Multiple NICs as Encapsulation Endpoints for Large Scale GPU Cloud

Girish Moodalbail, Han Zhou
NVIDIA

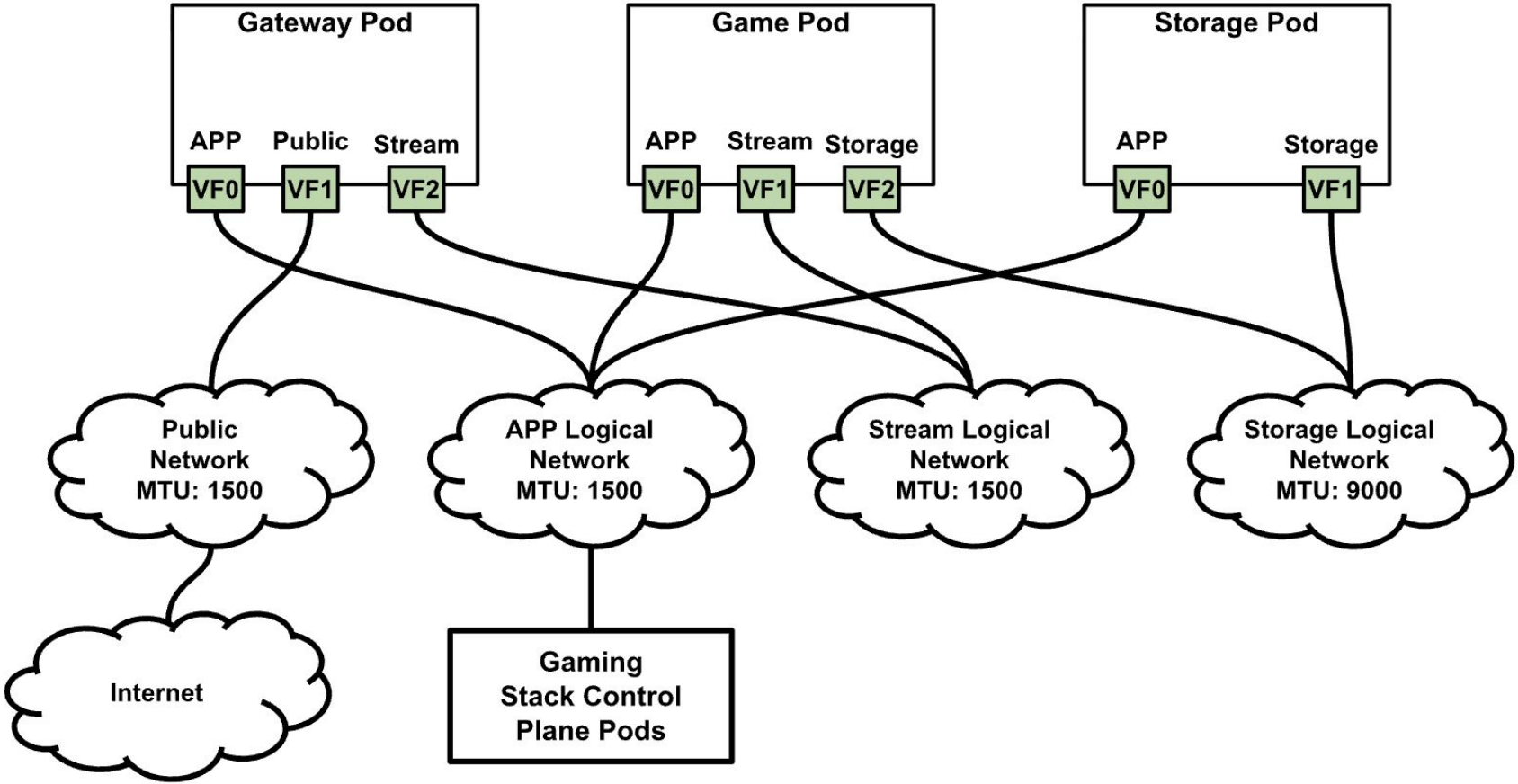
Agenda

- Need for Multi-Homed Pods
  - Cloud Gaming Use Case
  - AI Training/Inference Use Case
- LSP to OVS Interface to Encap-IP Mapping
- OVN-Kubernetes Changes
- OVN tunnel creation & selection
- Flow programming
- Special cases
- Scale considerations
- Future work

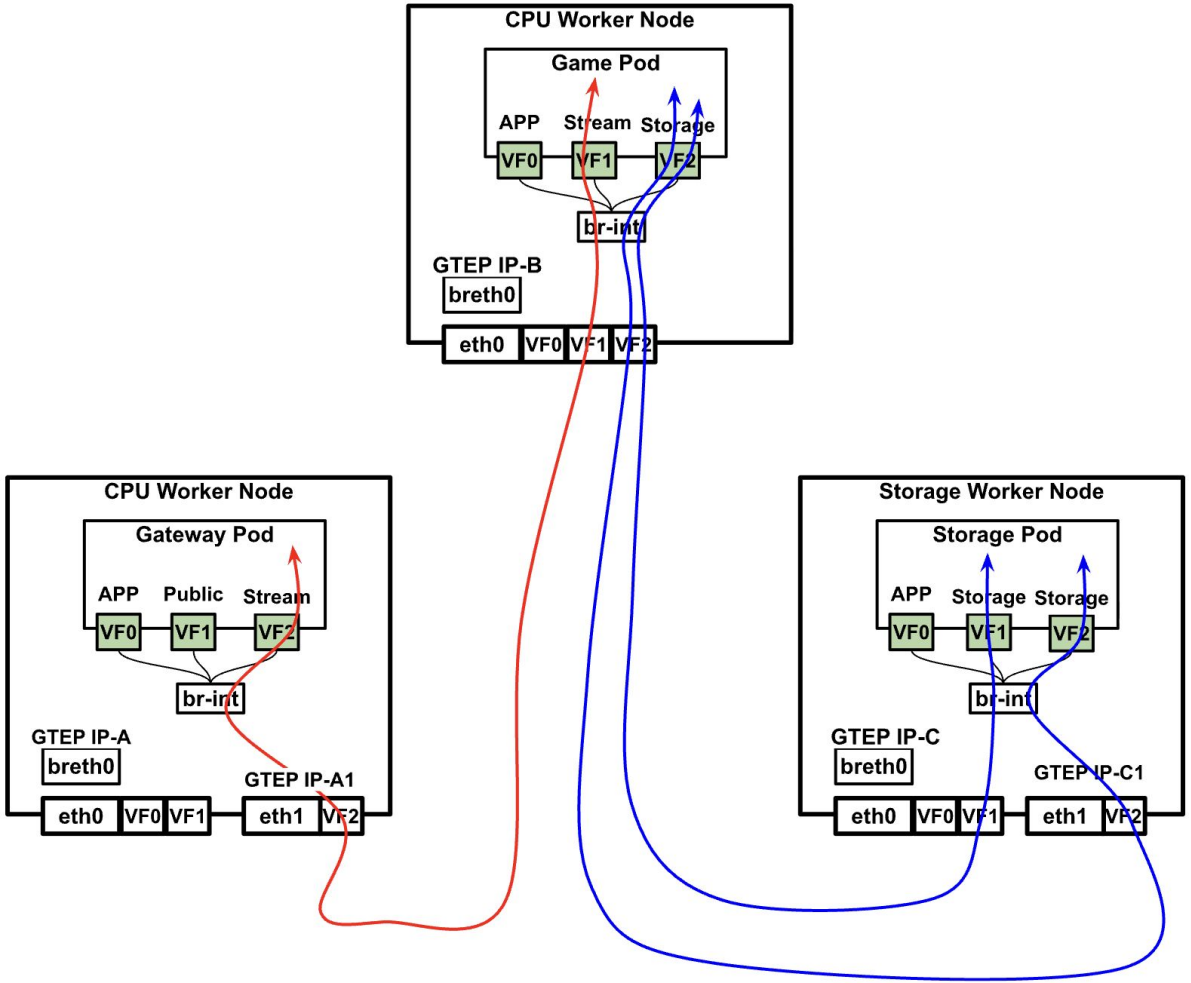
Multi-Homed Pods

- Multi-homed Pods are commonly used in cloud gaming and AI training or inference workloads
- These workloads often need to separate different types of network traffic:
  - Latency sensitive traffic
    - streaming game frames from GPU to client
  - High throughput storage traffic
    - mounting virtual disk images containing games
    - checkpointing/restoring in AI training.
  - Control plane traffic
    - such as downloading large LLM models, weights, input artifacts for inferencing from the Internet or
    - uploading user saves to S3 buckets

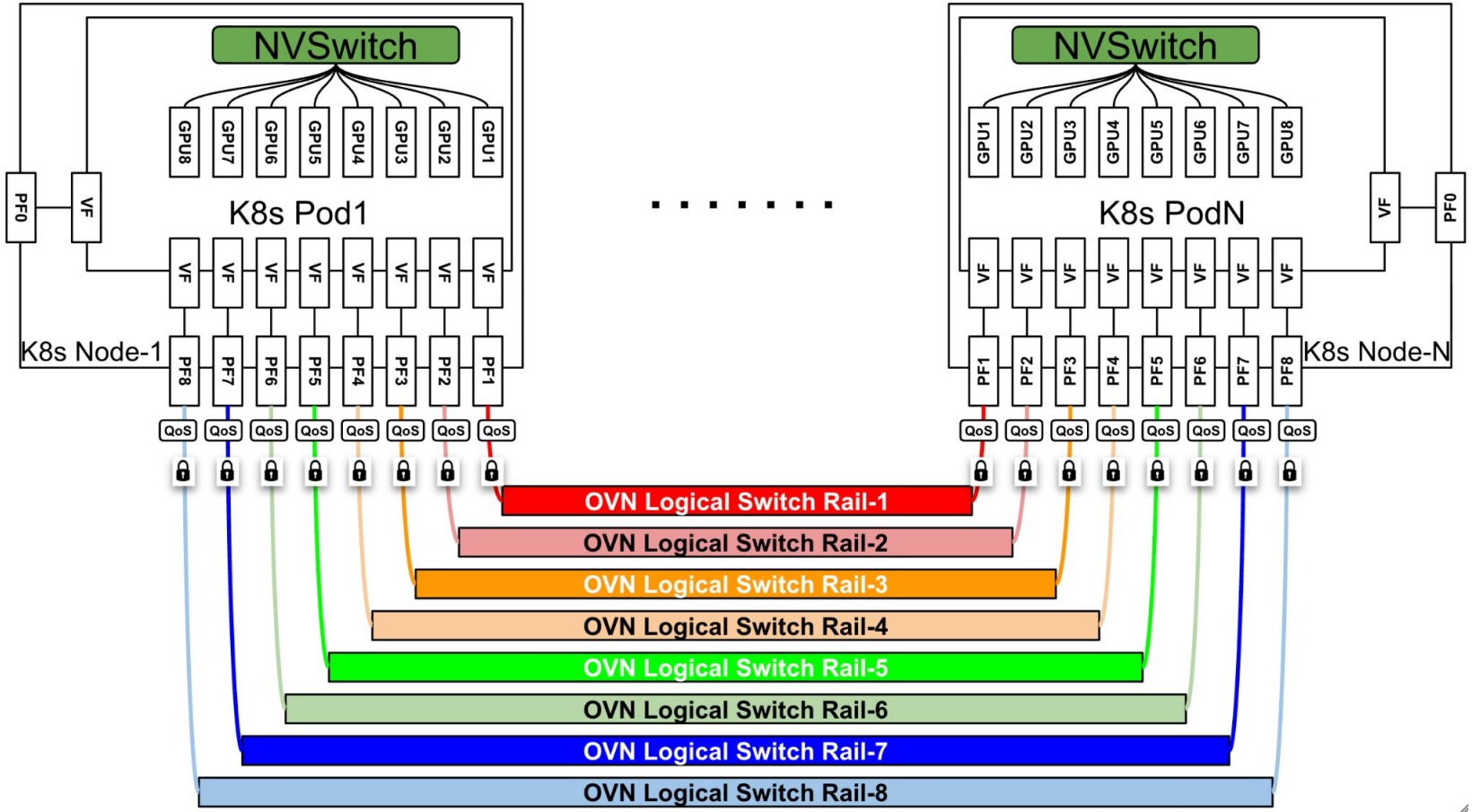
Cloud Gaming Use Case (Logical View)



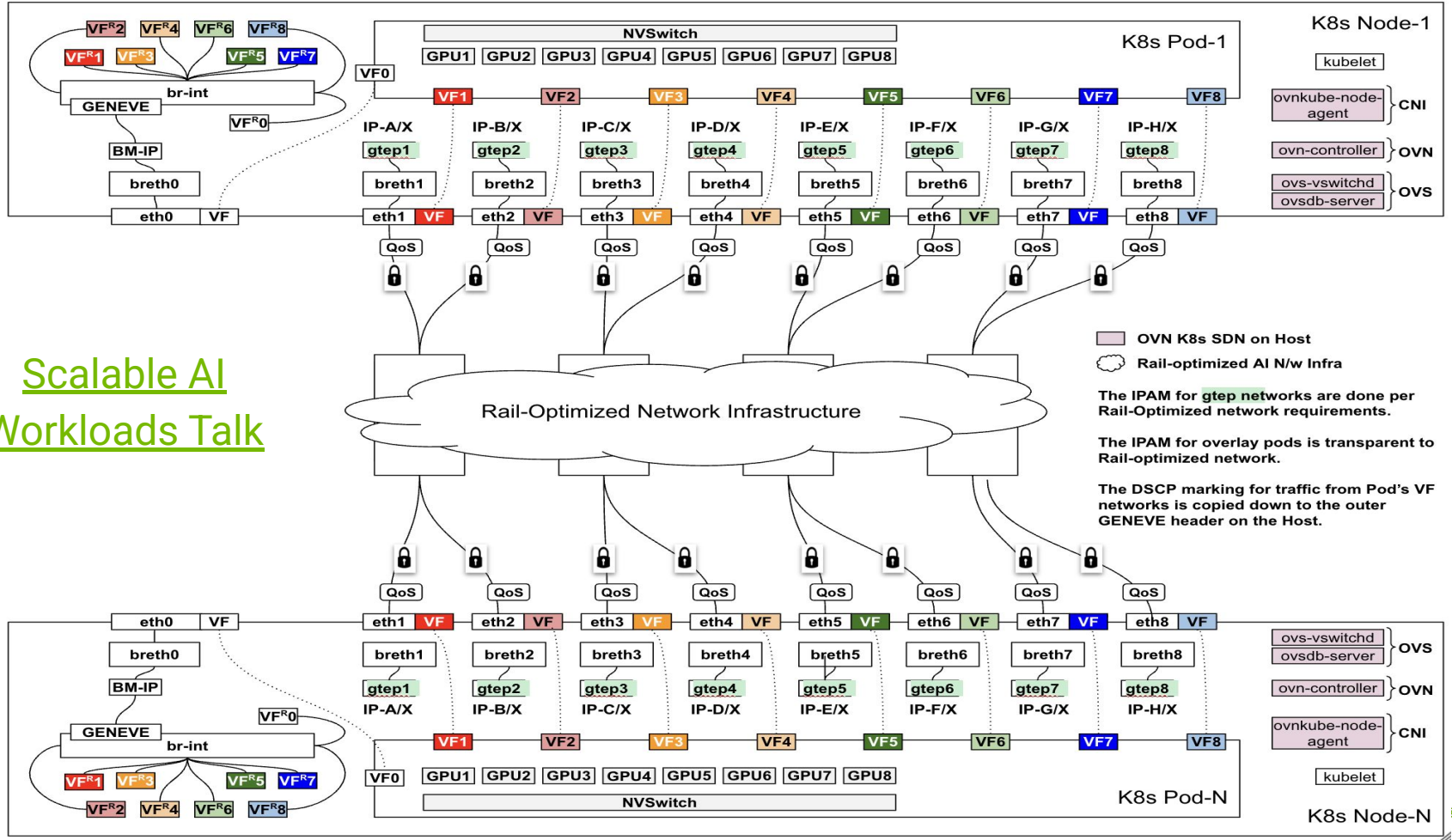
Cloud Gaming Use Case (Physical View)-2



Multi-Node AI Workloads (OVN Logical View)



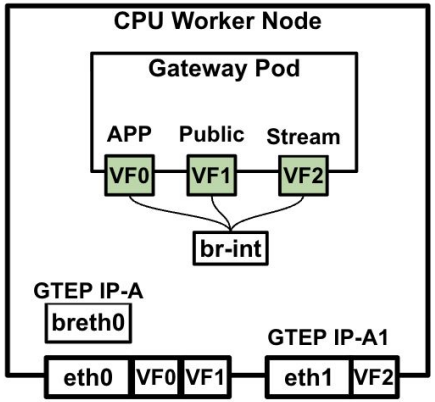
Multi-Node AI Workloads (Physical View)



Scalable AI Workloads Talk

OVN-Kubernetes Changes

LSP to OVS Interface to Encap-IP Mapping



Config for OVN-Kubernetes

```
# ovs-vsctl get Open_vSwitch . external_ids:ovn-pf-encap-ip-mapping
"enp129s0f0:10.2.0.43,enp193s0f0:10.2.0.45,enp1s0f0:10.2.0.41"
```

Config for OVN

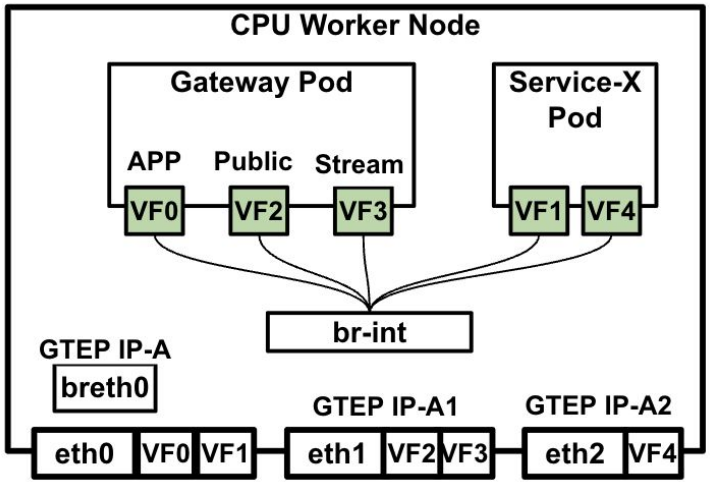
```
# ovs-vsctl get Open_vSwitch . external_ids:ovn-encap-ip
"10.2.0.43,10.2.0.45,10.2.0.41"
# ovs-vsctl get Open_vSwitch . external_ids:ovn-encap-ip-default
"10.2.0.41"
# ovs-vsctl get Interface enp1s0f0_12 external_ids:encap-ip
"10.2.0.41"
```

OVS DB Table	Option	Value	Notes
Used by OVN-K8s			
Open_vSwitch	external_ids:\ ovn-pf-encap-ip-mapping	<ifName:encap-ip-A>, <ifName:encap-ip-A1>	An hint to map a VF to PF and then to Encap IP
Used by OVN			
Open_vSwitch	external_ids:ovn-encap-ip	encap-IP-A,encap-IP-A1	Comma-separated values capturing all the Encap IPs on the Chassis
Open_vSwitch	external_ids:ovn-encap-ip -default	encap-IP-A	When ovn-encap-ip contains multiple IPs, this field indicates the default one.
Interface	external_ids:encap-ip	<encap-ip to use for this OVS interface>	

Host Networking Stack Configuration (OVS-Kernel)

- VTEP IPs from the same subnet create routing challenges in Linux
- Requires source-based policy routing
- Requires kernel {arp|rp}_filter configuration

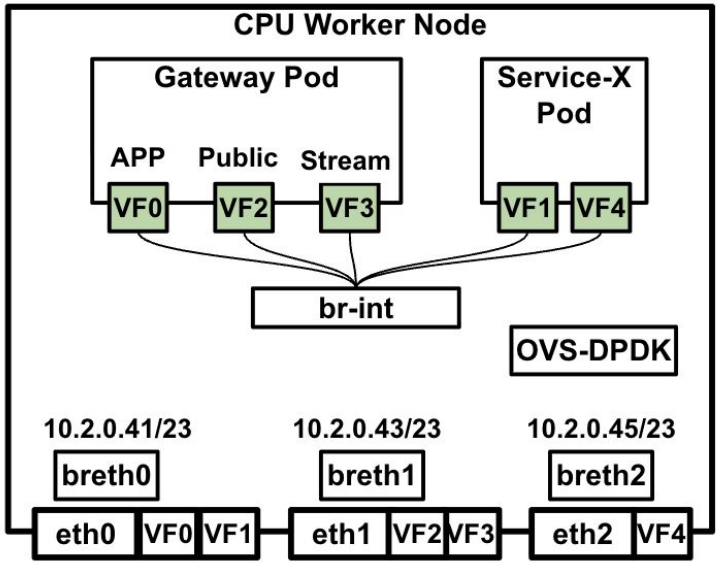
```
for i in `seq 0 2`; do
    sysctl -w net.ipv4.conf.vtep$i.rp_filter=2
    sysctl -w net.ipv4.conf.vtep$i.arp_ignore=1
    sysctl -w net.ipv4.conf.vtep$i.arp_announce=2
done
```




```
# ip -br a show |grep vtep
vtep0            UNKNOWN        10.2.0.41/23
vtep1            UNKNOWN        10.2.0.43/23
vtep2            UNKNOWN        10.2.0.45/23
# ip rule show
6081: from 10.2.0.41 lookup 6081 proto static
6082: from 10.2.0.43 lookup 6082 proto static
6083: from 10.2.0.45 lookup 6083 proto static
# ip ro show table 6083
10.2.0.0/16 via 10.2.0.1 dev vtep2 proto static metric 805
# ip ro show table 6082
10.2.0.0/16 via 10.2.0.1 dev vtep1 proto static metric 802
# ip ro show table 6081
10.2.0.0/16 via 10.2.0.1 dev vtep0 proto static metric 800
```

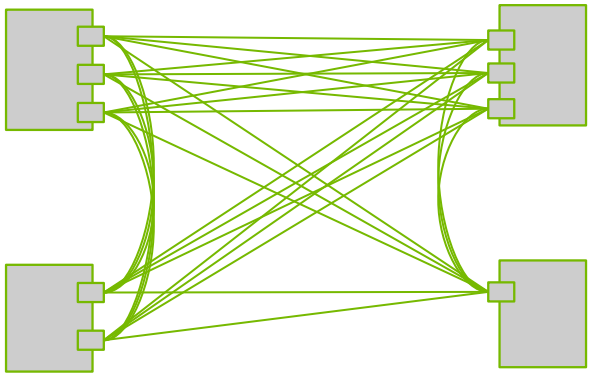
OVS-DPDK Configuration

- VTEP IPs from the same subnet create routing challenges in OVS-DPDK
- OVS-DPDK process learns routing only from the default table when it starts
- Initially ended up using the ***ovs-appctl ovs/route/add*** commands and had to deal with OVS restarts
- Submitted path upstream to fix reading routes from non-default routing tables as well - ovs-router: Multi-table routing infrastructure.




```
# Before as a Work Around

$ ovs-appctl ovs/route/add 10.2.0.0/16 breth0 10.2.0.1
src_match=10.2.0.41
$ ovs-appctl ovs/route/add 10.2.0.0/16 breth1 10.2.0.1
src_match=10.2.0.43
$ ovs-appctl ovs/route/add 10.2.0.0/16 breth2 10.2.0.1
src_match=10.2.0.45
```

Tunnel Creation

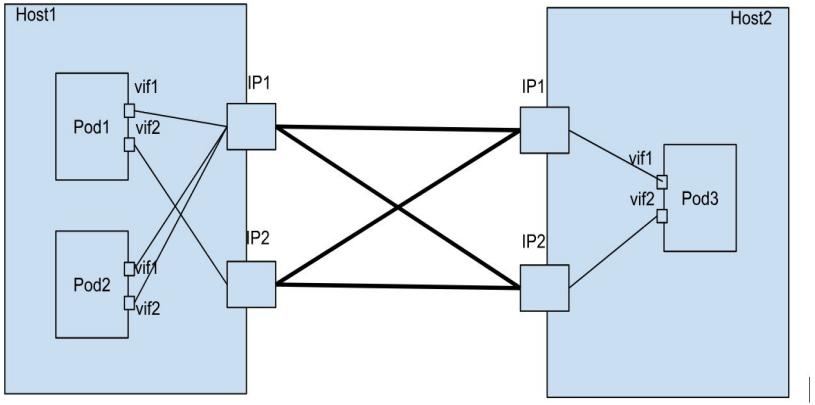


- Config on each node
  - open_vswitch:external_ids:ovn-encap-ip
    - A list of comma-separated IPs
- In SB-DB
  - Chassis:Encaps
- Full-mesh tunnel creation
  - All local x all remote encaps
- Tunnel ID: chassis@**remote_ip**%**local_ip**
  - Stored in OVS conf DB port:external-ids:ovn-chassis-id

Tunnel Selection

- Config on each node
  - Per-VIF: interface:external_ids:ovn-encap-ip
    - Source Encap IP for VIF-originated packets.
    - Dest Encap IP for packets destined to this VIF from a remote node.
  - Chassis level default (Optional): open_vswitch:external_ids:ovn-encap-ip-default

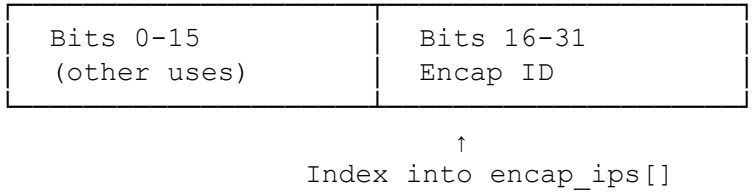
- In SB-DB
  - Port_Binding:Encap:IP
    - Populated by ovn-controller using the per-VIF encap-IP config.
    - Used by ovn-controller to determine how to reach remote VIFs.



- Pod1's vif1 <-> Pod3's vif1 ……. IP1@host1 <-> IP1@host2
- Pod1's vif2 <-> Pod3's vif2 ……. IP2@host1 <-> IP2@host2
- Pod2's vif1/vif2 <-> Pod3's vif2 .. IP1@host1 <-> IP2@host2

OpenFlow Programming

- Introduce Encap-ID, 1-1 mapping to Encap-IP, tracked in REG13[16:31], according to inport VIF's Encap-IP



- For each remote port-binding output, generate flows per-encap-id match, output to corresponding tunnel port
  - O(E x P), E = local Encap-IPs, P = remote Port Bindings

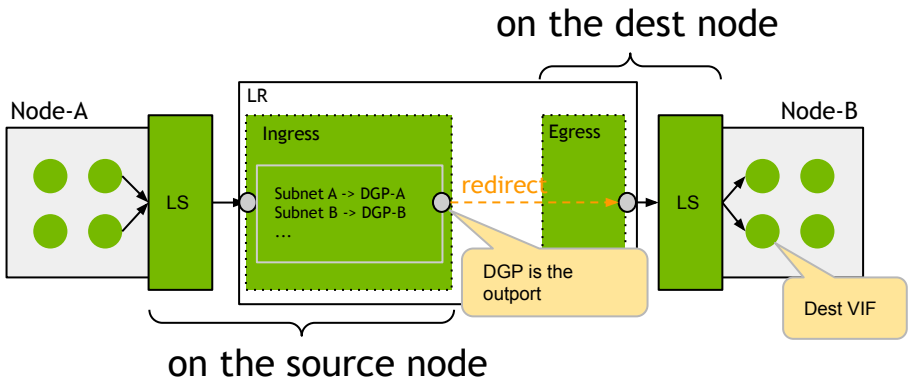
Ingress: Set Encap ID

```
# VIF with encap-ip=10.0.2.1 (index 1)
table=0, priority=100, in_port=5
actions=
    load:0x5->NXM_NX_REG14[],          # inport
    load:0x1->NXM_NX_REG13[16..31],    # Encap ID = 1
    resubmit(,8)
```

Egress: Match Encap ID

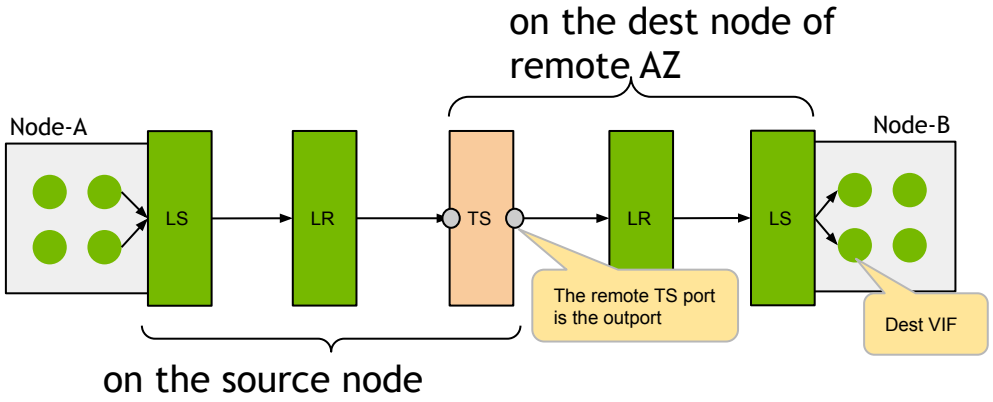
```
# flow1
table=37, priority=100, reg15=0xa,           # outport
                        reg13=0x0/0xffff0000 # Encap ID = 0
actions=
    set_field:0x3->tun_id,
    output:10  # Tunnel with local_ip=10.0.1.1

# flow2
table=37, priority=100, reg15=0xa,           # outport
                reg13=0x10000/0xffff0000     # Encap ID = 1
actions=
    set_field:0x3->tun_id,
    output:11  # Tunnel with local_ip=10.0.2.1
```

Special cases: outport is NOT the dest VIF (1)



- L3 connected through Distributed Gateway Port
  - The sender node outputs to DGP, instead of the dest VIF
  - Solution: Avoid DGP if encap-selection is required, or use the default encap
  - In ovn-k8s central mode, skip pinning LS to node for specific NADs
    - **`k8s.ovn.org/node-skip-pinned-ls-for-networks`**

Special cases: outport is NOT the dest VIF (2)



- Transit switch in ovn-k8s IC
  - The sender node outputs to the remote transit-switch port
  - Solution (proposal):
    - Create multiple TS-LR ports
    - Add /32 routes for each VIF via corresponding TS-LR port
    - Support “request-encap-ip” (TBD) through NB-DB

Scale Considerations

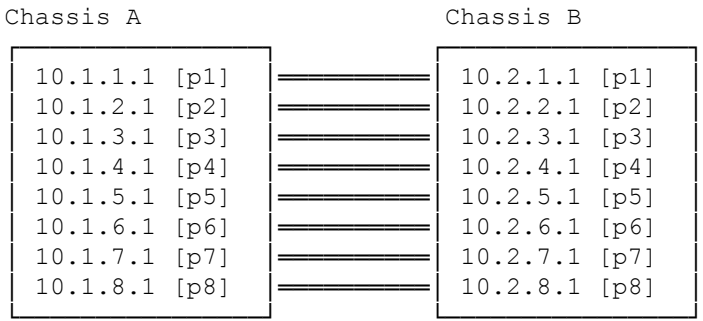
- Number of tunnel ports = O(C * N^2), C = Chassis, N = NICs
- E.g.: 8 NICs per node, 1000 Chassis
  - 64k tunnel ports on each node
- ovs-vswitchd may become a bottleneck. Improvement ideas discussed at [0]
- Scripts for metrics collection and troubleshooting may need adjustment
- Alternatively, use flow-based tunnel implementation [1]
  - Pros: only one port per tunnel type.
  - Cons: feature limitations - e.g. IPSec, BFD - required for GW chassis
- For isolated networks: Transport Plane

[0] - https://mail.openvswitch.org/pipermail/ovs-dev/2025-September/426255.html
[1] - https://mail.openvswitch.org/pipermail/ovs-dev/2025-November/427784.html

Transport Plane (TBD)

Concept: Group encap IPs into logical "planes"

- Only IPs in **same plane** create tunnels
- Different planes are **isolated**



Result: 8 tunnels (one per plane)

Reduction: 64 → 8 (87.5% fewer tunnels!)

Configuration:

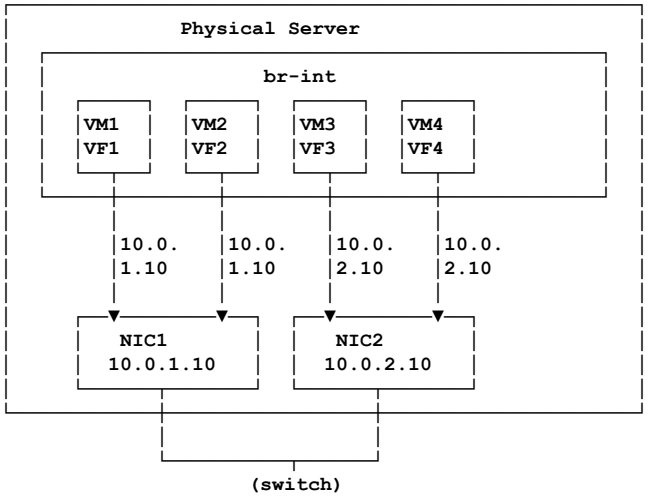
```
# Format: IP:plane_name,IP:plane_name,...
ovs-vsctl set open . external-ids:ovn-encap-ip=\
  10.1.1.1:plane1,\
  10.1.2.1:plane2,\
  10.1.3.1:plane3,\
  10.1.4.1:plane4,\
  10.1.5.1:plane5,\
  10.1.6.1:plane6,\
  10.1.7.1:plane7,\
  10.1.8.1:plane8
```

Rules:

- IP without suffix → "default" plane
- IPs with same plane name → connect
- IPs with different plane names → no connection

Intra-node Tunneling for HW offload (TBD)

- **Problem**: traffic between VFs belonging to different NICs on the same node breaks HW offload
- **Solution:** intro-node tunneling between NICs



- VM1 ↔ VM2: Local (via br-int, no tunnel)
- VM3 ↔ VM4: Local (via br-int, no tunnel)
- VM1 ↔ VM3: **Tunneled intra-node** (HW offload)
- VM1 ↔ VM4: **Tunneled intra-node** (HW offload)
- VM2 ↔ VM3: **Tunneled intra-node** (HW offload)
- …


NVIDIA
Thank You