



TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA



consorzio nazionale
interuniversitario
per le telecomunicazioni



COMMON NET

SRv6 in the Linux kernel

Andrea Mayer

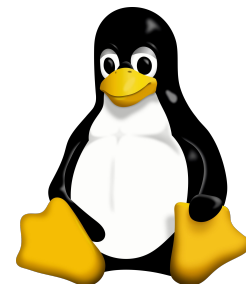
University of Rome Tor Vergata / CNIT / COMMON NET

Linux Netdev 0x19 - Zagreb, Croatia



- ❖ M.Sc degree in Computer Science and PhD in Electronic Engineering at University of Rome "Tor Vergata";
- ❖ Cooperates as a research engineer with:
 - CNIT (National Inter-University Consortium for Telecommunications);
 - University of Rome "Tor Vergata";
 - COMMON NET (Italian Internet Service Provider and Cloud Service Provider);
- ❖ Main interests focus on Linux kernel networking stack, IPv6 Segment Routing (SRv6), eBPF/XDP networking, Software Defined Networking (SDN);
- ❖ Developer and contributor to the SRv6 subsystem of the Linux kernel.

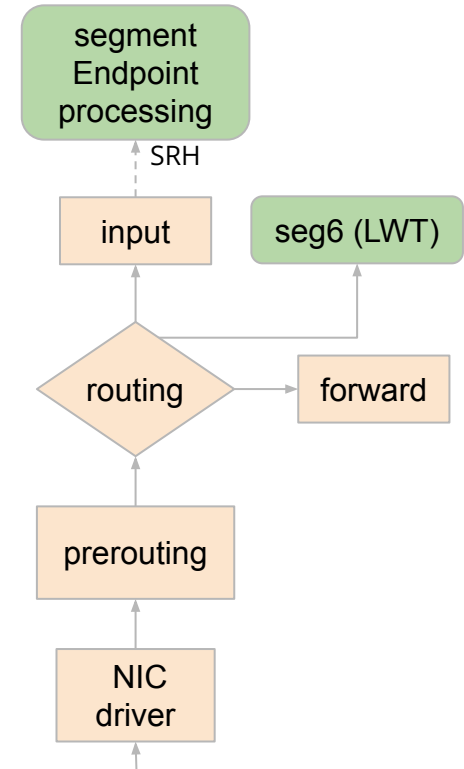
- ❖ **Quick journey** through the evolution of SRv6 in the Linux kernel:
 - SRv6 initial support (v4.10 ~ v4.18);
 - **Our contributions** from v5.0 up to now;
 - Kernel space/User space (i.e., `iproute2`).
- ❖ What's next?
- ❖ Conclusions.





4.10

- ❖ Support for SRv6 appears in Linux kernel v4.10;
- ❖ Implement **minimal support** for processing of SR-enabled packets:
 - Add SRH encapsulation and insertion (`seg6 - LWT`);
 - segment Endpoints processing (**require** SRH and DA = local):
 - Advance the next segment and re-routing;
 - Egress for encap packets: remove of outer IPv6 and SRH;
 - HMAC support.
- ❖ Endpoint enabled through *per-netns* **sysctl** knobs:
 - `net.ipv6.conf.default.seg6_enabled`
 - `net.ipv6.conf.<ifname>.seg6_enabled`





❖ iproute2 extended to:

- Support for SRv6 encapsulation (T.Encaps)* and insertion (T.Inserts) **;
- Set the source tunnel address;
- Handle the HMAC.

❖ Some examples:

- To encapsulate an IPv6 packet into an outer IPv6 + SRH:

```
$ ip ro add 2001:db8::1 encap seg6 mode encap segs fc00::1,fc00::2 dev eth0
```

Diagram illustrating the command components:

- `2001:db8::1`: Dst of packets that will be encapsulated
- `seg6`: seg6 LWT
- `mode encap`: encapsulation mode
- `segs fc00::1,fc00::2`: Segments belonging to the SID List

- To set the SRv6 tunnel source address (once per network namespace):

```
$ ip sr tunsr set 2001:2::1
```

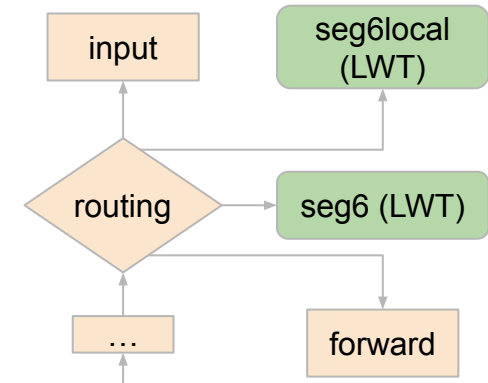
(*) T.Encaps changed in H.Encaps; (**) T.Inserts has been removed from RFC 8986



4.10

4.14

- ❖ SRv6 subsystem has been considerably improved, e.g.,:
 - Support **SR-encap of IPv4** packets and L2 ethernet frames (*) in IPv6 + SRH;
- ❖ Add support for advanced local segment processing (`seg6local` - LWT):
 - Implement several “local behaviors” (actions) such as: SRv6 End.X, End.T, End.DX4, etc;
 - A local behavior can be configured with different (**mandatory**) parameters/attributes;
 - i Packets to be processed **must have** IPv6 DA != local;
 - Some behaviors do not require SRH at all!



(*) Only support ethernet frames with IPv4/IPv6 as L3 proto.



4.10

4.14

- ❖ `iproute2` extended to support advanced local segment processing:
 - set up/destroy local behavior instances;
 - show instantiated behaviors with all the user-provided parameters/attributes.

❖ Few examples:

- Instantiate the SRv6 End behavior for the given SID:

```
$ ip -6 ro add 2001:db8::1 encap seg6local action End dev eth0
```

active SID seg6local LWT Behavior to be executed

- Instantiate the SRv6 End.T behavior for the given SID:

```
$ ip -6 ro add 2001:db8::1 encap seg6local action End.T table main dev eth0
```

attribute table, valorized with main



4.10

4.14

4.16

- ❖ IPv6 Segment Routing Header (SRH) support for Netfilter:
 - Provided as a kernel module;
 - `iptables` CLI integration to set matching rules.
- ❖ It allows matching packets based on SRH;
 - Supported **match options** include:
 - Next header, Header Extension Length, Segment Left, Last Entry, Tag.
- ❖ It can be combined with other Netfilter extensions to design complex filtering chains and actions:
 - e.g., implementing SRv6 network packet loss monitoring, delay monitoring, etc.



4.10

4.14

4.16

4.18

- ❖ SRv6 local processing enhanced with the new **End.BPF** action by:
 - A new BPF program type (`BPF_PROG_TYPE_LWT_SEG6LOCAL`);
 - BPF helpers to read/write some fields of the SRH (flags, tag and TLVs)
- ❖ End.BPF works like the SRv6 End:
 - SRH must be present;
 - Advance the next segment.
- ❖ End.BPF provides an hook for attaching an eBPF program:
 - It can **not** make arbitrary reads/writes directly into the packet;
 - Only **some fields** of the SRH (flags, tag and TLVs) can be altered through the helper functions.



4.10

4.14

4.16

4.18

- ❖ `iproute2` extended to *load&attach* an eBPF program to End.BPF;
- ❖ A file object can contains multiple eBPF programs in different sections:
 - Only one program can be attached to an End.BPF instance.
- ❖ For example:
 - *Load&attach* eBPF program “prog1” contained in “foo_obj.o” for the given SID:

```
$ ip -6 route add 2001:db8::6 encap seg6local action End.BPF endpoint \  
    object foo-obj.o section prog1 dev eth0
```

object foo-obj.o section prog1 dev eth0

File object eBPF program in
foo-obj.o section prog1



4.10

4.14

4.16

4.18

5.5

5.9

- ❖ Support for local delivery of decap packets in SRv6 End.DT6 (*);
- ❖ The Virtual Routing and Forwarding (VRF) subsystem is an enabling key for implementing new SRv6 behaviors (**);
- ❖ The VRF is extended by supporting the new **“Strict mode”**:
 - It imposes a *one-to-one* mapping between a VRF and the associated Routing Table;
 - Network-namespaces aware;
 - It can be turned on/off by acting on the “strict_mode” **sysctl** knob:
 - `net.vrf.strict_mode` (disabled by default for legacy purposes).

(*) since kernel v5.5, (**) since kernel v5.9



4.10

4.14

4.16

4.18

5.5

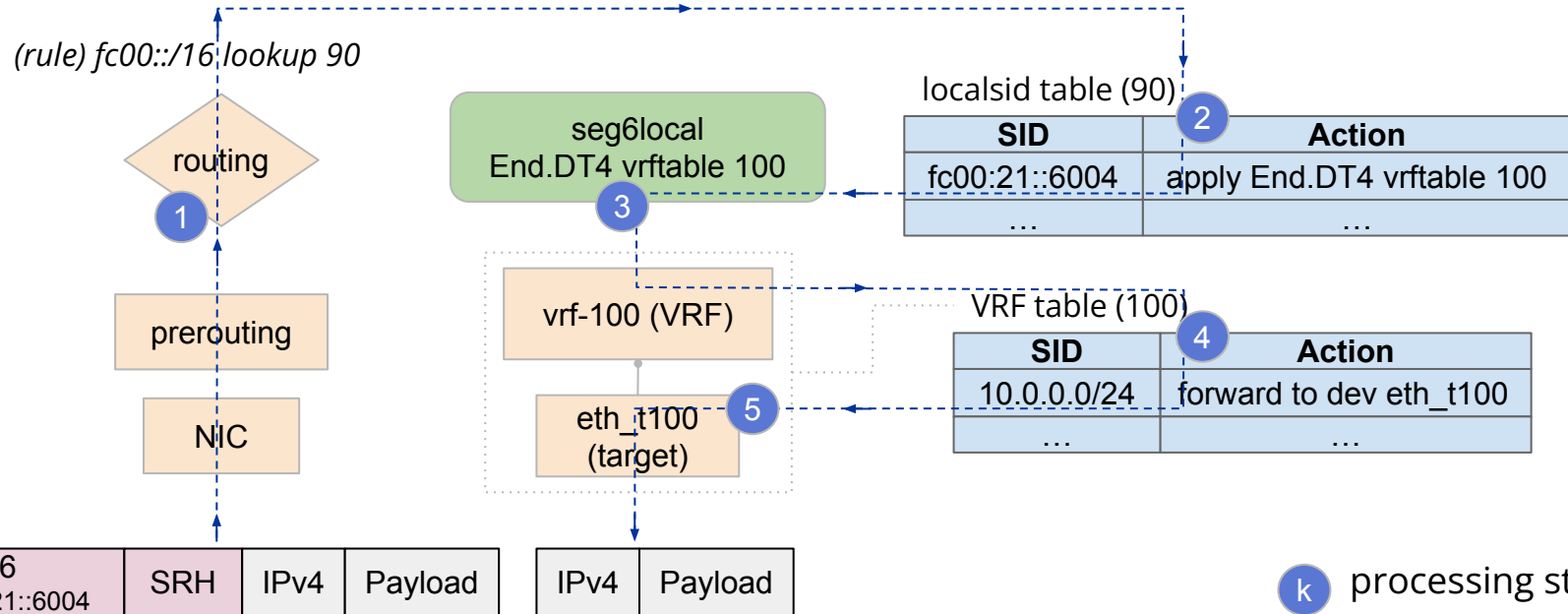
5.9

5.11

- ❖ Local processing of SRv6 (`seg6local`) subjected to heavy lifting:
 - Improved the management of behavior attributes;
 - Added support for **optional attributes** used by behaviors;
 - Added callbacks for customizing creation/destruction of behaviors.
- ❖ Add support for SRv6 **End.DT4** behavior:
 - It decaps inner IPv4 packets and performs lookup into a given Routing Table (RT):
 - Does not strictly require SRH.
 - It leverages the **VRF** to force the routing lookup into the associated RT:
 - **VRF "strict_mode" must be turned on!**
- ❖ Enhance the SRv6 End.DT6 operating mode:
 - Legacy mode (providing RT) or using a VRF as in the End.DT4 case.

4.10 4.14 4.16 4.18 5.5 5.9 5.11

- ❖ A high-level view on SRv6 End.DT4 behavior processing:





4.10

4.14

4.16

4.18

5.5

5.9

5.11

- ❖ `iproute2` extended to support both SRv6 End.DT4 and End.DT6 (VRF mode);
- ❖ `iproute2` does **not require** any **change** to support optional attributes for SRv6 local behaviors;
- ❖ For example, to instantiate an SRv6 End.DT4 behavior for a given SID:

```
$ sysctl -wq net.vrf.strict_mode=1
```

```
$ ip link add name vrf-100 type vrf table 100
```

[...] set the target device of the VRF connecting with the host [...]

```
$ ip -6 r a 2001:db8::d4 encap seg6local action End.DT4 vrftable 100 dev vrf-100
```

RT associated with
VRF vrf-100



- ❖ Add **counters** support for SRv6 local processing;
- ❖ For **each** local behavior instance they count:
 - Total number of correctly processed packets;
 - Total amount of traffic (in bytes) correctly processed;
 - Total number of packets **NOT** correctly processed.
- ❖ Counters are very interesting for:
 - Network monitoring purposes;
 - Checking whether a behavior is triggered, works as expected or not;
 - Troubleshooting purposes.
- ❖ Counters **can** be enabled on a behavior instance during the setup phase.



4.10

4.14

4.16

4.18

5.5

5.9

5.11

5.13

- ❖ By extending `iproute2`, each SRv6 behavior instance can be configured to make use of counters;
- ❖ SRv6 counters supported for any SRv6 local behavior (`seg6local`) as follows:

- Add a new SRv6 End behavior instance with the given SID and counters turned on:

```
$ ip -6 route add 2001:db8::1 encap seg6local action End count dev eth0
```

count is an optional attribute

- Per-behavior counters can be shown by adding “-s” to the `iproute2` CLI, e.g.:

```
$ ip -s -6 route show 2001:db8::1
```

```
2001:db8::1 encap seg6local action End packets 0 bytes 0 errors 0 dev eth0
```

counters (aka statistics)



4.10

4.14

4.16

4.18

5.5

5.9

5.11

5.13

5.14

- ❖ Add support for **SRv6 End.DT46** behavior:
 - With End.DT4 and End.DT6 is **not** possible to create SRv6 tunnel carrying both IPv4 and IPv6.
- ❖ End.DT46 decaps **both** IPv4/IPv6 traffic and routes traffic using a VRF:
 - It reuses the core implementation of End.DT4 and End.DT6 (VRF mode);
 - **The VRF "strict_mode" must be enabled.**
- ❖ Performance tests show no degradation in performance when DT46 is used w.r.t. End.DT4/6:
 - End.DT46 greatly **simplifies** the setup and operations of SRv6 VPNs.
- ❖ `iproute2` updated to support the new SRv6 End.DT46 behavior:
 - similar CLI and configuration required for setting End.DT4 and End.DT6 (VRF mode).



4.10

4.14

4.16

4.18

5.5

5.9

5.11

5.13

5.14

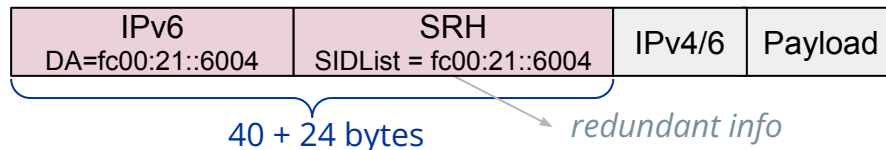
5.15

- ❖ Add optional **Netfilter hooks** to SRv6 processing;
- ❖ Netfilter hooks useful to track (*conntrack*) both outer flows and inner flows, i.e., flows carried by SRv6 packets.
- ❖ By default, Netfilter hooks for SRv6 are disabled:
 - It can impact on performance when turned on;
 - **sysctl** (system-wide) toggle for enabling LWT tunnel netfilter hooks:
 - `net.netfilter.nf_hooks_lwtunnel`
 - **NOTE:** Disabling the `nf_hooks_lwtunnel` requires kernel reboot.

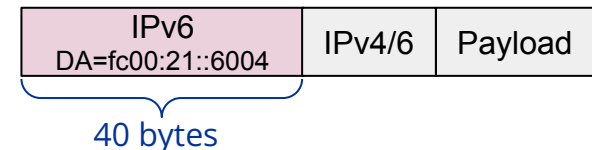


- ❖ Add support for SRv6 **Headend Reduced**:
 - H.Encaps.Red reduced version of H.Encaps.
 - H.L2Encaps.Red reduced version of H.L2Encaps.
- ❖ The H.(L2)Encaps.Red **reduces** the length of the SRH by:
 - Excluding the first segment (SID) from the SID List carried by SRH;
 - Pushing the excluded SID directly into the IPv6 DA.
- ❖ The H.(L2)Encaps.Red **can avoid** the SRH at all if the SRv6 policy contains only one SID.

H.Encaps (policy with 1 SID - *no other info in SRH*)



H.Encaps.Red (policy with 1 SID)





- ❖ `iproute2` updated to support both `H.Encaps.Red` and `H.L2Encaps.Red`;
- ❖ Two new *mode* are available to encap `seg6` in `iproute2` CLI:
 - `encap.red` for SRv6 `H.Encaps.Red` behavior;
 - `l2encap.red` for SRv6 `H.L2Encaps.Red` behavior.
- ❖ Same `iproute2` CLI syntax to perform reduced encaps, for example:
 - Perform a reduced encapsulation of an IPv4 packet into an outer IPv6 + SRH

```
$ ip -4 ro a 10.0.0.2 \  
    encap seg6 mode encap.red segs fc00::1,fc00::2 dev eth0
```

`H.Encaps.Red` SID List is transparently reduced by the Linux kernel



4.10 4.14 4.16 4.18 5.5 5.9 5.11 5.13 5.14 5.15 6.0 6.1

- ❖ Some SRv6 scenarios may require **large** number of SIDs;
- ❖ **Reducing** the **size** of a SID List is useful to:
 - Minimize the impact on MTU; enable SRv6 on legacy HW with limited processing power.
- ❖ Kernel v6.1 introduces the **NEXT-C-SID** (aka **uSID**) [1] mechanism for SRv6:
 - Efficient **representation** (compression) of the SID List;
 - Several SRv6 segments can be encoded within a single 128-bit SID.
 - NEXT-C-SID mechanism relies on the **“flavors”** framework (RFC 8986):
 - Additional operations that can modify/extend existing behaviors.
- ❖ SRv6 **End** behavior is extended with the NEXT-C-SID flavor support.

[1] - <https://datatracker.ietf.org/doc/html/draft-ietf-spring-srv6-srh-compression>



4.10

4.14

4.16

4.18

5.5

5.9

5.11

5.13

5.14

5.15

6.0

6.1

- ❖ iproute2 extended to support flavors framework;
- ❖ New “flavors” attribute to set up NEXT-C-SID **compression** on SRv6 End:
 - Nested **sub-flavors** attributes are allowed to further configure the behavior;
- ❖ NEXT-C-SID flavor for SRv6 End behavior can be configured using **optional** user-provided *sub-flavors* attributes:
 - lflen, i.e., attribute for Locator-Block length in bits (> 0 and evenly div by 8);
 - nflen, i.e., attribute for Locator-Node Function length in bits (> 0 and evenly div by 8).

- ❖ For example: *Locator-Block* *Locator-Node Function*

```
$ ip -6 ro a fc00:0:0100:0200::/48 encap seg6local action End \
  flavors next-csid lflen 32 nflen 16 dev eth0
```

flavors attribute accepts *nested attributes*

nested attributes are optional



- ❖ In some SRv6 scenarios we would like to:
 - Remove the SRv6 policy (i.e., the SID List) as we do not need it anymore;
 - Keep the IPv{4,6}-in-IPv6 encapsulation for traffic to be processed;
- ❖ Removing the SRH when all the SIDs have been processed aims to:
 - Reduce the MTU at some point in the network; enabling legacy HW with limited processing power;
- ❖ Kernel v6.3 introduces the **P**enultimate **S**egment **P**op (**PSP**) flavor:
 - The PSP reuses the “flavor” framework introduced with NEXT-C-SID patchset;
 - The PSP flavor allows an SRv6 End* behavior to **pop** the SRH on the penultimate SR Endpoint node listed in the SID List.
- ❖ SRv6 **End** behavior is extended with the PSP flavor support.



... 4.16 4.18 5.5 5.9 5.11 5.13 5.14 5.15 6.0 6.1 6.3

- ❖ No need to further extend `iproute2` to support the PSP flavor;
 - PSP flavor capability was already introduced with the NEXT-C-SID patchset;
- ❖ Reuse the “`flavors`” attribute to set up the PSP flavor on SRv6 End;
- ❖ For example:
 - ```
$ ip -6 ro a 2001:db8::1 encap seg6local action End \
 flavors psp dev eth0
```
- ❖ A PSP-enabled SRv6 End behavior instance applies:
  - PSP processing only when deployed on the SR penultimate node (Segment Left = 1).
  - “standard” End processing (ignoring the PSP flavor at all) in all other cases.





- ❖ In some SRv6 scenarios with large number of SIDs, we would like to:
  - Leverage the compressed SID List (NEXT-C-SID mechanism);
  - Forward processed packet using given L3 adjacencies (like End.X).
- ❖ NEXT-C-SID is combined with End.X, this enables arbitrary TE paths in SRv6 domains also when using compressed SID lists
- ❖ Kernel v6.6 introduces the **NEXT-C-SID** flavor for **End.X** behavior:
  - Reuse the “flavor” framework introduced with NEXT-C-SID patchset;
  - Enable the End.X behavior to use the NEXT-C-SID compression mechanism.
- ❖ SRv6 **End.X** behavior is extended with the NEXT-C-SID flavor support.



... 4.18 5.5 5.9 5.11 5.13 5.14 5.15 6.0 6.1 6.3 6.6

- ❖ No need to extend `iproute2` to support the NEXT-C-SID flavor in End.X;
  - The flavor capability was already introduced with the NEXT-C-SID patchset.
- ❖ The “`flavors`” attribute is reused to set up the NEXT-C-SID on SRv6 End.X;
  - Nested ***sub-flavors*** attributes are allowed to further configure the behavior.
- ❖ For example:
  - ```
$ ip -6 ro a 2001:db8::1 encap seg6local action End.X nh6 fd00::1 \
    flavors next-csid lflen 48 nflen 16 eth0
```
- ❖ A NEXT-C-SID-enabled SRv6 End.X behavior:
 - if the uSID list is not empty: uSID “next” processing and forward to an L3 adjacency;
 - if the uSID list is empty: “standard” End.X processing .

- ❖ We are still working to:
 - Introduce new SRv6 features into the kernel;
 - Cooperate **maintaining** the SRv6 subsystem, e.g. bug fixing, performance tests, etc.
- ❖ (Some of) Upcoming “desidered” features will:
 - Improve Observability:
 - Update the SRv6 subsystem to inform the user of the reasons why a packet has been dropped, e.g. `kfree_skb_dreason()`;
 - Add counters support to SRv6 HeadEnd behaviors (i.e.: `seg6`).
 - Enhanced Capabilities:
 - enable user-provided Traffic Class, Hop Limit in the outer IPv6 header on H.Encaps:
 - ✓ Or ... decide when Traffic Class should be inherited from inner Packet!
 - ✓ Useful to propagate ECN bits from inner packets to outer IPv6 + SRH.
 - Set the tunnel source address (outer IPv6 SA) on a per-tunnel basis (H.Encaps).


- ❖ SRv6 support enhanced considerably across the Linux kernel releases;
 - We heavily **contributed** to add new features and fix bugs.
- ❖ We extended the VRF and SRv6 subsystems by adding key features, e.g.:
 - **“Strict mode”** to 1:1 map between a VRF with its RT, used by SRv6 End.DT* behaviors;
 - **Optional attributes** to make possible complex SRv6 behavior configurations;
 - Observability with **SRv6 counters**.
- ❖ Our work was/is driven by both research and real use-cases needs:
 - Providing solution for Multi-tenant **IPv4/IPv6 VPNs**, e.g.: End.DT4/6/46;
 - Optimizing DC network utilization Traffic Engineering for AI distributed training;
 - Avoiding SRH **overhead** whenever possible, e.g.: with reduced encaps;
 - **SID compression** for supporting SRv6 legacy HW and reducing overhead, i.e., NEXT-C-SID.
- ❖ *Very active on SRv6: for suggestions/ideas, please drop us a message!*



Thank you for your attention!



andrea.mayer@uniroma2.it

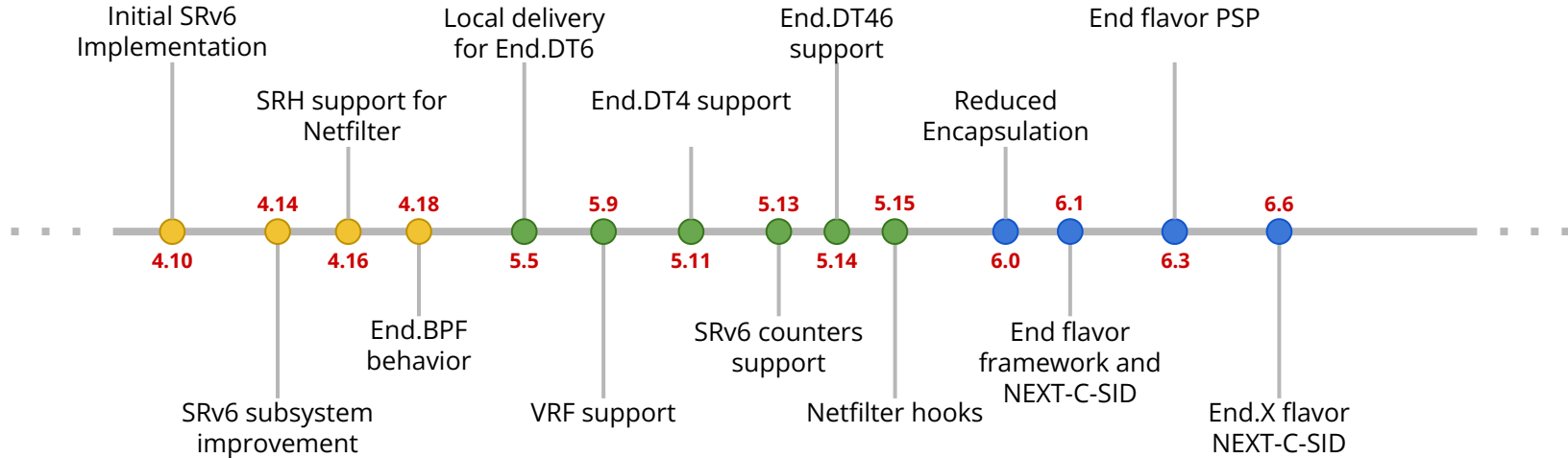


(some) Gaps and Limitations in SRv6 Linux subsystem

andrea.mayer@uniroma2.it



The evolution of SRv6 Networking Model



A lot of work has been done, but there are still problems and limits in using SRv6 in Linux.



- ❖ SRv6 End.X behavior:
 - SRv6 instantiation of an Adjacency SID [1], main purpose is for traffic-engineering policies.
 - A node has N link with its neighbours and traffic needs to be steered on a specific link.
- ❖ Currently SRv6 End.X implementation presents a limit (by design):
 - Cannot use link-local addresses to steer traffic on a specific link.
 - During the years, people complains about this limitation (mailing list/private inbox).
- ❖ A initial solution to this issue was proposed years ago [2]:
 - Add to the End.X behavior a user-provided outgoing interface (`oif`);
 - **Always** consider the outgoing interface (`oif`) while looking up the nexthop.
 - Unfortunately, this change broke perfect legitimate configurations!
- ❖ Patch [2] was not applicable at that time. So, what can we do nowadays?

[1] - <https://datatracker.ietf.org/doc/html/rfc8986#name-endx-l3-cross-connect>

[2] - <https://patchwork.kernel.org/project/netdevbpf/patch/20201015082119.68287-1-rejithomas@juniper.net/>



- ❖ With the support for **optional attributes** in the SRv6 Endpoints:
 - Differentiation between **mandatory** attributes and **optional** ones;
 - A behavior could be configured in different ways, depending on provided attributes;
- ❖ SRv6 End.X implementation limit could be overcome:
 - Considering what has been proposed in (rejected) patch [2], but with some changes, e.g.:
 - `oif` is an optional attribute BUT is considered mandatory when next-hop address (`nh6`) is a link-local one;
 - `oif`, if provided, can also be used for non link-local addresses (forcing lookup to consider also that interface);
 - When `oif` is **not** provided, End.X processing logic is left unchanged (legacy support).
- ❖ The same approach could be used for the other End DX4/DX6.

[1] - <https://datatracker.ietf.org/doc/html/rfc8986#name-endx-l3-cross-connect>

[2] - <https://patchwork.kernel.org/project/netdevbpf/patch/20201015082119.68287-1-rejithomas@juniper.net/>



- ❖ SRv6 End* processes (in some way) packets and then routes them;
- ❖ End/.X/.T and End.DX6/DT6 route packet w.r.t. the outer/inner IPv6 DA:
 - They all rely on the same routing “helper” function e.g., `seg6_nexthop_lookup()` ;
- ❖ End.DX4/End.DT4 routes (decap) packet considering the inner IPv4:
 - They all rely on the same routing “helper” function e.g., `ip_route_input()` ;
- ❖ However... depending on the Endpoint and the processed protocol, routing errors are handled differently...
 - IPv4 routing issues are immediately handled and SRv6 processing stops returning an error to the SRv6 subsystem :-)
 - IPv6 routing errors are not directly handled :-(
 - Instead, processing continues and `dst_input()` is called: `dst->input()` callback will decide the fate of the packet;
 - In case of routing error, packet are dropped but `dst->input()` returns 0: consumed!



- ❖ Uneven routing error handling (by design) in End* lead to some issues:
 - hides routing only IPv6 failures from the SRv6 network subsystem;
 - preventing aggregation of statistics on routing errors and per-behavior increasing error counters (again only for IPv6).
- ❖ We need to change the routing error handling design:
 - SRv6 End* dealing with IPv6 routing errors should return error codes to the SR subsystem;
 - Release resources as soon as the routing error is happened, e.g.: release the skb
- ❖ We would improve the SRv6 route error handling:
 - By checking the return code of `seg6_lookup_nexthop()`. When an error is returned:
 - Release the skb
 - Report the code to the SRv6 subsystem
 - Visibility of routing issues (for IPv6) in behaviors counter statistics.
- ❖ We have a patch ready to be sent for improving all of this!



- ❖ SRv6 behaviors alter packets depending on business logic:
 - encap/decap logic “exposes” the IPv{4,6} address on which routing lookup is performed;
 - Such addresses could “trigger” other cascading SRv6 behaviors!
 - Processing is carried out through special LWT types called `seg6` and `seg6local`.
- ❖ A (wrong) SRv6 config leading to kernel stack overflow was recently reported in the mailing list [1]:
 - It breaks the SRv6 programming model, where a SID provides both topological and service information;
- ❖ Such config is flawed, but presents a pathological issue worth noting:
 - It essentially creates a loop between two (different) LWTs input processing functions calling each other!
 - You can find my detailed explanation in [2].

[1] - <https://lore.kernel.org/netdev/2bc9e2079e864a9290561894d2a602d6@akamai.com/T/>

[2] - <https://lore.kernel.org/netdev/20241120181201.594aab6da28ec54d263c9177@uniroma2.it/>



❖ In the same host:

1. A plain IPv4 packet with DA=**X** is encapsulated into an SRv6 packet with IPv6 DA=**Y**, e.g.:
 - a. `$ ip ro add X encap seg6 mode encap segs Y dev eth0 vrf vrf9`
2. The produced SRv6 packet (DA=**Y**) is then decapsulated (within the same VRF), retrieving back the carried out plain IPv4 packet with DA=**X**, e.g.:
 - a. `$ ip ro add Y encap seg6local action End.DT4 vrf table 1009 dev vrf9`
(table 1009 is bound to vrf9)

❖ And here we have a loop:

- (2) will call (1), which in turn will call (2), and so on.

❖ We have no control over a misconfigured system, but how can we work to mitigate this situation?



- ❖ We've come up with some ideas on how to deal with the “loop” issue:
 - All fundamentally based on tracking (e.g., count) how many times a packet is handled by a LWT tunnel processing function (no matter of LWT type);
 - LWT processing counter operates on a per-packet basis;
 - When the counter exceeds a given value, the packet is dropped avoiding the stack overflow.
- ❖ How can we (try to) overcome this issue?
 - Different approaches to relate a LWT processing counter with a packet:
 - Store the counter in `skb->cb[]`;
 - Store a counter in the `skb` directly;
 - Create a new type of `skb` extensions to store the counter.
 - We have implemented a PoC using a new `skb` extension for counting how many times a packet has been processed by LWT input/output functions.
- ❖ All have pros and cons, discuss about them together!



- ❖ What is SRv6 H.L2Encaps?
 - Encapsulates Ethernet frames within SRv6 packets;
 - Facilitates Layer 2 Ethernet VPN (EVPN);
 - Used in combination with SRv6 Endpoint Decapsulation of L2 and X-connect (DX2);
- ❖ Implemented in Linux by leveraging Lightweight Tunnel (LWT - `seg6`):
 - LWTs are bound to Layer 3 protocols as IPv4 or IPv6!
- ❖ But... we need to encapsulate a generic Ethernet frame !?
 - H.L2Encaps **can not** encapsulate ethernet frames with **generic ethernet type protocols**;
 - ... and even in the case of IPv4 and IPv6 layers, need some tricks to make it “work”, e.g.:
 - “playing” with static ARP/NDP;
 - using bridge and dummy interfaces to emulate a quasi-pseudo-wire interface on encap node
 - spoofing the destination mac address on encap node or rewrite the mac address in the encap and decap nodes accordingly.



- ❖ How can we overcome H.L2Encaps limitation (build upon LWT)?
- ❖ Probability creating a pseudo-wire virtual ethernet device...
 - A new network device type “seg6” which is capable of encapsulating ethernet frames:
 - Add outer IPv6 + SRH (or performs a reduced encapsulation);
 - SID List to be pushed defined during the device setup;
- ❖ And what about the decap?
 - We can “re-use” the End.DX2 for decapsulating and forwarding packets;
 - It's worth noting that SID are usually not bound to any local address/interface:
 - We cannot leverage the input datapath;
 - We could intercept IPv6 + SRH traffic using netfilter callbacks and then process the inner L2 frame accordingly.
- ❖ But...what about creating a new TC action for implementing the SRv6 H.L2Encaps :-) ?