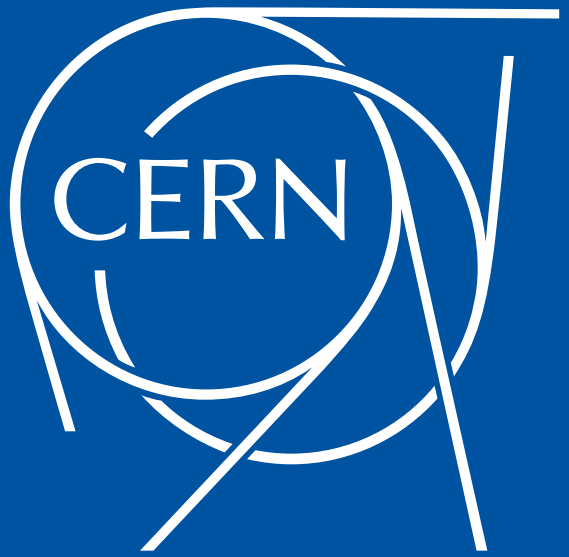

CERN

Programming switches for flow label accounting, forwarding and routing

CERN Data Center
IT-CS-NE Department

Carmen Misa Moreira
Edoardo Martelli

Outline

- Testbed
  - Programming protocol-independent packet processors: P4 language
  - EdgeCore Wedge100BF-32QS
  - GÉANT P4Lab
  - Network Operating System
  - Packet and flow marking specification
- On-going project and tests
  - Accounting and forwarding
    - First approach: layer 3
    - Second approach: layer 2
  - Routing
    - MultiONE
- Conclusions and future lines

Testbed

Programming protocol-independent packet processors: P4 language

Language for programming the data plane of network devices

- Define how packets are processed
- P4 program structure: header types, parser/deparser, match-action tables, user-defined metadata and intrinsic metadata

Domain-specific language designed to be implementable on a large variety of targets

- Programmable network interface cards, FPGAs, software switches and hardware ASICs.

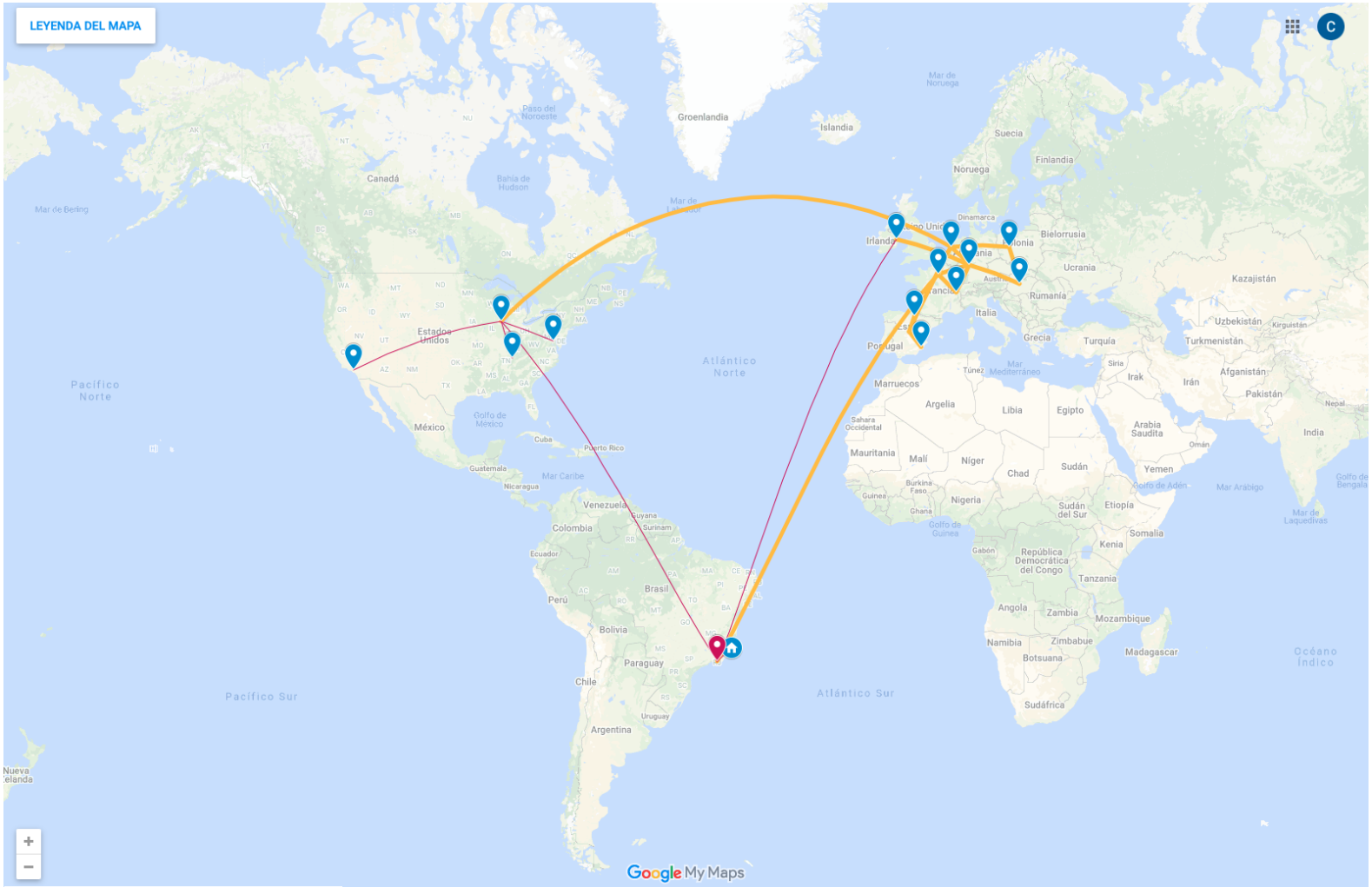
EdgeCore Wedge100BF-32QS

- 100GbE Data Center Switch
  - Bare-Metal Hardware
  - L2/L3 Switching
  - 32xQSFP28 Ports
- Data-Plane Programmability
  - Intel Tofino Switch Silicon
  - Barefoot Networks
- Quad-Pipe Programmable Packet Processing Pipeline
  - 6.4 Tbps Total Bandwidth
- CPU: Intelx86 Xeon 2.0GHz
  - 8-core/48GB/2TB SSD

Figure: Intel Tofino P4-programmable Ethernet Switch ASIC

Figure: EdgeCore Wedge100BF-32QS

GÉANT P4Lab


LEYENDA DEL MAPA
Groenlandia
Islandia
Canadá
Estados Unidos
México
Pacífico Norte
Atlántico Norte
Pacífico Sur
Atlántico Sur
Suecia
Noruega
Finlandia
Irlanda
Dinamarca
Bielorrusia
Ucrania
Kazajistán
Rumania
Italia
Grecia
Turquía
Portugal
Marruecos
Argelia
Libia
Egipto
Mauritania
Malí
Níger
Chad
Sudán
Nigeria
Etiopía
Somalia
Kenia
Tanzania
Angola
Zambia
Mozambique
Namibia
Zimbabue
Botsuana
Madagascar
Sudáfrica
Irak
Irán
Afganistán
Pakistán
India
Arabia Saudita
Yemen
Uzbekistán
Turkmenistán
Cuba
Puerto Rico
Guatemala
Nicaragua
Venezuela
Colombia
Ecuador
Perú
Brasil
Bolivia
Paraguay
Chile
Argentina
Uruguay
Google My Maps

Network Operating System

RARE/FreeRtr

- Controls the data plane by managing entries in routing tables
- Free and open source router operating system
- Export forwarding tables to DPDK or hardware switches
  - via OpenFlow or P4lang
- No global routing table
  - Every routed interface must be in a virtual routing table

Packet and flow marking specification

Flow label field of IPv6 header: 20 bits [Packet marking specification ref. [1, 2]]

- 5 entropy bits to match RFC 6436
- 9 bits to define the science domain
- 6 bits to define the application/type of traffic

Bits 12 - 13 Entropy	Bits 14 - 22 Science Domain	Bit 23 Entropy	Bits 24 - 29 Application	Bits 30 - 31 Entropy

Astro/HEP Science Domains:
Reserved - 0
Default - 65536
ATLAS - 32768
CMS - 98304
LHCb - 16384
ALICE - 81920
Bellell - 49152
SKA - 114688
LSST - 73728
DUNE - 8192

Application:
Reserved - 0
Default - 4
perfSONAR - 8
Cache - 12
DataChallege - 16

On-going project and tests

Accounting and forwarding

First approach: layer 3

Network configuration:

- Virtual Routing Forwarding
- Policy-based routing based on flow label field value
  - Flow label 10 → VLAN 40
  - Flow label 20 → VLAN 41
- SRV-01 managed by Cisco TRex Realistic Traffic Generator
  - Python script Scapy library: generate IPv6 packets flow label tagged
  - Cisco TRex Client: Python script → Scapy library
  - Cisco TRex Server: get statistic of the traffic in real-time
- SRV-02 managed by DPDK FreeRtr

First approach: layer 3

Results:

- Successful PBR routing based on the flow label field

```
E513-E-YECWH-1#show interfaces traffic
interface       state   tx          rx          drop
sdn1            up      0+0         0+0         0+0
sdn1.10         up      0+0         0+0         0+0
sdn1.20         up      0+0         0+0         0+0
sdn110          up      0+0         0+0         0+0
sdn110.40       up      0+165454    0+0         0+0
sdn110.41       up      0+0         0+0         0+0
sdn111          up      0+0         0+0         0+0
sdn111.60       up      0+165454    0+0         0+0
sdn3            up      0+0         0+0         0+0
sdn50           up      0+0         0+0         0+0
sdn50.50        up      0+0         0+164312    0+0
sdn51           up      0+0         0+0         0+0
sdn51.70        up      0+0         0+164312    0+0
```

Figure: P4 switch statistics per interface and VLAN

[NOTE: test performed for fl=10 → VLAN 40]

- Cisco TRex + DPDK FreeRtr exploits NIC at its full capacity

```
-Per port stats table
      ports |               0 |               1
-----------------------------------------------------------
   opackets |        76661051 |              41
     obytes |    115298161292 |            3662
   ipackets |             228 |        50986637
     ibytes |           28054 |     76683593170
    ierrors |               0 |               0
    oerrors |               0 |               0
      Tx Bw |      9.84 Gbps  |       0.00   bps

-Global stats enabled
 Cpu Utilization : 2.8  %  694.0 Gb/core
 Platform_factor : 1.0
 Total-Tx        :        9.84 Gbps
 Total-Rx        :        9.84 Gbps
 Total-PPS       :      818.20 Kpps
 Total-CPS       :        0.00  cps
```

Figure: Cisco TRex statistics on the transmitter and receiver side

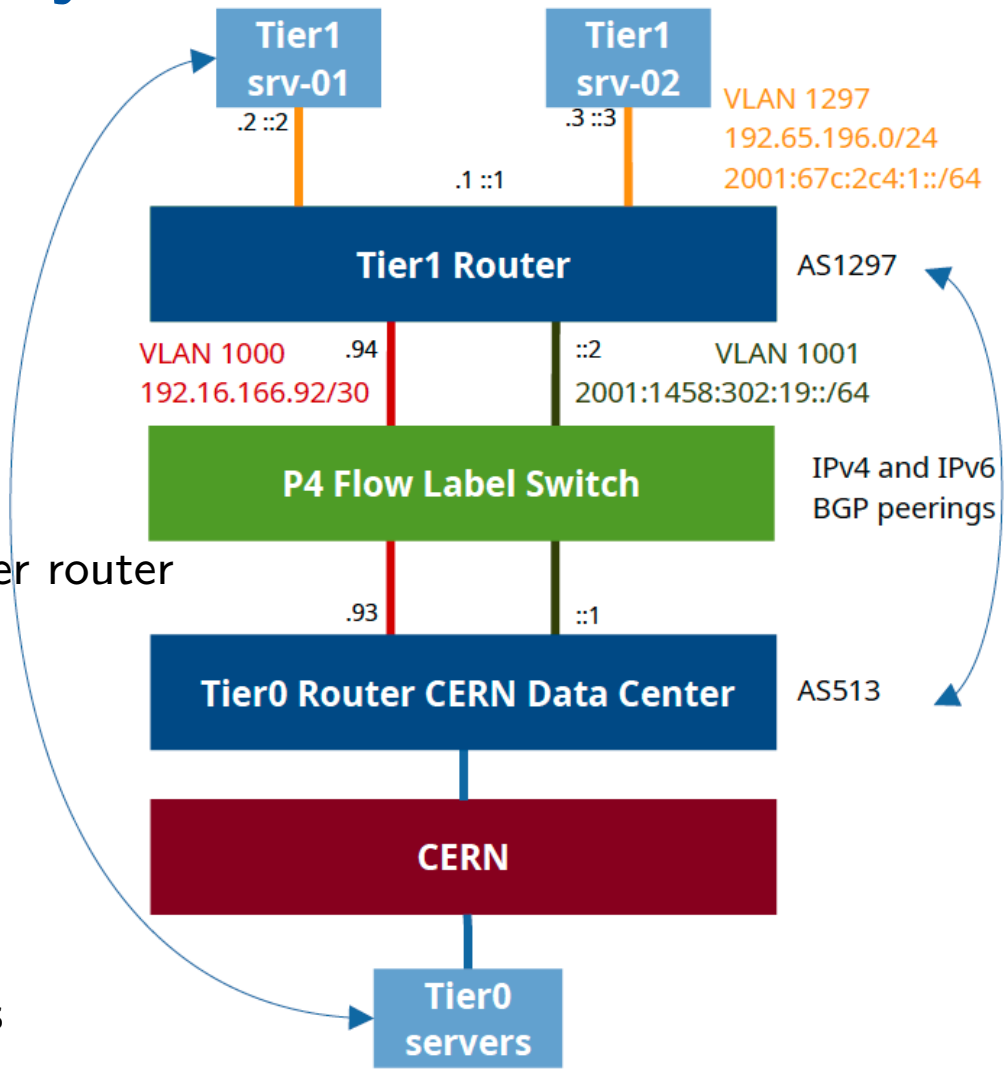
[NOTE: test performed with Intel 82599ES 10 GbE Controller: IXGBE driver]

Second approach: layer 2

Network configuration:

- Emulates a Tier 1/0 link
- Tier1/0 routers
  - IPv4/IPv6 BGP peerings
- Tier0 router
  - LHCOPN production border router
- Pure layer 2 bridges
  - VLAN 1000: IPv4 traffic
  - VLAN 1001: IPv6 traffic
- Tier0 servers
  - OpenStack product servers

Data transfers with flow label marked packets



Second approach: layer 2

P4 switch network configuration: pure layer 2 bridges

```
access-list acl_all_ipv6_flowlabels
     # Match <Experiment> and <ANY Application>
     sequence 10 permit all any all any all flow 131076 & 261884
     sequence 11 permit all any all any all flow 65540  & 261884
     sequence 12 permit all any all any all flow 196612 & 261884
     sequence 13 permit all any all any all flow 32772  & 261884
     # Match <Experiment> and <perfSONAR Application>
     sequence 20 permit all any all any all flow 131072 & 261632
     sequence 21 permit all any all any all flow 65536  & 261632
     sequence 22 permit all any all any all flow 196608 & 261632
     sequence 23 permit all any all any all flow 32768  & 261632
     # Permit the rest of the traffic
     sequence 30 permit all any all any all
     exit
```

```
ATLAS <any>
CMS   <any>
LHCb  <any>
ALICE <any>
```

```
ATLAS <perfSONAR>
CMS   <perfSONAR>
LHCb  <perfSONAR>
ALICE <perfSONAR>
```

```
interface sdn1.1000
     description [VLAN ID=1000]
     bridge-group 1
     no shutdown
     no log-link-change
     exit
```

VLAN 1000 belongs to bridge 1

```
interface sdn1.1001
     description [VLAN ID=1001]
     bridge-group 2
     bridge-filter ipv6in acl_all_ipv6_flowlabels
     no shutdown
     no log-link-change
     exit
```

VLAN 1001 belongs to bridge 2
Filter IPv6 traffic at the input based on the access-list sentences

Second approach: layer 2

IPv6 packets flow label tagged were generated by using:

- iperf3
- ipv6_flow_label library developed by Marian Babik
- eBPF_flow_label library developed by Tristan Sullivan

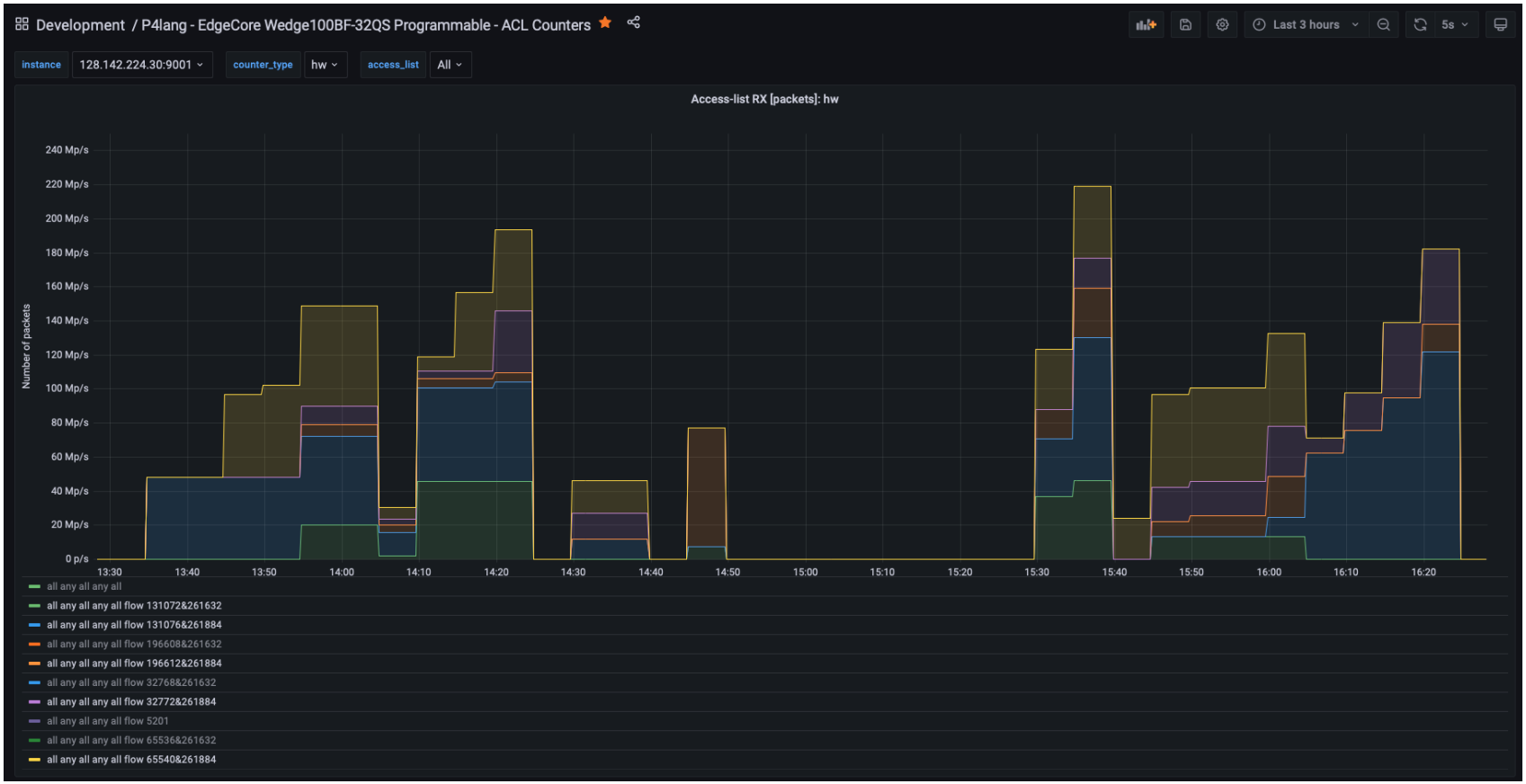
```
E513-E-YECWH-1#show access-list acl_all_ipv6_flowlabels
seq  txb  txp  rxb            rxp          last      timout    cfg
10   0+0  0+0  0+12374638771  0+8743031    00:03:02  00:00:00  permit all any all any all flow 131076&261884   ATLAS <any>
11   0+0  0+0  0+37019728635  0+24984028   00:02:30  00:00:00  permit all any all any all flow 65540&261884    CMS   <any>
12   0+0  0+0  0+23940164205  0+15797973   00:02:00  00:00:00  permit all any all any all flow 196612&261884   LHCb  <any>
13   0+0  0+0  0+18150017192  0+12017039   00:02:00  00:00:00  permit all any all any all flow 32772&261884    ALICE <any>

20   0+0  0+0  0+30346726207  0+20005622   00:01:29  00:00:00  permit all any all any all flow 131072&261632   ATLAS <perfSONAR>
21   0+0  0+0  0+25281078379  0+16663278   00:01:29  00:00:00  permit all any all any all flow 65536&261632    CMS   <perfSONAR>
22   0+0  0+0  0+28556351375  0+19008806   00:00:58  00:00:00  permit all any all any all flow 196608&261632   LHCb  <perfSONAR>
23   0+0  0+0  0+37078713993  0+25770785   00:00:26  00:00:00  permit all any all any all flow 32768&261632    ALICE <perfSONAR>
30   0+0  0+0  0+2715536713   0+1802921    00:00:26  00:00:00  permit all any all any all
```

Figure: Counters of the access-list on the P4 switch

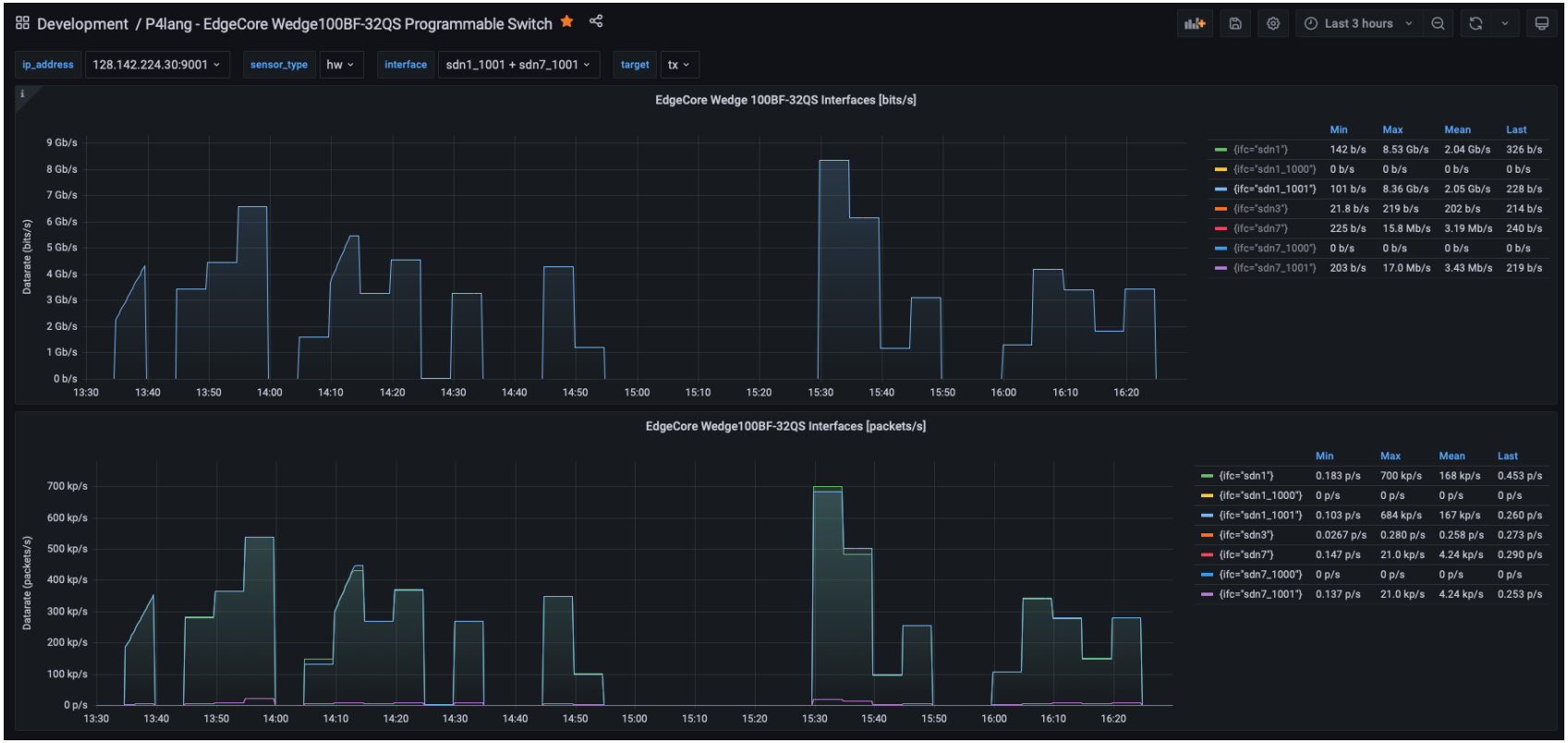
Second approach: layer 2

The matching ACL counters are exported to Prometheus and Grafana:



Second approach: layer 2

The interface counters are also exported to Prometheus and Grafana:



Routing

MultiONE

Separate the traffic into different VPNs based on the IPv6 flow label value.

- MultiONE network: 3 VPNs (blue, green, red)

  + a default VPN for IPv4 and untagged traffic

    - COTS routers with BGP and IPv6.
    - Peering with the site routers and redistribute the received prefixes.

- P4 site routers: to access the proper multiONE VPN based on the routes received from BGP a flow label tag of the packets.

    - P4 programmable switches [P4Lab].
    - Announce the IPv6 prefixes of the local servers to the connected VRFs via BGP.

- Site servers: generate and receive tagged traffic.

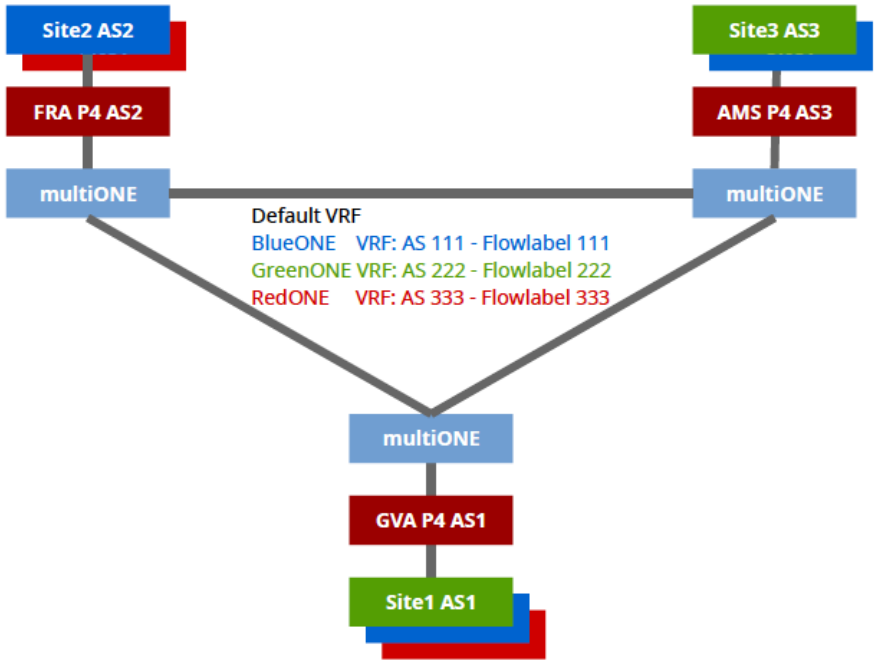


Figure: MultiONE testbed.

Conclusions and future lines

- IPv6 flow label accounting and forwarding can be implemented at layer 3 and layer 2.

- Next step, evaluate the use of the IPv6 flow label routing approach to implement MultiONE.

Thanks for your attention!

Programming switches for flow label accounting, forwarding and routing

CERN Data Center
IT-CS-NE Department

Carmen Misa Moreira
Edoardo Martelli


CERN