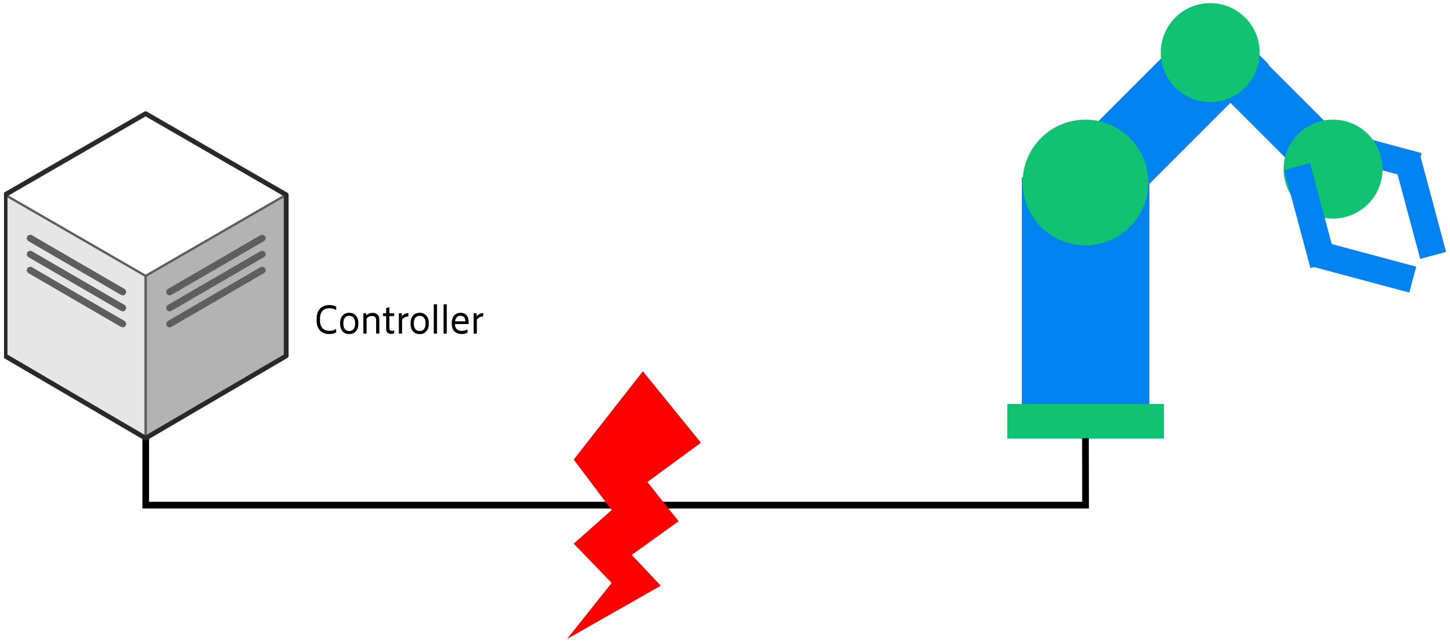
Lightweight Implementation of Per-packet Service Protection in eBPF/XDP

Ferenc Fejes, Balázs Varga, János Farkas (Ericsson Research Traffic Lab)

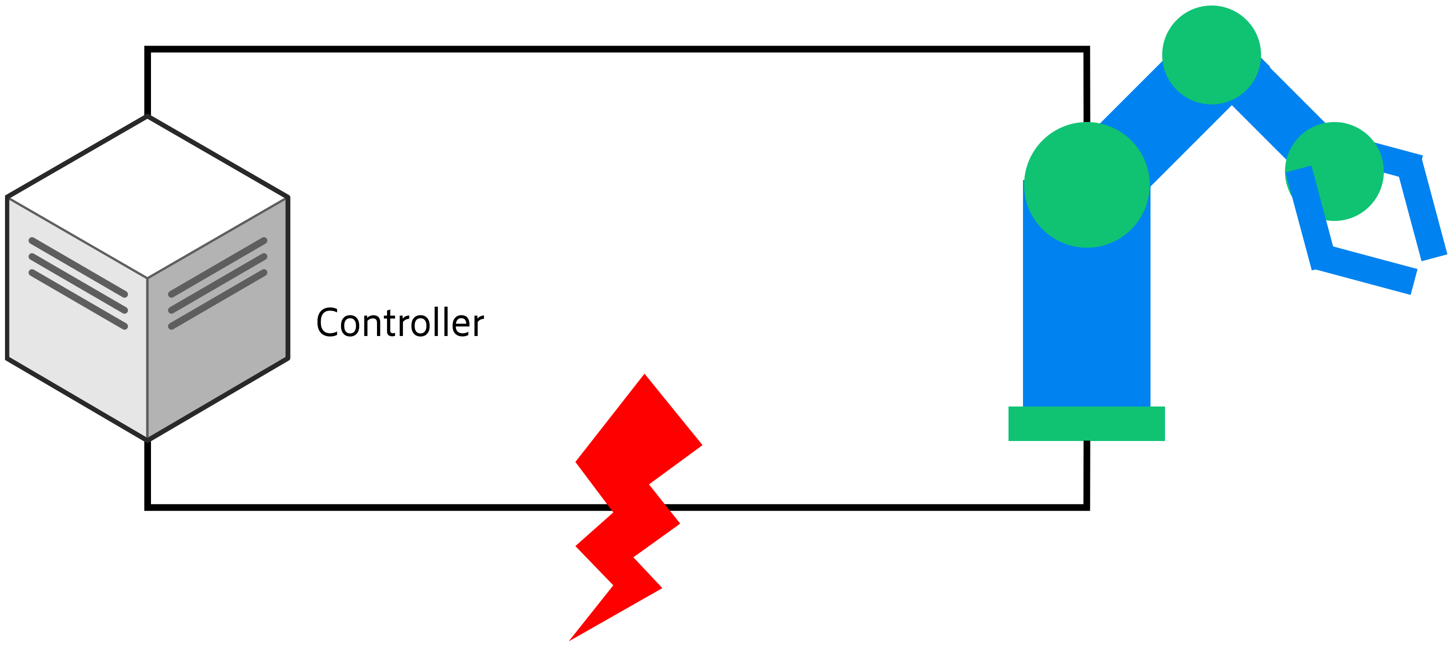
Ferenc Orosi (Eötvös Loránd University)

source: pixabay.com

Failure of the network can be costly and dangerous



Network redundancy can be added to reduce the chance of failure



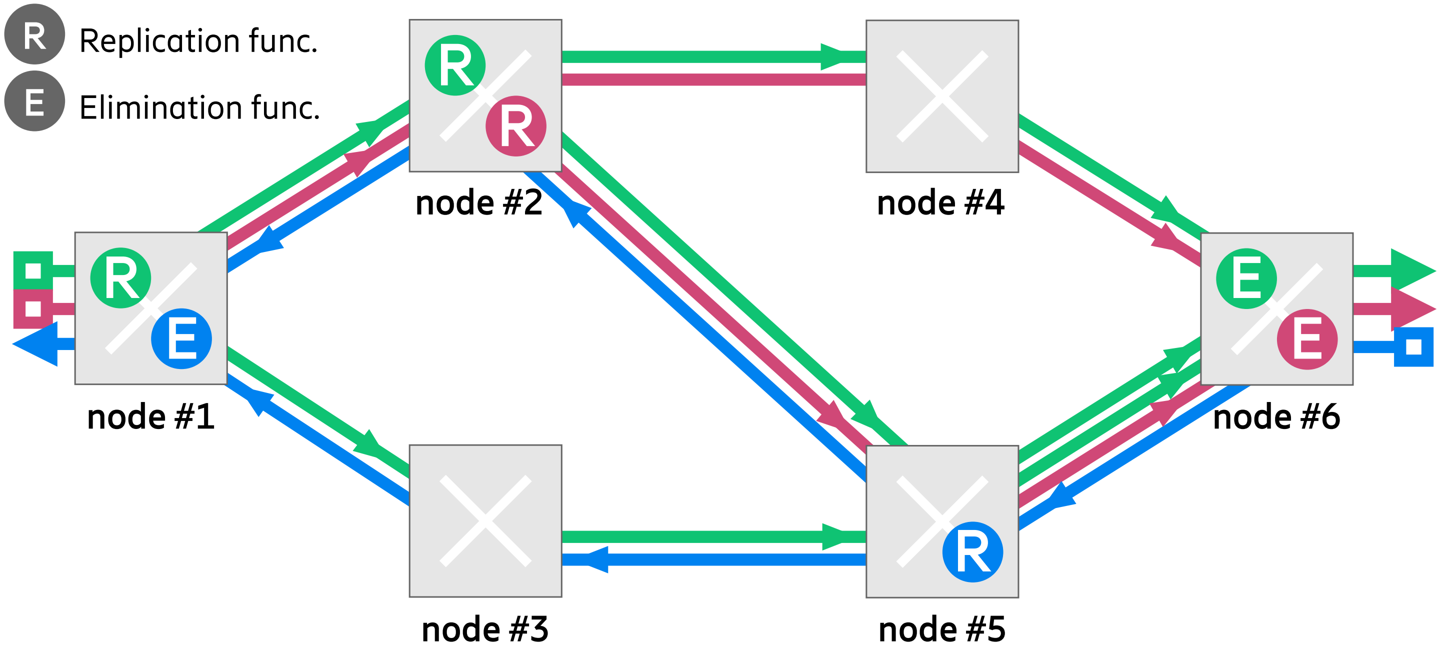
Network redundancy

- Require some level of physical layer redundancy e.g. disjoint cable paths, different frequency bands
- Vendors used to implement their own redundancy solution - not standard, out of scope
- We focusing on standardized Layer 2 redundancy

IEEE 802.1CB — Frame Replication & Elimination for Reliability (FRER)

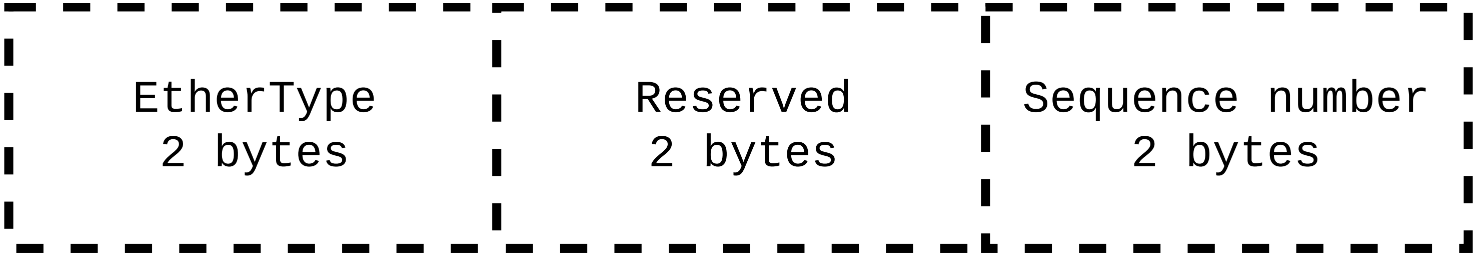
- Vendor agnostic standard for Layer 2 per-packet service protection
- Published in 2017, extended in 2021 (IEEE 802.1CBdb)
- Main functions: **replication** and **elimination**

FRER operation example



FREER encapsulation

- 6 bytes length redundancy-tag (R-tag), including ethertype, 2 reserved bytes and the sequence number



EtherType
2 bytes

Reserved
2 bytes

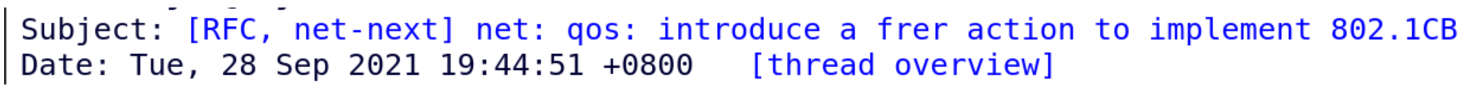
Sequence number
2 bytes

FRER R-tag

FRER support in Linux

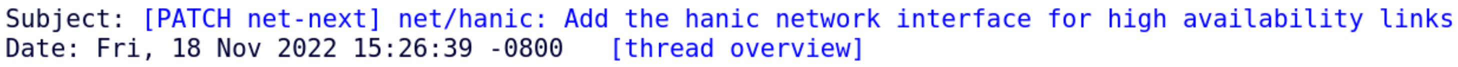
- There are switches on the market supporting FRER and running Linux (e.g. Marvell, Microchip, NXP)
- No mainline Linux support
- Proposals exists, submitted to the mailinglist

NXP



Subject: [RFC, net-next] net: qos: introduce a frer action to implement 802.1CB
Date: Tue, 28 Sep 2021 19:44:51 +0800 [thread overview]

Cruise LLC



Subject: [PATCH net-next] net/hanic: Add the hanic network interface for high availability links
Date: Fri, 18 Nov 2022 15:26:39 -0800 [thread overview]

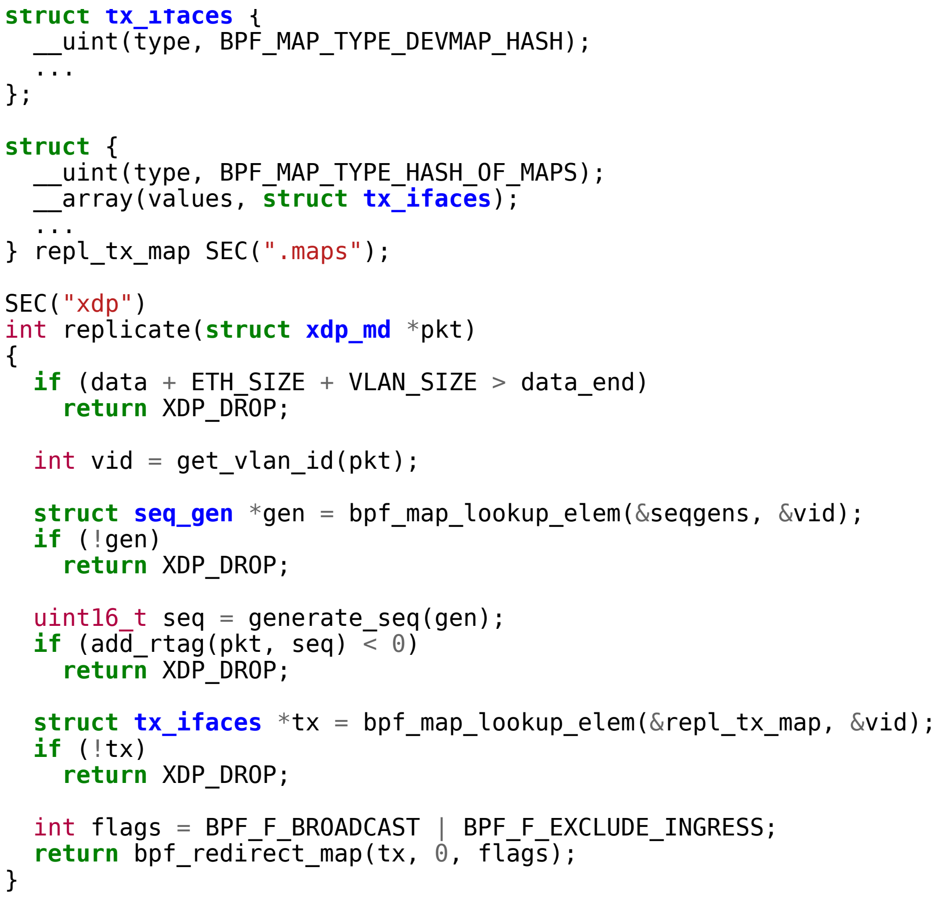
XDP FRER

- FRER functions can be implemented with the XDP APIs
- R-tag push/pop with head adjustment support
- Replication with broadcast transmission flag
- Elimination enabled by BPF locking API

XDP FRER R-tag push example

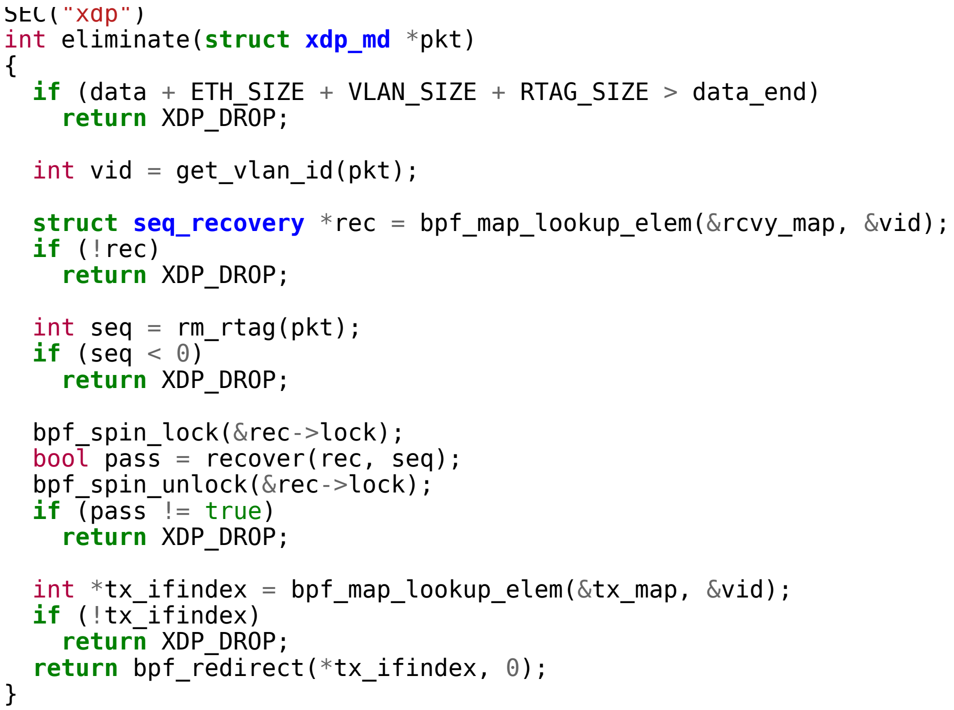
```
static inline int add_rtag(struct xdp_md *pkt, uint16_t seq)
{
  if (bpf_xdp_adjust_head(pkt, 0 - RTAG_SIZE))
    return -1;

  void *data = (void *)(long) pkt->data;
  void *data_end = (void *)(long) pkt->data_end;
  if(data + ETH_SIZE + VLAN_SIZE + RTAG_SIZE > data_end)
    return -1;

  /* Move Ethernet+VLAN headers to the front of the buffer */
  /* memmove(destionation_addr, source_addr, size) */
  __builtin_memmove(data, data + RTAG_SIZE, ETH_SIZE + VLAN_SIZE)
  struct vlan_hdr *vhdr = data + ETH_SIZE;
  struct rtaghdr *rtag = data + ETH_SIZE + VLAN_SIZE;

  /* Prepare the R-tag */
  __builtin_memset(rtag, 0, RTAG_SIZE);
  rtag->nexthdr = vhdr->h_vlan_encapsulated_proto;
  vhdr->h_vlan_encapsulated_proto = bpf_htons(0xf1c1);
  rtag->seq = bpf_htons(seq);

  return 0;
}
```

XDP FRER replication example



```c
struct tx_ifaces {
  __uint(type, BPF_MAP_TYPE_DEVMAP_HASH);
  ...
};

struct {
  __uint(type, BPF_MAP_TYPE_HASH_OF_MAPS);
  __array(values, struct tx_ifaces);
  ...
} repl_tx_map SEC(".maps");

SEC("xdp")
int replicate(struct xdp_md *pkt)
{
  if (data + ETH_SIZE + VLAN_SIZE > data_end)
    return XDP_DROP;

  int vid = get_vlan_id(pkt);

  struct seq_gen *gen = bpf_map_lookup_elem(&seqgens, &vid);
  if (!gen)
    return XDP_DROP;

  uint16_t seq = generate_seq(gen);
  if (add_rtag(pkt, seq) < 0)
    return XDP_DROP;

  struct tx_ifaces *tx = bpf_map_lookup_elem(&repl_tx_map, &vid);
  if (!tx)
    return XDP_DROP;

  int flags = BPF_F_BROADCAST | BPF_F_EXCLUDE_INGRESS;
  return bpf_redirect_map(tx, 0, flags);
}
```

XDP FRER elimination example

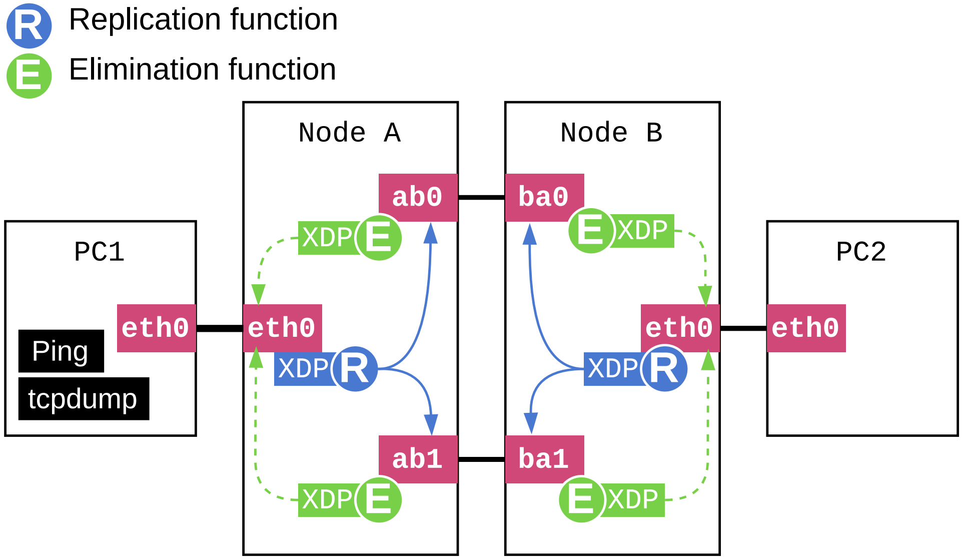


```c
SEC("xdp")
int eliminate(struct xdp_md *pkt)
{
  if (data + ETH_SIZE + VLAN_SIZE + RTAG_SIZE > data_end)
    return XDP_DROP;

  int vid = get_vlan_id(pkt);

  struct seq_recovery *rec = bpf_map_lookup_elem(&rcvy_map, &vid);
  if (!rec)
    return XDP_DROP;

  int seq = rm_rtag(pkt);
  if (seq < 0)
    return XDP_DROP;

  bpf_spin_lock(&rec->lock);
  bool pass = recover(rec, seq);
  bpf_spin_unlock(&rec->lock);
  if (pass != true)
    return XDP_DROP;

  int *tx_ifindex = bpf_map_lookup_elem(&tx_map, &vid);
  if (!tx_ifindex)
    return XDP_DROP;
  return bpf_redirect(*tx_ifindex, 0);
}
```

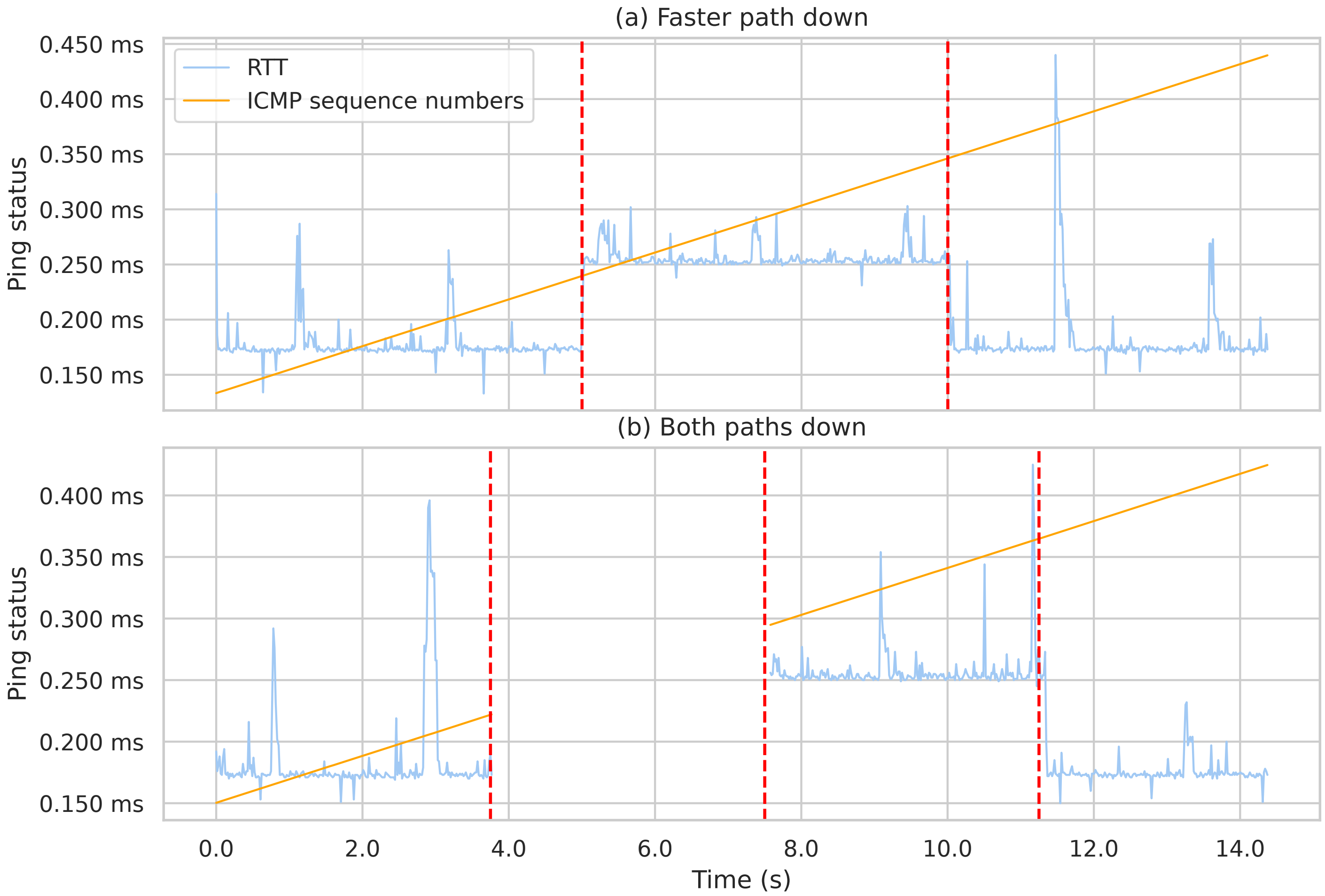
Evaluation

Testbed

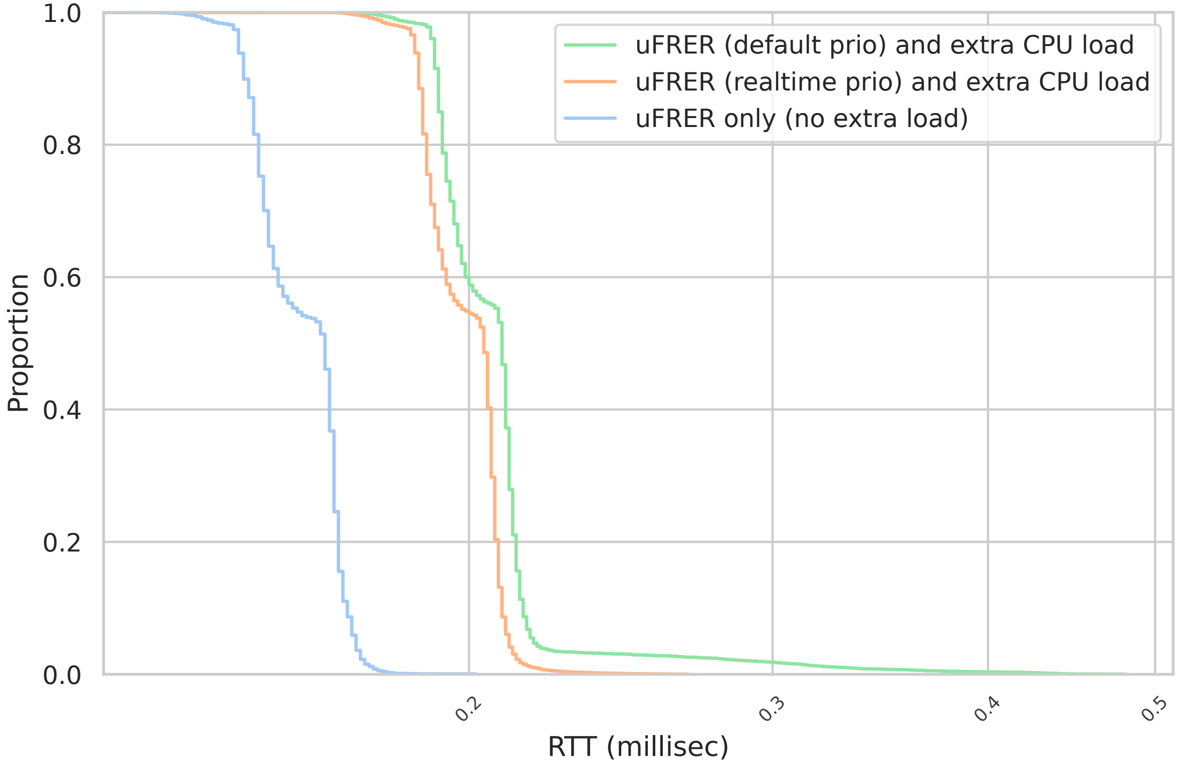
- EPYC 7402P CPU 24 cores fixed to 2.8GHz, Intel i225 NICs
- Ubuntu 23.04 (kernel v6.2 distro default), ping traffic, stress-ng load
- Tuning: irqbalance off, TX/RX queues pinned to isolated cores
- Baseline: userspace FRER (uFRER) using **AF_PACKET** API



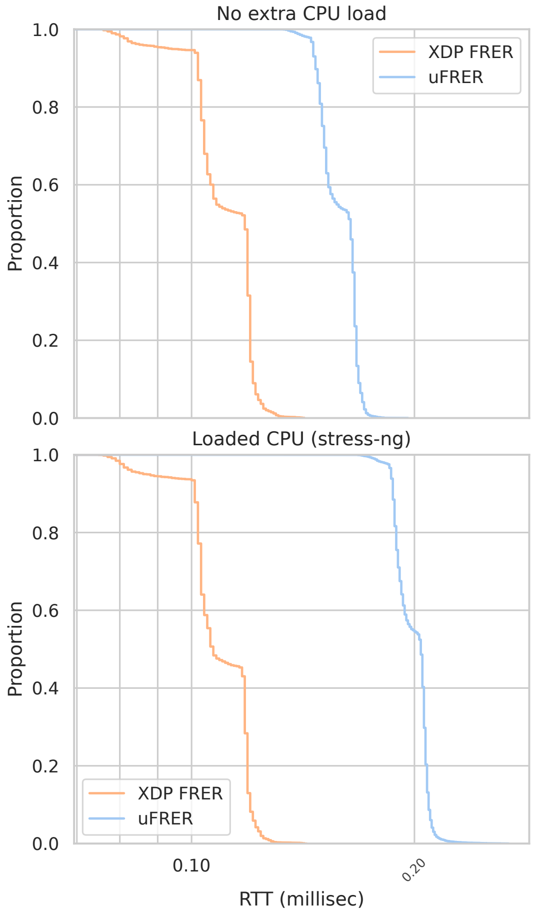
Link failure and XDP FRER



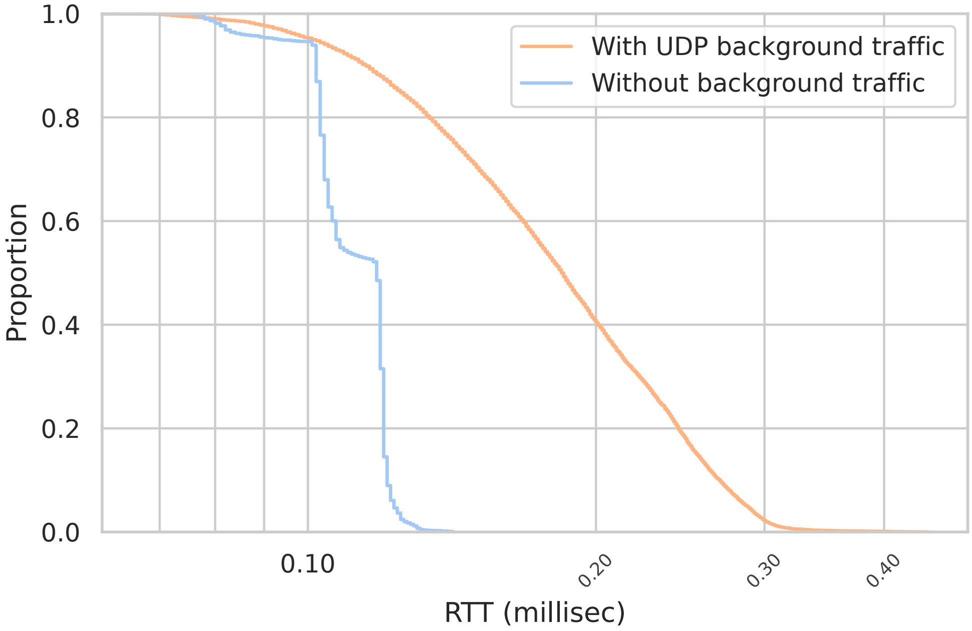
Userspace FRER and CPU load



XDP FRER vs uFRER with CPU load



XDP FRER with background traffic



Summary

- IEEE 802.1CB FRER can be implemented with XDP APIs
- Lightweight and portable (libs: *libbpf* and *libxdp*)
- Better average and tail latencies compared to userspace **AF_PACKET** implementation
- No hardware offload with XDP FRER

Thank you!

E-mail: ferenc.fejes@ericsson.com

Slides & paper: https://netdevconf.info/0x17/25