OpenShift Networking Transformed

Fully Embracing DPDK Datapaths in OVN-K8s!?

Jakob Meng

Maxime Coquelin

Red Hat

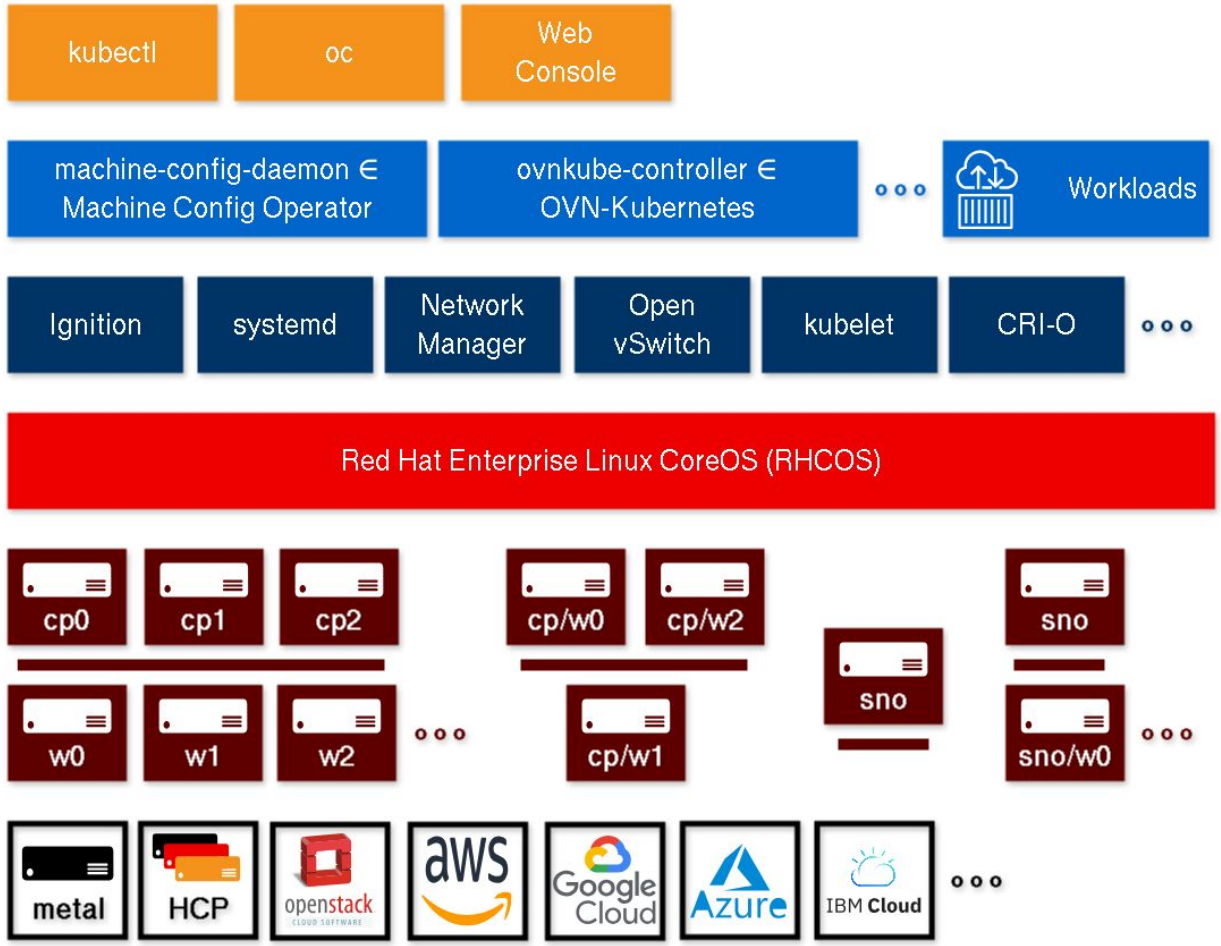
Agenda

- Introduction to OpenShift
- Userspace datapath
- Userspace datapath in Openshift
- Benchmarks
- Conclusion

Introduction to OpenShift

Introduction to OpenShift

Opinionated Kubernetes Distribution for Hybrid Cloud



Userspace datapath

Why would we need DPDK in OCP/K8s?

The OVS kernel datapath is already used in production and handles most of the use cases. But we believe there are several reasons why a userspace datapath could bring benefits in some cases:

- Partitioning: Isolation of the networking infra from the workloads
- Determinism: Provide better predictability of the packets latency
- Uniformity: Same datapath for primary and secondary networks

Kernel interface with DPDK datapath

VETH pairs are used with OVS Kernel for the primary network.

- Using AF_XDP on a VETH peer looked like a potential option
- **But does not support TCP Segmentation offload**

Virtio specification implements TSO and other offloads

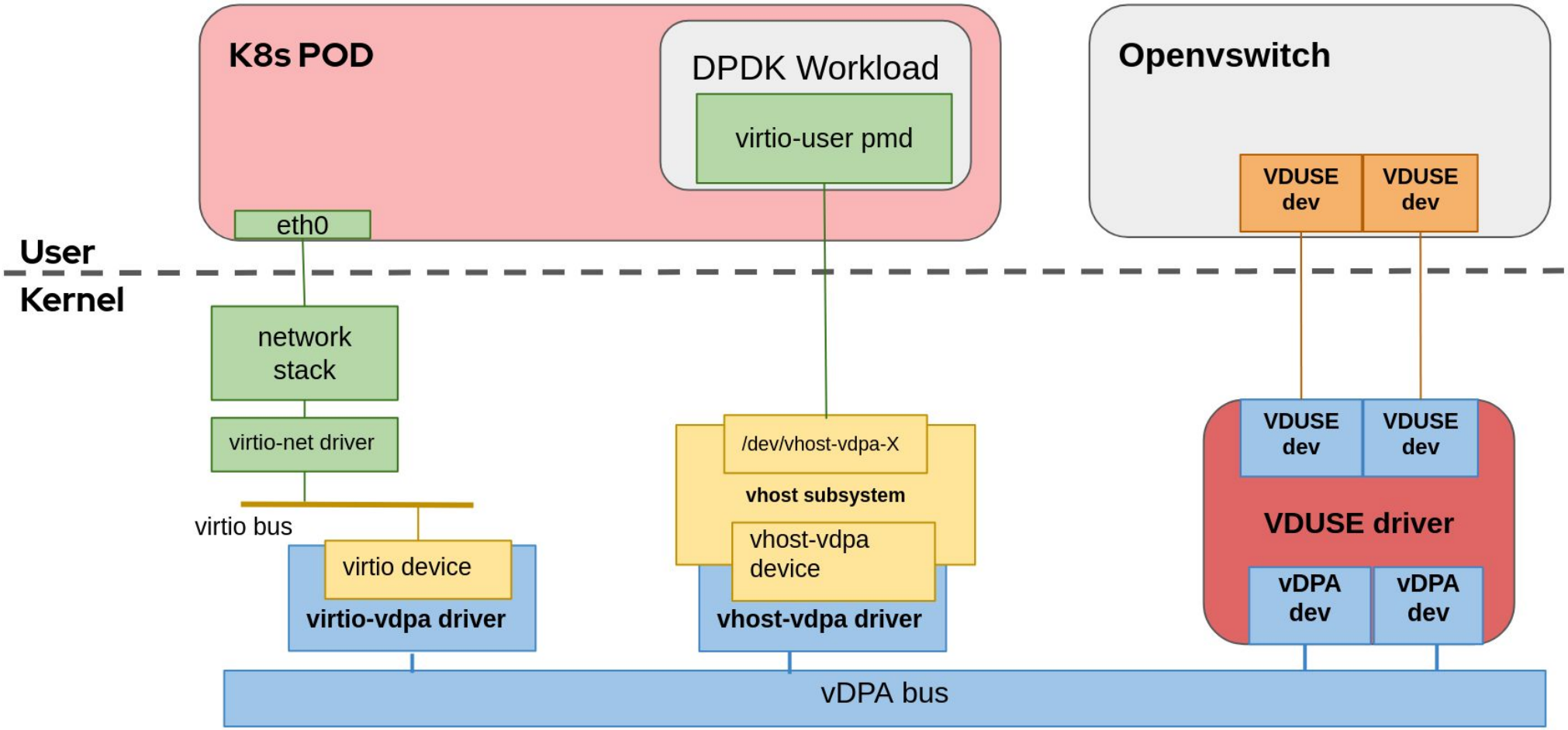
- Heavily used in virtualization environment with Vhost-user
- **But Vhost-user only provides a Unix socket, not a Netdev**

VDUSE

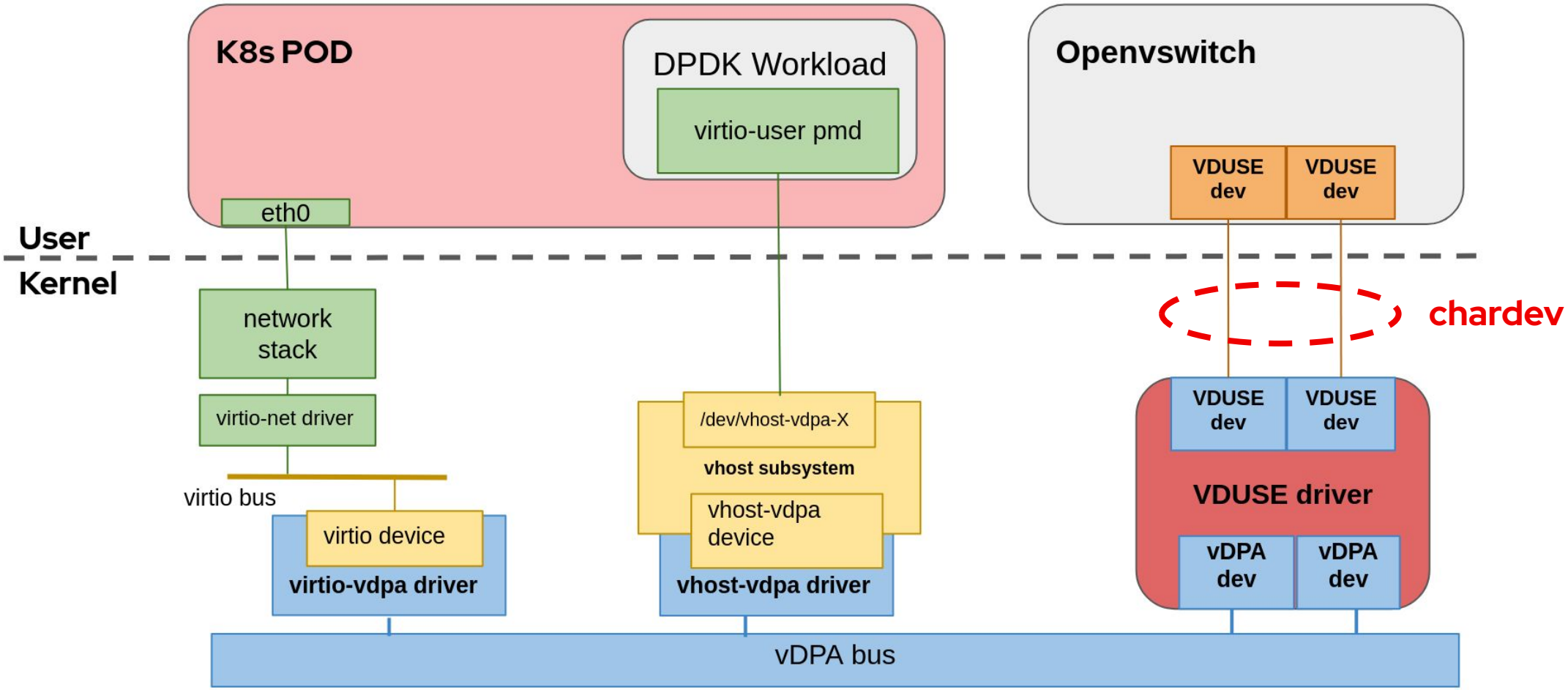
VDUSE stands for **v**DPA **d**evice in **use**rspace

- Usually, vDPA devices are physical (ConnectX6, Octeon, ...)
  - Devices implements the Virtio datapath
  - Vendor specific control path
  - vDPA drivers implement vDPA callbacks to control the device
- VDUSE is purely software, with two components
  - A kernel driver that connects to the vDPA bus
  - A userspace application that implements the actual device

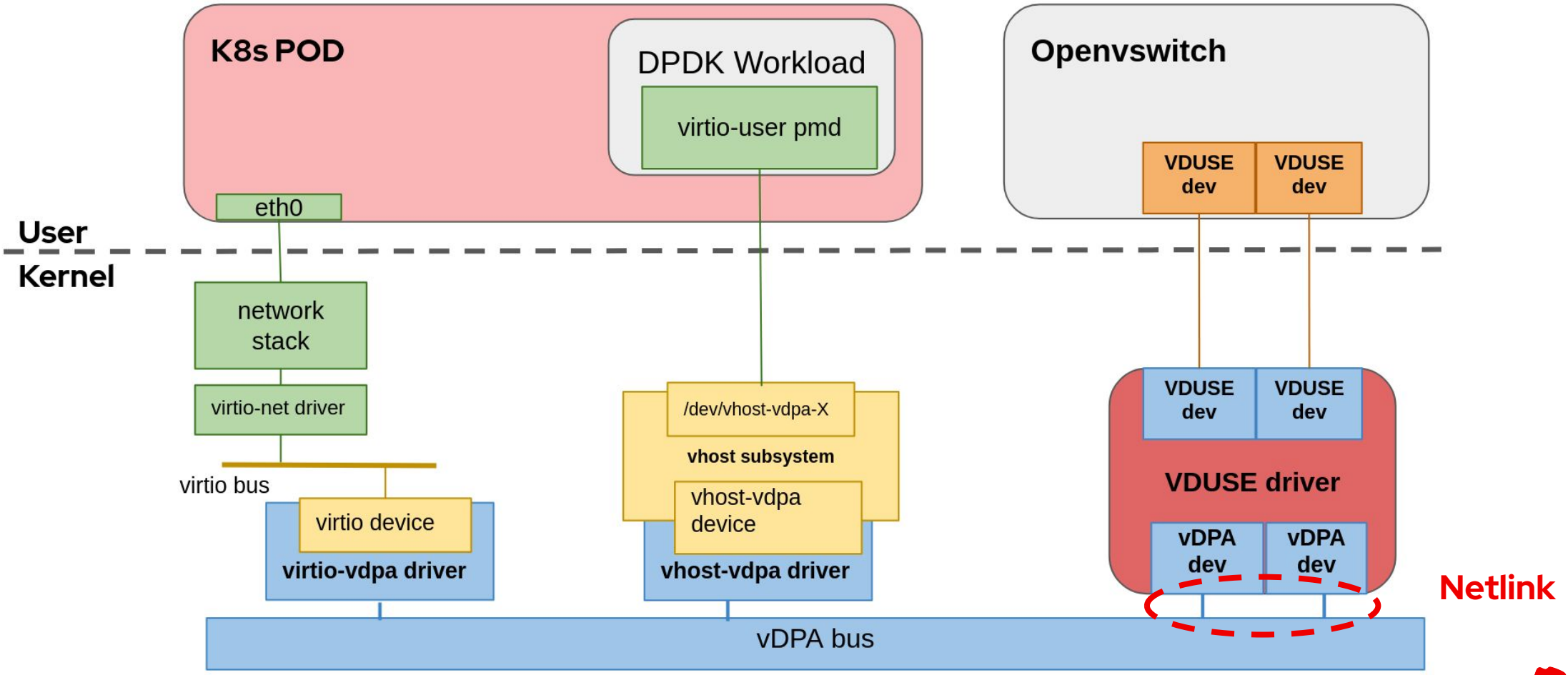
VDUSE for networking architecture



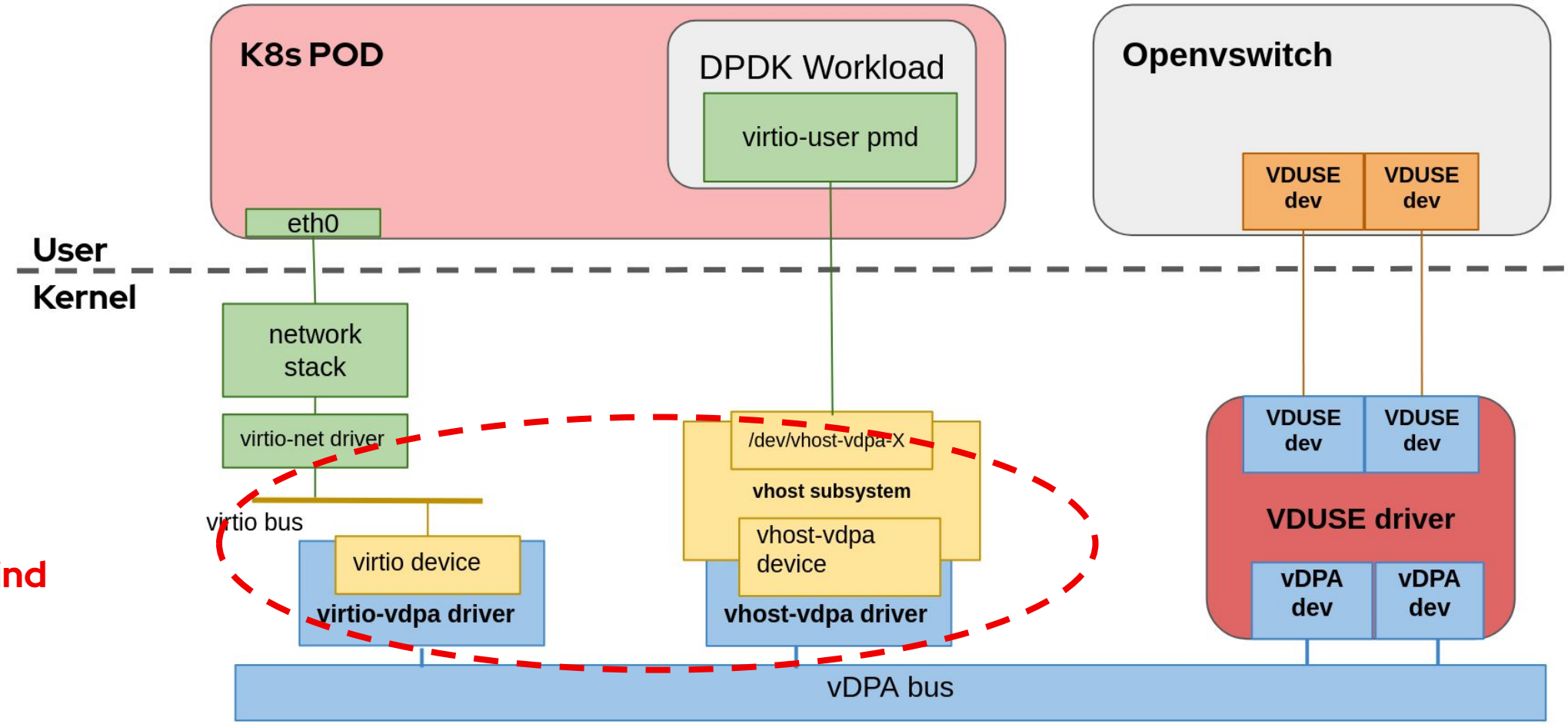
VDUSE for networking architecture



VDUSE for networking architecture

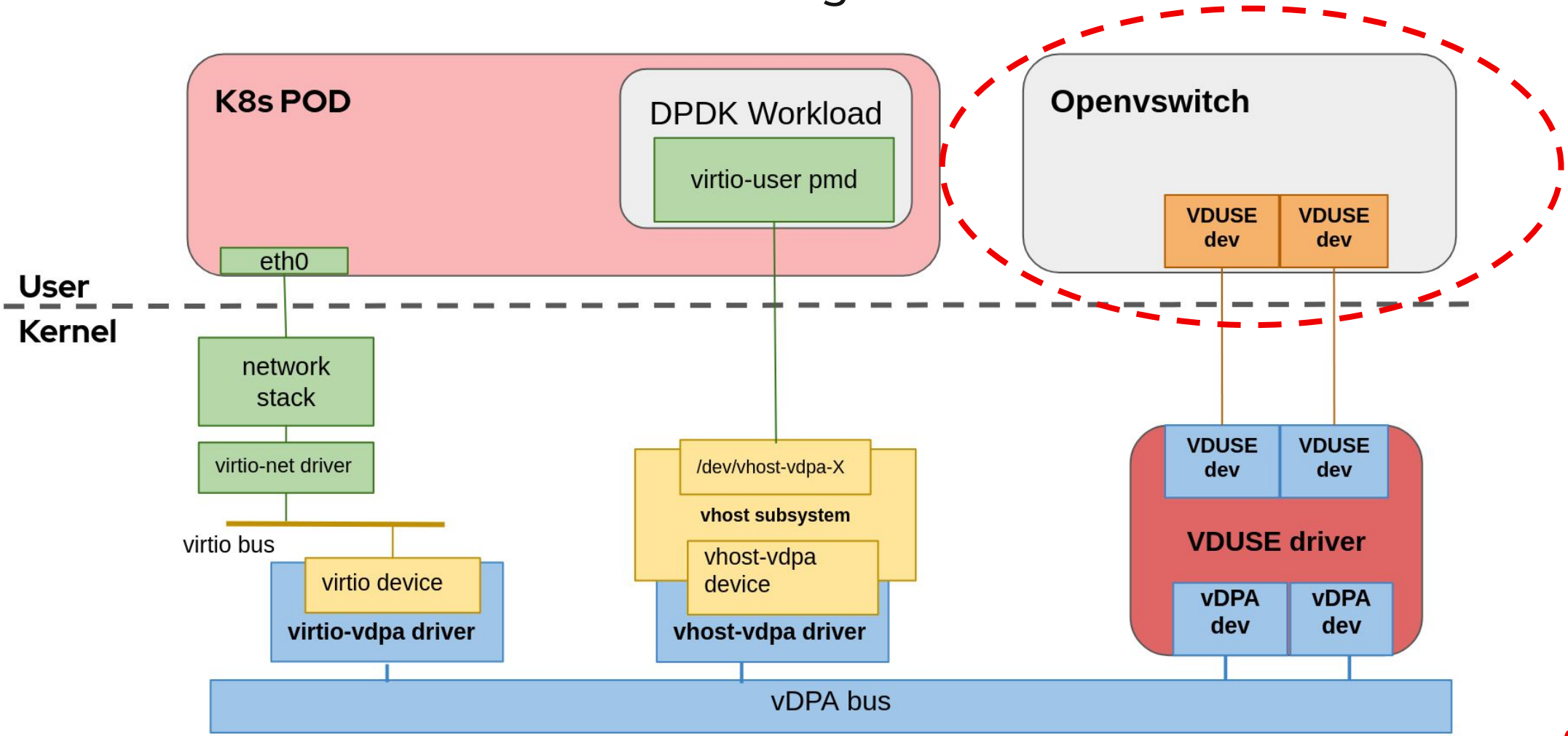


VDUSE for networking architecture



Dev bind

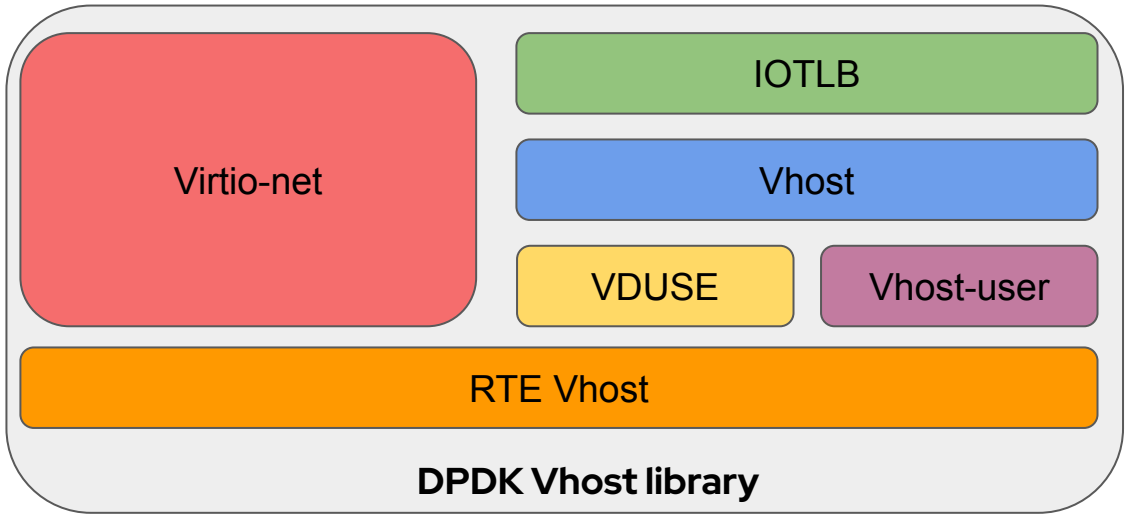
VDUSE for networking architecture



VDUSE in OVS

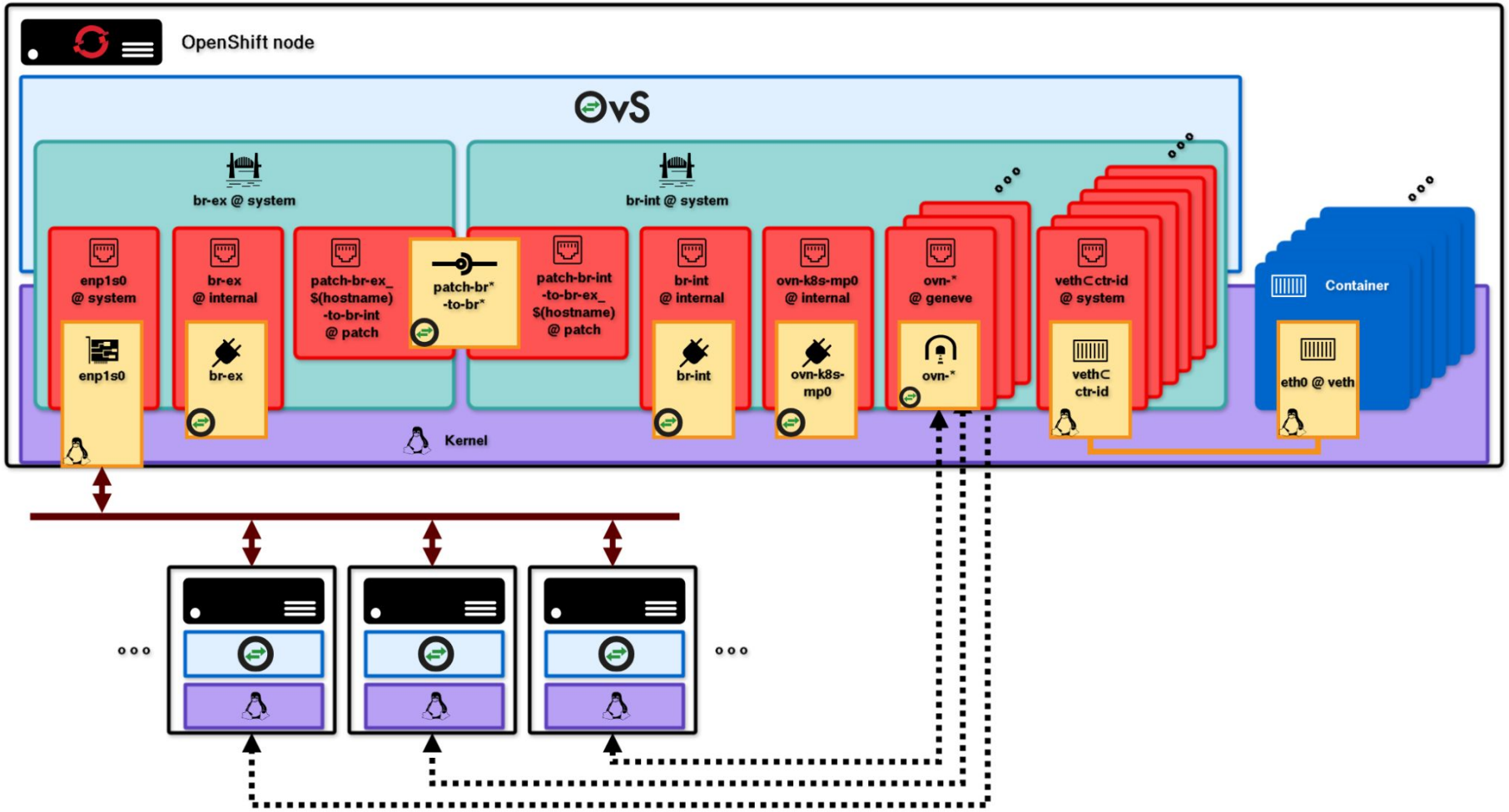
VDUSE support has been implemented in the **DPDK Vhost library**

- Reuse the existing datapath implementation, extensively used/optimized with Vhost-user
- Seemless integration into OVS
  - *dpdkvhostuserclient* is reused
  - Only a new API call is needed to limit the maximum number of queue pairs (OVS v3.5)



Userspace datapath in OpenShift

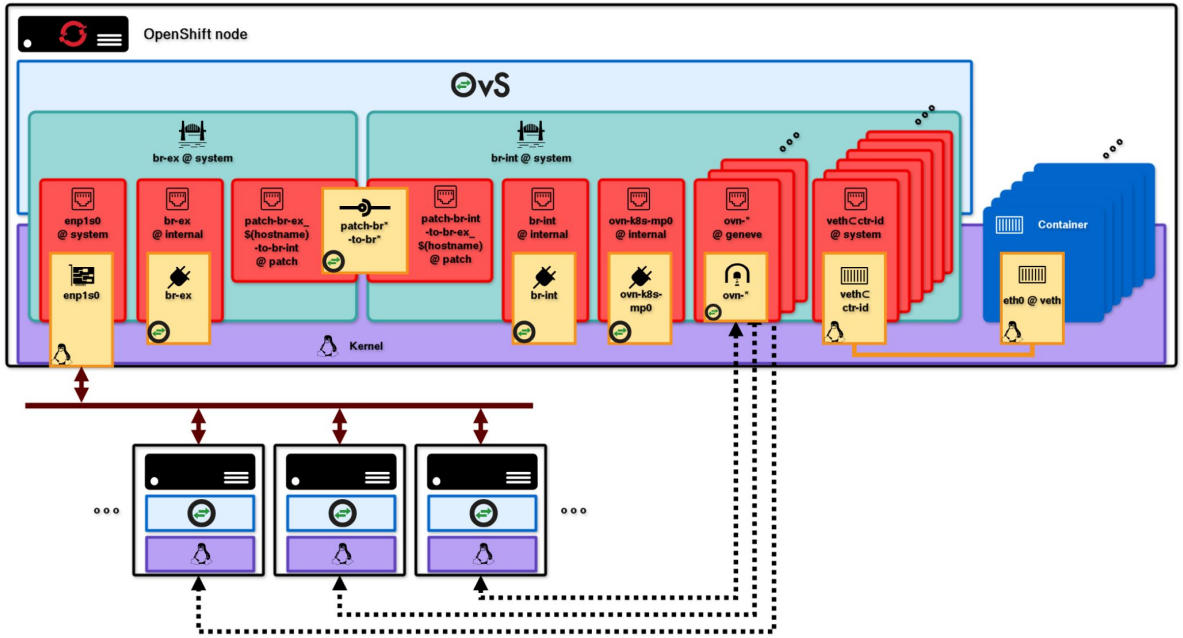
Network of an OpenShift node



Network of an OpenShift node

OVN's external bridge ***br-ex***, type is ***system***

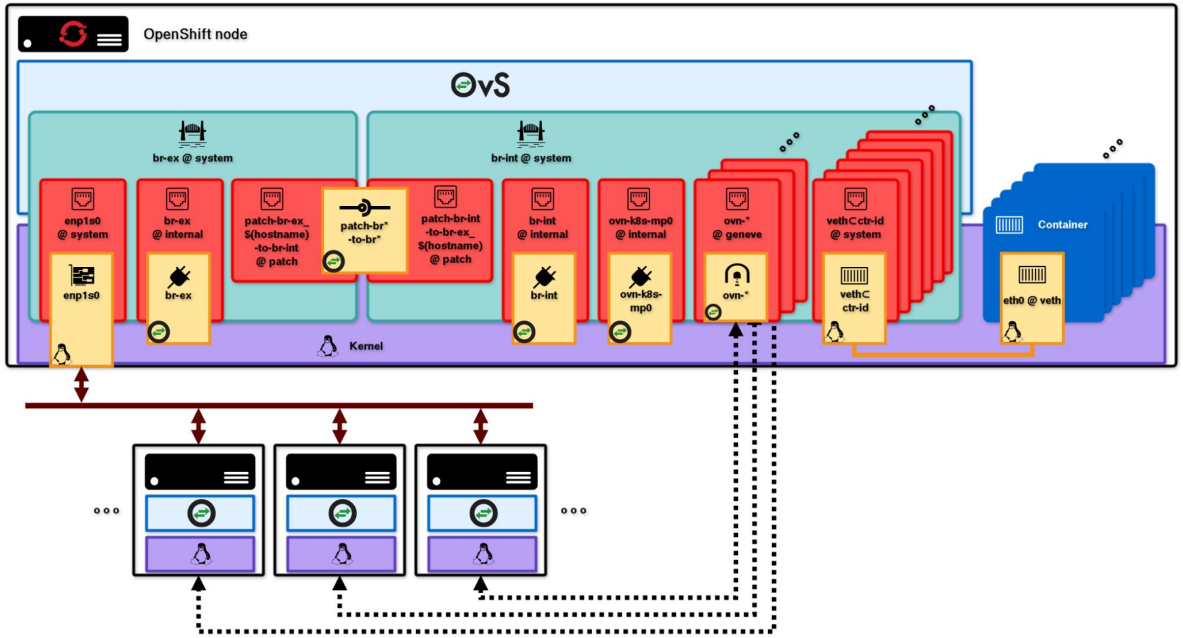
- Managed by NM connection ***br-ex***
- OVS system port, physical NIC and default gateway interface ***enp1s0*** (default network interfaces := preconfigured or if not, has default gateway). Managed by NM connections ***ovs-port-phys0*** and ***ovs-if-phys0***.
- OVS internal port ***br-ex*** for node connectivity, i.e. MAC address, DHCP Client ID and IP addresses of ***enp1s0***, but also IP address pool(s) for services (Default is 172.30.0.0/16). Managed by NM connections ***ovs-port-br-ex*** and ***ovs-if-br-ex***.
- OVS patch port ***patch-br-ex_$(hostname)-to-br-int***, peer=***patch-br-int-to-br-ex_$(hostname)***



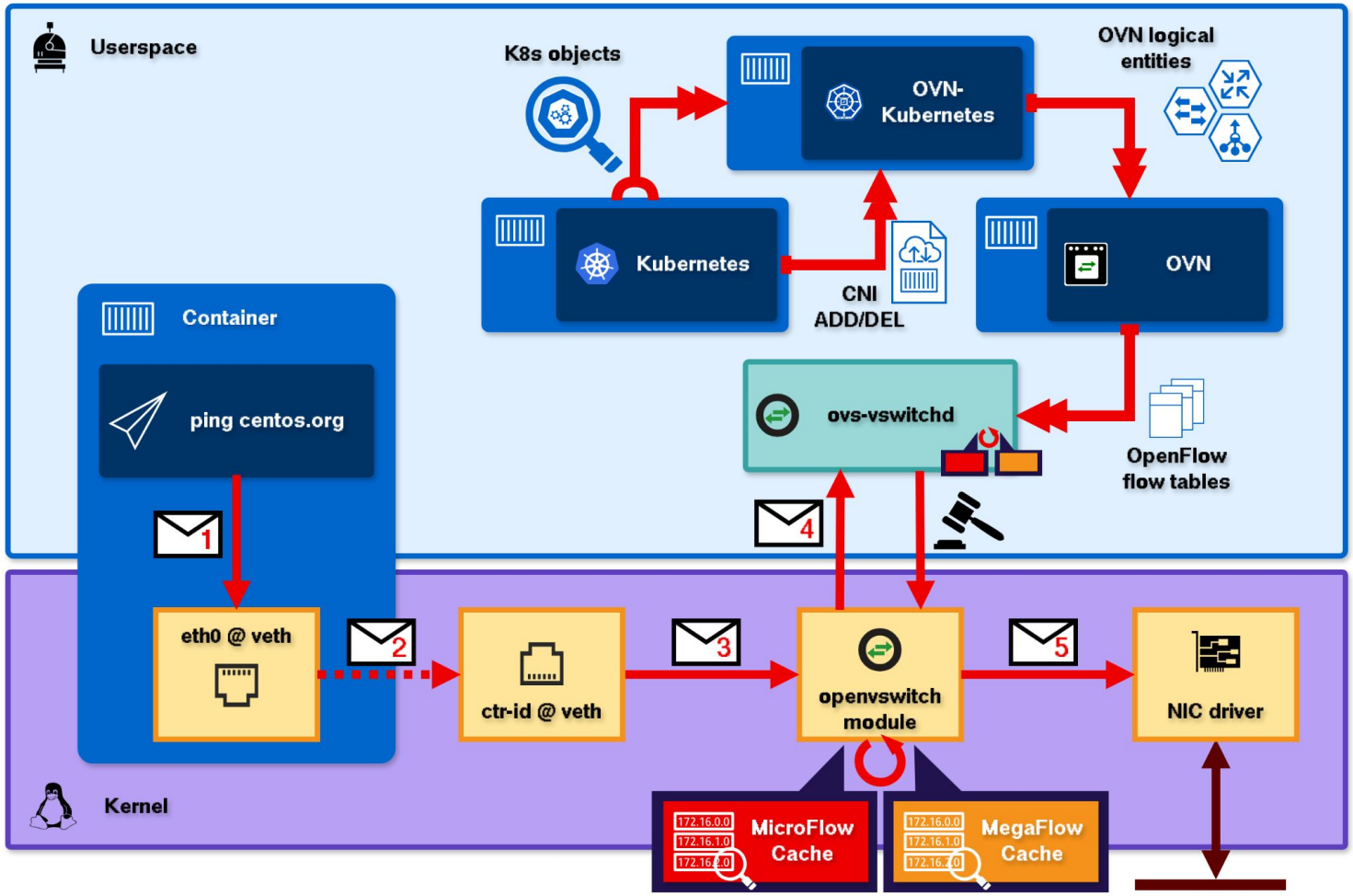
Network of an OpenShift node

OVN's integration bridge ***br-int***, type is ***system***

- Managed by OVN and OVN-Kubernetes.
- OVS patch port ***patch-br-int-to-br-ex_$(hostname)***, peer=***patch-br-ex_$(hostname)-to-br-int***
- OVS geneve ports for Geneve tunnels to all other control plane and worker nodes, for pod traffic between OCP nodes
- OVS internal port ***ovn-k8s-mp0***, access to OVN overlay network for pods (default is 10.128.0.0/14 with host prefix /23)
- OVS internal port ***br-int***
- OVS system ports for network connectivity in pods with veth devices. OVN-K8s derives OVS port name and interface name from a pod's sandbox id.

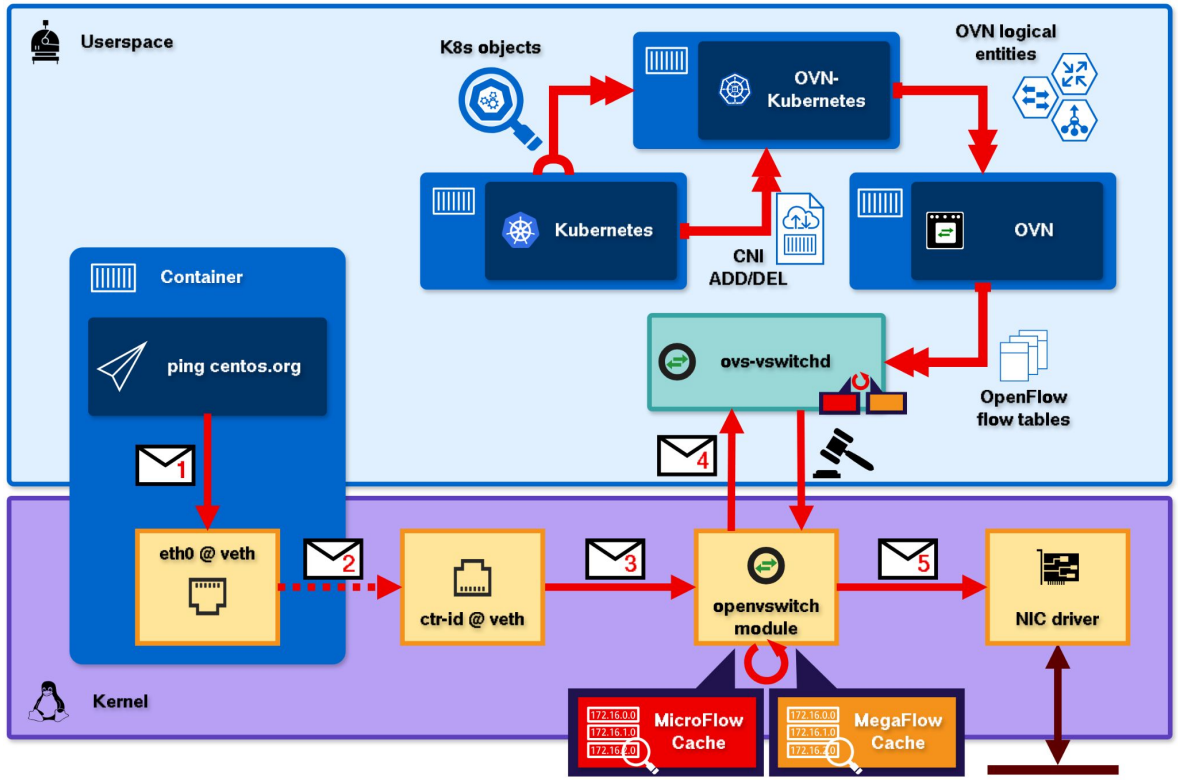


Pod-to-WWW network interface traversal at an OpenShift node



Pod-to-WWW network interface traversal at an OpenShift node

1. In a pod/container, a packet is send over veth device ***eth0***.
2. veth peer is attached to OVS bridge ***br-int*** as a system port whose port name is derived from the pod's sandbox id.
3. packet is passed to kernel module ***openvswitch***.
4. ***openvswitch*** kernel module classifies the packet to decide what actions it should do, i.e. it searches for a matching rule in its Microflow and Megaflow caches.
5. packet is passed to default network interface ***enp1s0***, handled by NIC kernel driver and transmitted on the wire.



OpenShift with DPDK datapaths and VDUSE in OVN-Kubernetes

RHEL/RHCOS

- Develop, merge and ship VDUSE support in RHEL Kernel and Open vSwitch

OVN-Kubernetes

- Creates VDUSE devices instead of veth devices for pod networking when userspace bridges are being detected.
- Challenge:
  Change control flow from
  "*set up veth device pair in container netns with ips, gateways, routes, and finally attach veth peer as OVS system port to br-int*"
  to
  "*Create OVS dpdkvhostuserclient port at br-int first, then set up vDPA netdev in container ns with ips, gateways, routes*" while keeping support for both datapath types.
- commits/diff

OpenShift with DPDK datapaths and VDUSE in OVN-Kubernetes

Machine Config Operator

- new script ***prepare-ovs.sh*** and ***ovs-preparation.service***:
  - inserts ***vduse*** and ***virtio-vdpa*** kernel modules, and
  - enables (or disables) DPDK and userspace TSO in OVS when kernel arg ***ovs-dpdk-vduse-poc*** is detected
  - runs after ***ovs-dbserver.service*** but before ***ovs-vswitchd.service***
- commits/diff

- changed ***configure-ovs.sh***:
  - create OVS bridge ***br-ex*** with type ***netdev*** instead of ***system*** when kernel arg ***ovs-dpdk-vduse-poc*** is set,
  - optionally unbinds NIC (e.g. ***enp1s0***) from current kernel driver and binds it to DPDK-compatible kernel driver like ***vfio-pci*** when kernel argument ***ovs-dpdk-bind=enp1s0,vfio-pci*** is set,
  - attach default gateway interface (e.g. ***enp1s0***) to OVS bridge using DPDK when device is allow-listed with kernel arg ***ovs-dpdk-bind=enp1s0***
  - enable OVS-DPDK PMD thread load-based sleeping when kernel arg ***ovs-pmd-sleep-max*** is set.

Network configuration during OpenShift node boot

Default release

- ovsdb-server.service and ovs-vswitchd.service
- NetworkManager.service, *-wait-online.service

⏳ ...

- ovs-configuration.service / configure-ovs.sh

⏳ ...

- kubelet.service / kubelet.sh
- crio.service

PoC with DPDK and VDUSE

- ovsdb-server.service
- NetworkManager.service, *-wait-online.service

⏳ ...

- ovs-preparation.service / prepare-ovs.sh ⚠️

⏳ ...

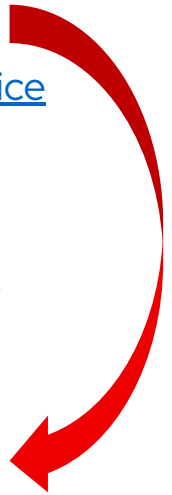
- ovs-vswitchd.service

⏳ ...

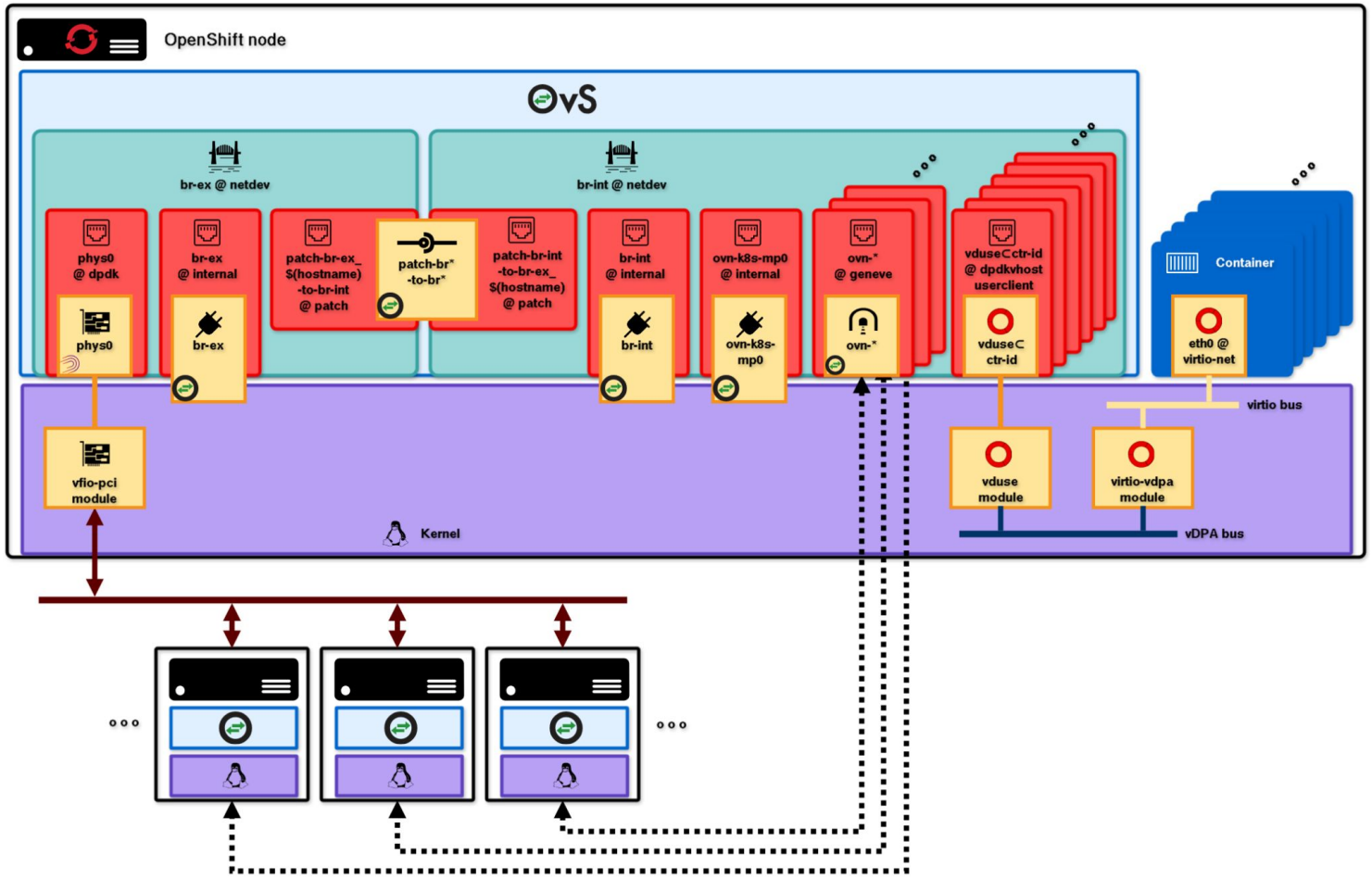
- ovs-configuration.service / configure-ovs.sh

⏳ ...

- kubelet.service / kubelet.sh
- crio.service



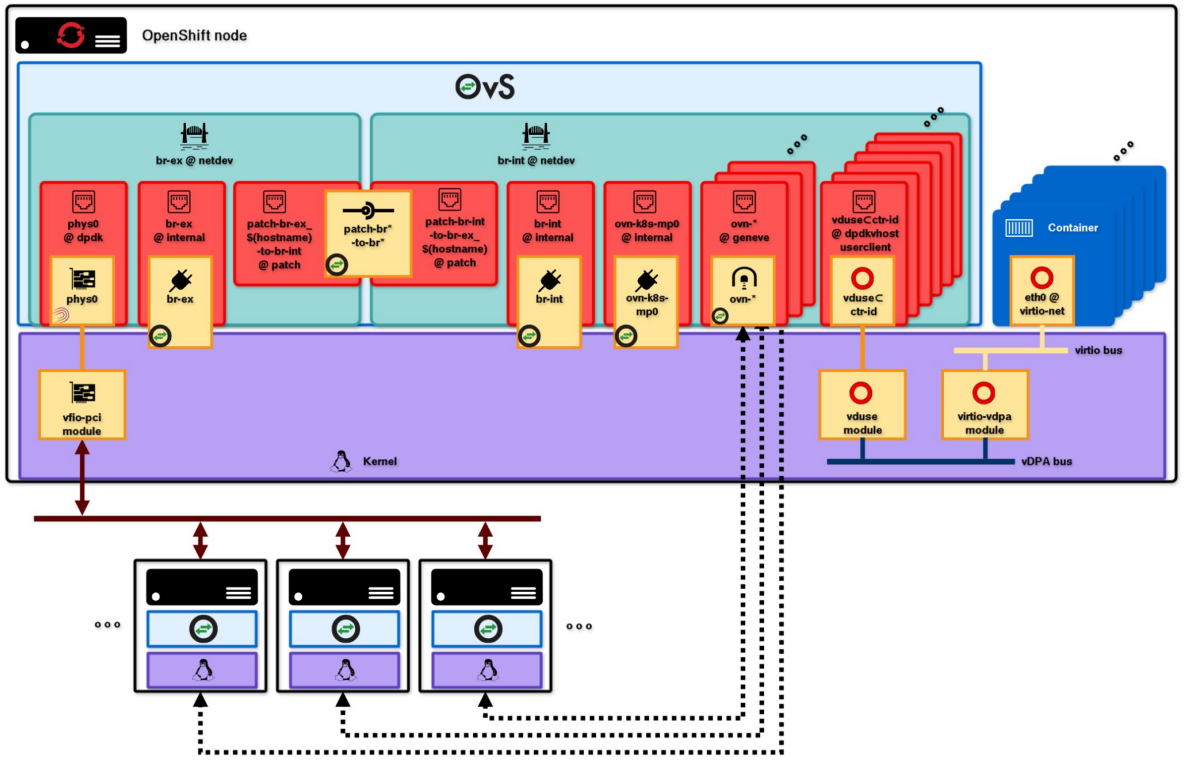
Network of an OpenShift node with DPDK and VDUSE



Network of an OpenShift node with DPDK and VDUSE

OVN's external bridge ***br-ex***, type is ***netdev***

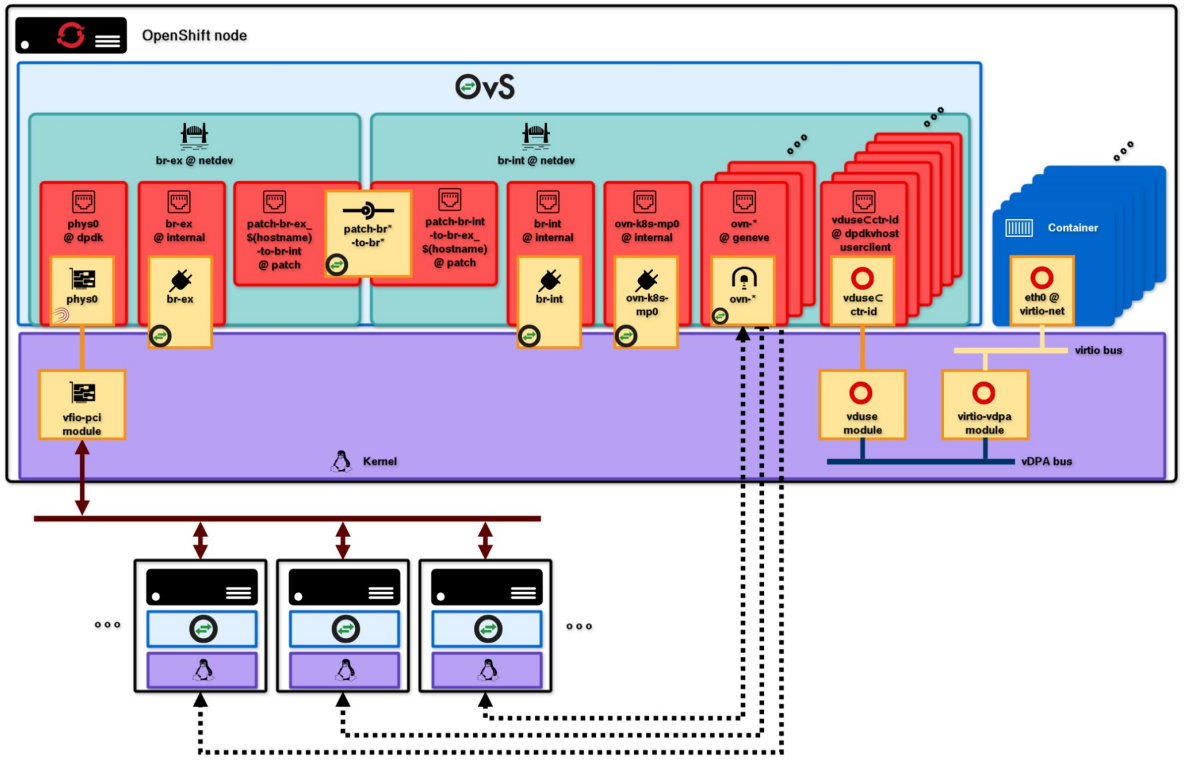
- Managed by NM connection ***br-ex***
- OVS dpdk port ***phys0***, PCI address of physical NIC which previously has been identified as default gateway interface ***enp1s0***. Managed by NM connections ***ovs-port-phys0*** and ***ovs-if-phys0***.
- OVS internal port ***br-ex*** for node connectivity, i.e. MAC address, DHCP Client ID and IP addresses of ***enp1s0***, but also IP address pool(s) for services (Default is 172.30.0.0/16). Managed by NM connections ***ovs-port-br-ex*** and ***ovs-if-br-ex***.
- OVS patch port ***patch-br-ex_$(hostname)-to-br-int***, peer=***patch-br-int-to-br-ex_$(hostname)***



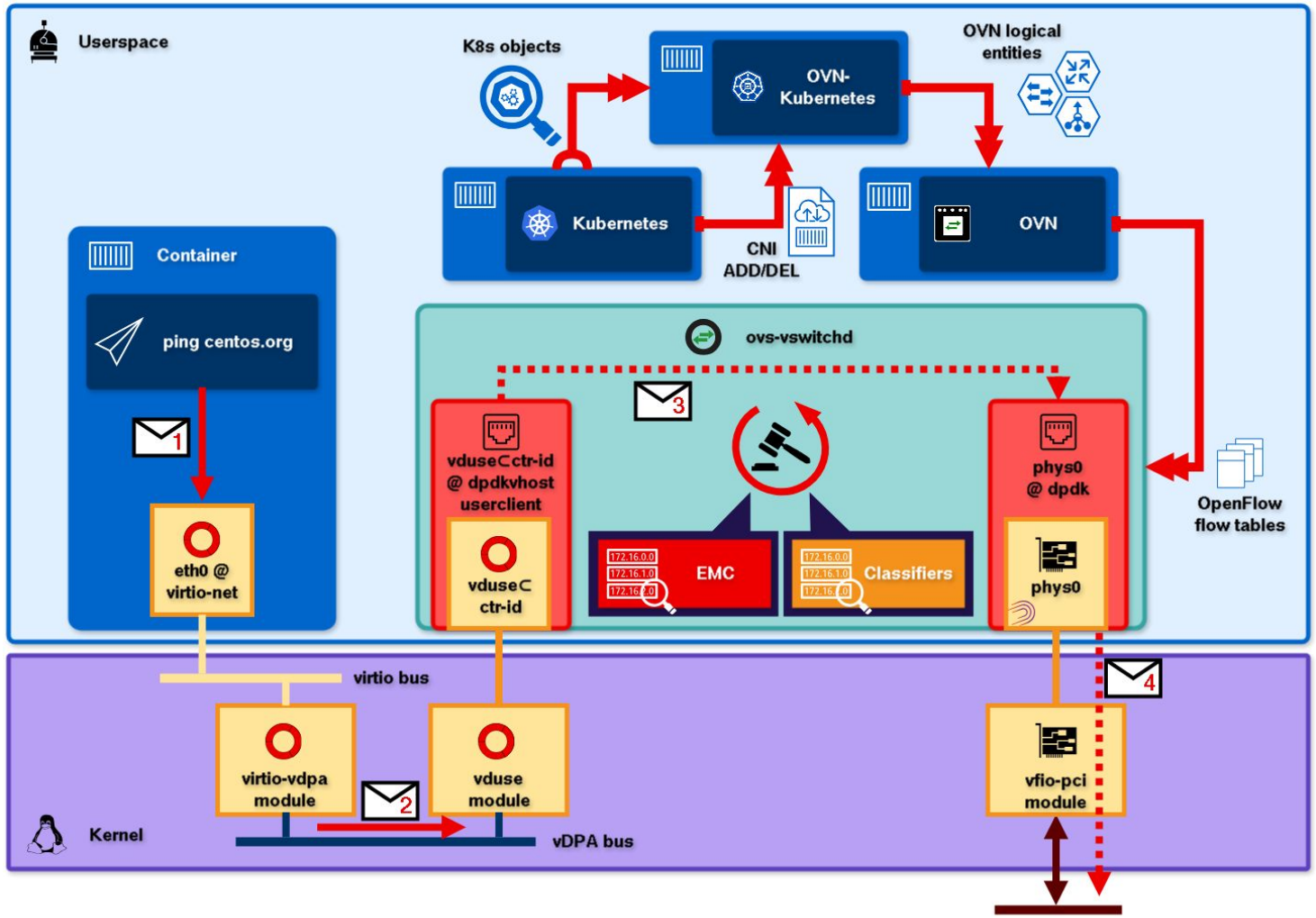
Network of an OpenShift node with DPDK and VDUSE

OVN's integration bridge ***br-int***, type is ***netdev***

- Managed by OVN and OVN-Kubernetes.
- OVS patch port ***patch-br-int-to-br-ex_$(hostname)***, peer=***patch-br-ex_$(hostname)-to-br-int***
- OVS geneve ports for Geneve tunnels to all other control plane and worker nodes, for pod traffic between OCP nodes
- OVS internal port ***ovn-k8s-mp0***, access to OVN overlay network for pods (default is 10.128.0.0/14 with host prefix /23)
- OVS internal port ***br-int***
- OVS dpdkvhostuserclient ports for network connectivity in pods with VDUSE devices. OVN-K8s derives OVS port name and interface name from a pod's sandbox id.

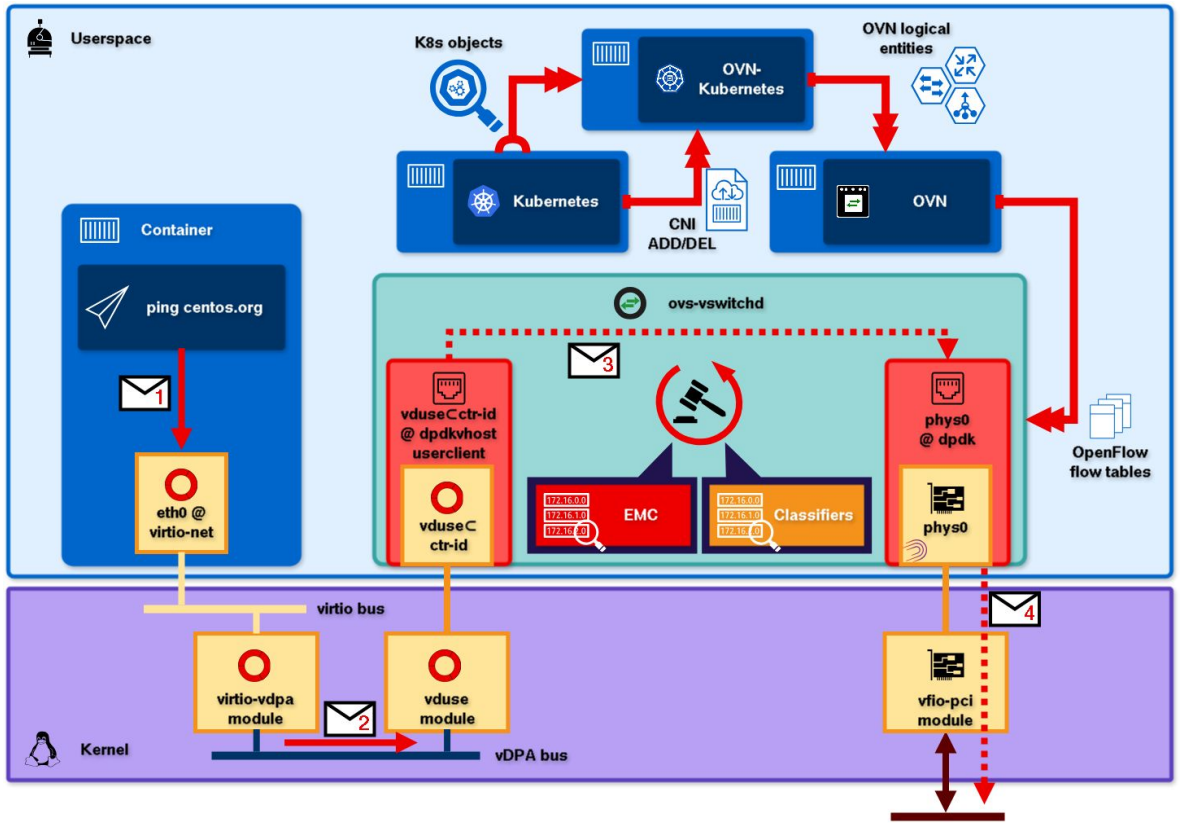


Pod-to-WWW network interface traversal with DPDK and VDUSE



Pod-to-WWW network interface traversal with DPDK and VDUSE

1. In a pod/container, a packet is send over vDPA netdev ***eth0***.
2. Its "peer" VDUSE device is attached via vDPA bus to OVS bridge ***br-int*** as a dpdkvhostuserclient port whose port name is derived from the pod's sandbox id.
3. ovs-vswitchd classifies the packet to determine the forwarding decisioning.
4. OVS transmits packet on the wire.

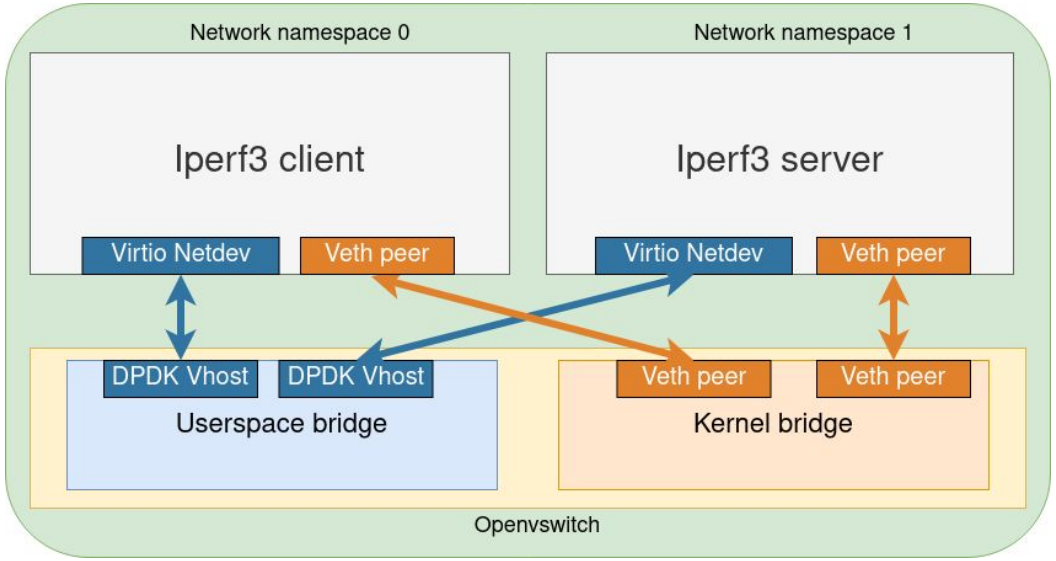


Benchmarks

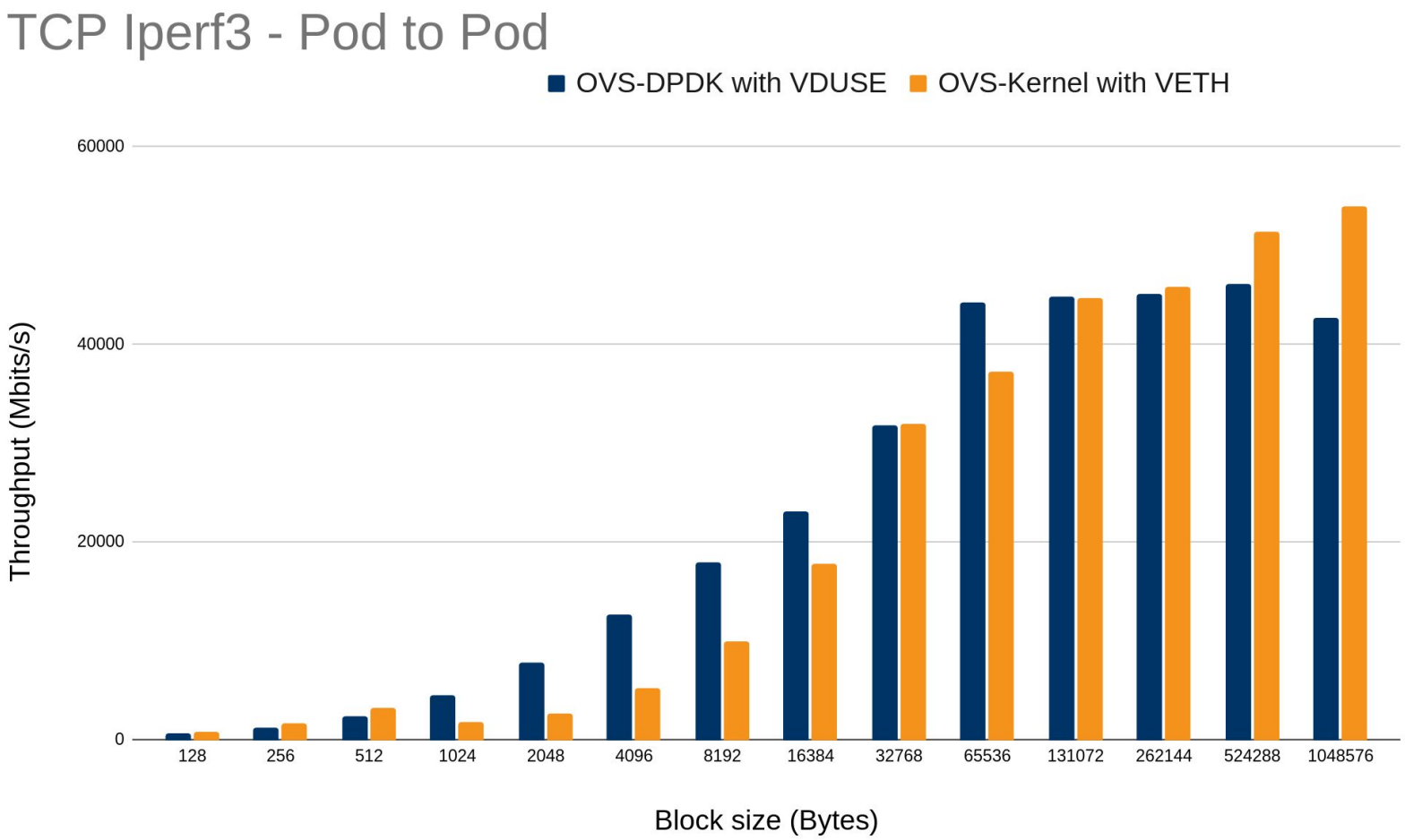
TCP benchmark

Comparison of TCP throughput between userspace and kernel datapath between two pods using iperf3:

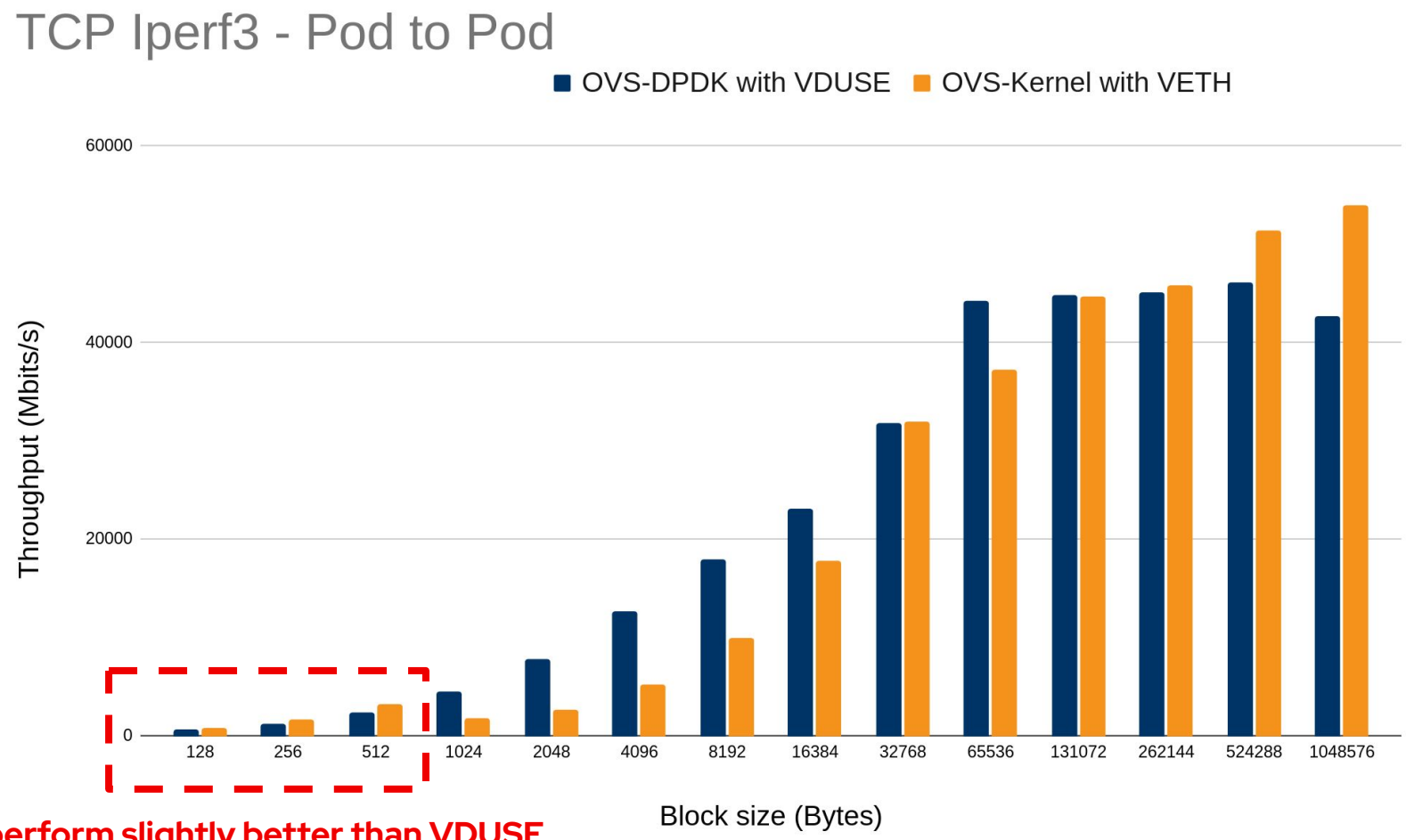
- Intel(R) Xeon(R) Gold 6438N
- RHEL 9.6 Kernel
- PMD threads and VDUSE virtqueues IRQ handlers pinned to isolated CPUs
- 1GB hugepages
- OVS v3.4+



TCP benchmark

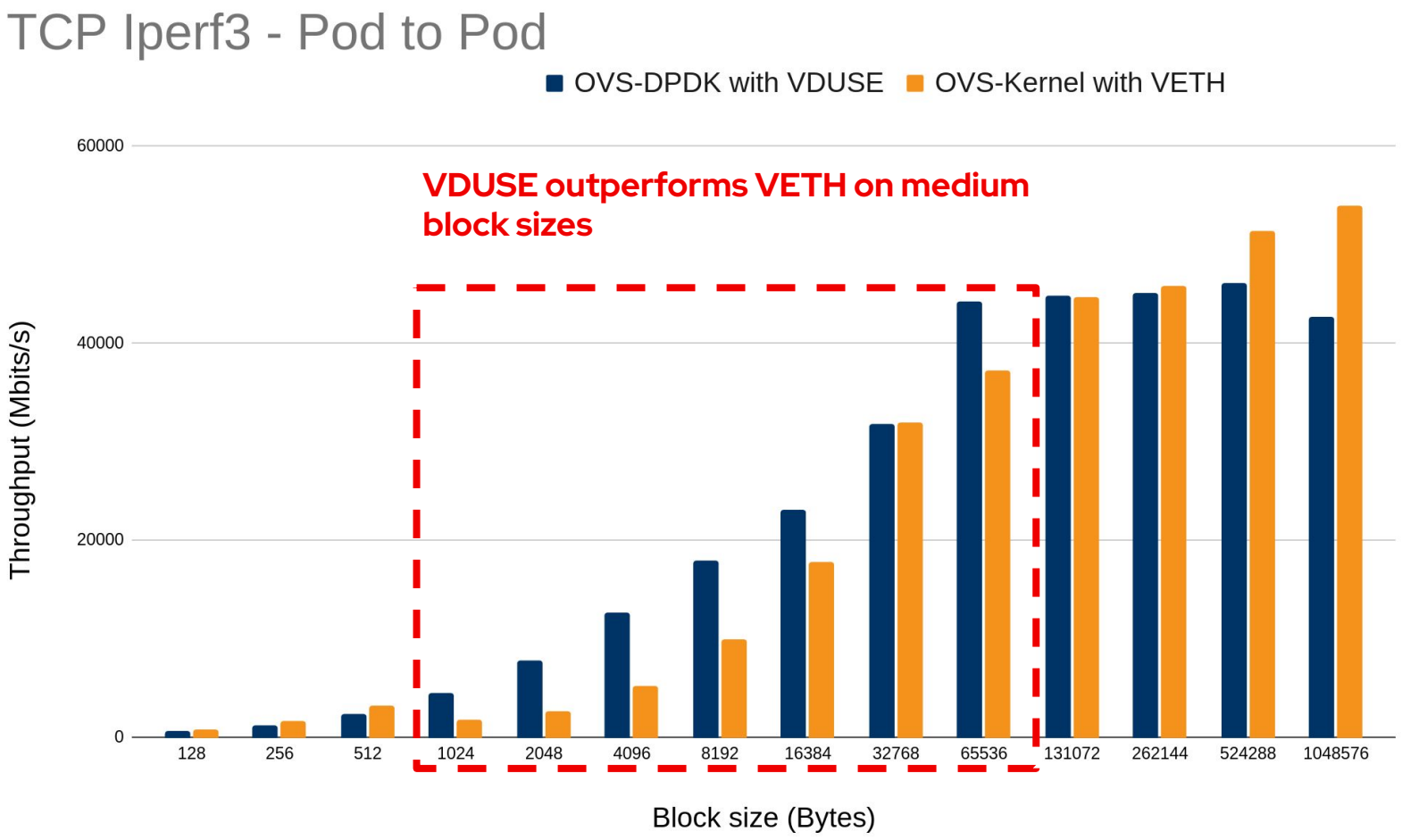


TCP benchmark

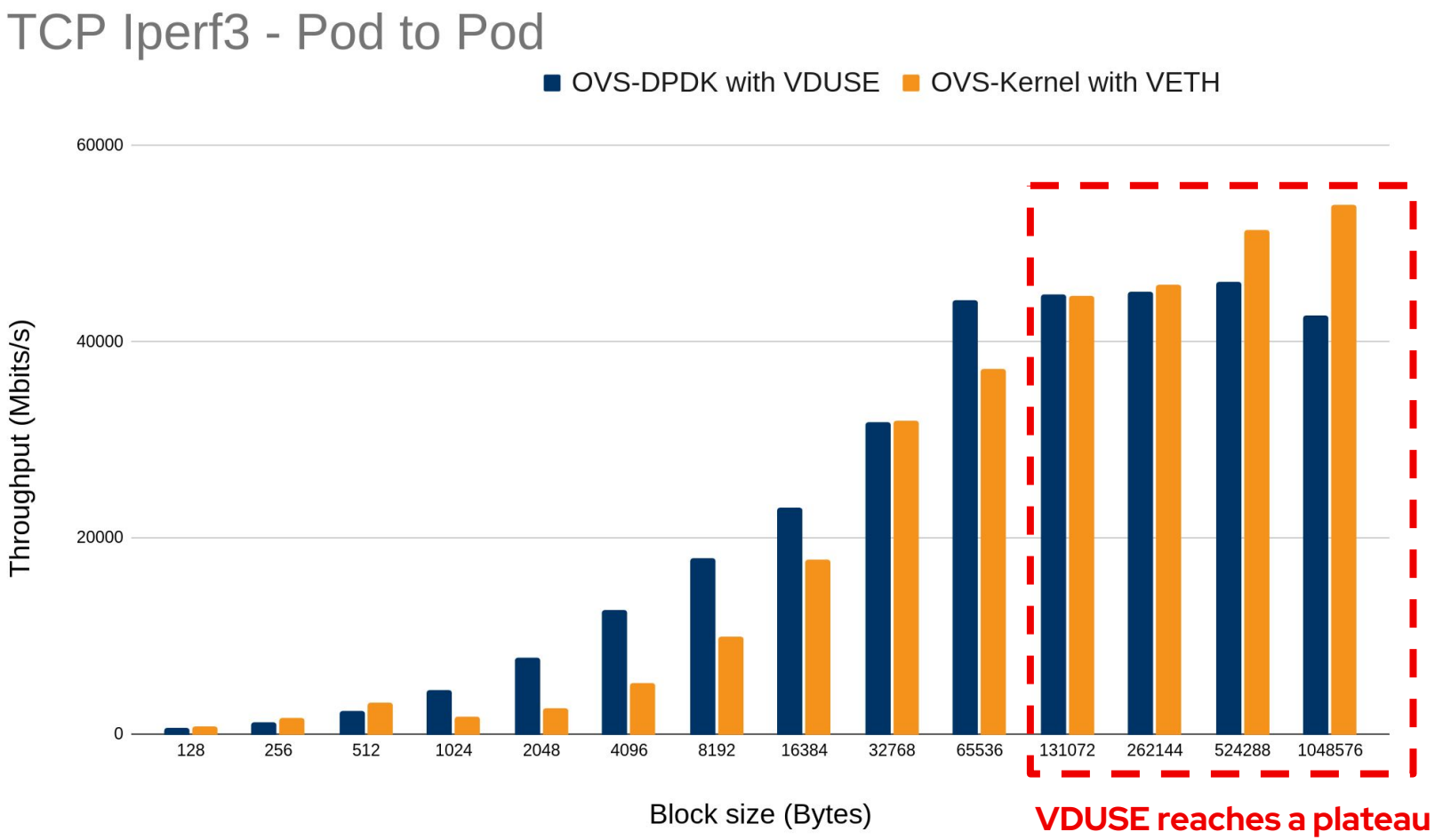


VETH perform slightly better than VDUSE on smaller block sizes

TCP benchmark



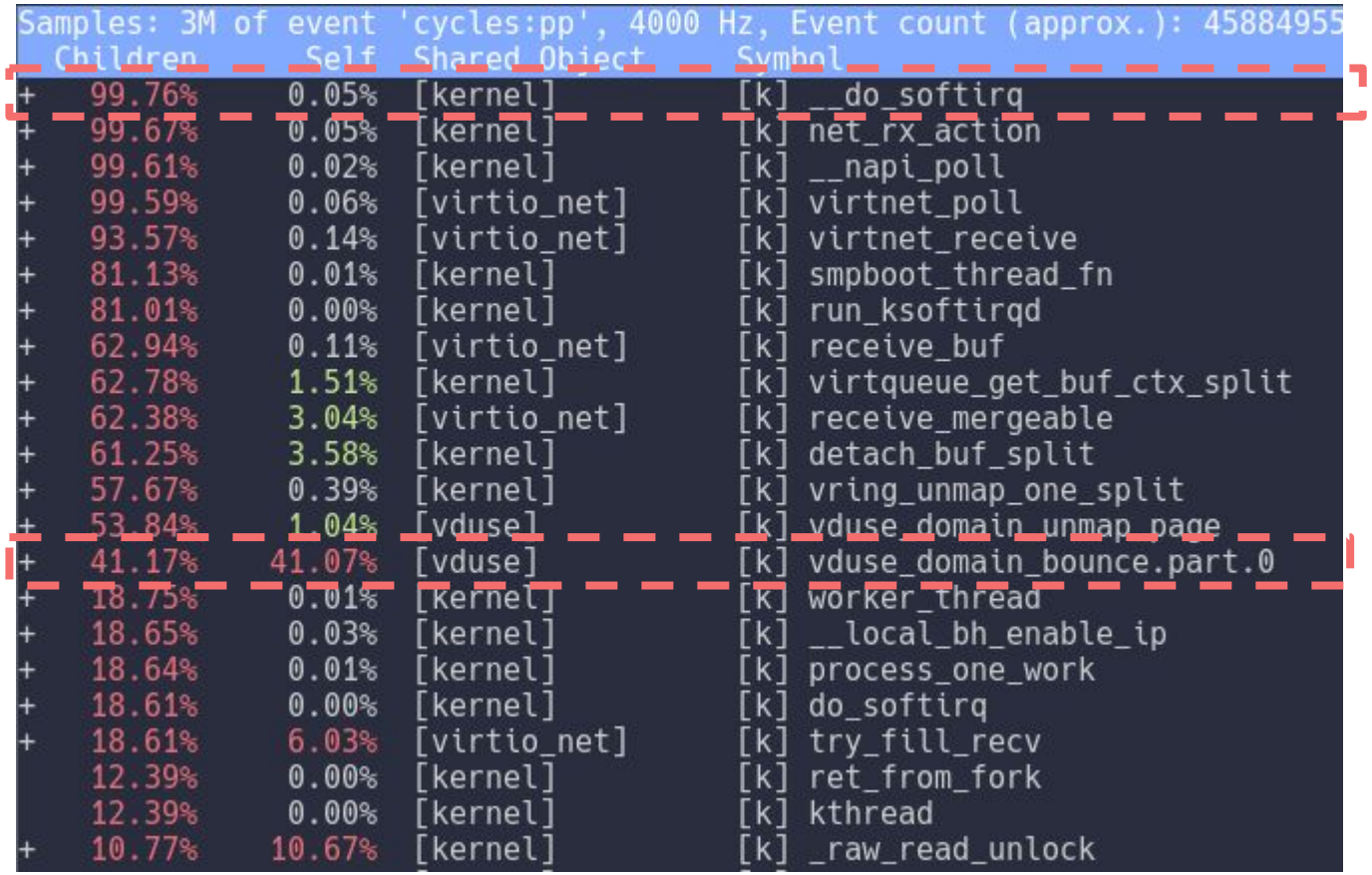
TCP benchmark



TCP benchmark

Running perf top on the Rx virtqueue interrupt handler CPU when benchmarking large blocks, we notice that:

- No idle, the CPU is 100% busy handling Rx packets
- ~40% of the cycles spent in the bounce buffer copy function
- **Ongoing investigations to remove bounce buffer copies, or improve its performance**



```
Samples: 3M of event 'cycles:pp', 4000 Hz, Event count (approx.): 45884955
  Children      Self  Shared Object     Symbol
+   99.76%     0.05%  [kernel]          [k] __do_softirq
+   99.67%     0.05%  [kernel]          [k] net_rx_action
+   99.61%     0.02%  [kernel]          [k] __napi_poll
+   99.59%     0.06%  [virtio_net]      [k] virtnet_poll
+   93.57%     0.14%  [virtio_net]      [k] virtnet_receive
+   81.13%     0.01%  [kernel]          [k] smpboot_thread_fn
+   81.01%     0.00%  [kernel]          [k] run_ksoftirqd
+   62.94%     0.11%  [virtio_net]      [k] receive_buf
+   62.78%     1.51%  [kernel]          [k] virtqueue_get_buf_ctx_split
+   62.38%     3.04%  [virtio_net]      [k] receive_mergeable
+   61.25%     3.58%  [kernel]          [k] detach_buf_split
+   57.67%     0.39%  [kernel]          [k] vring_unmap_one_split
+   53.84%     1.04%  [vduse]           [k] vduse_domain_unmap_page
+   41.17%    41.07%  [vduse]           [k] vduse_domain_bounce.part.0
+   18.75%     0.01%  [kernel]          [k] worker_thread
+   18.65%     0.03%  [kernel]          [k] __local_bh_enable_ip
+   18.64%     0.01%  [kernel]          [k] process_one_work
+   18.61%     0.00%  [kernel]          [k] do_softirq
+   18.61%     6.03%  [virtio_net]      [k] try_fill_recv
    12.39%     0.00%  [kernel]          [k] ret_from_fork
    12.39%     0.00%  [kernel]          [k] kthread
+   10.77%    10.67%  [kernel]          [k] _raw_read_unlock
```


Conclusion

Preliminary evaluation

Pros

- More determinism thanks to CPU pinning
- VDUSE offer higher performance than veth devices for small/medium buffer sizes
- Thanks to VDUSE, we can provide connectivity to both primary and secondary networks, including Kubevirt
- Better partitioning achieved by isolating network infra on isolated CPUs

Cons

- Higher memory bandwidth usage because of the bounce buffer copies.
- Increased complexity, i.e. extra kernel args, enable workload partitioning, reserve hugepages, NUMA tuning. Can partially be mitigated by proper product integration and documentation.
- Increased CPU utilization due to PMDs but can be lessened by PMD thread load-based sleeping (which increases wakeup latency).

Future work

- Improve VDUSE **performance** for large block sizes, e.g. remove or improve bounce buffer (ongoing work)
- Improve handling of hugepages in OCP w.r.t. to **system-reserved hugepages**
- **NUMA awareness** of OVS in OCP
- (Re)enable ***SELinux*** (currently not supported)
- Replace kernel driver (un)binding with ***driverctl*** integration
- Replace karg activation method and prepare-ovs.sh script with ***declarative configuration***
- Adjustments to **User-Defined Networks** (UDNs) (enhancement proposal, OCP 4.17 Tech Preview docs)
- Migration to **configure-ovs.sh alternative in OpenShift** based on kubernetes-nmstate
- Run OpenShift's conformance test suite (again), investigate **sig-network failures** and **ovs-vswitchd.log**
- **Benchmark, Tune, Repeat**
- **Document**, contribute and merge **upstream**, **productize**

Thank you

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

x.com/RedHat

Red Hat