Linux Kernel Support for IOAM Direct Exporting

Maxime Goffart, Justin Iurman, Emilien Wansart, Benoit Donnet

Table of Contents

Table of Contents

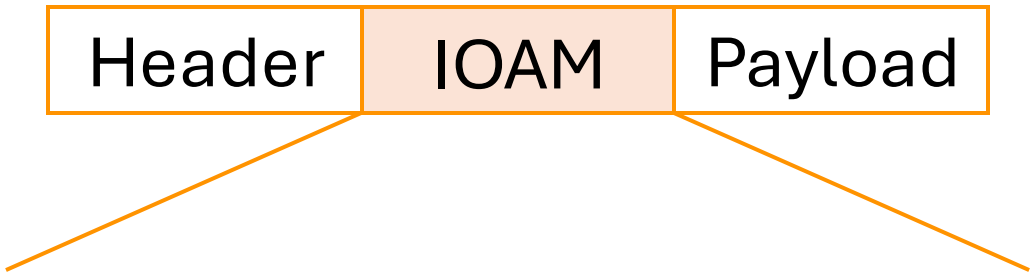
What is IOAM ?

In Situ Operations, Administration, and Maintenance

Standardized by IETF

- RFC 9197 for data fields
- RFC 9378 for deployment
- …

Hybrid Telemetry Method



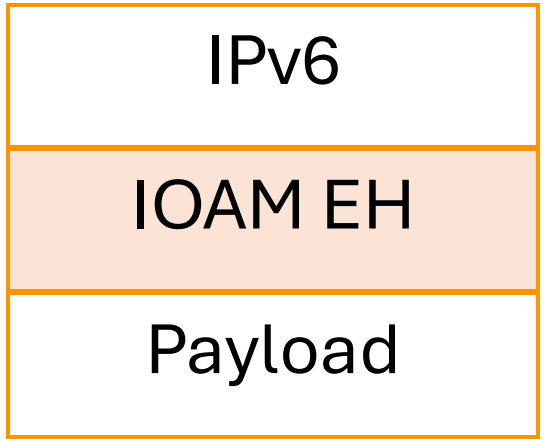
- Instructions on operations to perform
- (Potentially) telemetry data

IOAM – Encapsulation

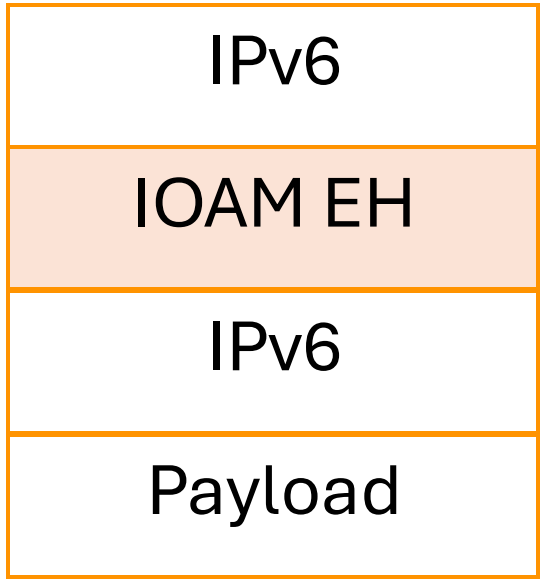
Encapsulation in Network Protocols

- NSH (RFC 9452)
- IPv6 (RFC 9486)

Inline

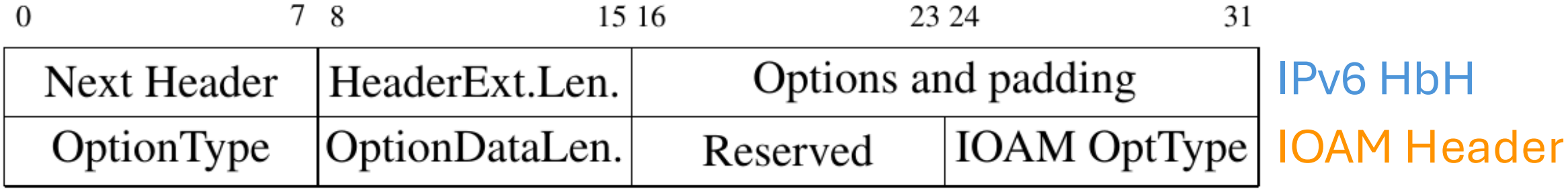


Encapsulation



IOAM – Operation Modes (option-types)

Option-Type	IOAM OptType
Pre-allocated Trace Option-type (PTO)	0
Incremental Trace Option-type	1
Proof-Of-Transit (POT)	2
Edge-to-Edge (E2E)	3
Direct Exporting (DEX)	4



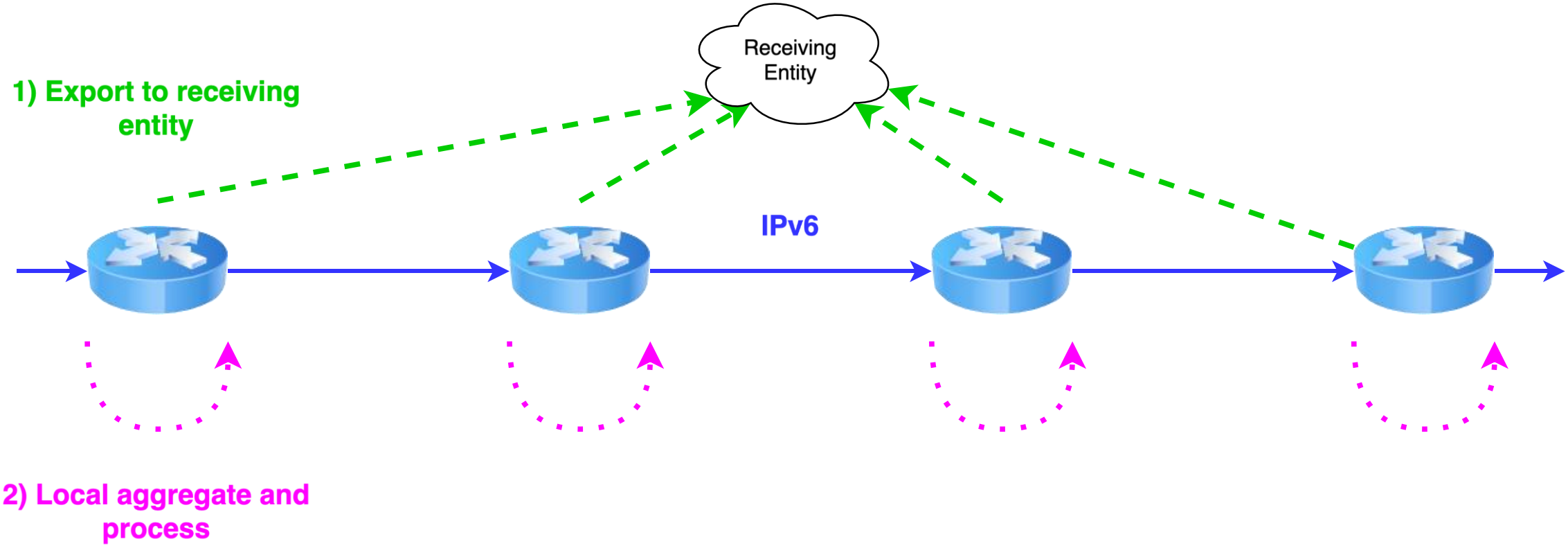
IOAM – Data fields (RFC 9197)

Telemetry information that can be gathered

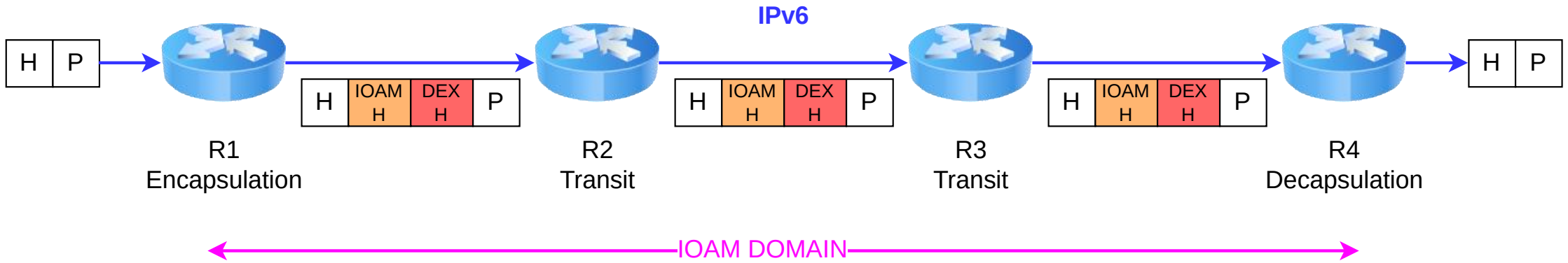
- Hop limit
- Node ID
- Interfaces IDs
- Timestamp
- Queue depth
- Opaque State Snapshot
- …

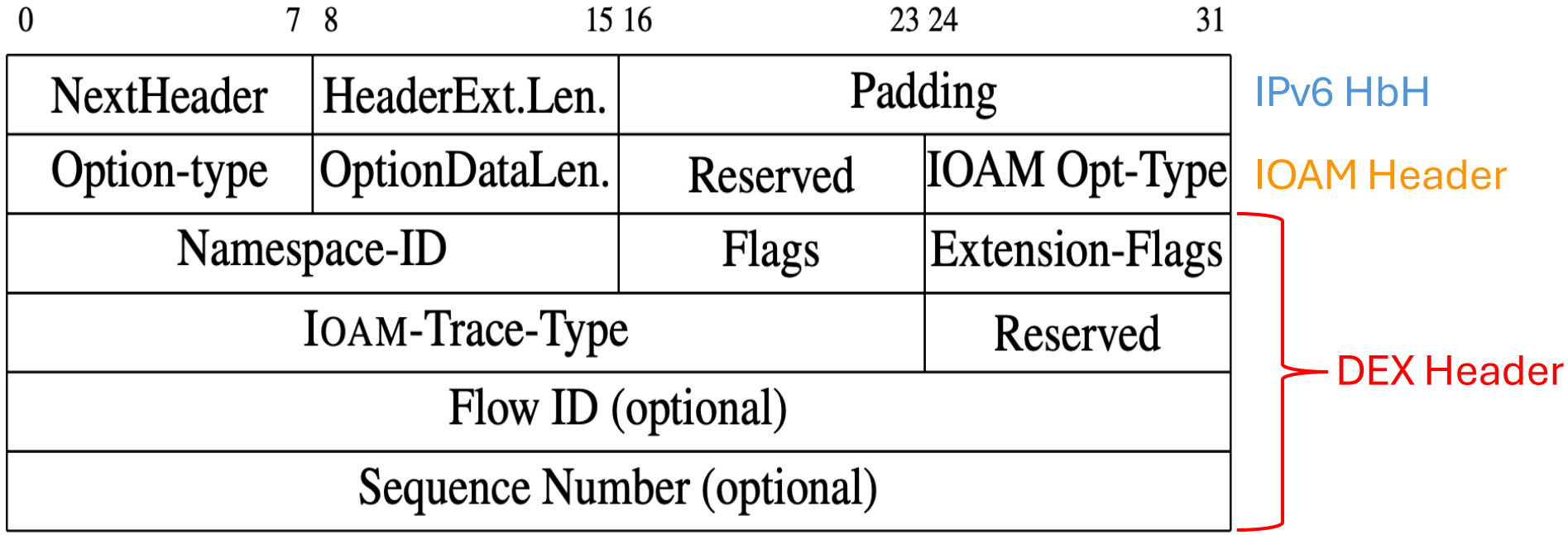
Table of Contents

IOAM Direct Exporting (DEX) – RFC 9326



IOAM Direct Exporting (DEX) – RFC 9326





IOAM DEX – Generic Approach

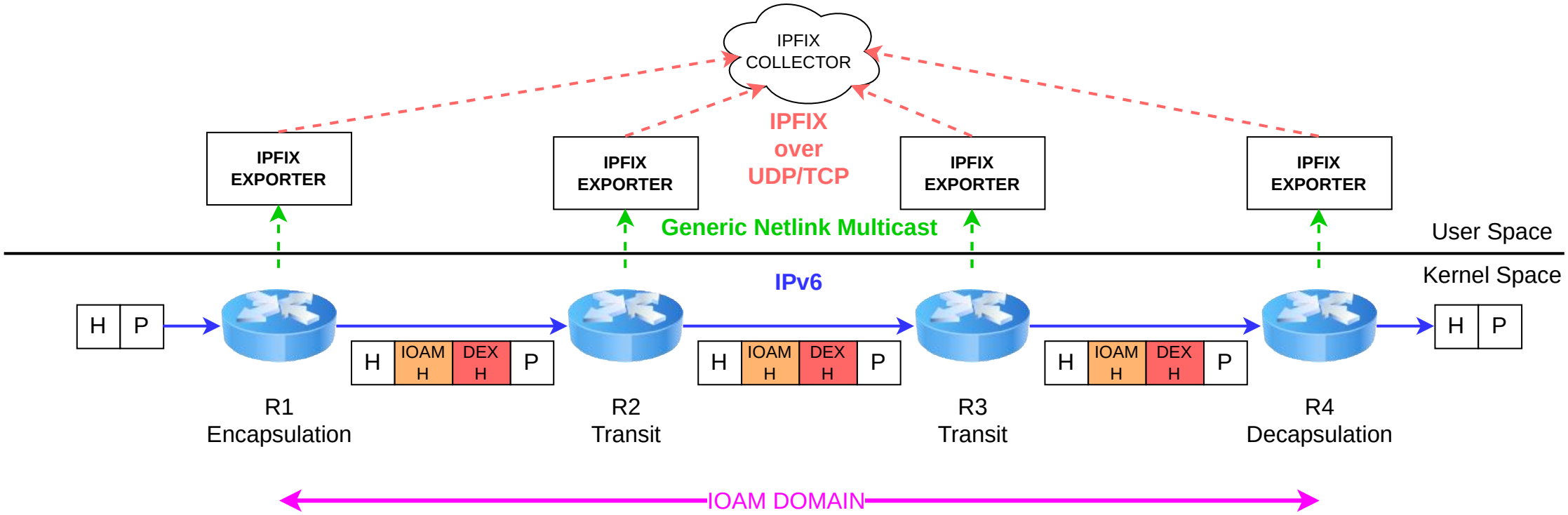


Table of Contents

IOAM in the Linux Kernel

IOAM Pre-allocated Trace Option-type (PTO)

- Initial version at NetDev 0x14
- Mainline since 5.15
- Update in 5.16 for in-transit packets (IPv6-in-IPv6 tunnel)
- Update in 6.9 for events with generic netlink

IOAM Direct Exporting (DEX)

- Extending current implementation for PTO
- Fully backward compatible

Implementation – Encapsulation (1/2)

Relies on LightWeight Tunnel (LWT)

```
struct ioam6_lwt {
    struct dst_cache cache;
    struct ioam6_lwt_freq freq;
    atomic_t pkt_cnt;
    u8 mode;
    bool has_tunsrc;
    struct in6_addr tunsrc;
    struct in6_addr tundst;
    struct ioam6_lwt_encap tuninfo;
    union {
        struct ioam6_lwt_encap tuninfo;
        struct ioam6_lwt_dex dexinfo;
    };
};
```

```
struct ioam6_lwt_dex {
        struct ipv6_hopopt_hdr eh;
        u8 pad[2]; /* 4n alignment*/
        struct ioam6_hdr ioamh;
        struct ioam6_dex_hdr dexh;
};
```

- Ensure backward compatibility
- Minimize memory footprint

Implementation – Encapsulation (2/2)

Generating and storing flow IDs and sequence numbers

```
static __be32 ioam6_dex_flowid(struct sk_buff *skb)
{
    u32 hash = skb_get_hash(skb);
    hash = rol32(hash, 16);
    return (__force __be32)hash;
}
```

Inspired by flow label in SRv6.

Other solution?

```
struct ioam6_pernet_data {
    struct mutex lock;
    struct rhashtable namesapces;
    struct rhashtable schemas;
    struct rhashtable dex_flows;
};
```

Implementation – Reporting IOAM data

Generic Netlink Multicast Group “ioam6_events”

Added Attributes

- Option-Type (PTO or DEX)
- DEX Namespace
- DEX Flow ID
- DEX Sequence Number
- One attribute **per** IOAM data field

Benefits

- Only report what is necessary
- Decision left to the users on the processing

Implementation – Transit

1. Verify if IOAM enabled on input interface
2. Check if namespace in header matches one of the namespaces on node
3. Get IOAM data pertaining to node
4. Trigger generic netlink multicast event

Implementation – Decapsulation

Check and report IOAM data as transit node

Depends on encapsulation mode

Inline

- Reached destination
- Usual kernel code path

Encapsulation

- Reached end of IOAM domain
- Remove both headers
- Forward packet

User Space Support

Wireshark

- Mainline repository
- Not a new version released

iproute

- rtnetlink
- ioam6_iptunnel.h uapi

IPFIX Exporter

- IOAM to IPFIX
- Golang application

```
$> ip -6 route add {} encap ioam6 [freq {}/{}] [mode
inline | encap| auto] [tundst {}] dex ns {} trace-type {}
ext-flags {} via {}
```

Usage

1) Configure IOAM namespace on each node

```
$> ip ioam namespace add ID [data DATA32] [data DATA64]
```

2) Enable IOAM on interface

```
$> sysctl –w net.ipv6.conf.{iface}.ioam6_enabled=1
```

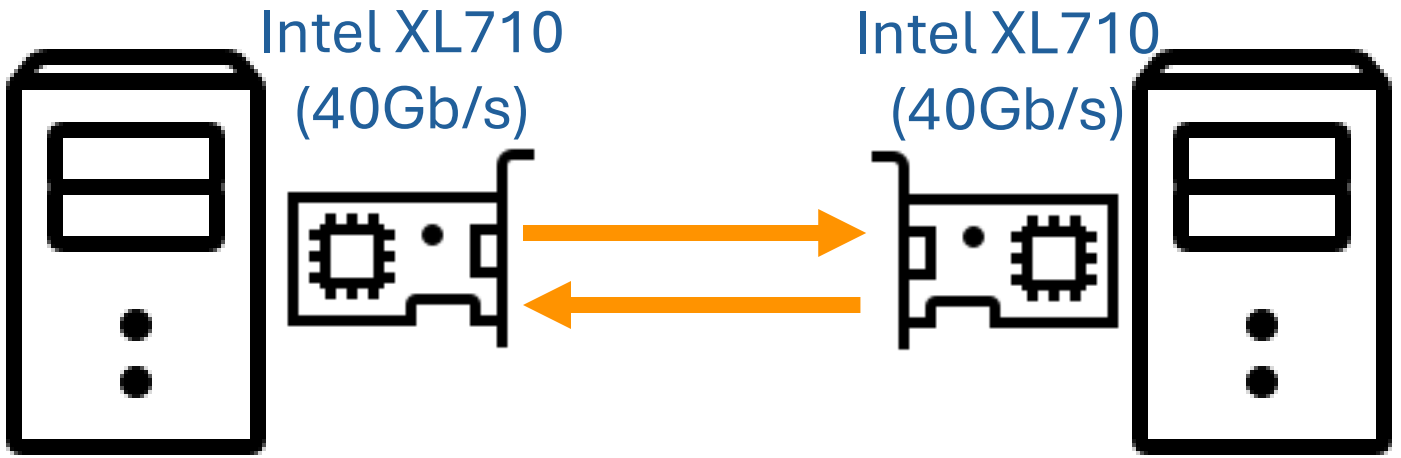
3) Configure node ID and interfaces' IDs

```
$> sysctl –w net.ipv6.ioam6_id=ID
$> sysctl -w net.ipv6.conf.{iface}.ioam6_id=ID
```

4) Configure route with iproute2

Table of Contents

Performance Measurements



Traffic Generator
Cisco TRex on Linux 5.17

Device Under Test (DUT)
Patched Linux 6.12

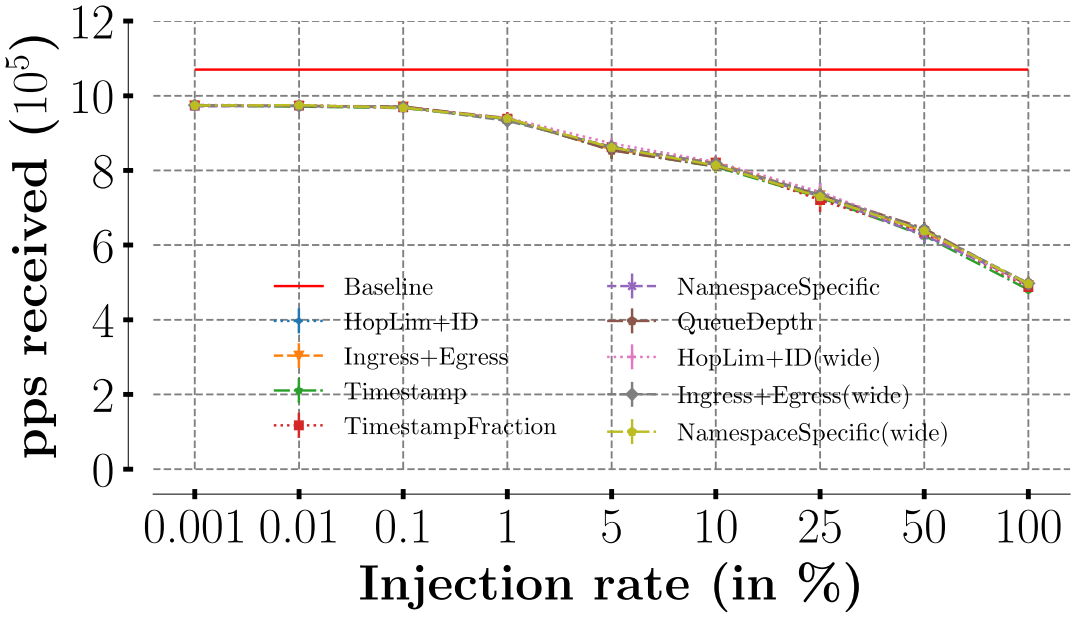
3 Operations

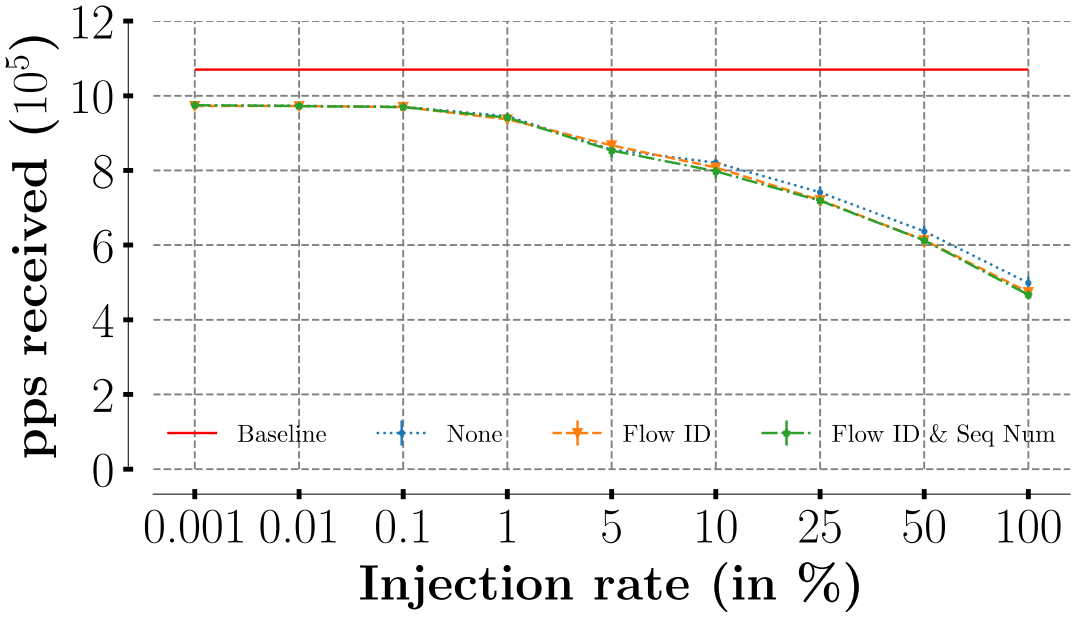
- Encapsulation
- Transit
- Decapsulation

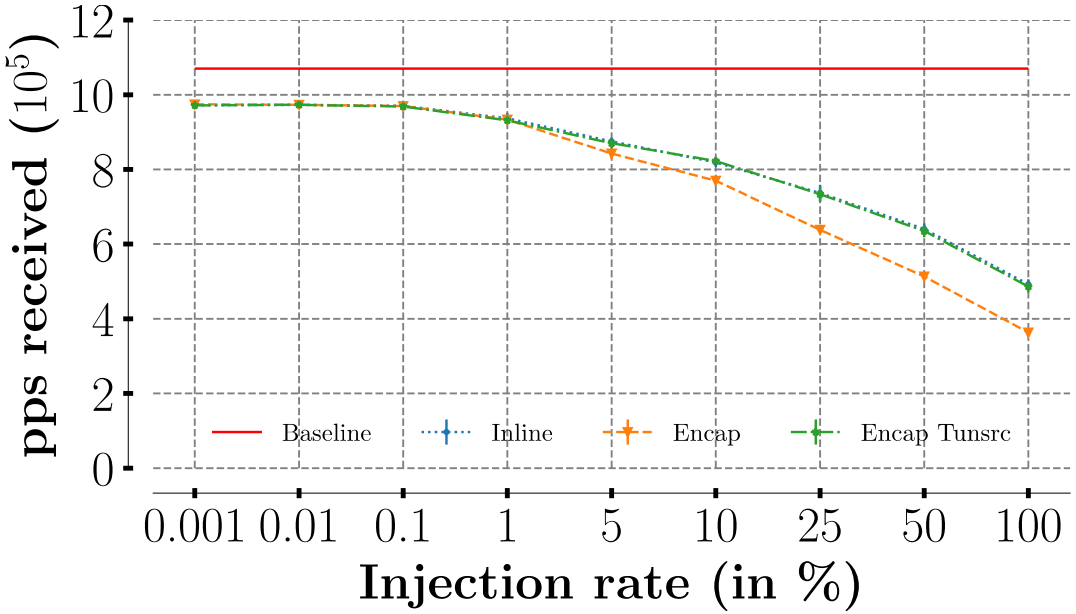
Configuration on DUT

- Scaling governor on performance
- No checksumming
- Receiving on single queue

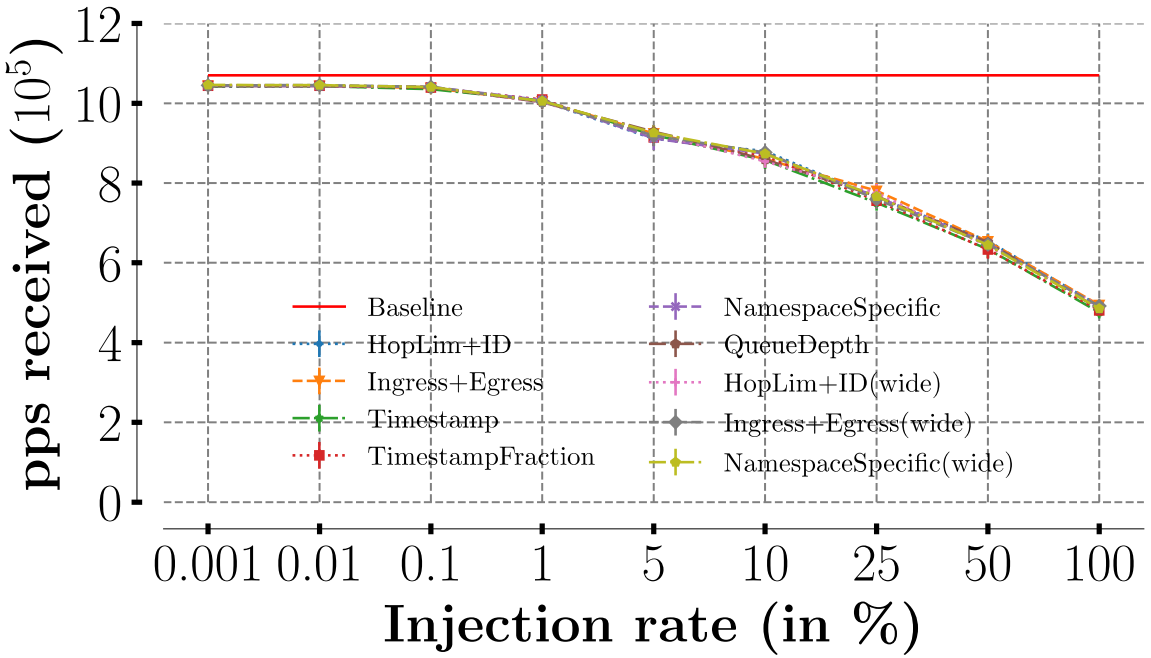
Performance – Encapsulation

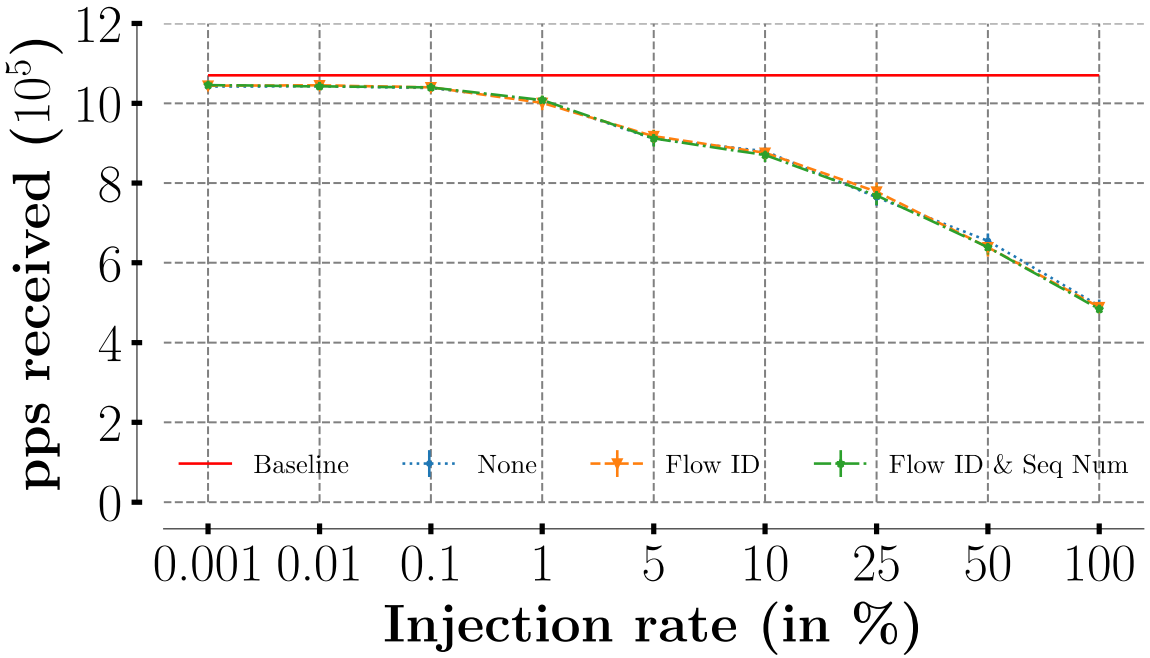






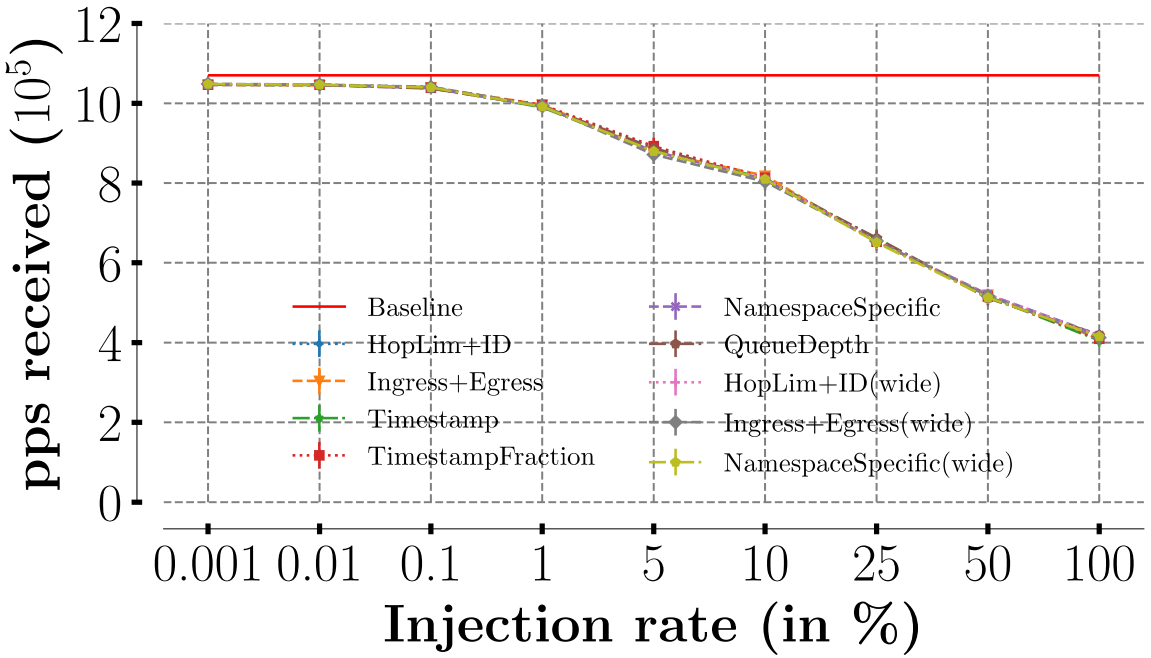
Performance – Transit

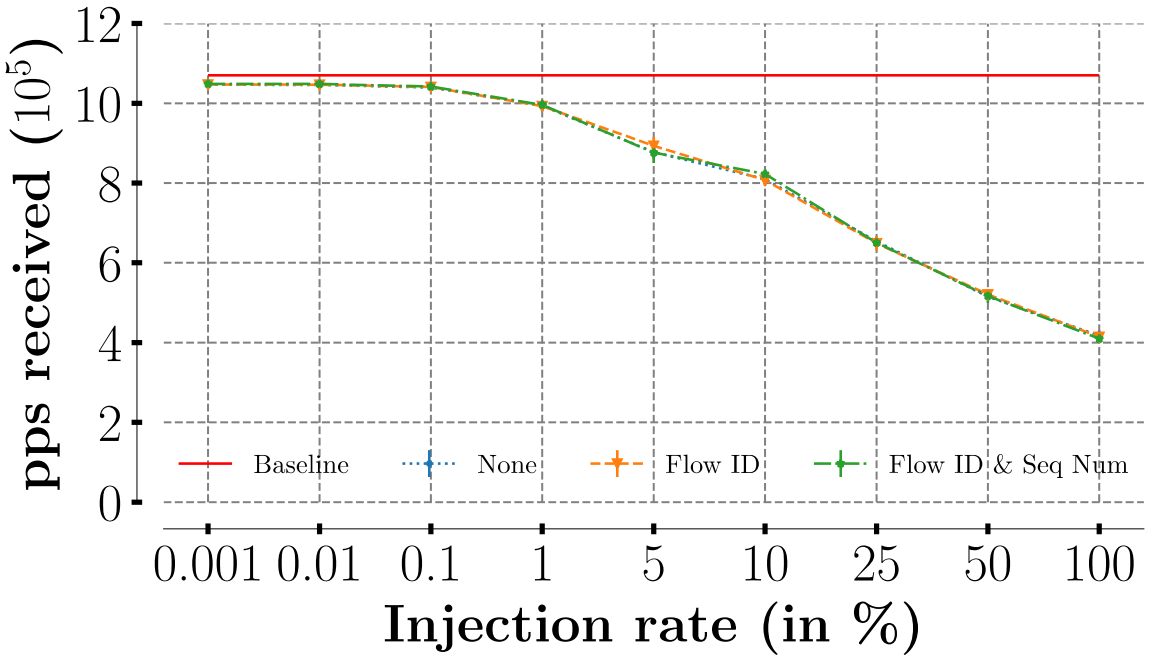




Better performance than encapsulation

Performance – Decapsulation





Worse performance than transit

Performance – Conclusion

- Extension flags have a negligible impact
- Data fields have no impact
- Encapsulation without specifying the source leads to worse performance
- Transit has smaller impact than encapsulation and decapsulation

Thank You For Listening

Do you have any question/feedback?

Code and experiments

maxime.goffart@uliege.be