ERSPAN in Linux

A short history and review.

Presenters: William Tu and Greg Rose

What is Port Mirroring?

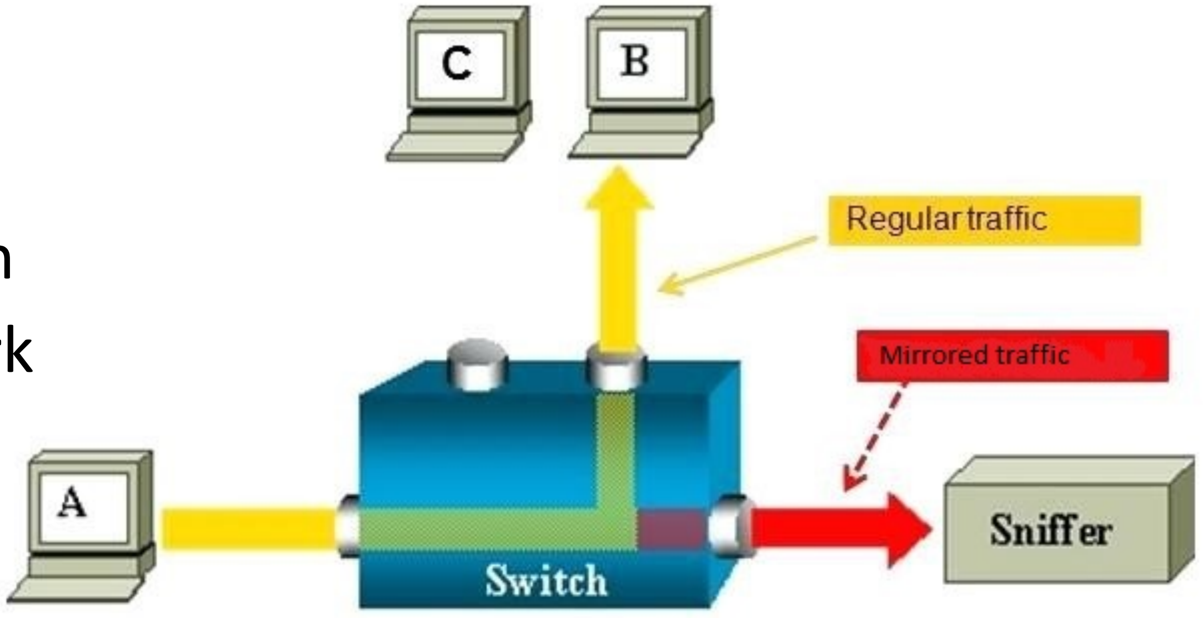
- Port mirroring is one of the most common network troubleshooting techniques.
- SPAN - Switch Port Analyzer - sends a copy of the monitored traffic to a local device.
- RSPAN – Remote Switch Port Analyzer – sends a copy of the monitored traffic to a remote device via VLAN tagging.
- ERSPAN - Encapsulated Remote SPAN – uses GRE encapsulation to extend the basic port mirroring capability from Layer 2 to Layer 3 which allows the mirrored traffic to be sent through a routable IP network.

Port Mirroring Examples

- Host A sends traffic to Host B
- A copy of the traffic is forwarded to sniffer

Three ways:

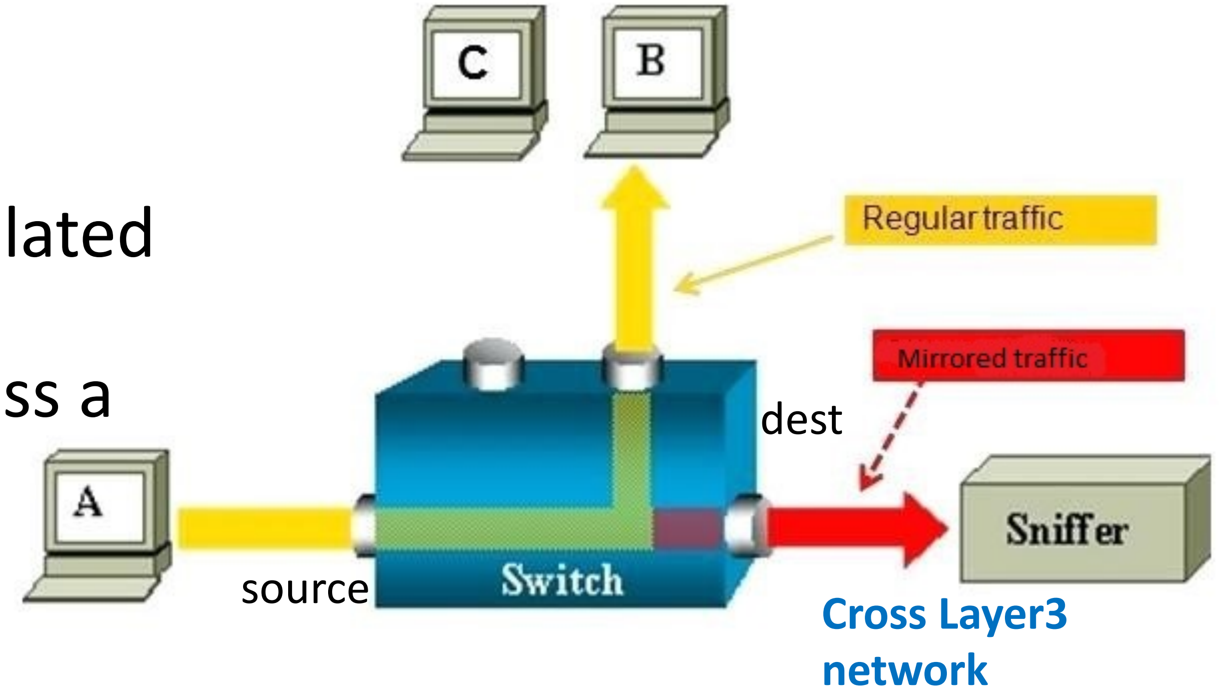
- SPAN: sniffer is at the same switch
- RSPAN: sniffer is at different switch
- ERSPAN: sniffer is across IP network



- Use cases:
  - Analyze, diagnose, detect malicious traffic

Encapsulated Remote Switched Port ANalyzer

- Mirrors traffic on source port(s) and delivers the mirrored traffic to destination port(s) on another switch
- Traffic (inner packet) is encapsulated in GRE (Generic Routing Encapsulation) so routable across a layer 3 network



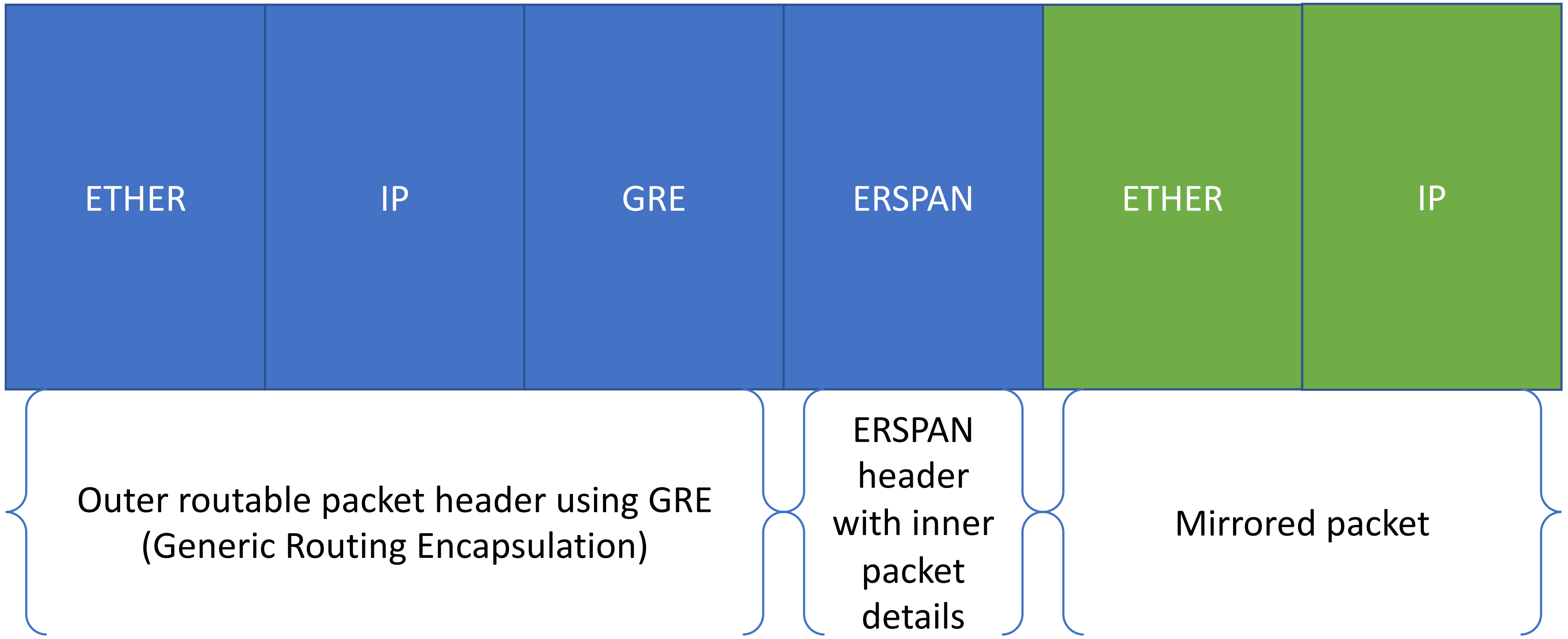
A Short History of ERSPAN in Linux

- Adding ERSPAN to Linux became possible when Cisco released the specification in 2014.
- ERSPAN for IPv4 was added into Linux kernel in 4.14, and for IPv6 in 4.16.
- Includes both transmission and reception and is based on the existing ip_gre and ip6_gre kernel modules.
- Allows Linux to act as an ERSPAN traffic source sending the ERSPAN mirrored traffic to the remote host, or an ERSPAN destination which receives and parses the ERSPAN packets generated from Cisco or other ERSPAN-capable switches.
- Provides native tunnel support and metadata mode tunnel support.

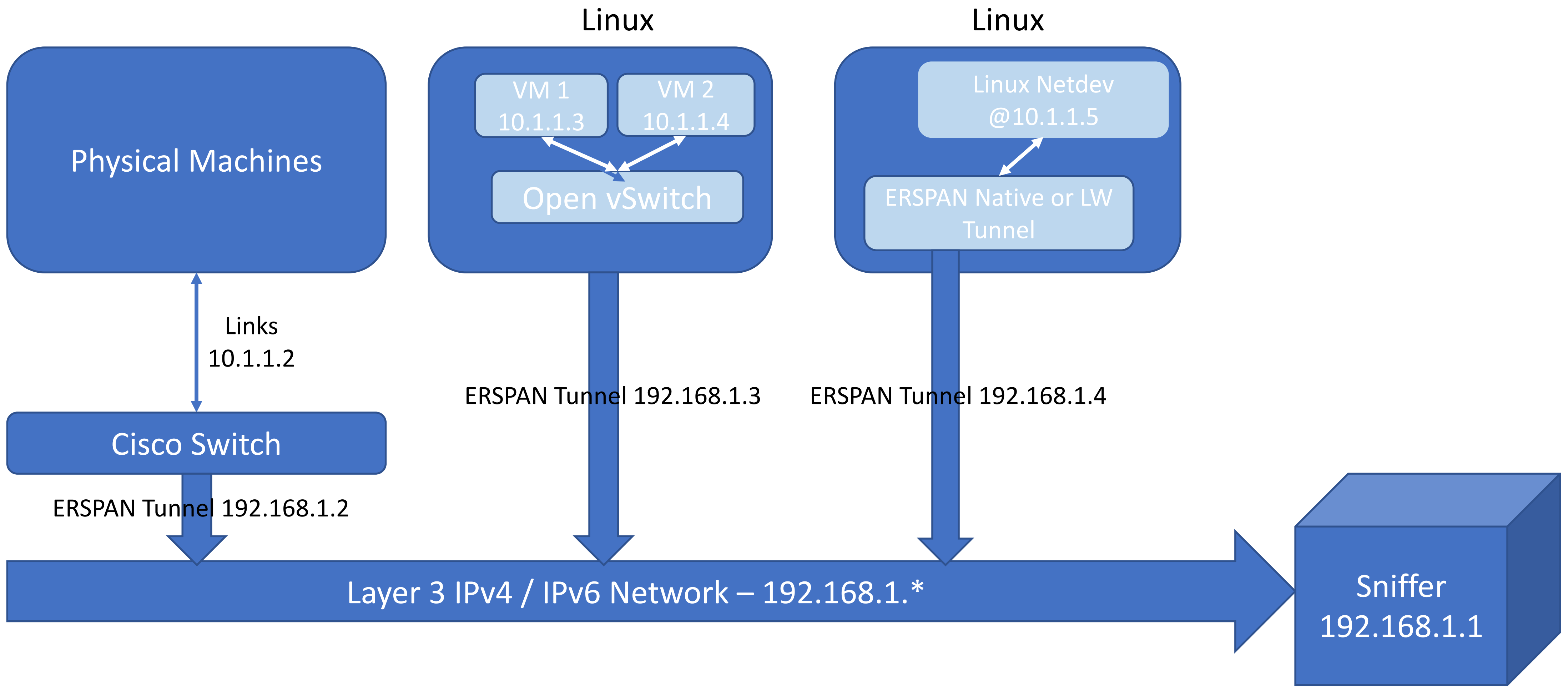
Common ERSPAN Use Cases

- Debugging network issues by tracking the control and data frames.
- Monitoring Voice-over-IP, VoIP, packets for delay and jitter analysis
- Monitoring network transactions for latency analysis
- Monitoring network traffic for anomaly detection

ERSPAN Packet Example


ETHER
IP
GRE
ERSPAN
ETHER
IP
Outer routable packet header using GRE (Generic Routing Encapsulation)
ERSPAN header with inner packet details
Mirrored packet

ERSPAN Tunnel Setup Examples


Linux
Linux
Physical Machines
VM 1 10.1.1.3
VM 2 10.1.1.4
Open vSwitch
Linux Netdev @10.1.1.5
ERSPAN Native or LW Tunnel
Links 10.1.1.2
Cisco Switch
ERSPAN Tunnel 192.168.1.3
ERSPAN Tunnel 192.168.1.4
ERSPAN Tunnel 192.168.1.2
Layer 3 IPv4 / IPv6 Network – 192.168.1.*
Sniffer 192.168.1.1

Linux native tunnel vs metadata mode tunnel

- There are two tunnel type implementations in Linux kernel: native tunnel and metadata-mode (light weight) tunnel.
  - https://lwn.net/Articles/651497/ See article by Thomas Graf, August 2016
- Native tunnel - created with per tunnel-specific configuration, tied together with the net device.
  - creating a GRE tunnel with key and sequence number can be done by:

```
# ip link add dev gre123 type gretap local 1.1.1.1 remote 2.2.2.2 seq key 0xfb
```

- As a result, N different tunnel configurations require creating N number of netdevs.
- Does not scale well.

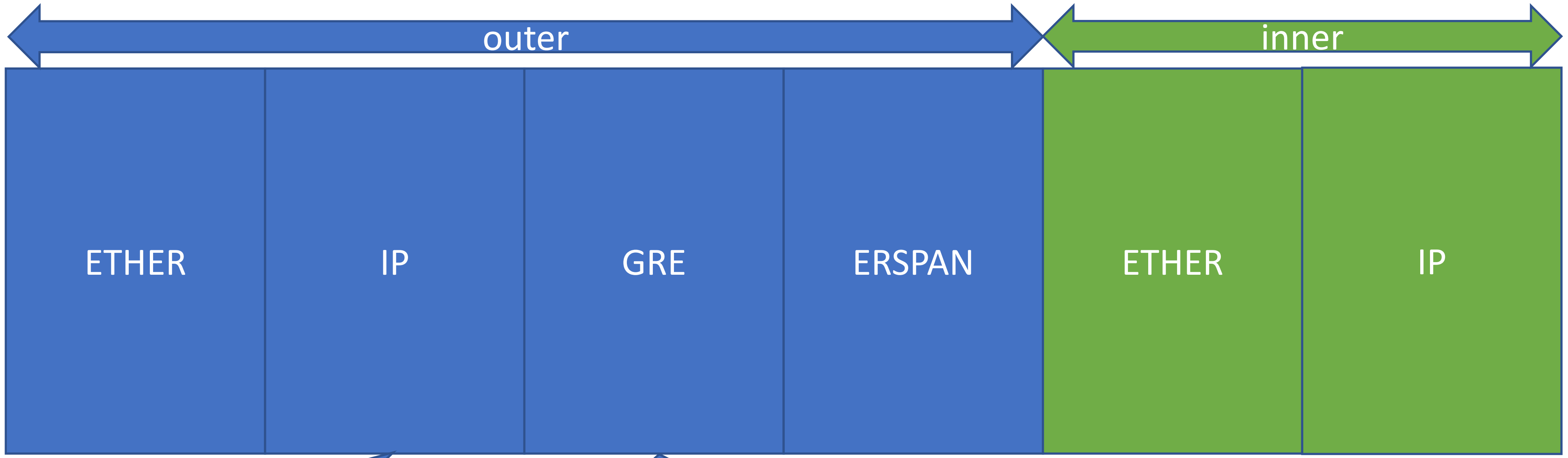
Native vs. metadata mode tunnel (cont…)

- Metadata mode tunnel (aka lightweight tunnel) resolves scaling issue.
- Only one netdev per tunnel type is required to represent multiple tunnels.
- This means that the tunnel configuration of a particular type of the tunnel must be passed to the tunnel netdev in order to encapsulate the packet.
- OVS uses lightweight tunnels.
- See example of a LWT with eBPF:

https://elixir.bootlin.com/linux/latest/source/tools/testing/selftests/bpf/test_tunnel.sh

https://elixir.bootlin.com/linux/latest/source/tools/testing/selftests/bpf/test_tunnel_kern.c

ERSPAN Protocol Headers



The use of the IP protocol as part of the outer header is important because it makes the mirrored traffic routable across any IP network.

Fixed 8 byte header with Seq #
Next Protocol 0x88be – ERSPAN Type II
Next Protocol 0x22be – ERSPAN Type III

GRE Header

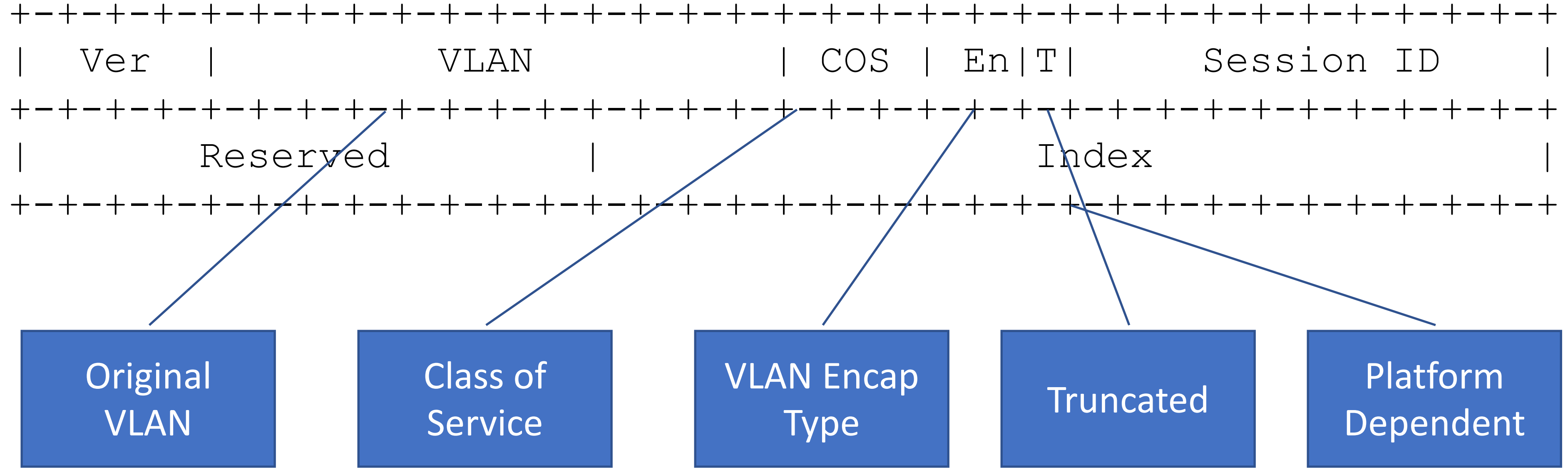
```
GRE header for ERSPAN encapsulation (8 octets) -- 8 bytes
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|0|0|1|0|  00000  |   000000   |   Protocol Type for ERSPAN    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Sequence Number (increments per packet per session)      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Sequence # useful at sniffer to determine out of order packets.

Next Protocol 0x88be – ERSPAN Type II
Next Protocol 0x22be – ERSPAN Type III

ERSPAN Header – Version 1 (Type II)

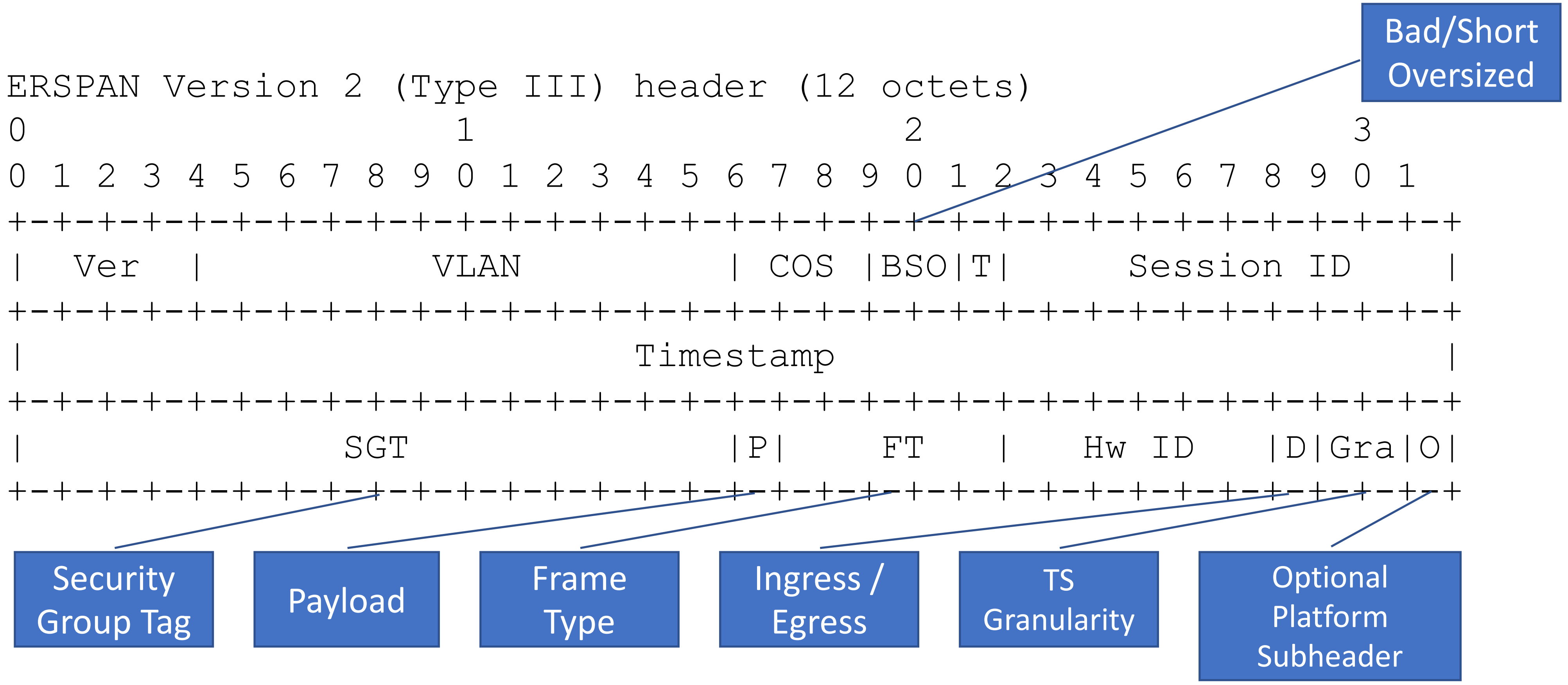
```
ERSPAN Version 1 (Type II) header (8 octets)
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Ver  |          VLAN         | COS | En|T|      Session ID    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reserved         |                  Index                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



ERSPAN Version 1 (Type II) Implementation

- Introduces two new iproute2 configurable fields to the netlink API.
  - Session ID
  - Index
- ERSPAN does not use GRE KEY so repurposes IFLA_GRE_IKEY, IFLA_GRE_OKEY for the Session ID.
- Index is also configurable via iproute2.
- COS and VLAN are extracted from original frame.
- Truncate bit is set if:
  - Skb length is greater than device MTU + device hard_header_len
  - IPv4 length is greater than skb length – network header offset
  - IPv6 length is greater than skb length – transport header offset

ERSPAN Header – Version 2 (Type III)



```
ERSPAN Version 2 (Type III) header (12 octets)
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Ver  |          VLAN         | COS |BSO|T|     Session ID    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Timestamp                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             SGT               |P|    FT   |   Hw ID   |D|Gra|O|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

ERSPAN Version 2 (Type III) Implementation

- Introduces another two fields to kernel through netlink API.
  - Hardware ID
  - Direction - ingress or egress
- COS, BSO, and T fields can be extracted or inferred from the mirrored frame.
- Timestamp value is calculated by calling the kernel ktime_get_real() with 100 µs granularity – Only 100 µs is supported.
- SGT is hard coded to zero.
- Non-Ethernet mirrored packet is not supported, so FT is always 0 and P is set to 1.
- There is no implementation of sub-headers, so optional bit is zero.

Cisco ERSPAN example

- We use Nexus 5000 switch and configure its ERSPAN tunnel on ports 11 and 12 as below

```
monitor session 10 type erspan-source
erspan-id 10
vrf default
destination ip 192.168.1.1
source interface Ethernet1/11 both
source interface Ethernet1/12 both
no shut
monitor erspan origin ip-address 192.168.1.2 global
```

With openvswitch.ko

```
# with 4.19-rc6+ kernel and iproute2-ss180813
# creating datapath named "mydp", attach veth1(port 1)
$ ovs-dpctl add-dp mydp
$ ovs-dpctl add-if mydp ovs-veth1 // connected to namespace ns0 peer veth1

# creating erspan dev named "myerspan" and attach
# Note that OVS uses a lightweight tunnel with “external” keyword
$ ip link add dev myerspan type erspan external
$ ovs-dpctl add-if mydp myerspan

# flow entry for port 1 to erspan tunnel
$ ovs-dpctl add-flow mydp \
"in_port(1),eth(src=00:01:02:03:04:05,dst=10:11:12:13:14:15),eth_type(0x0800),\
ipv4(src=35.8.2.41,dst=172.16.0.20,proto=5,tos=0x80,ttl=128,frag=no)" \
"set(tunnel(tun_id=20,dst=192.168.1.1,ttl=64,erspan(ver=2,dir=1,hwid=0x4),flags(df|key
))),2
```

Note that the OVS vswitchd daemon is not required for this case.

Linux Native Mode tunnel example

```
# with 4.19-rc6+ kernel and iproute2-ss180813
# Native-mode without using eBPF
$ ip link add dev myerspan type erspan seq \
key 30 local 192.168.1.4 remote 192.168.1.1 \
erspan_ver 1 erspan 123 dev ens3
$ tc qdisc add dev ens3 handle ffff: ingress
$ tc filter add dev ens3 parent ffff: \
matchall skip_hw action mirred egress \
mirror dev myerspan
```

Linux LWT ERSPAN eBPF example

```
# with 4.19-rc6+ kernel and iproute2-ss180813
$ ip link add dev myerspan type erspan external
$ tc qdisc add dev myerspan handle ffff: ingress
$ tc qdisc add dev ens3 handle ffff: ingress
$ tc filter add dev myerspan bpf da obj \
./test_tunnel_kern.o section erspan_set_tunnel
$ tc filter add dev ens3 parent ffff: matchall\
skip_hw action mirred egress mirror dev myerspan
```

See e.g.

https://elixir.bootlin.com/linux/latest/source/tools/testing/selftests/bpf/test_tunnel.sh

Conclusion

- Port mirroring is one of the most common troubleshooting techniques.
- ERSPAN extends its predecessors SPAN and RSPAN, by allowing the monitored traffic to route across IP networks.
- There are three primary ways to use ERSPAN in Linux:
  - Linux openvswitch kernel module with ovs-dpctl datapath tool.
  - Linux native mode ERSPAN tunnel.
  - Linux eBPF byte-code with metadata mode (Light Weight) tunnel.

Additional Resources

- Cisco's ERSPAN IETF Draft
  - https://tools.ietf.org/html/draft-foschiano-erspan-00
- T Graf. Lightweight flow based encapsulation. Linux kernel.
  - https://lwn.net/Articles/651497/ , August 2016.

Questions?