Challenges of DDoS Detection on Terabit scale

Hello

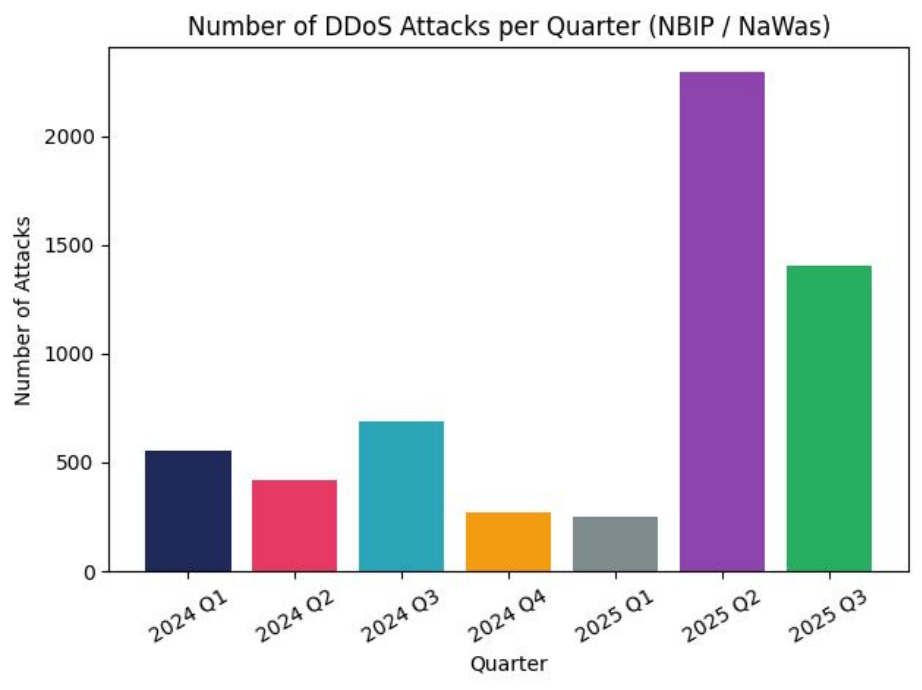
I'm Pavel Odintsov, Founder of FastNetMon LTD, London, UK

Ways to contact me:

- linkedin.com/in/podintsov
- IRC, Libera Chat, pavel_odintsov
- Email: pavel@fastnetmon.com
- Telegram: @pavel_odintsov
- Signal: pavel.50
- Wechat: wxid_voujoky6hll912

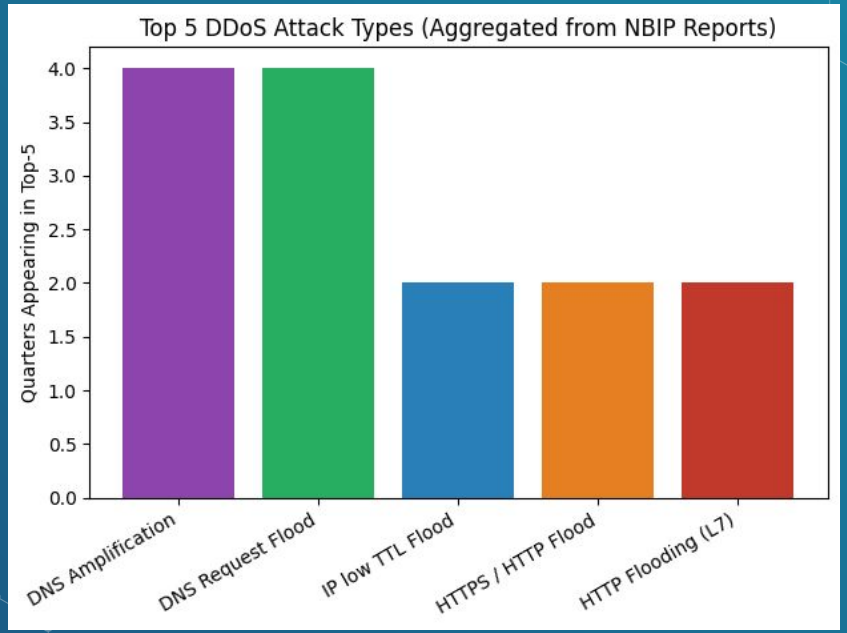
FastNetMon: users

Current DDoS Weather



Data provided by The Dutch National Scrubbing Center (NaWas)

Current DDoS Weather



Data provided by The Dutch National Scrubbing Center (NaWas)

What kind of DDoS? L3. IPv4

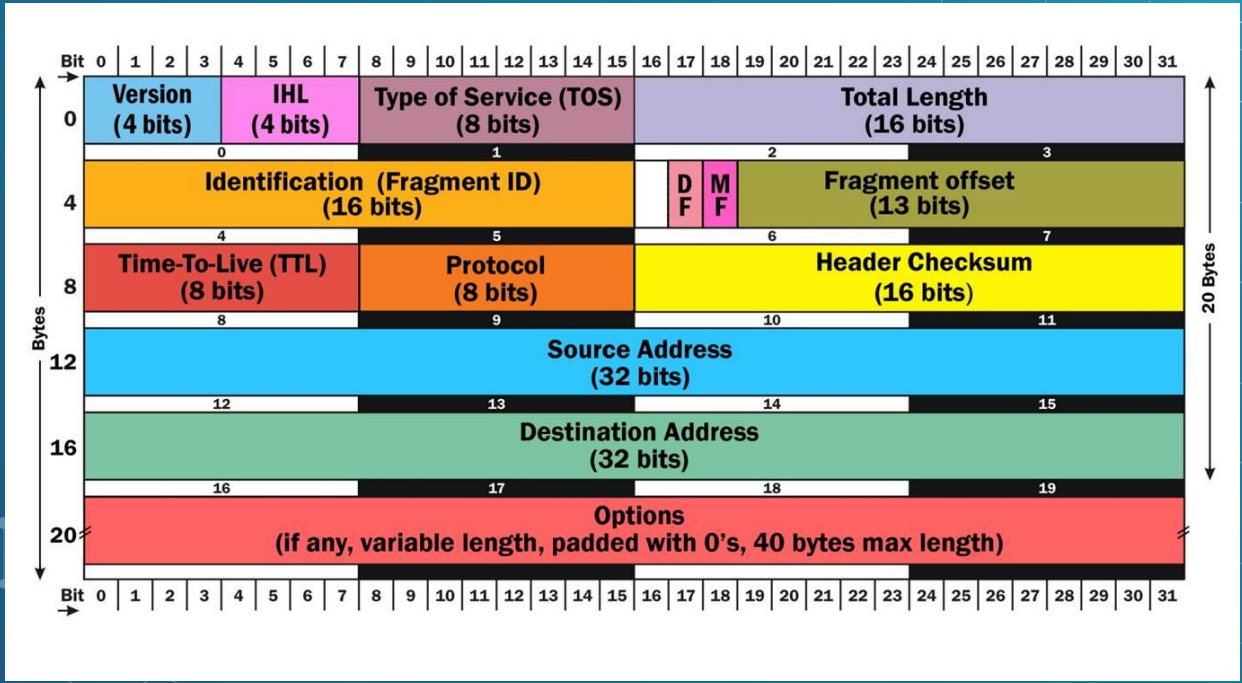


Image from https://thirdinternet.com/ip-packet-syntax/

What kind of DDoS? L3. IPv6

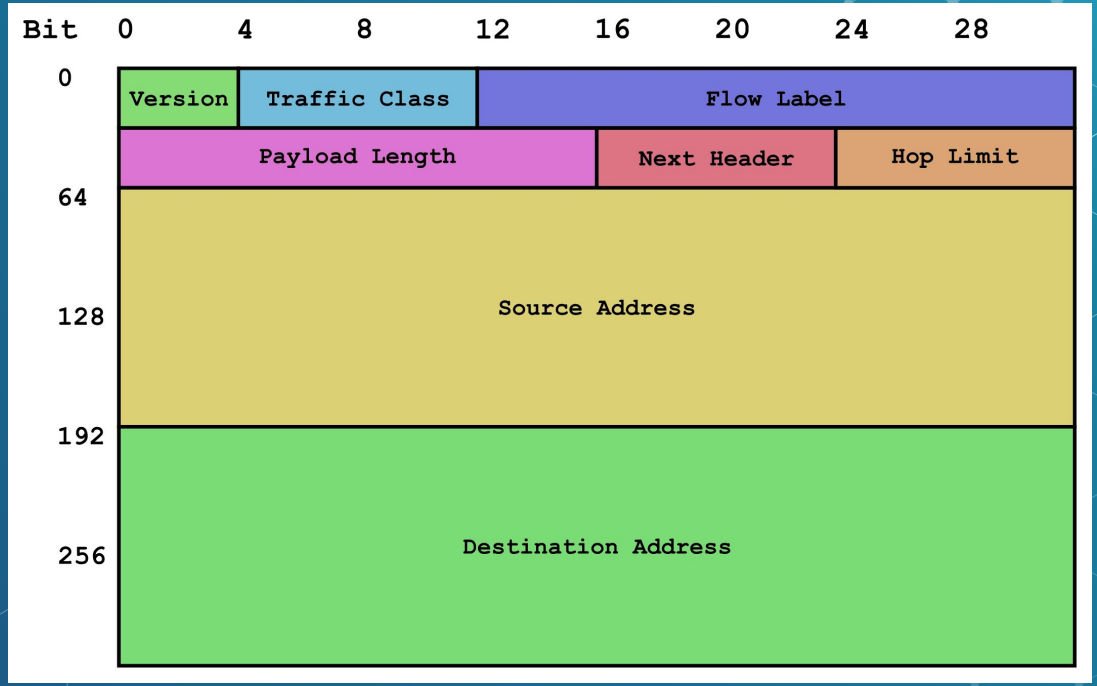


Image from https://networkel.com/ipv6-overview-communication-types/

What kind of DDoS? L4. TCP?

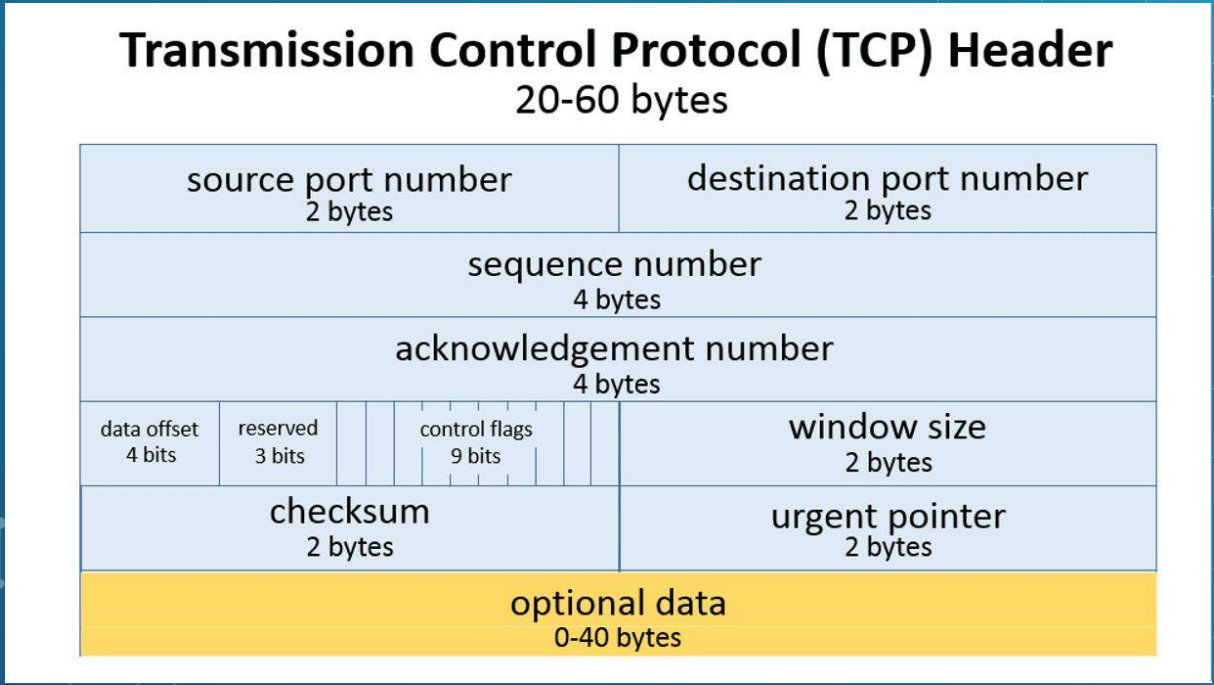


Image from https://www.lifewire.com/tcp-headers-and-udp-headers-explained-817970

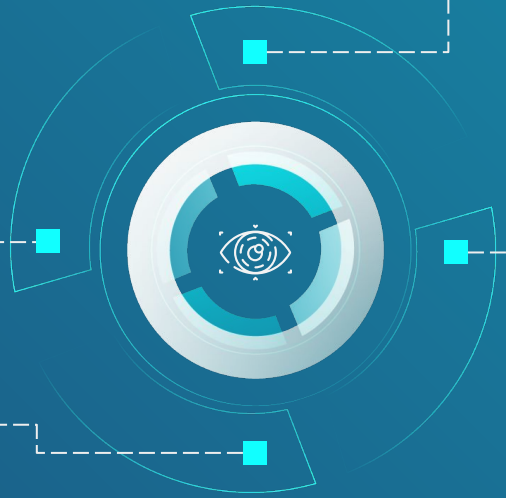
First Terabit scale project

- Located in West Asia
- 750.000 of mobile and fixed broadband customers
- 1T+ of total capacity
- Juniper based

Technical challenges

Inline monitoring services on PTX

Double and triple tagged vlans

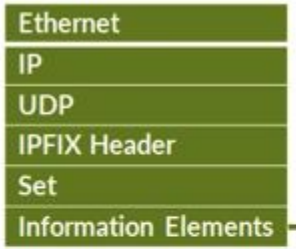


Linux Kernel bugs

Packet rate of UDP telemetry: 50 kpps

Are you up for challenge?

Juniper inline monitoring services



ID	Length	Description	Details
10	4B	ingressInterface	SNMP index of incoming interface
14	4B	egressInterface	SNMP index of outgoing interface when flowDirection=Output, otherwise 0.
61	1B	flowDirection	Direction (0: Input , 1:Output)
312	2B	dataLinkFrameSize	Length of sampled data link frame
315	Variable	dataLinkFrameSelection	N octet from data link frame of monitored packet. Reports actual monitored packet starting from Layer 2 as {Ethernet header/802.1Q header(any)/IP header/Payload ...] up to configured maximum-clip-length

Juniper inline monitoring services: benefits

- Covered in RFC 7011
- 1 second DDoS detection
- Real time graphs
- Access to all information in packet header (fragmentation, ttl, nested vlans)

Juniper inline monitoring services: MX

- MX uses 1 byte encoding to encode length which allows size of header up to 254 bytes

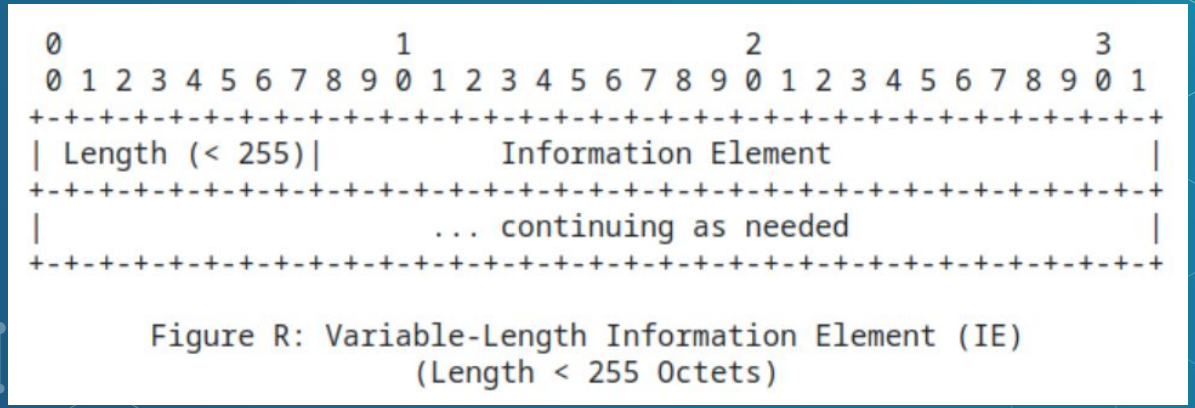


Figure R: Variable-Length Information Element (IE)
(Length < 255 Octets)

Juniper inline monitoring services: PTX

- PTX uses 2 byte encoding which allows size of header up to 65535 bytes

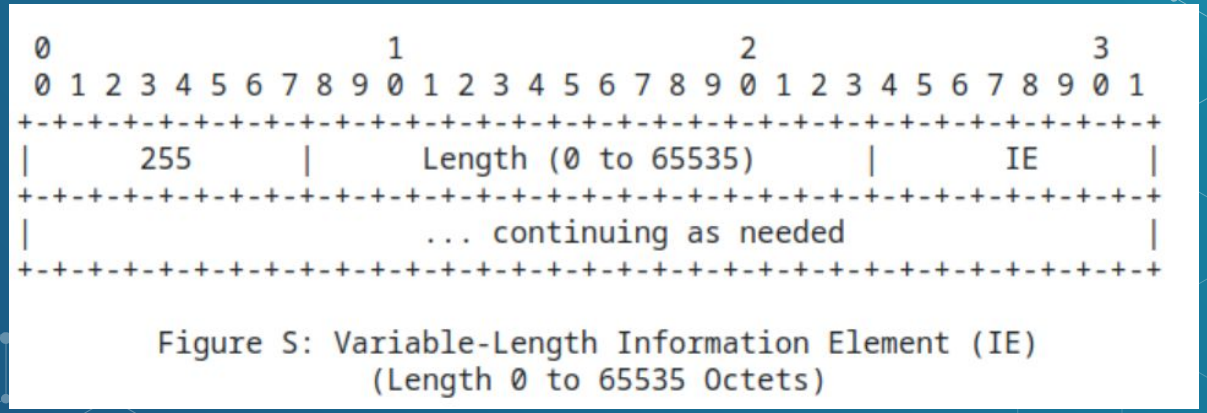


Figure S: Variable-Length Information Element (IE)
(Length 0 to 65535 Octets)

Juniper inline monitoring services: PTX

```
2024-01-29 23:36:58,463 [DEBUG] IPFIX variable field element was used
2024-01-29 23:36:58,463 [ERROR] Not supported two byte variable length encoding detected.
```

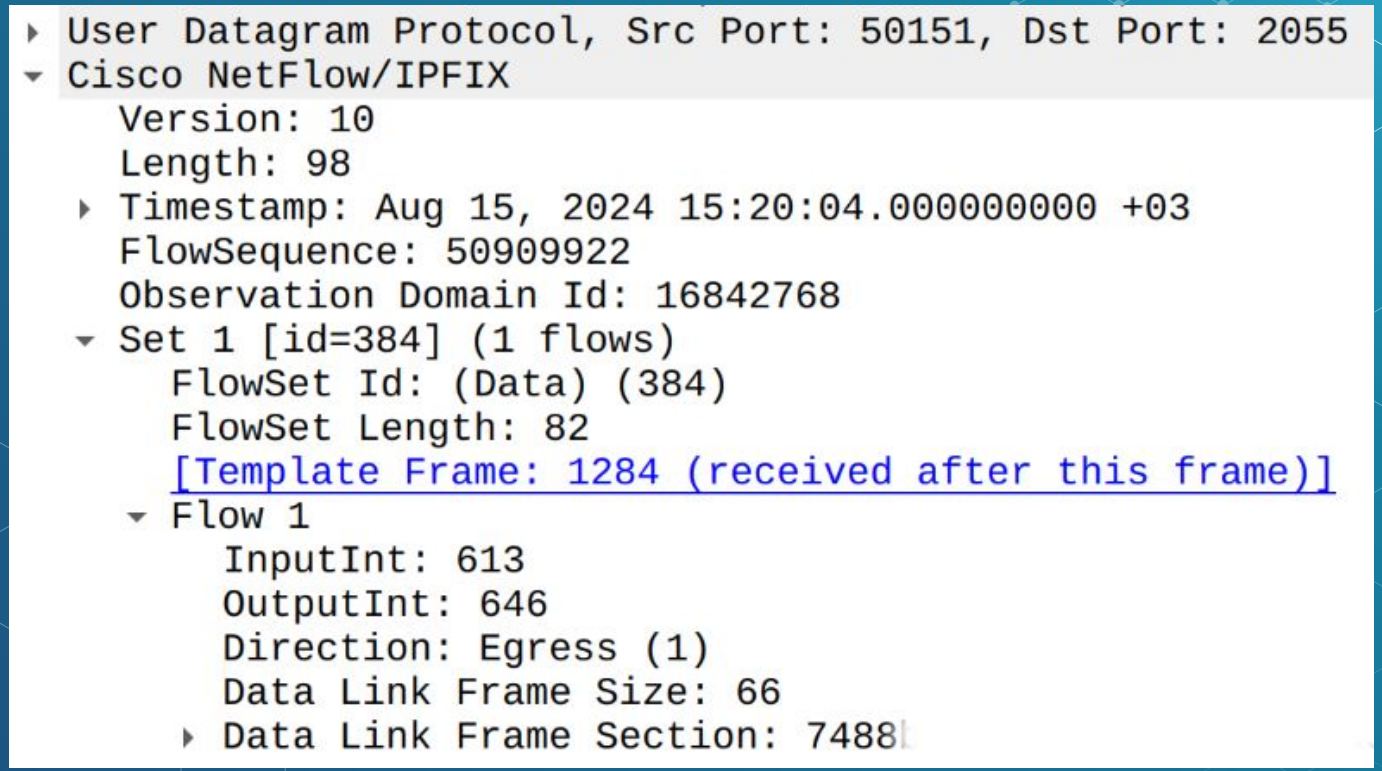
Juniper inline monitoring services: PTX

```
if (*field_length_ptr == 255) {
    // 255 is special and it means that packet length is encoded in two following bytes
    // Juniper PTX routers use this encoding even in case when packet length does not exceed 255 bytes

    // RFC reference https://datatracker.ietf.org/doc/html/rfc7011#page-37
    // In this case, the first octet of the
    // Length field MUST be 255, and the length is carried in the second and
    // third octets, as shown in Figure S.

    // Read 2 byte length by skipping placeholder byte with 255
    const uint16_t* two_byte_field_length_ptr = (const uint16_t*)(pkt + offset + sizeof(uint8_t));

    if (logger.getPriority() == log4cpp::Priority::DEBUG) {
        logger << log4cpp::Priority::DEBUG << "Two byte variable length encoding detected. Retrieved packet length: "
            << fast_ntoh(*two_byte_field_length_ptr);
    }

    // Pass variable payload length
    flow_meta.variable_field_length = fast_ntoh(*two_byte_field_length_ptr);

    // Override field length with length extracted from two bytes + length of placeholder byte itself
    record_length = flow_meta.variable_field_length + sizeof(uint8_t) + sizeof(uint16_t);;

    // Pass variable payload length
    flow_meta.variable_field_length = fast_ntoh(*two_byte_field_length_ptr);

    // Specify length encoding type as it's required for payload retrieval process
    flow_meta.variable_field_length_encoding = variable_length_encoding_t::two_byte;
```

Juniper inline monitoring services: PTX

```
// This packet is ended using IPFIX variable length encoding and it may have two possible ways of length encoding
// https://datatracker.ietf.org/doc/html/rfc7011#section-7
if (flow_meta.variable_field_length_encoding == variable_length_encoding_t::single_byte || flow_meta.variable_field_length_encoding == variable_length_encoding_t::two_byte) {

    if (logger.getPriority() == log4cpp::Priority::DEBUG) {
        logger << log4cpp::Priority::DEBUG << "Packet header length: " << flow_meta.variable_field_length;
    }

    if (flow_meta.variable_field_length != 0) {
        bool read_packet_length_from_ip_header = true;

        bool extract_tunnel_traffic = false;

        const uint8_t* payload_shift = nullptr;

        if (flow_meta.variable_field_length_encoding == variable_length_encoding_t::single_byte) {
            payload_shift = data + sizeof(uint8_t);
        } else if (flow_meta.variable_field_length_encoding == variable_length_encoding_t::two_byte) {
            payload_shift = data + sizeof(uint8_t) + sizeof(uint16_t);
        }

        auto result =
            parse_raw_packet_to_simple_packet_full_ng(payload_shift, flow_meta.variable_field_length,
                                                      flow_meta.variable_field_length, flow_meta.nested_packet,
                                                     extract_tunnel_traffic, read_packet_length_from_ip_header);
```

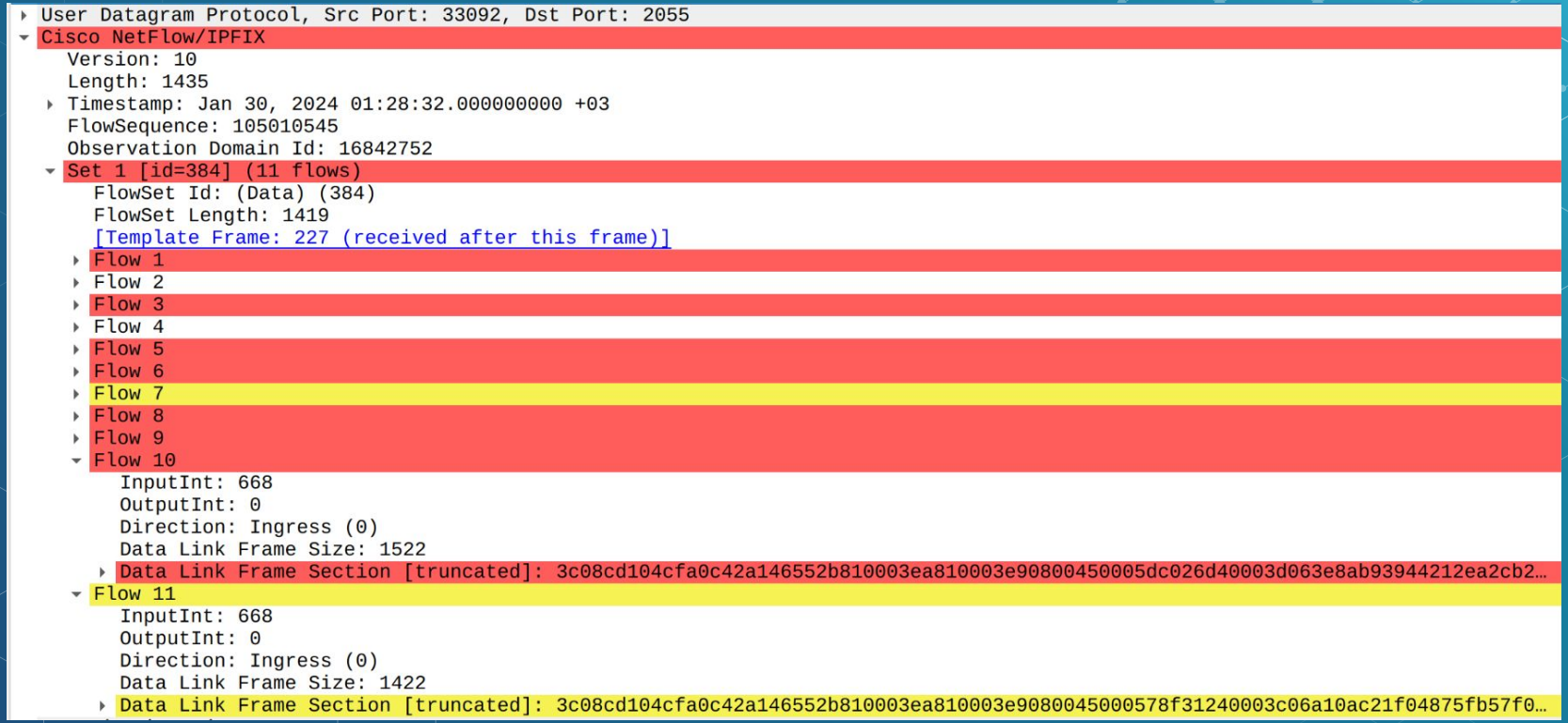
Juniper MX, single flow per UDP packet



Juniper MX, single flow per UDP packet

Protocol	Length	Info
CFLOW	140	IPFIX flow (98 bytes) Obs-Domain-ID=16842768 [Data:384]
CFLOW	200	IPFIX flow (158 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	200	IPFIX flow (158 bytes) Obs-Domain-ID=16842768 [Data:384]
CFLOW	200	IPFIX flow (158 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	200	IPFIX flow (158 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	200	IPFIX flow (158 bytes) Obs-Domain-ID=16842755 [Data:384]
CFLOW	200	IPFIX flow (158 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	200	IPFIX flow (158 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	200	IPFIX flow (158 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	140	IPFIX flow (98 bytes) Obs-Domain-ID=16842768 [Data:384]
CFLOW	200	IPFIX flow (158 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	128	IPFIX flow (86 bytes) Obs-Domain-ID=16842768 [Data:384]

Juniper PTX, multiple flows per UDP packet



Juniper PTX, multiple flows per UDP packet

Protocol	Length	Info
CFLOW	1442	IPFIX flow (1400 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1436	IPFIX flow (1394 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1452	IPFIX flow (1410 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1470	IPFIX flow (1428 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1420	IPFIX flow (1378 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1433	IPFIX flow (1391 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1440	IPFIX flow (1398 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1474	IPFIX flow (1432 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1477	IPFIX flow (1435 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1428	IPFIX flow (1386 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1440	IPFIX flow (1398 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1409	IPFIX flow (1367 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1442	IPFIX flow (1400 bytes) Obs-Domain-ID=16842752 [Data:384]
CFLOW	1393	IPFIX flow (1351 bytes) Obs-Domain-ID=16842752 [Data:384]

We did not expect more then single flow

```
// Get clean flowsets length to use it as limit for our parser
ssize_t current_flowset_length_no_header = flowset_length - sizeof(ipfix_data_flowset_header_t);

if (logger.getPriority() == log4cpp::Priority::DEBUG) {
    logger << log4cpp::Priority::DEBUG << "IPFIX variable field element was used";
}

// TODO: This implementation is rather limited as we read only single flowset here
// In many cases we use this stuff for IPFIX Inline monitoring and it uses just single flowset but it may change in future
```

Complete RFC implementation

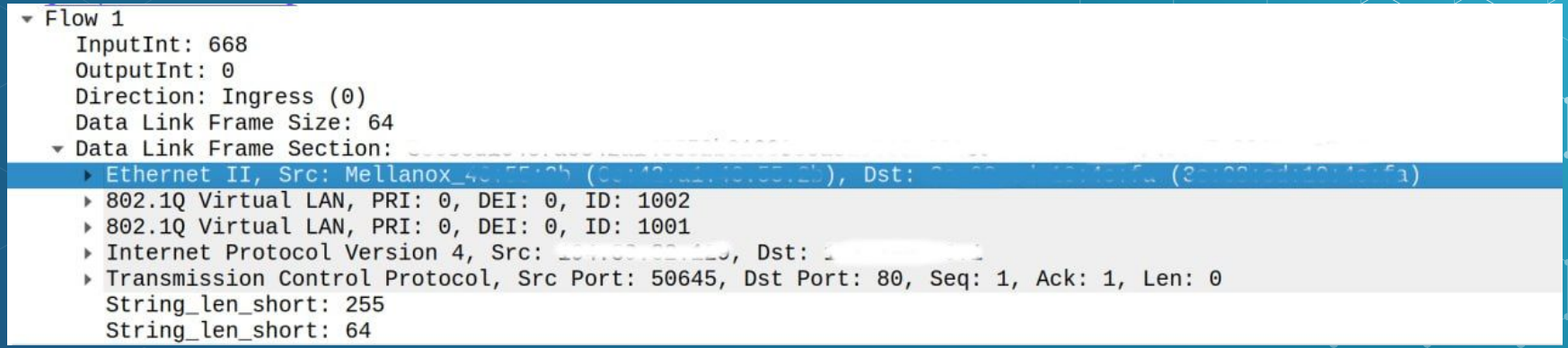
FastNetMon Advanced 2.0.359

fastnetmon-release-publisher released this Feb 9 v2.0.359 93aeed8

Changes:

- Added support for IPFIX IPFIX_UDP_SOURCE_PORT and IPFIX_UDP_DESTINATION_PORT encoding used by AMD Pensando
- Added support for IPFIX IPFIX_TCP_SOURCE_PORT and IPFIX_TCP_DESTINATION_PORT encoding used by AMD Pensando
- Extracted Netflow v5, v9 and IPFIX into separate modules
- Fixed bug with IPFIX sampling rate persistence which prevented us from storing it correctly to disk
- Added logic to remove IPFIX and Netflow v9 sampling between upgrades
- Added support for multiple flows per packet for IPFIX inline monitoring services used by Juniper PTX

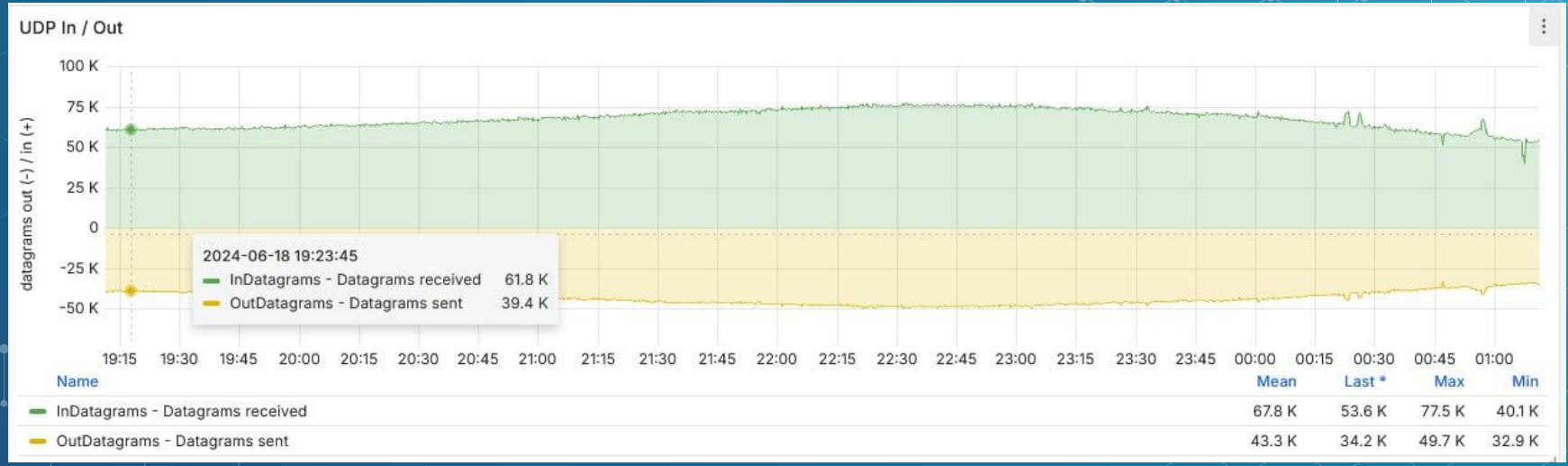
Double tagged vlans



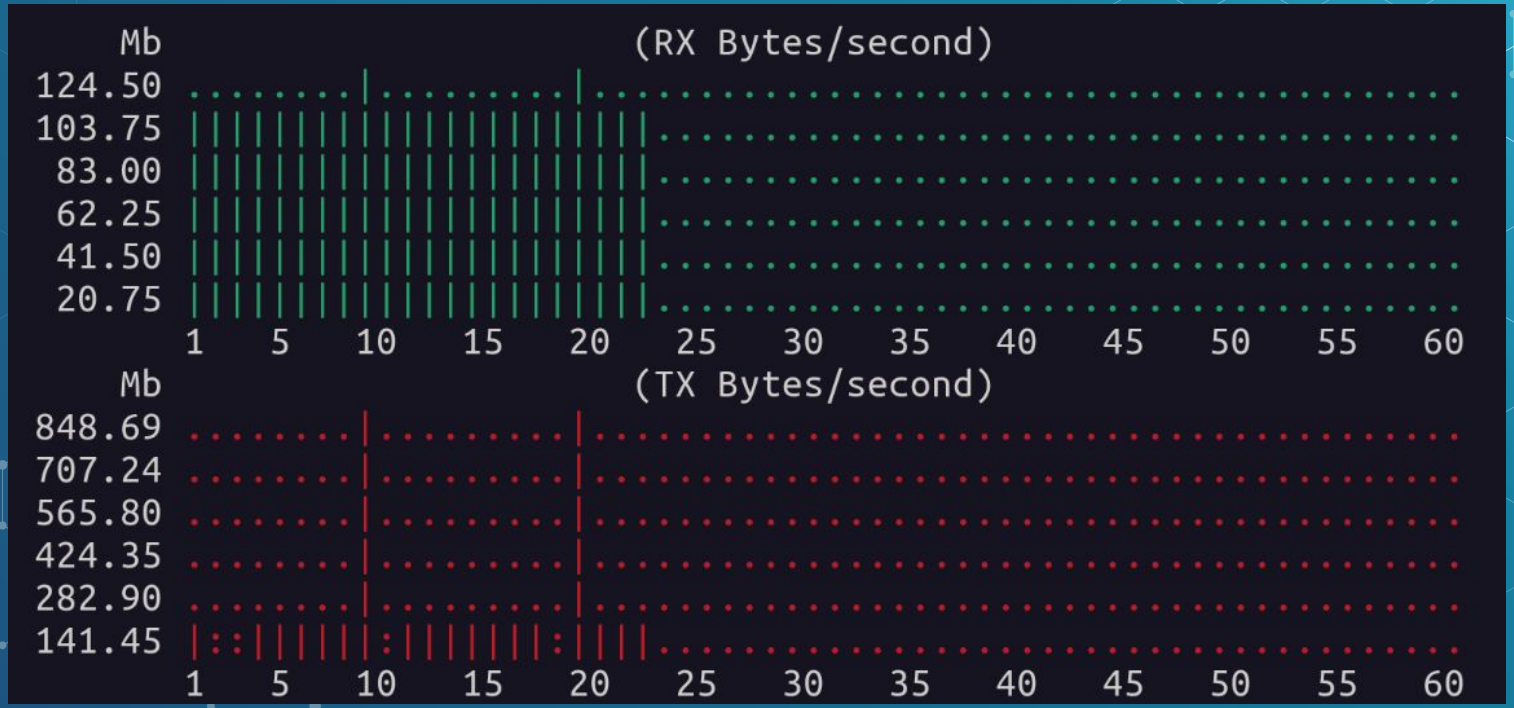
Double tagged vlans: code

```
// Copy ethertype as we may need to change it below
uint16_t ethertype = ethernet_header->get_ethertype_host_byte_order();

uint32_t number_of_stripped_vlans = 0;

// This loop will not start if ethertype is not VLAN
while (ethertype == IanaEthertypeVLAN) {
    // Return error if it's shorter than vlan header
    if (local_pointer + sizeof(ethernet_vlan_header_t) > end_pointer) {
        return parser_code_t::memory_violation;
    }

    const ethernet_vlan_header_t* ethernet_vlan_header = (const ethernet_vlan_header_t*)local_pointer;

    // We've agreed that this field keeps only outermost vlan
    if (number_of_stripped_vlans == 0) {
        packet.vlan = ethernet_vlan_header->get_vlan_id_host_byte_order();
    }

    local_pointer += sizeof(ethernet_vlan_header_t);

    number_of_stripped_vlans++;

    // We need to limit it to avoid possibility of attack which uses too many vlans tags to overload our parser
    if (number_of_stripped_vlans > maximum_vlans_to_strip) {
        return parser_code_t::too_many_nested_vlans;
    }

    // Change ethertype to vlan's ethertype
    ethertype = ethernet_vlan_header->get_ethertype_host_byte_order();
}
```

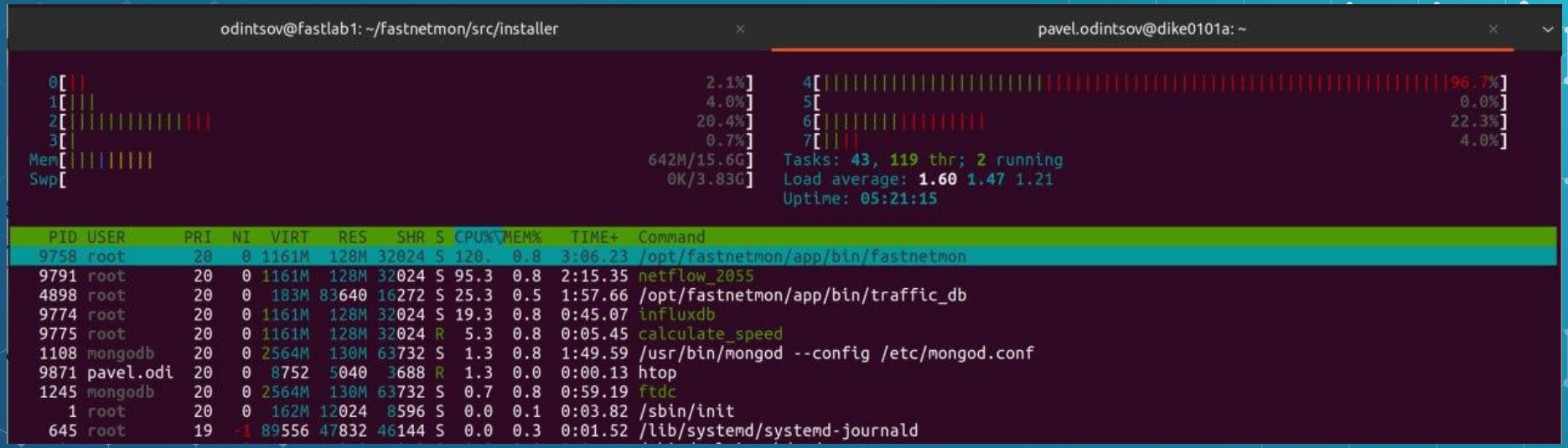
Performance challenges: 50 000 UDP pkts /s



Performance challenges: 100 Mbits



Performance challenges: single CPU core limit



Let’s enable UDP reuse_port with 4 threads

```
1740 root      20   0 1462M  204M 32028 S 19.4  6.9   0:09.61 netflow_2055_0
1741 root      20   0 1462M  204M 32028 S  0.0  6.9   0:00.00 netflow_2055_1
1742 root      20   0 1462M  204M 32028 R 97.0  6.9   0:45.75 netflow_2055_2
1743 root      20   0 1462M  204M 32028 S  0.0  6.9   0:00.00 netflow_2055_3
```

Why it did not work? Same 5 tuple

```
01:33:12.190302 IP 10.21.      .33092 > 10.      .2055: UDP, length 1398
01:33:12.190314 IP 10.21.      .33092 > 10.      .2055: UDP, length 1360
01:33:12.505186 IP 10.21.      .33092 > 10.      .2055: UDP, length 1434
01:33:12.505199 IP 10.21.      .33092 > 10.      .2055: UDP, length 1404
01:33:12.505211 IP 10.21.      .33092 > 10.      .2055: UDP, length 1451
01:33:12.505241 IP 10.21.      .33092 > 10.      .2055: UDP, length 1453
01:33:12.505251 IP 10.21.      .33092 > 10.      .2055: UDP, length 1436
01:33:12.505262 IP 10.21.      .33092 > 10.      .2055: UDP, length 1398
01:33:12.505275 IP 10.21.      .33092 > 10.      .2055: UDP, length 1470
01:33:12.505294 IP 10.21       .33092 > 10.      .2055: UDP, length 1402
01:33:12.505310 IP 10.21       .33092 > 10.      .2055: UDP, length 1453
01:33:12.505321 IP 10.21.      .33092 > 10.      .2055: UDP, length 1398
01:33:12.505334 IP 10.21.      .33092 > 10.      .2055: UDP, length 1464
01:33:12.505343 IP 10.21.      .33092 > 10.      .2055: UDP, length 1440
```

Solution? eBPF!

```
; Load random 32 bit value to A
ld rand
; Mod divide register A by number of worker threads
mod #4
ret a
```

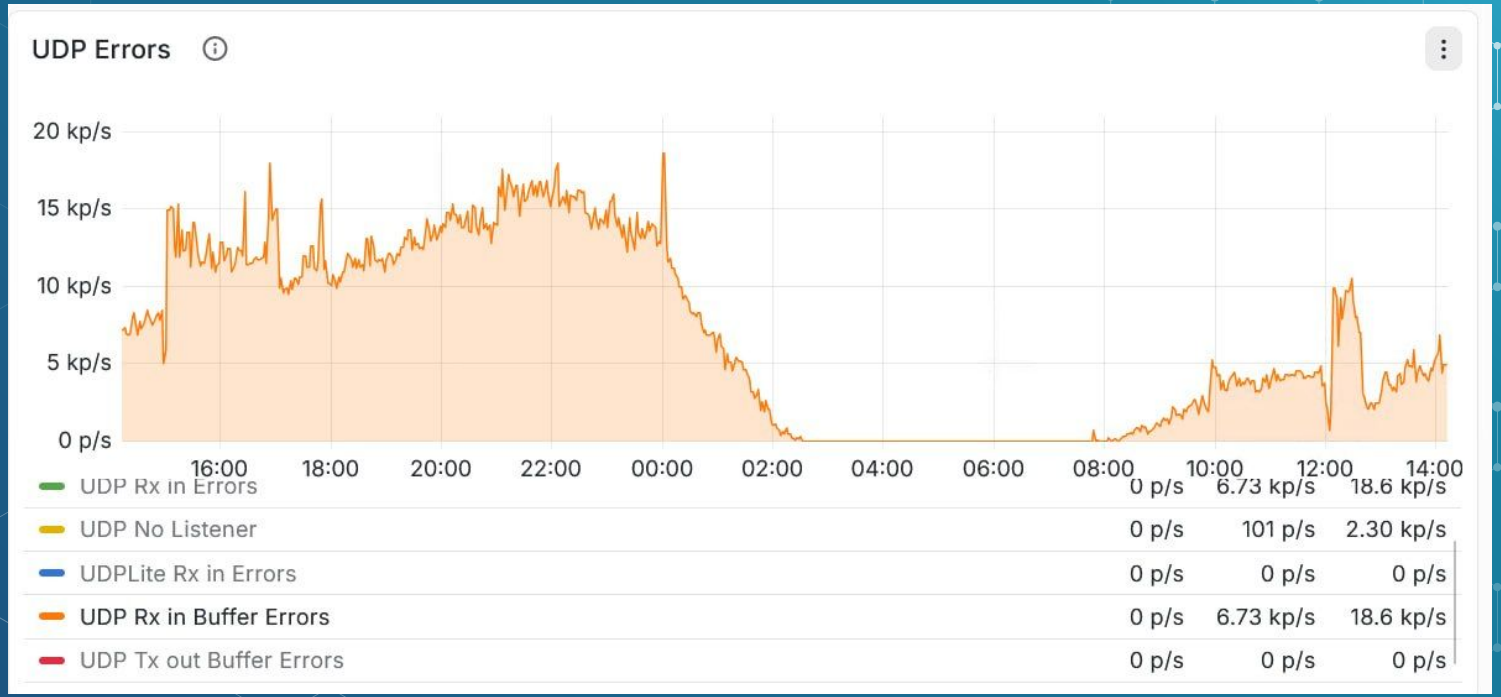
Linux: I do not like it

```
We will listen on :::2055 udp port
Setting reuse port
Loading BPF to implement random UDP traffic distribution over available threads
Successfully loaded reuse port BPF
Successful bind
We will listen on :::2055 udp port
Setting reuse port
Loading BPF to implement random UDP traffic distribution over available threads
Successfully loaded reuse port BPF
Can't bind on port: 2055 on host :: errno:98 error: Address already in use
Cannot create / bind socket
Started capture
```

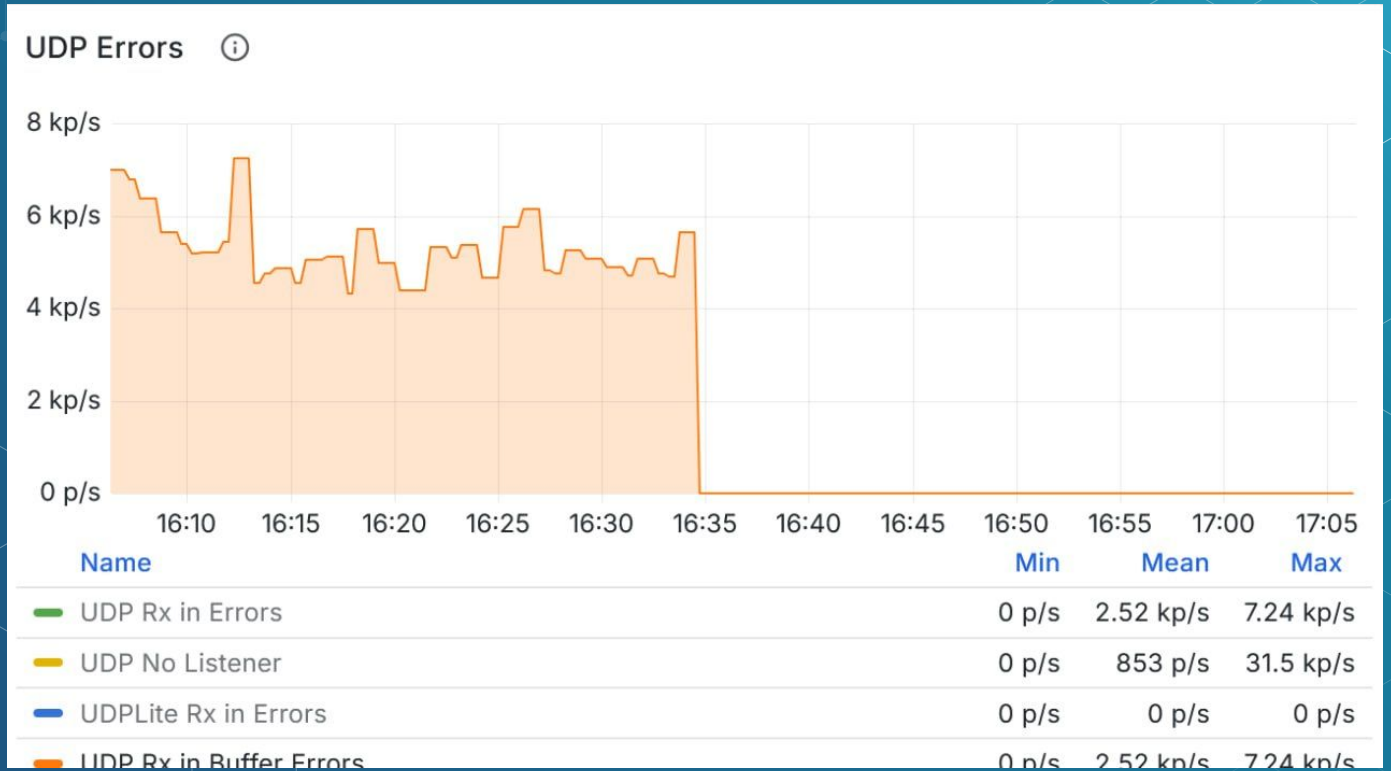
reuse_addr to save us

```
12428 root      20   0 1610M  455M 33108 R 58.8 15.3  1:33.59 netflow_2055_3
12425 root      20   0 1610M  455M 33108 S 56.1 15.3  1:33.52 netflow_2055_0
12426 root      20   0 1610M  455M 33108 S 53.5 15.3  1:33.67 netflow_2055_1
12427 root      20   0 1610M  455M 33108 R 52.1 15.3  1:33.36 netflow_2055_2
```

UDP Drops in Traffic Database



UDP Drops in Traffic Database: async insert



UDP reuse_port + eBPF Linux kernel bug

- https://pavel.network/rocky-road-towards-ultimate-udp-load-balancing-server-on-linux/
- https://pavel.network/rocky-road-towards-ultimate-udp-server-with-bpf-based-load-balancing-on-linux-part-2/
- https://pavel.network/multiple-linux-kernel-inconsistencies-when-so_attach_reuseport_cbpf-used-with-udp-sockets/

THANKS!

ANY QUESTIONS?

You can find me at:

- @odintsov_pavel
- pavel@fastnetmon.com
- linkedin.com/in/podintsov