

```
# ip -6 route add  
encap seg6 mde inline  
segs 2001:db8:1::1,  
2001:db8:2::1,  
2001:db8:3::1  
dev eth0
```

GNU/Linux a Segment Routing over IPv6 (SRv6)

LinuxDays 2026 • Radek Zajíc • radek@zajic.v.pytli.cz



Vy už jste někdy tuneloval? Já ne.

- neznámý politik

O čem to dnes bude

- **Proč tunelování** - segmentace provozu, VPN, traffic engineering
- **IPSec, WireGuard, EVPN/VXLAN** - známé technologie
- **MPLS/VPLS** - label switching
- **SRv6** - proč, koncept, implementace v Linuxu
- Živé demo na pětiuzlové topologii, packet captures
- Podpora SRv6 v démonech pro dynamické routování

Úvod a motivace

Proč vůbec tunelování?

- **Segmentace provozu, virtualizace sítě** - oddělit zákazníky/tenanty na sdílené infrastruktuře
- **VPN** - propojit vzdálené sítě přes nedůvěryhodný/veřejný underlay
- **Traffic engineering** - vynutit konkrétní cestu, ne jen tu, co zvolí IGP

Začneme cestou "zašifruj/zabal paket a pošli" (IPSec/WireGuard), přejdeme k "zabal L2 segment" (EVPN/VXLAN), abychom došli k závěru, kdy "paket nese instrukce pro síť" (SRv6).

IPSec

- **L3**
- **šifrování (ESP)** - tranzitní síť neví, co nese
- **IKE** - automatické sjednání klíčů
- **tunnel mode** (celý paket) vs. **transport mode** (jen payload)
- use-case: site-to-site VPN, šifrovaná propojení

```
> Internet Protocol Version 4, Src: 10.200.23.2, Dst: 10.200.12.1
  ▾ Encapsulating Security Payload
    ESP SPI: 0xc91e4dc5 (3374206405)
    ESP Sequence: 7
```

příklad s transport mode: `ESP(spi=...)` , uživatelská data (kromě IP) nejsou vidět

WireGuard

- L3
- **moderní, jednoduchý**, ale bez rozumné orchestrace
- **UDP transport**, Noise protokol pro key exchange
- žádný složitý control plane (na rozdíl od IPSec/IKE/MPLS/EVPN/SRv6)
- minimalistická hlavička - nejmenší overhead ze šifrovaných VPN

```
> Internet Protocol Version 4, Src: 10.200.12.1, Dst: 10.200.23.2
> User Datagram Protocol, Src Port: 51820, Dst Port: 51820
  < WireGuard Protocol
    Type: Transport Data (4)
    Reserved: 000000
    Receiver: 0x14ae3aab
    Counter: 5
    Encrypted Packet
```

Wireguard provoz: zašifrovaná data zabalená v UDP paketu

EVPN/VXLAN

- **L2 over L3/L4 (UDP)** (overlay)
- **MP-BGP** jako control plane - distribuuje MAC/IP informace
- **VNI** = segment (obdoba VLAN, ale škáluje do milionů)
- state v jádru sítě: VTEP/MAC tabulky
- use-case: multi-tenant L2 nebo L3 overlay v datacentru
- pro detaily vizte přednášku EVPN a VXLAN v Linuxu od Ondřeje Zajíčka

EVPN/VXLAN

```
> Internet Protocol Version 4, Src: 10.200.23.2, Dst: 10.200.12.1
> User Datagram Protocol, Src Port: 44631, Dst Port: 4789
> Virtual eXtensible Local Area Network
> Ethernet II, Src: e2:37:46:46:ec:f5 (e2:37:46:46:ec:f5), Dst: 32:f0:91:80:f2:97 (32:f0:91:80:f2:97)
> Internet Protocol Version 4, Src: 192.168.100.2, Dst: 192.168.100.1
✓ Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0x3a31 [correct]
  [Checksum Status: Good]
  Identifier (BE): 24176 (0x5e70)
  Identifier (LE): 28766 (0x705e)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  \[Request frame: 4\]
  [Response time: 0.525 ms]
  Timestamp from icmp data: Oct  1, 2026 10:48:49.262517000 CEST
  [Timestamp from icmp data (relative): 0.001069000 seconds]
> Data (40 bytes)
```

VXLAN(vni 100) nesoucí ICMP/ICMPv6/ARP

Srovnání

	vrstva	control plane	state v jádru sítě	šifrování	use-case
IPSec	L3	IKE (manuální/auto)	SA na endpointech	ano (ESP)	tradiční site-to-site VPN
WireGuard	L3 over L3/L4	žádný (statická konfigurace)	peer tabulka	ano (Noise)	jednoduchá moderní VPN
EVPN/VXLAN	L2 over L3/L4	MP-BGP	VTEP/MAC tabulky	ne	multi-tenant DC overlay

Co mají společné: **enkapsulace** řeší přenos paketů z místa A do B, tranzitní síť obsah neřeší.
Co uvidíme dál: SRv6 jde o krok dál - **paket může síti říct, co má dělat.**

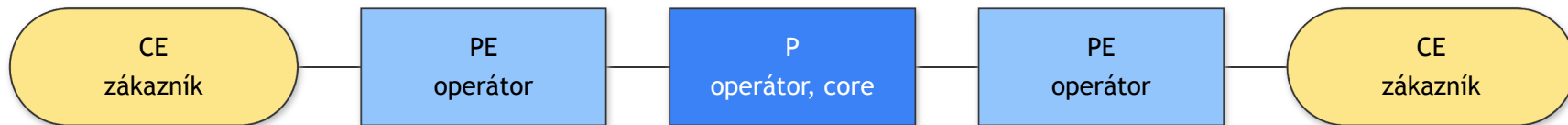
Segment Routing - koncept

Traffic engineering v kostce

- Klasický routing: síťový uzel se rozhoduje o routování sám, podle svojí routovací tabulky
- **Source routing**: odesílatel (`headend`) určí celou cestu dopředu
- **Segment Identifier (SID)**: předem známý hop na cestě paketu
- **Segment list (SL)**: seznam SID, který si paket nese s sebou
- **Segment Routing Header (SRH)**: IPv6 extension header se segment listem

Síť nutně nerozhoduje "kam s paketem dál" - paket routovací informaci **může nést v sobě**.

MPLS - tradiční předchůdce SRv6



- **CE** (Customer Edge) - zákaznický router, o MPLS nic neví
- **PE** (Provider Edge) - operátorský hraniční router, přidává/odebírá MPLS značky (labels)
- **P** (Provider core) - tranzitní router, neřeší obsah paketů, routuje podle MPLS značek

MPLS L3VPN vs. VPLS

MPLS L3VPN (routovaná varianta)

- L3 služba (přenáší IP pakety, neřeší Ethernet headery)
- PE router dělá lookup v routovací tabulce VRF a následně paket odroutuje
- control plane: LDP / RSVP-TE / BGP

VPLS (L2VPN pseudowire)

- L2 služba
- MPLS label identifikuje "pseudowire" (okruh)
- point-to-point nebo multipoint L2
- žádný IP lookup, jen přeposlá Ethernet rámce

MPLS label říká "kam s paketem dál".

SRv6 přebírá tuto myšlenku, trasu paketu **umí zakódovat přímo do IPv6 adresy nebo paketu.**

MPLS - capture

- ✓ Ethernet II, Src: 52:54:00:3a:00:02 (52:54:00:3a:00:02), Dst: 52:54:00:3a:00:11 (52:54:00:3a:00:11)
 - > Destination: 52:54:00:3a:00:11 (52:54:00:3a:00:11)
 - > Source: 52:54:00:3a:00:02 (52:54:00:3a:00:02)
 - Type: MPLS label switched packet (0x8847)
 - [Stream index: 0]
- ✓ MultiProtocol Label Switching Header, Label: 19, Exp: 0, S: 1, TTL: 64
 - 0000 0000 0000 0001 0011 = MPLS Label: 19 (0x00013)
 - 000. = MPLS Experimental Bits: 0
 - 1 = MPLS Bottom Of Label Stack: 1
 - 0100 0000 = MPLS TTL: 64
- > Internet Protocol Version 6, Src: fd00:12::1, Dst: fd00:ff::3
- > Internet Control Message Protocol v6

MPLS ethertype a label

Proč IPv6?

- MPLS potřebuje **extra label stack** nad IP
- VXLAN potřebuje **extra UDP/VXLAN hlavičku** nad IP
- SRv6 obsahuje segment list v IPv6 paketu nebo pomocí cílové IPv6 adresy rovnou řekne, kam paket doručit
- Žádná nová enkapsulační vrstva navíc - jen IPv6 header (+ volitelně SRH)
- Obrovský adresní prostor IPv6 dovoluje použít IPv6 adresu pro identifikaci cíle paketu (či funkce, která v cíli má paket zpracovat)

SRv6 dává IPv6 adrese význam.

SRv6 - mechanismy a implementace v Linuxu

RFC 8986: SID = instrukce, ne jen adresa

SRv6 Network Programming: SID je IPv6 adresa, na kterou je u cílového uzlu navázána konkrétní funkce. Paket nese **posloupnost instrukcí**.

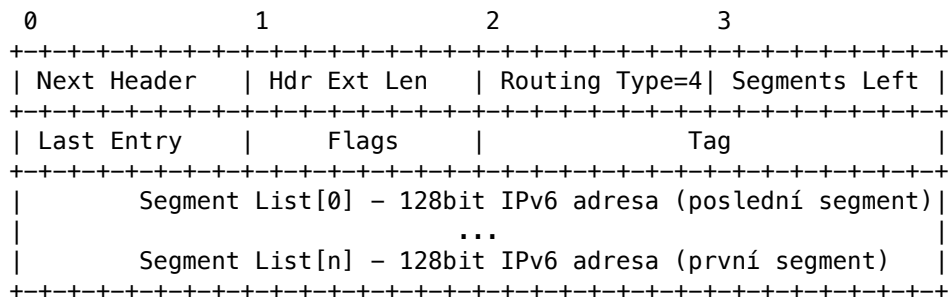
```
| 2001:db8:dead:beef: | :3 | :46 |  
| LOC | FUNCT | ARG |  
| kam paket doručit | co udělat | parametry |
```

- **LOC** - lokátor, dovede paket **běžným IPv6 routingem** na správný cílový uzel
- **FUNCT** - funkce, identifikuje lokální akci (**behavior**) na cílovém uzlu
- **ARG** - parametry funkce (volitelné)
- Vazba konkrétního SID na SRv6 behavior je lokální rozhodnutí cílového uzlu. Hodnota **FUNCT** není univerzálním kódem behavioru.
- Z hodnoty SRv6 SID samotné obecně nelze poznat, jaký behavior reprezentuje.
- Behaviors samotné jsou standardizovaná chování (povíme si dále)

Příklad z labu: `fd00:dead:beef:23::146` = "udělej End.DT46 , VRF 100"

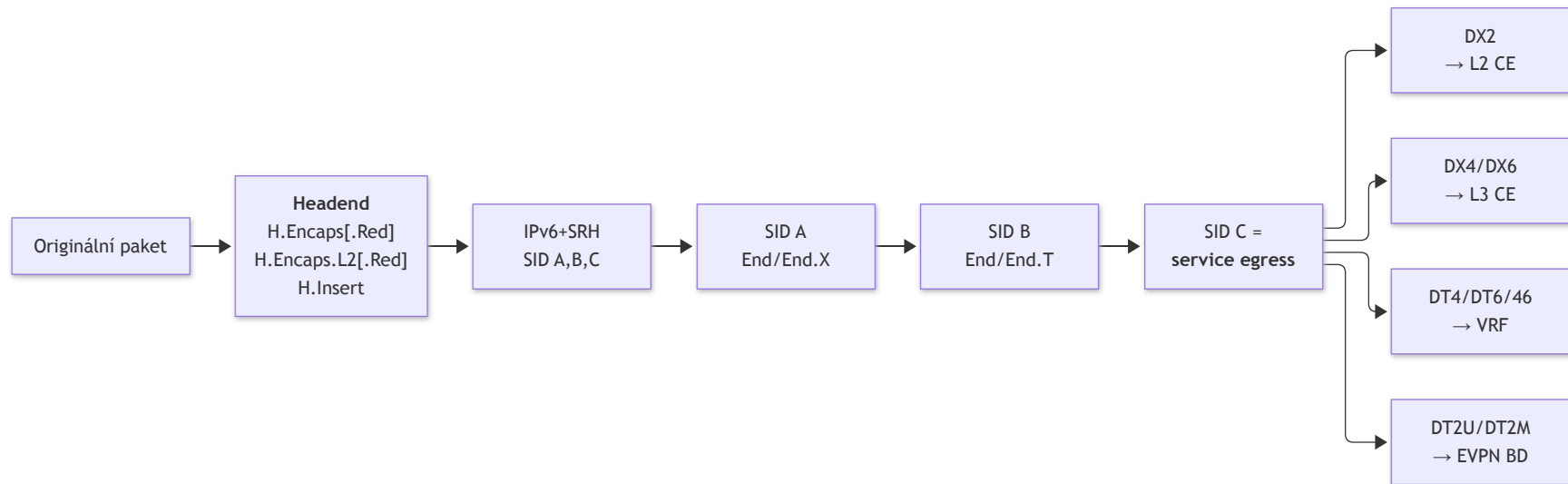
SRH - Segment Routing Header (voliteľná hlavička)

SID v cílové adrese už je instrukce, kam a co s paketem. IPv6 paket ale může v SRH nést i další informace o routingu. **SRH nemusí být vůbec přítomen.**



- **Segments Left:** kolik segmentů ještě zbývá navštívit (dekrementuje se s každým navštíveným routerem)
- **Segment List je v obráceném pořadí:** na pozici [0] je **poslední** segment (finální cíl), [n] je první segment (aktuální cílová IPv6 adresa)
- **Overhead: 8 B fixní hlavička + 16 B na každý segment**
- **Kdy SRH úplně chybí:** jediný segment + Reduced enkódování - samotná cílová IPv6 adresa (DA) je ten jediný segment, doplňková SR hlavička by nic navíc nepřidala

Celé RFC jedním obrázkem



Headend vytvoří → tranzit uzly předávají → egress uzel dekapsuluje a doručí

End.X = pokračuj přes konkrétní sousední link (vynucený next-hop interface).

End.T = pokračuj po lookupu v jiné routovací tabulce (pořád non-terminální, nedekapsuluje)

H.* (headend) akce - balíme a upravujeme pakety

```
ip -[4|6] route add DEST encap seg6 mode \  
  { encap | encap.red | l2encap | l2encap.red | inline } \  
  segs SID1,SID2,... dev IFACE
```

RFC 8986 akce	Linux mode	overhead	význam
H.Encaps	encap	40 B + SRH	Zabal paket do nové IPv6 hlavičky s plnou SRH
H.Encaps.Red	encap.red	40 B	Zabal paket bez použití SRH při 1 segmentu
H.Encaps.L2	l2encap	40 B + SRH	Zabal celý Ethernet rámec (L2 pseudowire)
H.Encaps.L2.Red	l2encap.red	40 B	Zabal Ethernet rámec bez SRH při 1 segmentu
H.Insert	inline	jen SRH	Vlož SRH přímo do existujícího paketu (bez tunelu)

H.* akce v detailu

- **Encaps vs. Insert:**

- `Encaps` = nový vnější IPv6 paket (+SRH hlavička) - tunelovací režim, obaluje se celý původní paket - funguje pro jakýkoli payload (IPv4, IPv6, Ethernet).
- `Insert` = SRH se vloží přímo do existujícího IPv6 paketu - proto funguje jen pro nativní IPv6 provoz; má menší overhead (jen přidání SRH), ale úprava je méně "transparentní" (modifikuje originální paket)

- `L2` **varianta:**

- `H.Encaps.L2` obaluje celý Ethernet rámec (protějšek `End.DX2`)
- `H.Encaps` bez `L2` obaluje přímo IP paket
- `Red` (**Reduced**) nemění cestu ani segmenty - jen optimalizace zápisu: první segment nemusí být zároveň zanesen v SRH (je vidět už jako cílová adresa `outer DA`). Při jediném segmentu se SRH nemusí přidat vůbec.

Behaviors: akce s paketem

Behavior	Význam/příklad
End	routuj na další SID
End.X	pošli paket přes konkrétní L3 sousední link
End.T	routuj podle lookupu v konkrétní IPv6 routovací tabulce
End.DX4 / End.DX6	vybal + L3 cross-connect (pošli paket do rozhraní)
End.DT4 / DT6 / DT46	vybal + lookup ve (VRF) tabulce
End.DX2	vybal + L2 cross-connect (pošli rámec do rozhraní)
End.DT2U / End.DT2M	vybal + MAC lookup / flooding (EVPN bridging)

End vs. DX vs. DT

End → zpracování SRv6: dekrementuje Segments Left v SRH, cílová adresa (DA) = další SID; odroutuj paket
End.DX* → vybal paket/rámec + pošli paket do definovaného rozhraní → konkrétní next-hop/interface
End.DT* → vybal paket + vyhledej v routovací tabulce a odroutuj → konkrétní VRF/FIB

D = Decapsulate (vybal paket)

Hypotetická náhrada VXLAN pomocí SRv6

EVPN/VXLAN služba	SRv6 headend	Egress behavior
L2 unicast bridging	H. Encaps . L2	End . DT2U
L2 BUM	H. Encaps . L2	End . DT2M
L2 point-to-point / VPWS	H. Encaps . L2	End . DX2
L3VNI / VRF, dual-stack	H. Encaps	End . DT46
L3VNI / VRF, oddělené AF	H. Encaps	End . DT4 + End . DT6

Behaviors a headends v Linuxu: `seg6` vs `seg6local`

keyword	umístění	funkce
<code>seg6</code> (encap)	ingress (headend)	zabalí paket/přidává SRH
<code>seg6local</code> non-terminální	tranzit (midpoint)	sníží <code>Segments Left</code> , forwarduje dál
<code>seg6local</code> terminální (DX*/DT*)	egress (endpoint)	vybalí paket/odstraní SRH, doručí paket

- `seg6` se váže na **cílovou adresu originálního paketu** (ne na SID)
- `seg6local` se váže na **konkrétní SID adresu** - "když přijde paket na adresu X, proved' akci Y"
- Tranzitní router bez `seg6local` behaviors jen normálně forwarduje - SRv6 pakety řeší pouze pokud zpracovává SRv6 akce (např. `End/End.X/End.T`), jinak jsou to pro něj jen IPv6 pakety

Podpora SRv6 v kernelu

verze	kdy	přibývší komponenty
4.10	2017	seg6 (encap), seg6local : End , End.X , End.DX6
4.14	2017	+ End.T , End.DX2 , End.DX4 , End.DT6 , IPv4 payload, H.Encaps.L2
5.14	2021	+ End.DT4 , End.DT46
6.3	2023	flavor PSP
6.6	2023	NEXT-C-SID (μSID) pro End.X

Zajímavost: End.DX4 je **novější** než End.DX6 (4.14 vs. 4.10) - IPv6 cross-connect byl v kernelu od začátku, IPv4 přibylo o rok později.

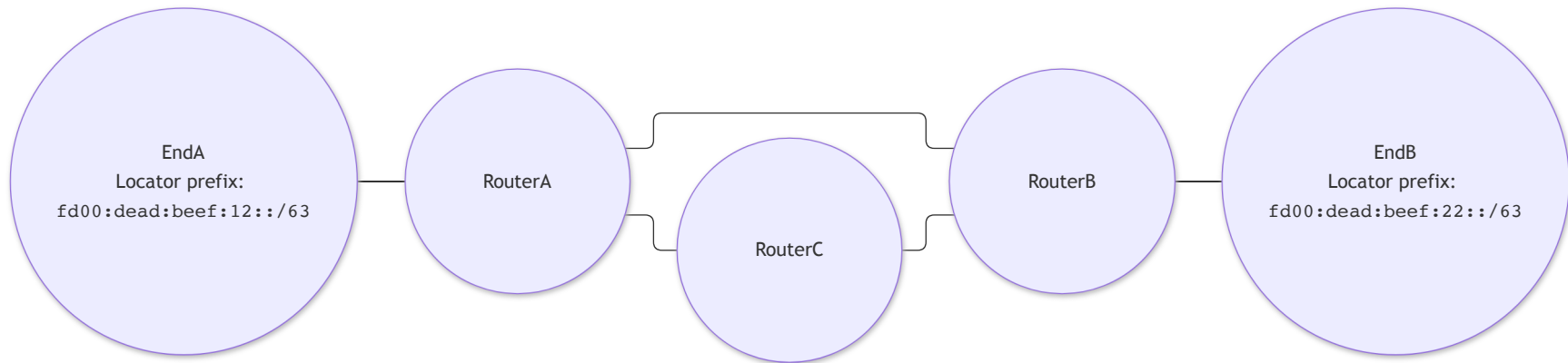
Drobné, ale zásadní nástrahy

- `net.ipv6.conf.<iface>.seg6_enabled=1` - povoluje SRv6 na rozhraní
- `net.ipv4.conf.<iface>.rp_filter=0` - je potřeba konfigurovat i v použitých network namespacech
- `net.vrf.strict_mode=1` - kernel vynucuje izolaci mezi VRF na úrovni socketů

Viz také Segment Routing IPv6 (SRv6) with FRR and Ubuntu.

Laboratorní demo

Topologie



- RouterA/B/C tvoří **trojúhelník** - umožňuje TE demo
- Normální IP routing EndA → EndB je skrze **RouterA → RouterB přímo** (krátká cesta)
- SRv6 segment list dokáže **vynutit routování přes RouterC**

End.DX2 - L2 pseudowire (cross-connect)

Potřebujeme: `ns-dx1` network namespace (CE strana) na EndA i EndB, MAC nastavená na MAC protistrany. Pseudowire je obousměrný - obě strany musí zabalovat svůj směr a vybalovat ten druhý.

```
# EndA: headend (-> EndB)
# zabal celý Ethernet rámec
ip route add 10.128.2.2/32 \
  encap seg6 mode l2encap.red \
  segs fd00:dead:beef:22::2 \
  dev l10
```

```
# EndA: egress (<- EndB)
# dekapsuluj, pošli na rozhraní
ip -6 route add \
  fd00:dead:beef:12::2/128 \
  encap seg6local action \
  End.DX2 oif dx1 count dev dx1
```

```
# EndB: headend (-> EndA)
# zabal celý Ethernet rámec
ip route add 10.128.2.1/32 \
  encap seg6 mode l2encap.red \
  segs fd00:dead:beef:12::2 \
  dev l20
```

```
# EndB: egress (<- EndA)
# dekapsuluj, pošli na rozhraní
ip -6 route add \
  fd00:dead:beef:22::2/128 \
  encap seg6local action \
  End.DX2 oif dx1 count dev dx1
```

```
# test - spouštíme na EndA, uvnitř ns-dx1
ip netns exec ns-dx1 ping -I 10.128.2.1 10.128.2.2
```

Point-to-point End.DX2 BUM neřeší - oif posílá vše na jeden SID. Multipoint bridge by potřeboval End.DT2M (flood) + End.DT2U (MAC unicast).

End.DX2 - capture

```
> Ethernet II, Src: 02:00:00:00:10:02 (02:00:00:00:10:02), Dst: 02:00:00:00:10:01 (02:00:00:00:10:01)
✓ Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:22::2
    0110 .... = Version: 6
    > .... 0000 0000 .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 0000 0000 0000 0000 0000 = Flow Label: 0x000000
    Payload Length: 98
    Next Header: Ethernet (143)
    Hop Limit: 63
    > Source Address: fd00:dead:beef:10::2
    > Destination Address: fd00:dead:beef:22::2
    [Stream index: 0]
> Ethernet II, Src: 02:00:00:00:d1:01 (02:00:00:00:d1:01), Dst: 02:00:00:00:d1:02 (02:00:00:00:d1:02)
> Internet Protocol Version 4, Src: 10.128.2.1, Dst: 10.128.2.2
> Internet Control Message Protocol
```

Ethernet rámeček zabalený v IPv6 paketu, bez SRH. SRv6 DA = fd00:dead:beef:22::2 .

V cíli bude forwardován do konkrétního rozhraní (akce DX).

End.DX4 - L3 cross-connect (IPv4)

Potřebujeme: `ns-dx0` network namespace (CE strana) s vlastní default routou, PE strana (`dx0`) s vlastní adresou ve stejné podsíti - na EndA i EndB. Opět obousměrné: každá strana zabaluje svůj směr a vybaluje ten druhý.

```
# EndA: headend (-> EndB)
ip route add 10.128.17.2/32 \
  encap seg6 mode encap \
  segs fd00:dead:beef:22::4 \
  dev l10
```

```
# EndA: egress (<- EndB)
ip -6 route add \
  fd00:dead:beef:12::4/128 \
  encap seg6local action \
  End.DX4 nh4 10.128.1.2 \
  count dev dx0
```

```
# EndB: headend (-> EndA)
ip route add 10.128.1.2/32 \
  encap seg6 mode encap \
  segs fd00:dead:beef:12::4 \
  dev l20
```

```
# EndB: egress (<- EndA)
ip -6 route add \
  fd00:dead:beef:22::4/128 \
  encap seg6local action \
  End.DX4 nh4 10.128.17.2 \
  count dev dx0
```

```
# test - spouštíme na EndA, uvnitř ns-dx0
ip netns exec ns-dx0 ping -I 10.128.1.2 10.128.17.2
```

End.DX4 - capture

```
> Ethernet II, Src: 02:00:00:00:10:02 (02:00:00:00:10:02), Dst: 02:00:00:00:10:01 (02:00:00:00:10:01)
✓ Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:22::4
    0110 .... = Version: 6
    > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 108
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 63
    > Source Address: fd00:dead:beef:10::2
    > Destination Address: fd00:dead:beef:22::4
    [Stream index: 0]
    ✓ Routing Header for IPv6 (Segment Routing)
        Next Header: IPIP (4)
        Length: 2
        [Length: 24 bytes]
        Type: Segment Routing (4)
        Segments Left: 0
        Last Entry: 0
        Flags: 0x00
        Tag: 0000
        Address[0]: fd00:dead:beef:22::4
    > Internet Protocol Version 4, Src: 10.128.1.2, Dst: 10.128.17.2
    > Internet Control Message Protocol
```

IPv4 paket zabalený v IPv6 paketu, včetně SRH. SRv6 DA = `fd00:dead:beef:22::4` (zároveň jako `address[0]` v SRH)
V cíli bude forwardován do konkrétního rozhraní (akce DX).

End.DX6 - L3 cross-connect (IPv6)

Stejná konstrukce jako DX4 , jen IPv6 - opět obousměrně:

```
# EndA: headend (-> EndB)
ip -6 route add \
    2001:db8:beef:f021::2/128 \
    encap seg6 mode encap \
    segs fd00:dead:beef:22::6 \
    dev l10
```

```
# EndA: egress (<- EndB)
ip -6 route add \
    fd00:dead:beef:12::6/128 \
    encap seg6local action End.DX6 \
    nh6 2001:db8:beef:f011::2 \
    count dev dx0
```

```
# EndB: headend (-> EndA)
ip -6 route add \
    2001:db8:beef:f011::2/128 \
    encap seg6 mode encap \
    segs fd00:dead:beef:12::6 \
    dev l20
```

```
# EndB: egress (<- EndA)
ip -6 route add \
    fd00:dead:beef:22::6/128 \
    encap seg6local action End.DX6 \
    nh6 2001:db8:beef:f021::2 \
    count dev dx0
```

```
# test - spouštíme na EndA, uvnitř ns-dx0
ip netns exec ns-dx0 ping -6 -I 2001:db8:beef:f011::2 2001:db8:beef:f021::2
```

End.DX6 - capture

```
> Ethernet II, Src: 02:00:00:00:10:02 (02:00:00:00:10:02), Dst: 02:00:00:00:10:01 (02:00:00:00:10:01)
✓ Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:22::6
    0110 .... = Version: 6
    > .... 0000 0000 .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 1100 1000 1010 1011 0111 = Flow Label: 0xc8ab7
    Payload Length: 128
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 63
    > Source Address: fd00:dead:beef:10::2
    > Destination Address: fd00:dead:beef:22::6
    [Stream index: 0]
    ✓ Routing Header for IPv6 (Segment Routing)
        Next Header: IPv6 (41)
        Length: 2
        [Length: 24 bytes]
        Type: Segment Routing (4)
        Segments Left: 0
        Last Entry: 0
        Flags: 0x00
        Tag: 0000
        Address[0]: fd00:dead:beef:22::6
    > Internet Protocol Version 6, Src: 2001:db8:beef:f011::2, Dst: 2001:db8:beef:f021::2
    > Internet Control Message Protocol v6
```

IPv6 paket zabalený v IPv6 paketu, včetně SRH. SRv6 DA = `fd00:dead:beef:22::6` (zároveň jako `address[0]` v SRH).
V cíli bude forwardován do konkrétního rozhraní (akce DX).

End.DT4 - L3VPN (VRF, IPv4)

Potřebujeme: VRF `vrf-dt0` (table 100) + dummy rozhraní `dt0` - na EndA i EndB. Obousměrně: oba uzly zabalují svůj směr a vybalují ten druhý.

```
# EndA: headend (-> EndB)
ip route add 10.128.16.1/32 \
  encap seg6 mode encap \
  segs fd00:dead:beef:23::4 \
  dev l10
```

```
# EndA: egress (<- EndB)
ip -6 route add \
  fd00:dead:beef:13::4/128 \
  encap seg6local action \
  End.DT4 vrftable 100 \
  count dev vrf-dt0
```

```
# EndB: headend (-> EndA)
ip route add 10.128.0.1/32 \
  encap seg6 mode encap \
  segs fd00:dead:beef:13::4 \
  dev l20
```

```
# EndB: egress (<- EndA)
ip -6 route add \
  fd00:dead:beef:23::4/128 \
  encap seg6local action \
  End.DT4 vrftable 100 \
  count dev vrf-dt0
```

```
# test - spouštíme na EndA, uvnitř vrf-dt0
ip vrf exec vrf-dt0 ping -I 10.128.0.1 10.128.16.1
```

End.DT4 - capture

```
> Ethernet II, Src: 02:00:00:00:10:02 (02:00:00:00:10:02), Dst: 02:00:00:00:10:01 (02:00:00:00:10:01)
> Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:23::4
    0110 .... = Version: 6
    > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 0000 0000 0000 0000 = Flow Label: 0x000000
    Payload Length: 108
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 64
    > Source Address: fd00:dead:beef:10::2
    > Destination Address: fd00:dead:beef:23::4
    [Stream index: 0]
    > Routing Header for IPv6 (Segment Routing)
        Next Header: IPIP (4)
        Length: 2
        [Length: 24 bytes]
        Type: Segment Routing (4)
        Segments Left: 0
        Last Entry: 0
        Flags: 0x00
        Tag: 0000
        Address[0]: fd00:dead:beef:23::4
    > Internet Protocol Version 4, Src: 10.128.0.1, Dst: 10.128.16.1
    > Internet Control Message Protocol
```

IPv4 paket zabalený v IPv6 paketu, včetně SRH. SRv6 DA = `fd00:dead:beef:23::4` (zároveň jako `address[0]` v SRH).
V cíli bude odroutován do VRF.

End.DT6 - L3VPN (VRF, IPv6)

- Linux implementuje End.DT4 / End.DT6 / End.DT46 sdíleným kódem (`end_dt_vrf_core()`) - End.DT6 je prostě ten samý mechanismus jako DT4 , jen pro IPv6 payload
- V labu samostatně nedemonstrujeme - End.DT46 (další slide) ho funkčně obsahuje jako svoji IPv6 část
- **Zajímavá asymetrie:** End.DT6 umí `table` (legacy) i `vrftable` , End.DT4 umí jen `vrftable` - RFC 8986 tohle nerozlišuje, je to čistě Linuxí implementační mezera

End.DT46 - L3VPN, skutečný dualstack

Jeden SID, oba protokoly, stejná VRF - nejčistší demonstrace "network programming" myšlenky. Potřebujeme VRF `vrf-dt0` (stejnou jako `DT4`) na EndA i EndB. Obousměrně, oba protokoly přes stejnou SID v obou směrech.

```
# EndA: headend (-> EndB)
ip route add 10.128.19.1/32 \
  encap seg6 mode encap \
  segs fd00:dead:beef:23::7 \
  dev l10
```

```
ip -6 route add \
  2001:db8:beef:f023::1/128 \
  encap seg6 mode encap \
  segs fd00:dead:beef:23::7 \
  dev l10
```

```
# EndA: egress (<- EndB)
ip -6 route add \
  fd00:dead:beef:13::7/128 \
  encap seg6local action \
  End.DT46 vrftable 100 \
  count dev vrf-dt0
```

```
# test - spouštíme na EndA, uvnitř vrf-dt0, oba protokoly přes stejnou SID
ip vrf exec vrf-dt0 ping -4 -I 10.128.3.1 10.128.19.1
ip vrf exec vrf-dt0 ping -6 -I 2001:db8:beef:f013::1 2001:db8:beef:f023::1
```

```
# EndB: headend (-> EndA)
ip route add 10.128.3.1/32 \
  encap seg6 mode encap \
  segs fd00:dead:beef:13::7 \
  dev l20
```

```
ip -6 route add \
  2001:db8:beef:f013::1/128 \
  encap seg6 mode encap \
  segs fd00:dead:beef:13::7 \
  dev l20
```

```
# EndB: egress (<- EndA)
ip -6 route add \
  fd00:dead:beef:23::7/128 \
  encap seg6local action \
  End.DT46 vrftable 100 \
  count dev vrf-dt0
```

End.DT46 - capture - IPv4

```
> Ethernet II, Src: 02:00:00:00:10:02 (02:00:00:00:10:02), Dst: 02:00:00:00:10:01 (02:00:00:00:10:01)
> Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:23::7
    0110 .... = Version: 6
    > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 0000 0000 0000 0000 = Flow Label: 0x000000
    Payload Length: 108
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 64
    > Source Address: fd00:dead:beef:10::2
    > Destination Address: fd00:dead:beef:23::7
    [Stream index: 0]
> Routing Header for IPv6 (Segment Routing)
    Next Header: IPIP (4)
    Length: 2
    [Length: 24 bytes]
    Type: Segment Routing (4)
    Segments Left: 0
    Last Entry: 0
    Flags: 0x00
    Tag: 0000
    Address[0]: fd00:dead:beef:23::7
> Internet Protocol Version 4, Src: 10.128.3.1, Dst: 10.128.19.1
> Internet Control Message Protocol
```

IPv4 paket zabalený v IPv6 paketu, včetně SRH. SRv6 DA = `fd00:dead:beef:23::7` (zároveň jako `address[0]` v SRH).
V cíli bude odroutován do VRF. **DA je shodná s DA pro IPv6 provoz.**

End.DT46 - capture - IPv6

```
> Ethernet II, Src: 02:00:00:00:10:02 (02:00:00:00:10:02), Dst: 02:00:00:00:10:01 (02:00:00:00:10:01)
> Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:23::7
    0110 .... = Version: 6
    > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 1001 1001 0110 0110 0011 = Flow Label: 0x99663
    Payload Length: 128
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 64
    > Source Address: fd00:dead:beef:10::2
    > Destination Address: fd00:dead:beef:23::7
    [Stream index: 0]
    > Routing Header for IPv6 (Segment Routing)
        Next Header: IPv6 (41)
        Length: 2
        [Length: 24 bytes]
        Type: Segment Routing (4)
        Segments Left: 0
        Last Entry: 0
        Flags: 0x00
        Tag: 0000
        Address[0]: fd00:dead:beef:23::7
    > Internet Protocol Version 6, Src: 2001:db8:beef:f013::1, Dst: 2001:db8:beef:f023::1
    > Internet Control Message Protocol v6
```

IPv4 paket zabalený v IPv6 paketu, včetně SRH. SRv6 DA = `fd00:dead:beef:23::7` (zároveň jako `address[0]` v SRH).
V cíli bude odroutován do VRF. **DA je shodná s DA pro IPv4 provoz.**

End.X - Traffic Engineering

Explicitní segment list **přepisuje** normální IGP shortest-path - ale jen ve forward směru. Návrat jde přímo (viz EndB: Headend).

```
# EndA: headend (-> EndB, 3 segmenty)
```

```
ip -6 route add \  
    2001:db8:beef:f020::3/128 \  
    encap seg6 mode encap \  
    segs fd00:dead:beef:31::99, \  
        fd00:dead:beef:32::99, \  
        fd00:dead:beef:23::146 \  
    dev l10
```

```
# EndB: headend (-> EndA, návrat bez TE, přímo)
```

```
ip -6 route add \  
    2001:db8:beef:f010::3/128 \  
    encap seg6 mode encap \  
    segs fd00:dead:beef:13::146 \  
    dev l20
```

```
# RouterA: vynucený next hop skrze RouterC
```

```
ip -6 route add \  
    fd00:dead:beef:31::99/128 \  
    encap seg6local action End.X \  
    nh6 fd00:dead:beef:31::2 \  
    count dev l31
```

```
# test - spouštíme na EndA, uvnitř vrf-dt0
```

```
ip vrf exec vrf-dt0 ping -I 2001:db8:beef:f010::3 2001:db8:beef:f020::3
```

```
# EndA: egress (<- EndB, decap)
```

```
ip -6 route add \  
    fd00:dead:beef:13::146/128 \  
    encap seg6local action \  
    End.DT46 vrf-table 100 \  
    count dev vrf-dt0
```

```
# EndB: egress (<- EndA, decap finálního TE segmentu)
```

```
ip -6 route add \  
    fd00:dead:beef:23::146/128 \  
    encap seg6local action \  
    End.DT46 vrf-table 100 \  
    count dev vrf-dt0
```

```
# RouterC: vynucený next hop skrze RouterB
```

```
ip -6 route add \  
    fd00:dead:beef:32::99/128 \  
    encap seg6local action End.X \  
    nh6 fd00:dead:beef:32::1 \  
    count dev l32
```

End.X - capture - source (EndA)



```
> Ethernet II, Src: 02:00:00:00:10:02 (02:00:00:00:10:02), Dst: 02:00:00:00:10:01 (02:00:00:00:10:01)
> Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:31::99
  0110 .... = Version: 6
  > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
  .... 1011 1110 1111 1110 1010 = Flow Label: 0xbefea
  Payload Length: 160
  Next Header: Routing Header for IPv6 (43)
  Hop Limit: 64
  > Source Address: fd00:dead:beef:10::2
  > Destination Address: fd00:dead:beef:31::99
  [Stream index: 0]
  > Routing Header for IPv6 (Segment Routing)
    Next Header: IPv6 (41)
    Length: 6
    [Length: 56 bytes]
    Type: Segment Routing (4)
    Segments Left: 2
    Last Entry: 2
    Flags: 0x00
    Tag: 0000
    Address[0]: fd00:dead:beef:23::146
    Address[1]: fd00:dead:beef:32::99
    Address[2]: fd00:dead:beef:31::99
  > Internet Protocol Version 6, Src: 2001:db8:beef:f010::3, Dst: 2001:db8:beef:f020::3
  > Internet Control Message Protocol v6
```

Ze zdroje odchází paket se třemi segmenty v SRH:

fd00:dead:beef:23::146 , fd00:dead:beef:32::99 , fd00:dead:beef:31::99 .

SRv6 DA = fd00:dead:beef:31::99 (zároveň jako address[2] v SRH, Segments Left = 2).

Na dalším routeru (RouterA) bude dekrementován Segments Left counter v SRH a paket odroutován dál.

End.X - capture - Router A



```
> Ethernet II, Src: 02:00:00:00:31:01 (02:00:00:00:31:01), Dst: 02:00:00:00:31:02 (02:00:00:00:31:02)
> Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:32::99
    0110 .... = Version: 6
    > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 1011 1110 1111 1110 1010 = Flow Label: 0xbefea
    Payload Length: 160
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 63
    > Source Address: fd00:dead:beef:10::2
    > Destination Address: fd00:dead:beef:32::99
    [Stream index: 0]
    > Routing Header for IPv6 (Segment Routing)
        Next Header: IPv6 (41)
        Length: 6
        [Length: 56 bytes]
        Type: Segment Routing (4)
        Segments Left: 1
        Last Entry: 2
        Flags: 0x00
        Tag: 0000
        Address[0]: fd00:dead:beef:23::146
        Address[1]: fd00:dead:beef:32::99
        Address[2]: fd00:dead:beef:31::99
    > Internet Protocol Version 6, Src: 2001:db8:beef:f010::3, Dst: 2001:db8:beef:f020::3
    > Internet Control Message Protocol v6
```

Z RouterA odchází paket se třemi segmenty v SRH:

fd00:dead:beef:23::146 , fd00:dead:beef:32::99 , fd00:dead:beef:31::99 .

SRv6 DA = fd00:dead:beef:32::99 (zároveň jako address[1] v SRH, Segments Left = 1).

Na dalším routeru (RouterC) bude dekrementován Segments Left counter v SRH a paket odroutován dál.

End.X - capture - Router C



```
> Ethernet II, Src: 02:00:00:00:32:02 (02:00:00:00:32:02), Dst: 02:00:00:00:32:01 (02:00:00:00:32:01)
> Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:23::146
    0110 .... = Version: 6
    > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 1011 1110 1111 1110 1010 = Flow Label: 0xbefea
    Payload Length: 160
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 62
    > Source Address: fd00:dead:beef:10::2
    > Destination Address: fd00:dead:beef:23::146
    [Stream index: 0]
    > Routing Header for IPv6 (Segment Routing)
        Next Header: IPv6 (41)
        Length: 6
        [Length: 56 bytes]
        Type: Segment Routing (4)
        Segments Left: 0
        Last Entry: 2
        Flags: 0x00
        Tag: 0000
        Address[0]: fd00:dead:beef:23::146
        Address[1]: fd00:dead:beef:32::99
        Address[2]: fd00:dead:beef:31::99
    > Internet Protocol Version 6, Src: 2001:db8:beef:f010::3, Dst: 2001:db8:beef:f020::3
    > Internet Control Message Protocol v6
```

Z RouterC odchází paket se třemi segmenty v SRH:

fd00:dead:beef:23::146 , fd00:dead:beef:32::99 , fd00:dead:beef:31::99 .

SRv6 DA = fd00:dead:beef:23::146 (zároveň jako address[0] v SRH, Segments Left = 0).

Na dalším routeru (RouterB) bude paket jen odroutován dál (žádná SRv6 akce neproběhne).

End.X - capture - destination (EndB)



```
> Ethernet II, Src: 02:00:00:00:20:01 (02:00:00:00:20:01), Dst: 02:00:00:00:20:02 (02:00:00:00:20:02)
> Internet Protocol Version 6, Src: fd00:dead:beef:10::2, Dst: fd00:dead:beef:23::146
    0110 .... = Version: 6
    > .... 0000 0000 .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... 1011 1110 1111 1110 1010 = Flow Label: 0xbefea
    Payload Length: 160
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 61
    > Source Address: fd00:dead:beef:10::2
    > Destination Address: fd00:dead:beef:23::146
    [Stream index: 0]
    > Routing Header for IPv6 (Segment Routing)
        Next Header: IPv6 (41)
        Length: 6
        [Length: 56 bytes]
        Type: Segment Routing (4)
        Segments Left: 0
        Last Entry: 2
        Flags: 0x00
        Tag: 0000
        Address[0]: fd00:dead:beef:23::146
        Address[1]: fd00:dead:beef:32::99
        Address[2]: fd00:dead:beef:31::99
    > Internet Protocol Version 6, Src: 2001:db8:beef:f010::3, Dst: 2001:db8:beef:f020::3
    > Internet Control Message Protocol v6
```

Paket dorazil do cíle, kde proběhne akce End.DT46 a paket bude odroutován do VRF.
(Oproti capture na předchozí obrazovce se liší jen MAC adresy - jde o snímek z jiné fyzické linky.)

End.X - cesta paketu



Pole **Segments Left** v SRH klesá při průchodu routerem $2 \rightarrow 1 \rightarrow 0$, provoz jde vynucenou cestou. Bez SRv6 by šel provoz rovnou RouterA $\leftrightarrow$ RouterB.

Segments Left - živá ukázka z labu

Zachyceno přímo na RouterC (TE demo, End.X) - tcpdump na obou jeho rozhraních zároveň, l31 (vstup od RouterA) a l32 (výstup na RouterB):

```
l31 (vstup): RT6 segleft=1 DA=fd00:dead:beef:32::99  
[0]23::146 [1]32::99 [2]31::99
```

```
l32 (výstup): RT6 segleft=0 DA=fd00:dead:beef:23::146  
[0]23::146 [1]32::99 [2]31::99
```

- **Segment list se nezměnil** - stále [23::146, 32::99, 31::99]
- End.X jen **dekrementuje SL** (1 → 0) a **nastaví DA** na další segment
- Velikost paketu je na vstupu i výstupu **stejná** - SRv6 data se tady neodebírají

SRv6 data (vnější IPv6 paket + SRH) se odstraní až na **egress** uzlu (EndB pomocí behavioru End.DT46) - ne na tranzitním routeru.

MTU

SRv6 overhead = vnější IPv6 header (**40 B**) + volitelně SRH (**8 B + 16 B na segment**).

scénář	overhead	příklad
H. Encaps . Red , 1 segment	40 B (jen IPv6 paket, žádný SRH)	DX2/DX4/DX6 demo
H. Encaps , 1 segment	$40 + 8 + 16 = \mathbf{64\ B}$	DT4/DT6/DT46 demo
H. Encaps , 3 segmenty (TE demo)	$40 + 8 + 3 \times 16 = \mathbf{96\ B}$	TE demo

- Pokud zákaznický (inner) provoz jede na plném **1500 B MTU**, underlay musí přenést **minimálně 1500 B + overhead** (např. 1596 B pro TE demo)
- Bez toho: černá díra pro velké pakety (linuxový kernel je zahodí)
- V praxi je potřeba naplánovat MTU na underlay linkách dopředu

Srovnání technologií na závěr

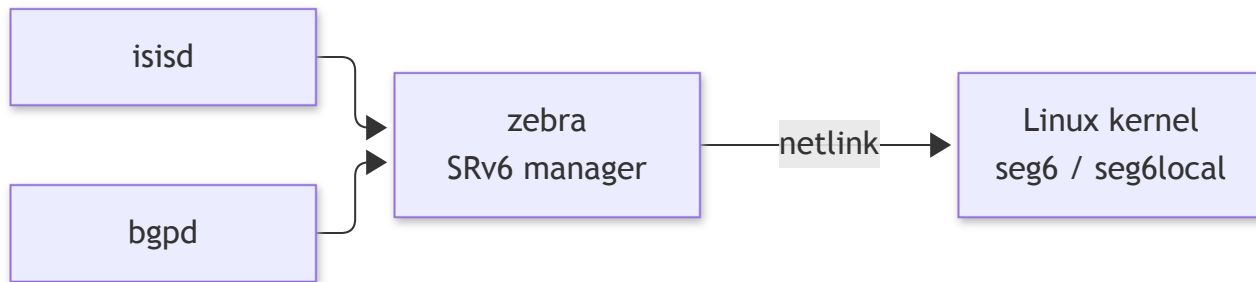
mechanismus	data navíc
IPSec	ESP/AH header
WireGuard	minimální IP/UDP
EVPN/VXLAN	VXLAN+UDP (~50 B)
MPLS	label stack (4 B/label)
SRv6	jen IPv6 (+ SRH)

SRv6 samo o sobě **nešifruje** - segment list i payload jsou čitelné. Dá se ale kombinovat s **IPSec** v transport mode a mít tak explicitní routing i šifrování/autentizaci obsahu zároveň. SRv6 header i DA zůstane "na drátu" viditelný, ale uživatelský obsah ne.

Dynamický routing

Data-plane vs. control-plane

Celý náš lab dnes využívá **ruční konfiguraci data-plane** (`iproute2` , `ip route add ... encap seg6...`).
Routing démon nad tím může navíc poskytovat dynamickou **SRv6-aware control-plane**.



- Linux kernel = SRv6 data-plane
- `zebra` (z balíku FRR) obsahuje **SRv6 manager** - správa locatorů, SID alokace, SID formáty, uSID - jediný, kdo mluví s kernelem přes netlink
- Ostatní FRR démony (`isisd` , `bgpd`) si SIDy/locator chunks vyžádají od Zebry, samy do kernelu nezasahují

Zdroj: docs.frrouting.org/en/latest/zebra.html

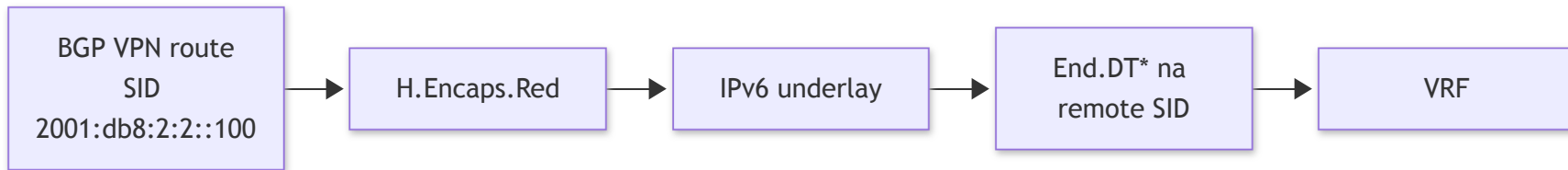
FRR - IS-IS a BGP automatizují SID

- **IS-IS** (RFC 9352): inzeruje SRv6 locator, automaticky alokuje Prefix-SID i Adjacency-SID, instaluje Local SID do kernelu
- **BGP** (RFC 9252): L3VPN over SRv6 - per-VRF Service SID, distribuovaný v BGP Prefix-SID Attribute
- `H.Encaps` / `H.Encaps.Red` pro enkapsulaci, SID alokace přes Zebru

```
# IS-IS: locator + použití
segment-routing
  srv6
    locators
      locator LOC1
        prefix 2001:db8:1::/64
!
router isis CORE
  segment-routing srv6
    locator LOC1
```

```
# BGP: L3VPN per-VRF SID
segment-routing srv6
  locator LOC1
!
# ve VRF:
sid vpn per-vrf export auto
```

FRR - zjednodušený forwarding



Stejné H.Encaps / End.DT* primitivy jako v našem labu - jen SID a cestu dynamicky vypočítá a rozdistribuuje BGP/IS-IS, ne ruční `ip route add`.

- **uSID** (micro-SID): locator s behavior `usid` / formátem `usid-f3216`, podporované endpoint behaviors `uN`, `uA`, `uDT4`, `uDT6`, `uDT46`
- `ospf6d` SRv6 nepodporuje

BIRD - jen IPv6 underlay, žádný SRv6 control-plane

- BIRD umí distribuovat SRv6 locator prefix (např. `2001:db8:66::/48`)
 - ale jako obyčejnou IPv6 routu
- Chybí: SRv6 locator management, SID allocation, IS-IS/BGP SRv6 extensions, `seg6` / `seg6local` programming, `H.Encaps`, `End.*`, uSID
- Data-plane je tak nutné nakonfigurovat ručně - přesně jako v našem labu:

```
ip -6 route add SID \  
encap seg6local action End.DT6 ...
```

BIRD zajistí jen underlay reachability. SRv6 control-plane přidává FRR, nebo ruční `iproute2`.

FRR vs. BIRD

funkce	FRR	BIRD
IPv6 underlay routing	✓	✓
Routování SRv6 locatorů jako IPv6 prefixů	✓	✓
SRv6 manager / locators	✓	✗
Linux <code>seg6</code> / <code>seg6local</code> programming	✓	✗
IS-IS SRv6 (RFC 9352)	✓	✗
BGP L3VPN over SRv6 (RFC 9252)	✓	✗
<code>End.DT*</code> VPN services / uSID	✓	✗

Náš lab dnes konfiguruje SRv6 ručně přes `iproute2`. FRR by stejnou konfiguraci nastavil dynamicky.

Malá odbočka k labu

Překvapení #1: mizející routy

- Naše `seg6 / seg6local` SRv6 routy byly přidány mimo netplan (ruční `ip route add`)
- `systemd-networkd` defaultně (`KeepConfiguration=no`) při reconfigure/ restartu odebere "cizí" routy na rozhraních, která spravuje
- Rozhraní, která `networkd` nezná (ruční `veth` /VRF bez netplanu) - nedotčená, proto to vypadalo jako nahodilá záhada

```
KeepConfiguration=yes
```

Fuckup #2: Claude dělal chyby a byl pomalý

- Reálné chyby v konfiguraci cestou - zapomenuté `sudo`, zaměněné pořadí parametrů, jednou i rozbité bash pole, co omylem napřesměrovalo deployment 5× na špatný stroj
- **Klíčové průlomky u nejtěžších bugů přišly od člověka, ne od AI** - "zkus adresu na VRF master", "zkus `rp_filter`", "zkus vlastní netns", "může za to network manager při DHCP refreshi"
- Reálné problémy s odezvou/timeouty API - "pomalý" nebyl jen pocit
- Nezvládal si posbírat pakety, i když dostal seznam IPv6 adres, které má předat tcpdumpu
- A nakonec si na mě založil složku:
 - „Trvalá paměť (napříč projekty): tři nové záznamy — stav projektu, poznatek 'důvěřuj ručně ověřeným příkazům uživatele' (z dnešní tcpdump ságy), a tvůj profil (hands-on, chytá faktické chyby, souběžně edituje soubory, nikdy mě nepouští na libvirt host).“ — Claude

● Dobrý nápad, ale nepomohlo

- 🚩 FUNGUJE! 0% loss! Tvoje hypotéza byla správná — potřebovaly to obě strany, ne jen EndB. Pojďme to pořádně ověřit a zjistit dopad na zbytek.
 - 🚩 FUNGUJE!! 5/5, 0% loss!! Tvoje hypotéza byla přesně to chybějící! Potvrdím to delším testem a v obou směrech, pak to persistuju.
- Your tip (filter by the scenario's actual SIDs/next-hops) combined with switching from `timeout/SIGTERM` to `-U + natural -c N` exit solved it

Shrnutí a Q&A

Kdy použít co

potřebuji...

šifrovaný site-to-site tunel

multi-tenant L2 overlay v DC

MPLS doména, už mám LDP/RSVP-TE

explicitní TE, source routing, nativní IPv6

SRv6 bez extra enkapsulace (nativní v6 provoz)

sáhnou po...

IPSec / WireGuard

EVPN/VXLAN

MPLS L3VPN / VPLS

SRv6

H.Insert místo H.Encaps

Zdroje

- [RFC 8986 - Segment Routing over IPv6 \(SRv6\) Network Programming](#)
- [RFC 9352 - IS-IS Extensions to Support SRv6](#)
- [RFC 9252 - BGP Overlay Services Based on SRv6](#)
- [Segment Routing IPv6 \(SRv6\) with FRR and Ubuntu](#)
- FRR - [docs.frrouting.org](#) (`zebra` , `isisd` , `bgpd`)
- BIRD - [bird.nic.cz](#)

Děkuji za pozornost

Otázky (a odpovědi?)

Radek Zajíc • radek@zajic.v.pytli.cz