IPv4 to IPv6 Transition Engine

Frédéric Beck, Olivier Festor, Isabelle Chrisment

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

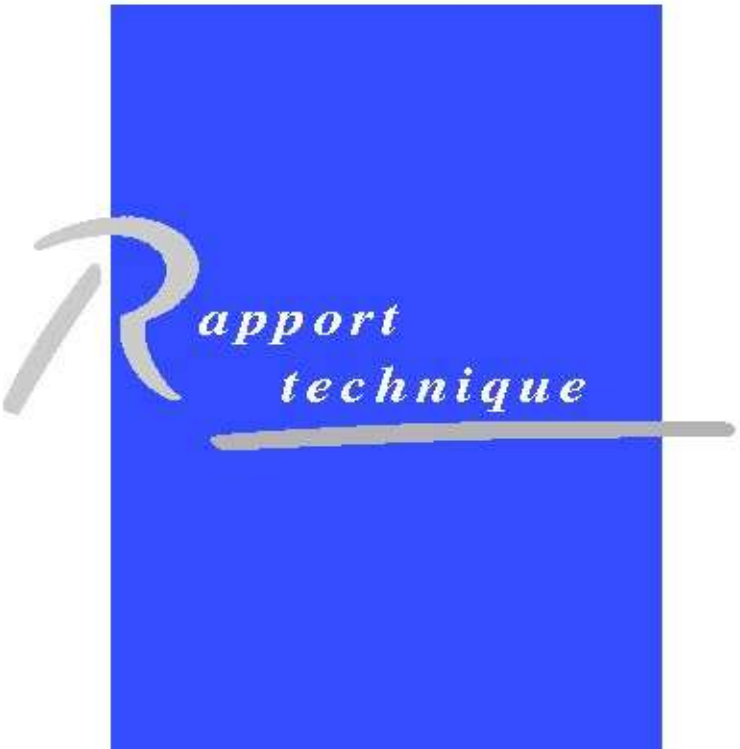
IPv4 to IPv6 Transition Engine

Frédéric Beck, Olivier Festor and Isabelle Chrisment

N° ????

July 2007

Thème COM



ISSN 0249-0803

IPv4 to IPv6 Transition Engine

Frédéric Beck, Olivier Festor and Isabelle Chrisment

Thème COM — Systèmes communicants
Projet MADYNES

Rapport technique n° ???? — July 2007 — 35 pages

Abstract: The transition from IPv4 to IPv6 will be a major issue in the next few years, as IPv6 deployment began. In this report, we present the outcome of a study on this subject. It is composed of an initial addressing algorithm, the tool implementing this algorithm, and a routing configuration generator.

Key-words: IPv4, IPv6, management, transition, network

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,

Outil de Transition IPv4 vers IPv6

Résumé : La transition d'IPv4 vers IPv6 sera une des difficultés majeures à surmonter dans les années à venir. Dans ce rapport, nous présentons les résultats d'une étude menée sur ce sujet. Ils se composent d'un algorithme d'adressage initial, de l'outil qui l'implémente et d'un générateur de configurations de routage.

Mots-clés : IPv4, IPv6, management, transition, réseau

Contents

List of Figures

1 Introduction

IP networks are widely spread and used in many different applications and domains. This kind of networks knew an exponential growth during the nineties, and with the latest technical advances, like wireless networks and devices and mobility issues, this phenomenon will continue. It is now well known that IPv4 has limits which will lead, sooner or later, to the exhaustion of available addresses, and all other issues (routing tables explosion...). Different mechanisms have been defined and deployed to postpone IPv4 death, such as NAT [7], but these are only temporary, and will not solve the problem at a longer term.

To answer these problems, IPv6 has been defined in 1998 [3]. It consist in a new version of the IPv4 protocol, with a bigger address space (the address is not anymore 32 bits long, but 128) and comes along with new built-in services (address autoconfiguration [8], native IPSec and other functionalities, routes aggregation, simplified header...). Many studies have been taking place on the subject, and IPv6 begin its deployment.

However, the transition from IPv4 to IPv6 is a delicate step, and can triggers many problems, in terms of security or service outage for example. Moreover, administrators are not always well prepared, because they don't know well the protocol itself and the related issues. For these reasons, and in the scope of our study on IPv6 network renumbering [1], we decided to define and implement a transition engine to ease the administrators task. Thanks to the experience gained on our previous studies, we aim at providing a "one click transition" tool.

In this paper, we will describe the metric we define, the prefix assignation algorithm and the prototype we developed.

2 IPv6 Initial Addressing

In this chapter, we will present all the IPv6 prefix assignation algorithm, beginning with the general idea, metric and the algorithm itself.

2.1 Overview

This addressing mechanism is based on a logic representation of the network to assign. That means that, for example, Virtual LAN are seen as different links in the graph, even if physically they are multiplexed on the same link. Leaves on that graph are the end-networks containing the end-nodes, or a router with no successor or end-network (present for further extensions of the network). A network may also appear in the middle of the graph, if it make possible the interconnection of more than two routers. Networks interconnecting only two routers are only shown as links. Finally, cycles are allowed in the graph. The router interconnecting with the ISP is the border router, and the root of the graph. If the network is multihomed, it will be represented with two logical views.

An example of such a network is shown in figure 1.

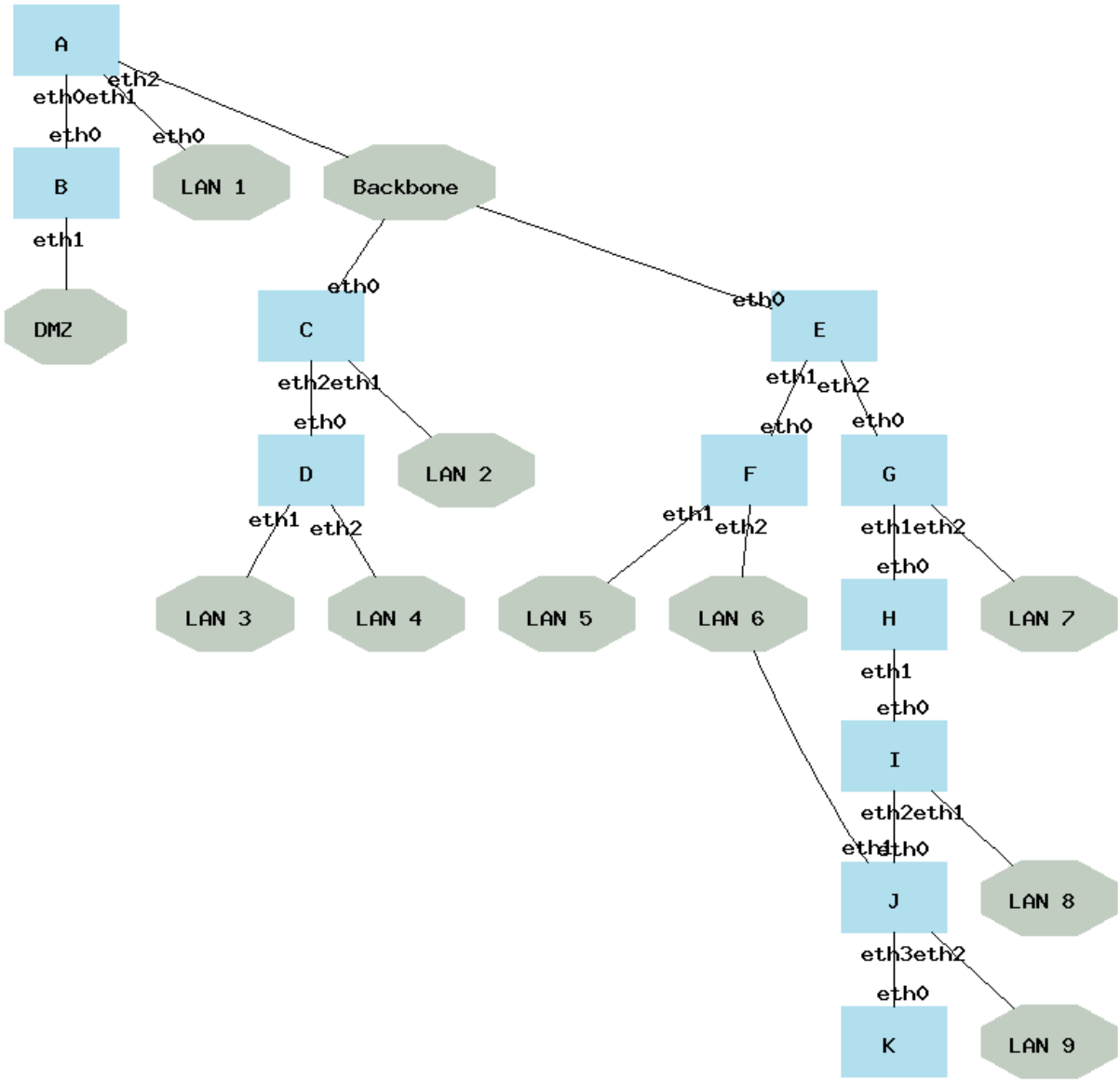


Figure 1: Example of network logical view

Weights are assigned to the links. By default, each link has a weight of 1, but is the source of the link is a network and the destination a router, the weight will be 0, to not interfere with the algorithm calculating the shortest paths to the root. In our example, the link between the *backbone* and the routers *c* and *e*, and the one between *LAN 6* and *j* have a weight of 0.

In this graph, all nodes, network or router, calculate their shortest path to the root. Thanks to that information, and the graph itself, we introduce several terms which are used for this procedure:

previous all the vertices in the graph which are source of a link which destination is the current vertex.

predecessor the router at distance 1 in direction of the root.

next all the vertices in the graph which are destination of a link which source is the current vertex.

successors or children all the routers at a distance of 1 which are not closer to the root

leave a router or network at distance of 1,for which we are the predecessor, and which does not have outgoing links

end-network a network at distance of 1,for which we are the predecessor, and which does not have outgoing links

2.2 Metric

2.2.1 Definition

The metric if for a vertex the number of networks it has under is authority. It is thus the number of /64 networks itself or its children have to assign. It is composed of three components:

local_metric the number of outgoing links or out degree

reserved_metric in order to prevent eventual further network extensions

child_metrics the metrics announced by the successors

The metric M_v for a vertex v is calculated as follows:

$$M_v = \sum_{N=0}^{Nbsuccessors} M_n + L_v + R_v$$

where L_v is the local metric of the vertex v, and R_v its reserved metric. For a leave, as it has no child, the sum equals zero, and the local metric also. The only factor is then the reserved metric, which is zero for an end-network, and can be zero or more for a router.

2.2.2 Propagation

This metric is propagated form the leaves (router or network) to the root. The algorithm it follows is:

$M = local\_metric + reserved\_metric$
for i in $self.child\_metrics$ **do**
  $M = M + M_i$
end
return M

Function `calculate_metric`

if $self$ is border **then**
  stop
end
if $len(self.child\_metrics) \neq nb(self.successors)$ **then**
  wait until $len(self.child\_metrics) == nb(self.successors)$
end
$M_i = calculate\_metric()$
$Pred_i = self.get\_predecessor()$
$Pred_i.child\_metric.append(M_i)$
$Pred_i.announce\_metric()$

Function `announce_metric`

foreach leave i **do**
  i.announce_metric()
end

Algorithm 3: Metric Propagation Algorithm

2.2.3 Future Work

At the moment, this metric only takes into account the logical view of the network. In the future, we will look how we can correlate it with the existing IPv4 network, and for example the length of the network mask.

2.3 Prefix Assignation

The assignation of the prefixes, contrary to the metric propagation, goes from the root to the leaves. The algorithm takes care of the route aggregation and tries to use as less address space as possible. To tune this feature, the *reserved_metric* can be used.

On each router, beginning from the root, we define the needs of each successors, and the local one in a list of couple *(id,metric)*, where id is the vertex id, and metric the metric announced by this vertex. In input, we give the prefix of the site which is delegated to the

root. The root assigns prefixes, from the vertex with the biggest need to the smallest one. Then, the same algorithm is performed on each successor. During the whole process, each vertex maintains locally a list of available prefixes for assignment. If this is not sufficient, a global list is maintained by the root.

Thus, the algorithm is called through the function $assign\_prefixes$:

Input: $site\_prefix = P$
$M = self.calculate\_metric()$
$needs = [M] + self.child\_metrics$
$self.available = [P]$
for $max(needs)$ to $min(needs)$ **do**
  $(R, metric_R) = needs$
  $Nb_{/64} = 64 - \lceil metric_R/2 \rceil$
  $min_{prefix} = get\_match\_prefix\_len(self.available)$
  while $len(min_{prefix}) \neq Nb_{/64}$ **do**
    $self.available.remove(min_{prefix})$
    $self.available.append(min_{prefix}.divide())$
    $min_{prefix} = get\_match\_prefix\_len(self.available)$
  end
  $R.prefix = min_{prefix}$
end
foreach successor R which is a router **do**
  $R.assign\_prefixes()$
end

Algorithm 4: Prefix Assignment Algorithm

where:

get_match_prefix_len gets in the list of available prefixes the one with the smallest mask equal or bigger to the requirements.

divide divides a prefix of length X in two prefixes of length X+1

2.4 Links and End-Points Addressing

Once prefixes are assigned to networks and routers, the router has to assign prefixes on outgoing links, and set the end-points addresses. Firstly, the prefix assigned to the router is divided in /64 networks which will be assigned. Then, link by link, if the destination is a router, we assign a prefix. If the destination is a network, it already has an assigned prefix, we simply assign the same prefix to the link. Then, we assign addresses on end-points, beginning by the interface ID *1*, and incrementing it each time.

This is done as follows:

```
Input: self.prefix
self.local_available = [self.prefix]
min_prefix = get_match_prefix_len(self.local_available)
while len(min_prefix) ≠ 64 do
    self.local_available.remove(min_prefix)
    self.local_available.append(min_prefix.divide())
    min_prefix = get_match_prefix_len(self.local_available)
end
foreach outgoing link L do
    if L.target is a router then
        L.prefix = min_prefix
        self.local_available.remove(min_prefix)
        min_prefix = get_match_prefix_len(self.local_available)
    else
        L.prefix = L.target.prefix
    end
    L.source_addr = L.prefix.get_first_available_address()
    if L.target is a router then
        L.target_addr = L.prefix.get_first_available_address()
    end
end
```

Algorithm 5: Links and End-Points Addressing Algorithm

where:

get_first_available_address returns the first available address for a given prefix

3 Implementation

To validate the addressing algorithm proposed in chapter 2, we developed a prototype using the Python language. This language has been chosen for being Object Oriented, and offering a large set of libraries. The code architecture is shown in figure 2.

This representation is non exhaustive, as only the key objects and attributes/methods are shown. The core of this model is the class *Site*. It stands for the whole network, and is the main class which has to be instantiated in the tool main.

3.1 Representing the network

As we already said, the network is represented as a graph. We decided to use the Dot language from the Graphviz framework [1]. The advantage of this framework, is that is make possible to represent the network with many characteristics, generate easily a graphical view exploitable in WEB or GUI interfaces, and the language itself is really easy. A network is defined as follows:

```
graph G {
        node [shape=box, color=lightblue2, style=filled];
        edge [arrowhead=none];
        chocolat [label="chocolat",comment="border",group="router"];
        kran [label="kran",group="router"];
        lan1 [label="LAN 1",shape=octagon,color=honeydew3,group="network"];
        chocolat -- kran [weight="1.0",taillabel="eth1",headlabel="eth0",dir=b];
        chocolat -- lan1 [weight="1.0",taillabel="eth2",headlabel=""];
        lan2 [label="LAN 2",shape=octagon,color=honeydew3,group="network"];
        kran -- lan2 [weight="1.0",taillabel="eth1",headlabel="eth0"];
}
```

All nodes are bounded to a group, which is either *router* or *network*. The border router, gateway in this example, has an additional comment *border*, and is the root of the graph. Finally, all edges have weights defined as explained in chapter 2, and have tail and head labels, corresponding to the physical interfaces on the routers. The edges do not have comments at the moments, but this attribute will be used for defining VLAN, tunnels or trunks.

Figure 3 presents a graphical representation of this network.

[1] `http://www.graphviz.org`

INRIA

Figure 2: UML Representation

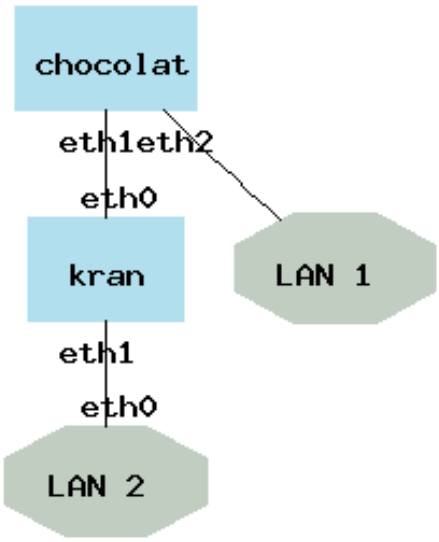


Figure 3: Example of Network Representation with Graphviz

The Dot representation of the network is passed as parameter to the program. It is parsed by using the Boost Graph Library [2] and its python bindings [3], and Python objects are generated accordingly. This library includes several algorithms, especially for shortest path calculation. In our implementation, we chose to use the Bellman-Ford algorithm [4].

Alongside to this dot representation, the program uses an XML file in input, to give more informations about the network. This file can be validated with the following DTD:

```
<!ELEMENT config (site_prefix,bgp?, write_memory?, routers,networks)>

<!ELEMENT site_prefix (#PCDATA)>

<!ELEMENT bgp (as,neighbor*)>
<!ELEMENT as (#PCDATA)>
<!ELEMENT neighbor (#PCDATA)>
<!ATTLIST neighbor as CDATA #REQUIRED>

<!ELEMENT routers (router*)>
<!ELEMENT router (id, border?, quagga?, cisco?, telnet?, ssh?, interfaces)>
```

[2] http://www.boost.org/libs/graph/doc/index.html
[3] http://www.osl.iu.edu/~dgregor/bgl-python/
[4] http://en.wikipedia.org/wiki/Bellman-Ford_algorithm

```
<!ELEMENT id (#PCDATA)>
<!ATTLIST id config (telnet|ssh) "telnet" routing (static|ripng|ospf|bgp) "ripng">
<!ELEMENT border (#PCDATA)>
<!ELEMENT quagga (#PCDATA)>
<!ELEMENT cisco (#PCDATA)>
<!ELEMENT telnet (addr, user?, password, enable_password,generate_zebra?,
                  generate_ripngd?,generate_ospf6d?,generate_bgpd?)>
<!ELEMENT ssh (addr, user, password, enable_password?)>
<!ELEMENT addr (#PCDATA)>
<!ELEMENT password (#PCDATA)>
<!ELEMENT enable_password (#PCDATA)>
<!ELEMENT generate_zebra (#PCDATA)>
<!ELEMENT generate_ripngd (#PCDATA)>
<!ELEMENT generate_ospf6d (#PCDATA)>
<!ELEMENT generate_bgpd (#PCDATA)>
<!ELEMENT user (#PCDATA)>

<!ELEMENT interfaces (interface*)>
<!ELEMENT interface (id, upstream?, mac?, lla?, ipv4_address?, ipv6_addresses?)>
<!ELEMENT upstream (#PCDATA)>
<!ELEMENT mac (#PCDATA)>
<!ELEMENT lla (#PCDATA)>
<!ELEMENT ipv4_address (#PCDATA)>
<!ELEMENT ipv6_addresses (ipv6_address*)>
<!ELEMENT ipv6_address (#PCDATA)>

<!ELEMENT networks (network*)>
<!ELEMENT network (id, autoconf?, dhcpv6?, ipv4_address?, ipv6_prefixes?)>
<!ELEMENT autoconf (#PCDATA)>
<!ELEMENT dhcpv6 (#PCDATA)>
<!ELEMENT ipv6_prefixes (ipv6_prefix*)>
<!ELEMENT ipv6_prefix (#PCDATA)>
```

In this file, we set the prefix the ISP accorded to the site, and we describe the routers and the networks which are present in the Dot representation. An example is given in section 4.1.2.

Later on, this file will also contain the expected behavior of the tool, in terms of tuning of the addressing and configuration process.

3.2 Addressing Algorithm Implementation

The Addressing Algorithm has been implemented by using recursive functions. The metric propagation goes from the leaves to the root, while the address assignation goes from the root to the leaves.

3.3 Remote Configuration

Once all the prefixes and addresses are assigned, the program configures remotely the networks to make IPv6 available on the network. This is done by using Telnet and SSH remote configuration, and is perform on Quagga [5] and Cisco routers. The routing protocol used at the moment is RIPng [5].

3.3.1 SSH

SSH connection is used at the moment only for copying the initial RIPng configuration to the GNU/Linux node running Quagga, and then restart the service. This feature will be improved to support remote configuration on Cisco routers.

SSH connection are implemented by using the pexpect library [6]. The primitives available are:

- scp_to_remote
- scp_from_remote
- start_service
- stop_service
- restart_service

3.3.2 Telnet

Telnet is used for remote configuration of Quagga and Cisco routers. It is implemented by using the Python telnetlib [7]. All basic commands have been implemented: enable, configure terminal, write terminal/memory, end, exit... The primitive *write_read* sends a command to the router and returns the result. It is used for the configuration of the interfaces or the routing protocol.

If the configured router is a Cisco, a single Telnet connection is sufficient. But if the remote node is a Quagga router, the program connects first to the zebra daemon for configuring the addresses, and then reconnects to the ripng (or any other routing protocol) daemon to configure the routes.

3.4 Routing Protocols

In order to ensure routing on an IPv6 network, different routing protocols are available. In this section, we present the ones the tool implements. In order to choose the routing

[5] `http://www.quagga.net/`
[6] `http://pexpect.sourceforge.net/`
[7] `http://docs.python.org/lib/module-telnetlib.html`

protocol used on a router, the attribute *routing* is used in the configuration file, on the tag *id* of the router description.

First of all, a very common solution for small networks with few routers and subnets is the usage of static routes. For each router, we get the neighbors, and set a route to their *routed_prefix*. the default route is set with the predecessor as next hop.

Then, we implement several common routing protocols. The protocols implemented are RIPng [5], OSPFv3 [2] and BGP [6]. We took into account the specificities and possibilities of each protocol and implementation in the routers. The border router always initiates the default route for the site. Some parameters have been arbitrarily chosen, such as the process ID for RIPng and BGP, while some other ones can be configured in the configuration file, like the BGP Autonomous system (AS) or external neighbors.

During the implementation, we encountered several issues, and only one could not be solved. Following an update in the Quagga distribution (version 0.99.7), the statement *default-information originate* which is originating the default route in the AS is not working [8]. The bug is solved in the CVS since the 14th June 2007, but this correction has not yet been integrated in official releases. Older releases of Quagga do not have this problem, 0.99.7 is the only one.

Finally, we ensured than the redistribution between all these protocols is working. By default, all protocols are redistributing static routes and connected networks. If a neighbor is using a different routing protocol, this routing protocol is enabled automatically on the router, and the redistribution commands are added. We also make sure that only one router (the one closer to the border) redistributes the protocol. The chapter 4 is a good example for this protocol redistribution.

4 Example

In this chapter, we will present a running example of the prototype on our testbed. We will present all configuration issues, and the output produced by the program.

4.1 Initial Network

Figure 4 presents the network before the transition to IPv6, which will lead to a dual stack network.

This network is using the RIP routing protocol, and all routers are running the Quagga routing software. The IPv6 prefix delegated by the ISP to this network is *2001:660:4501:3200::/56*.

4.1.1 Dot

This network representation has to be translated in the Dot language. This representation will represent the network graph used by the program. The router *Chocolat* is the border router.

[8] `http://bugzilla.quagga.net/show_bug.cgi?id=370`

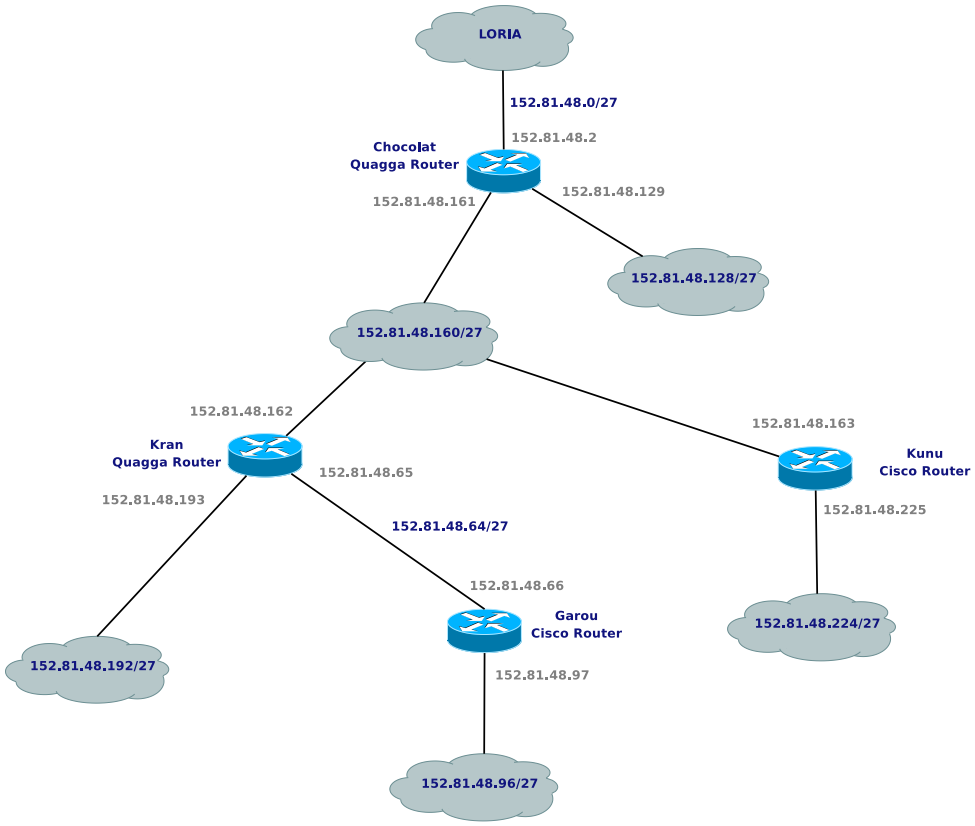


Figure 4: Initial IPv4 Network

The Dot representation is:

```
graph G {
node [shape=box, color=lightblue2, style=filled];
edge [arrowhead=none];

chocolat [label="chocolat",comment="border",group="router"];
kran [label="kran",group="router"];
kunu [label="kunu",group="router"];
garou [label="garou",group="router"];

lan1 [label="LAN 1",shape=octagon,color=honeydew3,group="network"];
lan2 [label="LAN 2",shape=octagon,color=honeydew3,group="network"];
lan3 [label="LAN 3",shape=octagon,color=honeydew3,group="network"];
lan5 [label="LAN 5",shape=octagon,color=honeydew3,group="network"];
backbone [label="backbone",shape=octagon,color=honeydew3,group="network"];

chocolat -- lan1 [weight="1.0",taillabel="eth2",headlabel=""];

chocolat -- backbone  [weight="1.0",taillabel="eth1",headlabel=""];
backbone -- kran [weight="0.0",taillabel="",headlabel="eth0"];
kran -- lan2 [weight="1.0",taillabel="eth1",headlabel=""];

backbone -- kunu  [weight="0.0",taillabel="",headlabel="Gi0/0"];
kunu -- lan3 [weight="1.0",taillabel="Gi0/1",headlabel=""];

kran -- garou [weight="1.0",taillabel="eth2",headlabel="Gi0/0"];
garou -- lan5 [weight="1.0",taillabel="Gi0/1",headlabel=""];
}
```

4.1.2 XML Configuration

The information contained in the Dot description not being sufficient, the network representation is completed by an XML file. In this file, each network component is described more precisely.

Firstly, we have some basic informations about the network in general, and the expected behavior from the tool. In our case, sole the IPv6 prefix assigned to the network, and a flag for saving the generated configurations are set:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<config>
  <site_prefix>2001:660:4501:3200::/56</site_prefix>
  <write_memory/>
  <bgp>
    <as>1664</as>
    <neighbor as="1773">2001:660:4501:32::1</neighbor>
```

```
  </bgp>
  <...>
</config>
```

Then, the routers are described. these declarations include all information for remote configuration, and description of local interfaces:

chocolat, the border router.

```
<router>
  <id config="telnet" routing="bgp">chocolat</id>
  <border/>
  <quagga/>
  <telnet>
    <addr>152.81.48.2</addr>
    <password>dfgdfg</password>
    <enable_password>dfgdfg</enable_password>
    <generate_bgpd/>
  </telnet>
  <ssh>
    <addr>152.81.48.2</addr>
    <user>root</user>
    <password>dfgdfg</password>
  </ssh>
  <interfaces>
    <interface>
      <id>eth0</id>
      <upstream/>
      <mac>00:01:02:E3:60:8A</mac>
      <lla>fe80::201:2ff:fee3:608a</lla>
      <ipv4_address>152.81.48.2</ipv4_address>
      <ipv6_addresses>
        <ipv6_address>2001:660:4501:32::2</ipv6_address>
      </ipv6_addresses>
    </interface>
    <interface>
      <id>eth1</id>
      <mac>00:11:11:20:3A:9E</mac>
      <lla>fe80::211:11ff:fe20:3a9e</lla>
      <ipv4_address>152.81.48.161</ipv4_address>
    </interface>
    <interface>
      <id>eth2</id>
      <mac>00:10:4B:CD:E2:99</mac>
      <lla>fe80::210:4bff:fecd:e299</lla>
      <ipv4_address>152.81.48.129</ipv4_address>
    </interface>
  </interfaces>
```

```
</router>
```

kran

```
<router>
  <id config="telnet" routing="bgp">kran</id>
  <quagga/>
  <telnet>
    <addr>152.81.48.162</addr>
    <password>dfgdfg</password>
    <enable_password>dfgdfg</enable_password>
    <generate_bgpd/>
  </telnet>
  <ssh>
    <addr>152.81.48.162</addr>
    <user>root</user>
    <password>dfgdfg</password>
  </ssh>
  <interfaces>
    <interface>
      <id>eth0</id>
      <mac>00:02:A5:8F:A6:F3</mac>
      <lla>fe80::202:a5ff:fe8f:a6f3</lla>
      <ipv4_address>152.81.48.162</ipv4_address>
    </interface>
    <interface>
      <id>eth1</id>
      <mac>00:50:DA:E2:DB:05</mac>
      <lla>fe80::250:daff:fee2:db05</lla>
      <ipv4_address>152.81.48.193</ipv4_address>
    </interface>
    <interface>
      <id>eth2</id>
      <mac>00:60:08:50:CB:E9</mac>
      <lla>fe80::260:8ff:fe50:cbe9</lla>
      <ipv4_address>152.81.48.65</ipv4_address>
    </interface>
  </interfaces>
</router>
```

kunu

```
<router>
  <id config="telnet" routing="ripng">kunu</id>
  <cisco/>
  <telnet>
    <addr>152.81.48.163</addr>
    <user>madynes</user>
    <password>dfgdfg</password>
```

```
    <enable_password>dfgdfg</enable_password>
  </telnet>
  <ssh>
    <addr>152.81.48.163</addr>
    <user>madynes</user>
    <password>dfgdfg</password>
    <enable_password>dfgdfg</enable_password>
  </ssh>
  <interfaces>
    <interface>
      <id>Gi0/0</id>
      <mac> 00:1a:e2:5d:ef:d8</mac>
      <ipv4_address>152.81.48.163</ipv4_address>
    </interface>
    <interface>
      <id>Gi0/1</id>
      <mac>00:1a:e2:5d:ef:d9</mac>
      <ipv4_address>152.81.48.225</ipv4_address>
    </interface>
  </interfaces>
</router>
```

garou

```
<router>
  <id config="ssh" routing="ospf">garou</id>
  <cisco/>
  <telnet>
    <addr>152.81.48.66</addr>
    <user>madynes</user>
    <password>dfgdfg</password>
    <enable_password>dfgdfg</enable_password>
  </telnet>
  <ssh>
    <addr>152.81.48.66</addr>
    <user>madynes</user>
    <password>dfgdfg</password>
    <enable_password>dfgdfg</enable_password>
  </ssh>
  <interfaces>
    <interface>
      <id>Gi0/0</id>
      <mac>00:1b:2a:eb:fd:48</mac>
      <ipv4_address>152.81.48.66</ipv4_address>
    </interface>
    <interface>
      <id>Gi0/1</id>
      <mac>00:1b:2a:eb:fd:49</mac>
      <ipv4_address>152.81.48.97</ipv4_address>
    </interface>
  </interfaces>
</router>
```

Finally, the networks are defined. They possess a flag indicating if they will be configured with IPv6 Address Autoconfiguration [8] or DHCPv6 [4]:

```
<networks>
  <network>
    <id>backbone</id>
    <autoconf/>
    <ipv4_address>152.81.48.160/27</ipv4_address>
  </network>
  <network>
    <id>lan1</id>
    <autoconf/>
    <ipv4_address>152.81.48.128/27</ipv4_address>
  </network>
  <network>
    <id>lan2</id>
    <autoconf/>
```

```
      <ipv4_address>152.81.48.192/27</ipv4_address>
    </network>
    <network>
      <id>lan3</id>
      <autoconf/>
      <ipv4_address>152.81.48.224/27</ipv4_address>
    </network>
    <network>
      <id>lan5</id>
      <autoconf/>
      <ipv4_address>152.81.48.96/27</ipv4_address>
    </network>
  </networks>
```

4.2 Running the Tool

To run the tool on this network, it must be launched by passing the Dot file as parameter:

```
python main.py testbed.dot
```

4.3 Results

After executed, the program has configured the network and enabled IPv6. In this section, we will present the configuration of the network.

4.4 Dot Representation

First of all, the program regenerates a Dot file representing the IPv6 network addressed. It keeps the same logical view than the one given in input, and shows the addresses and prefixes assigned.

For our test network, the output is shown in figure 5.

The Graphviz framework make possible to attach a URL to each component of the graph, and generates the according HTML maps. This feature may be used for the development of a WEB or GUI interface later on.

4.5 XML Representation

Secondly, the tool generates a new XML file including the new addresses and prefixes assigned. This file describes all the informations about the addressing of the network components.

On our test network, the generated file is:

```
<site>
  <site_prefix>2001:660:4501:3200::/56</site_prefix>
  <routers>
```

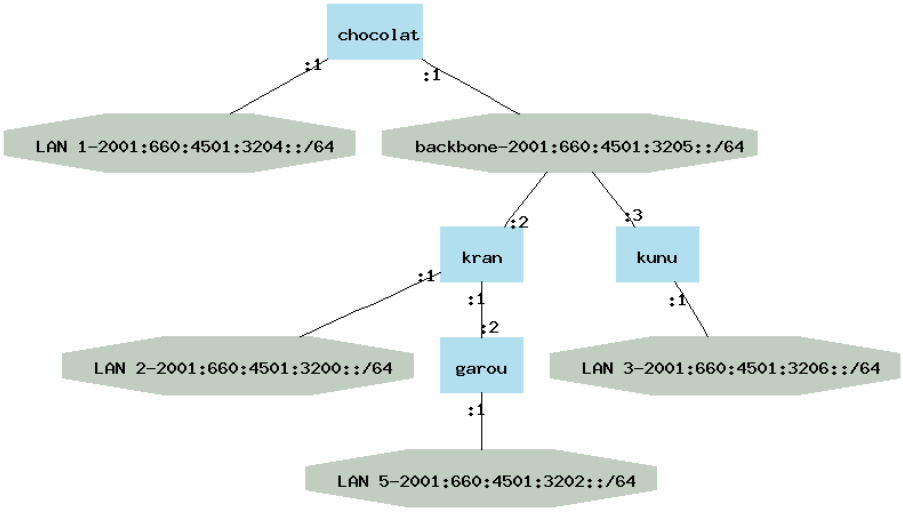


Figure 5: IPv6 Network after Transition

```
<router>
  <id>chocolat</id>
  <border/>
  <interfaces>
    <interface>
      <id>eth2</id>
      <mac>00:10:4B:CD:E2:99</mac>
      <lla>fe80::210:4bff:fecd:e299</lla>
      <ipv4_address>152.81.48.129</ipv4_address>
      <ipv6_addresses>
        <ipv6_address>2001:660:4501:3204::1</ipv6_address>
      </ipv6_addresses>
    </interface>
    <interface>
      <id>eth1</id>
      <mac>00:11:11:20:3A:9E</mac>
      <lla>fe80::211:11ff:fe20:3a9e</lla>
      <ipv4_address>152.81.48.161</ipv4_address>
      <ipv6_addresses>
        <ipv6_address>2001:660:4501:3205::1</ipv6_address>
      </ipv6_addresses>
    </interface>
    <interface>
      <id>eth0</id>
```

```
        <mac>00:01:02:E3:60:8A</mac>
        <lla>fe80::201:2ff:fee3:608a</lla>
        <ipv4_address>152.81.48.2</ipv4_address>
        <ipv6_addresses>
          <ipv6_address>2001:660:4501:32::2</ipv6_address>
        </ipv6_addresses>
      </interface>
    </interfaces>
  </router>
  <router>
    <id>kran</id>
    <interfaces>
      <interface>
        <id>eth0</id>
        <mac>00:02:A5:8F:A6:F3</mac>
        <lla>fe80::202:a5ff:fe8f:a6f3</lla>
        <ipv4_address>152.81.48.162</ipv4_address>
        <ipv6_addresses>
          <ipv6_address>2001:660:4501:3205::2</ipv6_address>
        </ipv6_addresses>
      </interface>
      <interface>
        <id>eth1</id>
        <mac>00:50:DA:E2:DB:05</mac>
        <lla>fe80::250:daff:fee2:db05</lla>
        <ipv4_address>152.81.48.193</ipv4_address>
        <ipv6_addresses>
          <ipv6_address>2001:660:4501:3200::1</ipv6_address>
        </ipv6_addresses>
      </interface>
      <interface>
        <id>eth2</id>
        <mac>00:60:08:50:CB:E9</mac>
        <lla>fe80::260:8ff:fe50:cbe9</lla>
        <ipv4_address>152.81.48.65</ipv4_address>
        <ipv6_addresses>
          <ipv6_address>2001:660:4501:3201::1</ipv6_address>
        </ipv6_addresses>
      </interface>
    </interfaces>
  </router>
  <router>
    <id>kunu</id>
    <interfaces>
      <interface>
        <id>Gi0/0</id>
```

```
        <mac>00:1a:e2:5d:ef:d8</mac>
        <ipv4_address>152.81.48.163</ipv4_address>
        <ipv6_addresses>
          <ipv6_address>2001:660:4501:3205::3</ipv6_address>
        </ipv6_addresses>
      </interface>
      <interface>
        <id>Gi0/1</id>
        <mac>00:1a:e2:5d:ef:d9</mac>
        <ipv4_address>152.81.48.225</ipv4_address>
        <ipv6_addresses>
          <ipv6_address>2001:660:4501:3206::1</ipv6_address>
        </ipv6_addresses>
      </interface>
    </interfaces>
  </router>
  <router>
    <id>garou</id>
    <interfaces>
      <interface>
        <id>Gi0/0</id>
        <mac>00:1b:2a:eb:fd:48</mac>
        <ipv4_address>152.81.48.66</ipv4_address>
        <ipv6_addresses>
          <ipv6_address>2001:660:4501:3201::2</ipv6_address>
        </ipv6_addresses>
      </interface>
      <interface>
        <id>Gi0/1</id>
        <mac>00:1b:2a:eb:fd:49</mac>
        <ipv4_address>152.81.48.97</ipv4_address>
        <ipv6_addresses>
          <ipv6_address>2001:660:4501:3202::1</ipv6_address>
        </ipv6_addresses>
      </interface>
    </interfaces>
  </router>
</routers>
<networks>
  <network>
    <id>lan1</id>
    <autoconf/>
    <ipv4_address>152.81.48.128/27</ipv4_address>
    <ipv6_prefixes>
      <ipv6_prefix>2001:660:4501:3204::/64</ipv6_prefix>
    </ipv6_prefixes>
```

```
    </network>
    <network>
      <id>lan2</id>
      <autoconf/>
      <ipv4_address>152.81.48.192/27</ipv4_address>
      <ipv6_prefixes>
        <ipv6_prefix>2001:660:4501:3200::/64</ipv6_prefix>
      </ipv6_prefixes>
    </network>
    <network>
      <id>lan3</id>
      <autoconf/>
      <ipv4_address>152.81.48.224/27</ipv4_address>
      <ipv6_prefixes>
        <ipv6_prefix>2001:660:4501:3206::/64</ipv6_prefix>
      </ipv6_prefixes>
    </network>
    <network>
      <id>lan5</id>
      <autoconf/>
      <ipv4_address>152.81.48.96/27</ipv4_address>
      <ipv6_prefixes>
        <ipv6_prefix>2001:660:4501:3202::/64</ipv6_prefix>
      </ipv6_prefixes>
    </network>
    <network>
      <id>backbone</id>
      <autoconf/>
      <ipv4_address>152.81.48.160/27</ipv4_address>
      <ipv6_prefixes>
        <ipv6_prefix>2001:660:4501:3205::/64</ipv6_prefix>
      </ipv6_prefixes>
    </network>
  </networks>
</site>
```

4.6 Remote Configuration

Finally, the routers on the network are remotely configured by Telnet. In this section, we will present the configuration of the two routers on the test network. As both of them are Quagga routers, the configuration is separated in two instances, Zebra for the addressing, and RIPng for the routing.

As it is only possible to connect by Telnet to Zebra or RIPngd on the remote host is the process is running, an initial and minimal configuration file is generated, remotely copied

by SCP, and the service is started via SSH. The configuration file generated is, for Zebra on chocolat (only the log file name changes for other daemons):

```
hostname chocolat
password dfgdfg
enable password dfgdfg
log file /var/log/quagga/zebra.log
```

4.6.1 Chocolat

Chocolat is the border router. It has three interfaces, and is the gateway for two of them. The zebra configuration is the following:

```
!
hostname chocolat
password dfgdfg
enable password dfgdfg
log file /var/log/quagga/zebra.log
!
interface eth0
 ipv6 address 2001:660:4501:32::2/64
 ipv6 nd suppress-ra
!
interface eth1
 ipv6 address 2001:660:4501:3205::1/64
 no ipv6 nd suppress-ra
 ipv6 nd ra-interval 30
 ipv6 nd prefix 2001:660:4501:3205::/64 router-address
!
interface eth2
 ipv6 address 2001:660:4501:3204::1/64
 no ipv6 nd suppress-ra
 ipv6 nd ra-interval 30
 ipv6 nd prefix 2001:660:4501:3204::/64 router-address
!
interface lo
!
ip forwarding
ipv6 forwarding
!
line vty
!
end
```

The routing is set in the configuration of the protocol BGP:

```
!
hostname chocolat
password dfgdfg
enable password dfgdfg
log file /var/log/quagga/bgpd.log
!
router bgp 1664
 bgp router-id 152.81.48.2
 neighbor 2001:660:4501:32::1 remote-as 1773
 neighbor 2001:660:4501:3205::2 remote-as 1664
 neighbor 2001:660:4501:3205::2 default-originate
!
 address-family ipv6
 network 2001:660:4501:3200::/56
 network 2001:660:4501:3200::/62
 redistribute kernel
 redistribute connected
 redistribute static
 redistribute ripng
 neighbor 2001:660:4501:32::1 activate
 neighbor 2001:660:4501:3205::2 activate
 exit-address-family
!
line vty
!
end
```

The router chocolat has the router kunu as neighbor. This router is running the RIPng protocol, and chocolat is its predecessor. Thus, the redistribution of RIPng in BGP is performed by chocolat with the statement *redistribute ripng*. It must run a RIPng instance, and also redistribute the BGP routes in RIPng, so that kunu can have all the routes. It has two BGP peering, with kran and with an external router.

The RIPng configuration is:

```
!
hostname chocolat
password dfgdfg
enable password dfgdfg
log file /var/log/quagga/ripngd.log
!
router ripng
 network 2001:660:4501:3200::/56
 network 2001:660:4501:3204::/64
 network 2001:660:4501:3205::/64
 redistribute kernel
 redistribute connected
```

```
 redistribute static
 redistribute bgp
!
line vty
!
end
```

4.6.2 Kran

Kran is a Quagga router with three interfaces and one network for which it acts as the gateway. Zebra is thus configured with:

```
!
hostname kran
password dfgdfg
enable password dfgdfg
log file /var/log/quagga/zebra.log
!
interface eth0
 ipv6 address 2001:660:4501:3205::2/64
 ipv6 nd suppress-ra
!
interface eth1
 ipv6 address 2001:660:4501:3200::1/64
 no ipv6 nd suppress-ra
 ipv6 nd ra-interval 30
 ipv6 nd prefix 2001:660:4501:3200::/64 router-address
!
interface eth2
 ipv6 address 2001:660:4501:3201::1/64
 no ipv6 nd suppress-ra
 ipv6 nd ra-interval 30
 ipv6 nd prefix 2001:660:4501:3201::/64 router-address
!
interface lo
!
interface pimreg
 ipv6 nd suppress-ra
!
ip forwarding
ipv6 forwarding
!
line vty
!
end
```

And we have for BGP:

```
!
hostname kran
password dfgdfg
enable password dfgdfg
log file /var/log/quagga/bgpd.log
!
router bgp 1664
 bgp router-id 152.81.48.162
 neighbor 2001:660:4501:3205::1 remote-as 1664
!
 address-family ipv6
 network 2001:660:4501:3200::/62
 redistribute kernel
 redistribute connected
 redistribute static
 redistribute ospf6
 neighbor 2001:660:4501:3205::1 activate
 exit-address-family
!
line vty
!
end
```

Kran has the router garou as neighbor. This router is using the OSPF protocol, and kran is its predecessor. Thus, the redistribution of OSPF in BGP is performed by kran with the statement *redistribute ospf6*. It must run an OSPF instance, and also redistribute the BGP routes in OSPF, so that garou can have all the routes. It has sole one BGP peering, with chocolat, the other BGP router in the network.

The OSPF configuration is:

```
!
hostname kran
password dfgdfg
enable password dfgdfg
log file /var/log/quagga/ospf6d.log
!
debug ospf6 lsa unknown
!
interface eth0
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0
```

```
!
interface eth1
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0
!
interface eth2
 ipv6 ospf6 cost 1
 ipv6 ospf6 hello-interval 10
 ipv6 ospf6 dead-interval 40
 ipv6 ospf6 retransmit-interval 5
 ipv6 ospf6 priority 1
 ipv6 ospf6 transmit-delay 1
 ipv6 ospf6 instance-id 0
!
router ospf6
 router-id 152.81.48.162
 redistribute kernel
 redistribute connected
 redistribute static
 redistribute bgp
 interface eth0 area 152.81.48.2
 interface eth1 area 152.81.48.2
 interface eth2 area 152.81.48.2
!
line vty
!
end
```

4.6.3 Kunu

Kunu is a Cisco router with two interfaces and one network for which it acts as the gateway. Its configuration is:

```
!
interface GigabitEthernet0/0
 description Configured by Telnet...
 ip address dhcp
 duplex auto
 speed auto
 ipv6 address 2001:660:4501:3205::3/64
 ipv6 enable
```

```
 ipv6 rip ripng enable
!
interface GigabitEthernet0/1
 ip address 152.81.48.225 255.255.255.224
 ip flow ingress
 ip flow egress
 duplex auto
 speed auto
 ipv6 address 2001:660:4501:3206::1/64
 ipv6 enable
 ipv6 nd prefix 2001:660:4501:3206::/64 2592000 604800
 ipv6 nd ra interval 30
 ipv6 rip ripng enable
!
ipv6 router rip ripng
 redistribute connected
 redistribute static
!
```

As chocolat is redistributing all the routes in RIPng, kunu does not need to run a BGP instance, it only uses RIPng.

4.6.4 Garou

Garou is a Cisco router with two interfaces and one network for which it acts as the gateway. Its configuration is:

```
!
interface GigabitEthernet0/0
 ip address dhcp
 duplex auto
 speed auto
 ipv6 address 2001:660:4501:3201::2/64
 ipv6 enable
 ipv6 ospf authentication null
 ipv6 ospf 1337 area 152.81.48.2
!
interface GigabitEthernet0/1
 ip address 152.81.48.97 255.255.255.224
 duplex auto
 speed auto
 ipv6 address 2001:660:4501:3202::1/64
 ipv6 enable
 ipv6 nd prefix 2001:660:4501:3202::/64 2592000 604800
 ipv6 nd ra interval 30
 ipv6 ospf authentication null
```

```
 ipv6 ospf 1337 area 152.81.48.2
!
ipv6 router ospf 1337
 router-id 152.81.48.66
 log-adjacency-changes
 redistribute connected
 redistribute static
!
```

As it was the case for kunu, garou only runs an OSPF instance as kran is redistributing the routes from BGP.

5 Future Work

The tool is only at a prototype stage, there are still many functionalities and features which must be added.

For the algorithm itself, new data and constraints will be added and taken into account. The existing IPv4 network must be taken into account. We can for example imaging to set by default the *reserved metric* on the routers by using the size of the IPv4 prefix delegated to the router. Another point would be to add constraint about the IPv6 addresses generated. It would be interesting to take as parameter eventual addresses already assigned on a subset of the network, and try to keep them, or force a given /64 prefix on a network, and make sure the other prefixes allocated respect the aggregation.

It is also mandatory to study more precisely the behavior of the tool and the algorithm in cases of multihoming and VLAN, and modify, if required, both the algorithm and the input/output of the program accordingly.

Concerning the tools implementation, the protocol redistribution is working, but can eb optimized by enabling the secondary routing protocols only on the required interfaces (the ones that do the peering with the router).

Previously, we studied the impact of IPv6 Renumbering on the firewalls, and developed an update engine for these. It would be interesting to modify it, and make it support the transition from IPv4 to IPv6. This would involve a study on how the rules would be rewritten, and how the IPv4 policy would be adapted to IPv6. Some other management tools, such as NetSV [9], exist for managing and monitoring IPv6 networks. the output of this tool may be used for configuring these ones, and enable a full IPv6 support and management for a network.

Finally, we are planning to develop a GUI for the tool, which would simplify the generation of the input files generation, and the reading of the output. This tool would make possible to generate the logical view of the network from the physical one by selecting graphically a subset of the network, fill and show all the information about the network components... This GUI could be a regular application with binding to the tool (GTK ?) or a WEB interface piloting the tool through a CGI.

[9] `http://netsv.sf.net`

6 Conclusion

The transition from IPv4 to IPv6 is a very complex operation, and can lead to the death of the network. Many administrators are reluctant to take the step, because they do not know well IPv6, and it seems complicated to deploy. There is thus a real need for a tool making that transition easy, the ideal being a "one click" transition.

In this paper, we presented a first reflection we had on the subject, in terms of IPv6 addresses and prefixes assignation. We presented an addressing algorithm, and a prototype we wrote to perform this operation.

Of course, this is only the first step, as the study is only at its beginning. We will pursue our thinking, and improve both the algorithm and the tool, to ensure it will work for most of the cases.

References

[1] F. Bak, E. Lear, and R. Droms. Procedures for Renumbering an IPv6 Network without a Flag Day. RFC 4192 (Informational), September 2005.

[2] R. Coltun, D. Ferguson, and J. Moy. OSPF for IPv6. RFC 2740 (Proposed Standard), December 1999.

[3] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998.

[4] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 3315 (Proposed Standard), July 2003.

[5] G. Malkin and R. Minnear. RIPng for IPv6. RFC 2080 (Proposed Standard), January 1997.

[6] P. Marques and F. Dupont. Use of BGP-4 Multiprotocol Extensions for IPv6 Inter-Domain Routing. RFC 2545 (Proposed Standard), March 1999.

[7] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), January 2001.

[8] S. Thomson and T. Narten. IPv6 Stateless Address Autoconfiguration. RFC 2462 (Draft Standard), December 1998.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803