Management of UCLouvain network

Group 3: Pierre LAURENS, Vincent LAURENS, Tanguy LECORRE, Jan RAHM

Abstract - This Document describes how we setup and configure a dual homing network based on the network of University Catholic of Louvain.

1.INTRODUCTION

During this project we have deployed a dual homed network based on the University Catholic of Louvain's real network. The objective was to reach internet from our network by using different prefixes. Then we should make our network work correctly in order to provide good services to the end users.

We divided our document into 5 sections. In a first part we are going to present our network topology and its addressing plan. Then in a Second part we will describe our routing configuration. In a third section we are going to develop our firewall policies. In a fourth one we are going to show up how we provide DNS and DHCP services and in the final part we will show up our monitoring strategy.

2.NETWORK'S TOPOLOGY AND ADDRESSING PLAN

2.1.Topology of the network

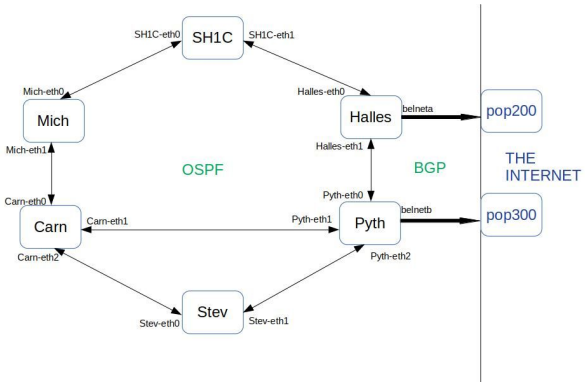

figure 1: Topology of network

The routers Halles and Pyth (figure 1) represent the gateways to access providers. Halles uses prefix fd00:200:3\48 and Pyth prefix fd00:300:3\48.

Each router is directly bonded to its neighbors. Notice that we create a redundancy link between Carn-Pyth which is the link between Halles-Pyth backup.

Furthermore we choose to use Halles and Pyth as DHCP and DNS servers, with Halles the primary server and Pyth the backup one.

Then we set firewall rules up on each router to protect our network against possible intrusions or attacks coming from the Internet. To finish we will use one of these two routers (Pyth) as monitoring server to test if network users can reach every services and the quality of service.

2.Addressing plan

In this part we are going to explain what addresses we will choice in each network site.

address IPv6 patron : "PREFIX:SITE::ID_ROUTER"

So as we said our network is dual homed which means we have two providers which offer two differents prefixes to machines of the network:

PREFIX[0-48]: Halles : fd00:200:3\48

Pyth : fd00:300:3\48

then we have site ID which is a number between 0 and 6.

SITE[48-62]

and finally the router_id which corresponds to the number of direct connected router and it is between 1 and 6.

We create three LANs, one on Pyth which handles monitoring and two host LANs on Carn.

Notice that each router interface has two IPv6 addresses, one for each prefix.

Thus we have an addressing plan below:

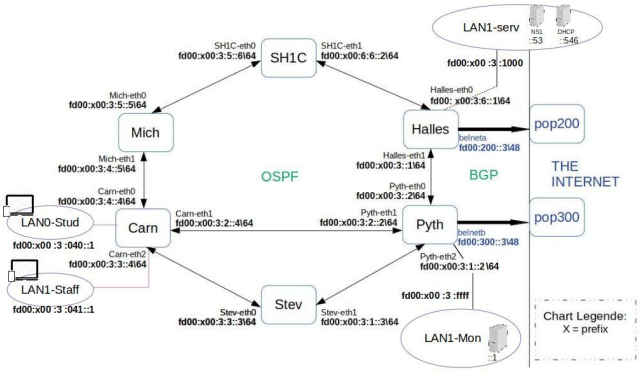

figure 2: Addressing plan

3.ROUTING(VINCENT)

In this part we are developing the most important part of the project because it represents the basis of our network. If a link is down a site can be isolated from the others and then users can't access Internet.

First of all we define the kernel protocol code of bird:

```
router_bird_file.write("protocol kernel {\n"
                       "        learn;\n"
                   "        scan time 20;\n"
               "        import all;\n"
               "        export all;\n"
                       "}\n\n"
```

In this function we define "learn" parameter to Kernel protocol because we want kernel protocol to be able to learn routes generated by others protocols like BGP, OSPF or static routes.

3.1.Routing inter-Domain

3.1.1.BGP Peering with Providers - ISP

Our first action is to build the link between our future network and Internet Service Provider.

```
router_bird_file.write("protocol static static_bgp {\n"
                       "        import all;\n\n"
               "        route fd00:200:3::/48 reject ;\n"
               "        route fd00:300:3::/48 reject ;\n"
                       "}\n\n"
                         )
```

These lines of code define a static protocol of bgp which rejects all the routes having prefixes because those routes are using only for internal network routing by OSPF

Then we define BGP protocols for routers Halles and Pyth:

```
for bgp, bgp_configs in configs["isp"].items():
        router_bird_file.write("protocol bgp
 provider"+bgp_configs["name_bgp"]+"{ \n"
            "    local as "+bgp_configs["asn"]+";\n"
                       "    neighbor
  "+bgp_configs["neighbor_address"]+" as
    "+bgp_configs["name_bgp"]+";\n"
                "    import where net = ::/0;\n"
          "    export where proto = \"static_bgp\";  \n"
                       "}\n\n"
                           )
```

We define the AS of provider that corresponds to provider number defining prefixes meaning 200 for "pop200" and 300 for "pop300".

3.2.Routing intra-Domain

We configure protocol OSPF in Intra-domain in every router to exchange table of routing between each neighbors router of the topology.

We export and import default route in order to permit OSPF propagate routes.

```
if net = ::/0 then accept;\n\n
```

We add this code line in export and import filter of OSPF. So routers can use OSPF to propagate this default route and by consequence each site can access provider.

3.3.Prefixes Load-Balancer

Our network is a dual home network with two different prefixes as we saw previously. We define a default route on each provider which routes all packets which are not coming from our network and have not our prefixes in their source address.

So we need to filter flow packets by only keeping ones coming from our network. That why we add rules to both providers.

We decided to define two routing tables: one to define routes where flow coming from our network transits and a second one which handles prefixes routing problems.

Below the two first rules accept only traffic passing through our network.

The third and last rules add a route in the load-balancer table

and define a preference which forces the traffic coming from our network but intended to the outside one to use this flagged router interface.

"ip -6 rule add from fd00:200:3::/48 to fd00:200:3::/48 pref 1000 table main",

"ip -6 rule add from fd00:200:3::/48 to fd00:300:3::/48 pref 1000 table main",

"ip -6 route add ::/0 via fd00:200:3:1::2 dev HALL-eth1 metric 1 table lb",

"ip -6 rule add from fd00:200:3::/48 pref 2000 table lb",

Now that we handle traffic we have to imagine if one of two provider falls down: how to redirect the outgoing flow?

To answer this question and solve the problem we should create a script which detect when a provider is down and redirect flows to the other provider by previous IP rules.

Finally we will write a script which detects if the primary link between Halles and Pyth is down and change the default route in lb table to switch flow routing to the second provider.

3.4.LAN & VLAN

We have set up a Local Area Network which host end-users on each router of the network even on Halles and Pythagore because even if they are used for services they can have students and teachers who need an Internet or Intranet access. To separate students and staff in our network we defined on our LAN, Virtual LANs which allow to segregate Staff flows from students ones. But we run out of time and we decided to create on Carn two LANs: 1 for students and a 2nd for staff to simulate VLAN.

Furthermore in Student and Staff LANs we have added static IPv6 addresses to interface because unfortunately our DHCP server is not running yet.

3.5.Settings and Monitoring

While we deploy OSPF protocol we met a routing problem. Indeed when we pinged a router for instance Pyth from Mich whereas the ping should pass through a gateway, the ping wasn't routed by the gateway.

The problem was that. *router_boot.sh* script hadn't enough permission to be run on router. Therefore forwarding feature wasn't activated on router.

We defined scripts which test if BGP link is up with the providers and that every site does have an Internet access.(Cf part 6 Monitoring)

This part has been entirely automated by Python script which set up BIRD configuration file in the routers /etc/ directory before to start the BIRD daemon. He had also addresses which both prefixes to every interface of routers and LAN interfaces. By this way, if we have to change a router configuration we don't have to do it manually in every router. We just need to modify a simple json file where configuration information will be stored. Then we run a shell script of starting up to delete network and recreate it with the new configuration.

4.Firewall(Pierre)

The security is an important part when we build a local network. The local network convey information across the network. For instance, an university is composed by teachers (staff), students and so on. If I am a teacher, I would not want my session is accessible to a student's session(using SSH, Telnet,..). When we work in the university network, we would like also surf on Internet. It can be useful.

To secure our end_users, we add rules to filter differents packets which transit by our network. These packets can be forwarded by routers.

In our project we configure multiple virtual router instances bounded together in a common environment. In this way, it is recommended to implement list of rules in order to maintain an overview of the access rules that refer to every virtual router instance.

It seems normal to add a security point on each router. We ask a question and try to answer it: What rules do we add? What is the policy we would like to follow?

To answer this question, we study two approaches:

- The first approach is a blacklist i.e we authorize everything and we add rules to block flow or to remove permission for some actions, some groups of people. In our opinion this approach presents some risks because administrators of the network can forget to block some flows. Consequently we can let weaknesses in our network which can be used by attackers..
- The second approach is a whitelist. Everything is blocked and we need to add rules to authorize actions on every router. This approach is more strict and demands on a strict organisation. We are 100 percent sure that to forget a rule will impact to the router and we protect each host more effectively on our local network.

We decide to implement the whitelist approach.

Indeed, we prefer this approach because we open only what we want to authorize.

To secure our network and create our whitelist on each router, we use the free software "IP6tables" which uses the Netfilter software which implements a firewall in the Linux Kernel.

In the first time we have listed each protocol we would like to authorize.(DNS, SNMP,ICMP(Ping), traceroute,...)

We began by cleaning the table of rules "filter". Afterward, we wrote two rules which allow to soften a bit our firewall and to return it usable. Now, no information has exchanged by routers because no protocols and hosts are authorized by our firewall. We can add rules allowing to authorize the following protocols (DNS,SNMP,ICMP(Ping), traceroute,...) in a certain configuration based on our topology described in 2.1.

ip6tables -F

ip6tables -X

The two lines above clean up the existing Firewall configuration on each router. In our case, the router doesn't contain rules but it's better to add our own rules to be sure that our configuration is aligned with the target.

ip6tables -P INPUT DROP

The line above it's a part of our policie. We wrote this type of commands for INPUT, FORWARD and OUTPUT. It means we prohibit all kind of flows by ports or protocols which are entering, crossing and leaving from each router in our local network.

This three lines implement, enforce our policy. As we said previously our policy is a whitelist. Therefore, we need to authorize every data flow transiting on each router. Furthermore, we need to unlock the essential protocols which are deployed on the network if we want the network to work correctly with an appropriate behaviour. Before deploying

rules for those essential protocols, we should accept flow which will use local router interface. With these rules our firewall will be more flexible and it will be easier for us to manage it.

-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

ip6tables -A INPUT -i lo -j ACCEPT

Now we are going to describe the command which authorizes or refuses some protocols. We are also going to apply rules on different groups of hosts belonging to the University Catholic of Louvain's sites (Students, Staff, Administration...).

For instance, we unlock the routing protocols OSPF or BGP on each router with the kind of following command:

ip6tables -A INPUT -p 89 -j ACCEPT

We accept the OSPF to go in (INPUT), to go through (FORWARDING) and to leave the router (OUTPUT).

We also use this kind of command to authorize the ICMP protocol which is a main protocol used to debug routing problems on network.

We would like to explain the end of the command especially on the part of the command after "ACCEPT". The echo request generated by ICMP protocol (number of messages "Hello" the ICMP protocol sends) is limited by and in this part of command. By this way, we eliminated the risk of flooding.

-A INPUT -p icmpv6 --icmpv6-type echo-request -j ACCEPT --match limit --limit 5/minute

The following command we present here is a command which needs ip address. For example, the command blocks all flows between the student group (every machines on the student LAN) and the teacher group (every machine on the student LAN).

ip6tables -A FORWARD -s lanstudent -d lan teacher -j DROP

The last following command also needs an ip address. It authorizes SNMP, SMTP, IMAP, POP protocols. These protocols are used by monitoring servers to check if the network is accurately operating. In addition they alert the administrator of the network about network's health.. We have authorized the ssh for the Monitoring LAN because it is a critical point of the network. For example, the monitoring LAN needs to be accessible by the administrator of the network. Indeed the administrator will have to add monitoring agent when the WAN will grow up.

ip6tables-A INPUT -p tcp -d ip adresse of Monitoring LAN -m tcp --dport 161 -j ACCEPT

ip6tables-A INPUT -p udp -d ip adresse of Monitoring LAN -m udp --dport 162 -j ACCEPT

This part has been also entirely automated by Python script which develops a backup configuration in the folder named "iptables/" before launching every configuration script of firewall on every router. We wrote a main Shell script which allows to execute our router iptables rules script.

In order to test the rules currently deployed on each router, we are based on the monitoring scripts (cf part 6) and for the moment we just tried to connect to (by tapping telnet command, ssh command,etc) each router and we analysed the result of different requests.

5.DNS /DHCP(TANGUY)

The DNS is configured on a LAN on Halles in our Network. It is built thanks to Bind9 daemon. Today, we have configured only one DNS server. Our goal is to add a second server on a LAN connected to Pythagore. Indeed, we want to be able to ensure that the second would take over in case of problems on one of our routers. The configuration is done in two stages. We must first declare to our server what will be the domain names it will have to manage, we call it zones. Then we will have to configure these zones, thanks to a zone configuration file. NS1 is responsible for the zone group3.ingi. .

Here is the declaration of the zone:

```
zone "group3.ingi" {
        type master;
        file "/etc/bind/db.group3.ingi";
   };
```

The type indicates that it is master on the zone, that is it can make the updates.

In the zone file, we have different types of records. We use those types:

SOA: Give the info of the zone.

AAAA: mean that our host have a ipv6 address.

NS: Let's define our DNS server domain.

PTR: matches an IP to a hostname (only used in the case of a reverse zone).

TTL is set for 1 hour, this means that the information that will be stored in cache will remain 1 hour.

```
$TTL 7200
@   IN SOA  @ group3.ingi.(
            2           ; serial
            7200        ; refresh
            900         ; retry
            1209600 ; expire
            3600 )  ; minimum
```

The Serial is a version number. It means that it must incremented with each modification. The Refresh is the time after which records are stored on the secondary server. Retry is the time that the secondary server will for a new test (if the master server is not accessible). Expire is the time that the secondary server will try to contact the master server. Minimum is the duration of the cache.

To check if our zone are working correctly we will use DIG, that provide a good diagnostic.

6.Monitoring(Jan)

For monitoring feature we have deployed “LAN-Mon” on router Pyth how can be seen from Addressing plan picture. There should be computer or some other device which provides monitoring services.

In our case this concrete LAN network has specific “SITE” prefix which is “ffff”. And the device where is running our monitoring has ip address “fd00:x00:3:ffff::/64” and with name “Mon”.

The whole configuration is located in folder “monitoring”.

For monitoring strategy we wish to monitoring everything but that is not possible because it would be unbearable burden for our network and also we do not have enough time to implement everything.

So we have decided to implement for the time being:

Reachability of providers - Every 8 second we are checking if the BGP link (with pop200 and pop300) is established. The status is saved in file “isp_status_log”.
The time of 8 seconds was selected using an experimental visual method. The same method was also used for all other "timer" settings.
If the BGP link is established we can see in BIRD remote control on Hall and Pyth.

Reachability of DNS - Availability of DNS service we are checking also every 8 seconds. Here we are only doing simply “ping” of DNS ip address. The status is saved in file “dns_status_log”.

Connection of routers - In this monitoring service we are testing if every interface of every router is accessible from “Mon” device. The test is only start once after initialization of our network and another additional 20 second (we need some time for establishing of OSPF).
Again it is simple “ping”. The status of this test is saved in file “conn_status_log”.

Bird logs - Every router is configured through BIRD daemon and it has own log file. This file we are periodically (every 10 second) saving on “Mon” device in folder “bird_logs”.
With this setting we can easily access log informations when some of the devices start to have some problems.

Every log file/folder is saved on the device in folder /etc/log/.

7.Conclusion

In this project we have worked in autonomy and we have defined, deployed, secured and monitored a WAN dual-homed network. We have learned BIRD technology and we have improved our network skills in another environment than CISCO technology.

We met problems we had to resolve. We have to keep moving to respect deadlines and objectives of the client.

This project simulated the work of a network administrator of a company. We could train for the job we will do in our future career. That was a great experiment to be lived.

References

https://debian-facile.org/atelier:chantier:dns-bind9-sur-wheezy

https://www.supinfo.com/articles/single/1709-mise-place-serveur-dns-avec-bind9