LINGI2142 : Computer networks

Maxime de Streel
65691400
maxime.destreel@student.uclouvain.be

Julian Roussieau
37571400
julian.roussieau@student.uclouvain.be

William Visée
67181500
william.visee@student.uclouvain.be

Jeremy Minet
08411500
jeremy.minet@student.uclouvain.be

January 13, 2019

Contents

1 Introduction **3**
- 1.1 Objectives . . . 3
- 1.2 Requirements . . . 3
  - 1.2.1 Generation scripts . . . 3

2 Addressing plan and network topology **3**
- 2.1 Addressing plan . . . 3
- 2.2 Network topology . . . 5

3 Routing - William Visée **5**
- 3.1 BGP . . . 5
- 3.2 OSPF . . . 5
- 3.3 Tunnel between HALL and PYTH . . . 6
- 3.4 Remarques de Mathieu . . . 7

4 End-user management - Julian Roussieau **7**
- 4.1 Dynamic address assignment . . . 7
  - 4.1.1 DHCPv6 . . . 7
  - 4.1.2 SLAAC . . . 7
  - 4.1.3 DHCPv6 + SLAAC . . . 7
- 4.2 Domain Name System . . . 8
  - 4.2.1 The SOA record . . . 8

5 Security - Maxime de Streel **8**
- 5.1 Main rules . . . 9
- 5.2 Border routers . . . 9
- 5.3 Users policies . . . 10
  - 5.3.1 Routers policy . . . 10
  - 5.3.2 Services policy . . . 10
  - 5.3.3 Teachers policy . . . 10
  - 5.3.4 Students policy . . . 10
  - 5.3.5 Guest policy . . . 11
  - 5.3.6 Administration policy . . . 11
- 5.4 Monitoring packets . . . 11
- 5.5 Tests . . . 11
- 5.6 Improvements . . . 11
- 5.7 Remarques de Mathieu . . . 11

6 Monitoring - Jeremy Minet **12**
- 6.1 Monitoring server . . . 12
- 6.2 Simple Network Management Protocol . . . 12
- 6.3 Report scripts . . . 13
- 6.4 Improvements . . . 14
- 6.5 Remarques de Mathieu . . . 14

7 Conclusion **14**

8 Erratum **14**
- 8.1 Addressing plan . . . 14
- 8.2 Firewall . . . 14
- 8.3 Monitoring . . . 15

1 Introduction

For the course of LINGI2142, we had to implement a network. To use consistent naming and design a system that fits reality, we had to base our network on the one of UCL (Université Catholique de Louvain-la-Neuve). In each part, we will describe the different implementation choices and the motivations behind those choices. To know how to run our network, please refer to our `README.md` file.

1.1 Objectives

We had to build a stable network that provides Internet to the whole city. It had to be able to host different services like telephones, cameras, printers, etc. To ensure those objectives, we had to go through policy routing, security, addressing plans, etc.

1.2 Requirements

The network should be designed to fit some requirements. First, security should be enforced to protect our network from different threats, coming from inside or outside our network. Then, we must be able to resist some failures like routers, servers or hosts going down. If one of those crashes, our network has to go on properly. Finally, our network must be able to support a lot of devices, meaning everybody should be able to connect to it.

1.2.1 Generation scripts

To generate the different scripts we made in the different routers, we have used a tool named `puppet`. Each router has a `node.yaml` file with information about itself. In the `site.pp` file of each router, we have included which classes we want to generate in this router and in the `hiera.yaml` file we have specified the configuration file, which is in the template file.

2 Addressing plan and network topology

This section will describe how we decided to organise our network and what addressing plan we chose.

2.1 Addressing plan

For the addressing plan we received two IPv6 prefixes of 48 bits : fd00:300::/48 and fd00:200::/48. The last 64 bits are reserved for the interface ID. We have 16 bits to organize. We used 4 bits for the sites (S), users (U) and locations (L) because their are already 6 sites, 8 uses and 6 locations (thus 3 bits for each category) and we add one bit for each category for further growth. This leaves us 4 bits which we use for the subnet (T).

Addressing plan fd00:X00:4:SULT::/64

- [48:52]: Sites (S):
  - Louvain-La-Neuve
  - Woluwe
  - Tournai
  - Saint-Louis
  - Saint-Gilles
  - Mons
- [53:56]: Uses (U):
  - Staff
  - Student
  - Guest
  - Accommodation
  - Servers
  - Voice IP
  - ICTV
  - Infrastructure
- [57:60]: Locations (L):
  - PYTH
  - HALL
  - STEV
  - CARN
  - MICH
  - SH1C
- [60:64]: Subnet (T)

We started with the sites because each of them may have a different addressing plan. Then we decided to put the uses, the location and finally the subnet in accordance to what was said in the document "IPv6 Address Planning" [1]. Starting with the uses first (for the Louvain-La-Neuve site) has a small impact on

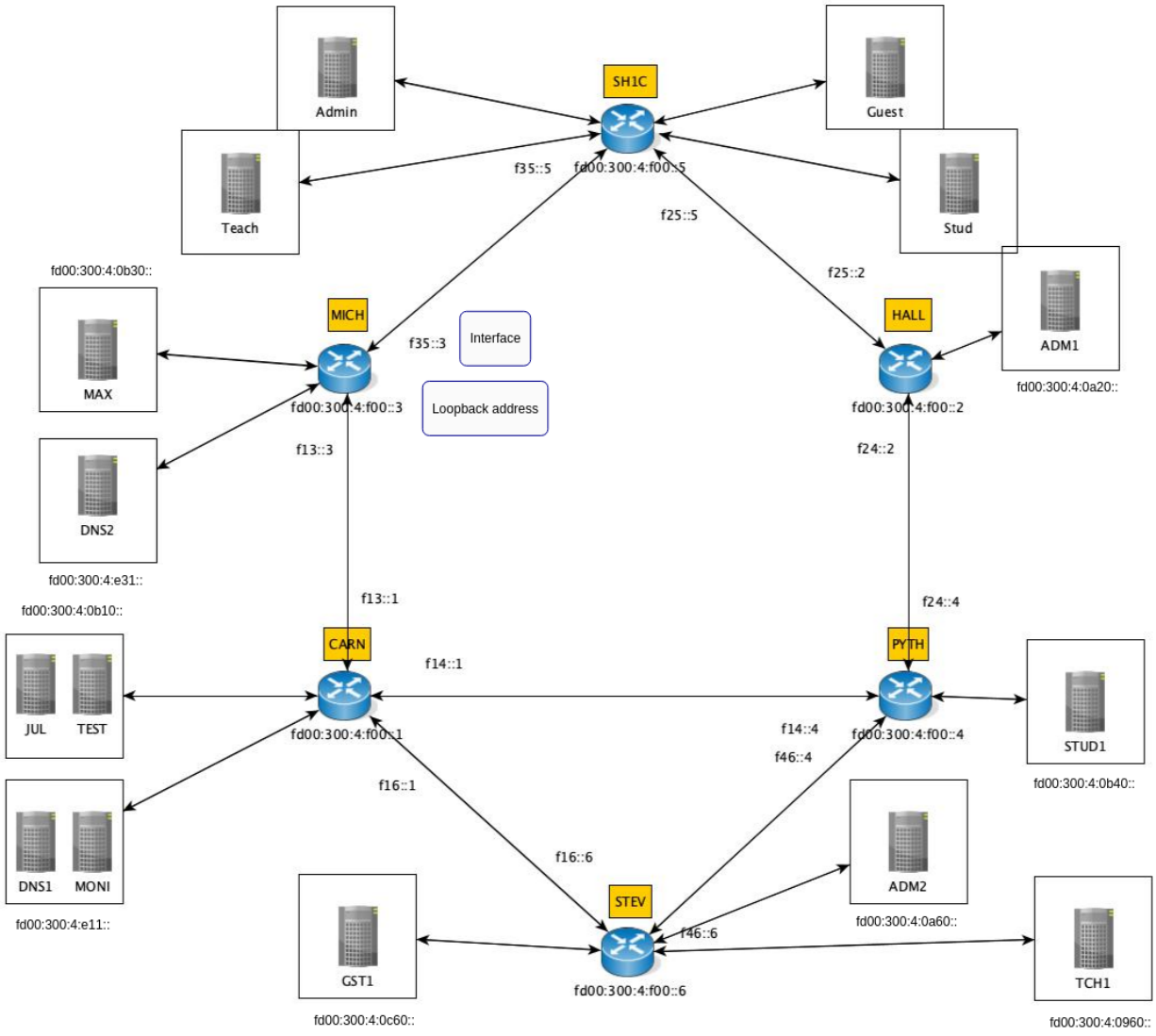

Figure 1: Network topology

the routing performances because nowadays, routers can support huge routing tables and will limite the number of rules in the firewall since the firewall mostly manages the use-types. This is why we chose a use-type first inside the Louvain-la-Neuve site.

We used a monotonic allocation for the sites, uses, locations and subnets because this is the simplest method.

2.2 Network topology

Our network topology is described on Figure 1. We can see on the topology that the servers/hosts that are on the same LAN are in the same group. We have DNS servers on the LAN attached to CARN and MICH and our monitoring server is also attached to CARN. The different hosts attached to the router STEV are in different LANs and have a static IP with a prefix that changes depending on their type. They allow us to easily test the firewall.

3 Routing - William Visée

Now, we will talk a little bit about how our network is working. We will look how we have configure the inter-network protocol which is OSPF and the protocol named BGP to communicate with the big internet. We use the daemon Bird6 to run OSPF and BGP over our network. We will also explain why it was necessary to use a tunnel between the router HALL and PYTH.

3.1 BGP

Import Export In our network belnet is the ISP. HALL is connected to belnetb and PYTH is connected to belneta. belnet advert some prefix but the routers only accept the prefix ::/0. HALL advert belnetb with the prefix fd00:200:4::/48 to let the internet know the existence of our network. PYTH advert belneta with the prefix fd00:300:4::/48.

Situation When HALL or PYTH receive a packet with a destination address other than one of the network address (fd00:200:4::/48 or fd00:300:4::/48), it will match with the ::/0 and the packet will be forwarded to belnet. We can imply the same reasoning when a computer outside our network send us packets. belnet will send it to HALL or PYTH because he knows our prefix.

3.2 OSPF

IP on each router As we have said in the topology section(2.2), each router has got several interfaces. Each public interfaces has got 3 IP which are mentionned in the topology. One with the fd00:200:4 prefix, one with the fd00:300:4 prefix and finally one with the local address. Each router has also got one loopback interface on which we have add an address of the following form : fd00:300:4:f00::<ID> with ID the number of the router. This address is the static address of the router. All of this addresses are assigned to their interface by the script static_router.sh which is generate by puppet in the directory /etc/static_router.

Import Export All the addresses of the routers mentioned in the section above and the ::/0 announced by BGP are exported via OSPF. This will let each router know the existence of every routers and let the routers to forward don't know packets to HALL or PYTH (::/0). Every routers of the network then knows where to forward packets inside the network. OSPF accepts every routes that he receives.

Values of parameters We specified only 4 parameters in the bird6 configfile name bird6.conf.erb.

1. The hello and dead parameter: We have set these values to 1 (hello) and 3 (dead) because we wanted something very responsive. 1 sec and 3 sec are really small values and so it overload a little bit the network but the network can adapt much faster after link failures.

2. The stub parameter: We have set this value to 1 because we only consider one stub area in our network.

3. the cost parameter : All the link of the network have a cost of 10 except the link HALL-PYTH that has got a link of 11. Indeed, with this cost, if STEV ping SH1C, the path that is going to be used is STEV-CARN-MICH-SH1C and inversely. This has got 2 advantages. First, the link HALL and PYTH will be less charged. HALL and PYTH is in average the most used link in the network and should then be avoid. Second, without this cost, the ping of STEV to SH1C could take a different return path than the go path and the firewall will not allow it.

There are also some other parameter that we decided to let to their default value.

1. wait : it's default value is 4*hello. It's the time that the router wait to start building adjacency. 4 seconds is a good time to let all the routers power on before starting the communication between each other.

2. retransmit : it's default value is 5. It's the time in seconds that the router wait before the retransmissions of unacknowledgement. We have this high value because the router is maybe overloaded.

3. poll : we don't have neighbors on NBMA networks so we do not touch to this parameter.

3.3 Tunnel between HALL and PYTH

Explanation of the problem When PYTH or HALL doesn't match with any IP addresses of their forwarding table, the router will match with the last input of the table that is the ::/0 prefix (import by the belnets).

By example : If PYTH receive a packet with the destination address fd00:200:4::/48, he will only match the prefix ::/0 and send it to belneta.

The problem is that we cannot send a packet from an address with the prefix fd00:200:4::/48 to belneta because we export only the prefix fd00:300:4::/48 to belneta with BGP. So when belneta will receive a packet from the prefix fd00:200:4::/48 he will not manage it correctly. The same problem exist with belnetb and the prefix fd00:300:4::/48.

Situation with no tunnel Imagine we are connected to CARN. We want to ping one of Google server. if we ping Google from one of our fd00:300:4::/48 address it will work because CARN will forward the packet to PYTH and PYTH will match the destination packet with the prefix ::/0 and will forward it to belneta. Seeing that the packet has got a source address of the prefix fd00:300:4::/48 belneta will forward it correctly toward the google server.

Now imagine the same situation but we ping the google's server with one of our fd00:200:4::/48 address. CARN will forward the packet toward PYTH and PYTH will forward it to belneta with the same reasonning that the paragraph above. belneta will not forward it correctly and the ping will not have any answer.

Set up of the tunnel : PRB rules To create a solution to our problem, we have first to create some policy routing base rules inside HALL and PYTH. First, if the source IP is from the network (So with the prefix fd00:200:4::/48 or fd00:300:4::/48) and if the destination IP is also from the network, we forward it base on the table main. This table is the table manage by OSPF. Now we create another rule after these one. For PYTH, if the source IP match with the prefix fd00:200:4::/48 we forward it base on the table 10. For HALL, if the source IP match with the prefix fd00:300:4::/48 we forward it base on the table 10. The table 10 contains only one route. it is default dev the_tunnel_interface. So every packet will be forwarded via the tunnel.

Set up of the tunnel : The tunnel The tunnel only encapsulate the packet inside an another packet with the destination of the endpoint of the tunnel. So with a tunnel from HALL to PYTH, HALL will encapsulate the packet inside a packet for PYTH. With the tunnel from PYTH to HALL, PYTH will encapsulate the packet inside a packet for HALL. When the router of the endpoint of the tunnel receive the packets, they unencapsulate them and forward them to the next destination. Seeing that the packets are to the extern network, they will forward them to their belnet.

Situation with a tunnel We can take back the same example than above. Now if CARN ping one of google server by example, the packet will be forward to PYTH. PYTH will look inside his rules. PYTH will then look inside the table 10. The packet will be forwarded to the tunnel. The tunnel will encapsulate the packet and send it to HALL. HALL will receive the packet and de-encapsulate it. HALL will then look inside his table main and forward it to belnetb.

Failure of HALL-PYTH link What if the link between HALL and PYTH fails ? Thanks to the tunnel because the packet will be redirect by an other path. Indeed OSPF will see that the link HALL-PYTH has failed and will inform all the routers about it. So when a packet will be encapsulate, it will have the destination address of the endpoint of the tunnel and the routers will forward it correctly toward it.

Test the tunnel There is a unit test in python inside the directory tests/main who simulate the crash of the link HALL-PYTH. To test it, execute the

following command inside the lingi2142 directory : make test routing

3.4 Remarques de Mathieu

Les points qui n'ont pas étés:

1. La partie BGP a été expliqué trop simplement. Il aurait voulu part exemple que l'on explique quelle valeur du keepalive mettre etc ... (Les paramètres entre autres)

2. Les tests du points de vue du code n'ont pas été pour un seul point. Les routeurs internes n'arrive pas a accèder à internet. Par exemple, il explique qu'on ne peut ping google.com que depuis les routeurs HALL et PYTH qui sont les routeurs externes. Il explique que ce bug résulterait d'un problème de liaison entre OSPF et BGP. Le problème serait le suivant, la route par défaut ::/0 de OSPF serait redirigé dans le réseau et non pas la route par défaut ::/0 de BGP. Je pense que la solution serait de modifier l'export filter d'OSPF du fichier bird6.conf.erb seulement pour les routeurs PYTH et HALL qui annonce visiblement pas la route BGP par défaut.

3. J'ai posé une question qui peut-être toujours intéressante de savoir. Pourquoi les paquets avec le préfix source fd00:200:4::/48 sont laisse tombé par l'ISP 300 (inversement avec le préfix 300 et l'ISP 200) ? Il explique que de manière général l'ISP forward que les paquets de ses clients et que si on veut qu'il forward aussi les paquets d'autre clients il faut payer car cela apporte des rules supplémentaires et du traffic en plus ... Il explique que les paquets sont sûrement "drop" avec une régle de firewall. On résout ce problème avec le tunneling.

4 End-user management - Julian Roussieau

In our network we have static ip for servers and we need dynamic ip for our hosts that disconnects and disconnects from the network. The different ip use the topology explained above.

4.1 Dynamic address assignment

In order to assign the addresses dynamically, we have several possibilities

- DHCPv6
- SLAAC
- DHCPv6 + SLAAC

We have documented ourselves in order to find which protocol to use. We will review the benefits of each.

4.1.1 DHCPv6

DHCPv6 uses a server that will distribute addresses within the network. It allows to have a good management of the network, which address we will use. It can ban mac addresses, track users and monitor what they do. One of its major disadvantages is that it is not supported by android. However, it is very easy to declare dns servers in the DHCP server.

4.1.2 SLAAC

SLAAC is a plug and play solution, just run the daemon on each router. The router will declare the prefix that hosts can use by routers advertisement. The host will create his own ip address with the prefix he has received, for example, by adding his mac address. With this solution, we do not need additional servers. It is also possible to be less trackable, changing its address whenever we want. Recently, it is possible to declare dns servers through the rdnss field, unfortunately all devices are not compatible.

4.1.3 DHCPv6 + SLAAC

The third solution is to combine the two. We use SLAAC to get an address to let the compatibility to android and we give the DNS servers through DHCP. In our solution, we decided to use only one protocol and to make the hypothesis that all our devices are compatible rdnss. One of the main arguments is that we do not want to track users.

Here are the different options chosen :

- AdvSendAdvert : Allows sending routers advertisement.
- MinRtrAdvInterval : The minimum time allowed between sending unsolicited multicast router advertisements from the interface, in seconds.
- MaxRtrAdvInterval : The maximum time allowed between sending unsolicited multicast router advertisements from the interface, in seconds.

- RDNSS : the ip of the dns servers.
- AdvOnLink : Indicates that this prefix can be used for on-link determination.
- AdvAutonomous : Indicates that this prefix can be used for autonomous address configuration.
- AdvRouterAddr : Indicates that the address of interface is sent instead of network prefix.

4.2 Domain Name System

In a network like ours, it's easier to use a dns server instead of registering the ip's of the different routers by hand in the host file. We deployed 2 servers to obtain redundancy in case of failure. The first is in the data center connected to the CARN router and the second in the one connected to the MICH router. The first server is the master server, it is he who knows all the resolutions when starting the network. The second server will copy and update on the first on the first server, it is a slave server. In case of failure of the main server, it will still be possible to resolve dns requests by copying the second server. Both servers can resolve the group4.ingi and 4.0.0.0.0.3.0.0.0.d.f.ip6.arpa requests. and resolve external queries for example for google.com.

As explained above, there are several ways to give the coordinates of dns servers. We can write their addresses in the resolv.conf file, that's what we do for routers. For dynamic hosts, we use the rdnss feature present in the advertisment routers. On our version of linux, the protocol is not supported, the part of the package that receives the rdnss field when performing a tcpdump.

rdnss option (25), length 40 (5): lifetime 10s, addr: fd00:300:4:e11::2 addr:fd00:300:4:e31::2

Currently, we have decided to keep the DNS resolver and the DNS name server on the same machine because we do not need to put them on two different machines. If we add machines, we add costs and maintenance. If you want, you can separate the two services to improve security and to prevent external users from having access to all the domain names.

4.2.1 The SOA record

Our network is in a stable state (normally, we do not have to change it too much), so we defined long TTLs in order to minimize the costs and increase the speed thanks to the cache.

- Refresh : number of seconds after which secondary name servers should query the master for the SOA record, to detect zone changes.
- Retry : number of seconds after which secondary name servers should retry to request the serial number from the master if the master does not respond.
- Expire : number of seconds after which secondary name servers should stop answering request for this zone if the master does not respond. The sum of refresh and expires.

Here is our record SOA.

```
$TTL 604800
@ IN SOA ns1.group4.ingi. root.group4.ingi. (
      7 ; Serial
 604800 ; Refresh
  86400 ; Retry
2419200 ; Expire
 604800 ) ; Negative Cache TTL
;
```

5 Security - Maxime de Streel

In this section we will discuss about the different threats our network is exposed to and how we manage them. We handle this with `ip6tables`, a module of `Netfilter` which implements firewall within Linux Kernel. The firewall.sh script is generated with puppet, the configuration file is available in gr4_cfg/templates/firewall.conf.erb. The references contains some of the websites that were consulted to learn good security practices and to have a better understanding of the `ip6tables` tool: [4] [5] [6] [7].

Preamble `ip6tables` works with three default `chains` : `INPUT`, `OUTPUT` and `FORWARD`, corresponding to the packet incoming, outgoing and passing through the router. Concerning the use of `ip6tables`, each packet coming through the `firewall` will be tested through a set of rules in a specific order until it matches a rule or reaches the default policy. When the packet matches a rule (`ACCEPT`, `DROP` or `REJECT`) it quits the `firewall` with the specific action of the rule it matches.

5.1 Main rules

Each rule has its own `firewall` file that contains the set of rules deployed at the start of the network. Before going into details we will explain the general rules we have.

White-listing policy The white-listing policy is a policy that by default `DROP` all the packets and only `ACCEPT` a few rules that we explicitly specified. Thus if the packet doesn't match any of the rules we set up it will be `DROP`. We chose this policy since this is quite a safe policy. If we forgot a rule that could lead to a big security breach the whhite-listing policy will `DROP` it. We're thus better protected.

Existing connections This means we `ACCEPT` all `ESTABLISHED` or `RELATED` connections from anywhere to anywhere. This means that all existing connections are allowed. An existing connection is a connection that has already been authorized by the firewall in one direction. This solution has two drawbacks: the firewall has to keep some information in memory (state-full firewall) and the response packet should take the same path. To make sure the response takes the same path we increased the path cost between PYTH and HALL by one (11, default is 10). Indeed, when increasing the cost of 1 between HALL and PYTH will ensure symmetry in our network. Let me explain the use of symmetry, let say that an user on STEV is allowed to established a connection on a server located at SH1C and that the server is not allowed to establish a connection with this user (seems quite logic). When STEV sends a packet over port 22 in TCP to SH1C they are two best routes, one via PYTH and HALL and one via CARN and MICH. Let say that the ssh server on SH1C receives the packet via CARN-MICH and response with an ACK/RST via HALL-PYTH. Since the ACK/RST is not going back through the same route as the first ACK send by the user on STEV, the firewall will consider this as a new connection and will `REJECT` it because the server is not allowed to establish connections with users. If the ACK/RST packet is send via the same route (CARN-MICH) the firewall will consider this as an already established connection and will `ACCEPT` it. How to be sure that increasing the cost of one between HALL and PYTH unsure symmetry? The main idea here is to avoid having two possible routes between two routers. SH1C and STEV are the only two routers having two different routes to reach each other. Increasing the cost of one (thus 1/10 of the actual cost) will only differentiate two routes with the same cost. For routes with different costs, for example 10 and 20, adding 1 to one of those routes will not change the best route, but if we have two routes with the same cost it will differentiate them.

Multi-homing Because or network is `dual-homed` we duplicated all our rules to take into account the `dual-homing`.

`DROP` **and** `REJECT` The main difference between `DROP` and `REJECT` is that `DROP` will act like an open port with no service, the sender of the request will never know if the service for which he was asking is available or not. `REJECT` will explicitly says that the port is unreachable, the sender will know that the service is there and that he doesn't have access to it. We decide to use `REJECT` inside or AS for users who we trust in (thus every user except *guests*). For all connections with the outside we prefer to give as less as possible information about our network thus we chose to `DROP` the connections.

5.2 Border routers

As mentioned in the network topology section (2.2), our network has two border routers. They are the doors between the outside and our network, therefore they need extra rules to protect our network as good as possible. It must not only protect the network from incoming traffic but it has also to prevent some traffic to leave our network.

BGP In order to allow connections with the internet we must `ACCEPT` the `BGP` packets to our border routers. We `DROP` the BGP connections in the forwarding table since the border routers are the only routers that have a BGP session running.

OSPF In order for routers to be able to communicate among each other (in fact to know where which router is, but this is explained in the section 3) we must `ACCEPT` the `OSPF` packets inside our network. We block those packets that want to go outside our network (misbehaviour of bird), it would forward the topology of our network and that is not something we want.

RA To allow router advertisement (see section 4 for more information about RA) we had to `ACCEPT icmpv6` packets with type 134 and who had on error code of 0 (`--icmpv6-type 134/0`) and `icmpv6` packets with type 133 and error code 0 (`--icmpv6-type 133/0`). We accept those packets in `OUTPUT` and `FORWARD` because there is no need for a router to receive such a packet. For the same reasons as `OSPF` we block those packets who try to leave our network.

Spoofed IP addresses If a packet is coming from the outside (on the belnet interface) it is not supposed to have a source address containing the prefix of our network (`fd00:300:4::/52` or `fd00:200::/52`). We block those packets because it can only mean that those packets have been spoofed by an attacker.

Outside traffic Concerning the open ports to the outside, we allow the following services: SSH (tcp: 22), SMTP (tcp: 25), DNS, (tcp: 53, udp: 53), HTTP (tcp: 80) and HTTPS (tcp: 443). Those are the strict minimum ports to open for outside traffic. We assume that each host on or server has to be able to consult his mails, visit a secured or unsecured web server and to reach a ssh server. Furthermore, since ucl is part of Eduroam we have to have those ports open in outgoing [8].

5.3 Users policies

In this section we will discuss a bit more the policy deployed for each of our user types.

5.3.1 Routers policy

Limited access Except from administrators and servers nobody should have access to a router. This is why we drop all packets coming from students, teachers, services and guests towards a router.

Tunneling Since we use a tunnel to redirect the traffic from HALL to PYTH and otherwise to prevent packets going to wrong ISP we had to accept packets that encapsulate ipv6 in headers (protocol number 41).

Restricted ports Each user can send traffic to a certain number of ports, by default we drop all traffic towards ports except those open for the user in question. For more information about the ports allowed, please refer to the section of the user in question.

Neighbour discovery attack Since the standard subnet size of our network is a /64, there are many addresses that are not allocated. An attacker could use this knowledge to perform a scan towards a huge amount of addresses in this prefix. Our router will have to make a neighbour discovery for each address that is not known for this router. This can fill the memory of the router and cause a DoS (denial of service). To prevent this we limited the number of neighbour solicitations to 10/minute. The limitation of 10/minute has not really been calculated ... If we had done the things properly we would have calculate the mean bandwidth used in our network and the CPU usage of PYTH and HALL to estimate how much neighbour solicitations per minute we could handle before or network saturate. We put 10/minute because we estimated that a client has no reason to send us more than 10 neighbour solicitations per minute.

5.3.2 Services policy

allowed ports For the different services available on the campus like voice, video, ect we allowed communication through different ports like tcp,udp: 53 (allow DNS for our DNS server), tcp: 5001 (allow the iperf tool for testing), tcp,udp: 5060,5061 (allow isp for voice and video).

allow hosting We allow services to host other services since servers have to be able to host.

5.3.3 Teachers policy

Teachers are normal users, they don't have special permissions.

allowed ports Teachers are allowed to communicate over all the common ports, like ssh, IMAP, HTTP, HTTPS, ... It seams quite logic that teachers are allowed to read their emails, consult web pages, make ssh connections, ... We also allow them to use the iperf tool (tcp: 5001) for testing purposes.

host We consider that teachers can host services because they maybe need it for some courses.

5.3.4 Students policy

Like teachers student are normal users, they have the same open ports as teachers because those ports are very common ports. Unlike teachers, students can't host services. We consider that students are better

protected like this and we feel no need for students to be able to host services.

5.3.5 Guest policy

Open ports Since guests are in a matter of speaking external users we decided to restrict them to the strict minimum. The services that we allowed for guests are the following: SMPT, DNS, HTTP and HTTPS. For more explanation about why we allow those ports for quests please refer to the border routers section – outside traffic 5.2.

Not a host We consider that guests are not able to host services, those privileges are reserved for users from our network with special privileges like teachers and administrators.

5.3.6 Administration policy

Full access This kind of user has full access since they are administrators.

Monitoring servers Administrators are the only ones who can access our monitoring server, administrators will have to analyse the network to improve it.

5.4 Monitoring packets

The logs of `ip6tables` have been activated through `ulogd`. The logs of the packets that matches a line containing `-j NFLOG --nflog-prefix "<some␣ text>"` are gathered at /var/log/ulog/syslogemu.log. We decided to print each packet (input, output and forward) that is dropped by the default policy. Logging the packets could help an attacker to DoS the border routers. If the attacker sends a lot of requests that pass through all our firewall they would all been logged, logging a packet take a bit of the CPU so if they are a lot it may saturate the router. But there has to be a lot of those packets before it saturates our router. So it has a small impact, but nothing dramatics.

5.5 Tests

In order to test the firewall we made some tests (available in tests/main/). They test the different accesses of the different users to make sure they don't have more access than we gave them. We used nmap to send traffic to a specific port with a specific protocol. For more information about how to launch our tests please refer to the `README.md`.

5.6 Improvements

Of course there are some improvements possible. Even if the white-listing policy makes our firewall quite safe and that we protected our network against topology discovering their are still lots of ways to put a network down. The most common one is the DoS, we tried to prevent it by dropping the neighbour solicitations packets that are coming too often (more that 10/minute) but there are lots of ways to make a DoS attack. We could try to see from which prefix is coming a packet and if we receive to many packets from this prefix at the same time we could discard them. It would made our network a bit more secure but still vulnerable if the DoS is coming from different prefixes.

5.7 Remarques de Mathieu

De manière générale, très bon rapport et code. Les points qui n'ont pas étés:

1. Le premier problème est celui du established. Le principe du established: Le routeur va mettre dans sa table de "connection établie" toutes les connections dont il a vu le SYN et ACK du three-handshake (par exemple). Quand il verra un message sur cette connection il l'accepte. Ceci permet de mieux proteger un client. En gros une fois que t'as établi la connection avec le serveur de ton choix, tu peux etre sur que l'information qui vient de se serveur viendra uniquement par la connection que tu as établie (personne ne pourra se faire passer pour le serveur car ca sera vu comme une nouvelle connection). Donc c'est une très bonne idée d'utiliser cette fonctionnalité. Pour que ca fonctionne bien il faut s'assurer que le transfere d'information se passe TOUJOURS par le même chemin (vu que j'ai mit la règle sur chaque routeur). Pour ce faire j'avais fait en sorte qu'il n'y ai pas de symétrie dans notre réseau (pas deux routes entre deux routeurs avec le même poids). Avec ca j'étais sur que le chemin utiliser pour établir la connection était unique. Le seul truc c'est que je n'avais pas pensé au fait qu'un routeur pouvait crash et que donc la route par défaut entre deux routeurs pouvait changer. Une fois le chemin changé, le communication va passer par de nouveaux routeurs qui eux n'avait

pas la connection dans leur table. Cette connection risque donc fortement d'être DROP (car le client n'a probablement pas le droit d'héberger donc tout packet en direction du client est DROP vu que si c'est le client qui a établi la connection le packet aurait ete ACCEPT par la regle d'established) par la police (souvent whitelisting). Pour pallier à ce problème il aurait fallu de manière générale, accepter toutes les connections en forward sur tous les routeurs. Chaque routeur aurait du avoir pour ses lans avoir des règles d'established (on est plutot sur que le packet passera par le premier et dernier routeurs, sinon c'est que le client/serveur est down). Vu que chaque routeur laisse passer le traffic en forward (si c'est pas destiné a qq1 directement connecté chez lui) on a plus besoin d'équilibrer le réseau vu qu'on s'en fous de par ou ca passe. Evidemment les routeurs de bords ne peuvent pas tout forward (genre ospf ne doit pas sortir du réseau ect).

2. L'autre "soucis" était que j'utilisai le white listing, qui par définition DROP tous les packets qui n'ont pas matché une règle plus haut. En cours mathieu avait dit qu'on ne pouvait pas DROP avec le whitelisting policy mais c'est pas tout a fait vrai, le term "matcher" ne veut pas dire ACCEPT, ca veut juste dire que ca a trouver une règle (peut importe si ca accept, reject ou drop). Mais de manière générale il faut éviter de mettre des drop au milieu du firewall car il y a beaucoup plus de chance de faire des erreurs (blocker des connections qu'on voulait accepter, comme mathieu l'a bien démontré avec mon firewall...). C'est probablement moins performant car chaque packet dropper va traverser tt le firewall mais au moins t'es sur de ce que t'acceptes. Il faut donc uniquement mettre des ACCEPT car de toute manière si ca a pas match une règle ca sera drop a la fin. Ca fait également plus de règles a mettre mais a nouveau, c'est plus safe. Le firewall doit avant tout etre safe, ensuite performant.

6 Monitoring - Jeremy Minet

In this section we will discuss about the different tools and mechanisms that we deployed to monitor our network. The three main resources that we have used are `SNMP`, report scripts and log files. The Simple Network Management Protocol enables the possibility to keep an eye on the network, more especially on the bandwidth usage, the different kinds of errors or even the discarded packets. See section about SNMP (6.2) for more details. Next to this, we create some scripts dedicated to report if everything is working fine, like if the `BGP` sessions are still active, the `OSPF` is still running correctly or the major link beetwen HALL and PYTH is still up. See section about report scripts (6.3) for more details. The results of all those scripts are kept and reported in log files, stored on the monitoring server.

6.1 Monitoring server

We made the choice to place the monitoring server on a LAN on the router CARN. The reason we are doing this is first because we don't want to overload our two boarder routers HALL and PYTH and because our monitoring server doesn't really need to be close to the ISPs. The second reason is because CARN is physically more able to handle multiple infrastructures. The MONI-eth0 interface of our monitoring server can be accessed by the following address : *fd00:300:4:e11::3/64*.

6.2 Simple Network Management Protocol

SNMP daemon In order to be able to perform the different `SNMP` requests we want to question the network, we use the deamon `SNMPD` provided by `Net-SNMP` . This tool is constantly running since the network creation and allow us to deploy configuration files on every agent machine. Based on this, our monitoring server is able to query the different routers and retrieve informations like the number of incoming packets, the time since interface is down/up or even the number of discarded packets due to protocol errors, thanks to the Object Identifiers (`OIDs`).

Configuration files As explain above, all agents (so it corresponds to every router that should be questioned) have a configuration file called *"snmpd.conf"*. Except the name and location, the informations are equivalent for every routers. Indeed this file specifies the agent behaviour, which must be the same for every devices queried by our monitoring server. We set up the agentAddress to `udp6:[::]:161` and not `udp6:[::1]:161`, so that the router isn't restricted to receive only local requests but can be reached by an

another server, in this case our monitoring server, on the udp port 161. We also create a user with the correct permissions (read and write). Because we are using SNMPv3, we're able to specify security policies with an authentication and encryption password. Eventhough we know that SHA1 and AES are better authentication algorithm protocols [3], we had some compatibility issues to make them work correctly so we decided to stick to the default protocols MD5 and DES. Normally, this file is encrypted so that only the users with the right credentials can access the agent MIB.
On the client side, we use another configuration file called *"snmp.conf"*. The file is made to simplify the different requests by specifying special MIBs we want to access and informations like the user and kind of authentication used so that we don't have to write them down every time we made a request. Note that we didn't setup any special MIBs since our SNMP requests doesn't need more than the ones provided by the *snmp-mibs-donwloader* package.

PySNMP In order to ease the automation of SNMP requests, we decided to use the PySNMP engine [2] in order to perform requests in Python.

SNMP script The monitoring server is constantly running a script that retrieve informations on every routers. For each of them, the script will list all interfaces and compute the bandwidth, the number of errors due to a failure in the transport layer and the amount of packets voluntarily discarded, due for example to a lack of space or because the resource already exists. This is made by sending first WALK and then GET requests, in order to first determine the different OIDs and then select the ones that are relevant for the computations. Those different statistics result from the application of a specific formula that needs an interval of time in order to perform the calculation. We decided to set a timer of 2 minutes in order to collect enough relevant informations and still be able to notice anomalies quite fastly. The output of the script is a dictionary sorted by addresses. For each of them, the different interfaces are listed and contain an array with the IN and OUT statistics.

6.3 Report scripts

Once the *create_network.sh* script is finished, we make a call to one of our script in an other process that keeps sending PING requests from one router of our network to another (here we choose Stevin and Halles since they're not adjacent). If not any PING requests succeed, we conclude that they're is a problem with the network creation. Otherwise, we launch a script whose job is to run the SNMP and all the report scripts and redirecting their output to a */etc/logs/* folder in the monitoring server that keeps all logs in the appropriate file. This main script never stops in order to keep an eye on the network situation. We decided to put a timer of 5 minutes between each execution of all the scripts because since the scripts output a lot of informations, the log files could easily become really huge, and since the objective is to run those scripts during a long period of time, we thought that 5 minutes is acceptable to detect anomalies. Moreover, putting this timer also allows the SNMP script to finish completely since it needs 2 minutes of computation. Here below we describe more precisely the report scripts we use and how they work.

BGP To test if the BGP sessions are correctly established on Pythagore and Halles, we make two little checks. First, thanks to the BIRD6 deamon, we look at the status of the BGP session by asking to show all protocols on the router. We then simply return if either or not the status is "Established". The second idea is simply to ping the ISPs to see if PYTH and HALL still have BGP peering with their respective AS.

OSPF According to us, the best way to see if OPSF is still working correctly is to check every interfaces of each router. To do that, each router simply pings all the other routers on our network and waits for an answer.

DNS The idea is pretty much the same as the one used to check OSPF. Indeed, from every router we try to DIG all the other routers of our network. So for example on Stevin, we could try to do dig Carnoy by doing *dig carn.group4.ingi @fd00:300:4:e11::2*. We also ensure that the two namespaces can be resolved.

Link failure This script has been made to simulate whether or not the link between Halles and Pythagore fails. This is done by first setting down their interfaces that allow them to reach each other, we wait some time for OSPF stabilization and then make a call to the script assigning the static routes. We then check if the interfaces are up again. If not, there is probably a link failure.

SNMP We simply connect ourselves to the monitoring server and launch the `SNMP` script described earlier.

6.4 Improvements

We are totally conscious that this part is clearly incomplete. This is mostly due to compatibility problems with `SNMPD` that we didn't manage to solve during many weeks, even with the assistant's help, and thus stop us to make work relative to `SNMP`. But we of course know that this is not a valid excuse, so this is why we thought that it would still be relevant to explain what we would have done if we had more time. First, we would use `SNMP` requests in scripts in order to retrieve meta-data on the different routers instead of making them manually in bash. Then, we would have used safer authentication and encryption protocols. We would also retrieve more informations in the `SNMP` script, like errors due to a bad protocol or the memory usage of each router. Linked to that, we would act when some thresholds would have been crossed and thus set up some `SNMPTRAPs` to be able to recover from the different problems. Another improvement would concern the creation of the different files. Indeed, by ease in the beginning of the project, we decided to create both `SNMP` configuration files on all routers, eventhough only the *snmpd.conf* is relevant for the routers and only the *snmp.conf* is useful for the monitoring server. A last optimization we could have done is to create another monitoring server in order to have redudancy and be more robust in case of link failure.

6.5 Remarques de Mathieu

Pas grand chose à dire, il a juste spécifié que donc ici l'aspect monitoring qui avait été implémenté était plutôt général, style regarder le système dans son ensemble et regarder si tout fonctionne bien pour prévenir les modérateurs en cas de souci mais pas de réelles interventions ni solutions. Il a confirmé l'intérêt d'implémenter un 2e serveur de monitoring pour faire office de back-up qui effectuerait du coup le même travail que le premier. Il a également précisé que de manière générale les requêtes SNMP de type *snmpwalk* sont assez coûteuses en terme de bande passante puisqu'on interroge tout un sous-arbre de la MIB qui est assez conséquent. Du coup, dans notre cas, ce n'était pas très grave vu la quantité de paquets mais dans le cas d'un vrai déployement il faudrait essayer de minimiser au plus possible ce genre de requêtes et faire directement des *snmpget* étant donné qu'on sait déjà à l'avance ce que l'on veut monitorer.

Après reflexions je me rappelle pourquoi on faisait des *snmpwalk* à chaque fois avant de récupérer les ressources, c'est parce que leur OID assignés changaient assez fréquemment et donc on pouvait pas *snmpget* directement.

7 Conclusion

To conclude, we can say that this project was a great opportunity to learn about network configuration through a real example. We have learned how to set up, tune and troubleshoot a network from scratch, using concrete tools. We discovered the good and bad practices, the vulnerabilities that networks could face and the security issues that the network administrators have to deal with.

It was a really challenging project, both on the decisions we had to make and because, at the beginning, networking was something entirely new for us. First of all, we had to learn which tools to use and how a network works. Although it was a project, we had to take into account the real costs when making our choices, something we never did before and that was really interesting to do.

Along these lines, according to our group, this project is a success. Certainly, there is much room for improvement as we explained in the different sections of this report. We feel that we better understand computer networks. This is our biggest achievement.

8 Erratum

First change we made was to install and specify `Python3` for our script executions since, by default, the `python` command was using `Python2.7` to run the scripts, resulting in possible errors.

8.1 Addressing plan

For this section, we mainly provided more explanations on our choices for the addressing plan and completed the image of the topology.

8.2 Firewall

For the firewall we mostly correct some spelling mistakes. We also replace some `DROP` by `REJECT` since inside our network it is preferable to use `REJECT` (more

explanation about this choice in the firewall section 5).

8.3 Monitoring

A first thing we did was to justify the delay time of 5 minutes to launch the different monitoring tests. We also reduced the delays in *check_link_failure.sh* since 20 seconds is enough for OSPF stabilisation on our computers. The timer of the **SNMP script** has also been reduced and justified. Another modification we made was to kill the script launched after the network creation that was running infinitely. Indeed, when restarting the network, we were just creating a new instance of the process, resulting in a lot of similar background computations. Now, each time we clean the network, the scripts launched from `lingi2142/` are killed. A last change we made was to make the output of the SNMP script more readable by changing the different addresses by the name of the routers or interfaces.

References

[1] Timothy Rooney,"IPv6 Address Planning: Guidelines for IPv6 address allocation", `https://www.internetsociety.org/resources/deploy360/2013/ipv6-address-planning-guidelines-for-ipv6-address-allocation/`

[2] `https://makina-corpus.com/blog/metier/2016/initiation-a-snmp-avec-python-pysnmp`

[3] `https://info.townsendsecurity.com/bid/72450/what-are-the-differences-between-des-and-aes-encryption`

[4] P.Alilovic.Debuggingiptablesruleswithtcpdumpandwireshark. `https://sgros-students.blogspot.be/2013/05/debugging-iptables-rules-with-tcpdump.html`

[5] Debian. Debian firewall. `https://wiki.debian.org/DebianFirewall`

[6] A. Kis-Szabo. Man page of ip6tables. `http://ipset.netfilter.org/ip6tables.man.html`

[7] LinuxManPages. Man page of iptables-extensions. `http://ipset.netfilter.org/iptables-extensions.man.html`

[8] Information and communication technology section, `https://portal.ictp.it/icts/eduroam`, consulted on 04/12/2018.

[9] Information about SOA record, `https://en.wikipedia.org/wiki/SOA_record`, consulted on 05/12/2018.