EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

bw
NET

FACULTY OF
SCIENCE
Communication Networks

A Caching SFC Proxy Based on eBPF

Marco Häberle, Benjamin Steinert, Michael Weiss, Michael Menth

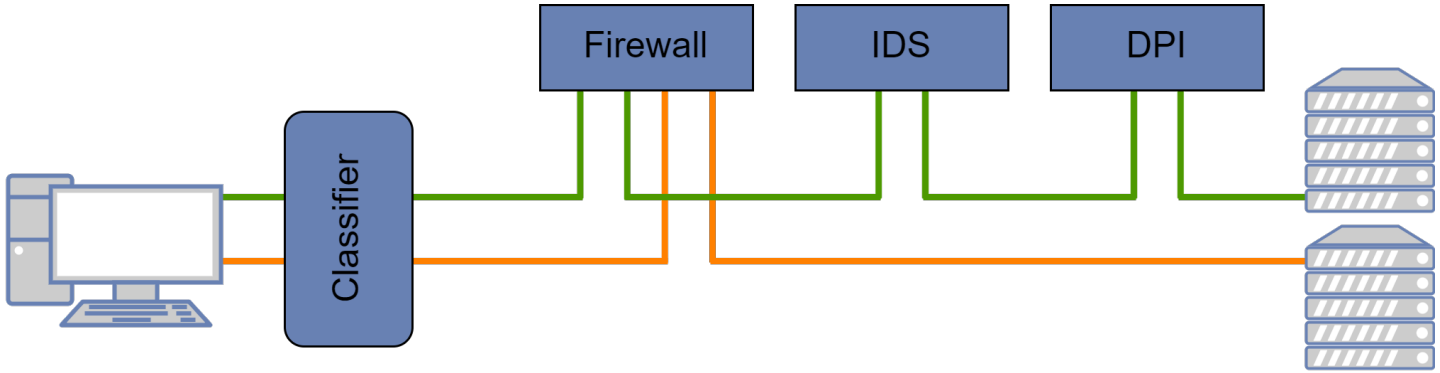
http://kn.inf.uni-tuebingen.de

Outline

- Motivation
- eBPF and XDP
- Caching SFC Proxy
  - Principles
  - Header Caching
  - Prototype
- Evaluation

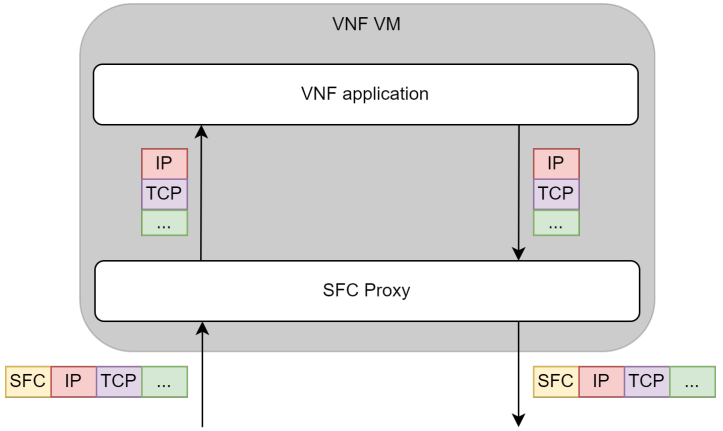
Motivation

- Service Function Chaining (SFC)
  - Steering of packets through an ordered set of service functions
  - SFC classifier assigns each packet to a Service Function Chain
  - SFC encapsulation: Identifies chain and position in path
    - Network Service Header (NSH)
    - Alternative: Service Programming with Segment Routing (SRv6, SR-MPLS)



Motivation

- SFC Proxies
  - Enable using SFC-unaware Service Functions (SF) in an SFC-enabled domain
    - Remove SFC encapsulation before processing by SF, add it again afterwards



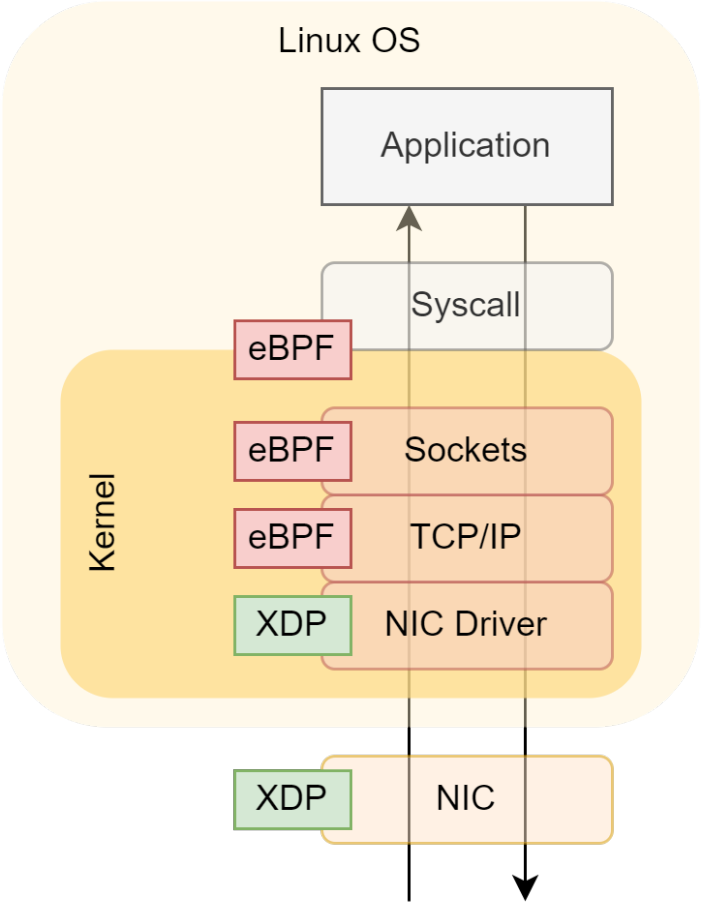
- Problem: Existing SFC proxies don‘t cache headers
  - Information about chains either configured statically or learned
  - No support for dynamic chains (e.g. skipping of specific service functions)
  - In-packet metadata is lost at proxy
- Solution: Cache headers at SFC proxy implemented with eBPF

eBPF and XDP

- eBPF – extended Berkeley Packet Filter
  - Technology for executing programs isolated and secure within the Linux kernel
  - No need to change kernel source code or load custom kernel modules
  - Event-driven execution, different hooks available (System calls, kernel tracepoints, network events, XDP,…)

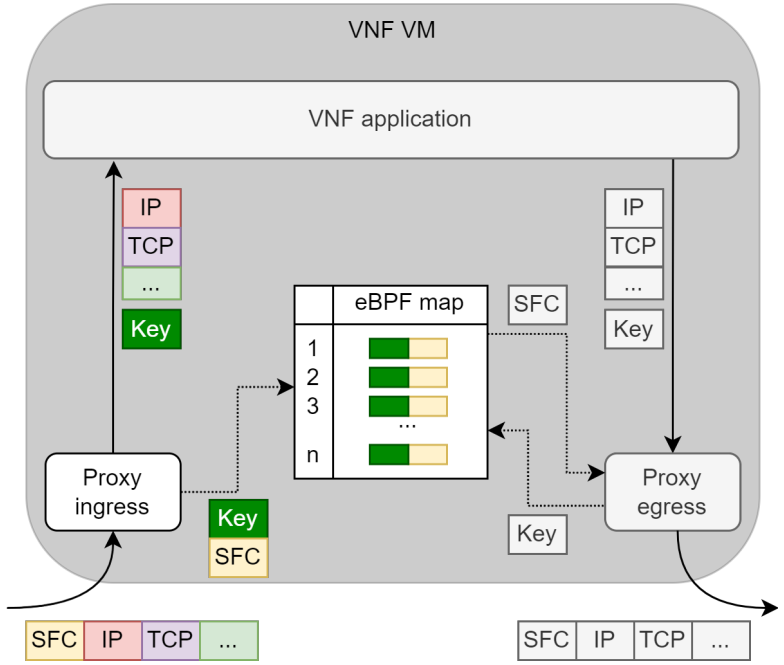
- XDP – eXpress Data Path
  - Idea: Provide hook for eBPF packet processing at the earliest possible stage in the network stack
  - Processing logic is given via a user-defined eBPF program

⇒ Programmable, high-performant packet processing within the Linux kernel



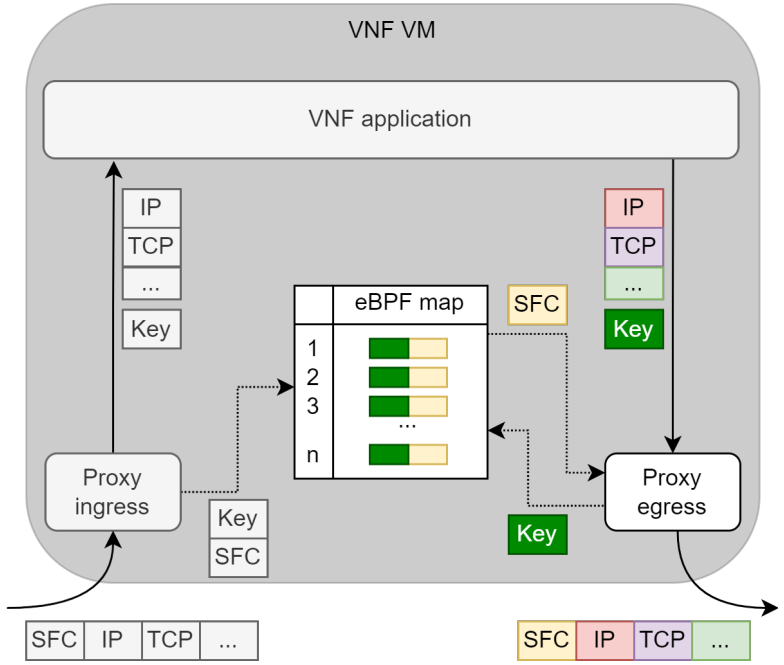
Caching eBPF-based Proxy - Principle

- Idea: Cache SFC headers
- Workflow
  - Ingress
    1) Packet received by ingress Proxy
    2) Store SFC headers in a map
    3) Send packet to VNF without SFC headers
  - Processing by VNF



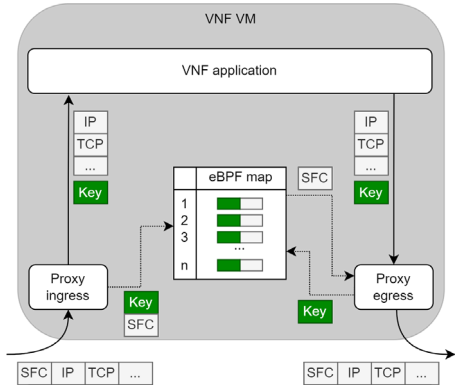
Caching eBPF-based Proxy - Principle

- Idea: Cache SFC headers
- Workflow
  - Ingress
    1) Packet received by ingress Proxy
    2) Store SFC headers in a map
    3) Send packet to VNF without SFC headers
  - Processing by VNF
  - Egress
    1) Packet received by egress Proxy
    2) Fetch SFC headers from map
    3) Add SFC headers to packet



Caching eBPF-based Proxy – Header Caching

- Key needed to cache headers
  - Packets before and after VNF must be identifiable



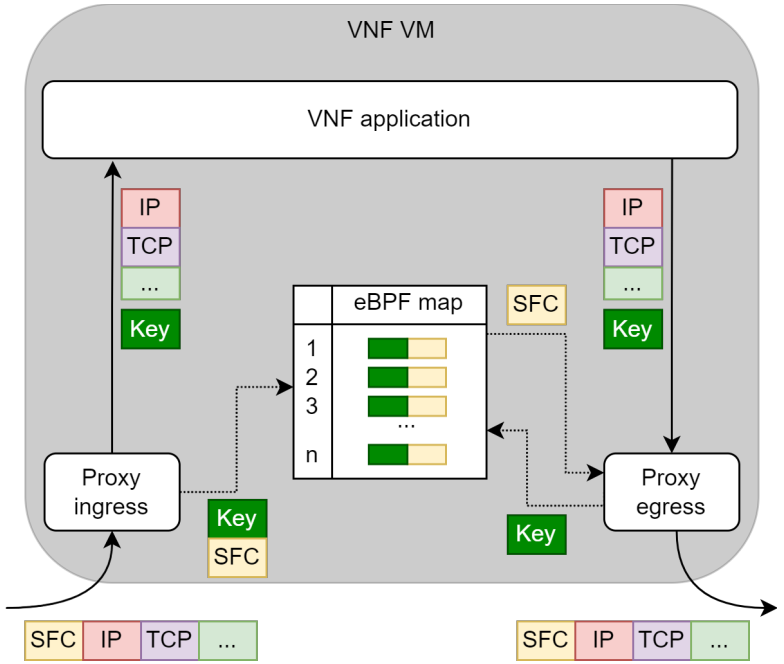
- Two operation modes:

1) Add artificial key to packet metadata
   - Generate artificial key (sequence number or random) at ingress
   - Store key in sk_buff
   - Limitation: Only possible for kernel mode VNFs (e.g. iptables)

2) Calculate hash from packet headers
   - Hashed headers need to be unique and immutable
     - IPv4 + TCP: IP src, IP dst, IP id, TCP src port, TCP dst port, TCP seq, TCP ack
     - IP + RTP: IP src, IP dst, RTP timestamp
     - …
   - Works with kernel and user mode VNFs
   - Limitation: VNF may not change the headers that form the key
     - Not compatible with e.g., NAT

Prototype: Design

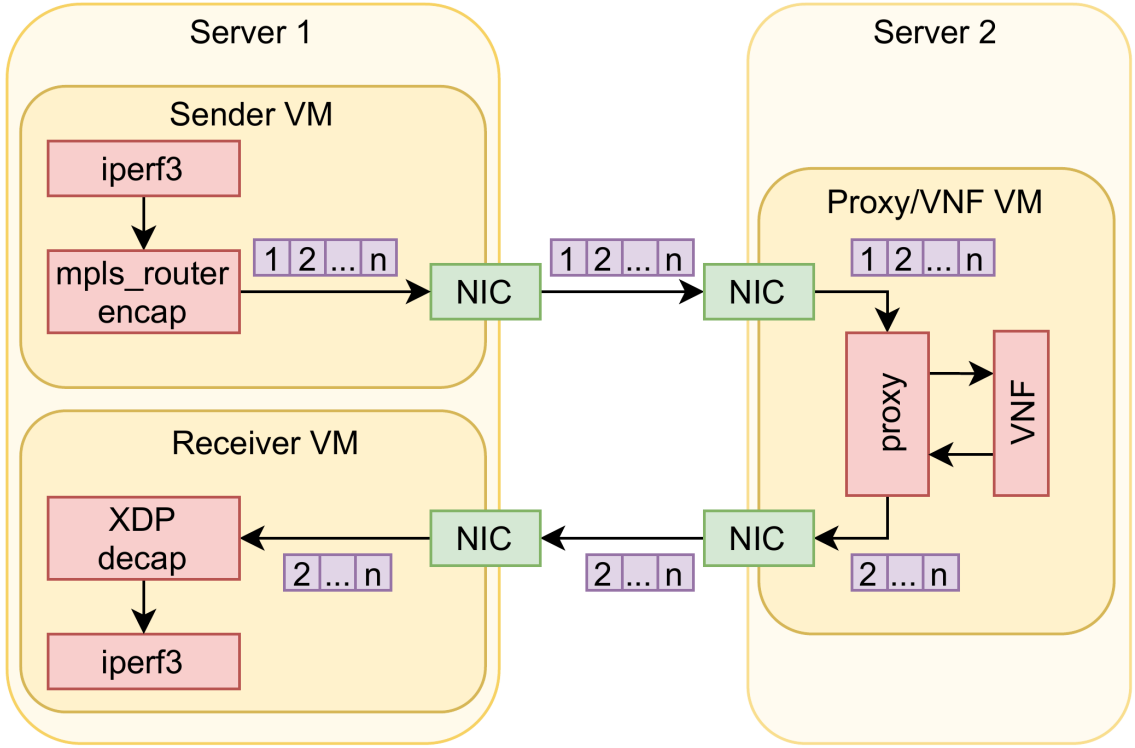
- SR-MPLS
- Based on eBPF/XDP (XDP at egress not implemented in Linux kernel)
- RX: XDP
  - Parse packet
  - Pop all MPLS headers
  - Place MPLS headers in cache (eBPF map)
- TX: eBPF
  - Parse packet
  - Fetch and remove MPLS headers from cache
  - Push MPLS labels on packet (except outermost label)
- Shared eBPF hashmap serves as header cache
  - Resolution of hash collisions: full key is stored with values



- Extension of proxy possible
  - INT
  - Proof of Transit
  - …

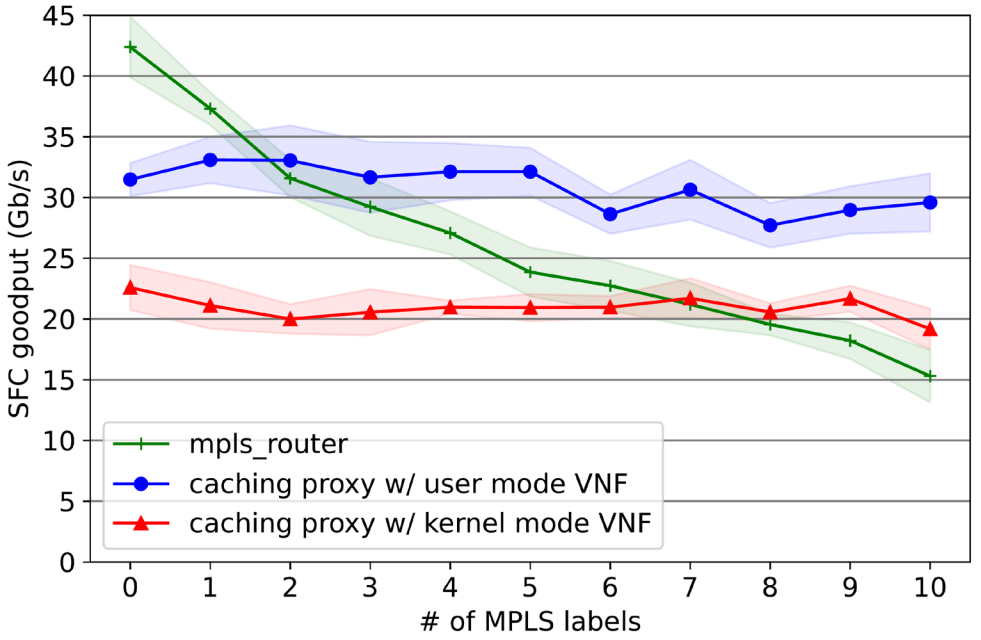
Performance Evaluation – Testbed

- Setup
  - 3 VMs
    - Sender
      - mpls_router encapsulation
      - Iperf3 client
    - VNF
      - Caching proxy
      - Comparison: mpls_router as proxy
    - Receiver
      - MPLS decapsulation with XDP
      - Iperf3 server
  - 100G connectivity
  - Baseline throughput: approx. 60 Gb/s



Performance Evaluation – Throughput Comparison

- Throughput
  - mpls_router
    - Linux kernel module used as static proxy
    - Small number of labels: highest performance
    - High number of labels: lowest performance
  - Caching proxy
    - Number of MPLS labels less significant
    - Outperforms mpls_router for large number of MPLS labels
    - Large gap between proxy variants
      - Caused by XDP execution points
      - User mode (hashed headers): Executed in NIC driver
      - Kernel mode (key in metadata): Executed in Kernel network stack
      - Mellanox mlx5 driver does not support adding metadata to sk_buff



Conclusion

- SFC proxies required for SFC-unaware service functions
- Existing proxies not able to cache headers
  - No in-packet metadata
  - No dynamic chains
- eBPF enables implementation of more advanced proxies
  - Header caching is feasible
  - Production-ready throughput
  - Extensible (INT, In-band OAM, …)

Contact

Marco Häberle
marco.haeberle@uni-tuebingen.de
University of Tuebingen, Dept. of Computer Science
Chair of Communication Networks
Sand 13, 72076 Tuebingen, Germany
https://kn.inf.uni-tuebingen.de/