INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt (2009)
Published online in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/nem.717

Preparing network configurations for IPv6 renumbering

Damien Leroy*† and Olivier Bonaventure

Université catholique de Louvain (UCL), Belgium

SUMMARY

The difficulty of renumbering a network—i.e., adding and removing an Internet Protocol (IP) prefix due to a provider change or a network merger—is a real issue for most network administrators. In this paper, we propose a set of macros that can be used in configuration files. These allow network administrators to write generic configurations that are independent of the public IPv6 prefixes allocated to their network. We explain how to apply our macros to key configuration files, namely firewall access lists, Domain Name Service and Dynamic Host Configuration Protocol, and how to use this mechanism in a full renumbering process. Copyright © 2009 John Wiley & Sons, Ltd.

1. INTRODUCTION

The current Internet Protocol (IPv4) was designed in the 1970s. At that time, IP addresses were divided into classes, and organisations willing to connect to the Internet had to obtain an address block from the Internet Assigned Numbers Authority. In the late 1980s, class-based addresses combined with the growth of the Internet caused two main problems [1]. First, the sizes of class A, class B and class C addresses were too rigid. Second, projections in the early 1990s indicated that the 32- bit Internet address space would become an increasingly limiting resource. The first problem was solved by the introduction of Classless Interdomain Routing (CIDR) [1], which supports variable-size subnets. To face the second problem, the development of IP next generation, now known as IPv6, began [2]. Compared to IPv4, the main benefit of IPv6 is its 128-bit address space.

From a scalability viewpoint, a key element of an addressing architecture is how address blocks are allocated. With IPv4, address blocks were initially allocated on a first-come, first-served basis. Two types of address blocks were the defined: Provider Independent (PI) and Provider Aggregatable (PA). PI address blocks are assigned by routing registries to Internet service providers (ISPs) and large customers only. PA address blocks are assigned by ISPs, from their PA block, to smaller customers. When a PI address block is allocated to an organisation, the organisation can use this address block while being connected to any provider. On the other hand, an organisation that received a PA address block from provider X cannot switch to provider Y without renumbering, i.e., updating all its IP addresses.

With IPv6, the initial address allocation plans [3] were strongly in favour of mainly allocating PA address blocks to avoid overloading the Border Gateway Protocol (BGP) routing tables and, thus, to permit routing scalability. The first policies used by Regional Internet Registries (RIR) basically assumed that IPv6 addresses would only be allocated to ISPs [4]. Today, small customers are lobbying to force the RIRs also to allocate PI address blocks to them [5]. Their main motivation is that, unfortunately, practical experience has shown that it is very difficult for a customer network to renumber when it needs to change provider. To successfully renumber and so to avoid provider lock-in, sites must indeed be able to update all configuration files in which IPv6 addresses appear. Typical examples include Domain Name Service

*Correspondence to: Damien Leroy, Universite catholique de Louvain, Place Ste Barbe 2, Louvain-la-Neuve 1348, Belgium
†E-mail: damien.leroy@uclouvain.be

(DNS) configurations, router configurations, firewall access lists and Dynamic Host Configuation Protocol (DHCP) configurations. Although this problem had been identified in the mid 1990s [6], the procedures that should be followed today for renumbering [7,8] require many manual operations and are therefore still complex, lengthy and error prone.

Besides, it can be considered that an IPv6 site will have to renumber one or several times during its existence. This can be a consequence of a network growth or merger, or, as claimed earlier, due to a provider change. From this perspective, the fact that many networks have not yet started their transition to IPv6 can be seen as an opportunity to write down configurations in such a way as to make further renumbering easier.

In this paper, we tackle the renumbering issue by showing that it is possible to partially automate the process. To do so, we split the problem into an addition and removal problem and define macros allowing configurations to be written that are independent of the currently available prefixes in the site. Later, each time a prefix change occurs, the actual configuration files are regenerated according to the available prefixes. This technique is mainly targeted at networks that aim at frequent renumbering but can be applied in a larger scope.

The remainder of this paper is organised as follows. Section 2 explains how a network can be simply prepared once for future renumbering through several case studies. Section 3 describes how, in a prepared network, a prefix can be added or removed. Finally, Section 4 concludes the work.

2. PREPARING CONFIGURATIONS FOR PREFIX CHANGES

An IPv6 site wishing to prepare for an easy addition or removal of its public prefixes has to be configured in an appropriate way. This configuration is made up of two parts. Firstly, in addition to its public IPv6 prefix or prefixes, we propose that the site also uses internally a ULA (Unique Local Addresses [9]) prefix. The ULA prefix is used to provide stable IPv6 addresses at least to all nodes containing a configuration file that would be affected by a prefix change (routers, servers, firewalls, middleboxes, etc.). Most regular IPv6 hosts either use autoconfiguration or DHCPv6 and thus do not contain configuration files that must be updated. These stable ULAs are of course only reachable inside the site. Secondly, all the configuration files are prepared for the changes. Basically, we introduce macros into the configuration files that allow automatic generation of updated configuration files containing the actual IPv6 prefixes whenever a prefix is added or removed. Actually, a renumbering event contains a period during which both prefixes coexist, the new prefix being preferred to the former one for new connections. Once most connections have switched to the new prefix, the former one is removed. Networks in which some long-lived connections must survive the renumbering are beyond the scope of this paper and should consider using protocols such as Host Identity Protocol.

Owing to space limitations, we concentrate on the configuration of the services that cause most problems [9,10] when an IPv6 network needs to be renumbered, namely DNS, firewalls, DHCP and Neighbour Discovery. For each of these services, we obtained ISP and campus network configuration files from Japan, China and Belgium. Similar solutions can be developed for other services. Both IPv4 and IPv6 configurations were used: IPv6 since it is the target of our mechanism and IPv4 since it large real configurations to be obtained that are not yet available in IPv6. We succeeded in rewriting all these configurations using our macros and without changing their semantics. For this purpose, we developed a toolbox called *Macro based Prefix Updater* (MPU), which can be downloaded from our website (http://inl.info.ucl.ac.be/MPU). The only requirement for our solution is that the configuration of each service can be generated from an ASCII file.

2.1 Macro definitions

Adding (or removing) an IPv6 prefix from the configuration files used by a site is more complex than simply performing search and duplicate (or remove) in all configuration files. For example, DNS configuration files will contain IPv6 addresses in both standard and reverse form. Other examples are firewalls where access lists can contain rules that are applicable to some prefixes but not all of them. Furthermore, multi-homed sites may have different policies concerning their public prefix allocation. For instance, a corporate network attached to one research and two commercial ISPs may use all prefixes inside its research lab while the configuration of its data centres will only use the prefixes allocated by the commercial ISPs. To cope with these issues, we propose flexible macros that are used to replace all prefixes in the configuration files.

The simplest statement in our macros is written as

```
[[<expression1> $$ <expression2>]<separator>]
```

This means that the string between the inner square brackets has to be repeated for each prefix after having replaced `$$` by the prefix. The separator (`<sep>`) is placed between each repetition; typical values for it are a colon, semicolon or a new line. For example, '`[[address $$:a::1],]`' in a network using the prefixes `2001:db8:1::/48` and `2001:db8:8::/48` would be converted to '`address 2001:db8:1:a::1,address 2001:db8:8:a::1`'.

Besides, some configurations may need to apply a function on prefixes such as DNS using the reverse IP notation. For this reason, a function can be defined as follows:

```
[[<expr>$ [<function_name>:]$<expr>]<sep>]
```

For example, `[[zone "$reverse:$.ip6.int" {...}]]` can be used to define a reverse DNS zone for each prefix of a site. Other functions are defined later for lifetimes. MPU can easily be extended with new functions.

Finally, we introduce colours permitting the discrimination of prefixes to use in each macro. For this purpose, each prefix is associated with one or several colours. For example, consider a network having both research and commercial prefixes. The research prefixes are tagged with the *rd* colour while the commercial prefixes are tagged with the *comm* colour. Assume that part of a configuration file should only be applied to research prefixes. Colours can thus be used to restrict the duplication of this part only to research prefixes. For instance, a statement starting with '`[comm,rd;rd[...`' means that two colour sets (separated by semicolons) are defined. The first one contains prefixes of colour *rd* or *comm* and set 2 contains the prefixes having colour *rd*. Given these sets, the statement can use the patterns `$1$` and `$2$` to respectively relate to prefix set 1 or 2. As a result, the statement is first duplicated for each prefix of set 1 replacing `$1$` by the corresponding prefix, then it is done for set 2. If, at one given moment, the network has one research prefix `2001:db8:3::/48` and one commercial prefix `2001:db8:7::/48`, the following firewall pseudo-rule can be written to express that all packets from commercial or research prefix to the host with suffix `::9` using research prefix have to be dropped;

```
[comm,rd;rd[if packet from $1$::/48 to $2$::9, then drop]]
```

For the sake of simplicity, an empty colour set is equivalent to all prefixes and no colour defined between dollars is equivalent to declare set 1. In '`[[address $$:a::1],]`', it thus means that set 1 is made of all prefixes and that this rule will be duplicated for each prefix of set 1, '`$$`' being replaced by the prefix value.

In the following, we detail how macros can be applied to DNS (Section 2.2), firewalls (Section 2.3) and some other services (Section 2.4).

2.2 *The Domain Name Service (DNS)*

For the DNS, our case study are BIND configuration files. BIND server configuration is a set of statement blocks containing options, a sample being given in Figure 1. Addresses appear either in a statement definition (`server` and `zone` in Figure 1) or within an option (e.g., `acl`, `query-source` or `masters`). If an address appears in a statement definition (e.g., at line 6 in Fig. 1), the entire statement has to be duplicated for each prefix. In options, addresses are mostly used in lists, e.g., at line 1 or 13 in Figure 1. In this

```
 1 acl "internals" { 2001:db8:1::/48; fd12:3:4::/48; }
 2 options {
 3             notify no;
 4             query-source address 2001:db8:1:a::d;
 5 };
 6 server 2001:db8:1:7::a { request-ixfr no; };
 7 view "local" {
 8     match-clients { internals; };
 9     recursion yes;
10     zone "mydomain.net" {
11         type slave;
12         file "db.net.mydomain";
13         masters { 2001:db8:1::a; };
14     };
15 zone "1.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa" { ...
16     };
17 };
```

Figure 1. A sample of BIND configuration file

```
 1 acl "internals" { [[$$::/48;]] fd12:3:4::/48; }
 2 options {
 3             notify no;
 4             [main[query-source address $$:a::d;]]
 5             ...
 6 };
 7 [[server $$:7::a { request-ixfr no; };]]
 8 view "local" {
 9     match-clients { internals; };
10     recursion yes;
11     zone "mydomain.net" {
12         type slave;
13         file "db.net.mydomain";
14         masters { [main[$$::a;]] };
15     };
16 [[ zone "$reverse:$.ip6.arpa" { ...
17     }; ]]
18 };
```

Figure 2. The generic version of the BIND configuration file of Figure 1

case, each address has simply to be duplicated for each prefix. However, some BIND options require a single address as a parameter: *query-source*[*-v6*], [*alt-*]*transfer-source*[*-v6*] and *notify-source*[*-v6*]. These define the source addresses that must be used for specific operations. A better way to deal with it would be to rely on the host's address selection mechanism [11] and thus not to use these options. If for any reason they must be used, a solution to express it with our macros is to use a colour bound to one prefix, the preferred one.

The information requested and contained in DNS replies is called a *Resource Record* (RR). RRs are stored in zone files as tuples `<name> [<TTL>] <type> <value>` . Sample RRs are given in Figure 3. Zones are declared in the configuration file (lines 10–16 in Figure 1) specifying the files where RRs are stored. A zone is named by what it defines, i.e., the domain name for direct resolution and the address prefix for reverse resolution. In BIND, addresses used for a reverse zone are inverted as seen on line 15 of Figure 1.

Using our macros, the configuration of Figure 1 can be abstracted as shown in Figure 2 to become independent of the global prefixes. At line 4 of this configuration, it is shown how an option taking only

```
1 | ns1              IN    CNAME    nameserver
2 | www              IN    AAAA     fd12:3:4:a:9
3 | www    7200      IN    AAAA     2001:db8:1:a:9
4 | 7.1.b.a.3.a.d.0.c.7.b.e.7.9.d.1.c.0.0.0 IN PTR abcd.mydomain.net.
5 | 9.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.a.0.0.0 86400 IN PTR www.mydomain.net.
```

Figure 3. Some sample BIND RRs

one IP address can be transformed under the assumption that the *main* colour is bound to one prefix. Line 16 uses the reverse function to transform the prefix for reverse DNS zones.

A sample of direct (or reverse) RRs that can be found in zone files is shown at lines 2–3 (or 4–5) of Figure 3. In direct zone files, names are mapped to IP addresses. When a new prefix is added, the RRs have therefore to be duplicated. In reverse zone files, RRs map an address suffix to a name. Those files do not need to be changed when a new prefix is added. Only their declaration statement in the server configuration (lines 15–16 in Figure 1) needs to be duplicated.

RRs can also contain a TTL when they are not using the default one (defined for the whole zone file), for instance 7200 and 86400 (24 hours) in Figure 3. This TTL gives to the DNS client the lifetime during which it can still use this record without sending a new request, typically a few days. When a new prefix is added, no special care has to be taken with this parameter. However, when a prefix has to be removed, the RRs associated with this prefix will remain in client caches for TTL time. This issue, briefly mentioned in some renumbering discussions [7,8], concerns only direct RRs since clients will still try to connect to hosts using addresses that no longer belong to them. There are two opposite solutions to this issue. First, the direct RRs could be removed DNS TTL-seconds before the actual removal of the corresponding prefix. This can only be applied if another prefix is routable in the network. Another solution is to set the TTL to zero TTL-seconds before prefix removal time and then to remove the RRs at the time of removal. This solution may lead though the DNS server to receive a large number of requests during the zero-TTL period. Of course, a trade-off should be chosen between these two solutions. Such a trade-off could be to decrease the TTL in several steps, e.g., to divide the starting TTL by two TTL-seconds before prefix removal, to divide it once more by two new-TTL-seconds before prefix removal, and so on until the TTL reaches a threshold at which it is set to zero.

In practice, in Figure 3, the only RRs that need a change for a generic configuration is the third one which would become:

```
3 | [[www $ttl:$ IN AAAA $$:a:9]]
```

End-hosts may also have entries in the DNS server. If so, their entries do not necessarily need to be updated using our tool when a new prefix is known. Indeed, some of them may need to use a different IP suffix for each prefix received, e.g., if they are using Cryptographically Generated Addresses (CGA) [12] or privacy extensions [13]. In this case, DNS records cannot be updated by simply replacing the former prefix by the new one. Dynamic updates [14] must be used by such end-hosts to update their own RRs.

In 2000, A6 records were proposed at IETF [15] to support renumbering. They rely on address chaining to obtain an IP address from a name. Each part of the address was placed in a separate RR, making site renumbering much easier. Unfortunately, A6 has been deprecated in favour of AAAA because of the potential overweight of the chaining [16].

2.3 Firewalls

Other services that need to be carefully updated when a network is renumbered are the firewalls and the access lists used on routers. Intuitively, when a prefix is removed, the rules matching the prefix (e.g., 'from prefix A, drop') must be entirely removed, while in the case of rules excluding the prefix (e.g., 'not from prefix A nor prefix B, drop'), only the conditions related to the prefix must be removed. When a new prefix is added, the same principles are applied, i.e., the entire rule is duplicated if the rule matches the prefix and only the condition is duplicated if the prefix is excluded.

In this section, we focus on firewall configurations that can be written as a list of rules made of a conjunction of conditions and a target (allow or deny). In practice, all commonly used firewall implementations fit this model. We verified this on real network configurations using *Netfiler/iptables*, *Cisco IOS* access lists and JunOS firewalls. If a firewall language allows disjunctions, these can be easily transformed by splitting them into two rules as shown in Table 1. Like Cheswick *et al.* [17], we use a table notation to represent firewall rules for the sake of readability.

Formally, a filtering rule can be expressed either as rules of type (1) or type (2). In these rules, C_i is a condition unrelated to IP addresses and P^A is a condition matching an address or address block from prefix A. The target defines the actions to be performed when the rule is matched; in practice common targets are *allow* or *deny*.

$$\text{Either } \bigcap^{i} C_i \cap P^A \Rightarrow \text{target} \tag{1}$$

$$\text{or } \bigcap^{i} C_i \cap \neg P^A \Rightarrow \text{target} \tag{2}$$

If a new prefix, B, is added, rule (1) can be rewritten as follows:

$$\bigcap^{i} C_i \cap (P^A \cup P^B) \Rightarrow \text{target} \equiv \begin{cases} \bigcap^{i} C_i \cap P^A \Rightarrow \text{target} \\ \bigcap^{i} C_i \cap P^B \Rightarrow \text{target} \end{cases} \tag{3}$$

This means that a rule containing a condition that matches a prefix must be entirely duplicated for each prefix. On the other hand, rule (2) can be rewritten as

$$\bigcap^{i} C_i \cap \neg(P^A \cup P^B) \Rightarrow \text{target} \equiv \bigcap^{i} C_i \cap \neg P^A \cap \neg P^B \Rightarrow \text{target} \tag{4}$$

which means that a rule containing a condition excluding a prefix must have only the condition duplicated for each prefix. For instance, consider a network owning prefix `2001:db8:1::/48` that uses the filtering rules listed in Table 2.[1] If this network obtains a second prefix, `2001:db8:abc::/48`, the first five rules would be updated as shown in Table 3. According to property (3), rule 1 gives two rules: 1a and 1b. On the other hand, using (4), rule 4 only has more conditions when a prefix is added.

#	Action	Source address	Port	Destination address	Port
1	allow	`2001:db8:1::/48` | `2001:db8:e::/48`	*	`2001:db8:a:4::1`	`http`
2a	allow	`2001:db8:1::/48`	*	`2001:db8:a:4::1`	`http`
2b	allow	`2001:db8:e::/48`	*	`2001:db8:a:4::1`	`http`

Table 1. In this example, rule 1 is semantically equivalent to rules 2a and 2b. It allows `http` packets from either `2001:db8:1::/48` or `2001:db8:e::/48` prefix to a specific IP address

[1] *In this table, '!' indicates a negation.*

#	Action	Source address	Port	Destination address	Port
1	allow	*	*	2001:db8:1:a::d	dns
2	allow	*	*	2001:db8:1:7::a	dns
3	deny	! 2001:db8:1:a::d & ! 2001:db8:1:7::a	dns	*	*
4	allow	*	*	!2001:db8:1:5::/64	ssh
5	deny	*	*	2001:db8:1::/48	telnet
6	allow	2001:db8:1::/48	*	2001:db8:1:a::4	smtp
	[...]				

Table 2. A sample firewall table in a network owning the prefix 2001:db8:1::/48

1a	allow	*	*	2001:db8:1:a::d	dns
1b	allow	*	*	2001:db8:abc:a::d	dns
2a	allow	*	*	2001:db8:1:7::a	dns
2b	allow	*	*	2001:db8:abc:7::a	dns
3	deny	! 2001:db8:1:a::d & ! 2001:db8:abc:a::d & ! 2001:db8:1:7::a & ! 2001:db8:abc:7::a	dns	*	*
4	allow	*	*	! 2001:db8:1:5::/64 & ! 2001:db8:abc:5::/64	ssh
5a	deny	*	*	2001:db8:1::/48	telnet
5b	deny	*	*	2001:db8:abc::/48	telnet

Table 3. The first five rules of Table 2 if a prefix (2001:db8:abc::/48) has been added to the network

6a	allow	2001:db8:1::/48	*	2001:db8:1:a::4	smtp
6b	allow	2001:db8:abc::/48	*	2001:db8:abc:a::4	smtp

Table 4. A first way of rewriting rule 6 of Table 2 when a new prefix is added. It does not allow a packet to be sent from one prefix to the other

6a′	allow	2001:db8:1::/48	*	2001:db8:1:a::4	smtp
6b′	allow	2001:db8:1::/48	*	2001:db8:abc:a::4	smtp
6c′	allow	2001:db8:abc::/48	*	2001:db8:1:a::4	smtp
6d′	allow	2001:db8:abc::/48	*	2001:db8:abc:a::4	smtp

Table 5. A second way of rewriting rule 6 of Table 2 when a new prefix is added. The rule is duplicated for each prefix pair

The last rule (rule 6) contains several conditions related to IP address. When a prefix is added, such a rule can be converted in two ways: either as shown in Table 4 or in Table 5. The first consists in duplicating the rule considering each prefix independently. It does not allow packets to be sent from one prefix to the other. The second considers each prefix pair for building rules. In Table 5, this allows packets to be sent from one prefix to another. If the rule target were 'deny', the second option should be chosen to prevent a host from bypassing the rule by sending packets from one prefix to the other. The choice between these two is dependent on the policy of the network and has to be made by an administrator. The second solution should be chosen unless administrators decide to deny cross-prefix connections within the network. Both can be expressed with our macros.

Thus, using the macro definitions described earlier in this section, the initial firewall of Table 2 could be written as Table 6 in order to make it prefix independent without changing its semantics.

#	Action	Source address	Port	Destination address	Port
1	[[allow	*	*	$$:a::d	dns]]
2	[[allow	*	*	$$:7::a	dns]]
3	deny	[[! $$:a::d]&] &	dns [[! $$:7::a]&]	*	*
4	allow	*	*	[[! $$:5::/64]&]	ssh
5	[[deny	*	*	$$::/48	telnet
6	[;[allow	1::/48	*	2:a::4	smtp]]
	[...]				

Table 6. Prefix-independent firewall table having the same semantics as that one of Table 2

In rules 3 and 4, the '&' (and) sign between closing brackets is the separator. Rule 6 uses a semicolon between the opening brackets; this expression means that both set 1 and set 2 correspond to all prefixes. As a result, if *n* is the number of prefixes, this rule is first duplicated *n* times replacing `$1$` by each prefix and then each duplicated rule is duplicated once more *n* times replacing `$2$` by each prefix.

2.4 Other configurations

Both DHCP and Router Advertisements (RA) are used to permit end-hosts connected on a LAN to obtain and use IP addresses. In practice, the router running DHCPv6 server (DHCPs) or Router Advertisement daemon (RAd) is often the egress router of the LAN. RAd is used for stateless IPv6 address configuration [18]. It floods on the LAN the prefixes that can be used by the end-hosts as well as their *preferred* and *valid lifetimes*. On the other hand, DHCPv6 [19] is used as a server-based address configuration. When it receives an address request, it replies with one or several addresses and their *preferred* and *valid lifetime*. The *preferred lifetime* is the remaining time during which this address should be *preferred* by the host, i.e., should be used for any connection. The *valid lifetime* is the time left during which this address can be used. When the *valid lifetime* expires, the address must stop being used by the host. When an address is not *preferred* anymore but still *valid*, it becomes *deprecated*. In this state, it cannot be used for new connections but can still be used for already established ones.

As for DNS RR, addresses or prefixes are therefore associated with TTLs, called lifetimes. There are two possible approaches for updating them for RAd and DHCPs. The first one is to apply the same procedure as for DNS, i.e., updating prior to renumbering the lifetime to a shorter one. Another method, which does not need early scheduling, is to reduce the prefix lifetime of end-hosts prior to its expiration. For DHCPv6 this can be done by using a DHCP *reconfigure message* that triggers the hosts to re-contact the server. For RA, it is done by sending new prefix information. However, RA specification [20] defines that, in order to avoid DoS, the lifetime reduction updates of unauthenticated prefixes[2] cannot be used if there are fewer than 2 hours left.

RAd and DHCPs use similar configuration files. They consist of several global parameters and some configuration blocks: one for each prefix. These blocks must be duplicated every time a prefix is added. They contain obviously the prefix value but also the different lifetimes.

Besides these services, some other configurations need some changes. The interface addresses of routers and servers that use neither RA nor DHCP have to be updated. Such configuration files can be very different from one operating system to another. Actually, these are quite simple and similar to DHCPs/RAd configurations.

[2]*Authentication of RA can be done using the SEND protocol.*

Routing protocols daemons and other strictly local protocols should local addresses [9] use as much as possible for their communications. However, even if global addresses are needed, we observed that configuration files of most of them are very similar to the ones we previously discussed in this paper.

2.5 On the cost of preparing configurations

In pure manual renumbering of configurations, each time a prefix change occurs, all the configuration files concerned have to be updated. With the solution discussed in this paper, the renumbering has to be prepared only once, prior to any prefix change. Later, the changes can be automated and should be error free. In fact, the configuration files using macros are parsed by MPU, which generates real configurations containing all the prefixes in use. The first preparation of the configuration files can be made by converting existing configurations to prefix-independent ones, as has been done for DNS (from Figure 1 to Figure 2) or for firewalls (from Table 2 to Table 6). Prefix-independent configurations can also be written directly from scratch when new services are configured.

It is pretty difficult to evaluate what is the complexity of building these macro-based configurations. When they are written directly from scratch, we can assume that the additional cost is negligible compared to the cost of writing the full configurations. The cost of modifying existing configurations seems to be fairly light. Actually, most of the process can be automated by using pattern-matching scripts. In the significant and large real configurations we examined during our analysis on DNS, firewalls and DHCP configurations, we observed ratios of respectively 250/253 (98.8%), 689/750 (91,7%) and 50/50 (100%) of statements that have been automatically transformed using simple scripts. For the remaining ones, we had to take a look in order to interpret them in the right way.

3. PREFIX ADDITION OR REMOVAL PROCEDURE

Section 2 explained how the configuration files should be prepared once, for any prefix change event. Here, we discuss how a prefix addition or removal should be performed to avoid service outage. In practice, a prefix change cannot be considered as an instantaneous event as different service updates must be done in a specific order. For instance, a new prefix should not be assigned to end-hosts while a firewall is still blocking this prefix. As a result, a prefix addition or removal procedure is composed of several steps [7]. This is often error-prone and should not be done manually.

The easiest way is probably to rely on a centralised management tool that knows which services are running on which nodes and that is able to plan and monitor the renumbering procedure. In practice, this tool sends orders to network nodes in accordance with the current state and monitors the results.

In order to perform a first validation of our tool, MPU, we used configurations we have obtained from ISPs and campus networks. For the services we could install in our lab, we deployed them in a network using SOAP processes for triggering the prefix changes. The services restarted correctly and their behaviour appeared to be coherent and correct after each update.

3.1 Related work

Detailed renumbering procedures have been discussed in several IETF documents [7,8] describing issues that may be raised by renumbering and existing solutions. They describe constraints that are taken into account in this paper, but do not propose any concrete automated configuration update mechanism.

For the distribution of prefix information within a site, distributed mechanisms could be used such as Router Renumbering [21] or those included in full renumbering propositions [22,23]. DHCPv6 using Prefix Delegation [24] could also be used as a centralised mechanism. However, these solutions do not enable transmission of information such as colours and distributed ones are probably not suitable for triggering each parts of the update. Finally, the monitoring part has been addressed by Beck *et al.* using the NetSV tool [25].

3.2 Removing a prefix in use

The removal of a prefix p is a three-step process: preparation for the removal, deprecation of p and definitive removal. The preparation consists, as explained earlier, in starting to decrease the TTL of DNS direct RRs and the lifetime of prefixes advertised on LANs. Depreciation corresponds to setting p as deprecated so that no host uses it any more for new connections. This has to be done for RAd and DHCP configurations as well as for static IP configuration on all network devices. DNS server configurations also have to be updated not to use p for requests (e.g., via the *main* colour in Figure 2). Finally, direct RRs using p have to be removed from the DNS. These four updates have no precedence constraints between them.

The removal of all addresses related to prefix p in configuration files is the last step of the procedure. Before starting this step, it must be ensured that DNS direct RRs concerning p are no longer used. In theory, applications should no longer use an address with an expired TTL. In practice, some applications and some recursive DNS servers do not observe this TTL at all. As a result, some additional time should be allowed before starting this last step to avoid service outage. First, addresses on LAN (via RAd and DHCPs) and on statically assigned interfaces have to be removed. As explained in Section 2.4, LAN updates may need to wait for some prefixed lifetime to expire on end-hosts. Once this is done, the last removals can be performed, in any order, in DNS server configurations, firewalls, routing protocols and reverse zone DNS records.

3.3 Adding a new prefix

Adding a new prefix in a network is simpler than removing it. First of all, firewall, routing protocols and reverse zone DNS records must be updated to consider the new prefix. Once routing has converged, addresses related to this prefix can be bound to statically configured interfaces. The DNS server can then be configured to use this new prefix, followed by LAN configuration protocols (RA and DHCP). When the prefix is used on LANs (it could take time with DHCP if reconfigure messages are not used to trigger the client to request the DHCP server), direct DNS records must be updated to take the new prefix into account.

4. CONCLUSION

Many network administrators are concerned, or will soon be concerned, by the difficulty of changing their configuration files each time a new IPv6 prefix is allocated to their network. For this reason, they lobby for obtaining provider-independent (PI) IPv6 prefixes. If this were applied, it would unfortunately lead to a risky growth of the BGP IPv6 routing tables [10]. However, the transition of networks to IPv6, requiring some configuration rewriting, can be seen as an opportunity to write configurations in a way that makes renumbering easier.

In this paper, we proposed an approach that relies on macros to allow text-based configuration files to be automatically updated upon the addition or removal of an IPv6 prefix. Our macros support prefix colouring to discriminate part of the configurations based on the prefix type. We described the procedure that should be followed when a prefix is added or removed. A prototype was implemented in Python and applied to real network configurations.

The main advantage of our approach is that it can be applied quite easily on a network, without requiring large changes in configurations or in the deployed services. Our further work is to continue our technique with a management tool for renumbering, thus building an all-in-one toolbox. Finally, the opposite approach in which services are aware of the prefix changes should be considered and analysed in comparison with the one proposed in this paper.

ACKNOWLEDGEMENTS

We would like to thank Maoke Chen, Hong Dan, Kenji Masui, Bruno Delcourt and Alain Fontaine for their help in collecting configuration files.

Damien Leroy was supported by a grant from Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture (FRIA), rue d'Egmont 5, 1000 Brussels, Belgium.

REFERENCES

1. Fuller V, Li T, Yu J, Varadhan K. Classless inter-domain routing (CIDR): an address assignment and aggregation strategy. *RFC 1519* (obsoleted by RFC 4632), Internet Engineering Task Force, September 1993.
2. Bradner S, Mankin A. The recommendation for the IP next generation protocol. *RFC 1752*, Internet Engineering Task Force, January 1995.
3. Rekhter Y, Li T. An architecture for IPv6 Unicast address allocation. *RFC 1887*, Internet Engineering Task Force, December 1995.
4. APNIC, ARIN, RIPE NCC. IPv6 address allocation and assignment policy: ripe-421, November 2007. http://www.ripe.net/ripe/docs/ipv6policy.html [29 December, 2008].
5. Palet J. Provider independent (PI) IPv6 assignments for end user organisations. RIPE Policy Proposal 2006–01, http://www.ripe.net/ripe/policies/proposals/2006–01.html [29 December 2008].
6. Carpenter B, Rekhter Y, Renumbering needs work. *RFC 1900*, Internet Engineering Task Force, February 1996.
7. Baker F, Lear E, Droms R. Procedures for renumbering an IPv6 network without a flag day. *RFC 4192*, Internet Engineering Task Force, September 2005.
8. Chown T, Ford A, Venaas S. Things to think about when renumbering an IPv6 network. Internet draft (work in progress) 'draft-chown-v6ops-renumber-thinkabout-05', Internet Engineering Task Force, September 2006.
9. Hinden R, Haberman B. Unique local IPv6 Unicast addresses. *RFC 4193*, Internet Engineering Task Force, October 2005.
10. Meyer D, Zhang L, Fall K. Report from the IAB Workshop on Routing and Addressing. *RFC 4984*, Internet Engineering Task Force, September 2007.
11. Draves R. Default address selection for Internet Protocol version 6 (IPv6). *RFC 3484*, Internet Engineering Task Force, February 2003.
12. Aura T. Cryptographically Generated Addresses (CGA). *RFC 3972*, Internet Engineering Task Force, March 2005.
13. Narten T, Draves R, Krishnan S. Privacy extensions for stateless address autoconfiguration in IPv6. *RFC 4941*, Internet Engineering Task Force, September 2007.
14. Vixie P, Thomson S, Rekhter Y, Bound J. Dynamic updates in the domain name system (DNS update). *RFC 2136*, Internet Engineering Task Force, April 1997.
15. Crawford M, Huitema C. DNS extensions to support IPv6 address aggregation and renumbering. *RFC 2874*, Internet Engineering Task Force, July 2000.
16. Austein R. Tradeoffs in domain name system (DNS) support for internet protocol version 6 (IPv6). *RFC 3364*, Internet Engineering Task Force, August 2002.
17. Cheswick WR, Bellovin SM, Rubin AD. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Longman: Boston, MA, 2003.
18. Narten T, Nordmark E, Simpson W, Soliman H. Neighbor discovery for IP version 6 (IPv6). *RFC 4861*, Internet Engineering Task Force, September 2007.
19. Droms R, Bound J, Volz B, Lemon T, Perkins C, Carney M. Dynamic host configuration protocol for IPv6 (DHCPv6). *RFC 3315*, Internet Engineering Task Force, July 2003.
20. Thomson S, Narten T, Jinmei T. IPv6 stateless address autoconfiguration. *RFC 4862*, Internet Engineering Task Force, September 2007.
21. Crawford M. Router renumbering for IPv6. *RFC 2894*, Internet Engineering Task Force, August 2000.
22. Chelius G, Fleury E, Toutain L. No Administration Protocol (NAP) for IPv6 router auto-configuration. *International Journal of Internet Protocol Technology* 2005; 101–108.
23. Leroy D, and Bonaventure O. A secure mechanism for address block allocation and distribution. In *Proceedings of IFIP Networking*, May 2008.

24. Troan O, Droms R. IPv6 prefix options for Dynamic Host Configuration Protocol (DHCP) version 6. *RFC 3633*, Internet Engineering Task Force, December 2003.
25. Beck F, Chrisment I, Festor O. A monitoring approach for safe IPv6 renumbering. In *Proceedings of the International Multi-Conference on Computing in Global Information Technology (ICCGI)*, August 2006.

AUTHORS' BIOGRAPHIES

Damien Leroy is currently a PhD student in the IP Networking Lab at Université catholique de Louvain (UCL) in Belgium. In 2006, he received the Alcatel Bell MSc Thesis Award for his master's thesis on firewalls. His current research topics are about IPv6 addressing,WiFi roaming and network security.

Olivier Bonaventure is currently a professor in the Department of Computing Science and Engineering at Université catholique de Louvain (UCL), Belgium. From 1998 to 2002 he was a professor at the Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium. Before that, he received a PhD degree from the University of Liège and spent one year at the Alcatel Corporate Research Centre in Antwerp. He is on the editorial board of *IEEE Network Magazine* and *IEEE/ACM Transactions on Networking*. He received the Wernaers and Alcatel prizes awarded by the Belgian National Fund for Scientific Research in 2001. His current research interests include intra- and interdomain routing, traffic engineering and network security.