[The Long Road to] Deployable OVS Hardware Offloading for 5G Telco Clouds

OVS Conference - 11th December 2019
Westford

Mark Iskra
Marketing Director
nuagenetworks
From Nokia

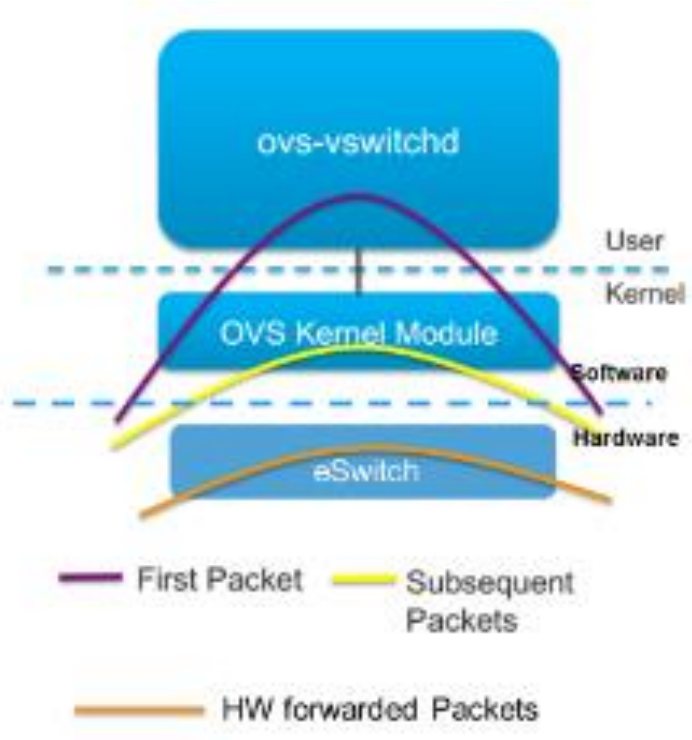
Majd Dibbiny
Software Director
Mellanox
TECHNOLOGIES

Anita Tragler
Product Manager
Red Hat

HW OVS Offloading Evolution

- 2016 Vision: what is a made for purpose processor could handle OvS flows?
- That vision has matured and required the support of many communities including OVS, Linux, OpenStack, etc..

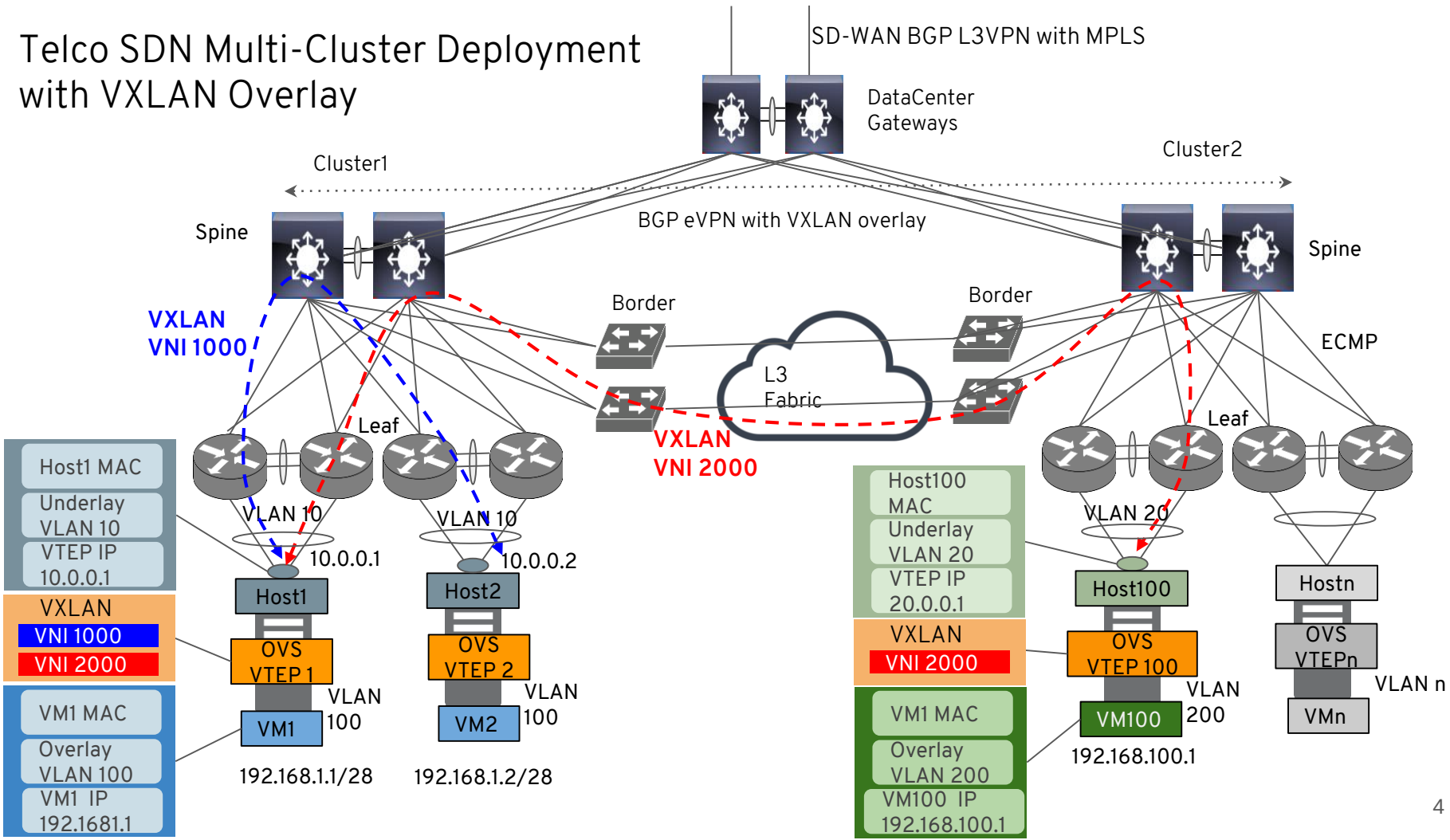
2016 Vision:



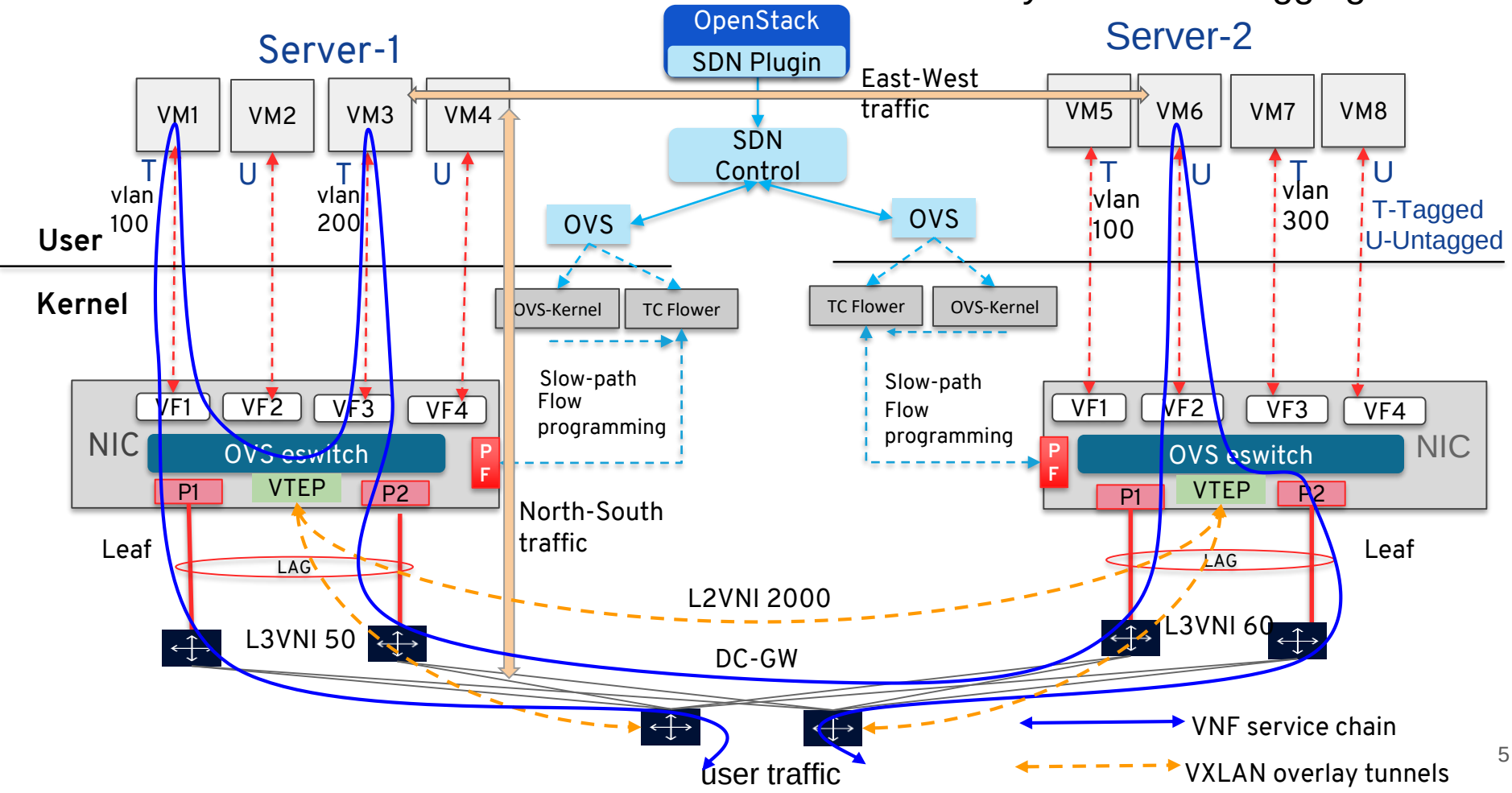
Telco Cloud: Common Network Requirements

- Virtual networking -- VxLAN
- Resilience through redundancy
- Bonded Uplinks with LACP
- VLAN-Trunking to hosted VNFs
- BGP for route discovery between leaves
- Mirrored traffic flows
- Performance:
  - Small packets at high rates and throughput with rate limiting
  - Rapid flow creation and eviction: churn
- Scale: especially for flows (>10K per sec)
- Cost and power efficiency
- OpenStack and K8s integration

Telco SDN Multi-Cluster Deployment with VXLAN Overlay



North South and East West traffic with VXLAN overlay and VLAN tagging



Bonding Offload - VF-LAG

VF LAG Overview

- Requirements: HW offloads uses SR-IOV, therefore there’s a need to offload the Linux Bond functionality:
  - Link aggregation: Combining number of physical ports (IEEE 802.11ad)
  - High availability: Supporting Link failover
- Expectations:
  - Single VF aggregating both physical ports of the same NIC for RX and TX
  - VF associated with either PF sends data over either physical port
  - Transparent to the VM
  - Initialized by creating a bond device that enslaves both ports of the NIC

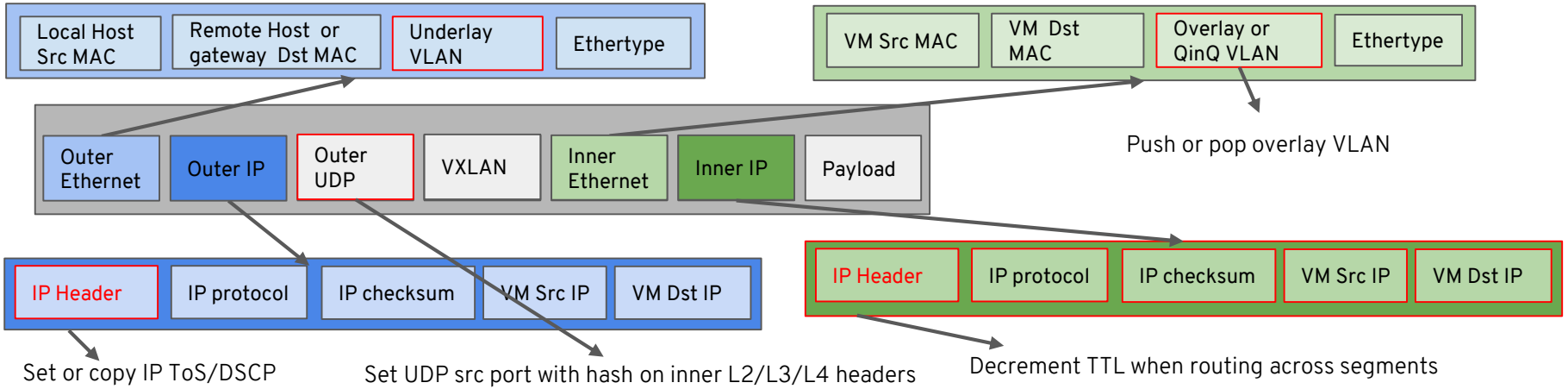
VF LAG HW Offload

- In LAG mode flow rules are offloaded to the FDB of both e-switches for high availability
  - For uplink representors, Shared TC Block is used to replicate the rules on both e-switches.
  - For representors, the driver is responsible to replicate the rules to both e-switches.
- The driver registers to netdev events to set the affinity of live ports
- Send queues’ affinity is set in a round-robin fashion to both physical ports

VLAN Tagging and Header Rewrite

VLAN Tagging and IP Header rewrite

- Overlay tagging:
  - Pushing VLAN tag for VM traffic
  - Popping VLAN tag for VM traffic
- Underlay tagging:
  - Mainly used to ensure segregation of different type of traffic (Control plane, User plane, Monitoring..)
- Header rewrites:
  - TTL decrement for routing across subnets
  - Copy TOS from Inner
  - UDP source port: Hash of inner headers to provide entropy for ECMP

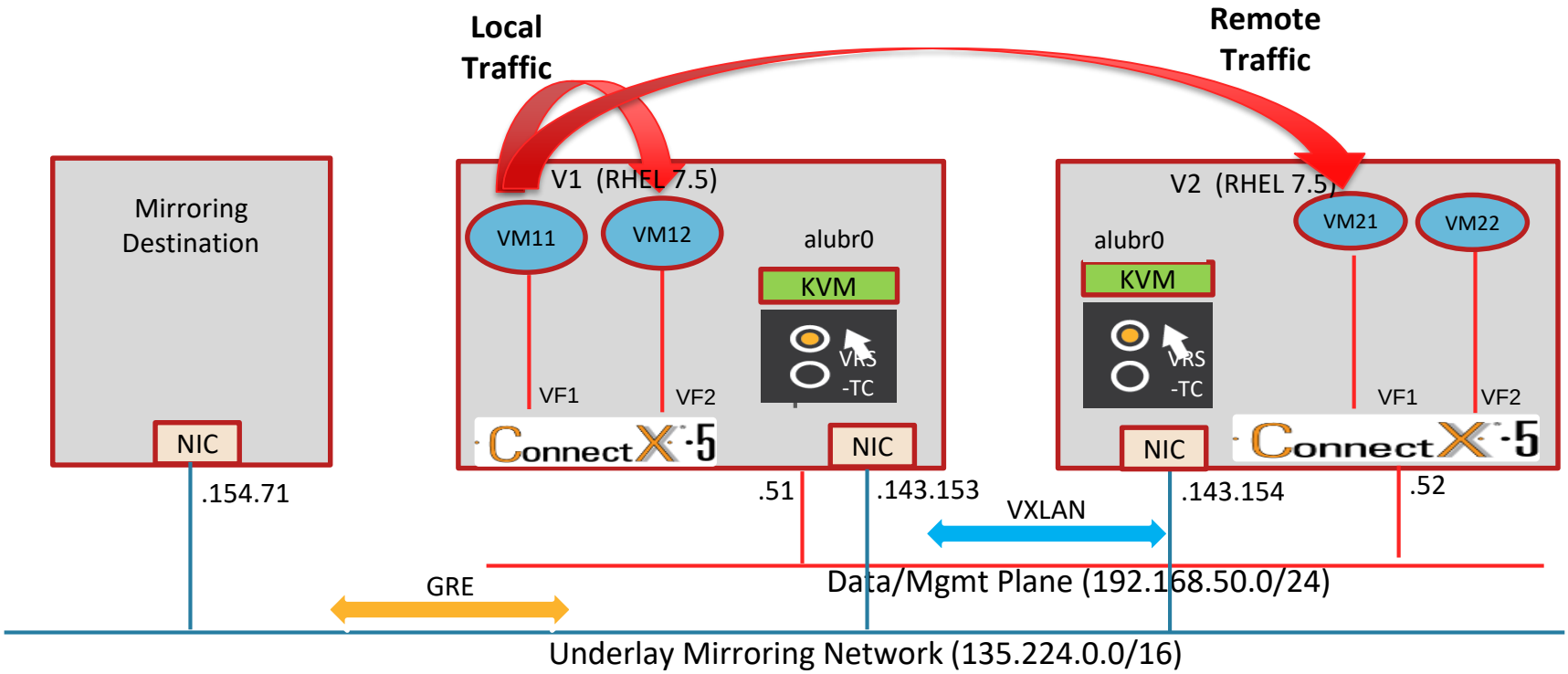


Remote Mirroring Overview

- Duplicating the traffic of one VF to a remote VF/Host
- Use cases:
  - Debuggability
  - Lawful intercepts
  - Monitoring
  - Other..

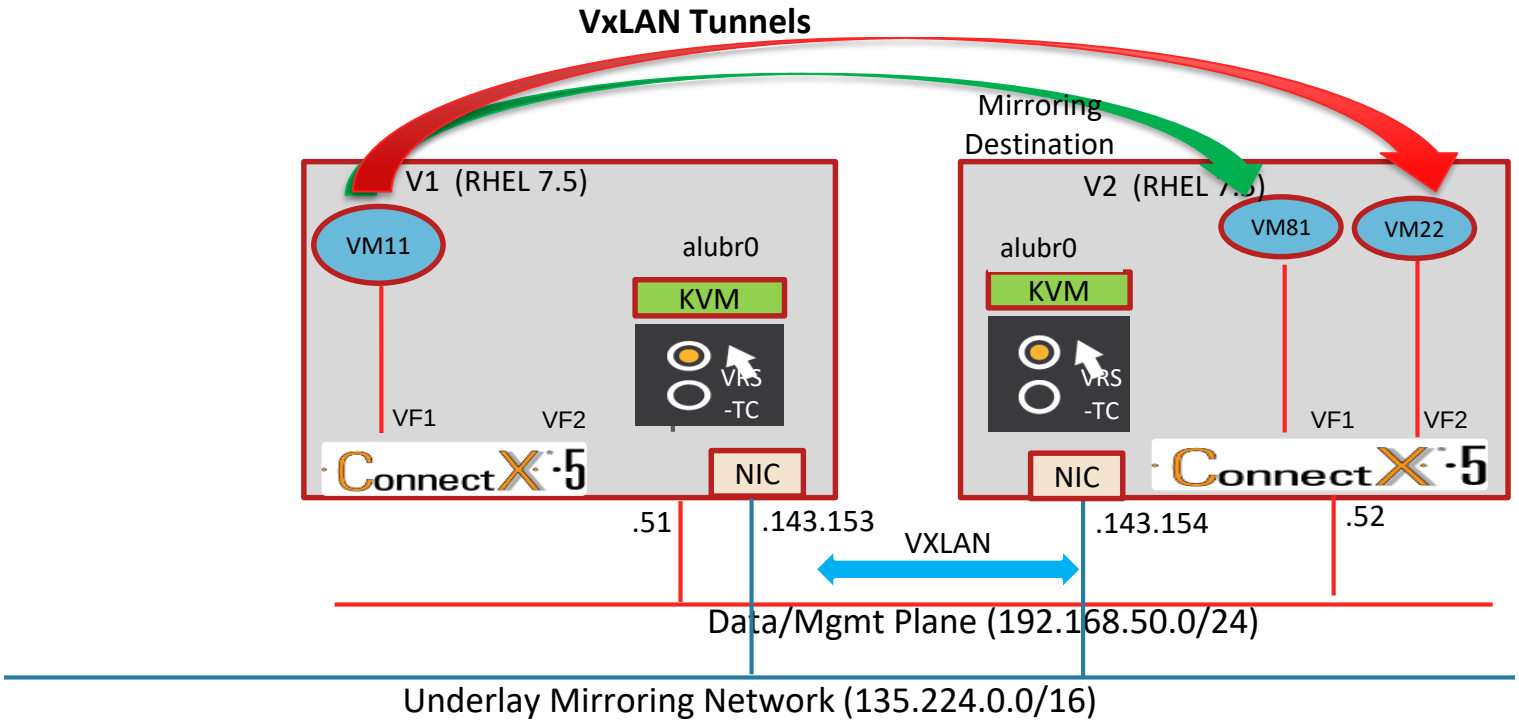
Remote Mirroring Configurations

Sample configurations for local and remote traffic



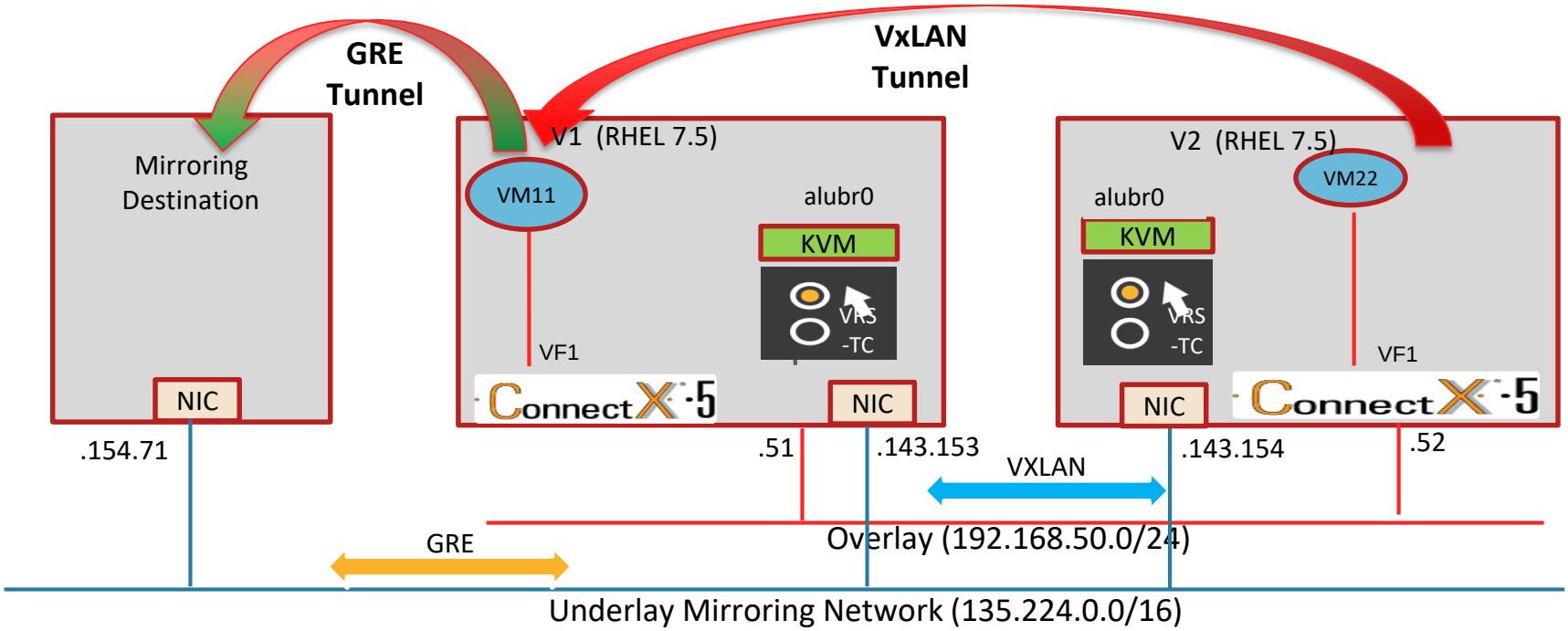
Remote Mirroring to the Overlay

Sample configurations for local and remote traffic



Remote Mirroring To the Underlay

VM11 Ingress & Egress both mirrored to 135.224.154.71 via GRE tunnel



VxLAN/GRE Offloaded Mirrored Flow Example

Decap Ingress and GRE Mirror to underlay (135.227.154.71)

```
[root@asap1 ~]# ovs-dpctl dump-flows -m
ufid:bc7c9de6-5042-42e3-8053-1171da6e4c90,
skb_priority(0/0),tunnel(tun_id=0xcb218d,src=192.168.50.52,dst=192.168.50.51,ttl=64,flags(+key)),sk
b_mark(0/0),in_port(vxlan_sys_4789),eth(src=00:00:00:00:00:00/00:00:00:00:00:00,dst=c6:17:f1:df:4e:
d2),eth_type(0x0800),ipv4(src=0.0.0.0/0.0.0.0,dst=10.10.0.111,proto=6,tos=0/0x3,ttl=0/0),tcp(src=40
96/0xf000,dst=0/0), packets:547607, bytes:63522412, used:0.480s, offloaded:yes, dp:tc,
actions:set(tunnel(tun_id=0x0,dst=135.227.154.71,tp_dst=0,flags(key))),gre_sys,ens15f0_0
```

Looking ahead for OVS TC/flower HW offload

OVS TC/flower Flow Offload

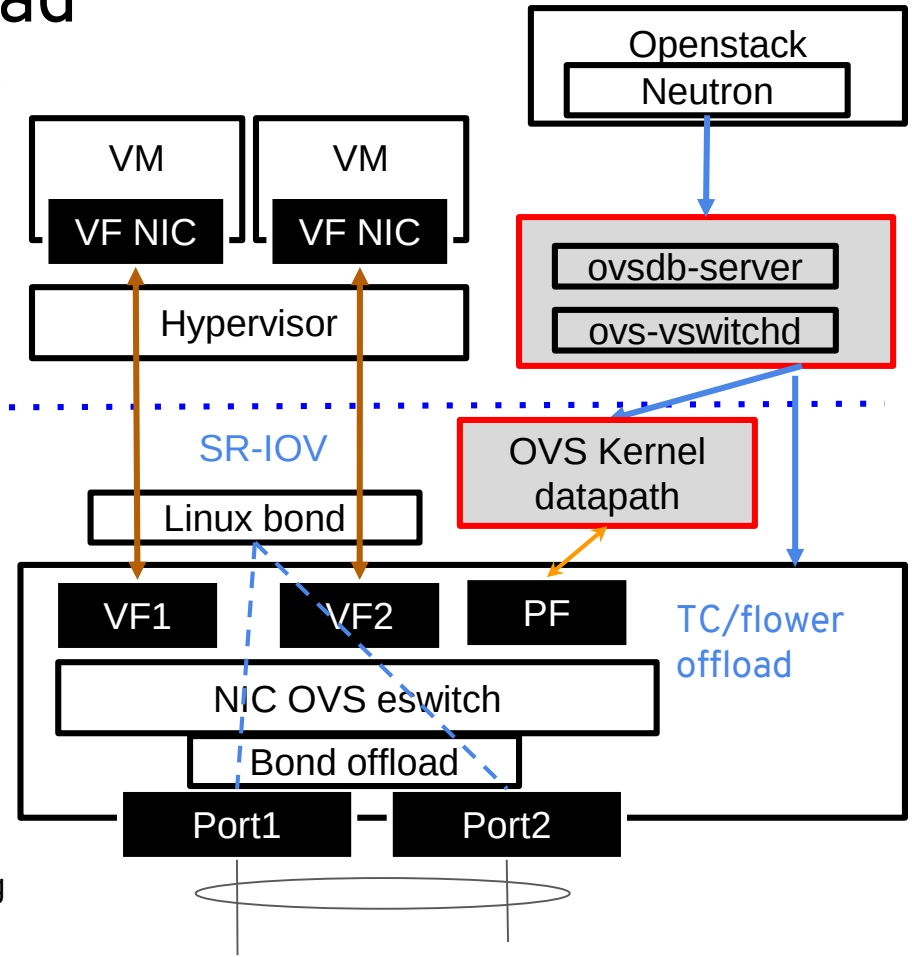
OVS 2.11 with RHEL 7.7, Mellanox ConnectX-5 (mlx5)

Current Features

- Line rate throughput 25Gbps, 50Gbps
- Flow Match - 5 Tuple, IPv4, IPv6, MAC, TCP/UDP port, protocol, VLAN, VXLAN
- Action - set QoS ToS/DSCP, set TTL
- Overlay encap - VLAN, QinQ, GRE, VXLAN
- VLAN aware VMs - VXLAN over VLAN
- Scale to thousands of flows with MAC learning
- ethtool SR-IOV per VF Rate limiting
- Linux bonding offload - VF-LAG
- Remote port mirroring with GRE encap

Work in progress (community collaboration)

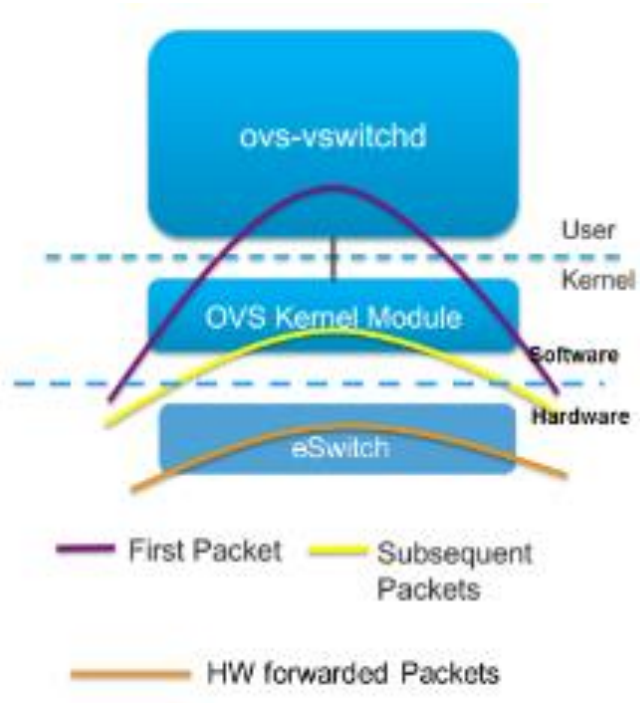
- Flow insertion rate - TC parallelization, NIC software steering
- Traffic engineering - MPLS Segment Routing
- Load balancer & Firewall - Connection tracking
- vDPA - replace SR-IOV, live migration



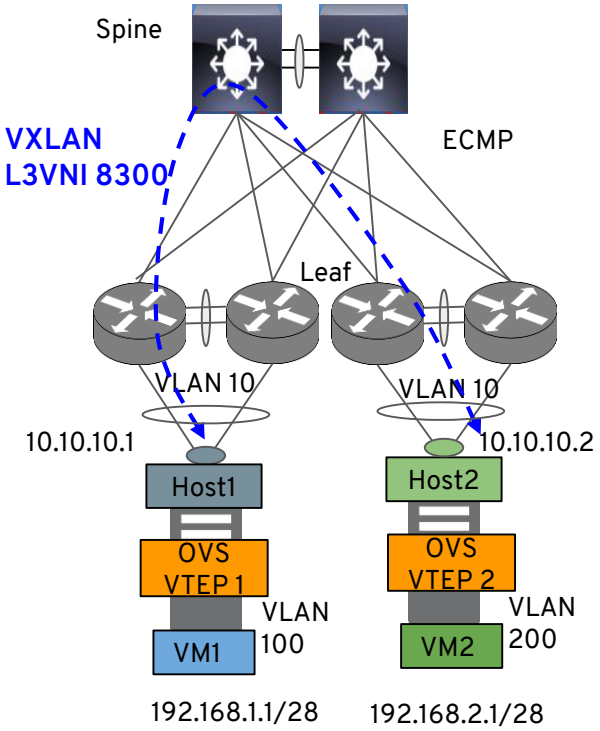
Flow aggregation and insertion rate

Flow aggregation and insertion rate

- Are flows static or dynamic?
- With flow aggregation, individual user (micro) flows are not exposed?
- Flow expansion occurs in the NIC
- **But this is not always the case**.
- Flows can be dynamic and user flows are exposed and need to be handled at the 5-Tuple micro-flow level
- Flow churn and higher insertion rate



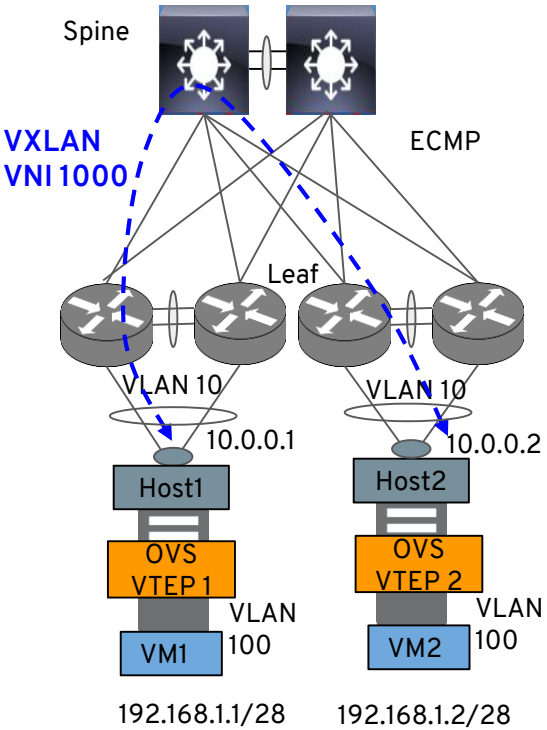
User flow aggregation: Routed flows with TTL decrement




```
Routed Flow VXLAN Encap
skb_priority(0/0),skb_mark(0/0),in_port(ens15f0_2),eth(src=c6
:17:f1:df:4e:d3,dst=68:54:ed:00:8e:10),eth_type(0x0800),ipv4(
src=192.168.0.0/255.255.0.0,dst=192.168.2.1,proto=6,tos=0/0x3
,ttl=64),tcp(src=32768/0x8000,dst=0/0), packets:3381147,
bytes:5288111012, used:0.700s, offloaded:yes, dp:tc,
actions:set(tunnel(tun_id=0x86d288,dst=10.10.10.2,tp_dst=4789
,flags(key))),set(eth(src=68:54:ed:00:13:5c,dst=92:eb:fc:be:f
1:c8)),set(ipv4(ttl=63)),vxlan_sys_4789
```

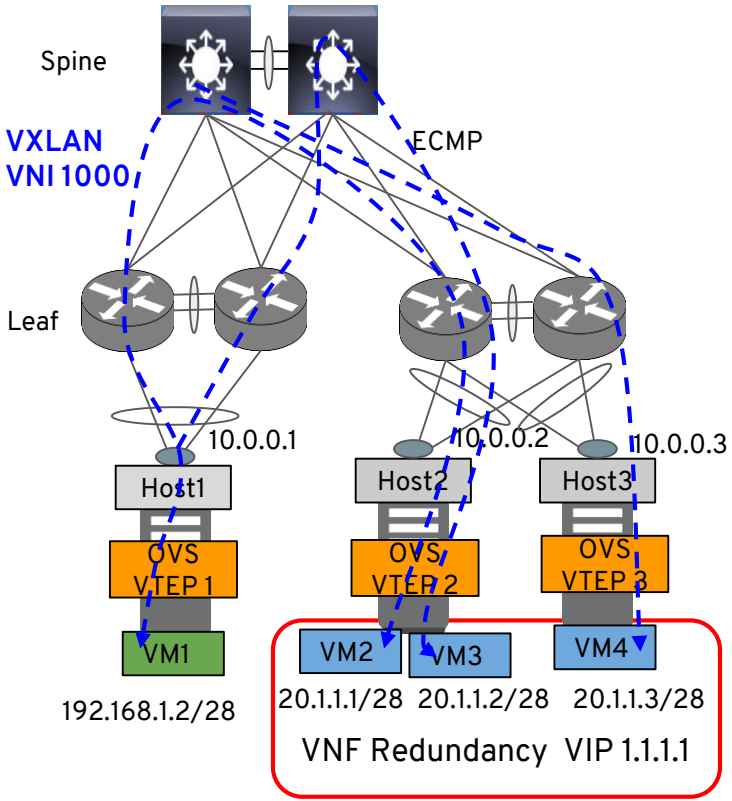
- Typically all North-South flows are routed, East-West flows between VNFs are routed, Remote mirroring traffic is routed
- Routed flows need TTL decrement, which is done as kernel match and set
- This can result in explosion of user flows
- Kernel patch: [PATCH net-next] openvswitch: add TTL decrement action

User flow aggregation - Hash for ECMP/Entropy



- Load balance traffic across 4 or more ECMP paths
- **Hash (dp_hash)** on Inner L2/L3/L4 headers; 5-tuple hash IP src/dst, TCP/UDP port, protocol
- **Set the outer VXLAN UDP source port** with hash results
- This is dp_hash bug, first packet (ovs-vswitchd) and consequent packets are hashed differently in kernel/NIC
- Today hashing is being done in userspace, hence explosion of flows
- Move ECMP/entropy hash to kernel and offload to NIC

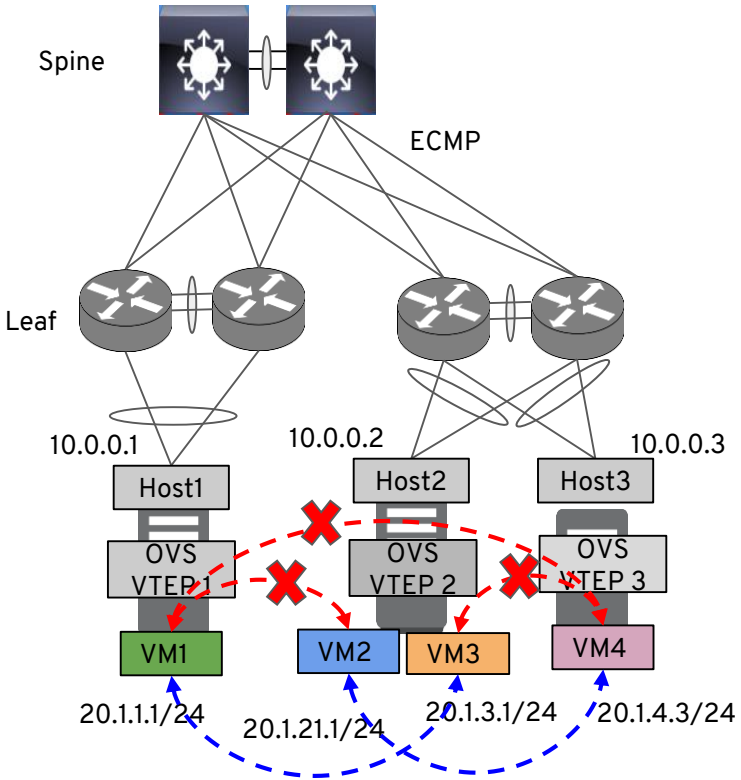
Future - Load Balancing for East-West Services



Future use cases with potential for flow explosion and flow churn
Inner IP/TCP/UDP headers need to be processed on the fly

- OVS for load balancers to distribute East-West traffic across 3-8 remote VNFs
- Resolve VIP and alternate destination VNF MAC/IP for each flow
- User Flows from VM1 need to be distributed across 3 VNF Set (VM2, VM3, VM4) on Host 2 and Host3
- Flows will be distributed across 4 ECMP paths

Future - Network Policy, ACLs, Connection tracking



Future use cases with potential for flow explosion and flow churn

- Firewall and network policy (Security Groups) between services
- Inner IP/TCP/UDP headers need to be examined on the fly for firewall rule
- Allow traffic from VM1 to VM3 and VM2 to VM4 but deny all other traffic
  - Offload Stateless ACLs
  - Offload Stateful security groups with OVS Connection Tracking
  - Conntrack Offload - Flow setup in slow path and then offload flow to NIC
- Connection tracking offload patches https://patchwork.ozlabs.org/cover/1203705/

Improving flow injection rate to NIC

Increase flow insertion rate to > 10K flows/s

- In kernel 5.4 - TC parallelization for flow injection and bypass the RTNL-lock

```
e474619a2498 ("net: sched: flower: don't check for rtnl on head dereference")
195c234d15c9 ("net: sched: flower: handle concurrent mask insertion")
259e60f96785 ("net: sched: flower: protect masks list with spinlock")
9a2d93899897 ("net: sched: flower: handle concurrent filter insertion in
fl_change")
272ffaadeb3e ("net: sched: flower: handle concurrent tcf proto deletion")
3d81e7118d57 ("net: sched: flower: protect flower classifier state with
spinlock")
c24e43d83b7a ("net: sched: flower: track rtnl lock state")
…
```

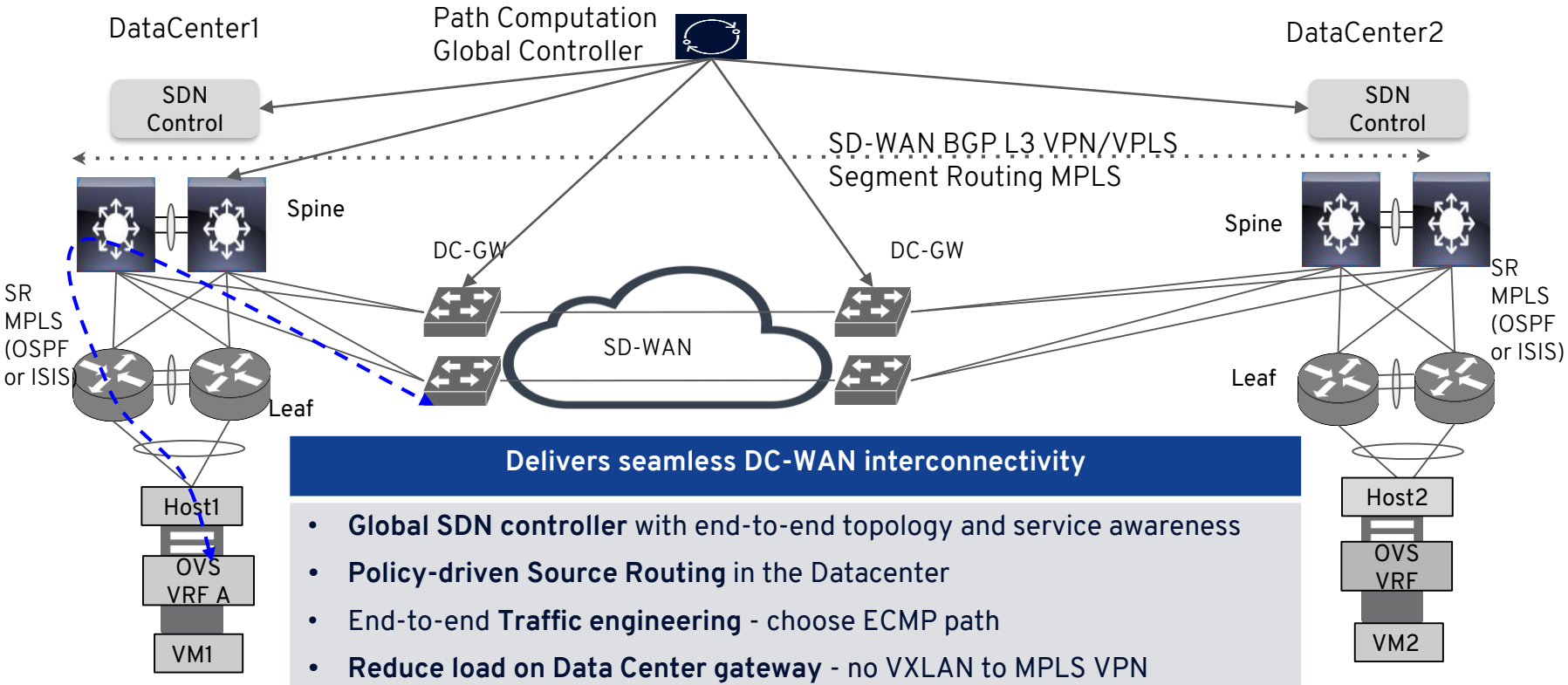
- Mellanox NIC driver - software steering and parallelization

  Fixed in rdma-core-26.0-1

  [net-next 00/13] Mellanox, mlx5 tc flow handling for concurrent execution [Part1] [Part2] [Part3]

MPLS over UDP and Segment Routing for the Edge

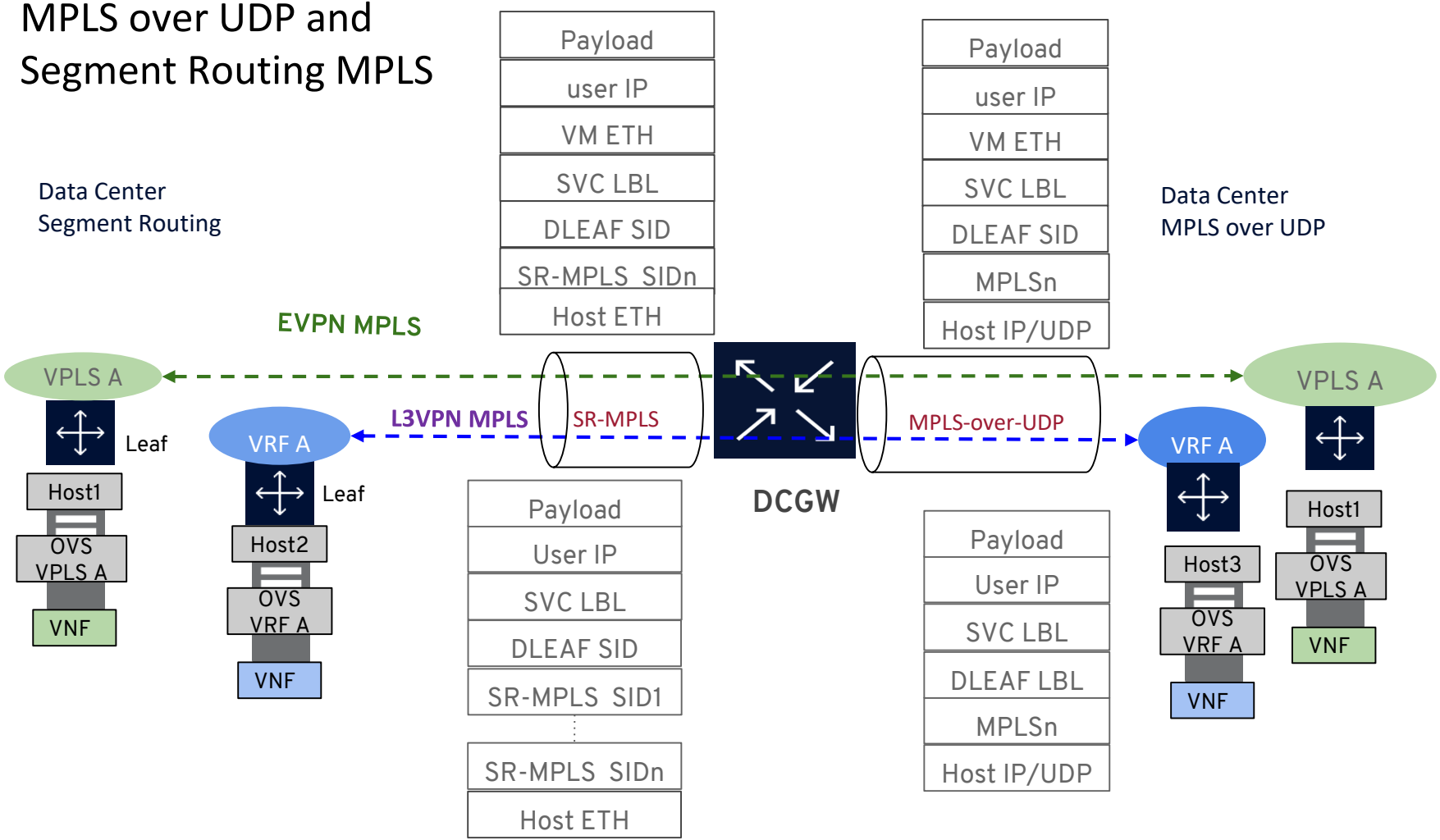
MPLS to the Edge - Source based traffic engineering



Delivers seamless DC-WAN interconnectivity

- **Global SDN controller** with end-to-end topology and service awareness
- **Policy-driven Source Routing** in the Datacenter
- End-to-end **Traffic engineering** - choose ECMP path
- **Reduce load on Data Center gateway** - no VXLAN to MPLS VPN translation
- Service Chaining and improved multicast support

MPLS over UDP and Segment Routing MPLS



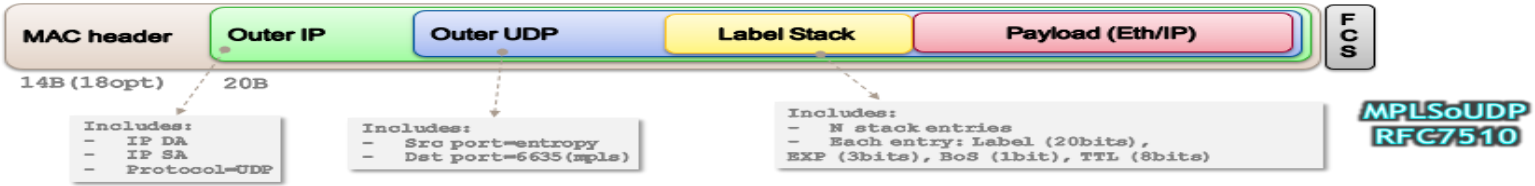
OVS and Kernel support for MPLSoUDP & Segment Routing

- MPLS Segment Routing - label manipulations in the OVS kernel layer:
  - Push 3-6 labels for tunneled traffic with kernel forwarding
  - Set the label TTL and EXP fields to potentially different values for each label
  - POP at least 2 labels on a received packet
  - Encap both L2VPN (Eth/IP) and L3VPN (IP) packets
- Kernel patches (in progress) lightweight tunnel MPLS support https://git.kernel.org/pub/scm/linux/kernel/git/davem/net.git/tree/net/mpls/internal.h#n6
- OVS kernel MPLS multi-label support - https://git.kernel.org/pub/scm/linux/kernel/git/davem/net-next.git/commit/?id=fbdcdd78da7c95f1b970d371e1b23cbd3aa990f3
- MPLS over UDP- insert/remove UDP/IP header on top of the MPLS pushed/popped label stack
  - Needed when MPLS packet is tunneled over an non-MPLS IP underlay
  - Packet header encapsulation as per RFC 7510
  - Kernel MPLS over UDP support - https://patchwork.ozlabs.org/patch/1173153/
- NIC offload capabilities to be addressed
  - RSS handled with UDP src-port hash / TCP segmentation/Re-assembly could be a challenge on deep encap.

Thank-you

PCAP@OVS egress

VNF to PNF example



RFC7510: MPLSoUDP

```
▶ Frame 30: 152 bytes on wire (1216 bits), 152 bytes captured (1216 bits)
▶ Ethernet II, Src: fa:16:3e:f0:62:10 (fa:16:3e:f0:62:10), Dst: fa:16:3e:17:55:d8 (fa:16:3e:17:55:d8)
▶ Internet Protocol Version 4, Src: 19.19.19.7, Dst: 92.168.96.71
▶ User Datagram Protocol, Src Port: 55513, Dst Port: 6635
▶ MultiProtocol Label Switching Header, Label: 50100, Exp: 0, S: 0, TTL: 64
▶ MultiProtocol Label Switching Header, Label: 524283, Exp: 0, S: 0, TTL: 64
▶ MultiProtocol Label Switching Header, Label: 524286, Exp: 0, S: 1, TTL: 64
▶ Data (98 bytes)
```

IPs

Src IP: VRS

Dst IP: DCGW

UDP Dst Port 6635

MPLS labels
BSID: 50100
HV SID: 524283
SL: 524286

OVS tables for SR over UDP

VNF to WAN endpoint example

Table 13 = Routing

```
[root@host-192-168-96-72 vm]# ovs-appctl bridge/dump-flows alubr0 |grep table_id=13
table_id=13, duration=876s, n_packets=0, cookie:0x1 n_bytes=0, priority=65535,vrf_id=0x702e02fa,evpn_id=0x6c756fad,tcp,reg17=0x2000000/0x2000000,nw_dst=10.10.10.1,tp_src=179,actions=mod_dl_dst:2a:ac:72:8a
:fa:0c,output:1
table_id=13, duration=876s, n_packets=0, cookie:0x1 n_bytes=0, priority=65535,vrf_id=0x702e02fa,udp,reg17=0x2000000/0x2000000,nw_dst=10.10.10.1,tp_dst=3784,actions=mod_dl_dst:00:00:fa:02:2e:70,output:1
table_id=13, duration=876s, n_packets=0, cookie:0x1 n_bytes=0, priority=65535,vrf_id=0x702e02fa,udp,reg17=0x2000000/0x2000000,nw_dst=10.10.10.1,tp_dst=4784,actions=mod_dl_dst:00:00:fa:02:2e:70,output:1
table_id=13, duration=876s, n_packets=13, cookie:0x1 n_bytes=1274, priority=32836,vrf_id=0x702e02fa,ip,reg17=0x1000000/0x1000000,nw_dst=10.10.10.56,actions=load:0x1b->NXM_NX_REG5[],set_evpn_id:0x6c756fad,
dec_ttl,resubmit(,12)
table_id=13, duration=874s, n_packets=10, cookie:0x1 n_bytes=980, priority=32834,vrf_id=0x702e02fa,ip,reg17=0x1000000/0x1000000,nw_dst=10.10.30.3,actions=load:0x1f->NXM_NX_REG5[],load:0x3e9->NXM_NX_REG8[]
,resubmit(,30),multipath(symmetric_l3l4+udp,0,iter_hash<3>,1,0,0,0,NXM_NX_REG2[]),dec_ttl,resubmit(,17)
table_id=13, duration=874s, n_packets=3, cookie:0x1 n_bytes=294, priority=32818,vrf_id=0x702e02fa,ip,reg17=0x1000000/0x1000000,nw_dst=10.10.40.0/24,actions=load:0x1e->NXM_NX_REG5[],load:0x3e9->NXM_NX_REG8
[],resubmit(,30),multipath(symmetric_l3l4+udp,0,iter_hash<3>,1,0,0,0,NXM_NX_REG2[]),dec_ttl,resubmit(,17)
table_id=13, duration=855s, n_packets=0, cookie:0x1 n_bytes=0, priority=32818,vrf_id=0x702e02fa,ip,reg17=0x1000000/0x1000000,nw_dst=10.10.10.0/24,actions=load:0x20->NXM_NX_REG5[],set_evpn_id:0x6c756fad,de
c_ttl,resubmit(,12)
table_id=13, duration=876s, n_packets=0, cookie:0x1 n_bytes=0, priority=0,vrf_id=0x702e02fa,reg17=0x1000000/0x1000000,actions=drop
```

Table 31 = Service Label

```
[root@host-192-168-96-72 vm]# ovs-appctl bridge/dump-flows alubr0 |grep table_id=31
table_id=31, duration=889s, n_packets=3, cookie:0x1e n_bytes=294, priority=1,vrf_id=0x702e02fa,reg4=0x6,reg5=0x1e,actions=mod_dl_src:96:03:b8:bb:8e:46,mod_dl_dst:c0:0d:ff:ff:ff:4c,load:0xd60a85c->NXM_NX_R
EG0[],encap(UNKNOWN),push_mpls:0x8847,set_mpls_label(524286),resubmit(,66),pause(1)
table_id=31, duration=889s, n_packets=10, cookie:0x1f n_bytes=980, priority=1,vrf_id=0x702e02fa,reg4=0x6,reg5=0x1f,actions=mod_dl_src:96:03:b8:bb:8e:46,mod_dl_dst:c0:0d:ff:ff:ff:4c,load:0xd60a85c->NXM_NX_
REG0[],encap(UNKNOWN),push_mpls:0x8847,set_mpls_label(524286),resubmit(,66),pause(1)
table_id=31, duration=82172s, n_packets=0, cookie:0x1 n_bytes=0, priority=0,actions=note:be.c0.01.de.ad.be.ef.02
```

Table 66 = Remote BGP LU SID

```
[root@host-192-168-96-72 vm]# ovs-appctl bridge/dump-flows alubr0 |grep table_id=66
table_id=66, duration=724s, n_packets=8, cookie:0x1 n_bytes=784, priority=1,reg0=0xd60a85c,actions=load:0x7->NXM_NX_REG1[],push_mpls:0x8847,set_mpls_label(524283),resubmit(,62)
table_id=66, duration=82201s, n_packets=5, cookie:0x0 n_bytes=490, priority=0,actions=drop
```

Table 62 = BSID

```
[root@host-192-168-96-72 vm]# ovs-appctl bridge/dump-flows alubr0 |grep table_id=62
table_id=62, duration=744s, n_packets=8, cookie:0x1 n_bytes=784, priority=1,reg8=0x3e9,actions=push_mpls:0x8847,set_mpls_label(50100),resubmit(,63)
table_id=62, duration=82221s, n_packets=0, cookie:0x0 n_bytes=0, priority=0,actions=resubmit(,63)
```