Design and Implementation of IPv6 Segment Routing

David Lebrun*, Stefano Previdi†, Clarence Filsfils†, Olivier Bonaventure*
*ICTEAM, Université catholique de Louvain
Louvain-la-Neuve, Belgium
†Cisco Systems

***Abstract*—Segment Routing is the modern variant of source routing being standardised by the Internet Engineering Task Force. It enables routers and endhosts to better control the path followed by their packets in the network. We first describe the utilisation of the IPv6 Segment Routing extension header and motivate the main design choices. We then explain how we have modified the Linux kernel to support this new header and evaluate its performance. Finally, we demonstrate our implementation on two very different use cases : parental control and firewall load balancing.**

I. Introduction

Several network protocols have included support for some forms of source routing. IPv4 supports a source routing option that allows a host to indicate intermediate hops on the path towards a given destination. Most IPv4 implementations support this form of source routing. However it is rarely enabled because of several security problems [2], and because endhosts have difficulties in obtaining accurate information about the path towards a given destination. Source routing was also a feature of the original IPv6 specification. IPv6 defines the Routing Header type 0 to encode a source routed path for a packet. IPv6 faced the same problems as IPv4 for the deployment and utilisation of source routing beyond labs and simple networks. From a security viewpoint, IPv6 did not solve the security problems that affected IPv4 source routing and serious attacks against IPv6 source routing were published [1]. Given the potential impact of these attacks, the Internet Engineering Task Force (IETF) eventually decided to completely deprecate the IPv6 type 0 Routing Header [1].

Despite of this, the ability to indicate inside each packet a set of intermediate hops remains a very useful feature to solve a wide range of networking problems [3]. Given the importance of these problems, network vendors and network operators convinced the IETF to create the Source Packet Routing in Networking (SPRING) working group to standardise a modern source routing solution. The first approach discussed within the SPRING working group leverages the MultiProtocol Label Switching (MPLS) dataplane. In a network using MPLS, routers receive information about the topology through the intradomain routing protocol (OSPF or IS-IS). Extensions to these protocols have been proposed to use them to distribute MPLS labels associated to routers or specific links[1]. With this information, any router in the MPLS network can select a source routed path towards any other router inside the network and encode it as a label stack which is attached to each packet. Various use cases are being defined and implemented with this MPLS dataplane in service providers networks.

The second approach pursued within the SPRING working group is to design a source routing extension for IPv6 [8]. IPv6 has a much broader applicability in the long term than MPLS which is restricted to service provider and some entreprise or datacenter networks. However, with IPv6 we cannot simply assume as in MPLS that only trusted routers will generate source routed packets. Any IPv6 node can generate source routed packets and a modern source routing extension for IPv6 must take security into account. In this paper, we explain the main design principles that underly the IPv6 Segment Routing extension header and report our experience with the first complete open-source implementation of this technology.

The paper is organised as follows. We first describe in section II the basic principles of the IPv6 Segment Routing extension. Then, we detail in section III our open-source implementation of IPv6 Segment Routing in the Linux kernel[2] with a performance evaluation and the lessons learned. Finally, we illustrate in section IV the benefits of IPv6 Segment Routing with two use cases that we prototyped on our implementation.

II. IPv6 Segment Routing

Segment Routing [5] enables to steer packets through an ordered list of instructions, which can be topological or service-based. These instructions are called *segments*. We distinguish two types of topological segments: *node segments* and *adjacency segments*. The presence of a node segment forwards packets through a specific network node. Conversely, the presence of an adjacency segment forwards packets along a specific link. An additional type of segment, the *service segment*, which is local to a node, represents a service to apply to the packet. This type of segment is important in order to support Service Function Chaining. An important point to note is that between two segments, the packets are still forwarded along the shortest path towards the next segment. Many IPv6 networks use Equal Cost Multipath (ECMP) to load-balance the traffic and Segment Routing is fully compatible with ECMP.

[1]Given the restriction on the number citations, we do not reference all IETF documents. A bibliography on Segment Routing is maintained at http://www.segment-routing.net

[2]Available from http://www.segment-routing.org

Consider Figure 1 for an illustration. Packets enter the network at router I and exit at router E. Consider that a packet enters the network and the ingress node pushes an SR header into the packet. We denote segments with the following notation: N_i segments are node segments corresponding to network node i, S_i^n segments are service segments, representing a service number n which is local to node i. Finally, A_{ij} segments are adjacency segments representing the link between node i and node j. The header pushed by node I is composed of the following segments: $N_D, N_B, S_B^0, N_F, A_{FE}$.

After having pushed the SR header, node I looks at the active segment, which is N_D, a node segment to network node D. The packet is forwarded along the shortest path from I to D, which is $I \rightarrow A \rightarrow D$. Then, node D looks at the next segment, N_B and again forwards it accordingly. Node B receives the packet and reads that the next segment, S_B^0, represents the service number 0 that it must apply. After having applied this service, node B looks at the next segment, which is a node segment to F. Node B thus forwards the packet to F. Finally, node F reads an adjacency segment from itself to node E and forwards the packet along this link and the packet exits the SR domain at node E.

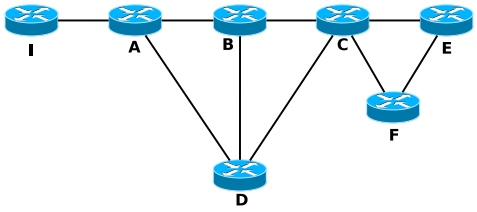


Fig. 1: Abstract Segment Routing domain. All links have a unit weight.

To enforce specific paths, the node that inserts the Segment Routing Header must know the segments that have been defined in the network. It can obtain this information through the intradomain routing protocol or through configuration from an SDN controller. For IPv6 Segment Routing, the segments are encoded as 128 bits IPv6 addresses. A node segment is represented by its loopback address. An adjacency segment is represented by an interface address. The IPv6 address of a service segment depends on the actual implementation. It could be a virtual machine, an hardware appliance, etc.

Since IPv6 addresses are already announced through OSPF or IS-IS, the support of SR-IPv6 inside a network does not require the deployment of yet another network protocol.

A. Segment Routing Header

In IPv4, options are encoded inside the IPv4 header that cannot be longer than 64 bytes. This poses a hard limit on the maximum length of an IPv4 source routed path. In IPv6, the header has a fixed length of 40 bytes. However, it contains the *next header* field which encodes the protocol number of the following header. Usually, the next header is the header of the transport protocol (e.g. TCP or UDP), but IPv6 supports different extensions that define their own header. An IPv6 extension header contains at least two fields, encoding the length of the extension, and the protocol number of the next header. In this way, it is possible to easily chain extension headers. The one of interest for us is the routing header (protocol number 43)[8].

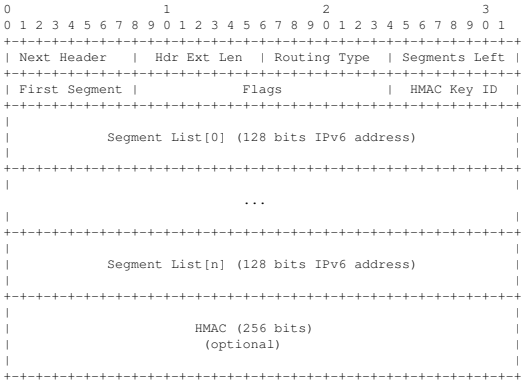


Fig. 2: Segment Routing IPv6 Extension Header

The Segment Routing header (SRH) is defined as an extension of the IPv6 Routing Header, with routing type 4 [8]. It is an evolution of the Routing Header type 0 (RH0), originally defined in [4], which has been deprecated for security issues [1]. The format of the SRH is shown in Fig. 2.

The SRH contains a list of segments (represented as IPv6 addresses), in reverse order. The last segment of the segment list is the first segment in the path, the penultimate segment of the segment list is the second segment in the path, and so on. The `Segments Left` field is the index of the current active segment in the segment list (e.g. if the packet is sent with n segments, then the `Segments Left` field has a value of $n-1$). The `First Segment` field has a fixed value and represents the first segment in the path. Its value is set as the index of the last segment in the segment list.

The `Flags` field contains one flag of interest for SR-IPv6: the *cleanup* flag. When set, the penultimate segment endpoint has to strip the SRH from the packet before forwarding it to the last segment. Thanks to this flag, routers can remove the SRH when the packet is transmitted to another network or to a host that does not support SR-IPv6.

The last field of the SRH is important from a security viewpoint. In an open environment like a university campus network, the network operator cannot assume that all hosts, including students' laptops, are trusted to only send packets containing a valid SRH. To enable routers to easily distinguish between valid (i.e. chosen by the network operator) and invalid SRHs, each SRH may contain a keyed-HMAC that is computed over a key and the SRH. If the HMAC is valid, the packet contains a trusted SRH and can be forwarded according to the SRH. Otherwise the packet is invalid and should be dropped. This mechanism addresses the security issues that led the IPv6 RH0 extension to be deprecated. Due to space limitations, we choose not to present the inner workings of the HMAC signature.

We expect that many SR-IPv6 deployments will be in networks were all routers are trusted and thus the HMAC authentication is not required. However, it could play an

important role in more exposed environments such as wireless mesh networks and public datacenters.

B. SR-IPv6 Operations

When a segment endpoint receives an SRH-enabled packet, it has to perform the following operations. First, it needs to inspect the SRH to verify its validity, and optionally, verify the HMAC signature if present. Then, the node decrements the `Segments Left` field and updates the IPv6 destination address of the packet to the next segment. If the *cleanup* flag is set and the `Segments Left` field reaches zero after decrement, the node strips the SRH from the packet. Finally, it forwards the packet to the next segment.

The SRH has to be inserted at some point in the network, by the *source SR node*. In theory, any node could perform this operation, but we mainly distinguish two types of source nodes: (*i*) a host originating an IPv6 packet and (*ii*) a *SR domain ingress* router encapsulating a received IPv6 packet into an outer IPv6 header, followed by the SRH.

When a packet is encapsulated by the SR domain ingress, it goes through a one-way virtual tunnel whose egress is the last segment of the path. When the packet reaches the tunnel egress, the outer header and the SRH are stripped from the packet and the inner packet is processed by the node. This mechanism allows an operator to transport other protocols than IPv6 (e.g. IPv4) as the inner packet.

Note that the SRH may also be inserted into the packet without adding an outer IPv6 header. This approach is not currently documented in [8]. However the industry is still working on it and supports it as long as the SRH is stripped before leaving the SR domain (as well as our implementation).

C. Advanced Operations

A segment endpoint can provide additional operations besides simple processing and forwarding, such as a service to apply to the processed packet. When it receives an SR-enabled packet, a segment endpoint inspects the active segment (i.e., the segment referenced by the `Segments Left` field), which is the current destination address of the packet. Then, it decides which service it should apply to the packet.

On the one hand, the service to apply can be simple like a modification of the packet header. For example, the SR node can swap the active segment by several other segments. This is called a *binding segment* [5]. Such a segment might be provided by a service provider to one of its customer who subscribed for a "fast lane" service. The ISP's ingress routers then recognize this *binding segment* and insert the appropriate node/link segments into the SRH, without the need for the ISP to disclose a part of its topology to the customers. Furthermore, it enables the ISP to provide a symbolic reference to its fast lane service, which is useful if its internal topology changes.

On the other hand, the service provided by the SR node can be much more complex. For example, a service could analyse or even modify the payload of the packet. Such a service would usually be performed by a dedicated entity, such as a hardware appliance or a virtual machine. It would be easy for an ISP to provide several such services that can be applied to a packet one after the other, and actually enforcing them by inserting the corresponding segments into the segment list. This provides a basis to perform Service Function Chaining.

III. Implementing IPv6 SR

We have implemented SR-IPv6 in the Linux kernel [3]. The currently supported kernel version is v4.4. In this section, we describe our implementation and show some performance results.

A. Implementation

As stated in subsection II-B, we need to support two basic operations: (*i*) forward a packet containing an SRH and (*ii*) inject an SRH in a packet. The second operation will typically be configured through access-lists to identify on which flows an SRH must be injected.

When the Linux kernel receives a packet, it first looks at the L3 protocol number contained in the L2 header. For example, in our case, the node will see the protocol number corresponding to IPv6. It looks up its translation table to find the function handler corresponding to the protocol and executes the IPv6 handler (`ipv6_rcv()`), which verifies the validity of the L3 header, then makes a routing decision. If the destination of the packet is the node, it calls the input processing function (`ip6_input()`). Otherwise, it calls the forwarding function (`ip6_forward()`) which does not inspect the packet any further.

When an SR-enabled packet is forwarded to a segment endpoint, it means that the destination address of the packet is the address of the segment endpoint. Thus, the packet will be processed by the `ip6_input()` function. This function follows the header chain of the IPv6 packets and stops once it reaches a supported protocol (e.g., TCP, UDP, ICMP) and calls the corresponding protocol handler for each header in the chain. Linux already provides a handler for the Routing Header extension (IPv6 *next header* value 43). This handler can process RH types 0 (deprecated source-routing header RH0) and 2 (IPv6 Mobility). We extend the handler to support type 4 (Segment Routing Header).

The handler then executes the processing operation described in subsection II-B. As the IPv6 destination address of the packet has been changed to the next segment, the node has to restart its routing decision process. The destination of the packet is no longer the node itself, thus the node stops processing the packet which goes through `ip6_forward()` and continues its journey through the network. We are now able to process SR-enabled packets and forward them to the next segment.

Now, we need to support SRH injection. There are two ways to perform this: either the SRH was already sent by the source of the packet, or the SRH is inserted by an intermediate router, encapsulating the packet in an outer IPv6 header. We start with the former.

When a local application transmits a packet, it is generated in a bottom-up manner. The upper layer data is constructed

[3]Our code and documentation are available from http://www.segment-routing.org

and the lower layer headers are stacked upon it. Just before adding the IPv6 header, the kernel checks if any IPv6 extension header needs to be inserted. The application can control this thanks to the `setsockopt()` system call. In this way, the application can specify the whole SRH to apply for all packets transmitted through a given socket.

Enabling a node to encapsulate a transiting packet requires a different technique. When a packet is encapsulated in an outer IPv6 header with an SRH, it can be considered to follow a virtual tunnel. There already exists encapsulation mechanisms in Linux (e.g., GRE) represented by virtual devices. The decision to use the device or not for a given packet is made by the routing engine. We reuse the same mechanism and provide a virtual device for the SR tunnels. Fundamentally, what we do is an IPv6-in-IPv6 encapsulation. Linux already provides such a mechanism through the `ip6tnl` module, however this module does not support the IPv6 extension headers. Thus, we have modified it as well as the `iproute2` tool to support the insertion of an SRH within an IPv6-in-IPv6 tunnel. Incidentally, we also support IPv6-in-IPv4 tunnels because they are handled by the same `ip6tnl` module.

In conclusion, we support the basic operations for SRH processing, SRH injection, and we provide control mechanisms in the form of the `setsockopt()` system call for local applications and a modification of `iproute2` for in-transit encapsulation. We have also implemented advanced operations such as service segments, but we choose not to present this part due to space limitations. Basically, our implementation allows to associate a segment to a virtual machine or a userland application. Our implementation of service segments is presented in [7].

B. Using IPv6 SR on endhosts and routers

In this subsection, we provide details on how endhosts and routers can be configured to use the correct segments.

Let us first consider a set of endhosts in an enterprise network. A backup application is installed on each host. This application uploads the important files to a server on a regular basis. However, the network operator does not want this backup traffic to interfere with production traffic. She wants to force backup flows to only use some parts of her network. In this case, a possible approach is to use the entreprise DNS server as an SDN controller. When the backup application requests an `AAAA` DNS record, the DNS server returns a new DNS record representing the segments that needs to be used in order to reach the given destination. With both the destination address and the segments, when the application creates its socket to connect to the server, it issues a `setsockopt` call to pass the SRH to the kernel. A code sample is shown in Figure 3. Thanks to this `setsockopt` call, the kernel knows that it must attach the given SRH to all packets transmitted through the socket.

This DNS solution cannot be used to configure routers that need to inject the SRH in packets towards some destinations. For this, the best approach is probably to distribute the required information through a BGP extension [9]. This BGP extension provides support for Segment Routing with MPLS but it is easily extendable to support IPv6 as well. The advantages of this solution are twofolds: first, it leverages BGP which is already used in many networks, and second, it enables to push SR policies to the routers as well as steer traffic through these policies. Indeed, other solutions such as PCEP [10] only provide a mechanism to install source routing policies on routers but not to steer traffic through them.

```
int fd, srh_len;
struct ipv6_sr_hdr *srh;

srh_len = build_sr_header(&srh,
                          segments);

fd = socket(AF_INET6, SOCK_STREAM, 0);
setsockopt(fd, IPPROTO_IPV6, IPV6_RTHDR,
           srh, srh_len);

connect(fd, ...);
```

Fig. 3: Sample code showing how to setup an SRH for a socket. The `segments` variable represents the list of segments retrieved by the application. Once the `setsockopt` call is issued, the specified SRH will be attached to each packet transmitted through the socket.

C. Performance evaluation

To evaluate the performance of our implementation, we consider low-end routers (TP-Link TL-WDR4900 [4]) that are equivalent to the broadband routers deployed by many ISPs. Our testbed was composed of two generic x86 servers (S_1 and S_2) and two routers (R_1 and R_2). They were physically connected with Gigabit Ethernet links in the following fashion: $S_1 \leftrightarrow R_1 \leftrightarrow R_2 \leftrightarrow S_2$. Both servers were running our SR-IPv6 implementation for Linux 4.4. The two routers used the same version of the kernel ported to OpenWRT [5]. Server S_1 was used as a traffic generator (with `iperf3`) and server S_2 was used as the sink. All packets were thus forwarded by the two routers. The servers are much more powerful than the two routers and we could saturate their packet processing capacity.

We ran four different experiments:

1) Routers forward packets with a one-segment SR header (the segment endpoint is R_1)
2) Routers forward and verify the HMAC of packets with a one-segment SR header
3) SRH encapsulation (R_1 inserts an SRH containing one segment S_2)
4) SRH Decapsulation (the SR egress is R_2)

Figure 4 shows the result of our experiments. In the top-left plot, we can see that the throughput for simple SR forwarding is very close to plain IPv6 forwarding. The lower average for 256-byte and 1024-byte packets is most likely caused by

[4]CPU PPC P1014@800MHz with 128MB RAM

[5]Available at https://github.com/segment-routing/openwrt.

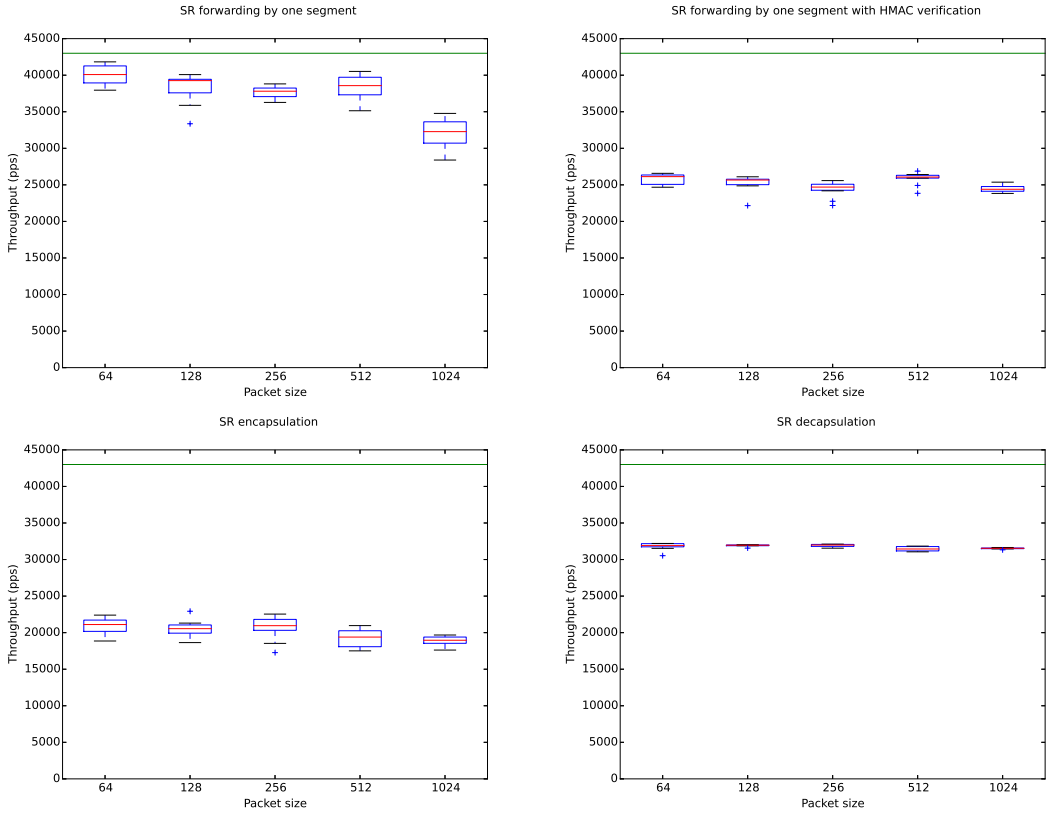


Fig. 4: Performance of our SR-IPv6 implementation. The green horizontal line on each plot marks the average throughput for plain IPv6 forwarding of 43,000 pps. The packet size is the actual payload size, without IPv6, SR and UDP headers. Each measurement ran for 10 minutes and each data point is the average throughput over 60 seconds.

the router's hardware limitations. The top-right plot shows the cost of verifying the HMAC field while forwarding SR-enabled packets. We can see a constant drop of about 35% of the packet processing speed. This could be improved with a per-flow HMAC cache in order to avoid recomputing the HMAC for each packet. In the bottom-left plot, we show the cost of encapsulating packets in an outer IPv6 header with an SRH. This is by far the most expensive SR operation with a performance impact of about 45%. This cost is inherent to the `ip6tnl` module and might be improved by using the Linux lightweight tunnels infrastructure that was recently merged into the mainline Linux kernel and whose purpose is to specifically add encapsulation instructions to routes [6]. Finally, in the bottom-right plot, we can see that the cost of SR decapsulation causes a performance drop of about 20%.

Overall, if we look at the worst performance of these experiments, which is encapsulating 1024-bytes packets at about 20,000 pps, we are still able to yield a throughput of 185 Mbps, which is large enough to saturate a 100 Mbps link. This corresponds to the target performance of our broadband routers.

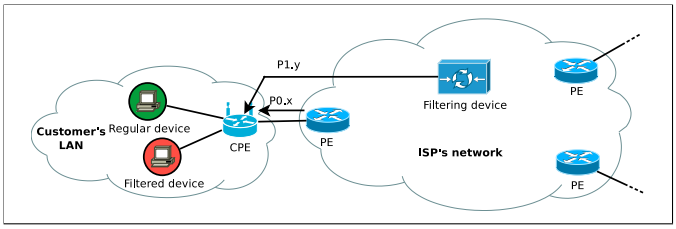
IV. Usecases

Various use cases have been discussed within the SPRING working group of the IETF [3]. Many of these use cases are valid for both MPLS- and IPv6-based dataplanes. Since MPLS is mainly used on routers and rarely used on endsystems, most of the MPLS use cases apply to routers while SR-IPv6 can also easily be used on endhosts and access routers. In this section, we describe two new use cases with IPv6 Segment Routing. Our first use case leverages the SR-IPv6 implementation on Linux-based access routers that are often deployed by service providers. The second use case applies to datacenters.

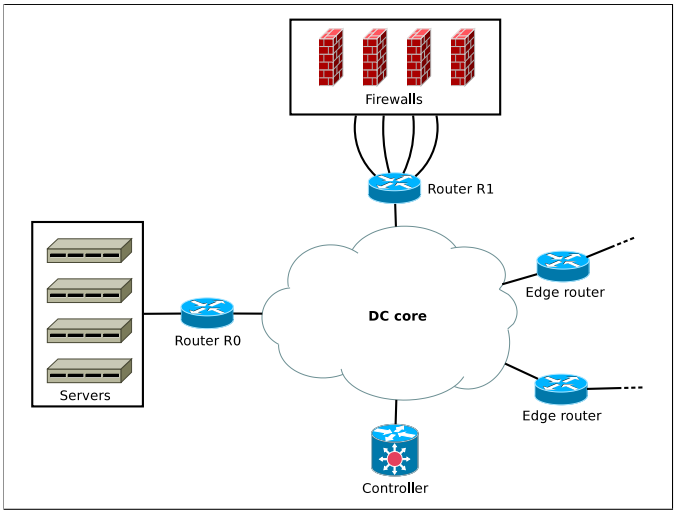
A. Network-provided parental control

Many network operators develop new added-value services that can be sold to their customers. Various operators offer storage services or video-on-demand to their customers by leveraging their cloud infrastructure. In this section, we illustrate how an operator could combine the large number of available IPv6 addresses and Segment Routing to provide new added value services. We use parental control as a motivating example, but other services could be designed based on the same principles.

Today's Internet allows users to access a wide range of contents. Some of this content is not suitable for young children and some parents have installed software on laptops used by their children to restrict their access to specific content. These services filter content based on keywords, feedback from users or more complex techniques. However, today's children often use their own tablet or smartphone and installing filtering software on all these devices can be complex and many parents



Scenario for the parental control use case.



Scenario for the firewall load-balancing use case

Fig. 5: Usecases

would prefer a value-added service provided by their network operator.

We assume that a home network contains two types of devices : parents-operated devices and children-operated devices. Parents use the GUI of their broadband router to tag the devices (e.g. based on its MAC address) that require content filtering. In an entreprise network, content filtering could be provided by software running on a server installed in the entreprise. Adding a new server on each home or placing the filtering software on the broadband router would be extremely costly as today's broadband routers have a low-performance CPU that cannot efficiently filter content.

Fig. 5 illustrates our scenario. The broadband router uses the MAC address of the device to identify whether it requires content filtering or not. The content filtering service is provided by servers placed in the operator's datacenters for scalability reasons. To provide content filtering for the children's devices, the network operator needs to ensure that packets to/from those devices pass through the content filtering server while packets to/from other devices take the direct path.

To deploy this content filtering service, we leverage the IPv6 address space and Segment Routing. The size of the IPv6 addressing space allows network operators to assign a `/64` or even a `/56` prefix to each broadband router serving a home.

To deploy the parental control service, the ISP divides its IPv6 address space in two large blocks : `P0` that is used for unfiltered access and `P1` that is used for filtered access. Inside the ISP network, prefix `P1` is advertised by the content filtering equipment (FE). When a broadband router is attached to the ISP network, it receives (e.g. through DHCPv6 prefix delegation) its main IPv6 prefix from the `P0` block, e.g. `P0.x`. In parallel, the FE delegates to the broadband router a prefix `P1.y` which is part of its global `P1` prefix. When a device whose access must be filtered connects to the customer's home network, it receives from the DHCPv6 server an address within the `P1.y` prefix. When the CPE has to forward traffic from filtered devices, it encapsulates the packets with an SRH containing a segment to the FE, which in turn applies its filtering rules and either forwards the packet or drops it. The reverse traffic, coming from the Internet to the filtered device, is automatically forwarded to the FE (as it advertises the `P1` block). Once the incoming packet passes the filter, the FE forwards it to the CPE by encapsulating the packet with an SR header containing a segment representing the CPE of the customer. Finally, the CPE removes the SRH and delivers the packet to the filtered device.

With this setup, we have successfully implemented an SR-IPv6-based parental control for an access ISP. Note that the ISP can provide other services with exactly the same mechanism. The only thing to change is the segment inserted by the CPEs.

B. *Datacenter firewalls*

Our second use case is a datacenter operator who wishes to firewall the incoming and outgoing traffic of all the servers behind a given router. To achieve this goal, the operator does not want to insert static rules, and the IP addresses of the servers may change at any time. Moreover, the traffic must be load-balanced among several firewalls to (*i*) lighten the load on the firewalls and (*ii*) use firewalls of different brands. Furthermore, the solution should guarantee that flows are routed through the same firewall in both directions.

Fig. 5 illustrates this scenario. The servers are connected to the datacenter network through router $R0$. The firewalls are behind router $R1$. This router is connected to each firewall through a dedicated link, but all firewalls use the same anycast IPv6 address : S_{fw}.

The links between $R1$ and the firewalls have the same IGP weight, thus enabling Equal Cost Multi-Path (ECMP) paths towards the firewalls. The datacenter also includes edge routers connected to various peers and an SDN controller.

To implement this use case, we first need to route the packets sent by the servers through the firewalls. In order to do so, we configure $R0$ to encapsulate the packets that it receives on the ports connected to the servers inside an outer IPv6 header containing an SRH. The SRH contains a segment with the anycast address of the firewalls (i.e. S_{fw}). Thanks to this segment, the packets will reach one of the firewalls. As $R1$ has multiple ECMP paths towards them, it will decide which path to use through its ECMP hash function. The selected firewall processes the packet and decides to drop or forward it and creates/updates the required state.

Now, we need to do the same for the ingress traffic. For this, the edge routers must be configured to route to the firewalls

the traffic to the servers. We reuse the same idea as for the previous use case and use the SDN controller to configure $R1$ to use an SRH to reach the firewalls on their way to the servers.

However, we cannot let the firewalls notify the controller when they receive egress traffic from the servers, because that would allow any ingress traffic to reach the servers as long as they have not transmitted anything. A better option is to let the servers announce themselves to the controller when they boot (or change of address). In that case, there is no need to manually configure $R0$, as the controller can do that automatically and dynamically as the servers announce themselves. Moreover, this option allows any device in the network to request firewalling, independently of the placement of the device in the network.

The requirements for this scenario are almost fulfilled. We still need to ensure that each flow goes through the same firewall for both directions. As the flows pass through the same router $R1$, we know that the same ECMP hash function will be used. We use the consistent hash function that is supported by the Linux kernel. Instead of simply hashing on the IP/ports source and destination pairs, this hash functions extracts the smallest IP/ports source and destination pairs from each packet and hashes them in arithmetic order. This ensures that packets of the same flow always use the same hash and are thus forwarded by $R1$ to the same firewall.

V. Conclusion

Segment Routing is a modern form of source routing that is being standardised within the Internet Engineering Task Force. Two variants of Segment Routing are being developed : an MPLS dataplane and an IPv6 dataplane. The former is mainly targeted at large ISP or datacenter networks. The IPv6 variant of Segment Routing can be used on both endhosts and routers that need to use specific paths. We have then described our open-source implementation of IPv6 Segment Routing in the Linux kernel and evaluated its performance. We have then illustrated two very different use cases where network operators can leverage our implementation to deploy new services for their customers. Given the interest in Segment Routing by the operator's community and the importance of Linux on both servers and embedded systems, we expect that other use cases will be tested and deployed in the future.

References

[1] J. Abley, P. Savola, and G. Neville-Neil. Deprecation of Type 0 Routing Headers in IPv6. Request for Comments, RFC5095, Dec. 2007.

[2] S. M. Bellovin. Security problems in the tcp/ip protocol suite. *SIGCOMM Comput. Commun. Rev.*, 19(2):32–48, Apr. 1989.

[3] Brzozowsk et al. IPv6 SPRING Use Cases. Internet draft, draft-ietf-spring-ipv6-use-cases-05, work in progress, Sept. 2015.

[4] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6). Request for Comments, RFC2460, Dec. 1998.

[5] C. Filsfils et al. Segment Routing Architecture. Internet draft, draft-ietf-spring-segment-routing-07, work in progress, Dec. 2015.

[6] T. Graf et al. Lightweight and flow-based tunneling. https://lwn.net/Articles/650778/, 2015.

[7] D. Lebrun. Leveraging IPv6 Segment Routing for Service Function Chaining. In *CoNEXT student workshop*, 2015.

[8] S. Previdi et al. IPv6 Segment Routing Header (SRH). Internet draft, draft-ietf-6man-segment-routing-header-00, work in progress, June 2016.

[9] A. Sreekantiah et al. Segment Routing Traffic Engineering Policy using BGP. Internet draft, draft-sreekantiah-idr-segment-routing-te, work in progress, Apr. 2015.

[10] J. Vasseur et al. Path Computation Element (PCE) Communication Protocol (PCEP). Request for Comments, RFC5440, Mar. 2009.