Fully Open Source Smart OpenStack Cloud: NOW and BEYOND

Ash Bhalgat, Sr. Dir, Cloud Marketing, Mellanox Technologies

Franck Baudin, NFV Product Manager, Red Hat

Mark Iskra, Technical Marketing Expert, Nuage Networks

OpenStack Summit, Berlin. Nov 14, 2018

Fully Open Source Smart OpenStack Cloud: NOW and BEYOND

NOW:

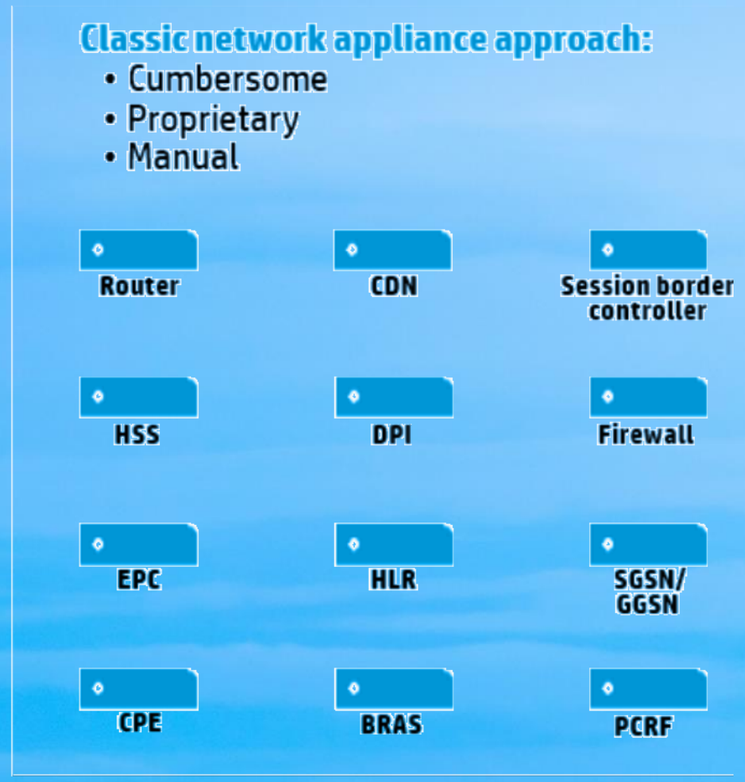
- State of Telco Cloud and need for offloads
- What are SmartNICs and how offloads work on SmartNICs ?
- What are the open source elements to enable offloads ? Blueprints, configuration tips/guides, etc.
- How is this all deployed ? Recorded demo!

BEYOND:

- What’s coming next ? Port Mirroring, VFLAG, QoS, Connection Tracking, Commercial VNF integration

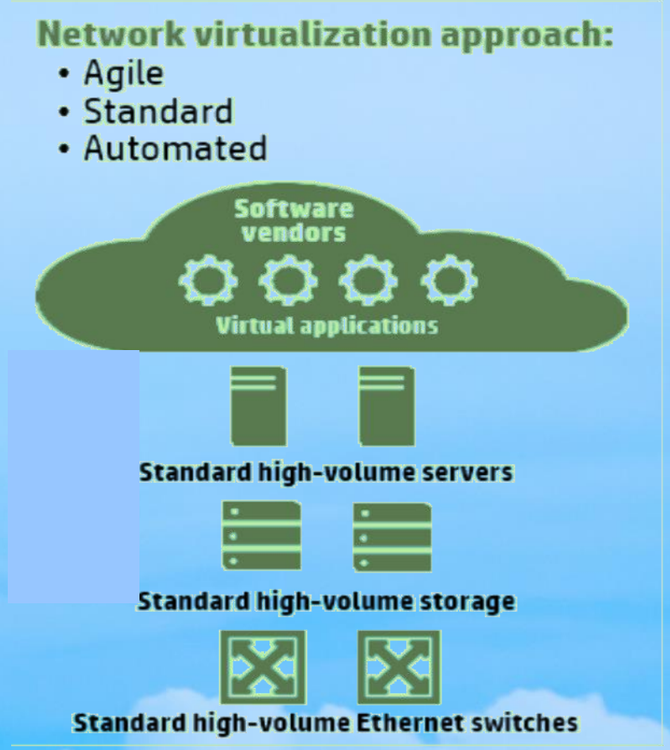
Transformation of Telco Cloud

Since 1980s



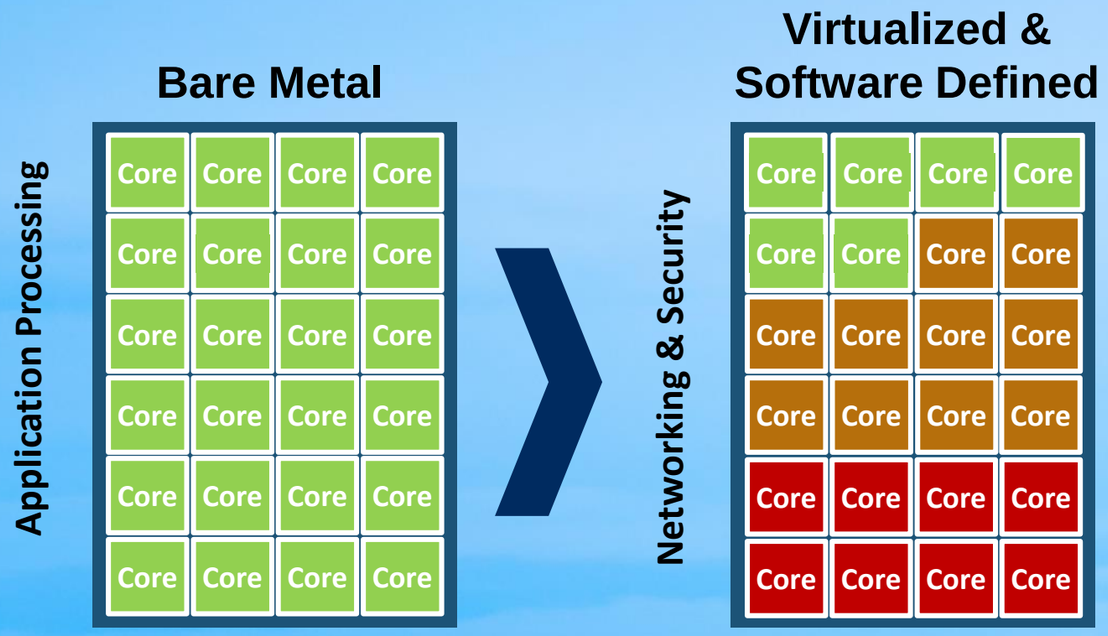
Hardware Defined Everything World

Starting in 2007 …



Software Defined Everything World

Software Defined Everything Creates Bottlenecks

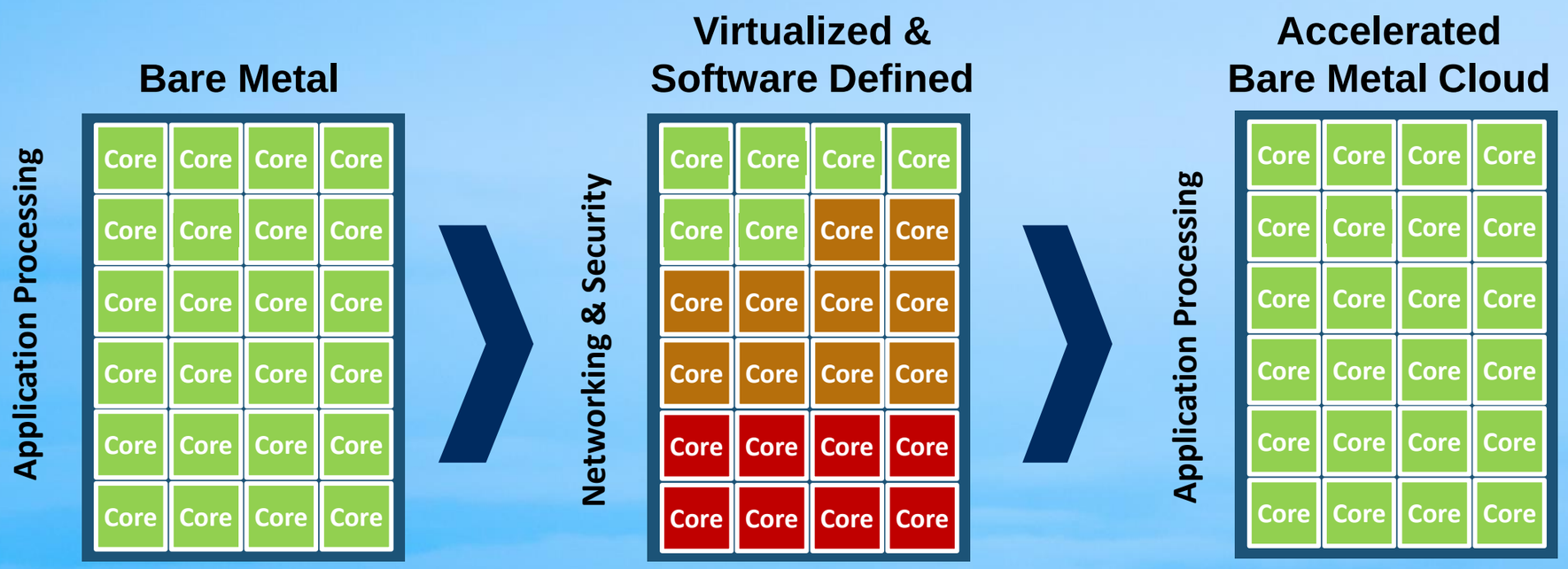


Core Available for Application Processing

Core Software Defined Everything (SDX) Consumes CPU cores for Packet/Flow Processing and Creates Bottlenecks
- Virtualization, Storage, Switching, Routing, Load Balancing

Core Security: Consumes CPU cores for Security Processing
- Layer 4 Firewall, encryption, host introspection
- Intrusion detection & prevention

Software Defined Everything Creates Bottlenecks

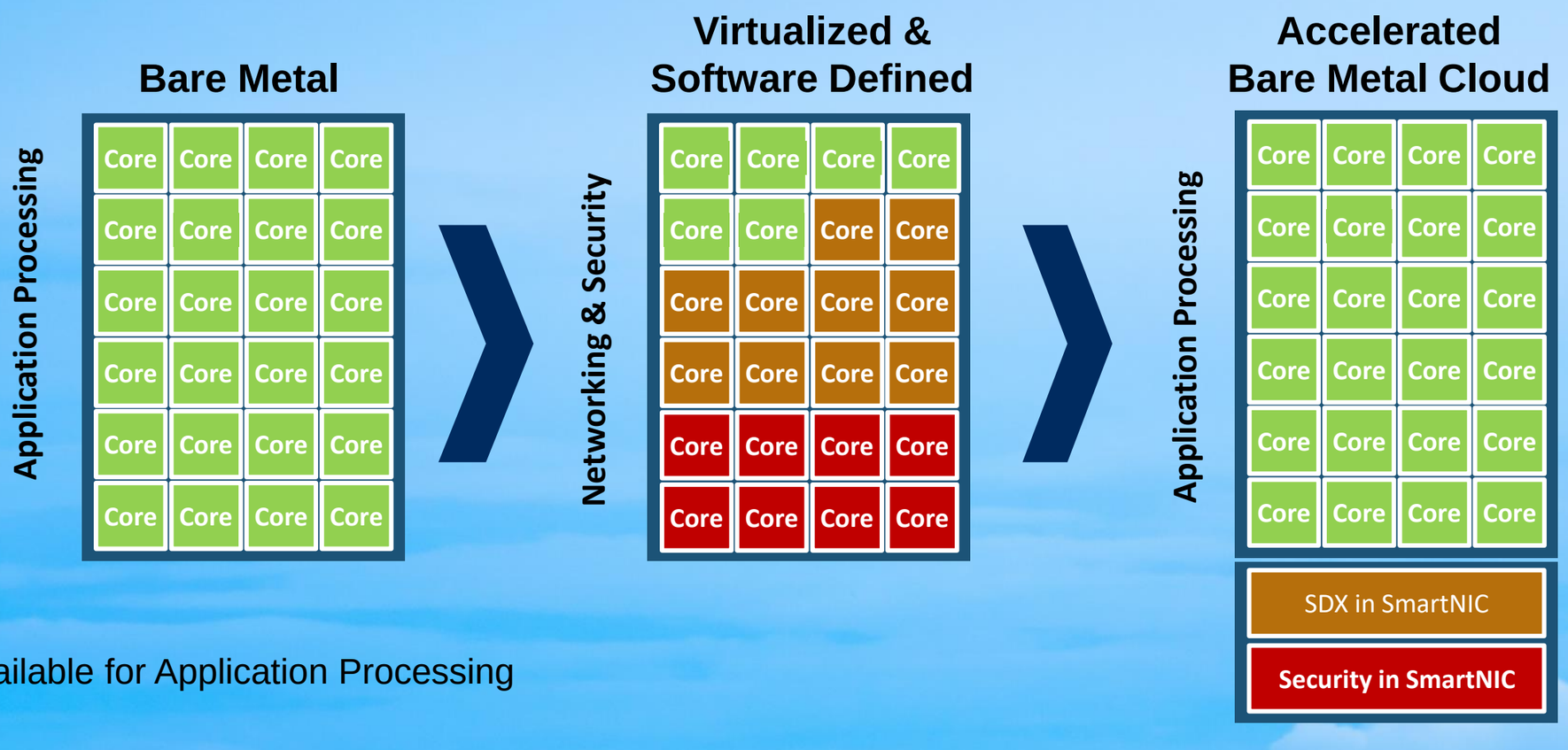


Core Available for Application Processing

Core Software Defined Everything (SDX) Consumes CPU cores for Packet/Flow Processing and Creates Bottlenecks
- Virtualization, Storage, Switching, Routing, Load Balancing

Core Security: Consumes CPU cores for Security Processing
- Layer 4 Firewall, encryption, host introspection
- Intrusion detection & prevention

Software Defined Everything Creates Bottlenecks



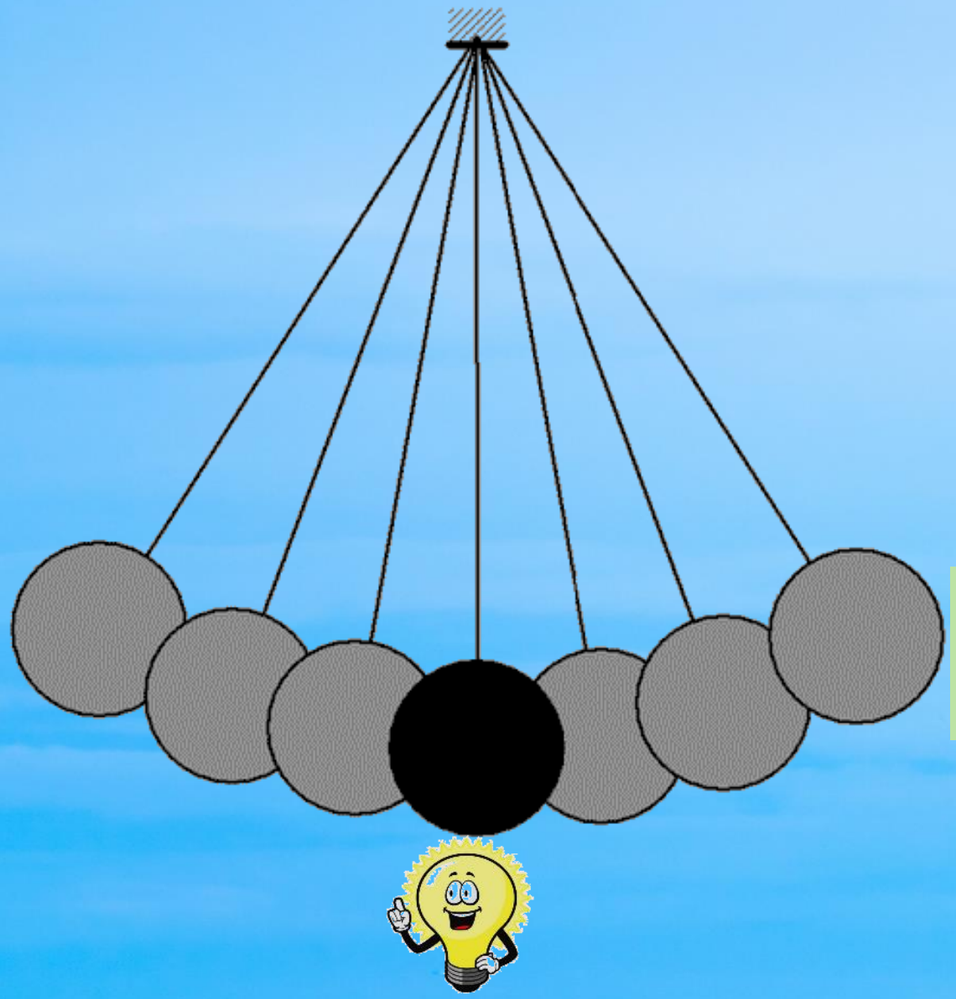
Core Available for Application Processing

Core Software Defined Everything (SDX) Consumes CPU cores for Packet/Flow Processing and Creates Bottlenecks
- Virtualization, Storage, Switching, Routing, Load Balancing

Core Security: Consumes CPU cores for Security Processing
- Layer 4 Firewall, encryption, host introspection
- Intrusion detection & prevention

The State of Telco Cloud

Hardware Defined Everything World



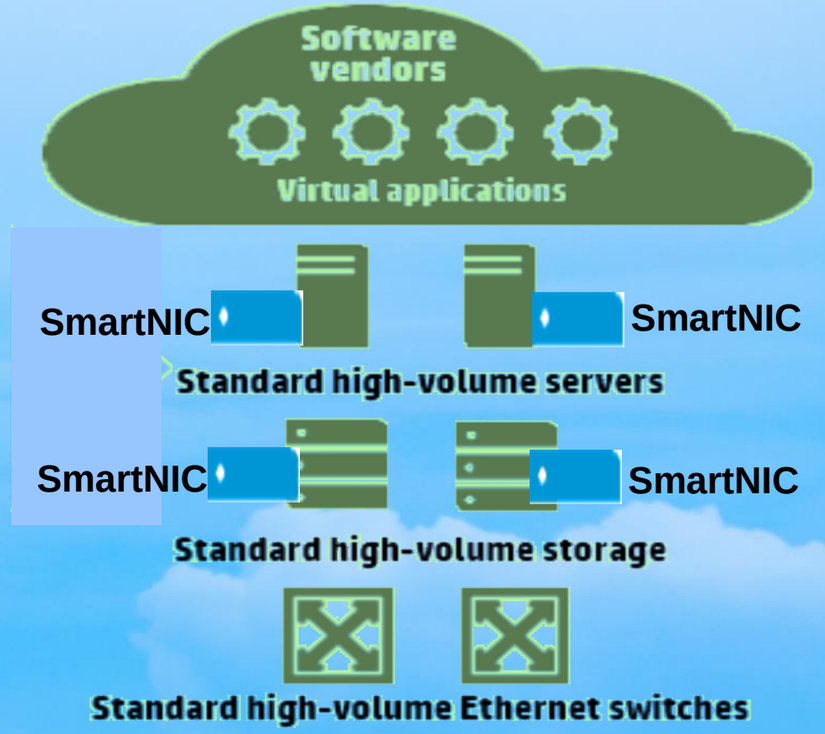
Software Defined Everything World

Hardware Accelerated | Software Defined Everything World

2015 Onwards…

Network virtualization approach:

- Agile
- Standard
- Automated



Open Virtual Switch (OVS) – A Love/Hate Relationship!

LOVE—OVS Benefits

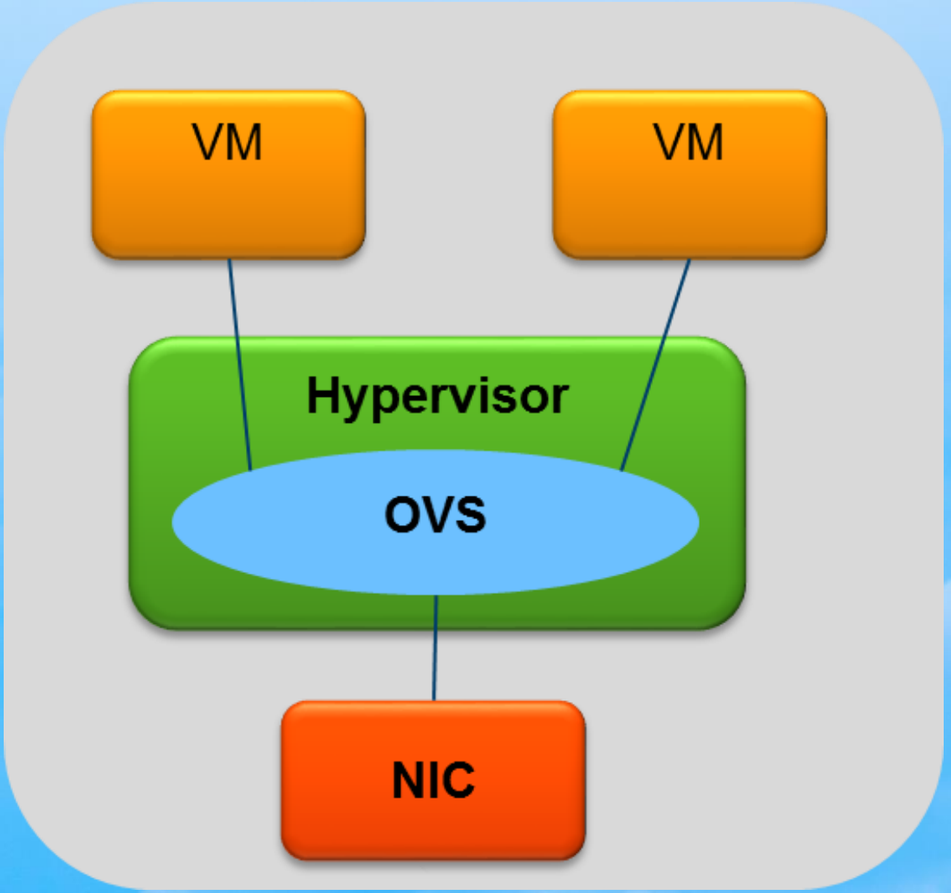
- Hypervisor uses virtual switches (like OVS) as the data forwarding plane
- Widely used in the cloud for SDN (e.g. enforce policies)
- Supports L2-L3 networking features
    - L2 & L3 Forwarding, NAT, Firewall, Load Balancer, etc.
    - Flow based

HATE—OVS Challenges

- Awful Packet Performance: <1M PPS with 2-4 cores
- Consumes Your CPU: Even with 12 cores, can't reach 1/3rd of a 100G NIC's bandwidth
- Poor Experience: High, unpredictable latency; packet drops

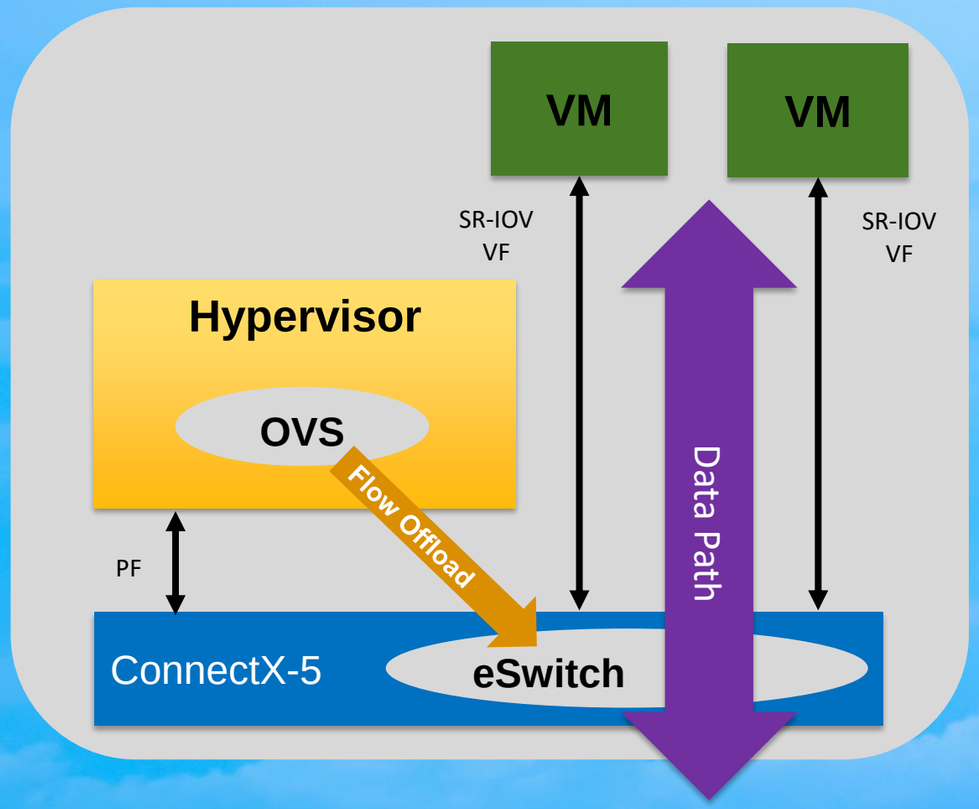
Solution

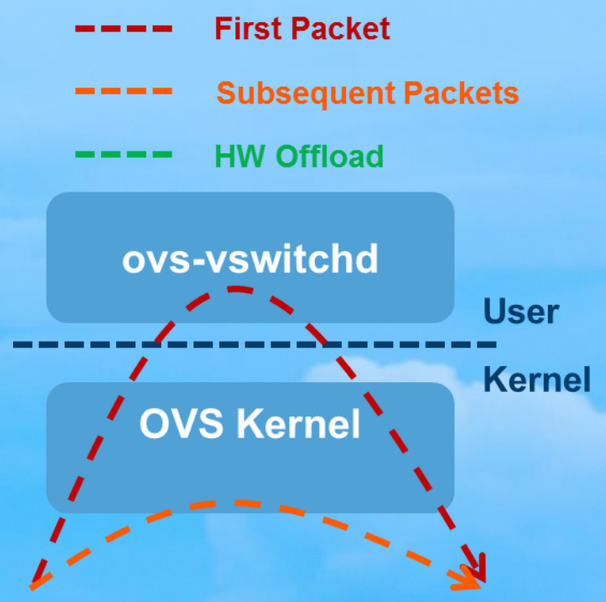
- Hardware Acceleration with Software Programmability
- All the benefits, no compromises



Full OVS Hardware Offload – Best of Both Worlds!

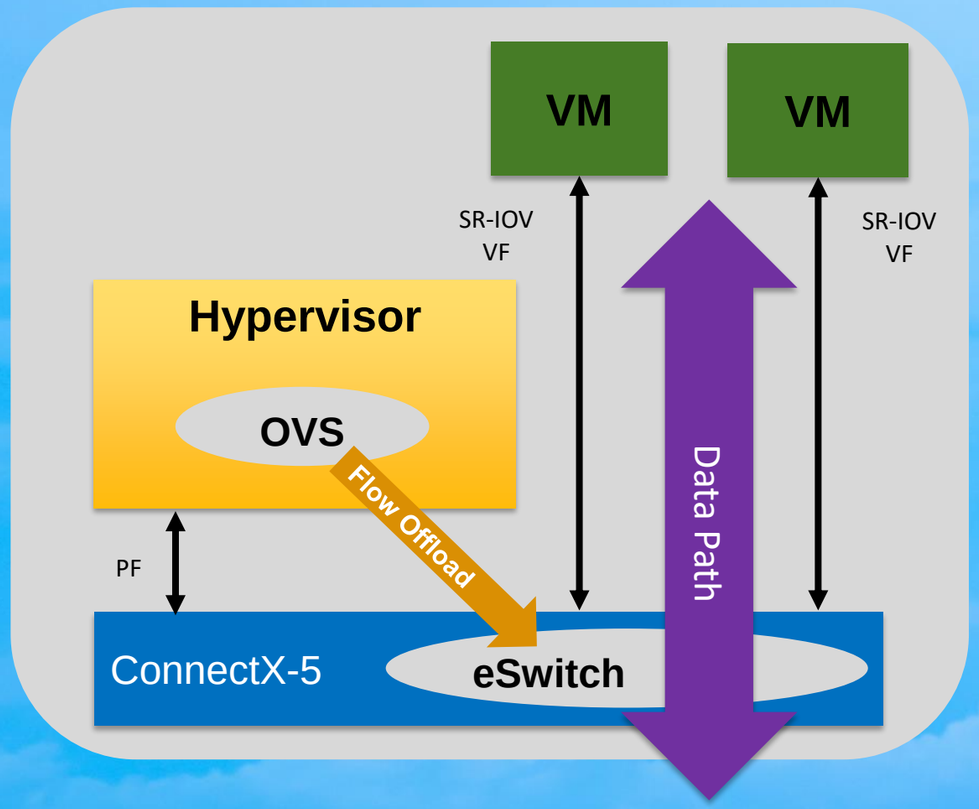
- **Mellanox OVS Hardware Offload: Accelerated Switching and Packet Processing (ASAP^2)**
  - Open vSwitch as Standard SDN Controller
  - OVS data-plane offload to NIC-embedded Switch (eSwitch) – SR-IOV Data Path
- **Best of Both Worlds**
  - SDN Programmability with Blazing Fast Switching Performance

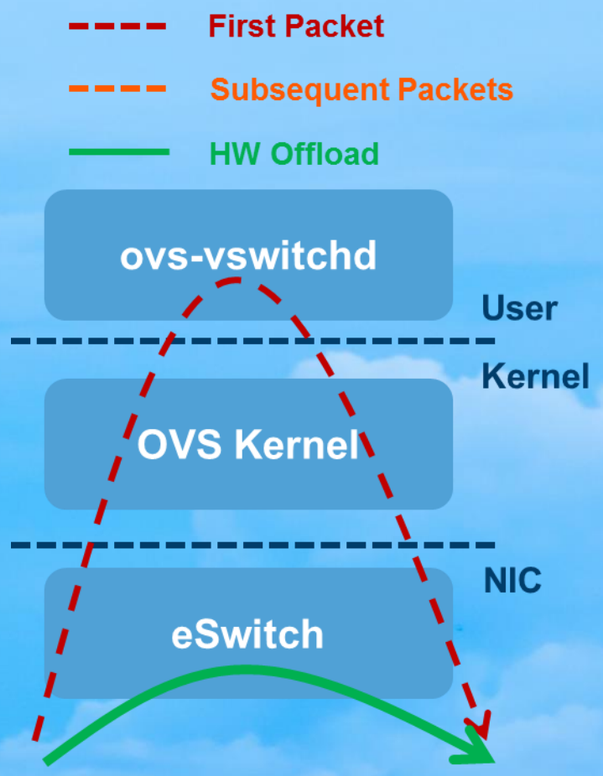




Full OVS Hardware Offload – Best of Both Worlds!

- **Mellanox OVS Hardware Offload: Accelerated Switching and Packet Processing (ASAP^2)**
  - Open vSwitch as Standard SDN Controller
  - OVS data-plane offload to NIC-embedded Switch (eSwitch) – SR-IOV Data Path
- **Best of Both Worlds**
  - SDN Programmability with Blazing Fast Switching Performance



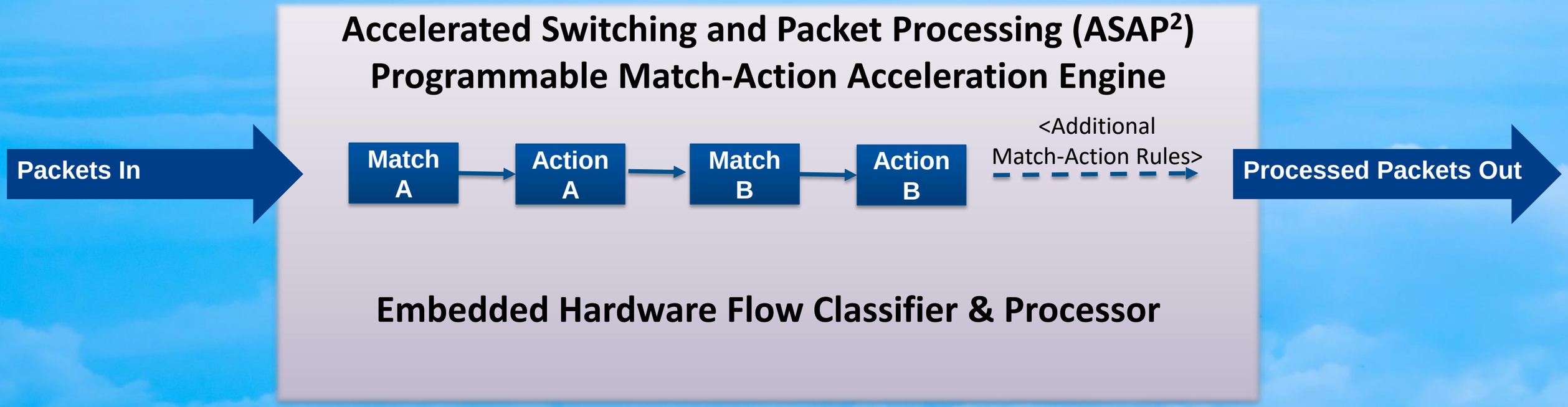


Accelerated Network Processing

- Control Path: SDN Controller
  - Program policies for virtual Switch packet forwarding
- Data Path: Flexible Programmable Engine for Packet Operation
  - Packet classification (flow match)
  - Action based on match result
- ConnectX ASAP^2 Technology
  - Hardware acceleration engine offloads both the classification and actions
- Common Actions:
  - Drop, Forward, Header Rewrite, Mirroring, Overlay Encap/Decap, Telemetry, Increment/Decrement counters

ASAP^2 NFV Application Acceleration

- Load balancing, NAT
- Virtual switching/routing
- Firewall, DDOS
- Monitoring, network traffic analyzer



Unmatched Application Performance & Efficiency

NFV

Network Function Virtualization

DPDK

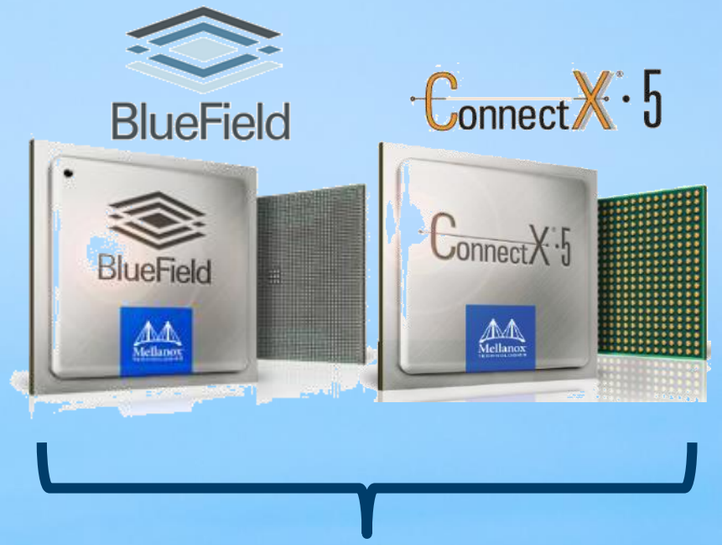
DATA PLANE DEVELOPMENT KIT

Watch Video

ASAP2

Accelerated Switching & Packet Processing

Watch Video



Both BlueField & ConnectX Fully Support both DPDK & ASAP² Delivering Unmatched Performance and Efficiency

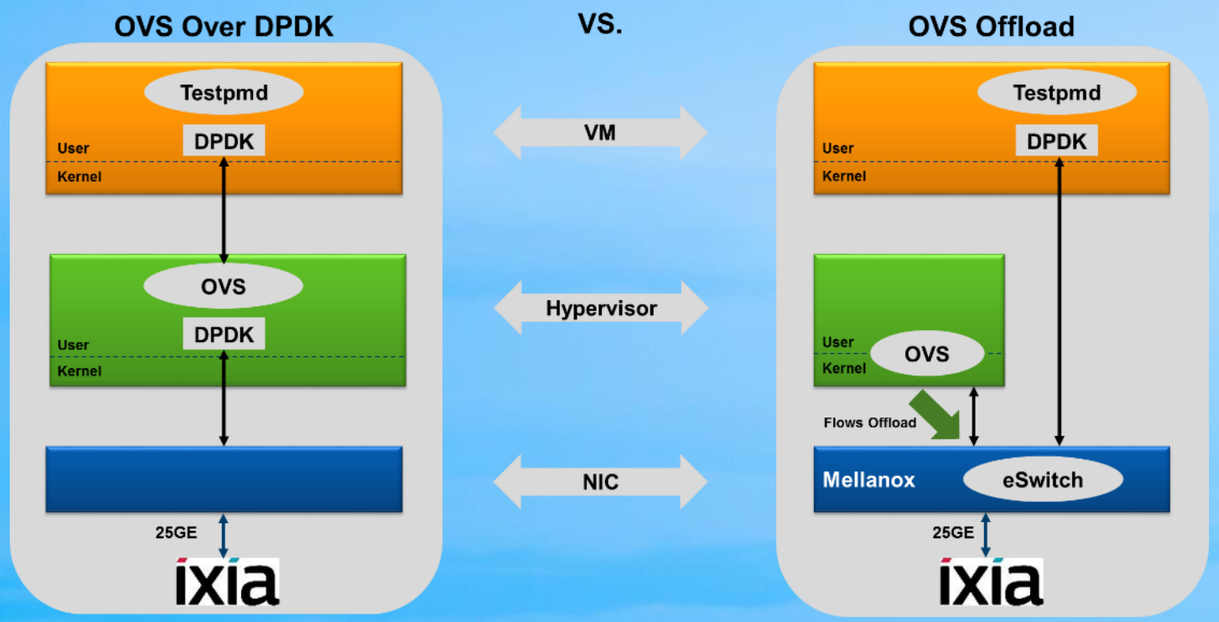
DPDK

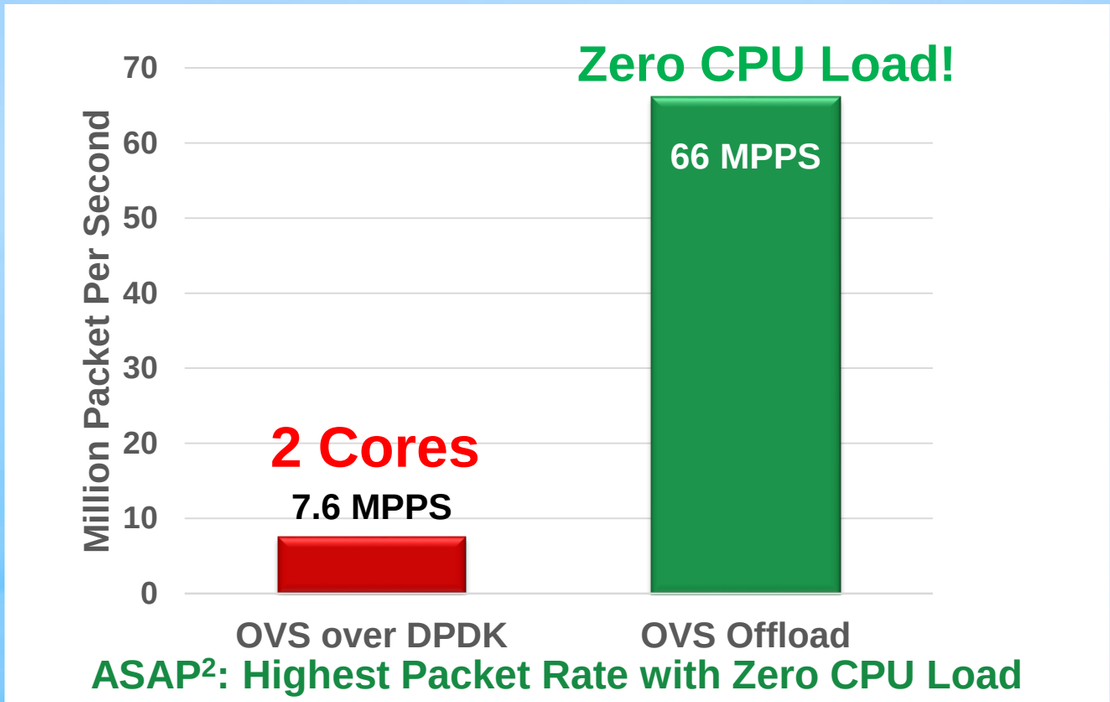
- CPU does the work
- Highest DPDK performance
- Line rate packet forwarding
- DPDK project maintainer

ASAP²

- NIC does the work
- HW OVS acceleration
- **8X-10X** performance gain
- **Zero CPU** utilization

OVS Performance: DPDK vs. ASAP²





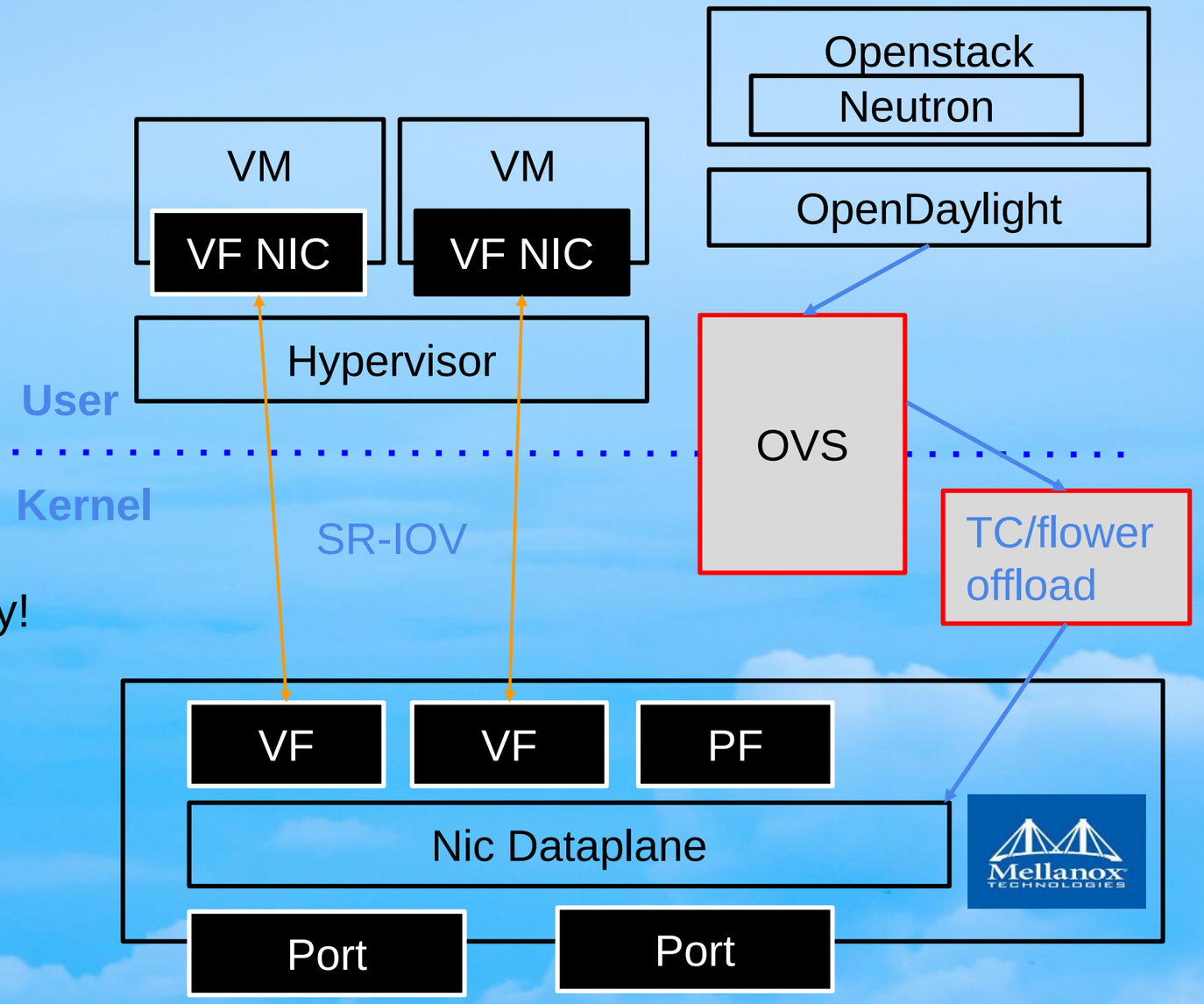
- **Mellanox OVS Offload (ASAP²)**
  - **20X** higher performance than vanilla OVS
  - **8X-10X** better performance than OVS-DPDK
  - **Line rate performance** at 25/40/50/100Gbps

- **Open Source - No Vendor Lock-In**
  - **Adopted broadly** by Linux community & industry
  - **Full Community Support** (OVS, Linux, OpenStack)
  - **Industry Ecosystem Support**
    - Nuage/Nokia, Red Hat, Open Contrail, etc.

OVS OFFLOAD

OVS kernel TC/Flower HW offload

- Enable OVS features day 1
  - fallback to kernel OVS
- Available since OpenStack Queens
- Same principles applies to other SDN/vSwitches/vRouters
- Required OpenStack work but not only!
- The magic is: **VF Representor Port**



Projects Involved

simplified view ;-)

CentOS®

libvirt VIRTUALIZATION API

QEMU

OvS Open vSwitch

OPNFV

RDO

openstack CLOUD SOFTWARE

OPEN DAYLIGHT

Blueprints (All implemented in Queens)

This is just the OpenStack part!

nova
https://review.openstack.org/#/c/398265/

neutron
https://review.openstack.org/#/c/275616/
https://review.openstack.org/#/c/499203/

neutron-lib
https://review.openstack.org/#/c/452530/

ovs-vif
https://review.openstack.org/#/c/398277/
https://review.openstack.org/#/c/475118/

Networking-odl:
https://review.openstack.org/#/c/448536
https://review.openstack.org/#/c/534622/

puppet-vswitch
https://review.openstack.org/#/c/502440/

puppet-neutron
https://review.openstack.org/#/c/518712/

Puppet-tripleo:
https://review.openstack.org/#/c/507100/

Tripleo-heat-templates
https://review.openstack.org/#/c/493344/
https://review.openstack.org/#/c/507401/

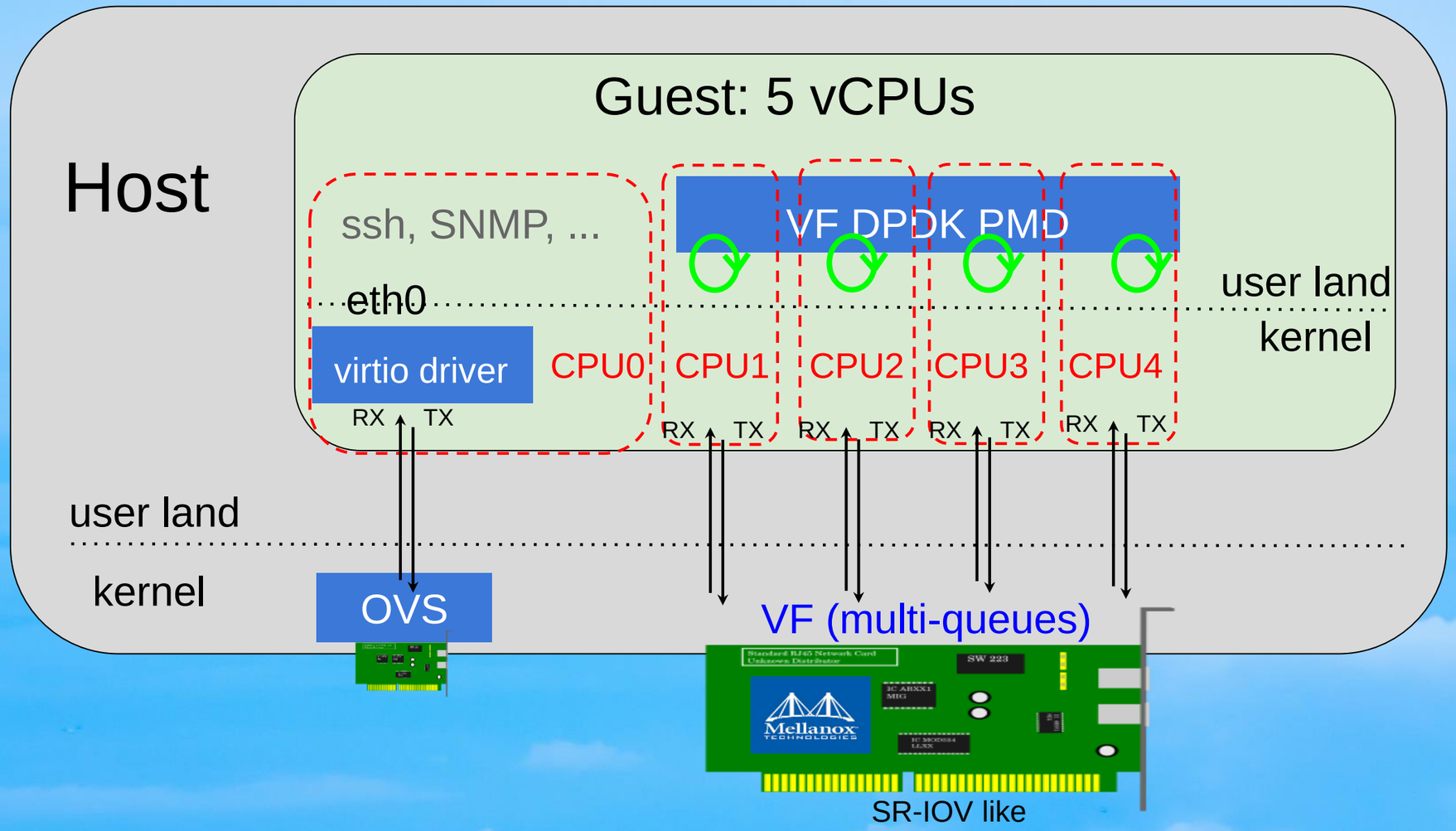
.../...

Typical VNFc on OVS HW offload: DPDK inside!

DPDK
DATA PLANE DEVELOPMENT KIT

ACTIVE LOOP

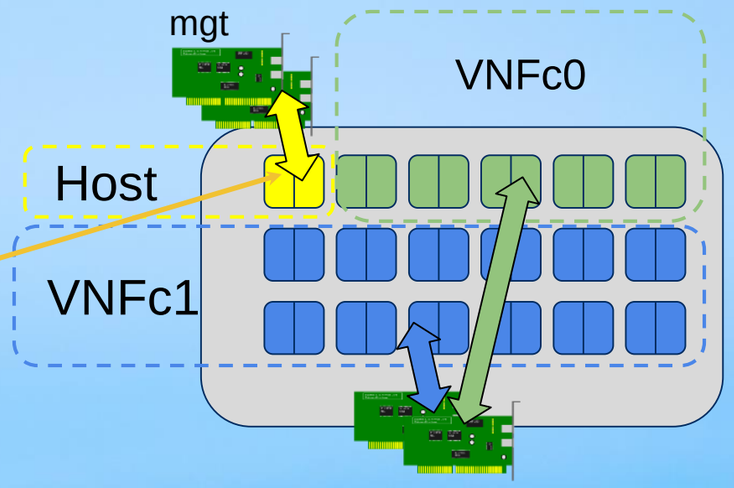
while (1) {
RX-packet()
forward-packet()
}



EPA & Partitioning TripleO parameters

Example for a dual node compute node: the first core of each NUMA socket is reserved for the host, all others are for VMs

Node0: CPUs 0-21, 44-65, first core CPUs (siblings): 0,44
Node1: CPUs 22-43, 66-87, first core CPUs (siblings): 22,66




```
ComputeKernelArgs: "intel_iommu=on default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt"

 # First core of each NUMA socket for the host, all others for the VMs

 # CPUs reserved for the VMs, nova.conf parameter

 NovaVcpuPinSet: ['1-21','23-43','45-65','67-87']

 # CPUs that won't be used by the host: tuned parameters and boot parameter isolcpus=

 HostIsolatedCoreList: '1-21,23-43,45-65,67-87' # IsolCpusList from RHOSP12 and onward
```

Same values: all cores but the host ones are available for the VMs

Deployment with TripleO

Network-environment.yaml: **like regular SR-IOV** with few additions

```
Parameter_defaults:
  NeutronMechanismDrivers:
['Opendaylight_v2','sriovnicswitch']
  NeutronSriovNumVFs: "p4p1:5:switchdev"
  NovaPCIPassthrough:
    - devname: "p4p1"
      physical_network: "north"
```

compute.yaml: **like regular OVS** kernel bridges (no changes)

Deployment: add ovs-hw-offload.yaml

```
openstack overcloud deploy .../... -e $THT_ROOT/environments/ovs-hw-offload.yaml .../...
```

Boot a VM: almost like SR-IOV

```
$ openstack port create foo --vnic-type=direct --network bar \
   --binding-profile '{"capabilities":["switchdev"]}'

$ openstack server create --flavor f1 --image i1 --nic port-id=foo vm1
```

Kernel network devices

```
$ sudo lspci | grep -ConnectX
af:00.0 Ethernet controller: Mellanox Technologies MT28800 Family [ConnectX-5 Ex]
af:00.1 Ethernet controller: Mellanox Technologies MT28800 Family [ConnectX-5 Ex]
af:00.2 Ethernet controller: Mellanox Technologies MT28800 Family [ConnectX-5 Ex Virtual Function]
af:00.6 Ethernet controller: Mellanox Technologies MT28800 Family [ConnectX-5 Ex Virtual Function]

$ ip link show p4p1
9: p4p1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP mode DEFAULT
group default qlen 1000
    link/ether ec:0d:9a:44:2c:f4 brd ff:ff:ff:ff:ff:ff
    vf 0 MAC 00:00:00:00:00:00, spoof checking off, link-state enable, trust off, query_rss off

$ ethtool -k p4p1 | grep ‘hw-tc-offload’
hw-tc-offload: on

$ ip link show p4p1VF0
23: p4p1VF0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-system state UP
mode DEFAULT group default qlen 1000
    link/ether d6:35:be:e3:11:fa brd ff:ff:ff:ff:ff:ff
```

Libvirt XML: exactly like SR-IOV

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <driver name='vfio'/>
  <source>
    <address domain='0x0000' bus='0xaf' slot='0x00' function='0x6'/>
  </source>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
</hostdev>
```

OVSDB: representor interface

Extract of: `ovs-vsctl list interface`

```
_uuid               : 7a147d47-3406-46fc-be6e-097b5d897c36
admin_state         : up
ifindex             : 23
mac_in_use          : "d6:35:be:e3:11:fa"
mtu                 : 1500
name                : "p4p1VF0"
ofport              : 2
status              : {driver_name="mlx5e_rep"}
```

TC Flower and OVS flows

```
# ovs-dpctl dump-flows -m| head -n 1 # output slightly edited for readability
in_port(1),eth(src=3c:00:00:00:00:d0,dst=3c:00:00:00:00:d1), eth_type(0x0800), ipv4(frag=no),
packets:11159099, bytes:669545940, used:0.370s, offloaded:yes, dp:tc, actions:2


# tc filter show dev p4p1 ingress filter protocol ip pref 1 flower | head -n 9
filter protocol ip pref 1 flower handle 0x1
  dst_mac 3c:00:00:00:00:d0
  src_mac 3c:00:00:00:00:d1
  eth_type ipv4
  ip_flags nofrag
  skip_sw
  in_hw
        action order 1: mirred (Egress Redirect to device p4p1VF0) stolen
        index 1 ref 1 bind 1
```

OpenStack Documentation

openstack. SEARCH SOFTWARE USERS COMMUNITY MARKETPLACE EVENTS LEARN DOCS JOIN LOG IN

https://docs.openstack.org/neutron/queens/admin/config-ovs-offload.html

OpenStack Documentation

neutron 12.0.5.dev39

Page Contents

Open vSwitch hardware offloading

UPDATED: 2018-10-31 08:00

The purpose of this page is to describe how to enable Open vSwitch hardware offloading functionality available in OpenStack (using OpenStack Networking). This functionality was first introduced in the OpenStack Pike release. This page intends to serve as a guide for how to configure OpenStack Networking and OpenStack Compute to enable Open vSwitch hardware offloading.

The basics

Open vSwitch is a production quality, multilayer virtual switch licensed under the open source Apache 2.0 license. It is designed to enable massive network automation through programmatic extension, while still supporting standard management interfaces and protocols. Open vSwitch (OVS) allows Virtual Machines (VM) to communicate with each other and with the outside world. The OVS software based solution is CPU intensive, affecting system performance and preventing fully utilizing available bandwidth.

Term	Definition
PF	Physical Function. The physical Ethernet controller that supports SR-IOV.
VF	Virtual Function. The virtual PCIe device created from a physical Ethernet controller.
Representor Port	Virtual network interface similar to SR-IOV port that represents Nova instance.
First Compute Node	OpenStack Compute Node that can host Compute instances (Virtual Machines).
Second Compute Node	OpenStack Compute Node that can host Compute instances (Virtual Machines).

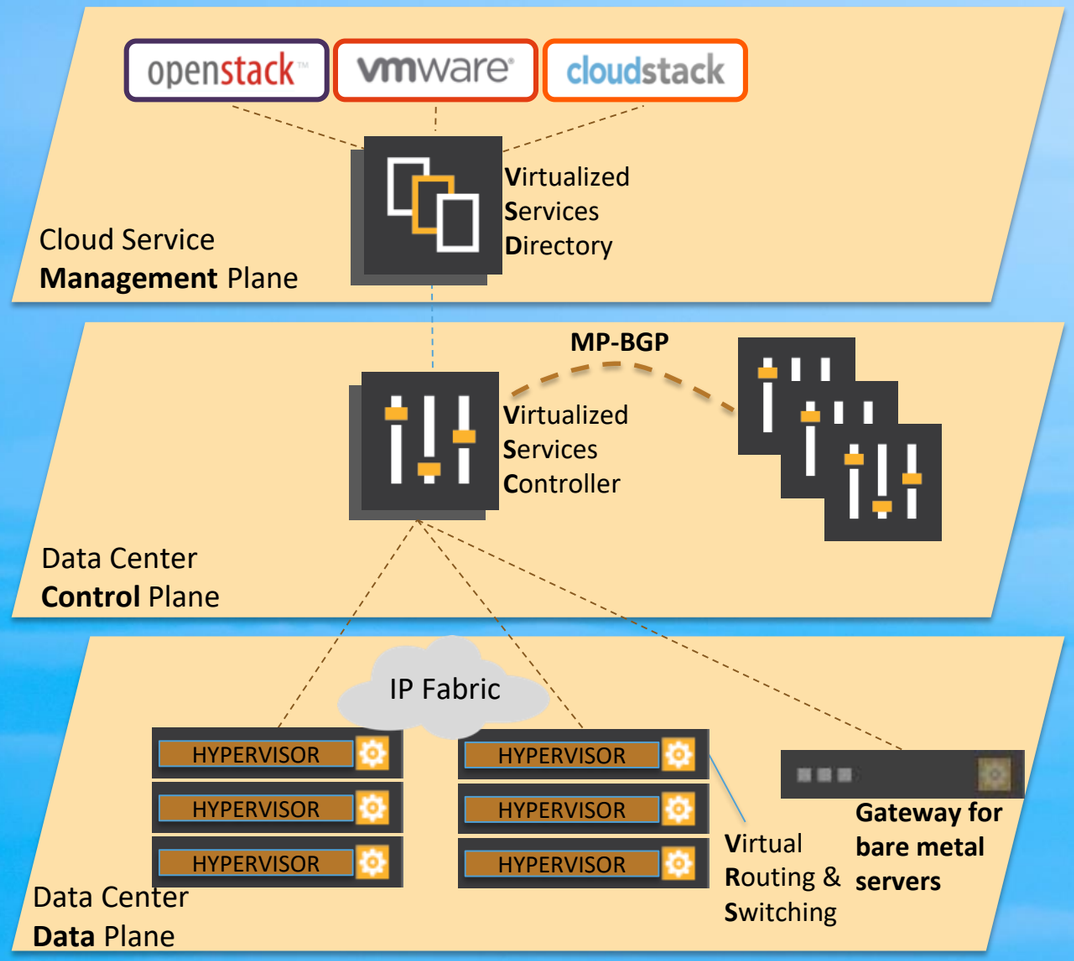
Supported Ethernet controllers

The following manufacturers are known to work:

- Mellanox ConnectX-4 NIC (VLAN Offload)
- Mellanox ConnectX-4 Lx/ConnectX-5 NICs (VLAN/VXLAN Offload)

Nuage Networks Virtualized Cloud Services (VCS) 5.4.1

SmartNIC Accelerated Virtual Networking



Nuage Networks
Virtualized Services Platform (VSP)

Virtualized Services Directory (VSD)
- Network Policy Engine – abstracts complexity
- Service templates and analytics

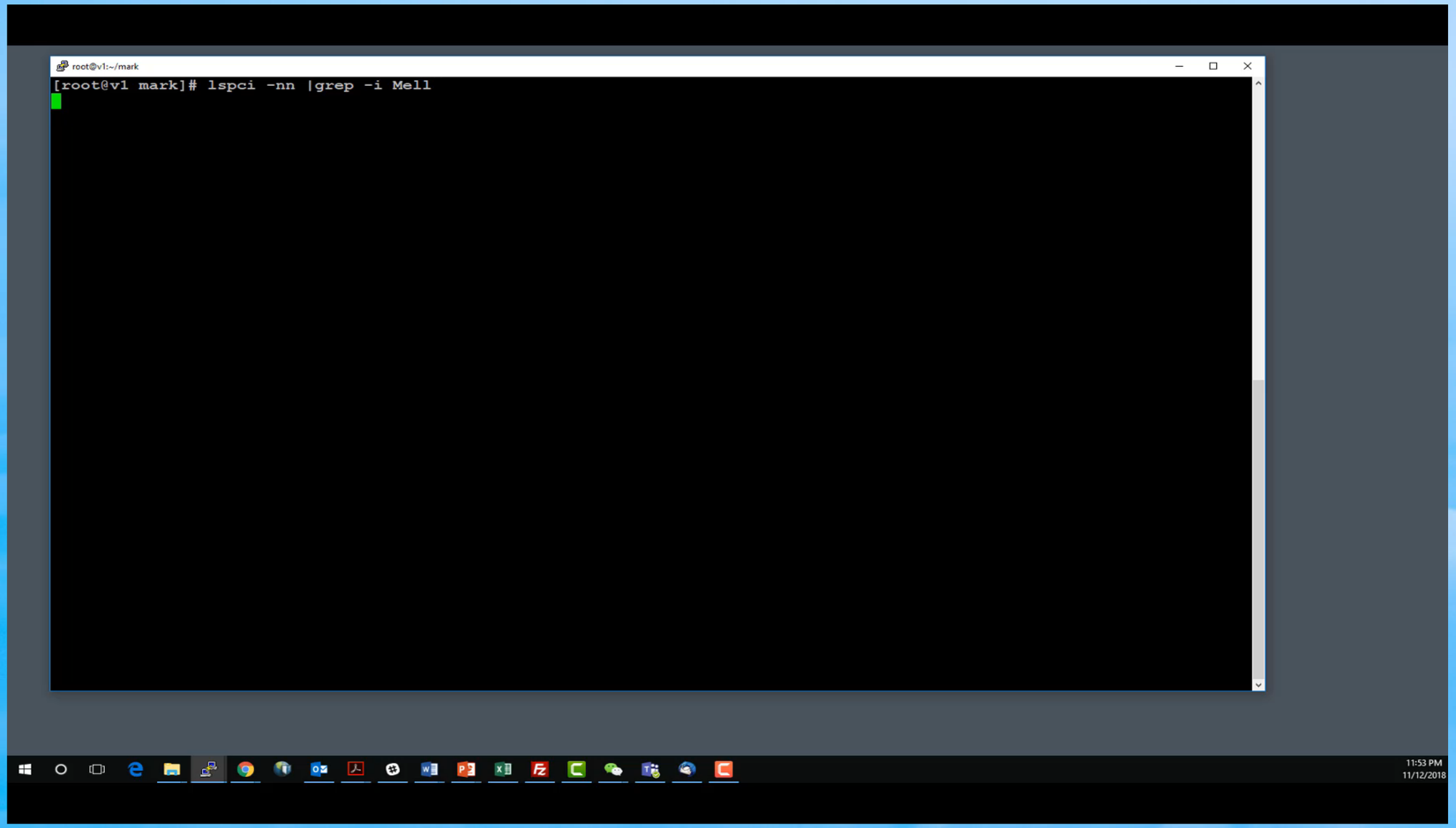
Virtualized Services Controller (VSC)
- SDN Controller, programs the network
- Rich routing feature set

Virtual Routing & Switching (VRS-TC)
- Distributed switch / router – L2-4 rules
- Integration of bare metal assets

NOKIA — PROPRIETARY AND CONFIDENTIAL
COPYRIGHT © 2016 NOKIA. ALL RIGHTS RESERVED

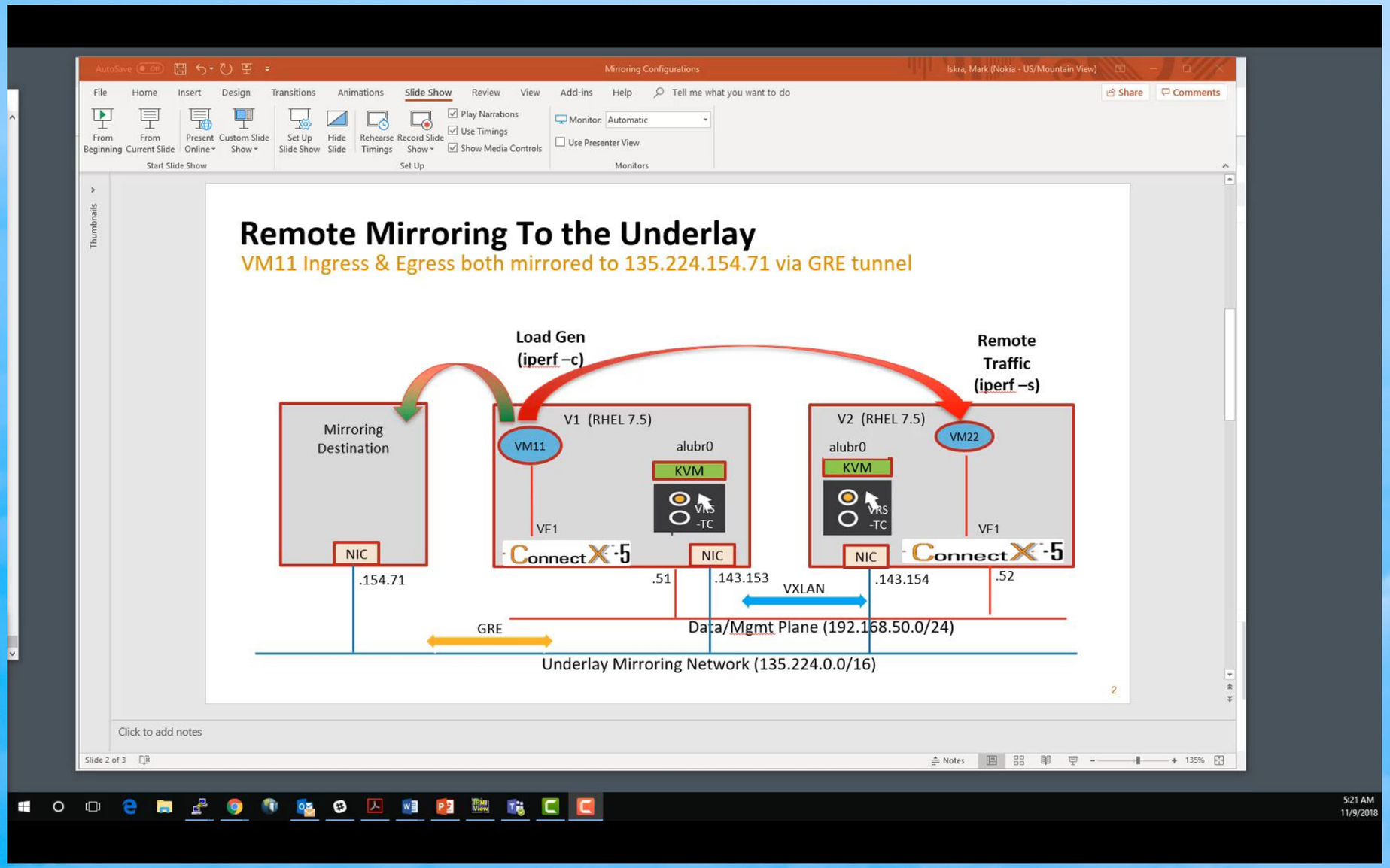
Demo: OpenStack Queens OVS Offload Integration

Nuage Networks VCS leveraging tc-flower for hardware flow offloading



Demo: Offloading Remote Mirroring to GRE Underlay

Nuage Networks VCS leveraging tc-flower for hardware flow offloading



Demo: Offloading Remote Mirroring to VXLAN Overlay

Nuage Networks VCS leveraging tc-flower for hardware flow offloading

Fully Open Source Smart OpenStack Cloud: NOW and BEYOND

NOW:

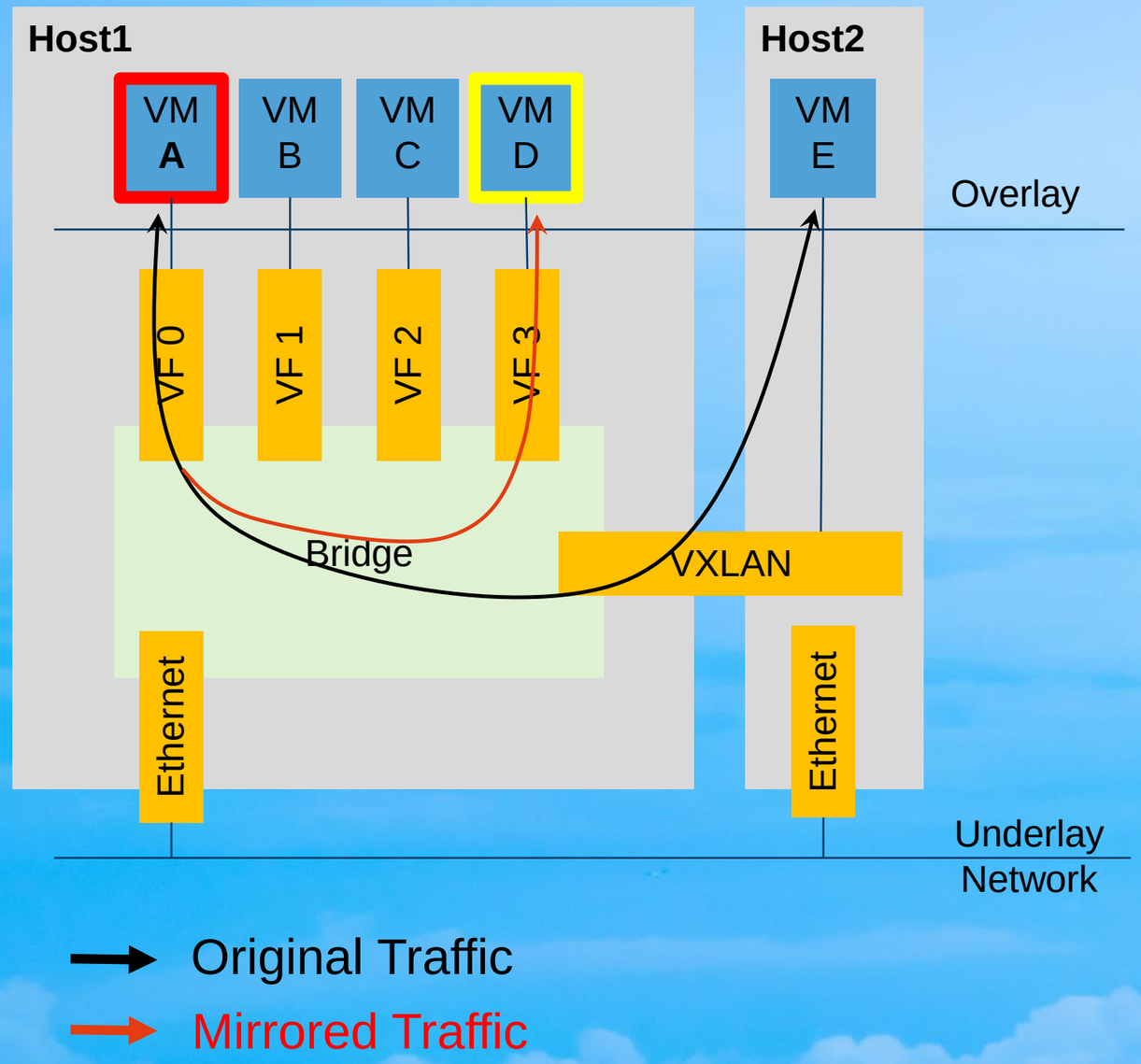
- State of Telco Cloud and need for offloads
- What are SmartNICs and how offloads work on SmartNICs ?
- What are the open source elements to enable offloads ? Blueprints, configuration tips/guides, etc.
- How is this all deployed ? Recorded demo!

BEYOND:

- What’s coming next ? Port Mirroring, VFLAG, QoS, Connection Tracking, Commercial VNF integration

Port Mirroring

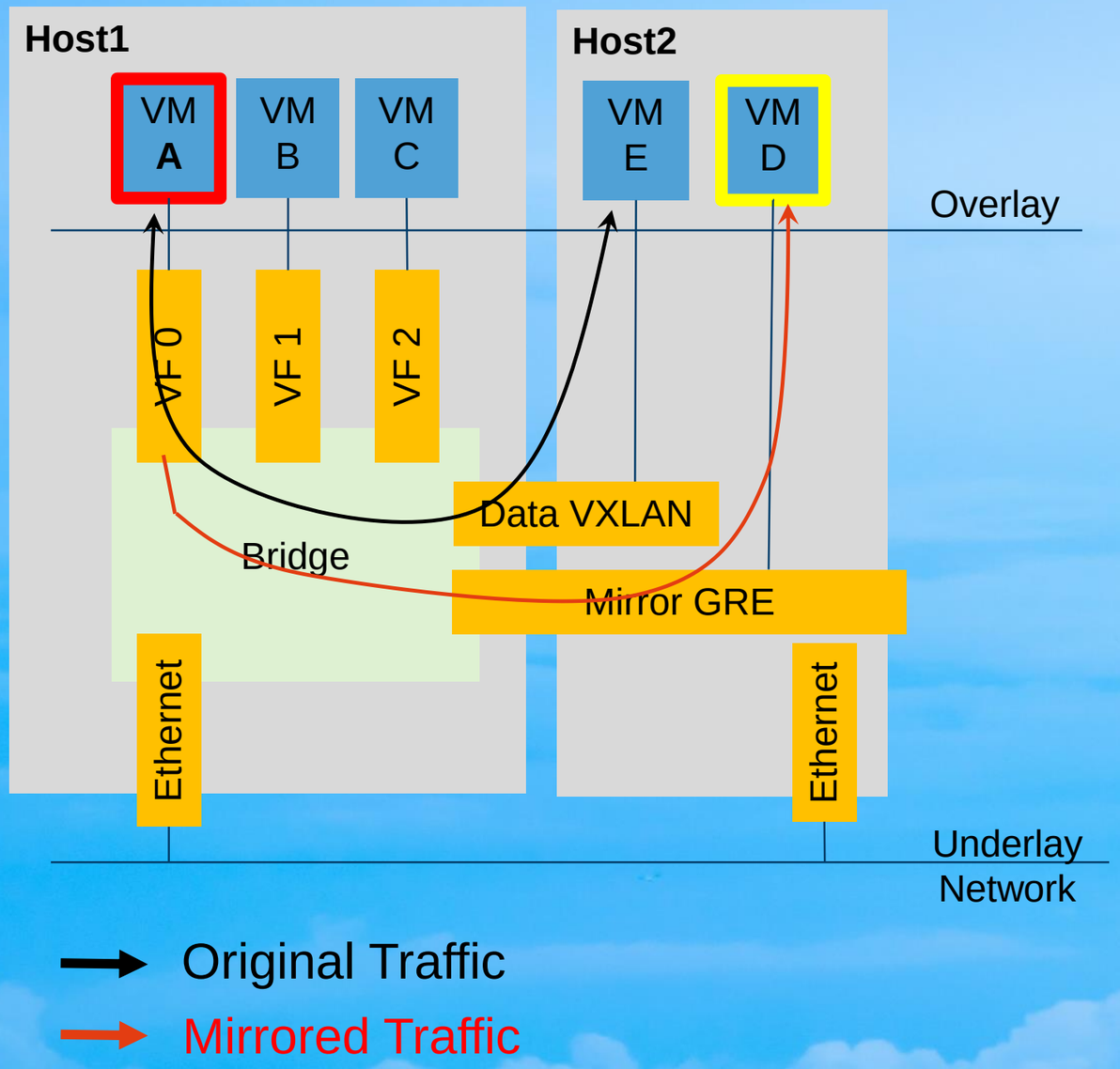
Local Mirroring Overview



- **Port Mirroring –All Traffic**
  - VF 0 is the probed interface
  - VF 3 is the mirror port
  - All VF0's traffic is mirrored to VF 3
  - Mirrored Traffic
    - Without VXLAN (Local Traffic)

- **Classification based Mirroring**
  - On VF0
  - Only specific flows are mirrored
  - Example – specific TCP flows
    - Classified by 5-Tupple
    - Ingress or Egress or both traffics

Remote Port Mirroring Overview



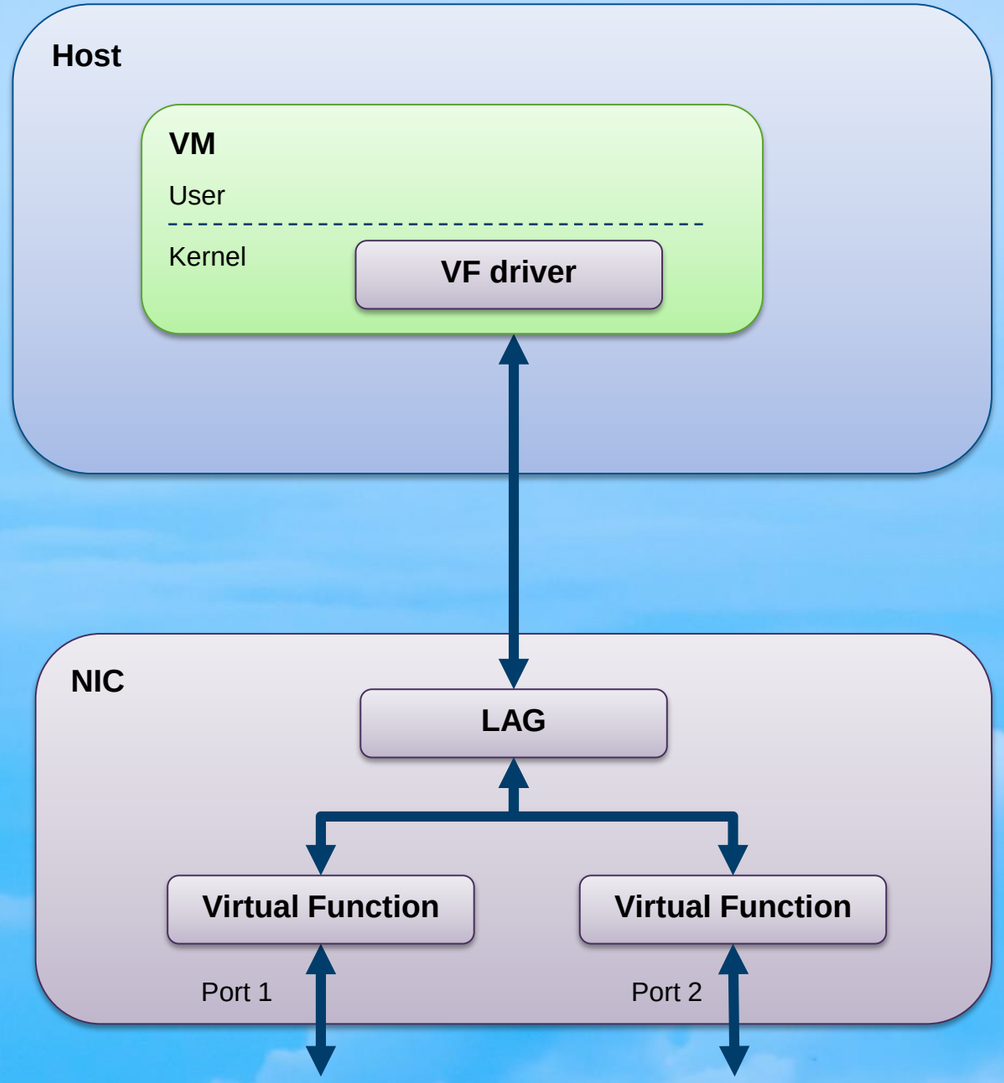
- **Port Mirroring –All Traffic**
  - VF 0 is the probed interface
  - VM D is a remote host, sees all the traffic as VM A sees it
  - Mirrored traffic to remote host is tunneled over GRE (can be VXLAN)

- **Classification based mirroring**
  - On VF0
  - Only specific flows are mirrored
  - Example – specific TCP flows
    - Classified by 5-Tupple
  - Ingress or egress traffic or both

High Availability

High Availability - VF LAG for OVS Hardware Offloads

- SR-IOV Guest VMs don't support bonding/HA natively with OpenStack
- Mellanox enable transparent SR-IOV VF HA between ports of a single NIC
- LAG will be implemented on Mellanox NIC so VM will only see a single Virtual Function (VF)
- **Modes supported**
  - Active Passive (Single port BW)
  - Active Active (Double port BW)
  - LACP



Quality of Service (QoS)

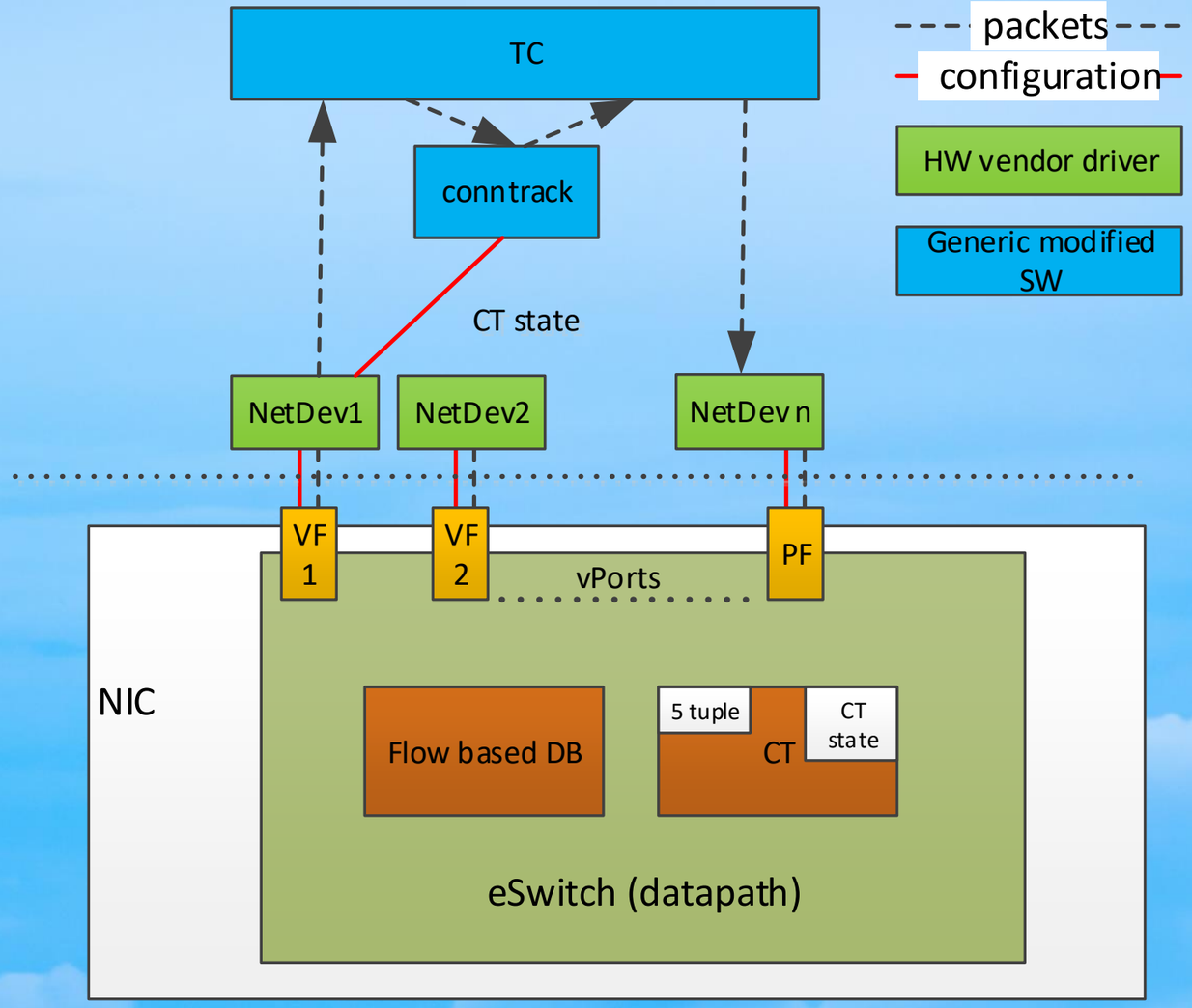
QoS for OVS Hardware Offloads

- Offload will use TC API
- Bandwidth Limit Options
  - Rate limit per VF (max limit)
  - BW guarantee per VF (min limit)
- DSCP Marking
  - Force DSCP per VF
  - Overlay (vxlan)
    - Set DSCP of tunnel packet
    - Copy the DSCP of the inner packet
- Changes needed in TC and Driver

Connection Tracking

Connection Tracking in Hardware concept

- Stateless ACLs are Supported today
- The TCP handshake is going to the OVS software conntrack
- Once the connection is established, TC adds the connection to the HW
- The following packets of the TCP stream are forwarded directly to the VF by the HW



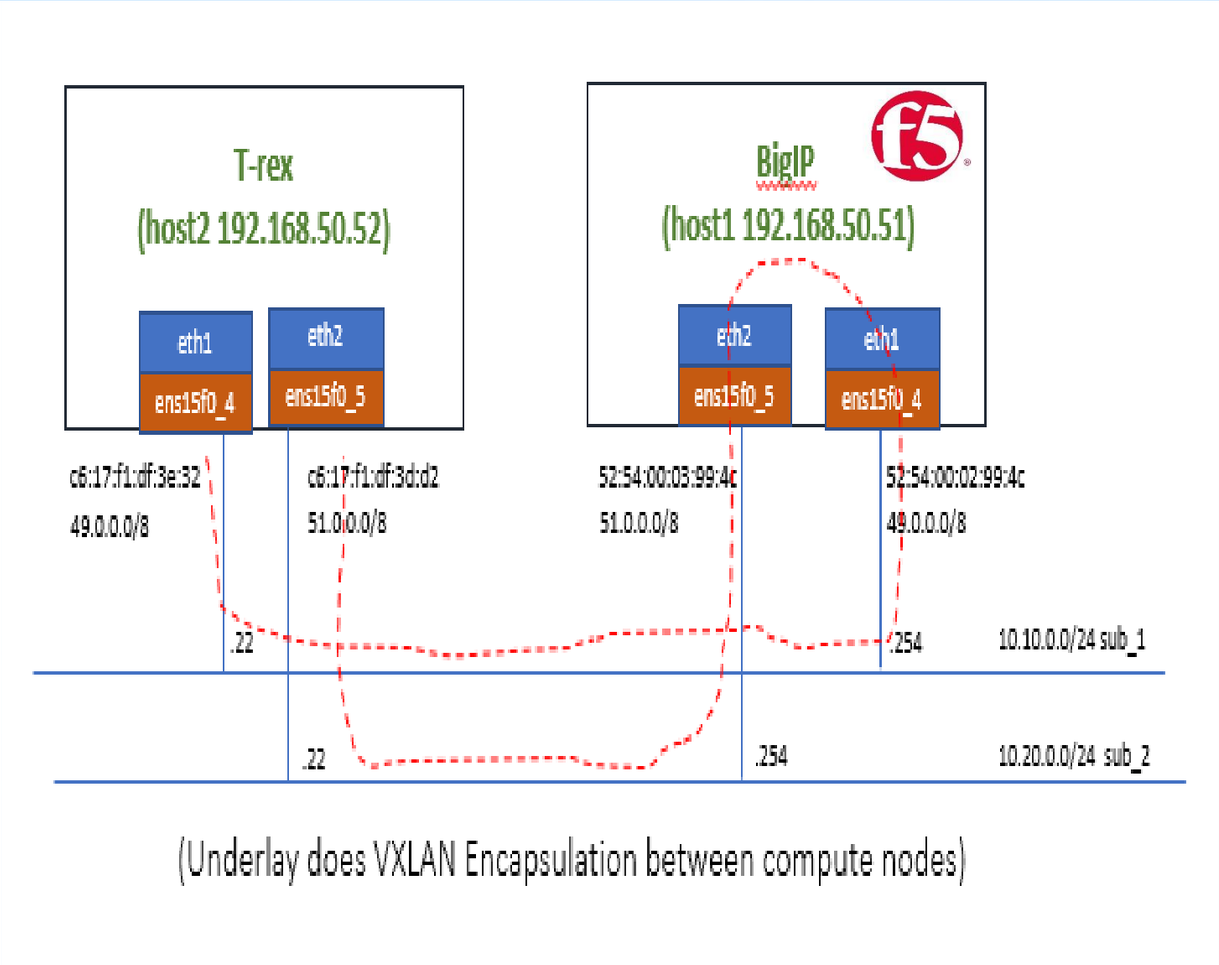
Connection Tracking – OVS Hardware Offload

- ConnectX-5
  - Hardware support match on 5 tuple
  - Hardware support match on TCP flags
  - Hardware Counter per flow for activity/aging

- Bluefield
  - TCP handshake is done by ARM software

Commercial VNF using OVS Offload: F5 Big-IP CGNAT

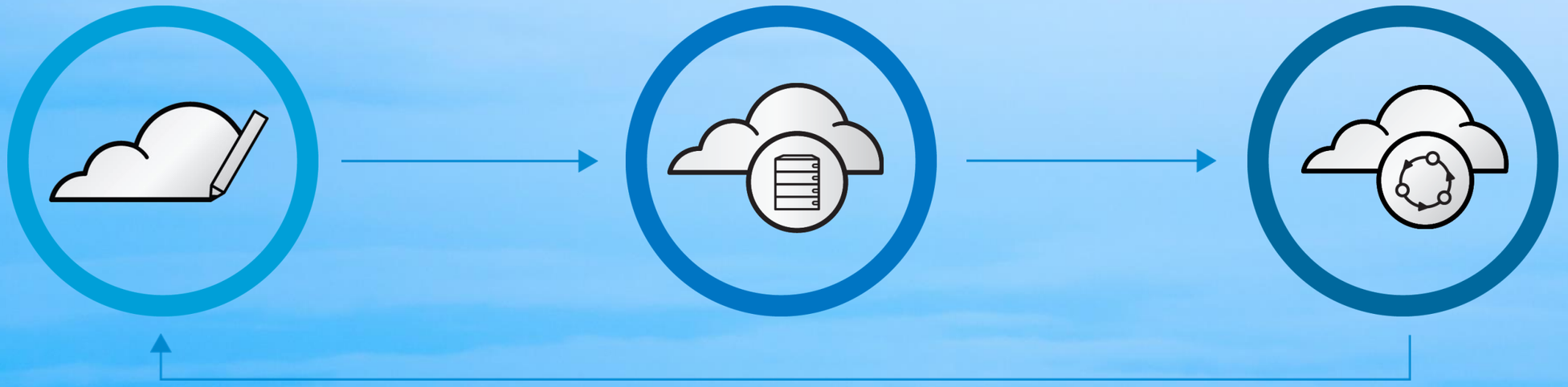
- **Use Case:** F5 BIG-IP CGNAT (NAT & PAT)
- Offloads without issues
- Compute hosts have zero resources used for VNF virtual networking
- Overlay CG-NAT forwarding is only a fraction of BigIP's capabilities, but a useful proof point for OVS hardware offload validation
- Performance: **70 Gbps** for IMIX traffic
  - Truly Carrier Grade



Thank You

?Questions?

OpenStack installation and orchestration: TripleO



PLANNING

- Network topology
- Service parameters
- Resource capacity

DEPLOYMENT

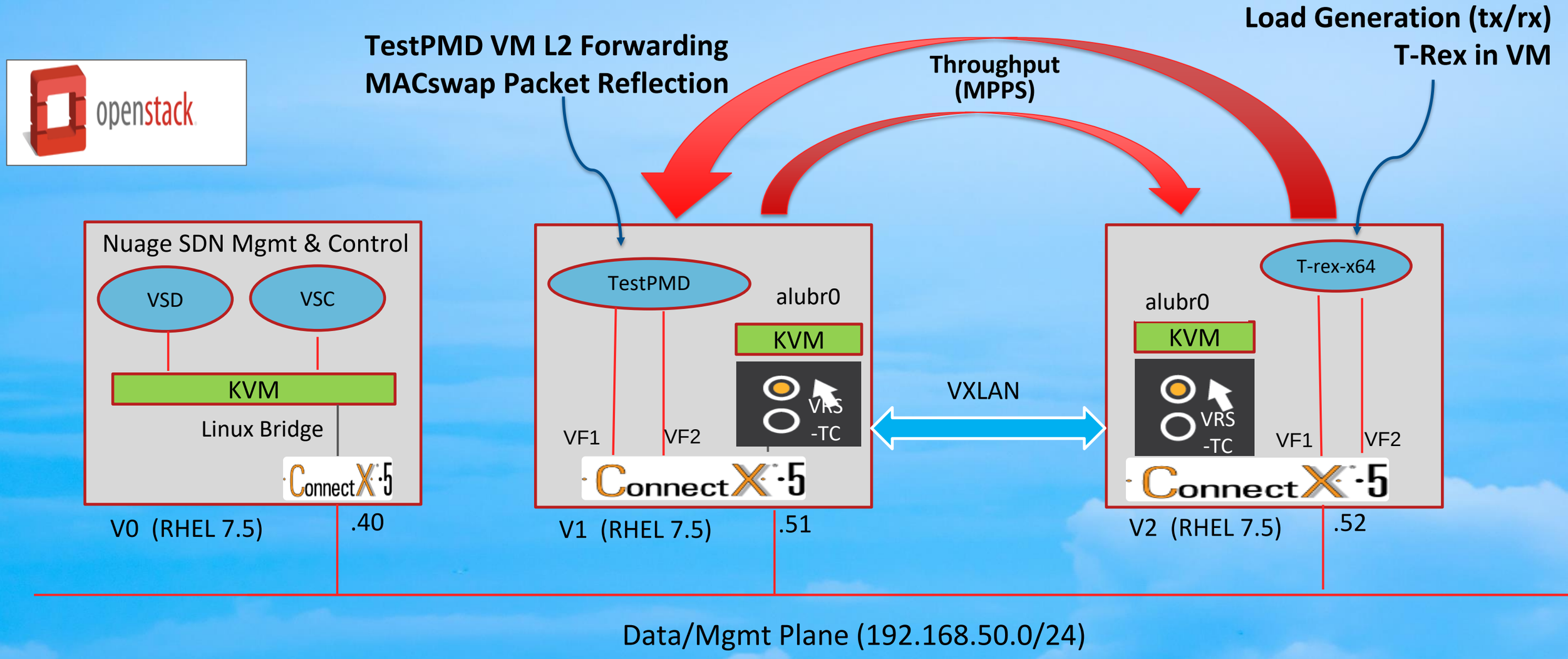
- Deployment orchestration
- Service configuration
- Sanity checks

OPERATIONS

- Updates and upgrades
- Scaling up and down
- Change management

Use Case: VXLAN Throughput – T-Rex & TestPMD

DPDK Applications generate and forward packets over Nuage manage VXLAN tunnel

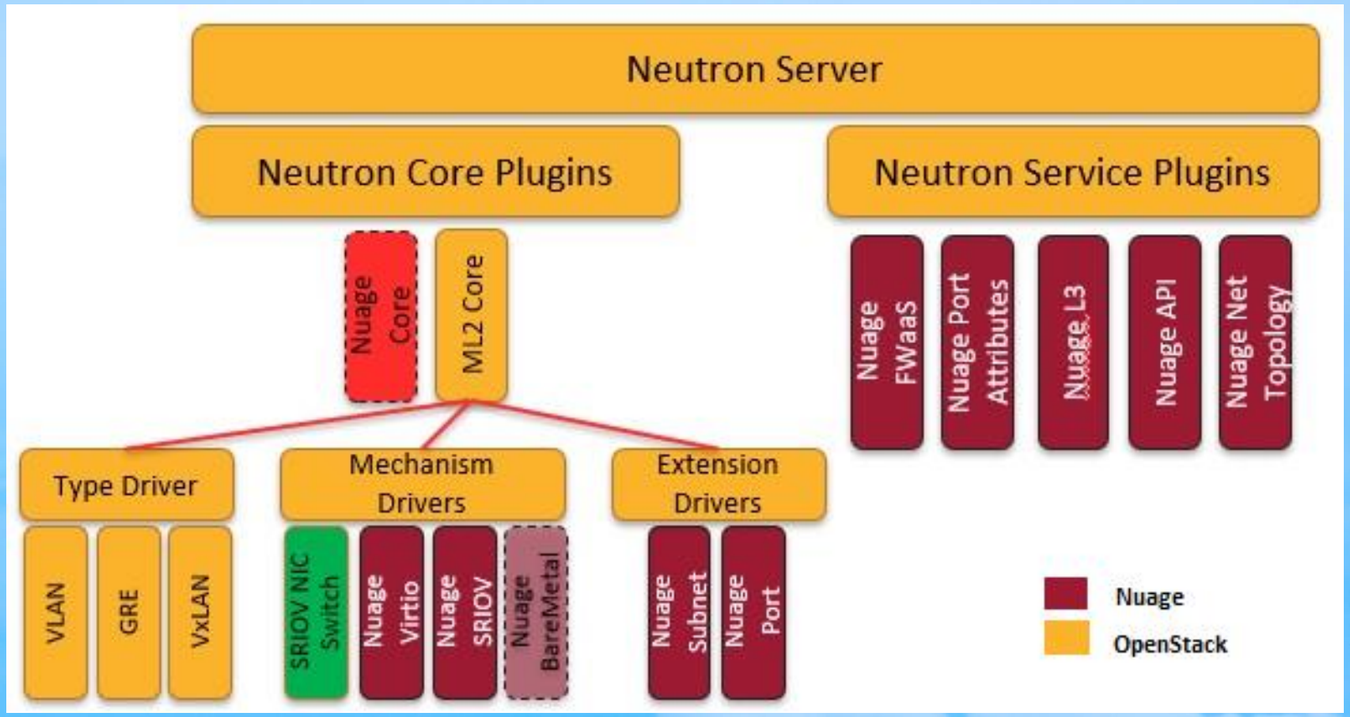


Nuage Networks VCS 5.4.1 OpenStack Integration

Nuage Networks ML2 Plugin Supporting OSPD 13 and beyond

OpenStack “Switch on NIC” deployment steps:

- Nova SR-IOV PCIPassThrough with VF representor to OVS
- `# openstack port create --network NorthSouthData --vnic-type=direct --binding-profile '{"capabilities": ["switchdev"]}' sr-iov_port1`
- assign port to guest instance vm1
- `# openstack server create --flavor m1.small --image rhel75 --nic port-id=sr-iov_port1 vm1`



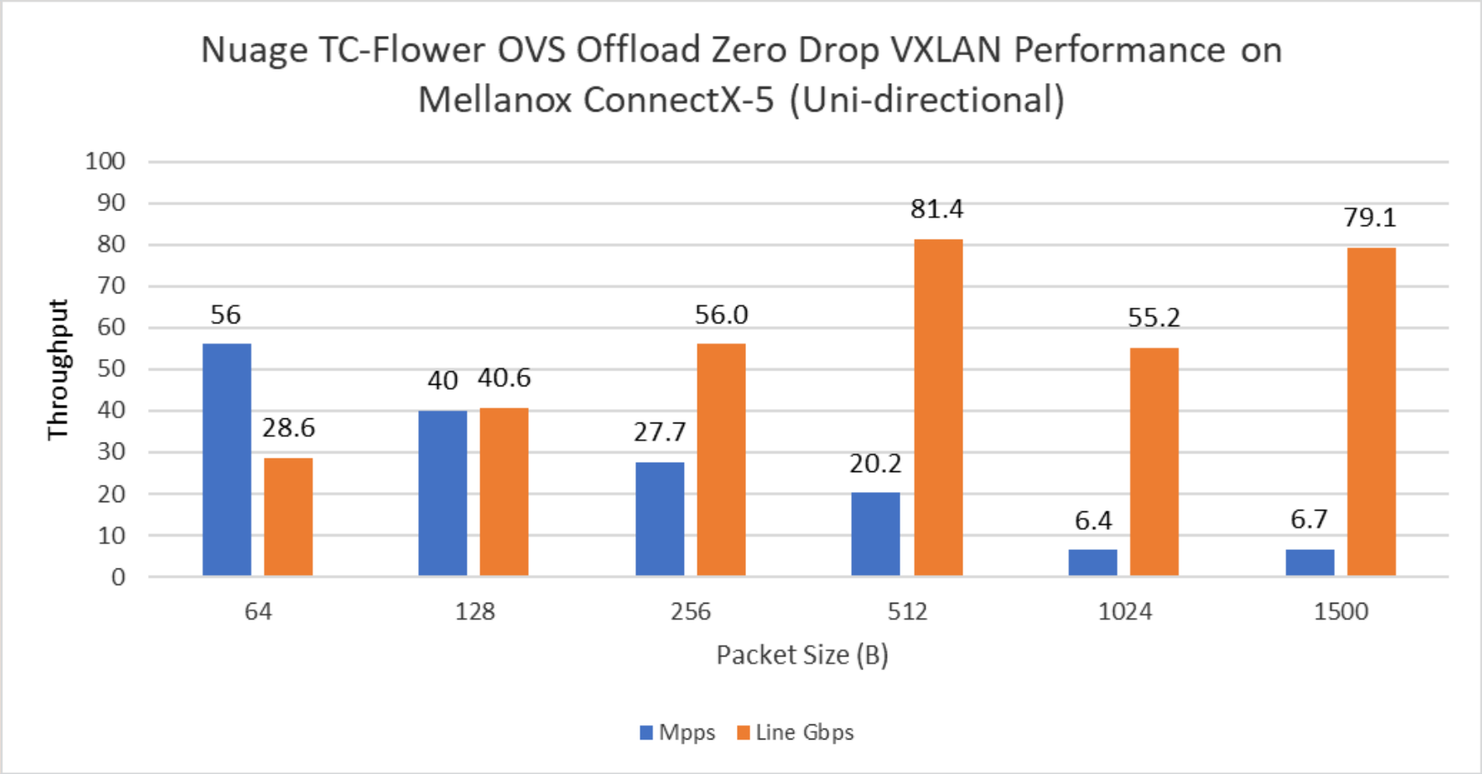
Reference: https://docs.openstack.org/neutron/queens/admin/config-ovs-offload.html

Nuage Networks VSP on ConnectX-5 Offload Performance

OVS Flows are programmed via tc-flower interface

Results:

- Zero CPU usage for VXLAN tunnels
- Uni-directional measurements (multiple by 2 for bi-directional)
- No packet loss in forwarding app
- T-Rex and TestPMD run in VMs
- 2 active tunneling flows



Larger packet throughput is limited by T-Rex load generation

System Specs:

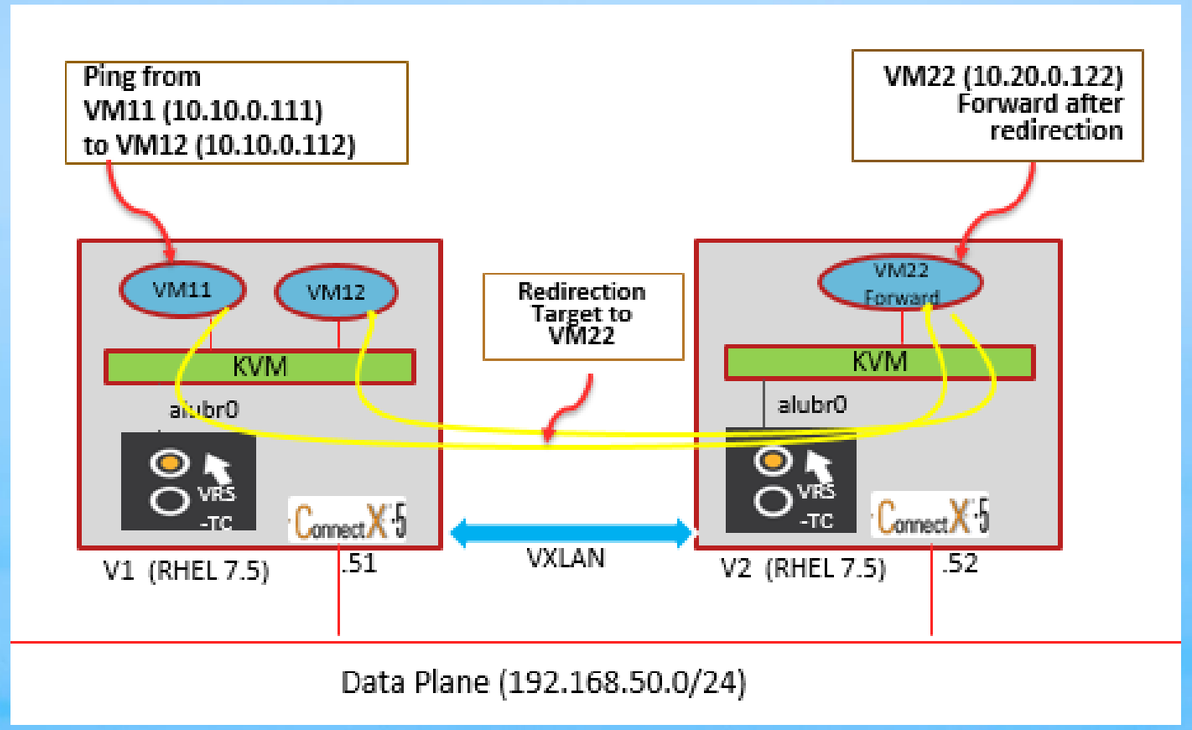
- E5-2667 v3 @ 3.20GHz
- Mellanox ConnectX-5 NIC (100Gbps)
- RHEL 7.5 Host and Guest
- Mellanox SN2100 Fabric Switch

Use Case: Service Function Chaining (SFC)

OVS Flows are programmed via tc-flower interface

Results:

- Hardware offloading works!
- Forwarding done in kernel on VM22 which limits throughput
- iperf can easily drive 20-30Gbps, DPDK apps needed for higher performance
- 2 pair of flows created: vm11→vm22 and VM22->VM12 (NIC switch flows dumped below)



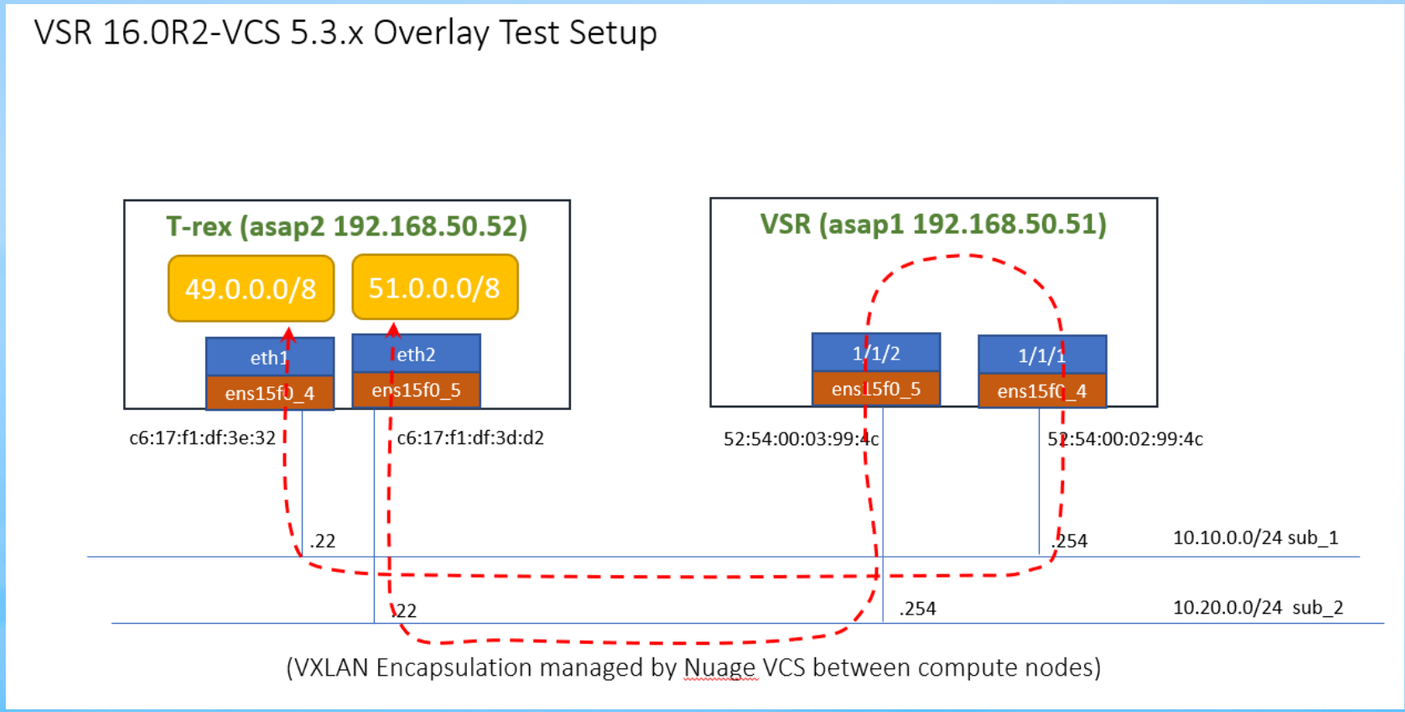

```
[root@vrs-tc-1 test2]# ./getNICflows.sh
in_port(vport1),eth(dst=8e:86:3f:aa:b0:cd),eth_type(0x0800),ipv4(src=0.0.0.0/1.0.0.0,dst=0.0.0.0/0.0.0.1,proto=6), counter:0x29, action:set(tunnel(encap_id=0x0)),uplink
tunnel(tun_id=0x8fadef,eth(dst=24:8a:07:9c:11:7e),ipv4(src=192.168.50.52,dst=192.168.50.51,proto=17,frag=no),tp_dst=4789),in_port(uplink),eth(src=68:54:ed:00:2b:5c,dst=8e:86:3f:aa:b0:cd),eth_type(0x0800),ipv4(src=10.10.0.111,dst=10.10.0.112), counter:0x2a, action:vport2
tunnel(tun_id=0x8fadef,eth(dst=24:8a:07:9c:11:7e),ipv4(src=192.168.50.52,dst=192.168.50.51,proto=17,frag=no),tp_dst=4789),in_port(uplink),eth(src=68:54:ed:00:2b:5c,dst=c6:17:f1:df:4e:d2),eth_type(0x0800),ipv4(src=10.10.0.112,dst=10.10.0.111), counter:0x2c, action:vport1
in_port(vport2),eth(dst=c6:17:f1:df:4e:d2),eth_type(0x0800),ipv4(src=0.0.0.0/1.0.0.0,dst=0.0.0.2/0.0.0.2,proto=6), counter:0x2b, action:set(tunnel(encap_id=0x0)),uplink
```

Use Case: Nokia VSR IP Forwarding Test

Test and Commercial VNF (VxWorks based)

Results:

- Offloads without issues
- IP forwarding is only a tiny fraction of VSR capabilities, but a useful proof point for driver validation
- VSR is a building block for more complex VNFs
- Low end (2GHz) hosts: ~1Mpps @128B no loss throughput, ~2K new flows/sec



[root@vrs-tc-1 sfc_testing]# ip l |grep enp130s0f0
Tb added