LINGI2142 – OVH Project Report

November 19, 2020

Hadrien Libioulle
NOMA: 10041700

Jan Suchara
NOMA: 17252000

Olivier Latteur
NOMA: 51541700

***Abstract*—OVHcloud is the Europe's largest cloud service provider in terms of both customer base and data center size. Its operation spans across 19 countries and 4 continents.**

The goal of this project was to choose a part of OVHcloud network, implement it in IPMininet and setup BGP, security and anycast inside that segment of the network. At first, we used OVH Network Weather Map [15] which provides and overview of the whole infrastructure together with real-time traffic information. We decided to model a part of European backbone network. The European backbone consists of 5 large data centers and Internet exchange points in Belgium, Britain, France and Germany. We have selected routers in Britain, France and Germany. We find this particular portion of the network interesting because it is most dense, shares direct links to all other data centers and a significant number of world's largest providers like Google or Amazon.

Next chapters describe our efforts to simulate the network and decision about IPv4 and IPv6 addressing plans, BGP setup with hierarchical route reflectors and BGP communities, anycast servers, etc. We also encountered a few issues where some features required certain changes to IPMininet, e.g. template files for daemon configuration, or could not be implemented at all. But all of them were only minor complications and did not prevent us from successfully completing our mission.

***Index Terms*—IPMininet, OVH ,OSPF, iBGP, eBGP, BGP communities, BGP Security, Anycast**

I. Network Model

For the network, we decided to base ours on the OVH European network and to take a part of it. We have selected the following routers:

- Roubaix:
  - `rbx_g1_nc5`
  - `rbx_g2_nc5`
- Gravelines:
  - `gra_g1_nc5`
  - `gra_g2_nc5`
- Frankfurt:
  - `fra_fr5_sbb1_nc5`
  - `fra_fr5_sbb2_nc5`
  - `fra_1_n7`
  - `fra_5_n7`
- Paris:
  - `par_gsw_sbb1_nc5`
  - `par_th2_sbb1_nc5`
- London:
  - `lon_thw_sbb1_nc5`
  - `lon_drch_sbb1_nc5`

A. External ASes and their routers

We have selected the following external `ASes`:

- Amazon (`AS16509`):
  - `amazon_r1` connected to `par_th2_sbb1_nc5`
  - `amazon_r2` connected to `lon_thw_sbb1_nc5`
- Cogent (`AS174`):
  - `cogent_r1` connected to `par_th2_sbb1_nc5`
  - `cogent_r2` connected to `par_gsw_sbb1_nc5`
  - `cogent_r3` connected to `lon_thw_sbb1_nc5`
- Google (`AS15169`):
  - `google_r1` connected to `par_th2_sbb1_nc5`
  - `google_r2` connected to `par_gsw_sbb1_nc5`
  - `google_r3` connected to `fra_5_n7`
- Telia (`AS1299`):
  - `telia_r1` connected to `fra_1_n7`
  - `telia_r2` connected to `fra_5_n7`
  - `telia_r3` connected to `lon_thw_sbb1_nc5`
- Vodafone (`AS1273`):
  - `vodafone_r1` connected to `par_th2_sbb1_nc5`
  - `vodafone_r2` connected to `par_gsw_sbb1_nc5`
  - `vodafone_r3` connected to `fra_1_n7`
  - `vodafone_r4` connected to `fra_5_n7`

Remark: For the IPMininet implementation, we have added intermediate routers between external ASes and the routers specified previously. We thought it was easier and better for the static route corresponding to prefix of OVH.

B. Address plan

Our network is compatible with IPv4 as well as IPv6. We have organized our IP addresses so as to minimize the number of used IPv4 addresses while keeping the network and structuring our IPv6 addresses future-proof.

*1) **IPv4**:* Given that the ASN of our network is `16276`, the network's prefix is `192.148.0.0/22` (`16` most significant bits correspond to `16276`). As we don't have enough

Name	IPv4	IPv6
Google_r1	8.8.4.1/32	2001:4860:0:1::/128
Google_r2	8.8.4.2/32	2001:4860:0:2::/128
Google_r3	8.8.4.3/32	2001:4860:0:3::/128
Voda_r1	2.16.35.1/32	2001:5000:0:1::/128
Voda_r2	2.16.35.2/32	2001:5000:0:2::/128
Voda_r3	2.16.35.3/32	2001:5000:0:3::/128
Voda_r4	2.16.35.4/32	2001:5000:0:4::/128
Cogent_r1	2.58.4.1/32	2001:550:0:1::/128
Cogent_r2	2.58.4.2/32	2001:550:0:2::/128
Cogent_r3	2.58.4.3/32	2001:550:0:3::/128
Telia_r1	2.255.248.1/32	2001:2000:0:1::/128
Telia_r2	2.255.248.2/32	2001:2000:0:2::/128
Telia_r3	2.255.248.3/32	2001:2000:0:3::/128
Amazon_r1	3.5.128.1/32	2001:4f8:b:1::/128
Amazon_r2	3.5.128.2/32	2001:4f8:b:2::/128
Amazon_h1	3.5.3.2/32	2001:4f8:1:3:::2/128
Google_h1	8.8.2.2/32	2001:4860:1:4::2/128
Voda_h1	2.16.34.2/32	2001:5000:1:5::2/128
Cogent_h1	2.58.8.2/32	2001:550:1:4::2/128
Telia_h1	2.255.4.2/32	2001:2000:1:4::2/128

Table I
EXTERNAL ROUTERS AND HOST IP ADDRESSES

bits to set up a subdivision of addresses according to geographical location, we have divided the addresses between the different routers and servers in a random way but with the following layout (see Figure III):

- The 23^{th} bit is assigned to determine whether the device is a host (0) or a router (1). We call it "host bit".
- The 24^{th} bit is assign to determine if the address is a loopback address (1) or a subnet address (0)

  For subnet addresses:
  - The next 6 bits are used to assign a specific address for each router or host.
  - The 31^{th} bit is what we call the "free bit", it allows us to differentiate whether we are on the host (Host==0) side of the subnet or on the router (Host==1) side.
  - Unfortunately IPMininet does not authorize to use the 32^{th} bit[1]. This bit is therefore free.

  For loopback addresses:
  - The last 8 bits remaining are used to assign a specific address for each router or host.

With this addressing plan we consider that our network is future-proof due to the fact that it can accommodate up to 255 devices (router and host) and up to 64 subnets.

2) ***IPv6****:* Given that the ASN of our network is `16276`, our network's prefix is `2001:41d0:0::/32` which leave us with 32 bits to assign. In contrast with IPv4, with IPv6 we have distributed the addresses according to geographical position.

The IPv6 address plan is the following (see Figure IV):

- 3 bits are used to define the continental position
- 5 bits are used to define the country

Prefixes	Country
2001:41D0:0:0000::/56	France
2001:41D0:0:0100::/56	Germany
2001:41D0:0:0200::/56	The United Kingdom
2001:41D0:0:1F00::/56	Inter-European connections

Table II
IPV6 COUNTRY PREFIXES

- The 57^{th} bit is assign to determine if the address is a loopback address (1) or a subnet address (0).
- The 58^{th} bit is assigned to determine if the device is a host (0) or a router (1). We call it "host bit".
- The 6 remaining bits are used to assign specific address for each router/host/submet of the same country.

II. THE OSPF CONFIGURATION

A. Configuration

OSPF in our network uses the default configuration. Every IGP cost is set to 1 because the network is concentrated in a small part of the world. Each time interval or delay (retransmission interval, transmission delay, hello interval and dead interval) has been set to its default value. In fact, we could have set the transmission delay based on a real world delay observed between two routers but we have not done this. For the other values, the default configuration is sufficient.

B. Response to Network Failures

As soon as a failure occurs, OSPF should find another route towards the destination (unless another route does not exist ...). With our IPMininet implementation, this does not work because of the static route advertisement of the OVH prefix through BGP. The whole explanation is available in section [VI-H][2].

III. THE IBGP ORGANIZATION

A. Route Reflectors Organization

We have implemented a 2-tier Route Refletor (RR) network. The top tier is composed of 2 routers (for backup purposes):

- `gra_g1_nc5`
- `gra_g2_nc5`

The lower tier RR routers are connected to these 2 routers as clients. The bottom layer is composed of these routers (always by peers for backup purposes):

- Roubaix:
  - `rbx_g1_nc5`
  - `rbx_g2_nc5`
- Frankfurt:

[1]When we assign a `/31` subnet to a link with 2 devices connected, IPMininet tells us that it needs a 2 address prefix space and the prefix space assigned is too small for it.

[2]If you are on the PDF, you can click on the reference between the brackets to be redirected to the section.

- `fra_fr5_sbb1_nc5`
- `fra_fr5_sbb2_nc5`

Finally, the rest of the routers are clients of the lowest tier RRs. Here are the different sessions:

- Clients of `rbx_g1_nc5`/`rbx_g2_nc5`:
  - `par_gsw_sbb1_nc5`
  - `par_th2_sbb1_nc5`
  - `lon_thw_sbb1_nc5`
  - `lon_drch_sbb1_nc5`
- Clients of `fra_fr5_sbb1_nc5`/`fra_fr5_sbb2_nc5`:
  - `fra_1_n7`
  - `fra_5_n7`

Router	IPv4
lon_thw_sbb1_nc	192.148.3.0/32
lon_drch_sbb1_nc5	192.148.3.1/32
gra_g1_nc5	192.148.3.2/32
gra_g2_nc5	192.148.3.3/32
fra_fr5_sbb1_nc5	192.148.3.4/32
fra_fr5_sbb2_nc5	192.148.3.5/32
fra_1_n7	192.148.3.6/32
fra_5_n7	192.148.3.7/32
rbx_g1_nc5	192.148.3.8/32
rbx_g2_nc5	192.148.3.9/32
par_gsw_sbb1_nc5	192.148.3.10/32
par_th2_sbb1_nc5	192.148.3.11/32

Table III
IPV4 ADDRESSES OF SELECTED ROUTERS

Router	IPv6
lon_thw_sbb1_nc	2001:41D0:0000:0280::/64
lon_drch_sbb1_nc5	2001:41D0:0000:0281::/64
gra_g1_nc5	2001:41D0:0000:0080::/64
gra_g2_nc5	2001:41D0:0000:0081::/64
fra_fr5_sbb1_nc5	2001:41D0:0000:0180::/64
fra_fr5_sbb2_nc5	2001:41D0:0000:0181::/64
fra_1_n7	2001:41D0:0000:0182::/64
fra_5_n7	2001:41D0:0000:0183::/64
rbx_g1_nc5	2001:41D0:0000:0082::/64
rbx_g2_nc5	2001:41D0:0000:0083::/64
par_gsw_sbb1_nc5	2001:41D0:0000:0084::/64
par_th2_sbb1_nc5	2001:41D0:0000:0085::/64

Table IV
IPV6 ADDRESSES OF SELECTED ROUTERS

B. Specific BGP configuration for IPMininet

There are two types of routers: **internal** routers and **border** routers. Border routers are setup with a static route corresponding to the prefix of our network (`192.148.0.0/16` and `2001:41D0::/48`) and they redistribute this route through BGP (using this bgp command: `redistribute static`). Internal routers are setup in a 2-tier route reflector configuration as explained earlier. Some of the "leaf" routers of this configuration are RR of the border router[3] corresponding to them (`par_th2_sbb1_nc5` is the RR of `par_th2_border` and so on, ...). External ASes are setup in the same way but with an iBGP full-mesh instead of RR configuration.

[3]Border routers should be distinguished from the internal routers because static routes are set on them to advertise our network. Therefore, these border routers should have only one link to our network in order to let OSPF/iBGP decide of the paths taken by the packets in our network. We could have done our implementation without these specific routers but we thought it was cleaner.

IV. BGP SECURITY

A. Introduction

BGP does not directly include mechanisms that control and secure BGP sessions between individual peers. However, guidelines to mitigate different types of known attacks against this protocol exists.

Most common threats against this protocol include session hijacking and various methods of Denial-of-Service attacks. Following chapter describes the measures we implemented to secure the network with provided functionality of IPMininet.

Unfortunately, there is no possibility to configure BGP daemon from the python script and make these changes persistent. This lead us to changing a template file *bgpd.mako* used by IPMininet to generate *bgpd* configuration. Adding commands to this file allowed us to execute specific commands on routers in our network.

B. MD5 Authentication

An adversary could reset the BGP session between two peers via spoofing TCP RST packets or inject packets and carry-out man-in-the-middle attack. This threat could be prevented by enabling Message Digest5 (MD5) authentication on a TCP connection between two BGP peers.

Every segment sent on a TCP connection will contain MD5 digest produced by applying the MD5 algorithm to specified parts of TCP packet and a key or password that is known to both sides. Upon segment reception, receiver recalculates the digest from the same data and validates it by comparing the two digests.

A password was set for each BGP neighbor in the network. Down side to the approach with editing the template file is that all the passwords were identical.

C. TTL Security

TTL security protects the network from BGP session spoofing and/or DoS attacks when an attacker floods the network with packets. In the latter case, even session authentication would not help to protect the router because digest for each packet would have to be calculated before rejecting it and this process is resource consuming.

We enabled TTL security with hop count of 1 on all border routers and their directly connected eBGP neighbors. This was again achieved by editing the IPMininet's template file *bgpd* configuration. We added an if statement checking whether the neighbor's AS number is the same as the current router's one. If it was, TTL security command was applied.

D. Limiting the Number of Prefixes

In practice, it is recommended to introduce the limitation of the number of installed prefixes from router's neighbors to decrease CPU and memory usage. In our set up we set the limit of 50 prefixes for each eBGP neighbor. Number of prefixes for iBGP peers was not set.

E. Access Control List

Common security practice is to specify hosts which are allowed to instantiate connection with the given router to limit potential attack surface of an adversary. To achieve this we used IPTables/IP6Tables daemons in IPMininet. We created a chain of rules for forwarding list which disables routing of so-called bogon IPs - addresses [17] which should not appear on public internet. Rules for input and output list was also created but that introduced problems with connection to the routers via xTerm and SSH in IPMininet so it was dropped to ensure functionality of our topology.

V. BGP COMMUNITIES

A. Introduction

As a reminder, a BGP community is a BGP attribute that can spread from one autonomous system to another and allows to tag routes with a 32-bit number. These tags provide capability for modifying BGP routing policy.

We will not explain the common communities (Local-as, Internet, No-Advertise and No-Export [1]) which are already implemented by default.

In order to be able to control some of the traffic in our network, we have decided to equip our network with communities and modify routing policies.

In the next sections, we will discuss `Do not announce` community, `Local-pref` community, `Prepend` community, and `Learn from` community.

B. `Do not announce to` community

The `Do not announce to` community allows to deny routes that have been tagged with this community to some part of the network or to some peers.

In our case, we use this functionality for two purposes:

* One is to ensure that only traffic between different peers transits through our network and it does not allow peers to use our network as a shorter path to reach another part of its own network.
* The second is to allow our peers/customers to specify if they do not want a specific route to be announced to specific peers.

To disable network traffic between same AS, we tag routes coming from the peer 'x' with the corresponding community that does not allow announcing them to 'x'. When a router connected to 'x' peer will receive a tagged route with "do not announce to x", it will not forward this prefix to peer 'x'.

The same applies whenever a peer does not want to advertise a specific route ('y') to some other peer, it simply tags it with a "do not announce to y" and routers directly connected to 'y' peers will not advertise these routes to 'y'.

The communities associated with each peer are the following:

Community	Do not advertise to
(16276:2010)	Telia
(16276:2020)	Cogent
(16276:2030)	Vodafone
(16276:2040)	Google
(16276:2050)	Amazon

C. `Local-Pref` community

The local-pref community allows customers/peers to specify their local-pref to our network and therefore to influence the decision process inside the network. We decided to let the peers choose between two local-pref values: LOW and HIGH. If the customer does not wish to specify a local-pref, it will be set to the default value.

When a router directly connected to a peer receives a route tagged with a local-pref community, it will then apply the local-pref associated to the right community to the route.

The communities associated with local-pref are as follows:

Community	Local-pref
(16276:7100)	LOW
(16276:7200)	HIGH
/	DEFAULT

D. `Prepend` community

Prepend communities allow peers to prepend their AS path if they wish, which means that our network will add one or more copies of an AS number to the left side of the AS path announced by the peer. This results in a longer AS path and therefore a less used route due to the BGP decision process.

We can imagine that an external autonomous system with several connections to our network would want one of these connections to be used only as a back-up connection. It could therefore prepend the AS path only on the back up connection and not on the primary ones. The traffic would then preferably routed through the other connections instead of the back up connection.

The communities associated with prepend are as follow:

Community	Description
(16276:411)	Prepend 1x
(16276:412)	Prepend 2x
(16276:413)	Prepend 3x

E. `Learn From` community

The "Learn from" community allows to know if a route has been learned by a customer or a peer. This could be very useful if a route was known thanks to a peer and a customer. In this situation we will probably prefer to use the customer route (cost could be less). However in our case, the network is a transit network and we mainly work with peers and a few hosts meaning this community is not very useful. We have therefore implemented it just to be able to support many customers in the future.

F. Implementation in IPMininet

With the current implementation of IPMininet it was not possible to introduce following features to our network model:

* `Local-pref` community: When the command "set local-preference" is applied in the route map where a route matches a specific community (by the use of an IPMininet function) its seems it denies the route anyway no matter if there is a match and this causes that the external routes cannot be reached anymore.

  It might seem that the solution would be to make sure that a route map authorizes to announce the routes if no match is found. Unfortunately, even by adding a default route map it does not seem to work.

  we have therefore decided to leave the IPMininet code related to this community in the source file but we commented it out.
* `Not advertise to` community: In order to block routes that are tagged with the `Do not announce to` community, we need to add a `route-map` that denies routes that match a specific community. Unfortunately, we have observed that the route maps defined by the `deny` IPMininet function does not seems to operate properly and is disturbing the rest of your network.

  We have therefore decided to leave this part of the implementation using route maps commented in the IPMininet code in order to not interfere with the main purpose of our network and its behaviour. However, the communities will still be tagged and therefore ready to be used if the route maps issue is solved.
* `Prepend` community: We did not succeed to apply a specific prepending according to a community match in IPMininet. This community is therefore not implemented in our IPMininet code.

VI. Anycast Servers

A. Introduction

Anycast is a network addressing and routing methodology based on the concept of One-to-One-of-Many. One anycast destination has multiple paths to two or more endpoints (the anycast servers). The paths and endpoints are hidden from the user's point of view and the choice of the path/endpoint is delegated to the routers that form the network. Actually, it is widely used for services like CDN (Content Delivery Network) or DNS where the user does not care to which server his request goes but the user actually wants a fast response with low latency.

B. IPv4 vs. IPv6

IPv6 was built with anycast in mind meaning that a certain number of IPv6 addresses are reserved for anycast purposes (127 to be precise) and these anycast addresses (reserved on each subnet prefix) MUST NOT be used for unicast purposes (this could lead to unexpected behavior). IPv4 was not built with anycast in mind but we can implement quite simply.

C. How to implement it?

Actually in both cases (IPv4 and IPv6), the principle is to use the same IP (v4 and/or v6) address for different endpoints and to advertise it from all the location at the same time. Routers inside the network will believe it is just corresponding to multiple paths toward a same destination and they will choose their preferred path based on their local preference (lowest metric/cost based on the IGP in general, but we could imagine some much more complex decision processes depending on the network administrator needs).

Anycast packets are not especially tagged and therefore, they are not distinguishable from unicast packets. It is good to notice that in some special cases, different packets from the same user could arrive to different destinations. And because of that, an anycast service should not be used in parallel with TCP (UDP is preferable because the TCP state transfer can be complex).

In general, these anycast addresses are set as loopback addresses of the servers for the same reason reason as the router's loopback address (to be independent from the interfaces and their configuration/status). Actually, it could be useful to have two loopback addresses (for each IP protocol): one for the anycast service **and** one for maintenance purposes.

D. Advantages

Thanks to the delegation to the routers of the path selection, some advantages can be highlighted:

- Reduced network load (less routers and links used)
- Simplified network configuration
- Network redundancy
- Load balancing
- Reduced latency
- More security (DDoS are less effective)

E. Two different implementations

1) Basic implementation:
   The same network is advertised from multiple places
   $\Rightarrow$ Multiple entries for the same prefix in BGP/OSPF tables

⇒ Each router will choose the best route based on local_pref, AS_PATH, ...

2) Advanced implementation:
   - Configure the network for load balancing between servers (and not between paths)
   - Static routing (implement specific paths depending on the source's location)
   - ...

F. Discussion of the two main implementations

1) Static routing: It is the easiest way to implement an anycast service by setting up static routes on the routers connected to the anycast server(s) in order for this server to be known by the whole network.

- **Pros (+)**:
  - Simplest way to configure (only static route on connected router)
  - No risk of running another protocol (and daemon) on the server side
- **Cons (-)**:
  - No response to server failure

2) Daemon based anycast server: More complex but can help to cope with server and link failures. The basic concept is to run a daemon (most of the time, it will be `bgpd` for security reasons. `ospfd` is a good solution too but it could lead to flood problems in the network if the server "looses his mind") on the server to advertise its anycast address to the network.

- **Pros (+)**:
  - Server is the route originator (good response to server failure)
- **Cons (-)**:
  - Server up and running does not mean that service is running ⇒ a guard daemon is needed to be sure the service is running
  - No mechanism to withdraw route if the service is unusable ...

Actually, every problem of this implementation can be solved with a specific daemon[4] that check the status of the service and update BGP accordingly. It is the best way to implement an anycast server.

G. Implementation in IPMininet

In this implementation, we have chosen the basic one (the closest server is the chosen one). We have put 3 servers in different locations :

- 1 in Germany
- 2 in France

This choice is based on the Route Reflectors position. Actually, the only protocol running on the servers is BGP and therefore the anycast servers should be directly connected to their RRs.

H. Network Failures

*1) **What is happening**:* In case of a failure, the traffic should be redirected to another server if the link/path to the closest server is down. Instead of this, we are encountering strange behaviors when an anycast server becomes unavailable:

- Packets are looping between border routers and their corresponding internal routers.
- From a host, the destination seems to be unreachable but this destination is reachable from inside the network (i.e. from a host, if the main destination is reachable through `par_4` and `par_2` then the output of `traceroute` and `paris-traceroute` stops after `par_2` but if I check the same destination from `par_2` then it is working fine).

This problem is caused by the fact that we are hiding the internal side of our network to the outside world (routes advertised by servers are tagged with the `no-export` community) then we are advertising only our prefix[5] (`192.148.0.0/16` and `2001:41D0:0000::/48`). When a server has a failure, the routers connected to them believe that they can reach it again through the border routers (the advertised network prefix collides with the IP address of the anycast server. When the server is down, the route leading to it is not discarded because all routers believe they can reach the server through the network prefix route advertised via BGP by the border routers)[6] announcing the network prefix of OVH and so, packets are looping between border routers and routers believing that they can still reach this server through the border routers. This could be solved by denying BGP routes corresponding to the prefix of OVH in order to stop the advertisement of our own prefix inside our network but the `deny` and `filter` functions from IPMininet do not seem to work properly (the OVH prefix still appears in BGP tables inside of the OVH network)[7].

*2) **What was expected**:* Without these problems, the network should react in a simple manner: as soon as the path/route toward the closest network is not usable anymore, a new route toward the new closest server from the current router is calculated (thanks to OSPF and

[4]Like this one: https://github.com/unixsurfer/anycast_healthchecker. It can check if the service is running and send withdraw/update to Bird (a router daemon) in case the service is down

[5]This prefix is announced inside and outside our network. Actually, we should not advertise it inside our network and so we wanted to use route-maps to solve this problem but as soon as we use a route-map to deny/filter this prefix, then all prefixes from outside our network are also filtered and denied ...

[6]This problem can occur with any kind of failure inside the network. Routers will know via `OSPF` that links/routers/servers are down but, in some cases, they will believe they can still reach them via the border routers.

[7]We also tried by tagging routes from border routers corresponding to the OVH prefix with the `no-advertise` community but again, it seems to not work and the `matching` clause does not do its work. All routes coming from the border routers are tagged with this community leading to the OVH network to be isolated from the external ASes

the Dijkstra's algorithm, or another algorithm depending on the daemon/protocol that is used as the IGP). The new selected server could be the same as before (if it is still the closest from the current location). The "closest" server is expressed in terms of the IGP cost of the path to the server.

3) ***Possible Workarounds****:*

- A first possible workaround would be to not use this static advertising of our prefix and just forward the routes provided by the BGP sessions of our servers but we thought that advertising specific routes (`/128` in `IPv6` and `/32` in `IPv4`) was not a good practice to keep small BGP tables. Only network prefixes should be advertised through a eBGP session.
- Another possible workaround would be to use `ospf(6)d` instead of `bgpd` on the servers. Then, routes would expired after some time and the problem would be solved. `ospfd` is not used because of security reasons as explained earlier.

Remark: When we disable the prefix announcement, the network is unavailable from outside but the problem is solved and the response to failure works as expected.

VII. Conclusion

The main goal of this project was to build a model of a selected part of OVH network, propose a possible configuration of that network segment and finally implement it inside IPMininet with additional features, namely BGP communities, setup security measures and an anycast service.

In this report, firstly, we have described the network part of the OVH network we have chosen and how we have modelled it (addressing plan, OSPF configuration and iBGP/eBGP configurations). Then, for the creative part, we have described the implementation of BGP communities to support a transit service on the OVH network. Afterwards, we have talked about the BGP security and how to secure BGP sessions from outside attacks. Eventually, we have described the implementation of an anycast service inside the OVH network. For these three creative parts, we have discussed how to implement it in real-life and how we have implemented it in IPMininet.

References

[1] BGP Community | No Export | No Advertise | Local AS – IpCisco. (2020, March 01). Retrieved from https://ipcisco.com/lesson/bgp-community/

[2] Anycast – Wikipedia. (2020, November 17th). Retrieved from https://en.wikipedia.org/wiki/Anycast

[3] Host Anycasting Service – IETF. (2020, November 17th). Retrieved from https://tools.ietf.org/html/rfc1546

[4] Reserved IPv6 Subnet Anycast Addresses – IETF. (2020, November 17th). Retrieved from https://tools.ietf.org/html/rfc2526

[5] IP Version 6 Addressing Architecture – IETF. (2020, November 17th). Retrieved from https://tools.ietf.org/html/rfc4291

[6] Build your own anycast network – Ripe.net. (2020, November 17th). Retrieved from https://labs.ripe.net/build-your-own-anycast-network

[7] How is anycast implemented ? – StackExchange. (2020, November 17th). Retrieved from https://stackexchange.com/how-is-anycast-implemented

[8] Implementing Anycast in IPv4 Networks – CiteSeerX. (2020, November 17th). Retrieved from http://citeseerx.ist.psu.edu/implementing-anycast-in-ipv4-networks

[9] BGP Documentation – FRRouting. (2020, November 17th). Retrieved from http://docs.frrouting.org/en/latest/bgp.html

[10] Cisco's BGP Documentation – Cisco. (2020, November 17th). Retrieved from https://www.cisco.com/configuring-a-basic-bgp-network.html

[11] Cisco's OSPF Documentation – Cisco. (2020, November 17th). Retrieved from https://www.cisco.com/ospf_configuration_guide

[12] Cisco's BGP Documentation – Cisco. (2020, November 17th). Retrieved from https://www.cisco.com/configuring-a-basic-bgp-network.html

[13] FRRouting's OSPFd Documentation – FRRouting. (2020, November 17th). Retrieved from http://docs.frrouting.org/en/latest/ospfd.html

[14] FRRouting's OSPF6d Documentation – FRRouting. (2020, November 17th). Retrieved from http://docs.frrouting.org/en/latest/ospf6d.html

[15] OVHcloud Weather Map. (2020, November 17th). Retrieved from http://weathermap.ovh.net/

[16] BGP community documentations – FRRouting. (2020, November 17th) http://docs.frrouting.org/en/latest/bgp.html#clicmd-bgpcommunity-liststandardNAMEpermit|denyCOMMUNITY

[17] BGP PrependAS – FRRouting. (2020, November 17th) http://docs.frrouting.org/en/latest/bgp.html#clicmd-[no]setas-pathprependAS-PATH

[18] IPGeolocation.io – Bogon IP Addresses. (2020, November 17th) https://ipgeolocation.io/resources/bogon.html

[19] Cisco – Protecting Border Gateway Protocol for the Enterprise. (2020, November 17th) https://tools.cisco.com/security/center/resources/protecting_border_gateway_pro

[20] Cisco – Securing a Core Network. (2020, November 17th) https://meetings.ripe.net/ripe-49/presentations/ripe49-eof-security-tutorial.pdf

APPENDIX

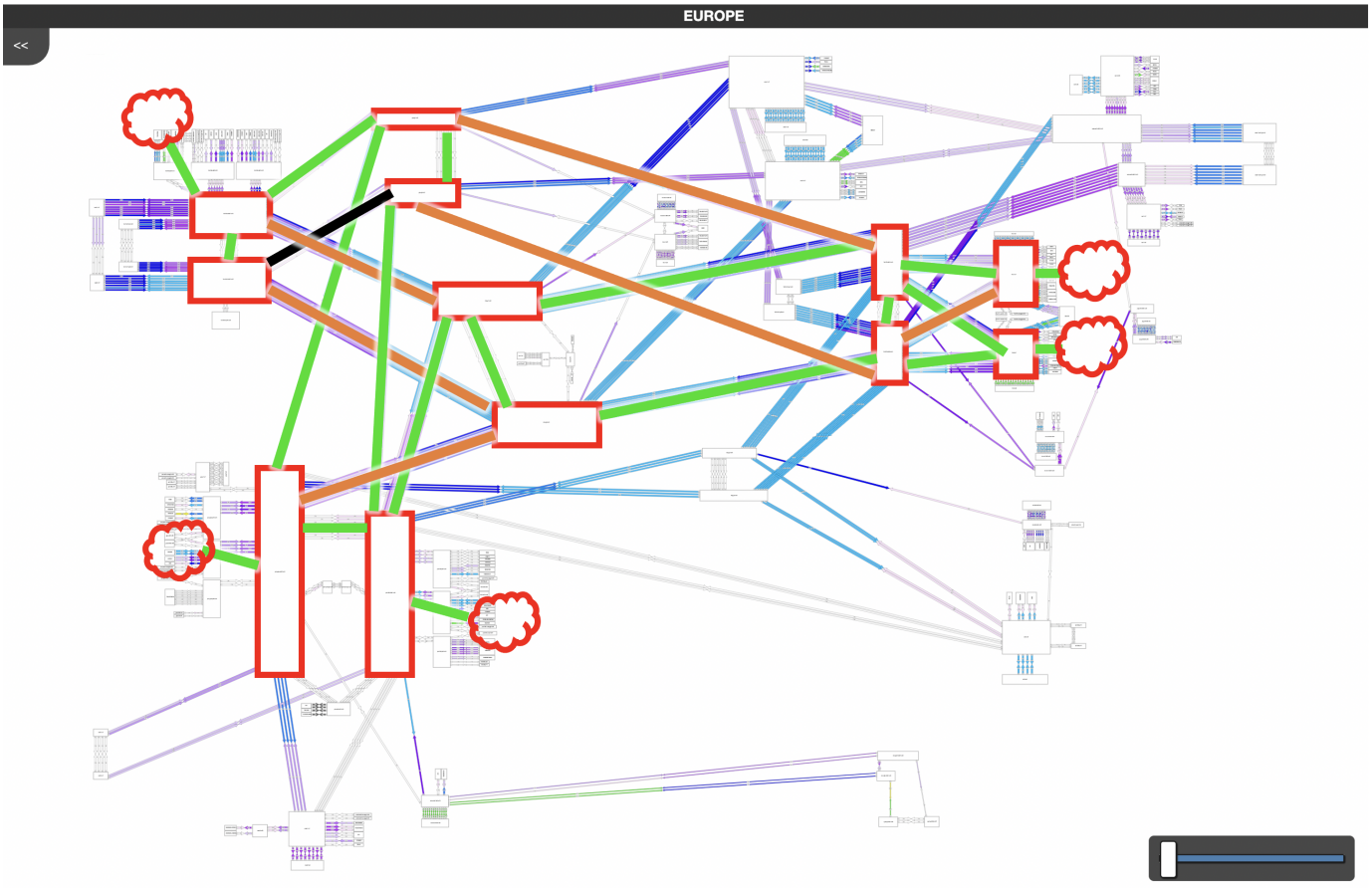


Figure 1. Schematic of our network in the european backbone of OVH

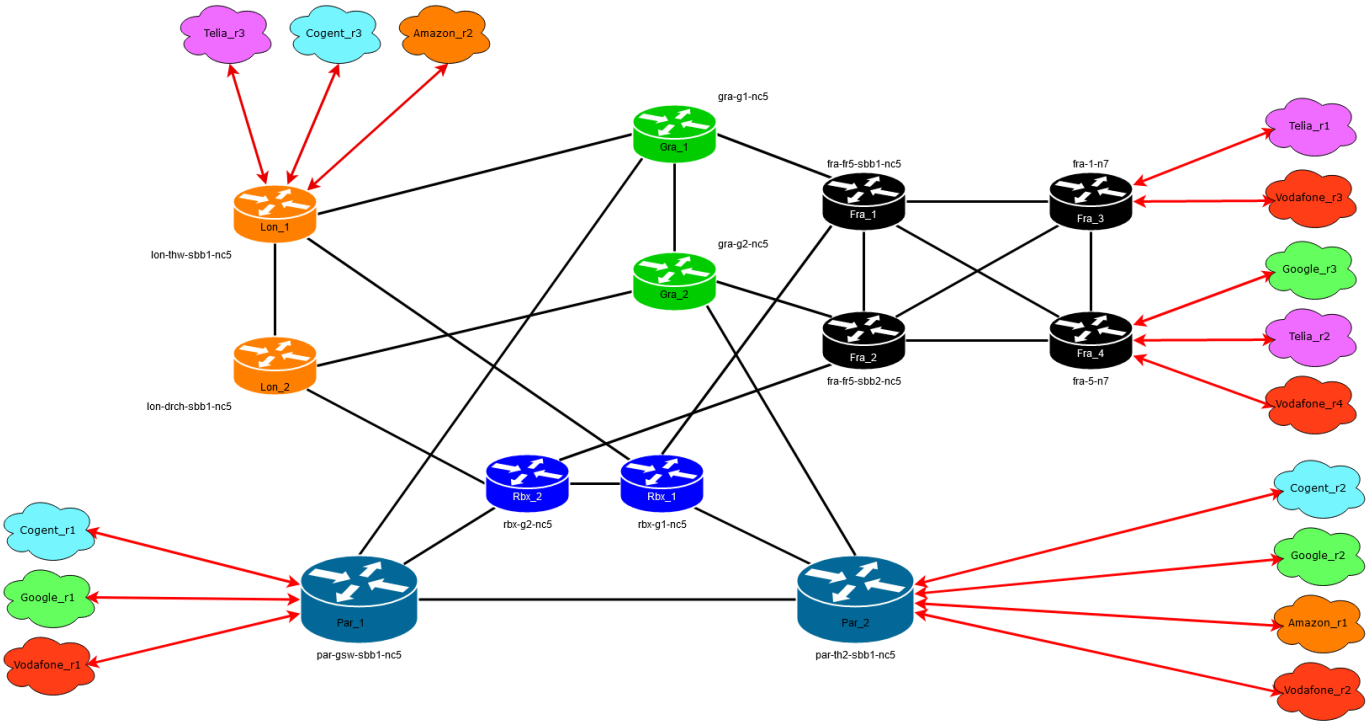


Figure 2. Schematic of our network

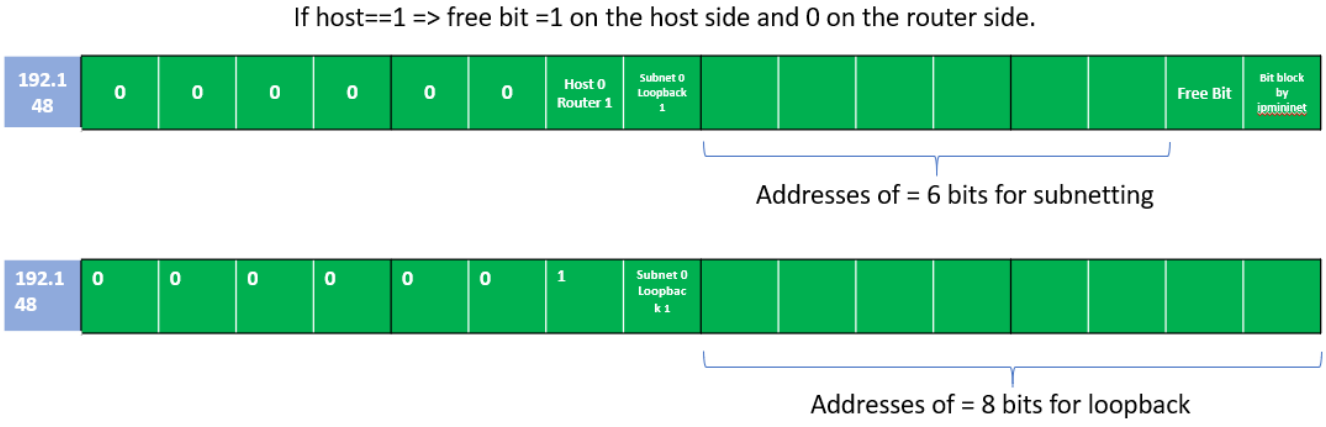
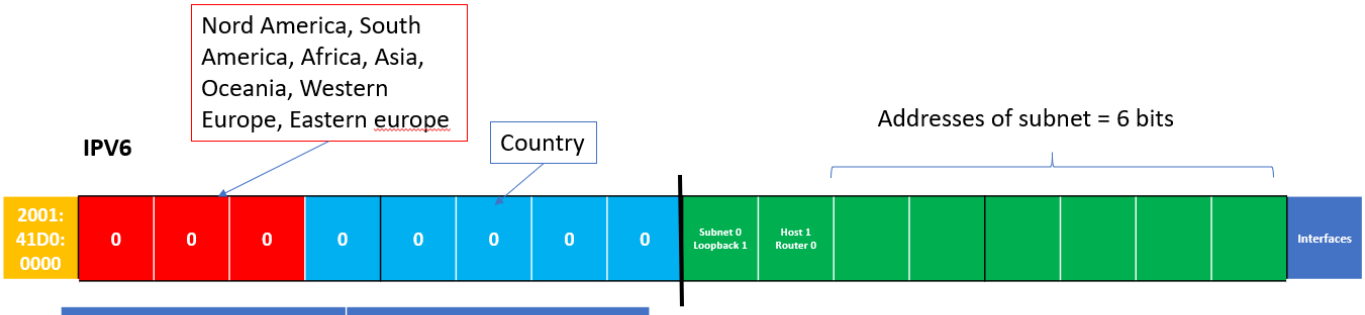


Figure 3. IPv4 addressing plan



Country code	Language
2001:41D0:0:0000::/51	Western Europe
2001:41D0:0:2000::/51	Eastern Europe
2001:41D0:0:4000::/51	Nord America
2001:41D0:0:6000::/51	South America
2001:41D0:0:8000::/51	Africa
2001:41D0:0:A000::/51	Asia
2001:41D0:0:D000::/51	Oceania
2001:41D0:0:F000::/51	Use to indicate that the subnet is inter-continental

Country code	Language
2001:41D0:0:0000::/56	FR
2001:41D0:0:0100::/56	DE
2001:41D0:0:0200::/56	EN
2001:41D0:0:1F00::/56	Inter-European connections

Figure 4. IPv6 addressing plan