HAL
open science

Solving the ingress filtering issue in an IPv6 multihomed home network

Etienne Gallet de Santerre, Samih Jammoul, Laurent Toutain

To cite this version:

Etienne Gallet de Santerre, Samih Jammoul, Laurent Toutain. Solving the ingress filtering issue in an IPv6 multihomed home network. ICN' 2010 : 9th International Conference on Networks, Apr 2010, Les Ménuires, France. hal-00486760

HAL Id: hal-00486760

https://hal.science/hal-00486760v1

Submitted on 26 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Solving the Ingress Filtering Issue in an IPv6 multihomed Home Network

Etienne Gallet de Santerre, Samih Jammoul and Laurent Toutain
Institut TELECOM ; TELECOM Bretagne ; RSM
Université Européenne de Bretagne , France
{etienne.galletdesanterre,laurent.toutain}@telecom-bretagne.eu
samihj@gmail.com

***Abstract*—The common practice in the industry to have several connections to the Internet through different service providers is called multihoming. Multihoming increases the capabilities and the efficiency of the Internet connectivity, allowing end-site to benefit from redundancy, traffic engineering and load spreading. Primarily limited to the largest companies, multihoming is attracting more and more enterprises as well as private customers. The fast growth of multihoming demands is facing unresolved technical issues, especially in home networks, which are not specifically administrated. One of them is the ingress filtering policy, implemented by Internet Service Providers to prevents spoofed-address attacks. This security policy discards packets whose source address is not known by the service provider. This article presents an enhanced routing scheme in multihomed IPv6 terminal sites, which allows to deal with the ingress filtering policy in ISP edge routers. In that purpose, we describe a new mechanism for selecting the default route based on the source address of a packet and present the way we implement it. This proposal solves the ingress filtering issue without implying any changes in Internet Service Providers' policy or terminal nodes. Additionally, we show that the proposed mechanism is easily deployable and needs only few changes in terminal site routers.**

I. Introduction

Many medium and large companies connect to the Internet through several Internet Service Providers (ISPs). This practice, called multihoming, allows traffic engineering, load sharing and redundancy. This last feature overcomes the failure of an ISP link by the use of another ISP link. In IPv6, a multihomed site can have several global prefixes, one for each ISP, which means several global addresses for each machine of the site. When a terminal host T_1 sends a packet to a terminal host T_2 in another site, it has to choose one source address among the pool of addresses it has. A security policy, called ingress filtering ([1], [2]) discards an outgoing packet at the ISP border if it does not have a source address which has been delegated by that ISP. So, if the terminal host T_1 chose a source address from ISP_A to send a packet and if that packet is going out the site through ISP_B, that valid packet is dropped for security reasons. In most of company networks, specific routers are configured to balance the traffic to one ISP or another regarding to the source address of a packet. But these routers, needing human configuration and maintenance, can not be envisioned in home networks, which do not have a customized administration.

A number of proposals under study, dealing with multihoming, can solve the ingress filtering problem in a multihomed site with an encapsulation scheme. The outgoing packet is encapsulated at the site exit router and tunneled to the correct exit ISP [3]. In another proposal [4], the packet can be forwarded directly through the core network in a tunnel to the destination site. However, these proposals, not only are difficult to deploy an maintain, but make the routing process more complex as well. In some cases, a new architecture and a new level of indirection are part of the solution ([5], [6], [7]). Other solutions propose to rewrite the IPv6 header, changing source and destination addresses at the exit router of the source's site and reversing that change at the destination's site border router ([8], [9]).

This paper proposes the Selection of the Default route based on the Source Address (SDSA) as a new solution to solve the ingress filtering issue in IPv6 home networks with several ISPs. SDSA modifies the classical routing scheme and makes it dependent on the source address when the forwarding mechanism has selected the default route. The SDSA mechanism requires few changes in site routers behaviour as well as in internal routing protocol to provide immediate benefits.

This article is structured as follows. Section II presents related work. Section III describes the theory of the SDSA mechanism, whereas deployment scenarios are in Section IV. Then, Section V gives an analysis of our proposal. Finally, we present our implementation of the SDSA mechanism in Section VI before concluding this paper in Section VII.

II. Related Work

The ingress filtering policy is applied on ISP border routers. This policy checks that a packet exiting a site is not lying about its identity by spoofing another machine's address. Consequently, the source address of the packet is checked to verify that it belongs to the ISP prefix pool. If not, the packet is dropped.

A. Provider Independent Addressing

Multihoming in IPv6 can be made in the same way than in IPv4. Namely, a unique prefix can be delegated to the site, directly by a Regional Internet Registry (RIR). This is independent from ISPs prefix pools. Thus, only one prefix is used to construct global addresses for all the machines in the site. Currently, Provider Independent (PI) prefix is used in IPv4 because it is not possible to have several addresses for an

interface. But important drawbacks of this practice in IPv4 can also be found in IPv6. The use of a unique PI prefix delegated by a RIR forces a multihomed site to have an agreement with all of its ISPs, so that the PI prefix is accepted by all ISPs (no ingress filtering on this PI prefix). Moreover, the PI prefix is not included in any of ISP prefix pools, so, to be reached, ISPs must collaborate with the site to advertise the specific prefix of the site in the core network with BGP (Border Gateway Protocol). These BGP announces participate to the increase of the core routing tables [10], which are already overloaded.

B. Tunneling to all ISPs

In an IPv6 multihoming environment with different delegated prefixes, a way to avoid ingress filtering is described in [3]. It involves the creation of tunnels between site border routers and ISP border routers. Each site border router is connected to every ISPs of the site, through a direct link (for one specific ISP) or through tunnels (for other ISPs). Then, each site border router checks the source address of an outgoing packet and forwards it to the ISP which has delegated the prefix of the source address. So, each outgoing packet is routed to the Internet through the correct ISP (no ingress filtering). However, as each site border router must create a tunnel to each ISP, this solution is complex to administrate, which makes it hardly appliable in home networks. In addition, this proposal needs a strong collaboration between the site and its ISPs to configure all the tunnels.

C. Locator / Identifier Split

A number of proposals support the decoupling of the locator and the identifier part of an IP address (known as Loc/ID split). The locator part is used to route the packet in the core network and the identifier is used for routing in the site network. However, a mapping between an identifier of a machine and one or several of its locators is required. With this technique, border routers have the specific role to exchange the identifier and the locator of each machine. In the site network, the identifier is used to route a packet from the source machine to the site border router. Then, the site border router uses the locator to forward the packet through the core network. The border router chooses an IPv6 address depending on the ISP for the source address of the packet and forwards it to the core network. Thus, no packet is dropped due to ingress filtering. Two kinds of Loc/ID split are under active research: *mapping and encapsulation* and *address rewriting*.

Mapping and encapsulation proposals append to IP packet a tunnel header with locator addresses to route the packet on the core network. LISP [4] is an encapsulation proposal and associated mapping proposals are described in [5], [6] or [7]. The encapsulation solution increases the overhead of each packet and can lead to fragmentation and path MTU issues. Moreover, the creation of a new architecture (mapping system) to map an identifier with its locators is a constraint. The mapping system to create and deploy entirely on the Internet makes the task harder.

The approach of address rewriting directly modifies addresses in the existing IP header, rewriting identifiers in locators and the contrary, like in [8] and [9]. With address rewriting, no new header is added to the original packet but a mapping system is still needed to know locators/identifiers correspondance. Consequently, this approch has the same drawback as the mapping and encapsulation approch.

D. Source Address Dependent Routing

An interesting approach to solve the ingress filtering issue in an IPv6 multihoming environment is described in the Source Address Dependent (SAD) routing proposal [11]. SAD routing proposes to have several routing tables on each router. Each routing table is dependent on a source prefix delegated to the site by an ISP.

In a SAD site, a border router, directly connected to one ISP, propagates its own routing information associated with the ISP delegated prefix. A SAD router processes this routing information and populates the routing table associated with the ISP delegated prefix. All border routers send their own routing information and all SAD routers populate different routing tables. To process a packet, a SAD router firstly checks its source address. Then, the router chooses the routing table associated with that source address. Finally, the destination address of the packet is compared to entries of the selected routing table and the packet is forwarded to the next hop. With this proposal, packets are routed to their destination through the correct ISP. However, several routing tables on each router highly increases the memory space needed for routing information. The construction and maintenance of such tables is time-consuming and all routes to site destinations are duplicate on each table. Due to their independence on the source address, this redundancy of local information is unnecessary. Moreover, we think that specific routes advertised by the ISP do not need to be associated with a source address. This particular point is discussed in Section V-C.

III. Selection of the Default Route Based on the Source Address

In this section, we propose the *Selection of the Default route based on the Source Address* (SDSA) to solve the ingress filtering issue in multihomed networks. We introduce the general idea of SDSA and precise the new routing table architecture. We describe the processing of a packet by an SDSA router and detail changes in routers and protocols to work with SDSA.

SDSA is a forwarding mechanism which leads to some changes in routing table architecture and in route diffusion. In the IPv6 multihoming environment, terminal endpoints have several global addresses. Inside the site, we propose to take the source address into account in the forwarding decision for outgoing packets. Each SDSA router has multiple default routes in addition to its classical routing table. Each default route is associated with a global prefix delegated by one of the site ISPs. So, there are, at least, as many default routes as there are ISPs.

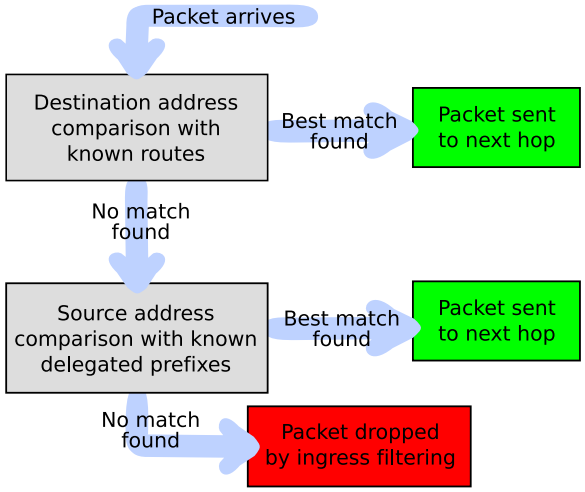

Figure 1. SDSA algorithm

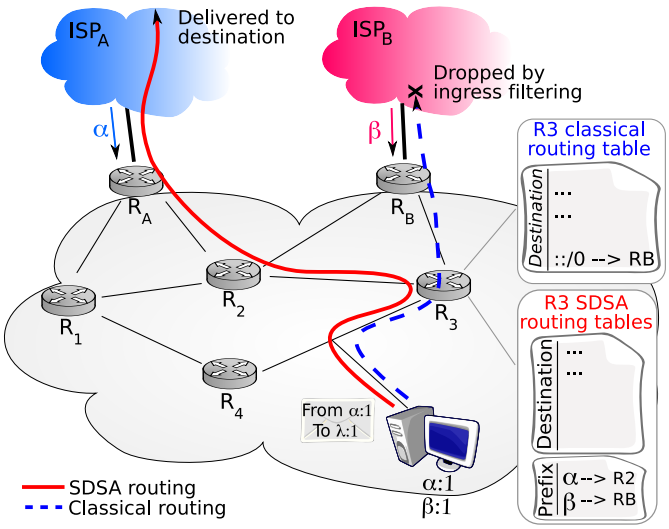

Figure 2. SDSA routing compared to classical routing

A. Packet Processing

The packet processing by an SDSA router is slightly different from the processing of a classical router to take into account the source address. For all the traffic to a known destination, routers perform destination based routing. SDSA occurs only when a terminal T_1 from a site A sends a message to terminal T_2 in a site B. The packet is forwarded until it reaches an SDSA router (see Section IV for deployment scenario). The SDSA router processes the packet following the algorithm in Figure 1. It checks the destination address and compares to all known routes (except the default route). If the destination address matches an entry, the packet is sent to the next hop and the algorithm ends. If not, the source address is checked and compared to the list of ISP delegated prefixes. If a match is found in this list, the packet is sent to the associated next hop. If there is no match, the packet is dropped, considered as a trial of address spoofing. With this process, a packet whom destination is unknown, is routed to the ISP border router which has delegated the source address prefix.

B. Evolution Implied by SDSA

To select the default route based on the source address, first, we need to change the routing table structure of SDSA routers to accept several default routes. Second, to populate this new routing table, the diffusion of routes has to support some modifications.

1) Routing Table Structure: Routing tables are separated in two complementary tables. The first one, which we call the *destination table*, contains all routes that we find currently in a routing table, except the default route. The entries of the *destination table* are mostly internal prefixes and some specific external prefixes announced by ISPs. Next hops are specified as in current routing tables. This table is used for the destination address check. The second table, called the *prefix table*, is populated with prefixes delegated by ISPs. Each prefix of this table has an associated next hop which will be used to route packets along a path to the ISP which has delegated the prefix. The *prefix table* participates in the source address check when there is no match during the destination address check.

2) Multiple Default Routes Diffusion: Each border router announces the same information as legacy routing systems and the prefix delegated by its directly connected ISP. SDSA site routers receive these announces (from different border routers) and construct their routing tables (*destination table* and *prefix table*).

Figure 2 shows a comparison between an SDSA routing scheme and a classical one. In this example, we show routing tables of R_3 only, with and without SDSA, but all routers can use the SDSA mechanism. We notice that several prefixes are in the *prefix table* of R_3, each one corresponding to a default route to a specific ISP. ISP_A (respectively ISP_B) delegates prefix α (respectively β) to R_A (respectively R_B). Routers R_A and R_B advertise their routing information (knwon routes and delegated prefix) to the site. SDSA routers use this routing information to populate their *destination table* and their *prefix table*. A packet to a destination address λ:1 from a source address α:1 is dropped by ISP_B in a classical routing scheme (blue dashed arrow in Figure 2), whereas the packet is forwarded to router R_A and sent through ISP_A until its destination, in the SDSA scheme (red arrow in Figure 2).

IV. Deployment of SDSA

As described in Section III-A, if the destination address does not match any entry in the *destination table*, the source address is compared to prefixes in the *prefix table*. The SDSA mechanism is only efficient if SDSA routers are aware of all prefixes delegated by site ISPs. Indeed, if an SDSA router does not know all prefixes, a source address check could have no match in the *prefix table* even if the prefix of the source address has been delegated by one of the site ISPs. So, each SDSA router has to receive routing information from each ISP of the site; particularly, the border routers.

As all routers involved in SDSA mechanism have to be aware of all prefixes delegated by ISPs, ***a necessary condition to use the SDSA mechanism is that there is a unique***

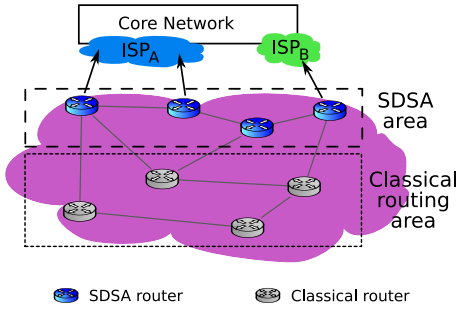

Figure 3. Necessary condition to an SDSA deployment

connected graph of SDSA routers including, at least, all border routers (as shown in Figure 3).

A. Partial SDSA Site

As SDSA requires router modifications in a site, it must have the possibility to be progressively deployed. In that purpose, SDSA does not need to be run on all site routers immediately and brings benefits even if few routers are using it.

In a partial deployment scheme, all routers are not using the SDSA mechanism, but it is possible to go to one border router to another along a path of SDSA routers (see Figure 3). A packet sent by a terminal host to an external destination is potentially processed by a router that does not use SDSA. Following the default route of each classical router, the packet is necessarily forwarded to an SDSA router. Then, the packet is routed by this SDSA router to the appropriate edge router (associated with the source address prefix). An evident drawback of this deployment method is that the path followed by the packet from source to edge router is not necessarly the shortest. The worst case occurs when the packet is routed by classical routers to a wrong edge router. Then, this edge router uses the SDSA mechanism to forward the packet on a path to the correct edge router. This increases the delay to deliver the packet to the destination. On the other hand, considering that, without SDSA, the packet would have been dropped by ingress filtering at ISP edge router, the deployment of SDSA in the site remains a positive evolution.

B. Total SDSA Site

In a total SDSA deployment scheme, all routers use SDSA, and are aware of ISP prefixes. A packet sent by a site machine to an external destination is directly processed by an SDSA router. Therefore, the packet is forwarded along a route to the border router which has advertised the prefix of the packet source address. Thus, the path followed by the packet from the source to correct ISP edge router is the best possible as the SDSA process is made as close as possible to the source. The deployment of the SDSA mechanism in all routers of a multihomed site not only solves the ingress filtering issue, but also makes internal routing as optimal as with current routing protocols.

destination address	source address: has an α-prefix	source address: has a β-prefix
Specific routes of R_A	delivered to ISP by R_A	dropped by ISP_A + Dest. Unreach. ICMP
Specific routes of R_B	dropped by ISP_B + Dest. Unreach. ICMP	delivered to ISP by R_B
External destinations	delivered to ISP by R_A	delivered to ISP by R_B

Table I
PACKET DELIVERY RESULTS IN SDSA SITES IN FUNCTION OF SOURCE AND DESTINATION ADDRESSES

V. SDSA ANALYSIS

We show a theoretical comparison of delivery results in a site with and without SDSA routing. We present SDSA time complexity to build and update routing tables and SDSA memory complexity.

A. Comparison with Classical Routing

All routers of a multihomed site have three kinds of routes in the routing table :

- Routes to destinations inside the site (*internal destinations*)
- Routes to external destinations advertised by a specific ISP (*ISP specific destinations*)
- Default route (corresponding to all external destinations)

Table I presents the delivery status of a packet from an SDSA site to different destinations in function of packet source address. This table uses the notations of Figure 2 for ISPs and router names as well as delegated prefixes. *Internal destinations* are not presented because the routing procedure in this case does not lead to any ingress filtering issue (the traffic is inside the site and ingress filtering only occurs at ISP borders). Table I also precises which border router has delivered the packet to the ISP network.

Thanks to the two first lines of Table I, we can notice that only some packets to ISP specific destinations are dropped. On the contrary, we note that, with SDSA, for all external destinations, there is no packet filtered by ingress filtering.

With SDSA, ingress filtering occurs on packets to ISP specific destinations if the source address is incorrect. But, [12] defines a specific Destination Unreachable ICMPv6 error code to inform that the *source address failed ingress/egress policy* (code 5). The terminal, who initially sent the packet, receives the ICMPv6 error message and sends the message again, using another source address. It retries until the selected source address corresponds to the prefix delegated by the ISP to go through. The packet is finally delivered, but, after several attempts. For classical routing, we obviously have the same result. So, we can conlude that for the case of ISP specific destinations, SDSA does not improve the routing; but does not worsen it.

The last line of the Table shows that all packets to *external destinations* are transmitted to the ISP network, *i.e.* are not filtered by ingress policy. As routers do not know the destination address of the outgoing packet, routers forward

the packet using the prefix table. As a consequence, a packet to an *external destination* is ensured to be forwarded to the correct ISP. On the contrary, in a site which do not use SDSA, the delivery of that packet depends on the topology and the default route on each router. So, for a terminal host, if the default route guides packets to ISP_A, then all packets with β source address are dropped, due to ingress filtering (same way of thinking with α prefix and default route guiding to ISP_B). In home networks, considering that most of the traffic sent by a terminal is to a locally-unknown destination (*e.g.* requests to the world wide web), we can conclude that the SDSA mechanism is more effective than classical routing in most of the cases.

Finally, we can deduce from Table I that, on an SDSA site, delivery results do not depend on the topology. As a function of packet source and destination addresses, we can know precisely if a packet will be dropped by ingress filtering or not, without any hypothesis on the network topology. On the contrary, in a site which do not use SDSA, the delivery of that packet depends on the topology and the default route on each router. It is not possible to foresee if a packet will be dropped without any information on the network topology. As a consequence, the SDSA mechanism makes more flexible the home network topology, as the topology can change, packets to *external destinations* will not hit ISPs ingress filtering policy.

B. SDSA Complexity

As described in Section III-B1, in the SDSA architecture, only two trees have to be constructed. The first for all known destinations (*internal destinations* and *ISP specific destinations*), the second for prefixes delegated by ISPs. For all calculations, following notations are used:

- k is the number of different prefix lengths in *destination table*
- l is the number of different prefix lengths in *prefix table*
- m is the number of ISPs
- n_0 is the number of internal routes
- n_i is the number of ISP$_i$ specific routes

According to [13], in Radix Trees schemes, the Buildup Time Complexity (BTC) is in $O(nk)$ in a table containing n entries with k different lengths and the Space/Memory Complexity (SMC) of such a table is in $O(n)$. For the SDSA proposal, the global BTC is the sum of the BTC of each table. The BTC of the *prefix table*, with above notations, is $O(m \cdot l)$. The BTC of the *destination table* is $O\big(n_0 + \sum_{i=1}^{m} n_i) \cdot k\big)$. Similarly, the global SMC is the addition of each table SMC.

Consequently, the SDSA Buildup Time Complexity is given by:

$$BTC_{SDSA} = O\Big(m \cdot l + (n_0 + \sum_{i=1}^{m} n_i) \cdot k\Big)$$

The Space/Memory Complexity for a SDSA routing table is given with the following formula:

$$SMC_{SDSA} = O\Big(m + n_0 + \sum_{i=1}^{m} n_i\Big)$$

Taking into account the number of links in the home networks (that are addressed as many times as the number of ISPs), the number of ISP specific routes advertised and the IPv6 multicast addresses, we can easily make the assumption that, the number of ISPs a home network is connected to, is greatly lower than the number of known routes in that home network. Because of the previous assumption, it is obvious that the number of different prefix lengths in *destination table* is bigger than one in *prefix table*. But, as it is difficult to know how big it is, we assume that numbers k and l are equivalent. So, we can simplify previous equations as follows :

$$BTC_{SDSA} = O\Big(k \cdot (n_0 + \sum_{i=1}^{m} n_i)\Big) \quad (1)$$

$$SMC_{SDSA} = O\Big(n_0 + \sum_{i=1}^{m} n_i\Big) \quad (2)$$

The SDSA mechanism has a BTC and an SMC equivalent to those of a classical routing mechanism.

C. SDSA compared to Source Address Dependent routing

Both the Source Address Dependent (SAD) routing, described in Section II-D and the SDSA proposal, presented in this paper focus on solving the ingress filtering issue. These approches are very close but some differences will be pointed out. As explained in Section II-D, SAD proposes to have a specific routing table associated with each delegated prefix. On the other hand, SDSA aims at only creating two routing tables: the *destination table* and the *prefix table* (see Section III).

1) ISP specific routes: When a packet is processed, the source address is checked first, to select the associated routing table, then the destination address is compared with elements of that table. That behaviour can create some problems with ISP specific destinations. When a terminal host sends a packet with an α source address to a specific destination advertised by ISP_B (notations of Figure 2). That packet is forwarded along the ISP_A default route until it reaches the border of ISP_A network, because the routing tables associated with α do not contain ISP_B specific routes. So, the packet exits the home network through ISP_A. At that stage, we can envision several problems :

- ISP_A does not know how to reach that specific destination because it is a service delivered to ISP_B customers only and not reachable by the Internet. The packet is discarded and a *no route to host* ICMPv6 error message is sent to the source.
- ISP_A can forward the packet to the destination in ISP_B network, but it is more costly than the site sending it directly to ISP_B.
- ISP_A can forward the packet to the destination in ISP_B network, but the path used is longer than the direct path between the customer and IS_B, so there is a delay for each sent packet.

In the first case, the terminal host can retry to send the packet, using another source address. In the two other cases, as the packet is transmitted, even with a certain cost or delay, there is no reason for the terminal host to try another source address.

On the contrary, in the SDSA mechanism, ISP specific routes are not bound to an ISP delegated prefix. So, regardless of the source address, packets to such a specific destination are routed to the ISP which has advertised it. Ingress filtering

may or may not occur, regarding to the selected source address of the packet. In the case it occurs (destination is an ISP specific route and source address is not in the ISP prefix pool), the Destination Unreachable ICMPv6 error message, code 5 indicates to the source host that the source address is rejected The source host can send the packet again, with another source address. After the necessary number of attempts, the connection can be established directly between the terminal host and the specific destination, through the ISP. So, SDSA has better performance regarding to the global delay or cost for a flow to an ISP specific destination.

2) Complexity differences: A SAD router builds as many routing tables as the number of ISPs, which means the same number of radix trees. In addition, another radix tree is constructed for the ISP delegated prefixes. If we calculate the Buildup Time Complexity and the Space/Memory Complexity required by the SAD proposal, using notations of Section V-B, we have:

$$BTC_{SAD} = O\Big(m \cdot l + \Big(\sum_{i=1}^{m}(n_0 + n_i)\Big) \cdot k\Big)$$

$$SMC_{SAD} = O\Big(m + \sum_{i=1}^{m}(n_0 + n_i)\Big)$$

We can make the same assumptions as previously : $k \sim l$ and $m \ll \sum_{i=1}^{m}(n_0 + n_i)$, and we obtain :

$$BTC_{SAD} = O\Big(k \cdot (n_0 + \sum_{i=1}^{m} n_i) + (m-1) \cdot n_0 \cdot k\Big) \quad (3)$$

$$SMC_{SAD} = O\Big(m + \sum_{i=1}^{m} n_i + n_0 \cdot m\Big) \quad (4)$$

Comparing Equation (1) with Equation (3), we note that the BTC of SAD is greater than the one of SDSA. This difference increases with the number of ISPs (m), the number of internal routes (n_0) and with the number of different prefix length. In the same way, Equations (2) and (4) show that the SMC of a routing table in a SAD site is increased by $n_0 \cdot (m - 1)$ in comparison with an SDSA routing table. So, we can conclude thanks to the previous results that as the site becomes bigger, complexity becomes better for SDSA rather than SAD. Consequently, in a bigger network site, it is more efficient to use SDSA than SAD.

VI. SDSA Implementation

We have implemented the SDSA mechanism on a testbed. That development was conducted in two steps. The first step considered the necessary changes in a routing protocol implementation. Those changes allow basic operations on source prefixes information (what we called *prefix table* in Section III), like adding, removing, and diffusing it to other routing entities. We chose the Quagga suite (version 0.98.6) as routing protocol implementation and implemented changes on RIPng module. The second step consisted of changes in Forwarding Information Base (FIB), to manage the new source prefixes information, and use those prefixes in the forwarding process. We implemented this part on Ubuntu 8.04, kernel version 2.6.24.

A. Routing Protocol modifications

As explained in Section III, we need the routing protocol to diffuse information about destinations and prefixes delegated by ISPs. We slightly modified the Routing Information Protocol to accept a new type of address that we call *source route*. To ease the implementation, we did not create a separate table to be populated with those *source routes*, but we inserted them in the current Routing Information Base (RIB). To make the difference between destinations and source routes, we set a high metric value to source routes. In the Quagga RIPng implementation, the metric is coded on 8 bits. So, we added the constant value of 224 to the metric of *source routes*, so that metrics of *source routes* are between 225 and 239. [14] specifies that the RIPng metric is an integer between 1 and 15 (inclusive). So, to authorize such values, we modified the restriction on metric values to accept values we specified (between 225 and 239).

We created instructions in Quagga RIPng implementation to add and delete *source routes*. These instructions are :

```
# route source <IPv6_addr> <IPv6_NH> <Iface>
# no route source <IPv6_address>
```

The first instruction create a new entry in the RIB :

- `<IPv6_addr>` indicates the prefix to advertise and is equivalent to a *source route*
- `<IPv6_NH>` specifies the next hop to forward packets with the specified source address
- `<Iface>` indicates the interface to use to forward packets with the specified source address

This instruction is intended to be used only on network edge routers and the specified next hop has to be the address of the provider edge router. So, only one *source route* per delegated prefix can be created. If a new *source route* is added for an existing prefix, but with other characteristics (next hop or interface), the new *source route* is rejected and the command returns an error. It explains why the *source route* deletion instruction does not need next hop or interface precision.

These code modifications allow each router running Quagga RIPng implementation to exchange information about delegated prefixes.

Quagga uses the Zebra module to accumulate information from its different routing modules (RIPng, OSPF, BGP, ...). During this process, we distinguish *source routes* from other routes, thanks to two criteria. First, the origin of the route, stored in the `rib->type` parameter, must be the RIPng module. Second, the `metric` parameter must have a value between 225 and 239. Then, the Zebra module communicates with the Linux kernel to update the Forwarding Information Base.

B. Kernel modifications

The first change in router behaviour is to accept and use in the FIB, the new kind of routes we defined in Section VI-A.

The flag `rt6i_protocol` is used in FIB to distinguish the origin of the routes (*e.g.* Zebra, DECnet, router advertisment,...). The Zebra module communicates with the kernel

```
Kernel IPv6 routing table

Destination                                    Next Hop                    Flags  Metric  Ref       Use Iface
::1/128                                        ::                          U      0       22          3 lo
2001:660:7301:49:20f:1fff:fe7c:f467/128        ::                          U      0       0           1 lo
2001:660:7301:49::/64                          ::                          UA     256     428928      0 eth1
2001:660:7301:d00:20f:1fff:fe7c:f467/128       ::                          U      0       0           1 lo
2001:660:7301:d00::/64                         ::                          UA     256     60978       0 eth1
fe80::20f:1fff:fe7c:f467/128                   ::                          U      0       15          1 lo
fe80::2a0:24ff:fead:495a/128                   ::                          U      0       13085       1 lo
fe80::/64                                      ::                          U      256     0           0 eth1
fe80::/64                                      ::                          U      256     0           0 eth2
ff02::1/128                                    ff02::1                     UC     0       3951563   255 eth1
ff02::9/128                                    ff02::9                     UC     0       8           0 eth2
ff00::/8                                       ::                          U      256     0           0 eth1
ff00::/8                                       ::                          U      256     0           0 eth2
::/0                                           fe80::219:56ff:fe49:8e99    UGDA   1024    2           0 eth1

-----------------------------------------------------------------------------------------------------------
Source Prefixes                                Next Hop                    Flags  Metric  Ref       Use Iface
2001:66::/32                                   fe80::219:56ff:fe49:8e99    UG     1       0           0 eth1
2001:660:7301:149::/64                         fe80::219:56ff:fe49:8e99    UG     1       0           0 eth1
```

Figure 4. SDSA Routing Table

to maintain known routes in the FIB, using `rt_netlink.h` library. The routes coming from Zebra - RIB are marked with the value `RTPROT_ZEBRA`.

We modified this behaviour to distinguish *source routes* from others in the kernel FIB. In that purpose, all *source routes* are marked with a new value : `RTPROT_SOURCE` (temporarily allocated to 31). Like in the Quagga implementation, we choose to insert *source routes* directly in the tree created for the FIB. We can note that the Forwarding Information Base consists of a radix tree, each leaf representing a route (destination or source). As each *source route* leaf represents a prefix delegated to a terminal site, it is impossible to have a destination route with the same value as a *source route*. Indeed, a destination route to a subnet of the terminal site is necessarily more precise than its related *source route*. So, there is no limitation to use the same tree.

To keep *source routes* consistent when they are inserted in the FIB, we modified the metric back to a normal value (between 1 and 15), subtracting the constant value 224 added in the RIPng process. So, we have a unique routing tree, with destination routes and with *source routes* that we can identify with the `RTPROT_SOURCE` flag. And we can use already existing kernel routines (add, update next hop, lifetime, interface, and so on) to update FIB with *source routes* entries.

The main modification in the router kernel code is the route lookup mechanism. More precisely, how the kernel look for and find the FIB entry for a packet. When a packet arrives, the kernel looks in the routing tree to find an entry matching the destination address of the packet, using the `fib6_lookup_1` function. If there is no matching entry, the root of the tree is returned, corresponding to the default route. It is important to note that our implementation does not suppress the default route from the FIB tree, but it is not used. When the default route is the only element of the tree matching the destination address of the packet, we add another lookup function called `fib6_lookup_2`. That function makes another lookup in the tree to find an element matching the source address of the received packet. When there is a match, the function checks for the `RTPROT_SOURCE` flag. If it is not set, the function checks the parent element of the matching entry to find the `RTPROT_SOURCE` flag. The process goes on until a *source route* element is found, then the router forward the packet to the associated next hop. If the `fib6_lookup_2` function returns the root element of the routing tree, it means that the source prefix used in the source address of the packet is not known by the SDSA mechanism. So, the packet is considered as a trial of address spoofing and is discarded by the router (*that functionnality has not been implemented yet*).

The last modification to the kernel concerns the activation/deactivation of the SDSA mechanism on the router. We appended a new flag called `accept_sdsa` in `/proc/sys/net/ipv6/conf/<interface>/` directories. Set that flag to 1 makes the router to use the SDSA mechanism; set to 0, it deactivates the mechanism. This is particularly usefull to highlight differences between routing process with and without SDSA on our testbed.

VII. Conclusion

Our proposal resolves in a simple way the ingress filtering issue in IPv6 multihomed networks with minimal changes in current behaviour of networks. Precisely, we demonstrate that it is not necessary to overload routing tables of site routers with different external routes because of the source address routing behaviour of our proposal. Only a unique default route per ISP is needed to lead a packet from its source to the site exit router. Moreover the progressive deployment capabilities of SDSA give immediate benefits to first adopters (*e.g.* no ingress filtering issue, no bottleneck), even if the deployment is partial.

Our proposal provides a solution to one of the multihoming issues. Our platform allows to verify the correctness of the SDSA proposal to solve ingress filtering issue. Future work is intended to define and measure metrics on our platform to compare SDSA routing mechanism and legacy routing protocols performance. Moreover we aim to study the combination of our proposal with other solutions for providing an answer to different multihoming issues. In this perspective, the combination of SDSA with SHIM6 [15] or SCTP [16], creating connection context and allowing session survivability, are envisioned.

Furthermore, multihoming and mobility capabilities present very interesting common points, a study of the SDSA behavior in a mobile environment could be an interesting way to explore.

References

[1] P. Ferguson and D. Senie, "Network Ingress Filtering : Defeating Denial of Service Attacks which employ IP Source Address Spoofing," RFC2827, May 2000. [Online]. Available: http://tools.ietf.org/html/rfc2827

[2] F. Baker and P. Savola, "Ingress Filtering for Multihomed Networks," RFC3704, Mar. 2004. [Online]. Available: http://tools.ietf.org/html/rfc3704

[3] J. Hagino and H. Snyder, "IPv6 Multihoming Support at Site Exit Routers," RFC3178, Oct. 2001. [Online]. Available: http://tools.ietf.org/html/rfc3178

[4] D. Farinacci, V. Fuller, D. Oran, and D. Meyer, "Locator/ID Separation Protocol LISP," Nov. 2007, draft-farinacci-lisp-05.

[5] E. Lear, "NERD: A Not-so-novel EID to RLOC Database," Sep. 2007, draft-lear-lisp-nerd-02.

[6] S. Brim, N. Chiappa, D. Farinacci, V. Fuller, D. Lewis, and D. Meyer, "LISP-CONS: A Content distribution Overlay Network Service for LISP," Nov. 2007, draft-meyer-lisp-cons-03.

[7] D. Farinacci, V. Fuller, and D. Meyer, "LISP Alternative Topology (LISP-ALT)," Nov. 2007, draft-fuller-lisp-alt-01.

[8] C. Vogt, "Six/One Router — Design and Motivation," 2008, vogt-2008-six-one-router-design.pdf.

[9] M. Menth, M. Hartmann, and D. Klein, "Global Locator, Local Locator, and Identifier Split (GLI-Split)," 2008, http://www3.informatik.uni-wuerzburg.de/staff/menth/Publications/papers/Menth08-GLI-Split.pdf.

[10] T. Bu, L. Gao, and D. Towsley, "On characterizing bgp routing table growth," *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 3, pp. 2185–2189 vol.3, Nov. 2002.

[11] M. Bagnulo, A. Garcia-Martinez, J. Rodriguez, and A. Azcorra, "End-site routing support for IPv6 multihoming," *Computer Communications*, vol. 29, no. 7, pp. 893–899 vol.29, Apr. 2006.

[12] A. Conta, S. Deering, and E. M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," RFC4443, Mar. 2006. [Online]. Available: http://tools.ietf.org/html/rfc4443

[13] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed ip routing lookups," in *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, 1997, pp. 25–36.

[14] G. Malkin and R. Minnear, "RIPng for IPv6," Jan. 1997, RFC2080. [Online]. Available: http://tools.ietf.org/html/rfc2080

[15] E. Nordmark and M. Bagnulo, "Shim6 : Level 3 Multihoming Shim Protocol for IPv6," Oct. 2007, draft-ietf-shim6-proto-09.

[16] R. Stewart, "Stream Control Transmission Protocol," RFC4960, Sep. 2007. [Online]. Available: http://tools.ietf.org/html/rfc4960