Leveraging IPv6 Segment Routing for Service Function Chaining

David Lebrun
<david.lebrun@uclouvain.be>

ICTEAM
Université catholique de Louvain
Louvain-la-Neuve, Belgium

IPv6 Segment Routing

- Source routing paradigm
- IPv6 Routing Header extension (type 4)
- Path encoded as stack of segments (IPv6 addresses)
- Segments distributed through IGP

IPv6 SR: illustration (1)

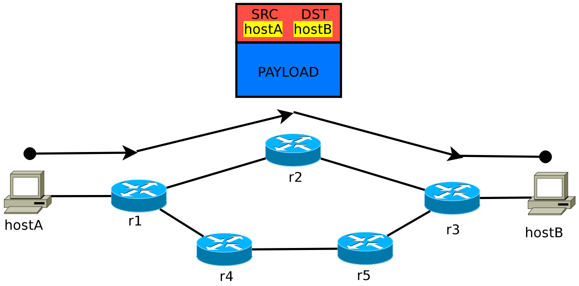

Figure: Regular shortest-path packet forwarding

IPv6 SR: illustration (2)

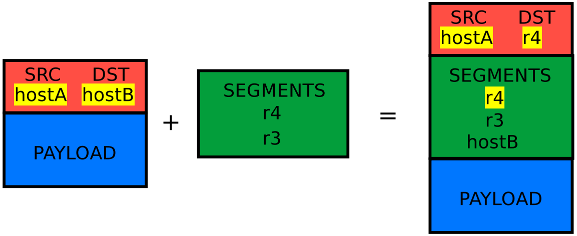

Figure: Transformation of a regular IPv6 packet into an SR-enabled packet. Note that the original final destination of the packet is appended at the end of the segment list and the first segment of the segment list is set as the destination address of the packet.

IPv6 SR: illustration (3)

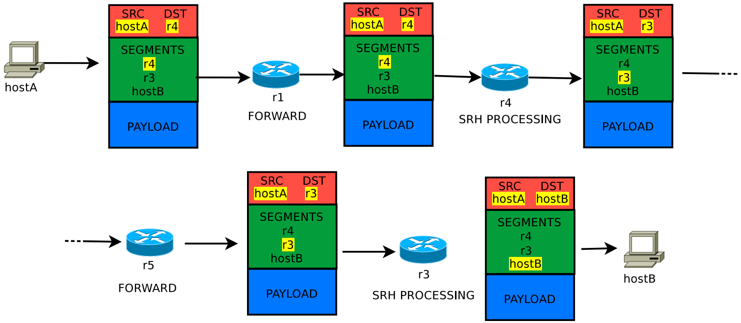

Figure: Journey of an SR-enabled packet through the example network. Note that at each segment endpoint, the destination address of the packet is updated to the next segment address.

Service Function Chaining with SR

- In "regular" Segment Routing, a segment is a waypoint. We extend that definition so that a segment can also represent a service.
- Upon packet reception, segment endpoint applies a service
  1. **Specific services**: in-kernel, simple, fast (e.g. header field rewriting)
  2. **Generic services**: application-controlled, programmable, more complex, slower
- We focus on generic services.

SFC-SR: configuration

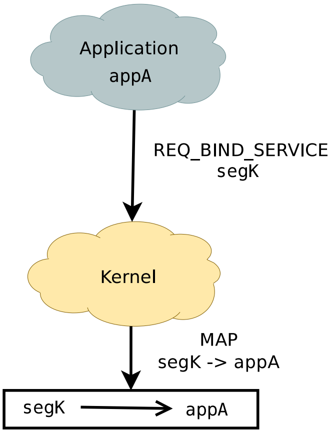

Figure: An application sends a request to receive packets with active segment `segK`

SFC-SR: processing

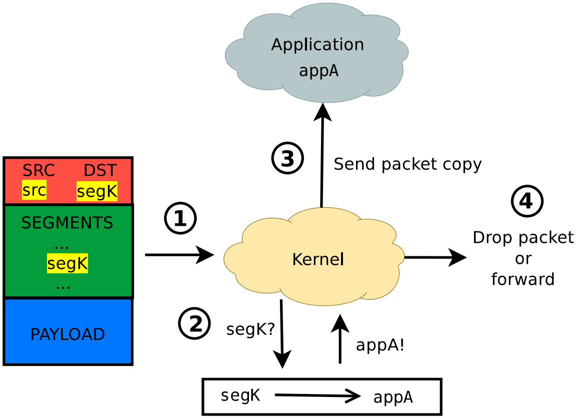

Figure: A packet is received and correspondingly transferred to the application.

SFC-SR: post-processing

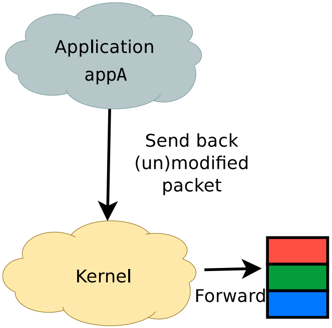

Figure: Optional: The application sends back the packet (either modified or unmodified) and the kernel forwards it into the network

SFC-SR: usecases

- Accouting
- DPI
- Encryption/decryption
- Compression/decompression
- Proxy
- On-the-fly video transcoding
- ...

Implementation architecture

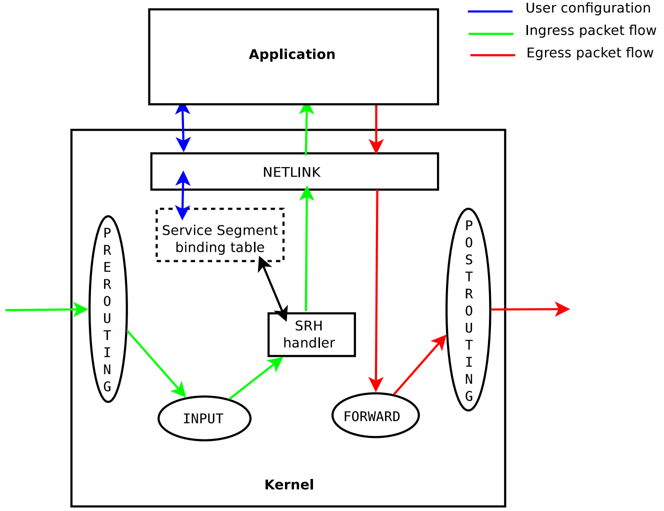

Figure: High-level overview of the main kernel components used in our implementation

Evaluation

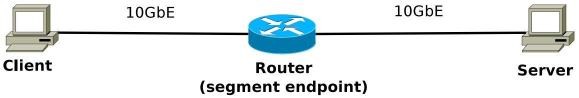

Figure: Experimental network used for evaluation

- 3 servers, 10GbE links
- 64 byte UDP packets sent with `iperf3`
- Service Segment handled by single-core packet counter

Evaluation

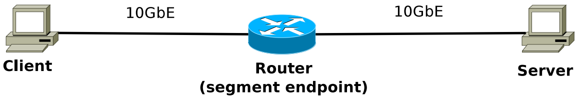

Figure: Experimental network used for evaluation

Experiment	Line rate	Packets/s	Bytes/s
Non-SR	10G	683 Kpps	330 Mbps
Non-SR	100M	205 Kpps	100 Mbps
SR (fwd)	10G	680 Kpps	328 Mbps
SR (fwd)	100M	201 Kpps	99 Mbps
SR (sync)	100M	195 Kpps	98 Mbps
SR (async)	100M	176 Kpps	90 Mbps

Table: This table shows the throughput in (64-byte) packets per second, for each experiment

Issues and future work

- Synchronous perfs better than asynchronous
- Perfs collapse at about 200Kpps
  - Zero-copy mode (`NETLINK_{T,R}X_RING`) needed for fast data transfer
  - No queues in zero-copy mode: ring full $\rightarrow$ packet dropped
  - Limits of NETLINK ?
  - Solutions ? Code path optimization, multithreading, DPDK-style engine, ...
- Designing and implementing relevant services
  - E.g.: tcp seq num adjustment for compression service
- Suggestions are very welcome !

References

- http://segment-routing.org
- http://github.com/segment-routing/
- https://tools.ietf.org/html/draft-previdi-6man-segment-routing-header-08