2017 - LINGI 2142 – Project report
Configuring and emulating an IPv6 campus network

Professor: Bonaventure O. Assisted by: Chiesa M. and Tilmans O.

Group 1

Clarembeau Alexis	22401300	alexis.clarembeau@student.uclouvain.be
Descamps Robin	33251300	robin.descamps@student.uclouvain.be
Floriot Rémi	11381300	remi.floriot@student.uclouvain.be
Martin Olivier	22341300	olivier.martin@student.uclouvain.be
Michel François	10321300	francois.michel@student.uclouvain.be

Abstract

The deployment and configuration of efficient, redundant and secure IPv6 only network for enterprise and campus has seen a major increase of interest recently. The replacement of old IPv4 technologies is a huge challenge for network administrators. Also, the lack of comprehensive example of configuration for such technology has made the task even harder and time consuming. In this document, we will provide a full working IPv6 campus network. We will explain how we built it, what were the main pitfalls we had to overcome and how we solved those problems, to give a basis that everyone can adapt to enable IPv6 in their company.

1. Introduction

In order to give context to the project, to use consistent naming and design a system that fits to the reality, we will base our work on an existing campus. Here, it will be the "Université Catholique de Louvain" (we will denote it later as the UCL). This university campus is located in a quite large town, with hundred of buildings. Furthermore, it supports libraries, hospitals and schools that are connected to its network.

1.1. Objectives

Our main objective is to build a network that covers the whole city. It hosts services such as web portals, telephones, cameras, printers, desktops and research devices. It provides local and Internet connectivity thanks to a set of wireless access points widespread over the whole city. As the UCL hold sites in other towns such as Woluwe, Mons, Tournai, Brussels, we also need to take care to provide a flexible solution that can easily be extended.

1.2. Requirements

As we can see, the network should be designed to fit certain requirements. First,security should be enforced to protect our system against external and internal threats that could affect our critical applications (such as hospitals, grading systems and research). We must also resist misusage of the different tools.

As resilience is very important in our context, we must be able to resist the failure of any single infrastructure machine (router, server, host, ...) in the network.

Then, the system has to be compatible with a broad range of devices. Everybody should be able to be connected.

Finally, our system must be fast and efficient. Bandwidth and infrastructure as a certain cost and we want to optimize everything while keep it as constrained as possible and still respecting the other requirements.

1.3. Structure of the document

This document will browse through the solution we've brought to fulfill the different objectives and requirements, point after point.

We will start by defining the technology used to run and simulate the campus network on our computers. Then, we will talk about the routing, addressing and naming (that we can group under the term "end user management") and also about the services, their quality and security. We will conclude this document by an explanation of the testing methodology we employed and by taking the most important lessons of this project.

2. Technology

There exist many different technologies and tools to build a network. Here, we decided to use only free, well known open source software.

2.1. System & emulation

For the operating system, we focused on Linux. It is widely used in network environment and respects our requirement to be free and open source. The distribution selected for this project is Debian Jessie and it runs inside a virtual box.

The simulation of many machines is done using network namespaces. Those will allow us to replicate routing, firewall, links inside a single host, while sharing a single virtual machine and file system.

2.2. Template

To configure namespaces, we built a python template engine. Its role is to avoid repeating ourself by using generic configuration files with placeholders (designated with the [[...]] notation) to generate files for every nodes (the final configuration will be placed in the `gr1_*` files and directories).

All our network is defined in the `template/` directory and it uses the following organization:

- `main_*.py`: are the files used to run the engine (they should not be modified)
- `configuration_*.py`: are the files to configure the topology and the starting scripts
- `test/`: will contain all the test targets
- all the other files: are resources used in the startup scripts

Compiling and running the network is simply achieved by running `make`. It will generate and link the different files. Then, it will run the `create_network` script to create namespaces and run the scripts inside the nodes. To shut down the network, simply run the `make clean` command.

3. [All] Topology

Now that we have a working environment, we can start to design the topology. We will try to explain the design choices we made to arrive to the final solution described in the next page (figure 1).

3.1. Machine placement

We started by placing the routers. They will be disposed in a standard 6 routers ring topology. This helps us to resist the failure of any link and it is a good trade-off between performance and cost.

The routers "PYTH" and "HALL" are then connected to our two Internet providers: "AS200" and "AS300" using a very high bandwidth link to reach the rest of the network.

At the router "CARN", we added a lan with a data center. It contains all the critical services such as web portal or DHCP servers. This data center is fully redundant thanks to an additional connection to the router "MICH" an the usage of a layered switch topology (this link is not present in our code because of the lack of control of layer 2 resource provided by the startup script). As the data center will need a lot of bandwidth and that communication with PYTH is supposed frequent, we added an extra link between "CARN" and "PYTH" routers.

To complete this setup we decentralized the DNS service by placing two instances alongside CARN and MICH.

To finish the placement of the machines we added the hosts (computers, phones, cameras...) on L3 switches (BFLT, ADMI, SUD, SCES, BARB and INGI) linked to the core routers. L3 switches define LANs that categorize the type of user. The choice L3 switches instead of routers is principally for cost reasons.

3.2. Addressing plan

After that, we defined an addressing plan. It sets up the IP addresses and prefixes for all the machines. It is based on the two prefixes advertised by our providers, respectively, for AS200 and AS300: `fd00:200:1::/48` and `fd00:300:1::/48`.

With those prefixes, we can derive specific plans for links between host and routers, for links between routers and also, for the DNS.

3.2.1 Hosts

Each L3 switch will advertise to the hosts a prefix which will be used later for stateless autoconfiguration. At this point, it is conventionally recommended to use a `/64` prefix [4].

The hosts needs to stay able to reach Internet in case of failure. So, they must dispose from IP addresses of the two providers. To stay consistent, we will thus advertise prefixes that are concatenation of the two `/48` with a common `/16` part that we need to define ourself.

This `/16` part is made from 4 nibbles. We will name them Z, U, B and S.

Z is the zone where the machine lies. We subdivided Louvain-La-Neuve into four zones. North is 0, West is 1, East is 2 and South is 3. Other zones are kept for further use. For instance, if we add the campus of Woluwe, it will be zone 4. Saint-Gilles is zone 5 and so on. U differentiates the user types: 0 is a network administrator, 1 is a student, 2 a teacher, 3 an equipment (like a phone or a camera), 4 a service (HTTP, DNS, ...) and the other values are reserved. B is the building number inside the zone (green number in the schema). Finally, S is the subnet number and prevents two networks to have similar prefixes.

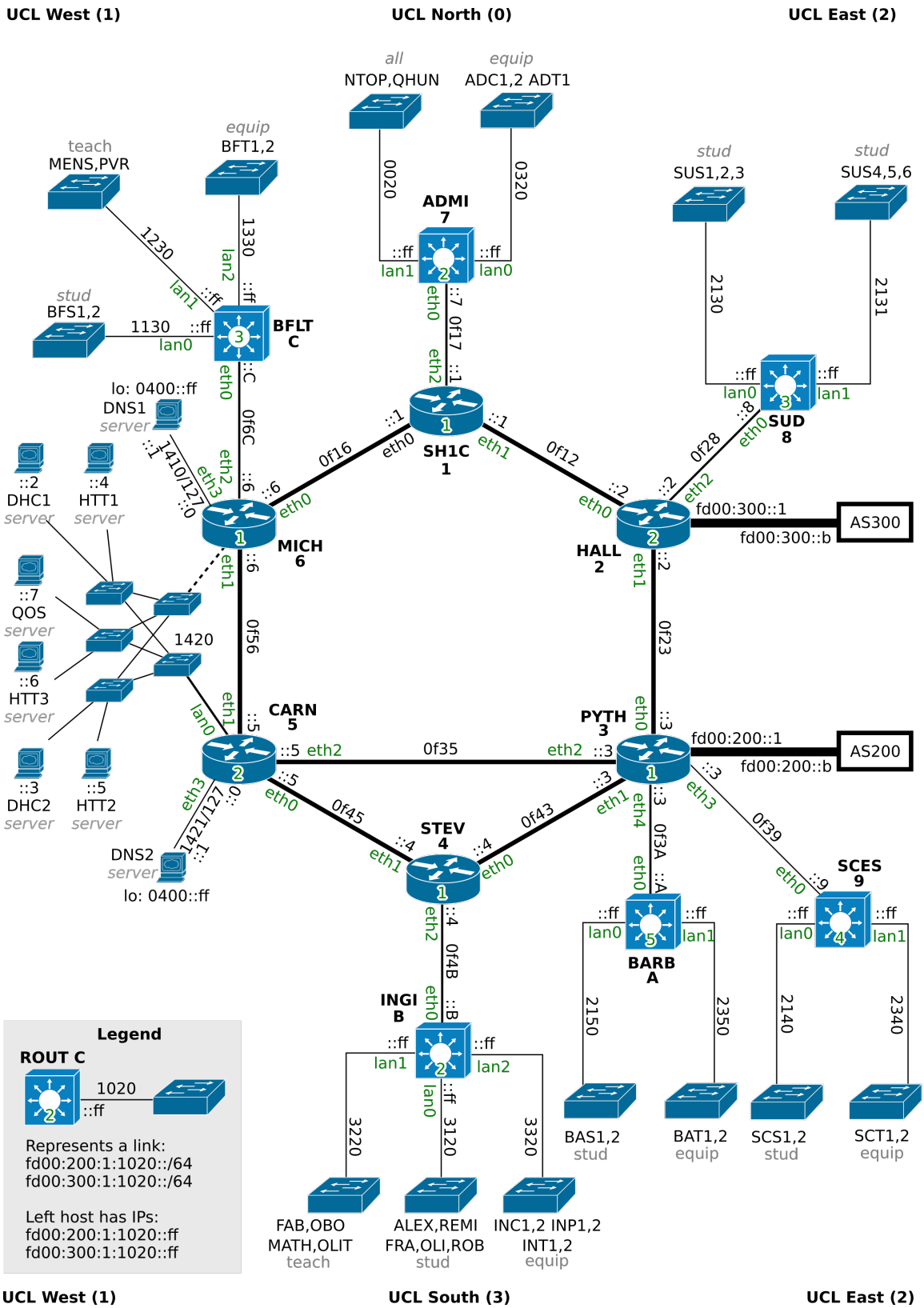

Figure 1. Entire topology and addressing plan of our network

This choice is a tradeoff between the amount of aggregation we will have in our routing tables (thanks to the zone prefix) and the simplicity of our firewall rules (thanks to the user type). The closer a nibble is from the start of the prefix, the easier it will be to filter.

Based on this convention, for instance, all machines in Louvain-La-Neuve West linked to the BFLT (building 3 inside west zone) of type "student", inside the first subnet will receive two prefixes:

- `fd00:200:1:1311::/64`
- `fd00:300:1:1311::/64`

Finally, the 64 last bits of the host will be a generated value derived from the MAC addresses for all machines, excepted for services, where we set manually the values to 1, 2, 3... to simplify the configuration.

3.2.2 Inter router links

For inter router links, even if it is not mandatory to use `/64` here, we decided to stay consistent and use the same prefix size. All the issues that will arise and are related to the absence of `/127`, as defined in [16] will be addressed differently in the security section (section 9.1.7).

Here, the prefix will be still derived from the two routable prefixes but now, we will interpret the nibbles differently. Zone will be 0, to share a common value between all routers. User type will be F, a special value reserved for routers.

Building and subnet will now reference the two routers that compose the link, in increasing order. Using unique number assigned in the scheme to the routers inside the topology (for instance, CARN = 5, MICH = 6), we can assign the link CARN-MICH with the prefixes below.

- `fd00:200:1:0F56::/64`
- `fd00:300:1:0F56::/64`

Finally, the IP is just completed with the router ID. If we talk about the link 5-6 on the side of 5, the IP will finish by `::5`.

Apart from this plan, routers also have a special loopback IP, that is only used for local troubleshooting and that is derived from prefix 200. Its value is `fd00:200:1:0F00::X`, where X represents the number of router X.

3.2.3 Special case: DNS

In our network, DNS are special nodes. We decided to use anycast in order to provide the answer from the closest DNS to each node, while keeping our network resilient.

To achieve this goal, we will provide the same static IP address to the two nodes: `fd00:200:1:0400::ff`. This IP will only be used by the end users to select an unique destination to reach "the closest DNS". As it is only used inside our network, it does not need to be doubled with a `fd00:300:1::/48`. And, we don't want our DNS to use this address as a source for the packets it will emit. That is why we defined it as a loopback (for reasons that will be defined in the naming section, subsection 6.1.1).

As the DNS cannot rely on loopback to send packets (this is discussed in 6.3.3), we need to add to each DNS IP addresses that are globally routable. Using the host addressing scheme defined above, we chose:

- For DNS1 :
  - `fd00:200:1:1410::1`
  - `fd00:300:1:1410::1`
- For DNS2 :
  - `fd00:200:1:1421::1`
  - `fd00:300:1:1421::1`

And, as all the wordarounds of 9.1.7 will not work here (because the DNS servers cannot be hidden from outside), we decided to use, and, only in this part of the network `/127` prefixes.

4. [Alexis] Routing

For the transport of the packets, we used a unique software: bird6. This software matches our open source requirement and is also rather used in large internet exchanges [24][8], where it replaced the quagga tool for scalability issues. It will serve as the basis tool for all the routing part.

4.1. Internal

The first part of the routing concerns the internal connectivity of the network. As a first step, each host must be able to contact any other host on the same campus network. This has been achieved using the OSPF link state IGP protocol. This is the de-facto standard for this task and is widely available [25].

In our project, it did not required a lot of configuration. We defined our network over a single area and set individual router IDs. We configured LAN networks as stub areas, as there is only one link from the L3 switch to the LAN. Also, we increased the cost of a link (here, PYTH-HALL) to add route symmetry (to support section 9.1.3).

4.2. External

Then, we built an external policy which defines how to reach the global internet.

We used BGP peerings between HALL and AS200 and between PYTH and AS300. Such peerings are configured with an import filter that only allows a default route (`::/0`). This is based on the expected behavior from the providers. We suppose that we have a contractual agreement with the provider that species this will be the only route announced.

In terms of export, we export the `fd00:200:1::/50` on AS200 and `fd00:300:1::/50` on AS300. The `/50` prefix specifies the Louvain-La-Neuve campus. As it has four areas, each of them begins with a double zero bit (on the first nibble, in binary notation, 0 is 0000, 1 is 0001, 2 is 0010 and 3, 0011).

These two import and export policies are sufficient to bring internet traffic on PYTH and HALL.

4.3. Mixing both

Now that we have local connectivity over the whole network and internet access on PYTH and HALL, the only step left is to redistribute the default route over the whole network.

For that, we configured bird to export the `::/0` route from BGP to OSPF. Combining BGP and OSPF this way makes our network reliable and resistant against the failure of any provider (the default route will flow through the nearest available provider on each host).

This will imply to use only the nearest provider. Based on the assumption that the load of our network is equally distributed over physical areas, it will share the load. If this was not true, we should use alternatives such as internal BGP to use a single provider over the whole network and use the other as a backup or more complex load balancing systems between the two ASes.

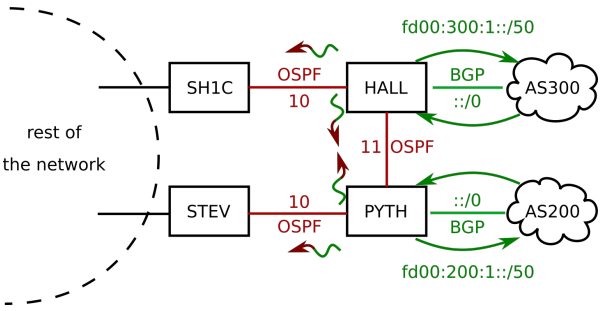

Figure 2. Our retained solution

4.4. Commercial agreements

Apart from that, we also agreed for a special high performance peering with another university. Supposing the existence of a optical fiber on PYTH between our group and group 3, we established an additional BGP peerings. PYTH now also establish the peering between our border router (address `fd00:200::1`) and that one of group 3 (address `fd00:200::3`, AS number 3).

Supposing a privileged connection of our west area and the entire network of that group, we export the prefix `fd00:200:1:1000::/52` and expect from them their `/48` prefix `fd00:200:3::/48`.

In order to keep a good compromise between cost and performance, we did not made the peering redundant and expect the traffic to flow through Internet in case of failure.

4.5. Blackhole

All the points cited above are sufficient to bring the connectivity (both local and distant, with commercial agreements). But, we needed to add an extra static rule to increase the security.

This rule discards (blackhole) all packets to our own destination prefixes: `fd00:200:1::/50` and `fd00:300:1::/50`. This will prevent a user to try to contact nodes that aren't defined in our network. This is based on the principle of white-listing and longest common prefix (LPM). If there isn't any more precise prefix in the routing table, matching the destination, learned with OSPF, the packets will be dropped.

5. [François & Alexis] End user management

In the section describing the topology, we stated that the interface identifier (last 64 bits of our IP addresses) can either be manually assigned, or derived from the MAC address. Here, we will discuss the two approaches. Then, we will talk about how to choose between the different available addresses.

5.1. Assignation [François]

Let's start by defining how the address are assigned. This section will explain how and why we preferred static or automatic assignation and also the technologies employed.

5.1.1 Static assignation

This is typically done on the different routers of the network and the nodes providing the different services in the datacenter (DNS, DHCP, HTTP servers). This permits us to ensure the fact that the IP addresses of these nodes are not going to change over the time and that the hosts do not depend on any other entity in order to have their IP addresses. The main drawback of this approach is the manual configuration overhead. This solution is not scalable and requires some human coordination to avoid IP conflicts. This is why we do not use it for the entire network.

5.1.2 Automatic assignation

Automatic assignation is thus the preferred solution for end and mobile hosts and provide a convenient way to get an

IP address. There are two major ways to compute this address: stateless autoconfiguration and stateful dynamic host configuration protocol.

Assigning with stateful DHCP

This approach has the advantage to avoid the manual configuration of the IP addresses. The idea would be to use a DHCP server to be authoritative on an IPv6 prefix and distributing complete addresses to any client that sends a DHCP request. The drawbacks in this approach is that it limits the number of IPv6 addresses that can be assigned to a client when performing a request. This is not recommended by [7] as it imposes a limit on the design of network applications.

Assigning with stateless autoconfiguration (SLAAC)

SLAAC permits an easy and plug-and-play solution to IPv6 addresses allocation on the hosts. The routers of the network announce the available IPv6 prefixes (in our case, there are two available prefixes) and the hosts build their IP addresses on their own, verifying that their address is free before using it. The main advantage is the fact that the clients does not require explicit requests in order to have an IP address (as it already receive the prefix, it can compute as many IPv6 addresses as they want), and it thus doesn't impose a limit on the number of IP addresses per request per host.

Our choice

In our network, we use static address configuration for stable services in our data center and SLAAC for the different clients of our network to avoid the overhead imposed by the static configuration and the limitations caused by stateful DHCP. We also like the idea of prefixes announced by routers instead of one node of the network (DHCP server). Furthermore, Android devices lacks support for DHCPv6 [26]. As we want our network to be usable by the largest possible number of types of users, using only DHCPv6 wasn't a solution for us. Nevertheless, for compatibility reasons, we didn't manage to avoid the use of a *stateless* DHCPv6 server as we will see in the section about name resolution.

5.2. Selection[Alexis]

After giving to every host two IP addresses, we needed to study which IP will be used as a source and if the choice will have an impact or not on the connectivity.

In our specific problem, we've got two prefixes, from different autonomous systems. AS200 gives us `fd00:200:1/48` and AS300, `fd00:300:1/48`. Traffic with first prefix won't be allowed to transit on the second AS and traffic of the second prefix can't go through AS200.

This means that when one provider is down, we need to use the other source address in order to keep internet access. This problem of the source address selection isn't simple and addressed in draft [2]. To summarize the problem, it seems that the source selection is done according to rules that doesn't take in account web reachability. As we can't force the hosts to install some kind of software, we can only influence this process through different attributes of the IP allocation.

The solution retained for this project is to advertise prefixes on router advertisement with different preferred lifetime. Thanks to a script periodically refreshed on the L3 switches, we can check for internet access (by pinging `fd00::d`) and then prefer one prefix over another.

For static IPS, such as those of the DNS server, and of the router inter-links, we also use the same mechanism of preferred lifetime, but we select its value with a periodic script running on the node.

6. [François] Naming

Now that we a have working internal campus network that is successfully connected to the outside, we should start thinking about the user experience, and the first thing that comes in mind is to use domain names instead of IP addresses to reach the different services. We thus created two hosts at different places in our network that are in charge of the name services. These are the nodes `DNS1` and `DNS2`. There are two nodes for redundancy. Each of these nodes is accessible from both prefixes, the ones announced by the AS 200 and 300.

Naming the nodes can be considered as a three step process. First, we need to define how we will resolve name in our network. Then, we will describe the precise implementation. Finally, we will explain how we distribute the DNS information to the entire network.

6.1. Name resolution

We will now define how we provide domain name resolution for the inside and outside of our network. We will explain what are the informations that are stored in our DNS server and how to provide answers for queries concerning zones outside of our network. Finally, we will explain how we manage sensitive informations that should not go out of our network in addition to the adaptation process of the informations provided by our DNS servers to the actual state of our different services providers, AS200 and AS300.

6.1.1 DNS local informations

`DNS1` and `DNS2` are responsible for the zone `group1.ingi.`. Every request concerning `group1.ingi.` can be redirected to `DNS1` and `DNS2`. For the caching of the informations we provide, we decided to specify a TTL of 300 (i.e. 5 minutes). Indeed, we are in a particular case where the informations stored in our DNS servers could change at any time (if an AS is down, the DNS will update their information to provide

IP addresses of the correct prefix) and it is also important for this informations to be quickly updated in our clients database, in order to provide services that are available most of the time. If one wants to make a comparison, a TTL of 300 is a TTL that is used for `google.be` records.

SOA Record Our DNS servers also provide the following `SOA` record :

```
group1.ingi.  IN  SOA
  ns.group1.ingi. root.ns.group1.ingi.(
  3     ; Serial
  1200  ; Refresh
  300   ; Retry
  2419200 ; Expire
  300     ; Negative Cache TTL
)
```

The `Serial` information will change each time we make a change to our DNS zones in order to indicate to a potential secondary DNS server that our informations have changed. The `Refresh` information indicates the time a secondary DNS will wait before refreshing its information concerning our zones by zone transfer. As we use two service providers that offers us two prefixes, if one provider is down, we would like to rapidly update our DNS informations in the case where we would have a secondary DNS server depending on ours.[1] This is why the `Refresh` values is not longer than 20 minutes, according to [3].

The `Expire` information would be used by a secondary DNS to know the delay of no-response of the primary DNS server on which the secondary should consider itself as authoritative for the zone. After this delay, the secondary server won't consider itself as authoritative for this zone anymore. As we would like an important amount of time to recover from failures, this value is set to 2419200, according to [3][2].

The `Negative TTL` information specifies the amount of time during which the "lack of answer" information should be cached by a querier. As `RFC1912` specifies it, this is not bigger than the original TTL value.

Remaining local informations The two DNS servers hold publicly the `AAAA` records (i.e. the records that map a domain name with an IPv6 address) for the different services present in the datacenter (for example : `website.group1.ingi` or `grading.group1.ingi`). They also contain hidden AAAA records for all the routers of our internal network.

These two DNS servers also contain publicly two NS records (i.e. the records containing the domain name of the DNS servers that are authoritative for this zone), one pointing to each server, `ns1.group1.ingi` and `ns2.group1.ingi`. One could wonder why in our `SOA` record, we put `ns.group1.ingi` as authoritative. This is a way to indicate to the potential secondary DNS servers that there are two authoritative DNS servers for this zone. Indeed, we mapped the IP addresses of `DNS1` and `DNS2` behind the domain name `ns.group1.ingi`, as we cannot provide two `SOA` records for one zone.

Our name services also provide reverse name queries resolution concerning the whole network, including our routers and services. Our DNS servers will thus answer to `PTR` queries, but as for the `AAAA` records, only `PTR` records concerning our services are provided to the outside of our network. Answering to reverse DNS queries concerning our routers is useful to provide more human readable information when using tools like `ping6` or `traceroute6`.

Reaching our DNS servers For the internal network, we have decided to provide only one anycast IPv6 address for both DNS servers. The anycast IP address is a unique IPv6 address representing more than one physical host. Using an anycast IPv6 address means that our two DNS servers will be reached through the same IPv6 address although they are physically located at different positions. The DNS queries will simply be routed to the closest DNS server from the querier.

In order to handle a failure of one of the two servers, they both use OSPF to announce their anycast address to the router with which they are connected. When one servers fails, the router will notice that they do not receive OSPF keepalives anymore, and all the queries will be routed to the remaining DNS server.

However, in order to provide a DNS service to the outside of our network, we still use global unicast addresses for our DNS server that are announced on the outside of our network. Furthermore, we configure the anycast address of the DNS servers as an address for the loopback interface in order to avoid them to use this anycast address as source when initiating a connection (note that if a DNS server receives a request for its anycast address, it will still use this anycast address as source for the answer to this query).

One might wonder why we do not use anycast IPv6 addresses for our DNS servers even for the outside of our network or to initiate connections with somebody else. This choice and the problems linked to anycast are explained later on this report (section 6.3.3).

[1] Note that we have allowed zone transfer on our DNS servers as it was asked in our assignment in order to perform interoperability tests between groups. This `SOA` configuration will thus also suit well a situation where one use secondary DNS servers.

[2] As we don't have any secondary DNS server but had to allow zone transfer for interoperability tests, we had to configure this value as a realistic example but not for a real use.

6.1.2 Digging the rest of the world

Now that our DNS servers resolve all the hostnames of the internal network, we also have to configure them to resolve domain names on the outside of our network. As we want our DNS servers to give the complete answers to the queries, we cannot use name servers that only answer to the user by giving them the address of a DNS server that could know the full answer to its query. We thus can configure our DNS servers in two ways :

- **Recursors** : in this configuration, the DNS server will recursively find the answer to the query by asking itself the query to the other authoritative servers for the different zones in the domain name hierarchy [21][3]. The advantages of this configuration are that the name servers give the complete answer to any query and the DNS will directly send the query to the DNS that are authoritative to the concerned domains. An important drawback is that a recursive DNS has quite a heavy job to perform for each query[4]. It is thus quite resource consuming as the DNS server has to perform the whole recursive process and it is prone to potential overloading.
- **Forwarders** : in this configuration, the DNS server will completely forward all the queries for which it is not authoritative to another server. The advantages are that the name servers give the complete answer to any query and it requires less work to our own name servers. The main drawback is that we have to find a trustworthy DNS server to delegate all our external queries.

As we consider in our simulation that we have a reliable external DNS server located at the address `fd00::d`, the two DNS servers are configured to forward their requests to the server located at `fd00::d`, which will answer to our queries for the domain names of the outside. Our two servers are thus configured as `authoritative` for the domain `group1.ingi.` and `forwarders` for all the rest.

6.1.3 Protecting our private infrastructure from the outside

As our DNS servers are also resolving domain names of our routers for debug, in addition to reverse queries for the routers, we could want to hide our infrastructure from the outside and avoid our DNS servers to resolve sensitive queries if the source of the query does not come from the internal network. We thus use the principle of *split horizon* [23][5] in order to filter the DNS queries from the source IP address. Typically, our DNS servers will answer to all queries if they come from the internal network, and answer to queries concerning our public services coming from everywhere.

6.1.4 Handling AS failures

As our DNS servers announce ip addresses used to join our services, it has to adapt if an AS goes down, because our services won't be reachable by its prefix anymore. We thus added an AS-check behaviour on our DNS servers. They will always announce only one prefix. They will announce the prefix announced by AS200 by default. If the AS200 is not reachable anyore by sending a packet to the internet with a source IP with the AS200 prefix, the DNS servers will whange their configuration and provide the prefix announced by AS300 instead, until AS200 is reachable again.

6.2. Our DNS infrastructure in practice

We used Linux hosts to represent our DNS servers. We used the `bind` service which offers a rather quick and easy to configure DNS both for IPv4 and IPv6. In addition, `bind` provides the concept of `views`, which is an implementation of the concept of *split horizon*.

Configuration The file `named.conf.local` located in `/etc/bind/` of `DNS1` and `DNS2` contains two views, one for the queries coming from the internal network and one for the queries coming from everywhere else. The first one is more complete and contains (normal and reverse) resolution for the different routers of our infrastructure. The file `named.conf.options` contains informations about the behaviour of the name server. It indicates that the DNS server does only forwarding (instead of doing recursive queries) when it receives requests for which it is not authoritative. It also specifies the DNS servers to which the queries are forwarded and the IPs to which it answers (in our case, the DNS server answers to everyone as it has to give the IP addresses of our public services).

Zone files The files `db.ingi.private` and `db.ingi.public` are zone files (i.e. files that contains informations about the zones for which the DNS servers are authoritative), one for the queries coming from the internal network, the other for queries coming from everywhere. Each of these different files thus corresponds to one of the two views specified above. As having redundant informations in two separate files is not a good practice and prone to errors or incoherences, these two files only contains the

[3] Please look at section 4.3.1 if you want a more detailed explanation of recursive DNS

[4] Note that most DNS servers use a cache that avoid them to process completely identical queries again and again.

[5] Please read section 4 of this reference for further informations.

SOA records and import other files containing the records of the DNS servers. db.ingi.private.200.reverse and db.ingi.private.300.reverse are reverse zone files dedicated to the reverse resolution of the IPs of the routers. They are thus served only to the queries coming from the inside of our network. As the name of the reverse zone depends on the IPv6 address that are reversed, we have one reverse zone per prefix, i.e. the zones 1.0.0.0.0.0.2.0.0.0.d.f.ip6.arpa. and 1.0.0.0.0.0.3.0.0.0.d.f.ip6.arpa..

Handling AS failures Our DNS servers have two directories in /etc/bind/: One fontaining the data to announce for the AS200 (the 200/ directory), and the other for the AS300 (the 300/ directory). As soon as AS200 is not reachable anymore, it will copy all the files present in the 300/ directory into the /etc/bind directory, erasing the files that were already present. It will then restart the bind9 service. As soon as AS200 becomes available again, it will do the same but with the 200 directory. It is an easy way (and maybe a bit overkill) to be adaptative to AS failures, and it works well il practice.

6.3. Providing the DNS addresses to the clients

Once we have DNS servers that are resolving the different domain names of our infrastructure, we have to think about the way to indicate the IP addresses of our DNS servers to the clients of the network. There are different ways to announce the IPv6 of our DNS in our network :

- **Adding the DNS servers addresses manually for each client.** This solution works but requires a manual configuration for each client of our network and is not scalable at all. This solution is thus discarded.

- **Announcing the name servers IP addresses with the help of a DHCPv6 server.** As we do not use a DHCPv6 to assign the IP addresses of the hosts of our network, that would mean to setup and configure a whole DHCP server only to this purpose. The DHCP server would thus only announce the name servers IP addresses without assigning an IP address to the host that would make a DHCP request (This is called stateless DHCP [11]). We would also have to setup DHCP relays at each router to relay the DHCP requests to the DHCP server. This is a solution that we would like to avoid because it implies the configuration of a whole DHCP and a DHCP relay infrastructure only to provide the IP addresses of our name servers.

- **Announce the DNS servers addresses by using the RDNSS extension of router advertisements [14][6].** This is a solution that requires less configuration than the setup of a DHCPv6 server and it reduces the network traffic by providing the IP addresses of the name servers directly in the IPv6 Router Advertisements.

After evaluating each of these alternatives, the last one using the RDNSS seems the most promising as it is only an extension to a service that we already provide. Nevertheless, choosing only one of these technology for the whole network would require all the different client to implement this technology in order to retrieve the DNS servers IP addresses. The fact is that the RDNSS extension to the Router Advertisements is not supported enough by the different Operating Systems to use it exclusively. Indeed, the support of RDNSS is for example not guaranteed by Windows[27]. As Windows is one of the most used Operating Systems for Personal Computers, we decided to provide a way for the Windows users to retrieve automatically the IP addresses of the DNS servers in our network. As the DHCPv6 is supported by Windows, we could use DHCPv6 instead of the RDNSS extension of Routers Advertisements. However, if Windows provides a support to query DHCPv6 servers, it does not seems to be the case for Android. Inversely, Android is compatible with the RDNSS extension to Router Advertisements. As we want our network implementation to be compatible with the biggest possible number of Operating Systems, we decided to provide both DHCPv6 and RDNSS. The clients of our network are thus allowed to use the one they want.

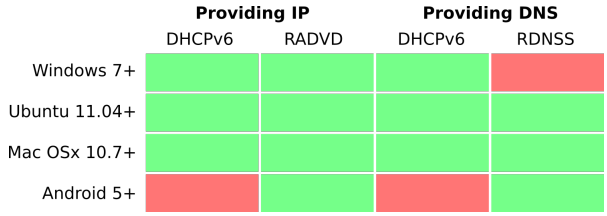

Figure 3. Naming / addressing compatibility table

6.3.1 DHCPv6 in practice

We provide two DHCPv6 servers for the redundancy. They are not authoritative to assign IPv6 addresses to the clients, they only provide the addresses of the DNS servers. We thus setup two Linux DHCP servers by using the dhcpd service. You can find the dhcpd config file in /etc/dhcp/dhcpd.conf. The DNS servers IP addresses are indicated by using the option dhcp6.name-servers. The DHCP will thus transmit these addresses each time a client asks for the name servers IP addresses in a DHCPv6 request. The file /etc/default/isc-dhcp-server specifies the interface on which the DHCP server will listen for requests and the arguments that will be passed to the dhcpd exe-

[6] Section 5.1

cutable (principally the `-6` arguments that specifies that we want to run a DHCPv6 server).

In order for our DHCP server to be reachable by the clients, we have to configure DHCP relays on each router of our network. These routers will listen for DHCPv6 requests and relay the information on their interfaces in order to finally join the DHCPv6 server. As our routers run under Linux, this is done by using the `isc-dhcp-relay` service. The router calls this executable at boot by specifying the interfaces on which it listens and on which it sends DHCP relay packets. Typically, the router listens on all its interfaces and can potentially relay packets on all its interfaces that are not leading to one of its LANs, excepted for the router connected to the LAN of the DHCP servers which will send these relay packets only to this LAN in order to reach the DHCP server. It will in practice send the relay packets to the interface from which it can join the IP address of the DHCP server in its routing table. If this route goes down because of a failure, as it can potentially send a relay packet to every interface, it will send the packet through another interface that can lead to the DHCP server.

6.3.2 RDNSS in practice

We added the RDNSS extension to Router Advertisements on each router of our network. As our routers run on Linux and already use `bird6` for OSPF and BGP peering, we reused `bird6` to send router advertisements. The RDNSS option is configured on every router of our topology connected to a LAN in the file `/etc/bird/bird6.conf`. `bird6` is not the only way to provide support for Neighbor Discovery Protocol and RDNSS, one can use another tool specifically dedicated to this like the `radvd` tool for Linux. We first used `radvd`, and a working example of `radvd` configuration can still be found on our git repository if you navigate through previous commits.

6.3.3 DNS Anycast

As we announce only the anycast addresses of our DNS servers in our internal network, one good improvement would be to do the same for the outside of our network. Indeed, using anycast addresses for our DNS servers would provide us some geographical load balancing by the fact that every packet towards an anycast address would be routed to the closest host that uses this anycast address.

However, in practice, we announce our two DNS servers separately with their global unicast IPv6 addresses to the outside of our network. The reason why we do not use unicast addresses for the outside of our network is because it will imply different routing and redundancy problems. Indeed, using anycast implies to route the packets to the closest host to the source. With such a system, we do not have any guarantee that the answer to a question asked by `DNS1` won't be routed to `DNS2` instead of `DNS1`. As our DNS servers act as forwarders, we could be in the case where the answer to a forwarded query would be forwarded to the other name server which did not forward anything. The initial client would thus never receive the answer to its initial query (this is illustrated in figure 4). We can have the same result for each situation in which one of the name servers would ask something to the outside and wait for an answer. For example, `AXFR` transfers are done on top of TCP, and this implies a connection between the two hosts, and thus packets that transit in the two directions.

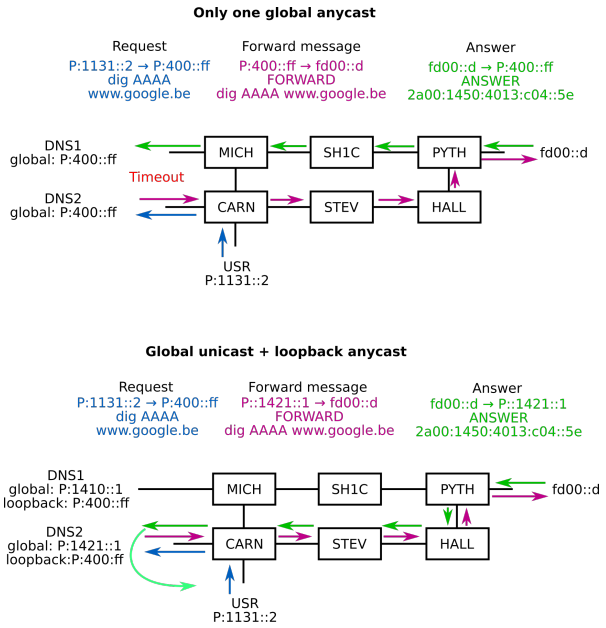

Figure 4. Illustration of why we need a unicast address in our network

To handle this problem, we need to ensure that the answers of the requests sent by any DNS anycast node would be routed to the node that emitted the request. Solving this problem would be a great improvement of our network.

7. [Olivier] Services

Once the network is built and operational, we can add different services on top of it for administrators and end-users. We chose to offer HTTP web services, remote SSH access and some network monitoring.

HTTP In order to propose web services to our end-users, we chose to use `lighttpd` because this is a fast, secure and flexible open-source web server [19]. In our case we do not need a full version of apache because we are only running small web servers.

We installed and configured several web services on nodes HTT1, HTT2 and HTT3 in our datacenter. Each of

these nodes propose the same HTTP services to ensure redundancy of services and load balancing for traffic even if a server fails. We also used virtual hosts to run multiple services on a single node. These web servers are running on port 80.

The load balancing is done at the level of the DNS: each time a client queries the DNS, the DNS returns the 3 IPs of HTTP servers in a different order following a round-robin procedure. Therefore, HTTP traffic will be shared between the 3 servers.

The available services are the following:

- `www.website.group1.ingi` is a basic example website.
- `www.chien.group1.ingi` is a website with an ascii dog.
- `www.grading.group1.ingi` is a website for teachers to grade students. Teachers need an account to log in.

SSH Nodes HTT1, HTT2, HTT3 and all core routers in our network are accessible via ssh on port 22. SSH is used by administrators to connect remotely to machines and manage them. To allow ssh on these machines, we installed and configured `sshd`[7], an OpenSSH Daemon program for `ssh`[8].

We disabled the root login for security reasons over ssh in order to protect our machines againts bots that try to log in with the root user with for example `ssh root@ip`. Indeed, if an attacker get a root access of one of our machine, all our network may be compromised. We also disable password authentication because this is a weak secutity to protect users in case the server is compromised [28] and against attack by dictionary.

We set up RSA keys with pass-phrases on private keys to protect our machines and establish authentication of administrators. Administrators can connect over ssh if they have the correct private key and the password of this key. Each machine has a different pair of public/private key. The public key of administrators are located in our machines in `~/.ssh/[[node]]_authorized_key` where `[[node]]` is the name of the machine. Note that there is only one user for all machines: the *vagrant* user.

We also added a custom welcome banner for each machine when administrators try to connect.

For further information on how to install or add `sshd` on a new node or how to connect with ssh in our network, consult the `README.md` located in `LINGI2142-setup/template/services/ssh/README.md`.

[7] http://man.openbsd.org/sshd
[8] http://man.openbsd.org/ssh

SNMP The network can be monitored by the administrator with SNMP request. To provide this service, we installed `snmpd`[9], a daemon that responds to SNMP requests on all core routers and L3 switches. These requests are using UDP on port 161. For security reasons, the firewall is configured to allow only administrators to send SNMP requests on port 161 with UDP. In our network, this is the QHUN node.

In addition to that, administrators also need an SNMP manager application like for example OpenNMS[10] to monitor the network. This kind of applications will make SNMP queries to produce graphics and statistics.

8. [Robin] Quality

The quality of service ensures that all services, particularly the sensitive ones like the VoIP or the camera streams, run properly. To do so, priorities are assigned to each of them. Higher priorities get link bandwidth before the lower priorities. The tool used so define our network QoS is `tc` [18]. This tool is applied to all of the interfaces belonging to the routers in the network, and all of the commands can be found in `/template/qos/shaping.sh`.

We define traffic shaping only in egress, that is to say packet sending (or upload). QoS over ingress is much more complicated, because we do not control the incoming streams. It will simply take time for the incoming stream to adapt to the reduced rate defined by ingress QoS, due to all buffers from the stream to the router. So, ingress shaping works with long-lived streams, but not well with short-lived streams, where packets will be dropped.

The objective here is to ensure that all services run properly, even in case of congestion. The VoIP and camera flows have the highest priority (but does not need of a great amount of bandwidth), then come the services like SSH and DNS. Below that, there is the default priority for other protocols, such as HTTP(S), with the greater part of the bandwidth. Finally, below everything else, we find the backup protocol (it has the lowest priority).

8.1. Shaping algorithm

Without Quality of Service set, the default behavior of the routers when forwarding packets is to place them in a queue (FIFO). The algorithm we use to shape the traffic is the Hierarchy Token Bucket (HTB) shaping algorithm [10], based on the Token Bucker Filter (TBF) algorithm. While the latter is classless, HTB allow us to create classes of traffics forming a hierarchy. This hierarchy is illustrated in the figure 5. The HTB algorithm works with "tokens" that are allocated to the packets incoming

[9] http://net-snmp.sourceforge.net/docs/man/snmpd.html
[10] https://www.opennms.org

to the corresponding interface. In case of congestion, packets are enqueued. Then, the packets are lead to their corresponding class in the hierarchy. We use HTB because we do not need to know the interfaces characteristics, unlike CBQ for example (we do not detail this algorithm here since we do not use it). With HTB, classes in the hierarchy can "borrow" bandwidth to other classes, if remaining bandwidth is available. A last remark, note that HTB algorithm does shaping, not to be confused with policing. With a policy, excessed traffic is dropped, while in a shaping algorithm excessed traffic is delayed with a buffer.

8.2. Traffic classes

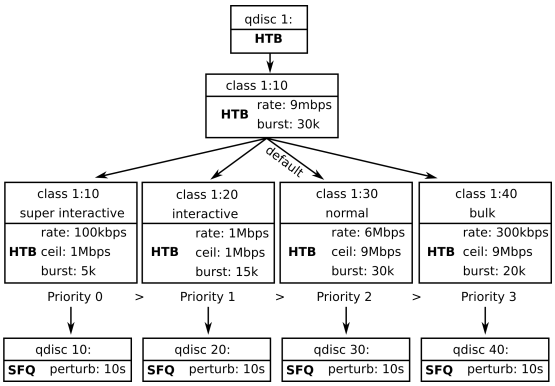

Figure 5. Explanation of shaping policy

We begin by defining a rate : 9Mbits/s. We consider here that the links have a limit of 10Mbits/s in terms of bandwidth. By setting the limit at 90%, we ensure that we never exceed the link capacity (indeed, 10Mbits/s is considered as a theoretical value).

Then, we define 4 classes of traffic: super interactive (1:10), interactive (1:20), normal (1:30) and low (1:40). Their priorities are respectively 0, 1, 2 and 3 (lower number are higher priorities). However, we consider that defining only priorities on traffic is not sufficient. Two problems arise: first, it is difficult to test (what speed must we observe in function of packets that we send ?). Second, for security reasons. Typically, super interactive class handles services that need a low latency (such as ICMP or VoIP) but not a great amount of bandwidth. Let's imagine a situation where an user manage to send an heavy (normally unauthorized) service (such as HTTP) through a VoIP phone. Without ceil on this traffic, the link will generate congestion for all other class of traffics. This is why we define **rate** and **ceil**. The **rate** is the part of the link's bandwidth occupied by the corresponding class, while the **ceil** is an upper bound that the class can never exceed (it means that the class cannot borrow more bandwidth if it reaches its ceil).

We also define bursts on the classes. Our super interactive classes, that need low ping and high priority, do not need to burst data. At the opposite, low classes work by sending burst of data (like HTTP), and thus need a high value of burst. The **burst** is the amount of data that can be instantaneously transferred. The computation of the maximum burst is the following: $Bc = Tc \times CIR$. Bc is the burst we calculate (the commited burst), Tc is the interval (calculated on one second) in which the burst is commited, i.e. the burst is sent (We will assume here that it is set to 125ms, a common default value in Cisco routers) and the CIR is the Committed Information Rate (the "contract" rate, here 9Mbits/s). Thus, the bursts defined in our classes cannot exceed that amount, and ensure that the parent class burst is greater or equal than all of its children's bursts, and thus is the maximum burst that we computed here [6] [22].

8.3. Stochastic Fairness Queueing

The "leaves" in our hierarchy is represented by Stochastic Fairness Queuing (SFQ) algorithm [17], in each class of traffic. This ensures that each flow of packets gets a fair part of the bandwidth, performing a round-robin scheduling on them. A flow is defined by a hash function based on source address, destination address and source port. Note the **perturb** parameter, that modifies this hash function every 10 seconds, in order to avoid blocked packets in case of collision with the hashing. By default, without SFQ, our packets would be placed in a unique FIFO, with a possibility to delay new connections. The SFQ helps also to mitigate Denial of Service attacks, by classifying flows in different queues.

8.4. Filters

Now that classes are defined, the only remaining element in our `tc` configuration are filters, that is to say how we class the different packets that arrive on the router's interfaces. To do so, we mark packets with specific numbers through a **prerouting** on ip6tables. As for the services and the security aspects, we differentiate the packets by looking at source port or destination port, and at the protocol (TCP or UDP) (except for the ping service, we differentiate ICMP only by looking at the protocol). In ip6tables rules, the **prerouting** is done before the **forward** described in the security section. The purpose of the **prerouting** is to add/modify informations on the packets, like we do here.

Here are the classes with their traffic details :

- Super interactive traffic: ICMP (ping), SIP (VoIP), RTCP (VoIP), Cameras
- Interactive traffic: SSH, SNMP, DNS
- Normal traffic: HTTP(S) and other (default class)

- Bulk traffic: Backup protocol

We chose to give the highest priority to the ICMP protocol. The advantage is that when we perform a ping on the network, the connectivity results are relevant, which would not always be the case if the ping had a lower priority and thus would possibly be dropped due to traffic congestion. The drawback is that the network is vulnerable to Denial of Service attack by the ping service. However, the super interactive traffic is limited to 1Mbits/s, and as seen previously, the SFQ helps to mitigate DoS attacks. The time indicated by ping also becomes not really correlated of the load of the network.

9. [Rémi] Security

In this section, we will explain the overall security issues that we have treated and we will detail our solutions to face them. Then we will discuss some possible improvement that could increase the security of the network. The references contains some of the website that were consulted to learn good security practices and to have a better understanding on the `ip6tables` tool: [1], [5], [9], [12], [13], [15], [20].

9.1. Security management: general principles and implementation

We will describe here the security rules deployed on the routers and the reasons that motivated these rules. The order of these security rules is important as when the firewall inspects a packet, it stops as soon as it finds a rule matching this packet. The majority of the rules described below are deployed on all the routers/L3 switches but some rules are more specific to a group of routers in the network (for example the border routers) and are thus deployed only on these particular machines. For a second reading, you could have a look in parallel on the implementation of the rules in the start script of one of the router/L3 switch as this detailed description tends to follow the order used in the implementation.

9.1.1 White-listing approach

First of all, we decided to apply a white-listing approach: a packet that does not match any of the rules we wrote will be discarded. It allows us to have more control on what can pass through the network. This can be cumbersome to manage but we prefer it for a security point of view as all packets that we did not take into consideration will simply be dropped. In the ip6tables, there are 3 filters to set-up in each router: `INPUT`, `FORWARD` and `OUTPUT`. It corresponds respectively to the filters that will inspect the packets that are destined for the router, that pass through the router which is not the destination or that are created and sent by the router. We apply this white-listing approach on these 3 filters.

9.1.2 Hiding the topology

Then we decided to hide our topology to the external users (to the users that do not belong to our network) by denying the `ICMPv6` packets toward our network as well as the traceroute that allows to see the different routers between a client and one of our servers. We made this choice to make the construction of a plan of attack against our network more difficult. Indeed for example, an attacker that wants to break our network could for example with a traceroute get the ips of the 2 border routers and launch a ddos attack only on these two machines. This would have the effect to cut the campus network from the rest of the internet. By hiding the topology, it is more difficult to study precisely which machines are the critical parts to target. Although dropping `ICMP` packets towards our network forbids a standard way to check connectivity between hosts (`ping6`), it prevents us from attacks against our topology, this is why we decided to drop them. To perform this blocking, we add on the firewall of the "border routers" (we call border router the 2 routers connected to belnet through a BGP session) some rules to block incoming ping and traceroute packets on the specific interfaces where the BGP session is set-up and we block the outgoing ICMP-time exceeded packets.

9.1.3 Trusting the established connections

We need to allow the established connections as, for example, if a user in the network send an `http` request to a server, the response should not be blocked. As generally a random port is used by a computer sending a request, we want to avoid to encode and especially allow all the possible ports for the responses. Allowing a huge range of ports goes against our white-listing considerations. Furthermore we should note that allowing all packets that have, for example a source port 80 (`http` response), is potentially unsecured as then an attacker may construct packets with source port equal to 80 and reach any normally blocked destination in our network. The solution we found is to allow the established connections (the connections for which we have already seen and authorized a packet in one direction). This solution has two drawbacks: the firewalls has to keep some informations in memory (state-full firewall) and the response packets should take the same path (with reverse direction) as the request packets. Indeed if a router see a response packet and did not see any other packet for this connection, it will drop the packet. This configuration is done in routing by increasing a little the weight of the link `PYTH-HALL`, which is sufficient to avoid any situation inside our network where there are 2 or more possible paths with same sum of weights.

9.1.4 Protection against IP spoofing

The firewall should prevent against IP spoofing. As explained in the section about addressing, there are several user types and the IP address will encode the number of the user type that sends a packet as well as a zone number, a building number and an additional subnet number. That's where our protection against IP spoofing comes: we have to deploy on the L3 switches directly connected to the lans that contains end hosts a firewall that checks all incoming packets from these lans. This firewall drops any packet with a source address inconsistent with the addressing schema. This is important to prevent for example that a student can act as an administrator by changing his source IP. For the implementation, as we have dual-homed source IP and so 2 source IP to allow, it is a bit tricky to encode this with the `ip6tables` tool that does not accept a rule like "*Drop any packet where source not in [first IP prefix, second IP prefix]*" in a single rule. The solution to allow the correct source IPs one after the other instead of dropping the bad source IPs is not a correct solution as the execution of the firewall filters is stopped as soon as a matching rule is found and so it will always let the packets go without further verifications. Our solution was to add a custom chain that is triggered from the main filter chains. In this custom chain we return directly to the main chain if the source IP is in the 2 subnets authorized (or if it is a link local address or a default unspecified source IP) and then we drop everything else as we consider it as a spoofed source IP.

9.1.5 Additional rules dedicated to the border routers

At the "border routers" (the routers that are connected with Belnet through a BGP session), we have also added an anti-spoofing filter: we drop any packet coming into our network from the Internet that have a source ip that belong to our network. The border routers play other roles: they block the outgoing packets that have the wrong source IP (depending on the BGP peering, only an address belonging to the prefix `fd00:200:1::/48` or the prefix `fd00:300:1::/48` will be accepted to continue towards the internet network). We also block on the interface that connects our network to Belnet any `DHCP` and `OSPF` packet in both directions to be sure that such messages cannot be injected in our network from the internet and to avoid in case of routing configuration error to send by mistake `DHCP/OSPF` packets to the internet and potentially let an attacker discover easily our full topology. Furthermore, the packets coming into the network and that target the prefixes `fd00:200:1:0f00::/56` and `fd00:300:1:0f00::/56` are blocked. These prefixes contains the IP of our routers and blocking these communications is a way to reduce the risk to let the external users discover our topology or that one of our router becomes the target of a ddos attack for example. Finally we don't forget to allow (as for before, only on the interface where the BGP session should be established) BGP packets to come in our 2 border routers (INPUT rule) and to go out of our 2 border routers (OUTPUT rule).

9.1.6 Rules for the routing packets, the testing packets and for the DHCP

After having discarded many unwanted packets, we can add rules on all the routers to let `DHCP` packets move inside the network. The `OSPF` packets are also allowed to be exchanged in the network but only on the `router-to-router` links. We also accept `traceroute` and `ICMP` packets inside the network, except the `ICMP router-advertisement` that is dropped in `INPUT` to prevent from a denial-of-service attack against the routers/L3 switches [5].

9.1.7 Protection against the neighbor discovery attack

As explained in [12], our routers/L3 switches are exposed to a security problem: as the standard subnet size in 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 responsible for the targeted prefix would then have to perform a neighbor discovery for each individual request of the scan when the destination is not known by this router. This can fill the memory of the router: as written in this RFC, it can "exhaust available memory and replace existing in-use mappings with incomplete entries that will never be completed". This "can induce a denial-of-service condition". We used 2 ways to address this problem:

- The routers are attributed address in a reserved subnet of our given prefix: On the `router-to-router` links the routers are assigned an ip in the prefix `fd00:0X00:1:0F00/56` (where X can be 2 or 3 in order to be able to send back response toward any AS. For example if the MTU is exceeded we are able to reply with `ICMPv6 packet too big`). To protect this part of the network, we drop every packet that target this specific prefix (only administrators are still able to contact directly a router).

- "Minimal Subnet Sizing": It was decided in the topology section that the DNS hosts will be directly linked to a router with an `OSPF` session established between them. As the DNS must be reachable, we can't assign to the link between the DNS and the router the `fd00:0X00:1:0F00/56` prefix used above to discard all the traffic. Here the chosen solution was to

give a /127 prefix so that only two hosts can belong to the prefix (the DNS and the router) and so we avoid this neighbor discovery attack there.

Finally the L3 switches have links to LANs where the users are connected. There we cannot easily deploy a solution against this attack, we assume as discussed with the teaching team that the switches manage this themselves.

9.1.8 Users management

Now we come to another important part of the security: the users management. Inviting the users to install anti-virus software is not sufficient. We know that the users inside the network can voluntarily or without knowing it (with an infected computer for example) attack the network. To reduce the risks we also apply a white-listing strategy for the services: we deny all services (protocols/ports) except the services we listed. As explain above, all users can send `ping` and `traceroute` requests from the inside of our network and can contact the DHCPv6 server. TCP on port 5001 is open for experimental purpose. Then, each type of user has its own list of other allowed services:

0 Administrators: System administrators have no restrictions inside the network.

1 Students: Over TCP, the allowed services (ports) are: SSH (22), HTTP (80), SMTP (25), DNS (53), IMAP (143, 220, 993), POP (110, 995) and HTTPS (443). Over UDP, we allow the DNS (53). This list can easily be extended if needed; we will restrict ourself to these services for this project. By the principle of white-listing, as the students don't have good reasons to run servers, we disallowed this. Teachers and machines may host freely, for academic purpose.

2 Teachers, Researchers: The same restrictions as those imposed to the students are set-up; except that a teacher can host services.

3 Other devices: Specific ports will be available for the other devices connected like cameras (let's say that it is configured to sends data over TCP on port 2100), research devices (let's say for this project that they use TCP on the ports in the range 2200 - 2299). We allow DNS (53 UDP/TCP). We also allow SIP (TCP and UDP, both on ports 5060 & 5061), RTP and RTCP (UDP 16384-32767) for voice over ip.

4 Servers: The same restrictions as those imposed to the students are set-up; but a server can host services.

Finally, we set-up rules to let the users outside our network send DNS requests (UDP/TCP, port 53), connect to our websites (TCP 80 & 443), send email to a user inside our network (TCP 25) and connect remotely to (secured) internal hosts through ssh (TCP 22).

9.2. Possible security improvements

With more time we could deploy other security elements. For example we could set up an intrusion detection system like `snort`[11] that inspects the network flows to try to detect and block dos attacks and intrusions. We could also use a software that blocks users that have made an important amount of failed tries to connect to an host through ssh like `fail2ban`[12].

10. Testing

We have written test suites for each part of this project. To launch the test, you can open a terminal in the folder LINGI2142-setup, build the project (with the command `make`), wait a minute to let the network converge and finally launch the command
`util/run_tests.sh [target]`
Where `[target]` can be one of the following: `services`, `addressing-dns`, `qos`, `routing`, `addressing-getns`, `firewall`. You can also specify the target `all` to launch all these tests one after the other. In this section we will describe for each part of this project what are the testing strategies we used.

10.1. Services [Olivier]

HTTP The HTTP test tests that all end users can access the website www.website.group1.ingi on port 80. It also tests the accessibility of the website from outside the network with the node TEST. After, the test stops the HTTP server on node HTT1 and check all end users can still access www.website.group1.ingi.

SSH Checks that the administrator QHUN can open an SSH connection on port 22 to all core routers and all HTTP servers. To automate tests, we only check if we can get the banner of the corresponding machine when we run ssh because performing a full connection require to enter manually a password for each key.

SNMP It tests that the administrator QHUN can perform a SNMP request on all core routers and all L3 switches on port 161 with UDP.

10.2. Addressing-dns [François]

In order to test the correct behaviour of the DNS servers, we make exhaustive tests that try to recover the differ-

[11] `https://www.snort.org/`
[12] `https://www.fail2ban.org`

ent records (AAAA, NS and PTR records) from the name servers. This test is behaviour-based.

We select one node of the topology which should have access to the DNS servers and this node will perform DNS queries, without looking at which server the query is sent. This test is performed on a node that belongs to our internal network and on a node belonging to the outside (TEST) in order to verify that sensitive informations (hidden AAAA or PTR records) are not reachable from the outside. In order to verify failures, the test proposes to the user to choose a DNS server (DNS1 or DNS2) to bring down to verify that names of our domain are still resolvable.

By doing tests on nodes inside and outside of the network, we also verify the reachability of our DNS from the outside of our network and the use of Anycast addresses inside of our network.

10.3. Addressing-getns [François]

These tests are here to ensure that the different end-users are able to get the IP addresses of our name servers. We will thus check that the IP addresses are present in the /etc/resolv.conf, for hosts relying on DHCPv6 and for hosts relying on RDNSS extension of router advertisements.

10.4. Qos [Robin]

In order to test the quality of the different services running in the network, we will use the iperf tool, as its purpose is to measure the maximum rate that it can perform between two nodes, with a possibility to set several parameters such as source port and destination port. A special node has been created for this test: QOS. Its purpose is to make the tests easier to perform. Indeed, the QOS node has no restriction on the services that it allows (it can be both an HTTP server and a phone, for example). All services are requested from INT1 for the VoIP and ALEX for all the other.

We perform successively 4 tests :

1. Only backup
2. Backup + HTTP
3. Backup + HTTP + SSH
4. Backup + HTTP + SSH + VoIP

The result tables contain the expected rate for each service launched by an iperf, and the actual measured rate. These two results should be as close as possible. Since the test results can vary, we do not set a boolean that indicates if the test pass or fail, since it would not be relevant to set an exact correct value.

10.5. Routing [Alexis]

To check the behavior of the routing part of the network, we decided to focus on reachability. Instead of checking all the expected hops of a message, we decided to send pings, from many sources to many destinations, only to check if they are reachable.

After checking the allowed traffic from the firewall part, we decided to perform the following tests:

- Communicating between two end users
- Establishing a communication from a host to a service
- Checking the response of each router, from the network administrator
- Finally, pinging google to check internet connectivity

Then, we propose the user to disable any router (excepted CARN, which is in fact non really redundant in our implementation because we did not added the double link with MICH as it is a limitation of our simulation tools that does not allow us to model a switch concretely).

For instance, you can disable PYTH or HALL router in order to check that AS switching is well done.After waiting a bit, the test is repeated to check if the connectivity still holds with a router down.

10.6. Firewall [Rémi]

This test verifies the behavior of the firewalls deployed on the different routers. We did not test all existing protocols and ports to see if they are allowed or denied as it would be very cumbersome. As we use a white-listing approach, we focus our tests on the testing of the rules that we have written and we assume that any packet not matching any rule will be discarded (we still test some of them to verify this default drop policy).

In this test we verify that the allowed services (protocols/ports) are the one we defined in this report. We do this test for each user type as the allowed services depend on the type of user. We also ensure that a student cannot host any service. There are some other tests to verify that ping and traceroute is blocked from outside of the network. Then we verify that the routing packet are not seen on non router-to-router links. Finally we test that any user inside our network cannot change his source ip (spoofing) and that any non-administrator user cannot contact directly a router.

11. Conclusion

To conclude, we can say that this project was a great opportunity to discover network configuration through a real example. We have learned how to set up, tune, troubleshoot and enhance a network, from scratch, using real tools. We have discovered the good and the bad practices, the common pitfalls and security issues network administrators may have to overcome. This was really interesting and instructive. The help of the teaching team was necessary and really useful.

This was challenging too. Of course, we had to face choices. Some of them were difficult: we sometimes had to restrict the scope of the project. Despite all these points, we have finally been able to bring our own solution. We did our best to provide a well design, efficient, extensible, well documented and thoroughly tested solution

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

References

[1] P. Alilovic. Debugging iptables rules with tcpdump and wireshark. https://sgros-students.blogspot.be/2013/05/debugging-iptables-rules-with-tcpdump.html, [Online; accessed 14-March-2017].

[2] F. Baker, C. Bowers, and J. Linkova. Enterprise multihoming using provider-assigned addresses without network prefix translation: Requirements and solution. Internet-Draft draft-bowbakova-rtgwg-enterprise-pa-multihoming-01, IETF Secretariat, October 2016. https://tools.ietf.org/html/draft-ietf-rtgwg-enterprise-pa-multihoming-00.

[3] D. Barr. Common dns operational and configuration errors. RFC 1912, RFC Editor, February 1996. http://www.rfc-editor.org/rfc/rfc1912.txt.

[4] M. Boucadair, A. Petrescu, and F. Baker. Ipv6 prefix length recommendation for forwarding. BCP 198, RFC Editor, July 2015.

[5] T. Chown and S. Venaas. Rogue ipv6 router advertisement problem statement. RFC 6104, RFC Editor, February 2011.

[6] Cisco. Cisco traffic policing and traffic shaping for bandwidth limiting - cisco. http://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-policing/19645-policevsshape.html, [Online; accessed 10-March-2017].

[7] L. Colitti, V. Cerf, S. Cheshire, and D. Schinazi. Host address availability recommendations. BCP 204, RFC Editor, July 2016.

[8] A. Davidson. Lonap's route servers, 2009. https://www.uknof.org.uk/uknof13/Davidson-LONAP_routeservers.pdf, [Online; accessed 8-March-2017].

[9] Debian. Debian firewall. https://wiki.debian.org/DebianFirewall, [Online; accessed 14-March-2017].

[10] M. Devera. tc-htb(8): Hierarchy token bucket - linux man page, 2004. https://linux.die.net/man/8/tc-htb, [Online; accessed 9-March-2017].

[11] R. Droms. Stateless dynamic host configuration protocol (dhcp) service for ipv6. RFC 3736, RFC Editor, April 2004.

[12] I. Gashinsky, J. Jaeggli, and W. Kumari. Operational neighbor discovery problems. RFC 6583, RFC Editor, March 2012.

[13] V. Gite. Linux: 20 iptables examples for new sysadmins. https://www.cyberciti.biz/tips/linux-iptables-examples.html, [Online; accessed 14-March-2017].

[14] J. Jeong, S. Park, L. Beloeil, and S. Madanapalli. Ipv6 router advertisement options for dns configuration. RFC 6106, RFC Editor, November 2010.

[15] A. Kis-Szabo. Man page of ip6tables. http://ipset.netfilter.org/ip6tables.man.html, [Online; accessed 14-March-2017].

[16] M. Kohno, B. Nitzan, R. Bush, Y. Matsuzaki, L. Colitti, and T. Narten. Using 127-bit ipv6 prefixes on inter-router links. RFC 6164, RFC Editor, April 2011.

[17] A. N. Kuznetsov. tc(8): Stochastic fairness queueing - linux man page. https://linux.die.net/man/8/tc-sfq, [Online; accessed 9-March-2017].

[18] A. N. Kuznetsov. tc(8): show/change traffic control settings - linux man page, 1999. https://linux.die.net/man/8/tc-htb, Online; accessed 9-March-2017].

[19] Lighttpd. Lighttpd fly light, 2017. https://www.lighttpd.net, [Online; accessed 8-March-2017].

[20] LinuxManPages. Man page of iptables-extensions. http://ipset.netfilter.org/iptables-extensions.man.html, [Online; accessed 14-March-2017].

[21] P. Mockapetris. Domain names - concepts and facilities. STD 13, RFC Editor, November 1987. http://www.rfc-editor.org/rfc/rfc1034.txt.

[22] NetworkLessons. Qos traffic shaping explained. https://networklessons.com/quality-of-service/qos-traffic-shaping-explained/, [Online; accessed 13-March-2017].

[23] J. Peterson, O. Kolkman, H. Tschofenig, and B. Aboba. Architectural considerations on application features in the dns. RFC 6950, RFC Editor, October 2013.

[24] T. Preston. Bird route server daemon deployment, 2010. https://www.uknof.org.uk/uknof15/Preston-Routeserver.pdf, [Online; accessed 8-March-2017].

[25] Wikipedia. Open shortest path first — Wikipedia, the free encyclopedia, 2004. https://en.wikipedia.org/wiki/Open_Shortest_Path_First, [Online; accessed 19-February-2017].

[26] Wikipedia. Comparison of ipv6 support in operating systems, 2017. http://en.wikipedia.org/w/index.php?title=Comparison%20of%20IPv6%20support%20in%20operating%20systems&oldid=765785922, [Online; accessed 14-March-2017].

[27] Wikipedia. Comparison of IPv6 support in operating systems — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Comparison%20of%20IPv6%20support%20in%20operating%20systems&oldid=765785922, 2017. [Online; accessed 14-March-2017].

[28] T. Ylonen and C. Lonvick. The secure shell (ssh) protocol architecture. RFC 4251, RFC Editor, January 2006. http://www.rfc-editor.org/rfc/rfc4251.txt.

Appendix: Differences between the final submission and the first one

[All] Technology & Topology

Since no negative review or proposal has been made for the technology or routing part, we did not changed anything.

We just included in the README the command to reload the bashrc file after setting up the LANG variables, if needed.

[Alexis] Routing

As the routing part was working in accordance with the requirements of the project, we did not modified this part of the project.

All the remarks about the description of the topology is related to the fact that the source selection is explained partly in the end user management section.

As for this project, we assume that `fd00::d` is reliable and always up, this is not really a problem to associate the connectivity with this server to the connectivity with the entire internet. However, in a practical situation, we should have chosen something more reliable (like a anycasted DNS root server).

Finally, we modified the `get_public_ips` to not include DNS anycast for the tests.

[François & Alexis] End user management

As nothing bad has come up from the reviews, nothing has changed compared to the first version of our work

[François] Naming

Here again, the reviews were quite positive and nothing in the first version of our work needed to be changed.

[Olivier] Services

Reviews suggested using nginx as load balancer. However, we chose to keep our DNS balancing. Each of these two solutions have their advantages: the nginx load balancer avoid problems discussed in the review, but running nginx on an additional machine would be more expensive solution. Since we are in a university with not infinite budget, we think this is nice if we can avoid to place an additional machine in our network. Moreover, the ngnix running on a machine would be a single point of failure: if the ngnix load balancer is down, all our datacenter would be unavailable. With our DNS load balancing, if a HTTP server is down, our network is still working (see our services tests). We could have placed 2 load balancer to assure redundancy, but having 2 load balancer for 3 HTTP servers is not an elegant solution for the same cost reasons explain above. As this was the only improvment suggestion received, the service part is unchanged compared to the previous submission.

[Robin] Quality

The reviews highlight the bad usage of the ceil value, in a way that we should not define a small ceil for cameras or phones for example, since we should allow these devices to take all the bandwidth if this is available. As explained, this is a security matter, considering the context. We consider that the newtork is deployed in a university campus, so the case in which the phones or the camera take all the bandwidth is unlikely. Thus, to protect the network against DoS attack (such as running a ping as detailed in the report), we made the choice to define these conventions in the QoS rules.

In addition to that, we chose not to make differences between students, researchers, teachers, ... in the QoS rules, since a student may have a more important usage than a staff members at a `t` instant, this is subjective. What is objective, it is the identified service, that's all that matters, this is why the QoS focus only on these.

Finally, the HTB algorithm is used because it is the best, in comparison with the other existing ones. It is explained (as the other, like SFQ) in a few words (mostly why not an other algorithm), because explaining them in a well-detailled way was not the point, this would weigh the report in a irrelevant way.

The QoS rules (and the corresponding tests) have not been updated, since we assume the decisions that are contested in the submitted reviews.

[Rémi] Security

No modification has been made since the review received says that everything is working fine and it does not suggest any modification or improvement. The Security part (implementation, tests and report) is thus the same as the one submitted previously on `hotcrp`.