Automated and Secure IPv6 Configuration in Enterprise Networks

Frédéric Beck, Olivier Festor, Isabelle Chrisment, Ralph Droms

Automated and Secure IPv6 Configuration in Enterprise Networks

Frederic Beck, Olivier Festor
INRIA Nancy - Grand Est Research Center
CS 20101
54603 Villers-lès-Nancy Cedex, France
Email: {frederic.beck,olivier.festor}@inria.fr

Isabelle Chrisment
LORIA - ESIAL Nancy-Université
Campus Scientifique - BP 239
54506 Vandoeuvre-ls-Nancy Cedex, France
Email: isabelle.chrisment@loria.fr

Ralph Droms
Cisco Systems
200 Beaver Brook Road
Boxborough, MA 01719, US
Email: rdroms@cisco.com

***Abstract*— Over the last decade, IPv6 has established itself as the most mature network protocol for the future Internet. Its recent deployment in core networks of operators, its availability to end customers of multiple ISPs together with the availability of native access to large services like Google assess the increasing penetration of IPv6.**

While its deployment from the inside of the network leading to the edges is successful, the transition remains an issue today for many enterprises which see it as a tedious and error prone task for network administrators.

To fill this gap, we present the necessary algorithms and provide the supporting tools to enable this transition to become automatic. Based on a model of an IPv4 network, we describe the algorithms to build an optimized IPv6 adressing scheme and to automatically generate the adequate security plan as well as the corresponding configurations for the different devices in the network.

I. Introduction

IP networks are widely spread and used in many different applications and domains. Their growth continues at an amazing rate sustained by its high penetration in both the home networks and the mobile markets. Although often postponed thanks to tricks like NAT, the exhaustion of available addresses, and other scale issues like routing tables explosion will occur in a near future.

IPv6 [1] was defined with a bigger address space (128 bits) and comes along with new built-in services (address autoconfiguration [2], native IPSec, routes aggregation, simplified header...). Despite its slow start, IPv6 is today more than ever the most mature network protocol for the future Internet. To faster its acceptance and deployment, it has however to offer capabilities reducing and often eliminating the man in the loop. We are convinced that such features are also required for the evolutionary aspects of an IP network, the transition from IPv4 to IPv6 being an essential one. Many network administrators are indeed reluctant to deploy IPv6 because they do not fully master the protocol itself and because they do not have sufficiently rich algorithmic support to seamlessly manage the transition from their IPv4 networks to IPv6. To address this issue, we investigate, design and aim at implementing a transition framework with the objective of making it self-managed.

The contributions of this paper are :

1) a set of algorithms that automate the generation of the IPv6 addressing scheme for an IPv4 enterprise network that can be enriched with on-the fly administrator constraints;
2) an algorithm that generates the security configuration of firewalls for the newly created IPv6 addressing plan;
3) the description of a fully operational, openly available and extensively tested in real environments transition engine that also propagates on-demand the configuration to the devices.

The structure of the paper is as follows. In Section II, we describe the network modeling we used and the algorithms we defined to enable an automatic addressing and configuration of an IPv6 network. Section III focuses on the security aspects, where the security plan of the new IPv6 network is automatically derived. We evaluate and validate in Section IV our transition engine implementation through various scenarios. Section V reviews related work on IPv6 deployment. In Section VI we conclude this paper.

II. IPv6 Addressing

We address enterprise networks as defined in [3], i.e. networks that have multiple internal links, one or more routers interconnections to one ore more Internet Service Providers (ISP). We assume that the IPv6 network has to be built without direct mapping from IPv4 to IPv6 addressing.

A. Network Model

The network topology is modeled by an oriented graph where a vertex consists of a router or a network (end-user or interconnection). An edge connects either two routers via a point-to-point network or a router to a network. This graph provides a logical view of the network to be deployed. The root of the graph is the border router connected to the IPv6 Internet. There will be as many graphs as border routers that are connected to an ISP. The interconnection between a border router and the IPv6 provider is not considered in our model. We only consider this interconnection from the filtering point of view, as firewall rules will be set at the border to protect the network. Inside the network, the IPv6 connection is seen as native.

To offer the possibility to the administrators to express their needs and requirements, we defined a set of constraints that our algorithms use to define the addressing plan or security policy. These constraints allow an administrator, for example,

to force a prefix on a subnet or link, to force the upstream router in case of multihoming, to reserve some prefixes on a router for network provisioning, or to specify the behavior of a particular subnet (e.g. DMZ or IPv4 NAT). We focus on network devices and do not modify or configure anything on the end hosts.

B. Metric

One of the main issue for addressing is the aggregation of IPv6 prefixes. The generated addressing plan must strictly respect the aggregation and must minimize the number of used prefixes.

To achieve this goal, a metric is calculated on each interface, before being summed at the router level.

The metric is the number of /64 prefixes that a router needs to address the networks under its authority. To respect the aggregation issue, the metric must be a power of 2. Therefore if a router needs k end-networks, it has to announce a metric equal to 2^q so that $2^{q-1} < k \leq 2^q$. Another value of this metric is to enable the direct deduction of the aggregated prefix length. A metric of M involves a prefix length of $64 - \log_2 M$.

We define four types of related metrics :

- *reserved_metric*: enables provisioning of network prefixes at the router level: e.g. if a new interface has to be added on the router
- *reserved_metric_per_interface*: enables provisioning of network prefixes at the interface level: e.g. if a new child router has to be added on an existing interface
- *local_metric_per_interface*: the number of outgoing links or out-degrees for a given interface, i.e. the number of links issued from this interface that we have to address
- *child_metrics*: the metrics announced by the successors, it is a tuple <successor_id, nb_/64_required, interface_to_successor>

This metric is propagated from the vertices without successors we called $leaves$ (end-router or end-network) to the root.

Let $N_{iv} = L_{iv} + R_{iv} + \sum_n^{Nbsuccessors} M_{ni}$ where L_{iv} is the local metric of the interface i of the vertex v, R_{iv} its reserved metric, and M_{ni} the metric announced by the child n via the interface i.

We calculate the announced metric M_{iv} for each interface i of a the vertex v as the closest power of 2 of N_{iv} :

$$M_{iv} = \begin{cases} 1 & \text{if } N_{iv} = 1 \ ; \\ 2^{\lfloor \log_2 (N_{iv}-1) \rfloor + 1} & if N_{iv} > 1. \end{cases} \quad (1)$$

Let $N_v = R_v + \sum_i^{Interfaces} M_{iv}$ with R_v the reserved metric of the vertex v.

The announced metric for the vertex v is then calculated as as the closest power of 2 of N_v :

$$M_v = \begin{cases} 1 & if N_v = 1 \ ; \\ 2^{\lfloor \log_2 (N_v-1) \rfloor + 1} & if N_v > 1. \end{cases} \quad (2)$$

In the example depicted in Figure 1, the metric to advertise for A would be: $M_A = 8$. Indeed $N_A = M_{eth1} + M_{eth2} + R_A = 6$ and the next closest power of 2 is $2^3 = 8$. Therefore,

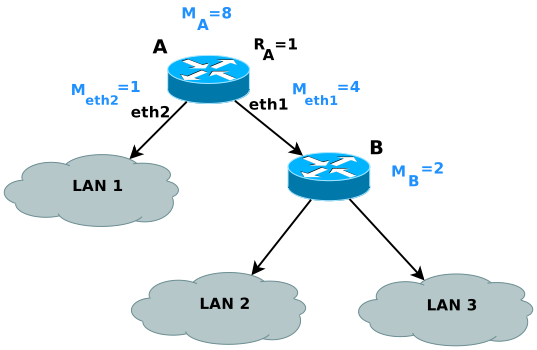

Fig. 1. Propagation example

A will demand a /61, and will be able to assign a /62 on eth1, and a /64 on eth2.

We defined a propagation algorithm whose principle is as follows (cf. Algorithm 1)

Algorithm 1: Metric Propagation Algorithm

```
Algorithm
Routine announce_metric( v:vertex )
Begin
    Mark v
    R_v = get_reserved_metric(v)
    N_v= R_v
    ForEach i Of interfaces of v Do
        N_iv = R_iv + L_iv
        ForEach successor s Of v via interface i Do
            If s not marked Then
                M_s = announce_metric(successor)
                N_iv= N_iv + M_s
            Endif
        EndFor
        M_iv = get_metric_to_adv(N_iv)
        N_v = N_v +M_iv
    EndFor
    M_v = get_metric_to_adv(N_v)
    Return (M_v)
End
```

The metric and its propagation can be summarized in figure 2.

C. Addressing algorithm

The addressing algorithm is then executed from the root to the leaves as described below in the prefix assignment Algorithm 2. As input, we give the prefix of the site which is delegated to the root.

On each router, the per interface needs have already been propagated and calculated during the metric propagation. The need of an interface or a child is, as we wrote in the previous

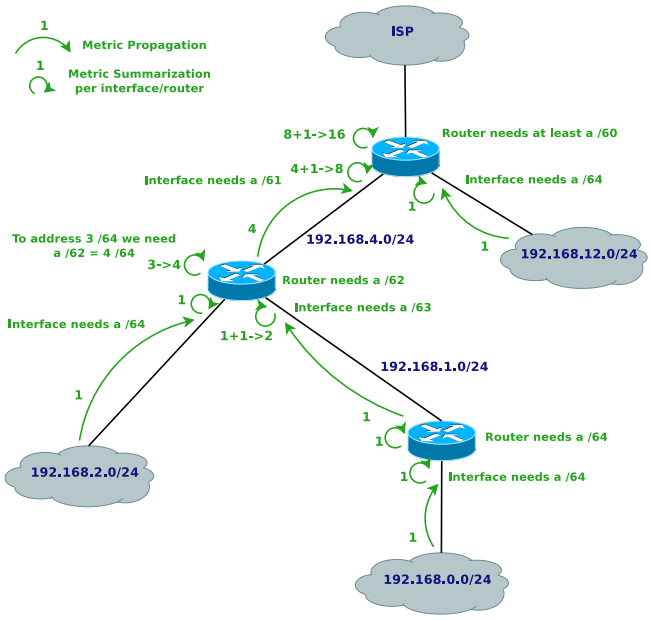

Fig. 2. Metric Propagation Summary

section, a <*id,metric,interface*> tuple, where *id* is the child or interface identifier, *metric* the metric announced by the child or calculated on an interface, and *interface* is the local interface on the router on which the child is connected or null if the tuple stands for an interface need.

When assigning prefixes to its interfaces or children, in the routine *assign_aggregated_prefix()*, each router has already been delegated a prefix matching its needs, which is either the site prefix when the algorithm is running at the border, or a prefix delegated by the parent otherwise.

First, it assigns an aggregated prefix to each of its interfaces. All prefixes that have not been assigned are kept in a list of prefixes globally available at the router level. Namely, if we use the reserved metric at the router level, the reserved prefixes will be kept in that list.

Then, on each interface, prefixes are assigned to each child (router, network or link interconnecting routers) according to the metric it announced. As it was done at the router level, all non-assigned prefixes are stored in a list of available prefixes.

Finally, the router configures addresses on all links end points and calls the *delegate_prefix()* routine to its successors.

Other variables and routines are used :

- $prefix_I$ is the current candidate prefix to be assigned to the interface I.
- $prefix_C$ is the current candidate prefix to be assigned to the child C.
- *divide()*: divides a prefix of length X in two prefixes of length X+1.
- *get_prefix_len*: takes a metric as entry, and returns the prefix length required to fulfill the metric needs.
- *get_match_prefix*: gets in the list of available prefixes the one with the smallest mask equal or bigger to the requirements.

Algorithm 2: Prefix Assignment Algorithm

```
Algorithm
Routine delegate_prefix( v:vertex )
Begin
    Mark v
    assign_aggregated_prefix(v)
    ForEach successor s Of of router v Do
        If s not marked Then
            delegate_prefix(s)
        Endif
    EndFor
End
Routine assign_aggregated_prefix( self:router )
Begin
    ForEach I Of interfaces of self Do
        < I,metric_I,interface_I > = get_needs_interface(I)
        required_length = get_prefix_len(metric_I)
        prefix_I = get_match_prefix(self , metric_I)
        remove_available_prefix(self, prefix_I)
        While len(prefix_I) ≠ required_length Do
            < P_1, P_2 > = divide(prefix_I)
            prefix_I = P_1
            append_available_prefix(self, P_2)
        EndWhile
        assign_prefix_interface(I, prefix_I)
    EndFor
    ForEach C Of children of self Do
        < C,metric_C,interface_C > = get_needs_child(C)
        iface = get_iface(interface_C)
        required_length = get_prefix_len(metric_C)
        prefix_C = get_match_prefix(iface, metric_C)
        remove_available_iface(prefix_C)
        While len(prefix_C) ≠ required_length Do
            < P_1, P_2 > = divide(prefix_C)
            prefix_C = P_1
            append_available_iface(P_2)
        EndWhile
        assign_prefix_child(C, iface, prefix_C)
    EndFor
End
```

The proposed addressing algorithm is illustrated in figure 3.

III. Securing the Network

Defining and deploying the addressing plan as presented in section II is a first step in the automation of the transition. The next step has to define and deploy a security policy matching the resulting IPv6 network. In this section, we detail the security policy of the site. We describe the default behavior and actions to perform, these aspects being configurable in the data model. We also provide some general recommendations to follow when defining such a policy. Then, we describe

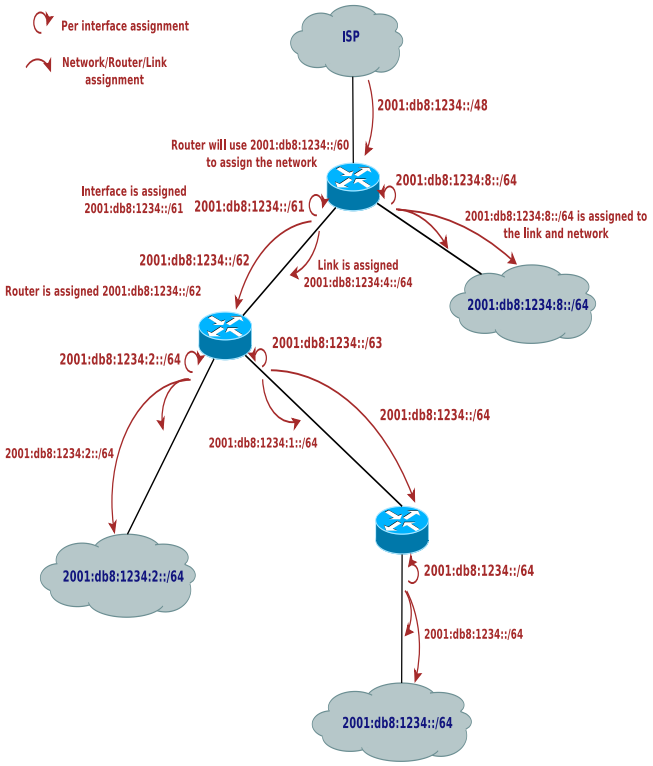

Fig. 3. Addressing Algorithm Summary

some global variables that can be used in both site and subnet levels. Then, we present some site-specific aspects (at the interconnection with the ISP) or to a subnet, before detailing the policy to be applied.

A. General recommendations

In this section, we highlight the general recommendations that should be considered when defining the site global policy:

- All firewalls deployed are stateful firewalls.
- All firewalls deployed have strict Reverse Path Filtering (RPF) enabled.
- Their default policy is to drop all inbound packets that do not match explicit pinholes.
- The outbound default policy can be DROP if the traffic is restricted or ACCEPT if it is unrestricted. If the outbound traffic is restricted, some pinholes need to be defined to allow the desired services to successfully bypass the filtering.
- The pinholes are as restrictive and detailed as possible.
- The minimum default set of permitted services, if the outbound traffic is restricted, are HTTP(S), SSH and FTP.
- If a packet is filtered, the firewall should send an ICMPv6 Destination Unreachable (Administratively Prohibited).
- If a DMZ is present with a DNS server, name resolution is permitted toward the server in that subnet; otherwise it is allowed toward the Internet.
- All Neighbor Discovery Protocol (NDP [4]) packets with a hop limit different from 255 should be dropped.
- All packets with link local source or destination with a hop limit different from 255 should be dropped.
- When addressing the routers, it is recommended not to use an obvious sequential addressing (e.g. ::1, ::2, ::3...). This feature can be configured within the transition engine, so that it is taken into account in the addressing plan. It is possible to use a configured sequential addressing (with a given start and increment), EUI-64 or identifiers following RFC 4941 [5].

Packet inspection and connection tracking is activated as well. As stated in RFC5382 [6], the time-out for TCP established idle connections is 7440 seconds (2 hours 04 minutes), and the transitory time-out is 0. For UDP, the idle time-out is 20 seconds. These rules will be set for inbound and outbound traffic inspection on all interfaces. For Netfilter firewalls, these operations are performed with the *state* module.

As we consider the internal network to be native IPv6, 6to4 and Teredo tunneling are prohibited. Outbound rules deny the usage of such addresses as source, and inbound filtering drops packets to such a destination, while permitting to communicate with nodes using these technologies in the Internet. Prohibiting these two technologies requires IPv4 filtering, by blocking UDP port 3544 and the resolution of `teredo.ipv6.microsoft.com` for Teredo or protocol 41 (IPv6 in IPv4 encapsulation) for 6to4. This can be bypassed by using 6to4 relays within the site. RFC 3964 [7] contains useful information on how to counter this kind of threat.

To simplify the definition of the security policy, we proposed a set of aliases that may be used to identify the source or destination of the filtering rules:

- *internal*: All global or ULA prefixes deployed within the site
- *local*: Only ULA prefixes deployed within the site
- *external*: Global prefixes from partners and providers to allow access to specific services or hosts for business related communications
- *management*: Allowed to perform management operations
- *any*: The default when nothing is specified

These aliases are lists of prefixes and/or hosts that match the description. They can be used for the site global policy or for subnet policies.

In our model, ACLs are set on all firewalls for both inbound and outbound traffic on all interfaces. Thus each interface *IFACE* will have 2 ACLs called *IFACE-IN* and *IFACE-OUT*.

For example, in figure 4, the interface *Gi0/0* has two ACLs defined: *Gi0/0-in* and *Gi0/0-out*. The direction of the inbound and outbound traffic is inverted between the upstream and the other interfaces. In this example, on the upstream interface, outbound traffic is going out on *Gi0/0*, whereas on *Gi0/1* it is coming in.

B. Site-ISP interconnection: bogon prefixes filtering

Protecting the network with traffic filtering is not sufficient. The announces received by the routing neighbors must be filtered as well to protect against advertisement of deprecated or bogus prefixes. This operation is called Bogon Prefixes

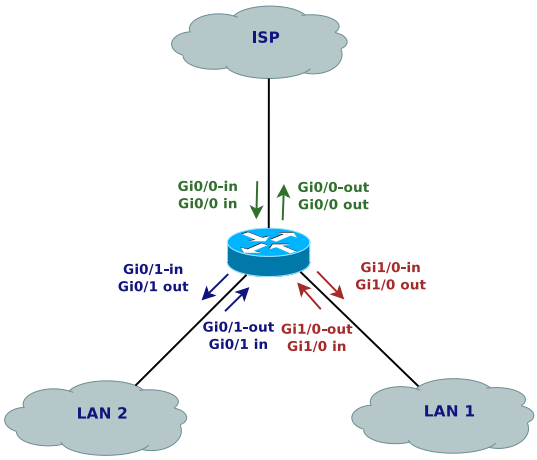

Fig. 4. ACLs Naming

Filtering, and should be enabled for all inbound and outbound routing protocol updates.

We identified the following prefixes that should be filtered in both the inbound and outbound prefix-list (presented in Cisco filtering terminology):

- *::/8 le 128*: IPv4 Compatible and Mapped addresses [1]
- *fe00::/9 le 128*: Link and Site Local Addresses
- *fc00::/7 le 128*: Unique Local Addresses
- *ff00::/8 le 128*: Multicast Addresses
- *2001:db8::/32 le 128*: Document Addresses
- *3ffe::/16 le 128*: 6Bone Addresses
- *2001:10::/28 le 128*: ORCHID

Teredo and 6to4 are special cases. The prefix used for these tunneling mechanisms should be accepted, to enable traffic sent to these hosts, but shorter prefixes should be dropped:

- *Teredo*: permit 2001::/32
  deny 2001::/32 le 128
- *6to4*: permit 2002::/16
  deny 2002::/16 le 128

Finally, the prefixes assigned to the site should be filtered as well. At the inter-connection with the ISP, the prefix must be filtered for inbound announces: *deny* P_{Site} *le 128*. Inside the network, inbound announces of the site prefix must be accepted, to allow routing between the site subnets: *permit* P_{Site} *le 64*. For outbound announces, at the site-ISP inter-connection, we permit the site prefix (*permit* P_{Site}), and within the site, each router is allowed to advertise the prefix it routes: *permit* P_{Router}.

To express the prefixes that are valid, we defined two mode of bogon filtering:

- *relaxed*: permit 2000::/3 le 48; does not require regular update, at least not until all 2000::/3 have been assigned to the Regional Internet Registries (RIRs)
- *strict*: explicitly permit only prefixes assigned to the RIRs by the IANA [2]; requires an update when a new prefix is assigned (last prefix assigned the 13th of May 2008)

By default, the relaxed mode is set.

It is possible to specify in the configuration whether the router should accept announces containing the default route. This may be useful in some cases within the site. By default, even if this feature is activated, it is not set on the inter-connection between the site and the ISP.

In all cases, the default rule is to deny everything that has not been explicitly permitted: deny ::/0 le 128.

C. Subnet templates

Subnets can have different roles in the network. Based on these roles, different filtering rules or behaviors are expected, which are expressed by the following templates:

- *DMZ*: Stateful firewall, deny as default policy for inbound and outbound traffic, pinholes to allow traffic
- *IPv4 NAT*: Stateful firewall, deny as inbound default policy with pinholes to allow traffic, outbound default policy of accept
- *Local*: no Internet access, all non-ULA prefixes are explicitly filtered, limit source and destination addresses of the packets to the internal alias

IPv4 NAT means that the IPv4 subnet migrated to IPv6 was using this technology. In this case, we want to keep the basic security implied by the address translation. Here, we also recommend the usage of Privacy Extensions [5] for that subnet, but as it is host-oriented and does not require any configuration at the network level, we can not do it automatically. Other benefits of NAT can be achieved as stated in RFC4864 [8], but they are hard to automate, and we are not convinced that these benefits are really required in such a case.

For other types of subnets, we do not recommend the usage of Privacy Extensions, and in particular the temporary addresses, as it makes the monitoring and local tracking of hosts complex, even if the benefits at a larger scale have been proven.

Some tools such as NDPMon [3] can help to locally resolve this problem of host tracking if the usage of Privacy Extensions is desired.

A *Local* subnet is restricted to traffic from and to the site. So, all traffic to/from this subnet is denied at the inter-connection between the site and the ISP. It is possible to specify whether the inbound traffic has to be filtered. If it is, the only services that are accessible are the ones that generate pinholes. Such a pinhole is propagated within the site.

Mechanisms such as NAT66 [9] or other IPv6 NAT proposals can be of some interest if they reached the standard status and could be added as another template here, and handled as another constraint in the addressing algorithm.

Of course, it is also possible to define a different policy for a subnet by setting the different parameters explicitly.

[1] *le /128* specifies that the range of prefix lengths that must be matched is from the prefix length argument (/8) to 128

[2] `http://www.iana.org/assignments/ipv6-unicast-address-assignments/`

[3] `http://ndpmon.sf.net`

D. Generic rules

In this section, we define the default filtering rules that should be applied on the border and on each internal firewall. These rules are set in all inbound or outbound ACLs defined in figure 4.

As it has been done for the routing advertisements, we must filter the bogon and deprecated prefixes in the packets themselves (e.g. IPv4 Compatible and Mapped addresses, Site Local addresses...). Moreover, as stated in RFC 5095 [10], an IPv6 packet with Routing Header 0 should be dropped, and other types should be accepted. Traffic from the site to the Internet needs to be specifically permitted, and restricted to the site's prefix as source only. In the same way, traffic from the Internet to the site must be limited to the site's prefix as destination, and dropped if a packet coming from the Internet has the site prefix as source. Reverse Path Filtering must be activated on all interfaces and explicit filtering rules should be set as well.

ICMPv6 packets are filtered as defined in RFC 4890's [11] *relaxed* mode. In *strict* mode, more restrictive filtering is required to protect against the discovery of the internal topology (e.g. echo requests and replies would be denied from the Internet). NDP messages RA/RS and NA/NS are also allowed with FF02::/16 as destination. This permits to receive messages multicasted to the all-nodes, all-routers... addresses. NDP NS with undefined source is accepted as well in input in the scope of the DAD procedure. This rule must be set before the anti-spoofing rules which deny the usage of the undefined address.

Packets with a multicast address as source should be dropped in any case. Moreover, special attention must be brought to the scope of the packets: no packet must be forwarded beyond its scope. To match this rule, four scenarios must be considered. At the site boundaries, only multicast packets which have a multicast destination with global scope should be forwarded. Within the site, only packet with global, site and organization scopes must be forwarded. Packets directed to the firewall itself must be accepted from any source for global scoped packets, from a prefix deployed within the site for site and organization scopes, and from an on-link prefix for the link local scope. Finally, packets emitted by the firewall are allowed to any multicast scope as long as it issued with a source address corresponding to an address set on the outgoing interface.

Incoming traffic to the firewall should be strictly restricted. Besides ICMPv6, bogus and multicast filtering already presented , all traffic should be dropped. The only traffic that should be permitted is management traffic. This includes SNMP, NTP, Telnet or SSH for remote configuration, FTP or TFTP for configuration up/download, ... This traffic should be restricted to explicit IPv6 addresses, i.e. addresses of the hosts belonging to the management plane (prefer the usage of Unique Local Addresses - ULA [12] if available). Special attention must be brought to the routing protocol used, and rules may be set accordingly, to avoid dropping packet used for internal or external routing.

Finally, we introduced the notion of blacklist, allowing the administrator to define a set of hosts or prefixes that are not allowed to communicate with the nodes within the site.

E. Pinholes

All the firewalls we deploy are stateful. This implies that we need to explicitly open pinholes to allow access to the services within the site or a subnet of it. These pinholes create rules as detailed as possible that permit some traffic to pass the firewall.

The pinholes can be set in input (e.g. to access a WEB server in the DMZ), in output if it is restricted (e.g. permit host within a NAT subnet to surf on the Internet), or both (e.g. a DNS server in the DMZ must be accessible from the internal network, but also needs to join other DNS servers in the Internet for recursion). Other pinholes are by default restricted to the management alias (e.g. SSH access to the servers in the DMZ). They can be expressed via service aliases or via explicit rules.

An alias consists in a couple composed of the name of the service and the host or prefix to use as destination. If the service runs on another port than the default one, it can be specified. Moreover, the access can be restricted to a given set of sources (hosts, prefixes or aliases), the default being any. We defined rules for the following aliases: SSH, HTTP(S), NTP, DNS, SMTP(S), POP3(S), IMAP(S), Windows Shares, NFS, FTP, SNMP, SCTP, DCCP, IPSec (AH+ESP). The definition of DNS differs from the other aliases, because we need to allow recursion over the Internet. We must thus specify a primary and eventually a secondary server for which the recursion will be permitted. Moreover, we do not consider here the security that should be performed on the server itself, e.g. disabling recursion for the DNS server if the request is issued by an external node.

However, it is impossible to have default aliases for all possible services. Therefore we allow the definition of custom rules, by using a generic representation of firewall rules that we defined. This generic representation allows to express in XML format all the rules that need to be deployed in this particular case, and in most of the real scenarios that one can encounter in SME networks.

As we have several firewalls in our network, the consistency of the security policy between them is important and will require special attention. If the usage of the common bogon and ICMPv6 filtering will ensure part of the consistency, the pinholes are more challenging. We have to propagate the pinholes on all firewall on the path to the border.

IV. Implementation and evaluation

All the algorithms have been implemented in an integrated environment called 6Tea. The framework is fully implemented in Python and offers the transition service both as a standalone tool and remotely through a convenient web service. All components are available under the GPLv2 license in the INRIA forge [4].

The environment comprises two main components : a transition engine and a security engine. The transition engine

[4] https://gforge.inria.fr/projects/v4-to-v6/

provides the generation and configuration of the addressing plan while the security engine handles the filtering aspects. Both engines work together over the whole transition process.

A. Transition and security engines

The transition engine takes as input a network model. In 6Tea, the network model is expressed in the Dot language from the Graphviz framework [5]. The advantage of this framework, is that it allows to represent the network with many characteristics, easily generate a graphical view exploitable in WEB or GUI interfaces, and the language itself is really easy. The DOT file is parsed using the Boost Graph Lib [6] and its Python bindings [7], 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.

Alongside the DOT representation, the program uses an XML file as input, to gather the necessary information about the network. This file contains all the information needed on the network devices, subnets, and how the administrator wants the new IPv6 network to behave in terms of addressing. This file also allows to tune the execution of the security engine by configuring various constraints (exclude and force prefixes, links end points addressing...), and give the information missing in the DOT file (site BGP AS and neighbors, do we write the configuration in the devices memory or is it a test deployment on a testbed ?). It also details the information about the routers interfaces (ID, IPv4 and MAC addresses, existing IPv6 addresses or prefixes assigned in an earlier execution of the engine...), the routing protocol they use, their type (Cisco or Quagga), and permits to set constraints at the router level (force prefixes, force predecessor). Finally, as for the routers, it gives more detailed information about the networks.

The security engine works in cooperation with the transition engine, as information about the network graph is required. The security policy to apply is defined in another XML file which contains parameters to tune all the rules presented in section III. All the defined services aliases have been implemented by sub-classing a generic template. These aliases are dynamically loaded by the tool at runtime, which enables the definition of custom aliases by the administrator without modification of the engine itself.

Network devices (routers and firewalls) can be configured via Telnet or SSH to deploy the proposed addressing and security plans on the network.

6Tea produces a DOT/XML pair of files describing the generated addressing plan which follows the same schema than the files used as input. For each firewall identified and configured by the security engine, 6Tea generates their configuration via a generic XML firewall representation plus a dedicated configuration in their respecting configuration language (ip6tables scripts for Netfilter or CLI for Cisco devices).

[5] http://www.graphviz.org
[6] http://www.boost.org/libs/graph/doc/index.html
[7] http://www.osl.iu.edu/~dgregor/bgl-python/

Two versions of the framework are available: an offline version with a GTK graphical user interface (GUI) and an online version. The online version will be hosted on a dedicated server, and a user will be able to use or test the tool by uploading the configuration files on the server and running the engine directly on the server. For privacy issues, the files will be stored only for a short amount of time.

B. Experimentations

6Tea has been validated experimentally on several of our testbeds, the one illsutrated here being composed of three Quagga (*chocolat*, *kran* and *luffy*) and two Cisco (*kunu* and *garou*) routers and seven subnets, as shown in figure 5.

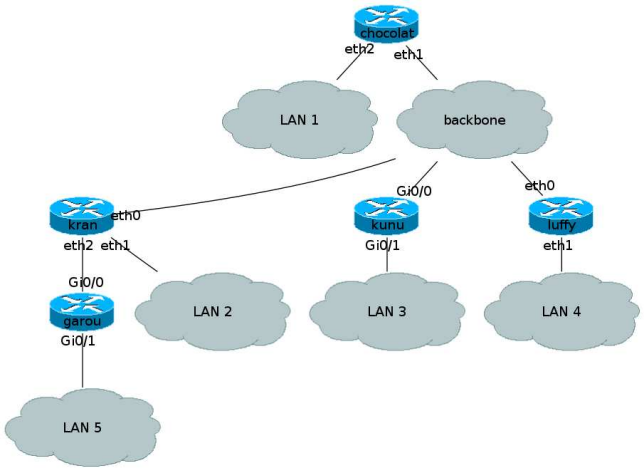

Fig. 5. Testbed - Initial network

We ran the tool on 19 different scenarios covering the different constraints addressed in the study. These scenarios consider the different constraints addressed in the study: force a prefix on a link/subtree/router, exclude prefixes from the addressing plan, different cases of multihoming (at subnet or site level) introducing loops in the routing infrastructure, end points addressing and os on. Other scenarios were mixing different constraints. All the scenarios testbed are shipped with the tool itself in order to show how they are configured and permit to replay them.

If we consider the simple case of a tree-like topology as represented in figure 5, and deploy the IPv6 prefix *2001:db8:1234::/48*, we obtain the IPv6 addressing plan shown in figure 6.

We applied a basic security policy on the site where *LAN1* is marked as a DMZ, *LAN5* as NAT and *LAN4* as local. We opened few pinholes (HTTP to the DMZ, Telnet to the whole site and SSH limited to the management alias to the NAT and local subnets). The routers *chocolat*, *luffy* and *garou* have been identified as firewalls and configured as such. The output consists in 1000 rules for *chocolat*, 675 rules for *luffy* and 447 rules for *garou*. Netfilter firewalls have more rules than Cisco firewalls as ip6tables handles packet directed to or emitted by the firewall itself in separated INPUT and OUTPUT chains in which the common rules defined in section III-D are

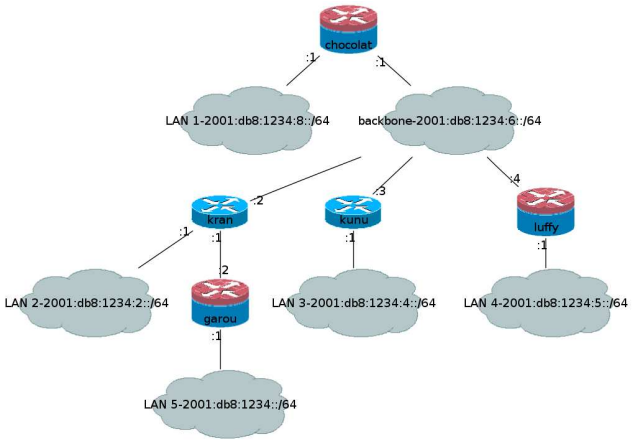

Fig. 6. Secured IPv6 network

duplicated. The router *chocolat* is at the inter-connection with the ISP, and has three interface when the other two firewalls only have 2 interfaces, which is why it has more rules than them.

V. Related Work

The vast majority of research that has been undertaken in the area of IPv4 to IPv6 transition addresses co-existence of the two protocol stacks. For this specific case, many proposals exist : dual stacks, IP/ICMP translation, NATPT, Dynamic Tunneling Interface as presented in [13]. 6Tea is radically different from these efforts by focusing only on the addressing scheme and considering a complete network transition.

While transition is heavily documented in mutiple RFCs and project deliverables, 6tea is the first effort that provides a fully automated support for enterprise network administratos to make this transition successful. Some tools exist that assess the readiness of IPv6 devices, none does automatically configure the network and actually perform the transition. These tools like for example Opnet's IT-Guru are useful above 6Tea, i.e. their output can be used as input to our automated adressing environment.

Regarding IPv6 addressing automation, a very interesting work was done by Jelger and Noel [14] in multi-provider mobile ad-hoc networks. They provide a fully distributed algorithm enabling network nodes to compute their addressing plan according to one metric, which is prefix continuity. While their method can be adapted to compute the addressing scheme, it is hardly usable in enterprise networks since it requires some convergence time and since it does not provide recovery schemes. Finally their approach does not at all consider the security configuration issues which are crucial in enterprise networks.

VI. Conclusion

While IPv6 is growing in core and large companies networks, the transition remains an issue for many SMEs. To ease this operation to be performed we have built network model and designed a set algorithms that enable an automatic transition. Our contribution is extended with an approach to secure the newly generated IPv6 network based on a global site policy model. 6Tea, the actual implementation of the framework is implemented and freely available. Its use demonstrates that automation is possible for most enterprise networks.

This work opens the way to several extensions. First the generic firewall model we built could be used in the Netconf community as a generic firewall configuration data model. Remaining in the firewall configuration part of our work, we plan to couple these configurations with a formal validation model and a test generation environment to actually test the deployed environments against the policies. The initial topology discovery has not yet been considered, as we thought of it more as an engineering aspect than research challenge. However, automating the discovery of the network to number and the generation of the input files are important steps on which we need to focus in the future. Finally we started to investigate the design of a fully distributed approach of making the transition allowing each device to make its own decisions based on context information like the number of neighbors and their metrics.

References

[1] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460 (Draft Standard), IETF, Dec. 1998, updated by RFC 5095. [Online]. Available: http://www.ietf.org/rfc/rfc2460.txt

[2] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration," RFC 4862 (Draft Standard), IETF, Sept. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4862.txt

[3] J. Bound, "IPv6 Enterprise Network Scenarios," RFC 4057 (Informational), IETF, June 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4057.txt

[4] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861 (Draft Standard), IETF, Sept. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4861.txt

[5] T. Narten, R. Draves, and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," RFC 4941 (Draft Standard), IETF, Sept. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4941.txt

[6] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh, "NAT Behavioral Requirements for TCP," RFC 5382 (Best Current Practice), IETF, Oct. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5382.txt

[7] P. Savola and C. Patel, "Security Considerations for 6to4," RFC 3964 (Informational), IETF, Dec. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3964.txt

[8] G. V. de Velde, T. Hain, R. Droms, B. Carpenter, and E. Klein, "Local Network Protection for IPv6," RFC 4864 (Informational), IETF, May 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4864.txt

[9] M. Wasserman and F. Baker, "IPv6-to-IPv6 Network Address Translation (NAT66)," draft-mrw-behave-nat66-02 (Internet-Draft), Nov. 2008. [Online]. Available: http://tools.ietf.org/id/draft-mrw-behave-nat66-02.txt

[10] J. Abley, P. Savola, and G. Neville-Neil, "Deprecation of Type 0 Routing Headers in IPv6," RFC 5095 (Proposed Standard), Internet Engineering Task Force, Dec. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5095.txt

[11] E. Davies and J. Mohacsi, "Recommendations for Filtering ICMPv6 Messages in Firewalls," RFC 4890 (Informational), IETF, May 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4890.txt

[12] R. Hinden and B. Haberman, "Unique Local IPv6 Unicast Addresses," RFC 4193 (Proposed Standard), IETF, Oct. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4193.txt

[13] H. Afifi and L. Toutain, "Methods for ipv4-ipv6 transition," *Computers and Communications, IEEE Symposium on*, vol. 0, p. 478, 1999.

[14] C. Jelger and T. Noel, "Algorithms for prefix continuity in ipv6 ad hoc networks, ad hoc and sensor wireless networks," *OCP Science*, vol. 2, no. 2, May 2006.