LINGI2142 - *OVH* network analysis

Aurélien Buchet · 28601700 Merlin Camberlin · 09441700 Thomas Weiser · 38371600

***Abstract*— Network service providers are companies that operate or sell infrastructures and services. *OVH* is one of them.**

This report specifies the different network configurations that the *OVH* provider may require to satisfy its needs. With regard to the BGP configuration, two route reflectors of different levels have been deployed and each BGP client is connected to two different route reflectors to ensure redundancy. With regard to OSPF, metrics have been implemented to model distance and delays between routers. With regard to the allocation of IP addresses, two different strategies have been implemented depending on IPv4 or IPv6. With regard to traffic engineering, BGP communities have been added to avoid specific routes to be distributed all around the world. With regard to securities, passwords have been added to avoid a malicious user to break OSPF or BGP configuration. Finally with regard to anycast, three anycast servers have been distributed in different continents.

I. Introduction

Since 1999, the *OVH* company has never stopped growing and has became an important global cloud provider. Nowadays, *OVH* is spread in three continents : Europe, America and Asia. The company is responsible of the equivalent of 20 Tbit/s of traffic [1]. Compared to other cloud providers, *OVH* is very interesting because it provides lots of information about their network: its load and topology are publicly released and are available on `http://weathermap.ovh.net/`. However, *OVH* do not release the configuration of their routers.

The aim of this report is to develop the best configuration that would meet *OVH*'s needs, including the configuration of `OSPF` and `BGP`, including the allocation of `IP` addresses, securities, traffic engineering and the implementation of anycast.

To answer these questions, the report is divided in several parts. First, the network studied will be presented. Since the *OVH* network is too large to be fully modelled using *IPMininet*, only a subset of the *OVH* network will be covered. The second part of the report describes how the `OSPF` protocol is used. In the third one, it's how the `IP` addresses are allocated that is explained. In the fourth one, the `BGP` configuration will be described. Fifthly, a special attention will be taken to set up ways to protect `BGP` and `OSPF` against attacks. After that, the different ways that can be used to do traffic engineering are going to be explained and finally, how anycast can be implemented.

II. Network modelled

The main focus of the *OVH* network study considered is centred around North America. To determine which routers would be part of the analysis, the amount of traffic were taken into account. Moreover, to have a global view of the *OVH* network, somes routers outside America have also been selected. According to these criteria, these are the routers selected:

Singapore (SIN)	Sydney (SYD)
Palo Alto (PAO)	San José (SJO)
Los-Angeles (LAx1)	Chicago (CHI1)
Chicago (CHI5)	Beauharnois (BHS1)
Beauharnois (BHS2)	Ashburn (ASH1)
Ashburn (ASH5)	Newark (NWK1)
Newark (NWK5)	New-York City (NYC)
London (LON-THW)	London (LON-DRCH)

On the figure 1[1], red boxes have been drawn to highlight the selected routers for the analysis. As shown, they are spread all around America and only a few one have been ignored.

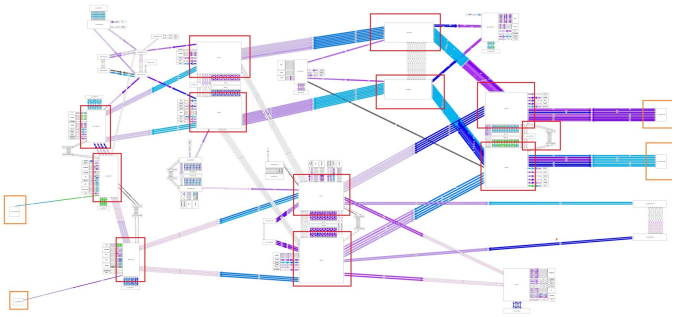

Fig. 1: Representation of selected router

The figure 2 presents a more schematic view of the different routers considered in the study. As illustrated, to model the different interactions of the American part of the *OVH* network with the rest of the world, a full-meshed physical link is assumed between the routers of APAC and Europe.

[1] A more visible view can be found on `http://weathermap.ovh.net/#usa`

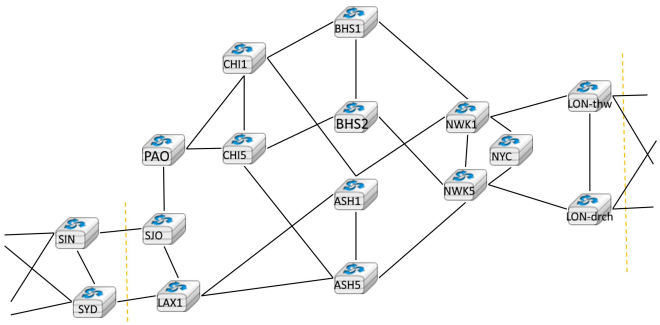

Fig. 2: Schematic view of the *OVH* network part studied

III. OSPF CONFIGURATION

`OSPF` and `OSPFv3` are both enabled on every router belonging to the *OVH* network. The `LSA` messages will be send with the default *IPMininet* parameters. Some complementary information is discussed in the `OSPF` security section.

A. Metrics

In order to compute the shortest path first (SPF) algorithm of Dijkstra, the links should have metrics to illustrate the distance between the routers. Four different metrics have been set up :

- small = 1
- medium = 3
- large = 8
- extra_large = 13

The `igp` metric of each link is set to one of these values depending on the real world distance between the two routers. The figure 3 below illustrates the different distribution of IGP metrics around the network.

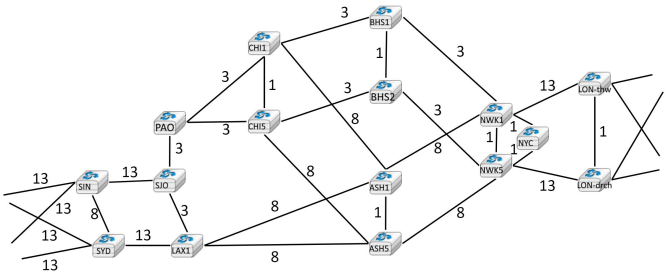

Fig. 3: Distribution of the different IGP weights around the network

IV. IP ADDRESSES PLAN

IP addresses in the network are distributed differently in `IPv4` and in `IPv6`. More details are given in their corresponding section. With regard to the links addresses, a local IP address is automatically allocated by *IPMininet* on each one (except the links between the anycast servers and the router attached). Between different start up of the network, different link addresses could be given to the same link. Nevertheless the network is not intended to be restarted, actually it should never been stopped. Knowing that the link addresses are allocated once by *IPMininet*.

A. IPv4 plan

Suppose the north America *OVH* network disposes of a prefix of length 24 bits. This means that they can host up to 2^8 different addresses. Due to the small number of addresses available the solution proposed is to manually allocate IP address to each routers inside the network. With the remaining addresses, a sub network has been allocated for each router inside the network. By doing so, every router can host up to 16 different `IPv4` addresses.

B. IPv6 plan

Suppose the north America *OVH* network disposes of a prefix of length 32 bits. This means that they can host up to 2^{96} different addresses. Due to this huge number of addresses available, it makes sense to use a hierarchical solution.

The idea is to allocate fewer and fewer addresses according to the hierarchy established bellow. Continents have a prefix of length 34.

America	2604:2dc0:0000::/34
APAC	2604:2dc0:4000::/34
EU	2604:2dc0:8000::/34
OTHER	2604:2dc0:2000::/34

V. BGP CONFIGURATION

This section will explore how `BGP` could work in the *OVH* network and configure `iBGP`, route reflectors and also the routing policies on the emulated `BGP` routers.

A. iBGP sessions

In order to configure `iBGP` on our network, we designed three layers of internal `BGP` sessions. At the top layer, three route-reflectors are connected in a full meshed. They are used to communicate the routes between different regions of the network so there is one on each continent : `ash1` for North America, `lon_thw` for Europe and `sin` for Asia. These routers have clients session with some lower level route-reflectors inside their region. These low level route reflectors have been chosen due to their central positions inside the network. Finally, some routers have client sessions with these route-reflectors. In order to choose the route-reflectors among the others, we looked at their position in the network and their number of links. As `ash1` had the most links, he was chosen to be the higher level route reflectors. Figure 4 and 5 show the different `iBGP` sessions inside the network at the top level and low level.

B. eBGP sessions

We linked one stub provider and three transit providers to the *OVH* network. We have chosen those peers thanks to the *OVH* looking glass website. They are listed on the figure 6. They are linked to multiple routers of our networks which gave them multiple ways to transfer information. To ease the reader, the figures 10, 11, 12 and 13 have been added in appendices to illustrate the physical link between *OVH* and its corresponding clients.

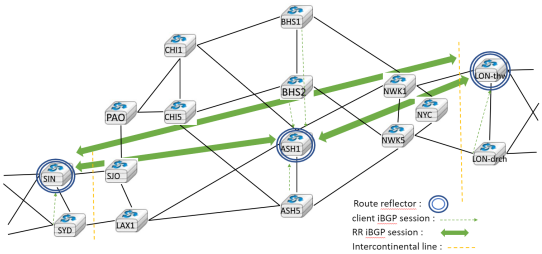

Fig. 4: `iBGP` sessions at the top level

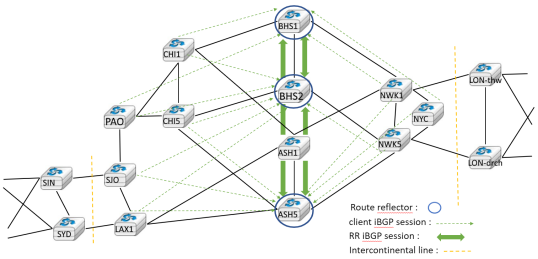

Fig. 5: `iBGP` session at the low level

NWK1	NWK5	ASH1	ASH5	CHI1
Cogent(1)	Cogent (2)	Cogent (3)	Cogent (4)	Cogent (5)
Level3(1)	Level3(2)	Google(1)	Google(2)	Level3(3)
Telia(1)	Telia(2)		Telia (3)	

CHI5	PAO	SJO	Lax	NYC
Level3(4)	Telia(5)	Cogent (6)		
Telia(4)		Level3(5)		

Fig. 6: eBGP peers and their link to *OVH*

- *Google* domain (named `ggl` in our configuration) is a stub provider. This means that *Google* will send and receive packets only for it's own connected hosts. *Google* is linked with a client `eBGP` session to *OVH*. The prefixes that *Google* advertises to these `eBGP` sessions will be advertise by *OVH* to all others *OVH*'s peers. *Google* is a client that pays *OVH* to diffuse its prefixes to the network.
- *Cogent*, *Level3* and *Telia* domains respectively named `cgt`, `lvl3` and `tel` in our configuration are three transit providers. It means that their routers forward packets whose source and destination do not belong to the transit domain. They are all linked with peer `eBGP` session to *OVH*. In this case, *OVH* will not advertise the prefixes learned from theses domains because *OVH* does not want to relay traffic of any other domain.

VI. OSPF AND BGP SECURITY

The routing commands presented in this section have been added to the ipminiet configuration that we submitted with our project. New parameters have been added to the OSPF and BGP daemons in order to specify the key and password to be used in the configuration files.

A. OSPF and OSPFv3

The first routing protection is the activation of the `Keyed-MD5` authentication. This method computes a hash value from the contents of the `OSPF` packet and a password. This hash value is transmitted in the packet. The hash value of the receiver should match with the hash value of the sender. This protection is effective against outsiders attacks as they are not supposed to know the key and the password. The routers of the networks know the same password to be able to authenticate the packets received form their neighbours. The field key ID allows the routers to reference multiple passwords. This makes password migration easier and more secure, this is not demonstrated in this project. These two commands are required to activate this feature : `ip ospf message-digest-key KEY md5 PSW ip ospf authentication message-digest`

The same commands need to be replicated on the neighbouring router as well. It is possible to set up multiple keys on a routers for every interface. To ease the deployment of the network, every *OVH* router use the same `KEY` and `PSW` for the authentication.

The second protection was already proposed by *IPMininet* and required no modification. It turns router interfaces which have no `OSPF` neighbor to passive mode. It prevents an attacker to answer false response to hello packets sent to an empty interface. An attacker could advertise false routes inside the network and compromise the installation. We prevent that to happen by setting these interfaces as passive.

For further improvements, many other things like `LSDB` checksums to look for inconsistencies in the packets or the activation of fight-back notifications when the `OSPF` fight-back mechanism enter in action could be deployed.

Note that the `OSPFv3` commands to setup the `Keyed-MD5` authentication are not documented in *FRRouting User Guide*, they do not seem to be working correctly.

B. BGP

The first securing feature is the `MD5` password to be used on the `TCP` socket which handle the connection with the remote peer. Like for `OSPF`, the same password is set for the entire *OVH* network with the following command :

```
neighbor PEER password PASSWORD
```

The second security concern prefix filtering. It allows a network administrator to permit or deny specific prefixes for each `BGP` neighbor. In our case, we will set a maximum number of incoming prefix that a `BGP` router will accept from a single `BGP` session. This precaution prevent a router to be flooded with route by a single malicious `BGP` connection. In our *OVH* topology, only a few number of prefixes will be advertise during a single `eBGP` session. We have set this number at 100. There is nearly no way that our little configuration overflows this number.

```
neighbor PEER maximum-prefix NUMBER
```

The third security protection aims at the `eBGP` sessions. It is a `TTL` security check. The received IP packet are compared

against a hop-count configured for each eBGP neighbor. If the TTL of the packet is greater or equal to the configured value, then it is accepted and processed. The hop-count is used to limit the number of hosts separating the two peers. The TTL value is calculated with the configured hop-count this way : TTL = 255 - hop-count. We have set the hot-count to be 2. The TTL of the received packets form the eBGP sessions will have to be at least 253. Attackers behind the eBGP connection will not be able to attack the *OVH* network through this connection.

```
neighbor PEER ttl-security hops COUNT
```

The fourth protection controls the as path of the packets received over adjacent ASes. For example, we want the *OVH* routers connected to *Google* to check that the packets received from this AS posses the *Google* AS number in their as-path. In this way, we prevent false or misconfigured packets to enter the *OVH* network. Such a protection would be activated with this command on the router.

```
bgp as-path access-list ggl permit ^2_
```

And this command on the route map to match the permit AS.

```
match as-path ggl
```

Some functions are missing on *IPMininet* to automatically add this as-path security to the configuration files of the routers. This feature is thus not enabled on our project but it is possible to add them manually with pexpect via telnet. We still have verify that packets coming from AS not accepted in the as-path are denied.

The last things to note are Access Control Lists (ACL). *Frrouting* explains that the only tool available to implement access-lists are import/export route-maps. With ACL, it is possible to filter traffic on an interface, to filter routing updates,... ACL do stateless inspection, which means that the access list looks at a packet and has no knowledge of what has come before it. It can not understand if a received packet is part of a conversation or not.

C. Firewalls with iptables

Iptables are used to setup firewalls to control and examine the traffic passing through the network. It does stateful inspection which means that it is possible to know if a packet belongs to a really existing conversation or not. The IPv4 and IPv6 firewalls have been configured in the same way : First drop all the input traffic by default and then we authorise traffic in case-by-case basis.

- The default rule for non matched input will be DROP
- ACCEPT internal traffic on the loopback interfaces.
- ACCEPT already established connections.
- DROP non conforming packets like those with invalid headers (also applied on the Output and Forward filter).
- ACCEPT useful IMCP packet types (RFC 792 states that all hosts MUST respond to ICMP ECHO requests)
- ACCEPT addresses of the packets coming from the IP subnets of the stub and transit providers.
- ACCEPT packets coming from the trusted routers of the *OVH* network.

It is possible to implement sophisticated against SSH brute-force attacks and ping flooding by setting a limits to new connections or ping that a single host can request [2].

These rules are applied to the iptables of both IPv4 and IPv6 protocols. It is important to notice that the IPv6 firewall accepts traffic coming from all prefixes. Once we impose the source of the IPv6 packets to belong to the prefix of the *OVH* network 2604:2dc0::/32, all the IPv6 traffic of the network is blocked. Every loopback address of the *OVH* network possess an address belonging to this prefixes. We do not face this problem with the IPv4 firewall.

Here is a view of the configuration of the IPv4 firewall of the routers.

```
  -P INPUT DROP
  -A INPUT -i lo -j ACCEPT
  -A INPUT -m conntrack --ctstate
RELATED,ESTABLISHED -j ACCEPT
  -A INPUT -m conntrack --ctstate INVALID
-j DROP
  -A INPUT -p icmp --icmp-type 0 -m
conntrack --ctstate NEW -j ACCEPT
  -A INPUT -p icmp --icmp-type 3 -m
conntrack --ctstate NEW -j ACCEPT
  -A INPUT -p icmp --icmp-type 8 -m
conntrack --ctstate NEW -j ACCEPT
  -A INPUT -p icmp --icmp-type 9 -m
conntrack --ctstate NEW -j ACCEPT
  -A INPUT -p icmp --icmp-type 10 -m
conntrack --ctstate NEW -j ACCEPT
  -A INPUT -p icmp --icmp-type 11 -m
conntrack --ctstate NEW -j ACCEPT
  -A INPUT -s 1.3.1.0/24 -j ACCEPT
  -A INPUT -s 1.4.2.0/24 -j ACCEPT
  -A INPUT -s 1.5.3.0/24 -j ACCEPT
  -A INPUT -s 1.6.4.0/24 -j ACCEPT
  -A INPUT -s 198.27.92.0/24 -j ACCEPT
  -P OUTPUT ACCEPT
  -A OUTPUT -m state --state INVALID -j
DROP
  -P FORWARD ACCEPT
  -A FORWARD -m state --state INVALID -j
DROP
```

Note that in a real network linked tho the internet, other rules can be considered. For example, get rid of bogons ips which have no legitimate use. As those unnecessary are not modelled in our project, these complications were not added.

VII. Traffic Engineering using BGP communities

The BGP communities attribute is an optional attribute that can be attached to routes. Community values can be used for many applications including traffic engineering which will be the focus of this section.

[2]See more details on the *GitHub* https://gist.github.com/jirutka/3742890

A. Principle

In order to control the traffic inside the network, some predefined community values are used to indicate that a route should be handled in a certain way. By making the values and their respective actions public, clients and peers can attach the attribute to modify the way their routes announced.

B. Implemented communities

In this section, we define the communities values that are supported by our network and their respective actions.

Informative communities	
16276:10	Route learned from NA
16276:30	Route learned from EU
16276:50	Route learned from APAC
16276:3	Route learned from a customer
16276:1	Route learned from a peer
16276:2	Route learned from upstream
Region filter communities	
16276:11	Route shouldn't be advertised outside NA
16276:31	Route shouldn't be advertised outside EU
16276:51	Route shouldn't be advertised outside APAC
Others	
16276:9	AS-path prepending should be use when exporting the route
16276:95	no-export community should be set when exporting the route
16276:10	Route should have a higher local-pref
16276:20	Route should have a lower local-pref

The informative communities are set by our network, they provide information to the peers and clients that receives routes from us. The region filter communities can be set on a route from one of our client or peer in order to limit the propagation of the prefix to one specific region of our network. Some others communities can be used by our neighbours to control the way their prefixes are announced, the community 16276:9 sets the `no-export` community when exporting the route which will limit its propagation after it had left our network. The 16276:95 indicates that our network should prepend itself to the AS-path, making it longer. It is also possible for neighbours ASes to modify the local preference of their routes using communities. Routes tagged with the community 16276:10 will have a higher local preference (225 instead of 200 for client and 125 instead of 100 for shared-cost). Routes tagged with the community 16276:20 will have a lower local preference (175 for customer and 75 for shared-cost). Using these values unsure that routes learned by customer always have a higher local preference than routes learned by shared-cost.

There also are several routes that are handled by default by *IPMininet* : the `no-export` community restricts the propagation of a route to a `BGP` confederation. The `no-advertise` community indicates that a route shouldn't be announced to any of its peer and the `local-AS` community indicates that a route must not be advertised to external `BGP` peers. The final community that can be used is the `blackhole` community described in [7]. We have chosen to honor by default the `blackhole` community on all the external `BGP` sessions, dropping all the traffic towards the tagged prefix.

C. Configuration

First of all, in order to be able to use the `BGP` community attribute on a *FRRouting* router, the command `neighbor PEER send-community` should be use. Otherwise, the attribute might not be sent towards the `BGP` peer (you can still receive it). It is also important to not overwrite the `BGP` community attribute when we only want to add one value. In order to do this, the appropriate command is: `set community additive VALUE`.

`BGP` can handle the actions to be taken for each community value using route maps. Each route maps is defined by a name, whether the route map should permit or deny routes, an order to indicate which routes are applied first (lower order first), some conditions to apply and actions to take. Traffic engineering can be made by setting the conditions and actions for the route-maps according to the rule defined for each community value.

There are multiple ways to call a route-map : the first one is to use the command `neighbor PEER route-map NAME [in|out]` that makes a call to the first route map with the name equal to NAME whenever a route is received or sent towards PEER depending on the value in or out. It is also possible to call a route map directly from another route-map using `call NAME`. The last way is the use `on-match next` inside a route-map to indicate that next route-map in terms of order with the same name should be call whenever the route-map is taken.

We created the rules for our network using the following method : each router has generals route-maps corresponding to each actions that can be taken and route-maps for each peering policies. The route-maps for policies check for community values that could be set on the route and call the corresponding general route-maps whenever it matches. Most of them use on-match next except the ones denying routes. They all end with a high order permit route-map that allows every route that hasn't been filtered or that didn't have any communities attached. When we establish a `BGP` session, we create two simple route-maps that call the route-maps for the import and export policies depending on the type of peering as well as the specific community for the peer and we use the neighbour command in order to use them when a route is received from or send to this peer. This allows us to easily add new peering with our routers because most of the route-maps are already defined.

Below is an example of route-maps configuration for a router of our network that have a peering with Google:

```
! Call to ggl route-maps
neighbor 192.168.46.1 route-map cust-ggl-in-ipv4 in
neighbor 192.168.46.1 route-map cust-ggl-out-ipv4 out

! Actions route-maps
route-map rm-set_no_export-ipv4 permit 8
set community additive no-export

route-map rm-blackhole-ipv4 permit 8
set community additive no-export
```

```
set community additive no-advertise

route-map rm-AS_prepend-ipv4 permit 8
set as-path prepend 16276

   ! Customer import policy
route-map rm-cust-in-ipv4 permit 8
match community blackhole
call rm-blackhole-ipv4

route-map rm-cust-in-ipv4 permit 12
match community localPrefL
set local-preference 175

route-map rm-cust-in-ipv4 permit 16
match community localPrefH
set local-preference 225

route-map rm-cust-in-ipv4 permit 20
set local-preference 200

   ! Customer export policy
route-map rm-cust-out-ipv4 permit 8
match community set_no_exportG
call rm-set_no_export-ipv4
on-match next

route-map rm-cust-out-ipv4 permit 9
match community AS_prepend
call rm-AS_prepend-ipv4
on-match next

route-map rm-cust-out-ipv4 permit 12

   ! ggl import policy
route-map cust-ggl-in-ipv6 permit 10
match ipv6 address all
call rm-cust-in-ipv4
on-match next

route-map cust-ggl-in-ipv6 permit 20
match ipv6 address all
set community additive 16276:10
set community additive 16276:3

   ! ggl export policy
route-map cust-ggl-out-ipv6 permit 10
match ipv6 address all
call rm-cust-out-ipv4
on-match next

route-map cust-ggl-out-ipv6 permit 20
match ipv6 address all
```

Note : the generals and policies route-maps have `IPv4` in their name because ipmininet requires them to have an address family but they can be used for both `IPv6` and `IPv4`.

D. To go further

BGP communities can be used for other traffic engineering as well, many interesting one are mentioned in [12] such as modification of the local pref attribute or community to allow to filter one specific peers. Another good idea is to filter prefixes that are too specific in order to avoid to saturate the `BGP RIB`. However, *FRRouting* doesn't support filtering using length of the prefix.

VIII. Anycast configuration

Anycast is, like unicast, multicast, broadcast, a transmission mode. Implementing anycast in a network has several advantages:

- It increases redundancy,
- It reduces latency times,
- It allows server load balancing,
- It increases security.

The following section describes how anycast works, how it is implemented and what are the choices of implementation that lead to the current implementation.

A. Principle

In the anycast transmission mode, a set of devices are identified by a single anycast group address. The same IP address is assigned to several different servers, potentially distributed across the world.

Clients transmit information to an anycast group address and not to the specific address of a server. It does not matter for clients which server responds as long as one provides the service or application desired.

The network ensures that the information will be delivered to one receiver that belongs to this anycast group. It's the routing protocol `BGP` that spread the anycast addresses inside the network and that indirectly automatically route packets from the source to the closest server.

To illustrate this point, suppose the simple topology shown on figure 7. Two anycast servers announce the same anycast address `1.1.1.1`. If a user wants to contact an anycast server, he contacts the first hop router that does a route lookup in its forwarding table. As two different routes are available to reach the same anycast address, the best one is selected and packets are routed to server A. [3]

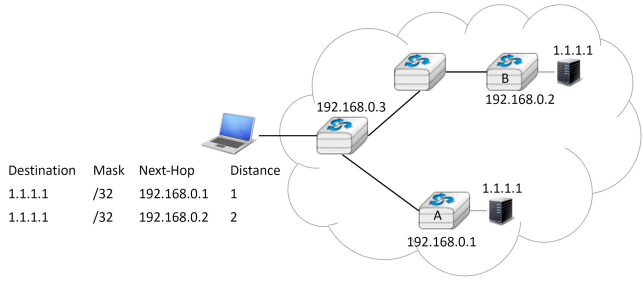

Fig. 7: Illustration of how anycast work in a simple network

Suppose now a subtle variation of the previous topology as illustrated on the figure 8. As there are two routes with the same distance to reach an anycast server, two situations can occur depending on what does the client's router.

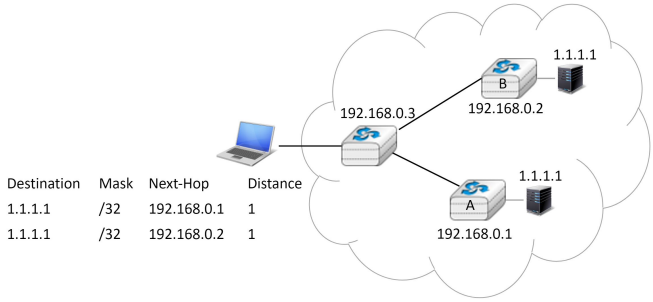

Fig. 8: Simple topology where two routes are available to reach an anycast server

If the client's router does per-packet load balancing then client's packets are sent to both servers A and B in a round-robin fashion.

- With the `transport control protocol`, this can make the anycast service unusable as the client's `TCP SYN` packet might go to server A, but his `HTTP GET` request might go to server B. Since server B doesn't have an active `TCP` connection with the client, it will send a `TCP RST` packet and client's computer will drop the connection and will have to restart a new one. [4]
- With the `user datagram protocol` (`UDP`), this problem is avoided since `UDP` does not need to keep a state of the connection.

If the client's router does per-flow load balancing then client's packets are always sent over the same path.

- Again, with the `user datagram protocol` (`UDP`), no problem occurs as `UDP` does not need to keep a state of the connection.
- With the `transport control protocol`, the previous problem with per-packet load balancing is avoided (unless a link goes down between the client and the server).

These problems would be taken into account while implementing anycast in the network studied.

B. In north America network

The following section will describe the choices of implementation made that lead to the final implementation of anycast in the network.

1) How to determine the number of servers to deploy ?

The choice of the number of anycast servers to deploy has been determined as follows. A trade-off has to be made between: installing enough servers to benefit from the advantages of anycast (resiliency, robustness, ...) but not too many to reduces issues while using `TCP` and per-packet load balancing (described in the previous section VIII-A). The number of three seems to minimise this problem but is still sufficient to benefit from the advantages by anycast.

2) How to distribute servers around the world ?

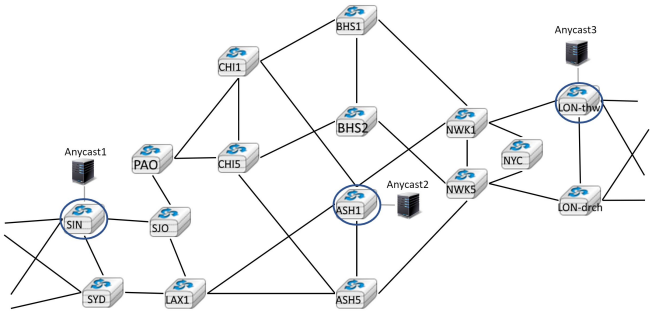

Fig. 9: Worldwide distribution of the different anycast servers

As illustrated on the figure 9, three anycast servers have been distributed across the world. Each of them covers a certain geographical region and should be used by client in this specific region.

- Singapore server is responsible for client in Asia Pacific,
- London server is responsible for clients in Europe,
- Ashburn server is responsible for clients in America.

Every server are identified by the anycast address `192.27.92.255` for `IPv4` and `2604:2dc1::0` for `IPv6`.

By splitting servers across the world, it permits to efficiently distribute the traffic into the different servers and to provide load balancing. Clients in America should be redirected to the American server in Ashburn, as well as clients in Europe to London server and clients in Asia Pacific to Singapore server.

Moreover it brings servers closer to clients and by the way reduces latency. In fact,fewer routers are going to be needed to reach one anycast server.

3) How to configure servers in the network ?

Since anycast servers and routers are not intended to be moved and are physically connected together, a first approach is to configure one static route. On the one hand, the forwarding table of the router have to contain an entry that point to the anycast server attached. On the other hand, the anycast server should by default respond via the router attached. However, this first approach can render the service unusable for numerous clients in case of a server failure. Consider a server anycast A responsible of delivering a service in a geographical region G. The server A is physically connected to a router R and there is a static route is configure between them. If server A fails, the router R cannot detected it and will continue to forward packets to this failed server resulting in rending the service unusable for all clients in the region G.

A second approach is to configure an `OSPF` daemon on the anycast servers since all servers are part of *OVH*.This simple solution works well and it relatively easy to implement in a network. Nevertheless this solution is generally not chosen by network engineers. The reason is the following: if for

any reasons the OSPF daemon bugs and continuously sends *link state packets* inside the networks, as in OSPF there is no way to limit the number of packets sent or received, this may cause major issues in the network.

A third approach is to configure on every anycast server a BGP daemon that announces the anycast IP address and a static route should be configure on the anycast server towards the router attached. To achieve this, the daemon Static should do the work as it handles the installation and deletion of static routes [2]. Unfortunately, for unknown reasons, while a static daemon was added on each connected router, no static entry in the forwarding table were created. After inspection, it turned out that no static daemon were running on the routers where they were supposed to. A simple htop or top command in the terminal of the router where static where configured can illustrate this point.

Finally the approach implemented is the following. On every anycast server runs a BGP daemon that announces the anycast address. To add a default route on the anycast server towards the router attached, the following commands are used :

```
route add default gw X
route add -A inet6 default gw Y
```

- Where X is the IPv4 interface address of the router on which router and server are connected,
- Where Y is the IPv6 interface address of the router on which router and server are connected.

The function setFRRoutingCommands(net) in the file configuration.py is responsible to automate this process.

Moreover to allow the anycast server to announce default routes, the command below has to be added on anycast servers. This is done in the bgpd.mako file.

```
neighbor PEER default-originate
```

A specific note to mention is that anycast servers are clients of route reflectors of higher level. Nevertheless to avoid the anycast server to receive routes from the route reflector, a specific method has been implemented. Its name is bgp_anycast() and its implementation is available in the bgp.py file. By calling this method, it specifies to the route reflector to not export its routes to the anycast server.

4) How to deal with failures ?

a) server failures: If a server goes down, no change are needed for the client to contact an anycast server. The packet should automatically be rerouted to the next closest anycast server available. In practice, as the server is down, it stops announcing the anycast address and the failure is detected.

b) link failure: If a link goes down, the OSPF protocols should detects the failure quickly and send a Withdraw message. By doing so, the first next hop router recomputes its routing table resulting in another path selected to reach an anycast server. It could happen that a different server is contacted.

c) service failure: To detect a service failure, running a script that regularly makes a request to the service running on anycast servers could do the work. If an error is returned while attempting to request the service then the service is considered as down and the script should gracefully shut down the iBGP session.

IX. Conclusion

There exists many ways to protect a network from outsiders and some insiders attacks depending on the network owner ambitions. The more sophisticated the network will be, the more breaches are left open for intruders. Having a simple configuration limit these risks.

The BGP community attribute is a powerful tool to help networks that want to have more control on the way transit providers advertise their routes and it can be configured to make it really to use and add new features.

Implementing anycast offers many advantages provided that a specific attention is taken to correctly configure and distribute the anycast servers.

This project made the authors realise the complexity to manage a whole computers network in an efficient way. At the end of this report, the important network configurations detailed should meet *OVH*'s needs.

References

[1] Wikipedia, available on: https://fr.wikipedia.org/wiki/OVHcloud (18-11-20)

[2] FRRouting User Guide, Kunihiro Ishiguro available on: http://docs.frrouting.org/en/latest/static.html?highlight=staticD#static (02-11-20)

[3] R. Louwersheimer, *Implementing anycast in IPv4 networks*, International network services, June 2004.

[4] Ritesh Maheshwari. *TCP over IP Anycast - Pipe dream or Reality?*, 5 septembre 2015 available on: https://engineering.linkedin.com/network-performance/tcp-over-ip-anycast-pipe-dream-or-reality (05-11-20)

[5] B. Quoitin, S. Uhlig, C. Pelsser, L. Swinnen and O. Bonaventure. *Interdomain traffic engineering with BGP*, may 2003 Available on: https://inl.info.ucl.ac.be/system/files/commag-may2003.pdf (19-11-20)

[6] Benoit Donnet, Olivier Bonaventure *On BGP Communities*, april 2008 Available on: http://www.sigcomm.org/sites/default/files/ccr/papers/2008/April/1355734-1355743.pdf (19-11-20)

[7] T. King, C. Dietzel, J. Snijders, G. Doering, G. Hankins. *BLACKHOLE Community*, October 2016 Available on: https://tools.ietf.org/html/rfc7999.html (19-11-20)

[8] Library of National Security Agency. *A Guide to Border Gateway Protocol (BGP) Best Practices*, 17 September 2018 Available on: https://apps.nsa.gov/iaarchive/library/reports/a-guide-to-border-gateway-protocol-bgp-best-practices.cfm (19-11-20)

[9] E. Jones, O. Le Moigne of Alcatel *OSPF Security Vulnerabilities Analysis*, June 2006 Available on: https://tools.ietf.org/html/draft-ietf-rpsec-ospf-vuln-02 (19-11-20)

[10] Jakub Jirutka on github. *Iptables rules templates*, June 2012 Available on: https://gist.github.com/jirutka/3742890 (19-11-20)

[11] E. Davies, J. Mohacsi. *Recommendations for Filtering ICMPv6 Messages in Firewalls*, May 2007 Available on: https://tools.ietf.org/html/rfc4890 (19-11-20)

[12] *Routing - NTT-GIN*, Available on : https://www.gin.ntt.net/support-center/policies-procedures/routing/

X. Appendices

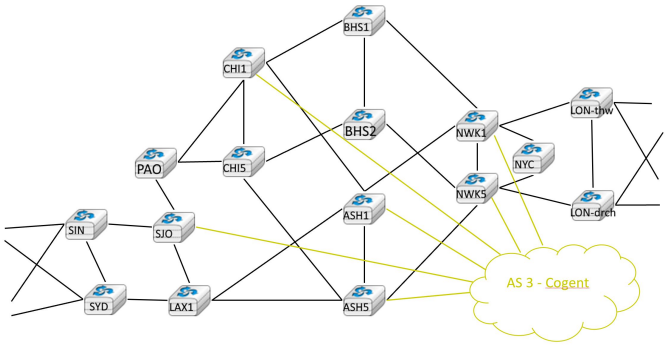

Fig. 10: Physical connections of *Cogent* with the OVH network

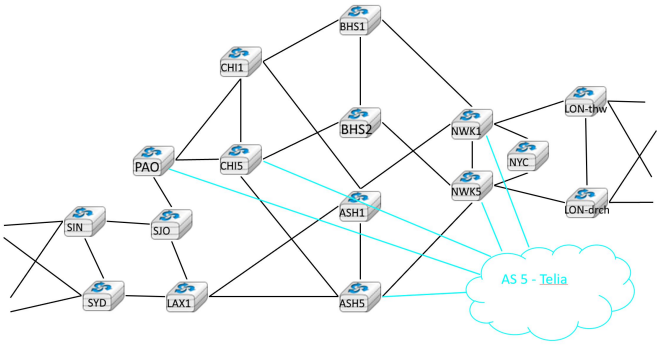

Fig. 13: Physical connections of *Telia* with the OVH network

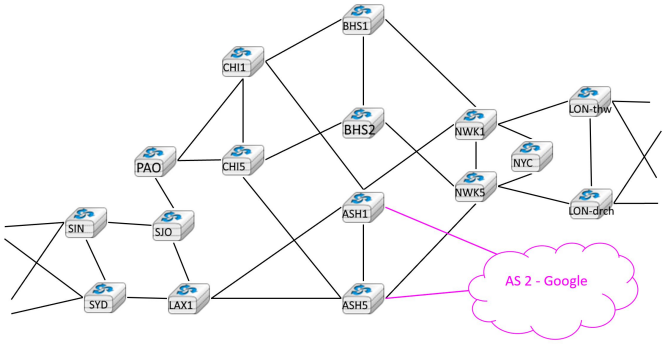

Fig. 11: Physical connections of *Google* with the OVH network

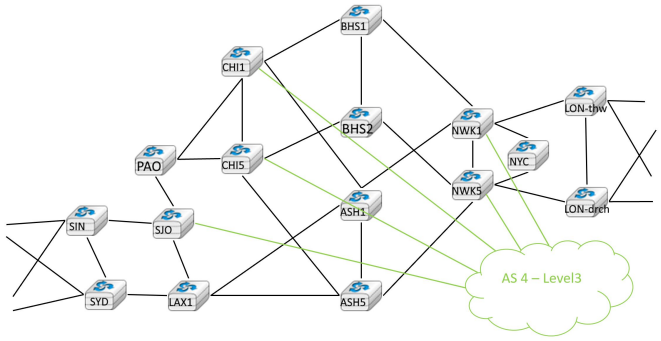

Fig. 12: Physical connections of *Level3* with the OVH network

CHANGES AFTER FIRST SUBMISSION

- Added support for local-pref high and local-pref low communities
- Automatise the process of adding default route on anycast router. This include giving address to link attached between the anycast router and the router attached.
- Details have been added about the securization of the network. A note has been added about the as-path check feature
- Add some comments and figure to clarify points in the report