS




iBGP
Route Reflectors (full-mesh)
Clients
eBGP
eBGP sessions
IGP = 3 elsewhere
same AS
Anycast Server
MRS
MRS
PAR
PAR
VODAFONE
IGP = 2
IGP = 20
IGP = 26
VODAFONE
VODAFONE
IGP = 20
IGP = 20
VODAFONE
SIN
SIN
SJO
ASH
ASH
IGP = 2
IGP = 20
IGP = 2
IGP = 2
EQUINIX
IGP = 20
IGP = 2
NTT
SYD
SYD
SJO
LAX
LAX
IGP = 2
IGP = 2
NTT
EQUINIX
IGP = 20
Anycast Server