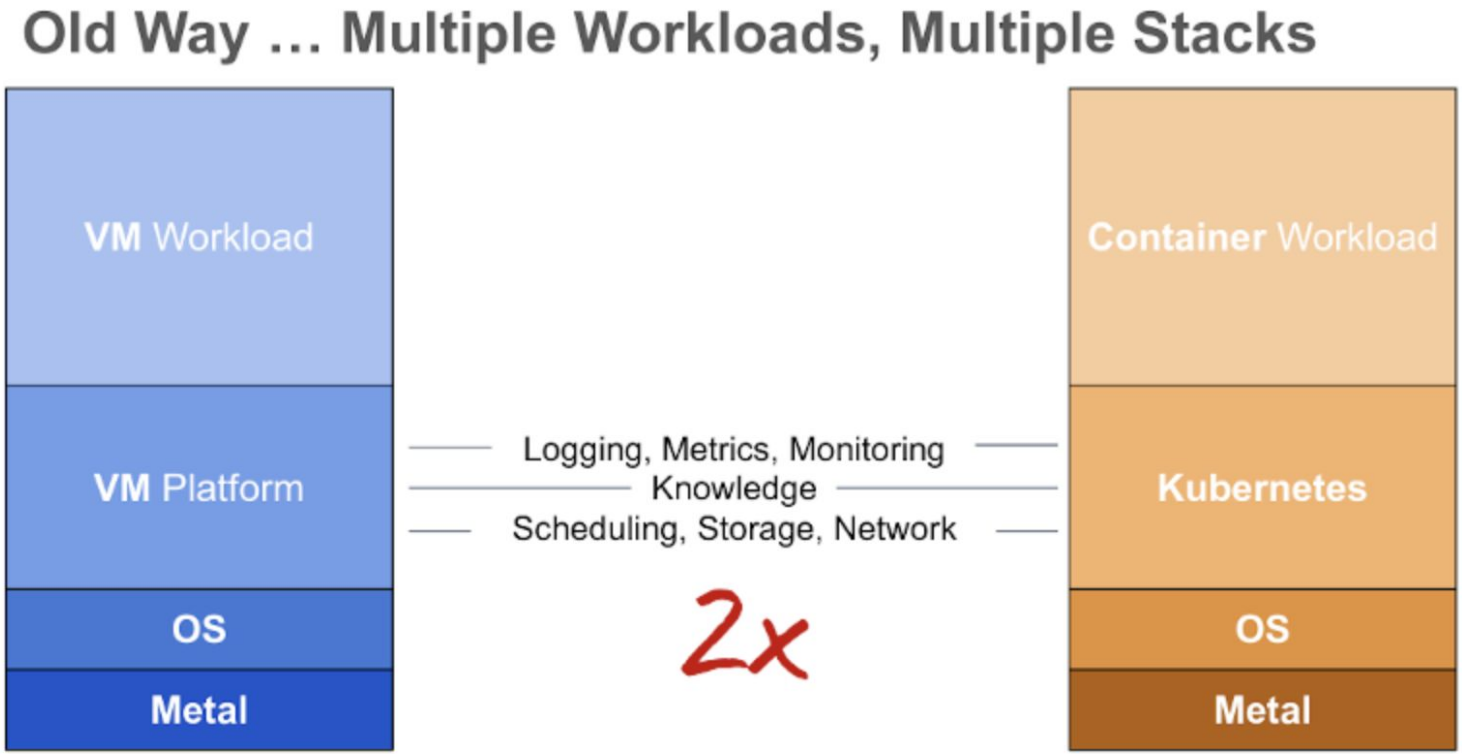
netkit for VM workloads

Daniel Borkmann (Isovalent at Cisco)

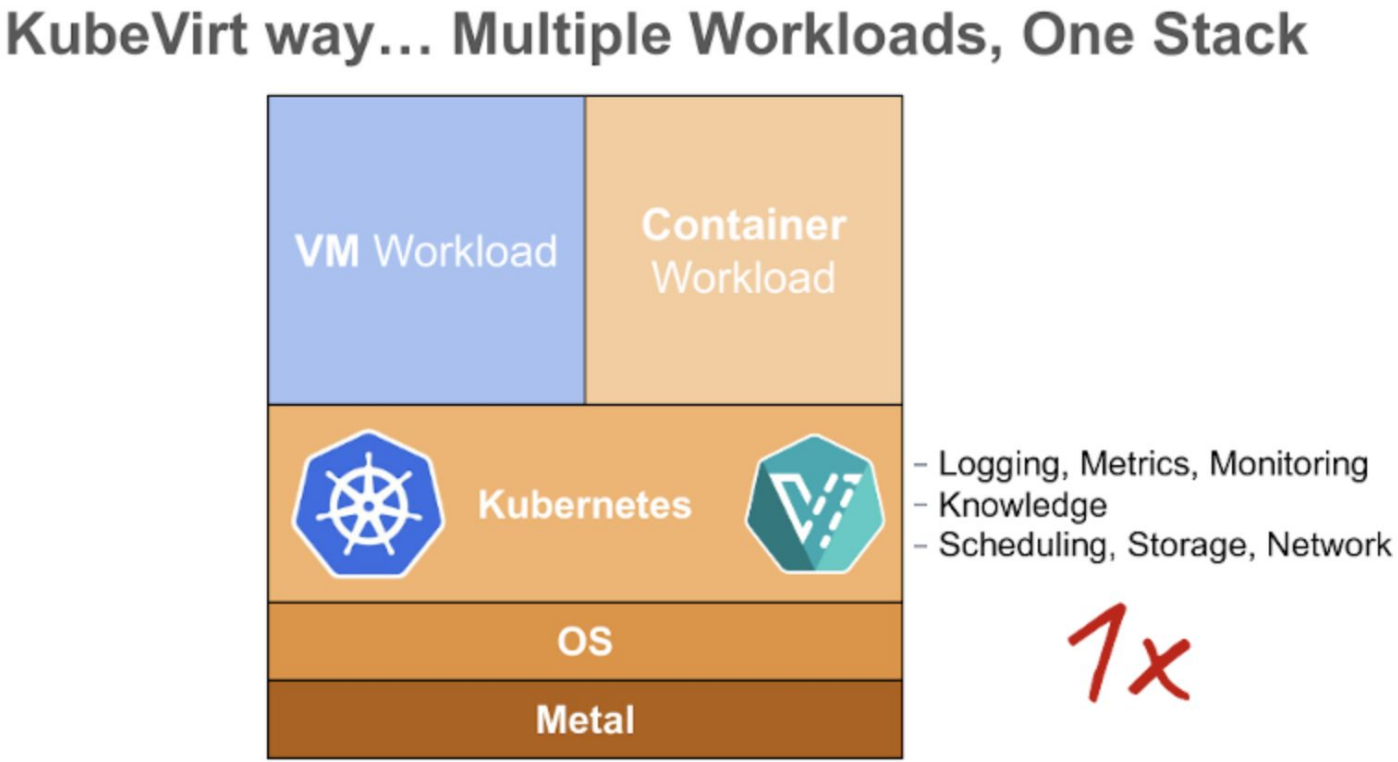
LSF/MM/BPF 2025

Infrastructure Convergence



img: kubevirt.io

Infrastructure Convergence



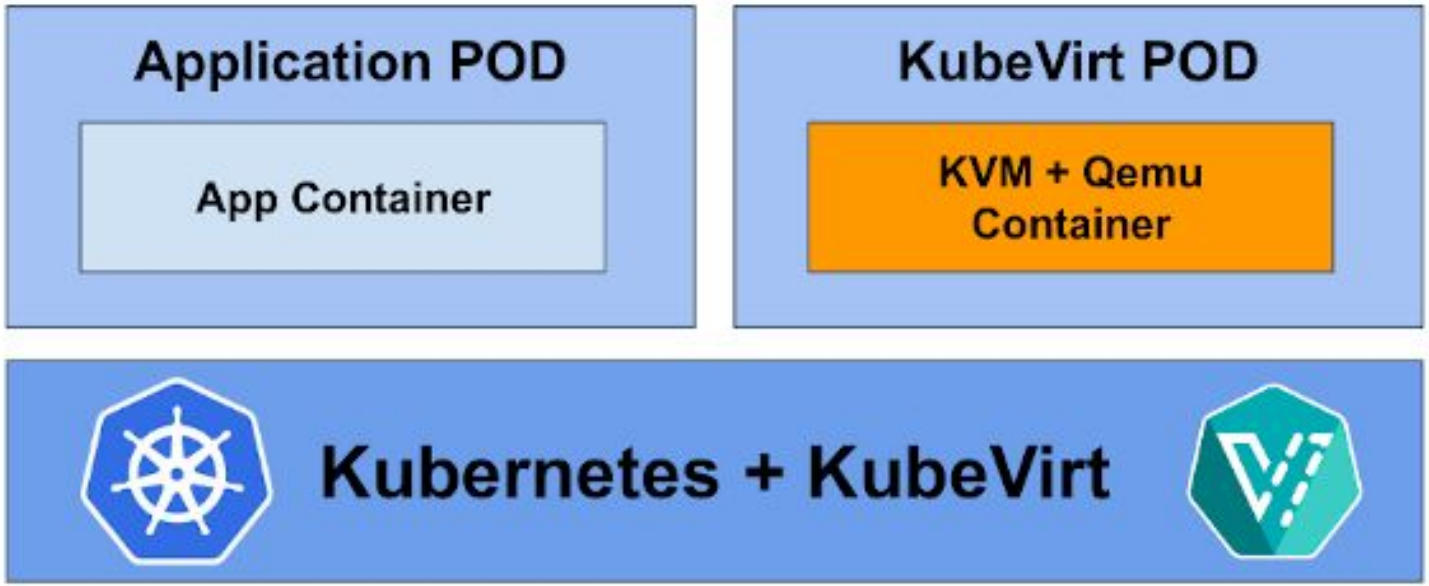
img: kubevirt.io

Infrastructure Convergence

The workflow convergence means that:

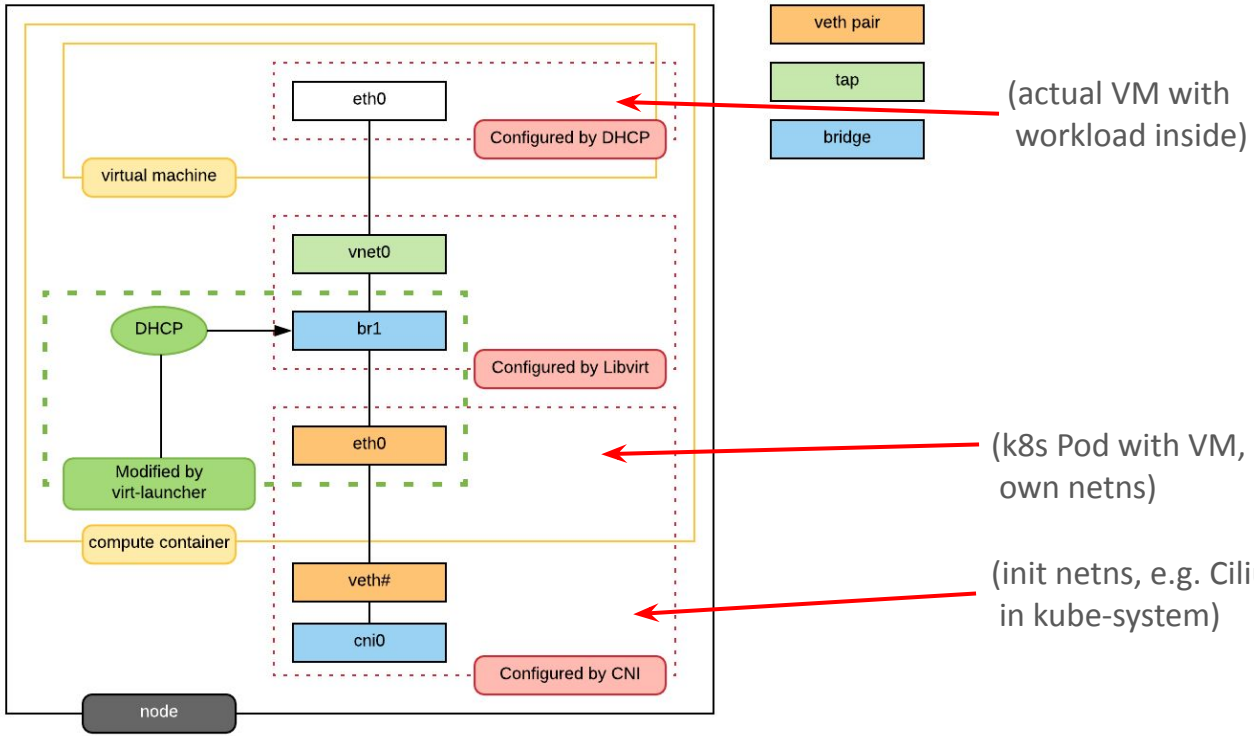
- Converging VM management into container management workflows
- Using same tooling (kubectl) for containers and Virtual Machines
- Keeping the declarative API for VM management (just like Pods, deployments, etc)

KubeVirt Overview



img: kubevirt.io

Networking PoV of KubeVirt Pod today



img: kubevirt.io

qemu’s AF_XDP backend

QEMU / QEMU / Commits / **cb039ef3**

Commit `cb039ef3` authored 1 year ago by **Ilya Maximets** Committed by **Jason Wang** 1 year ago

net: add initial support for AF_XDP network backend

```
AF_XDP is a network socket family that allows communication directly
with the network device driver in the kernel, bypassing most or all
of the kernel networking stack.  In the essence, the technology is
pretty similar to netmap.  But, unlike netmap, AF_XDP is Linux-native
and works with any network interfaces without driver modifications.
Unlike vhost-based backends (kernel, user, vdpa), AF_XDP doesn't
require access to character devices or unix sockets.  Only access to
the network interface itself is necessary.
```

qemu’s AF_XDP backend

Usage example:

```
-device virtio-net-pci,netdev=guest1,mac=00:16:35:AF:AA:5C

-netdev af-xdp,ifname=ens6f1np1,id=guest1,mode=native,queues=2,start-queue=14
```

Could KubeVirt benefit from qemu’s AF_XDP backend?

- KubeVirt Pods are in their own netns, like regular Pods
- AF_XDP support from inside netns very limited
- Only supported by veth, but performance is slow (hearsay)

Cilium Datapath for accelerating KubeVirt

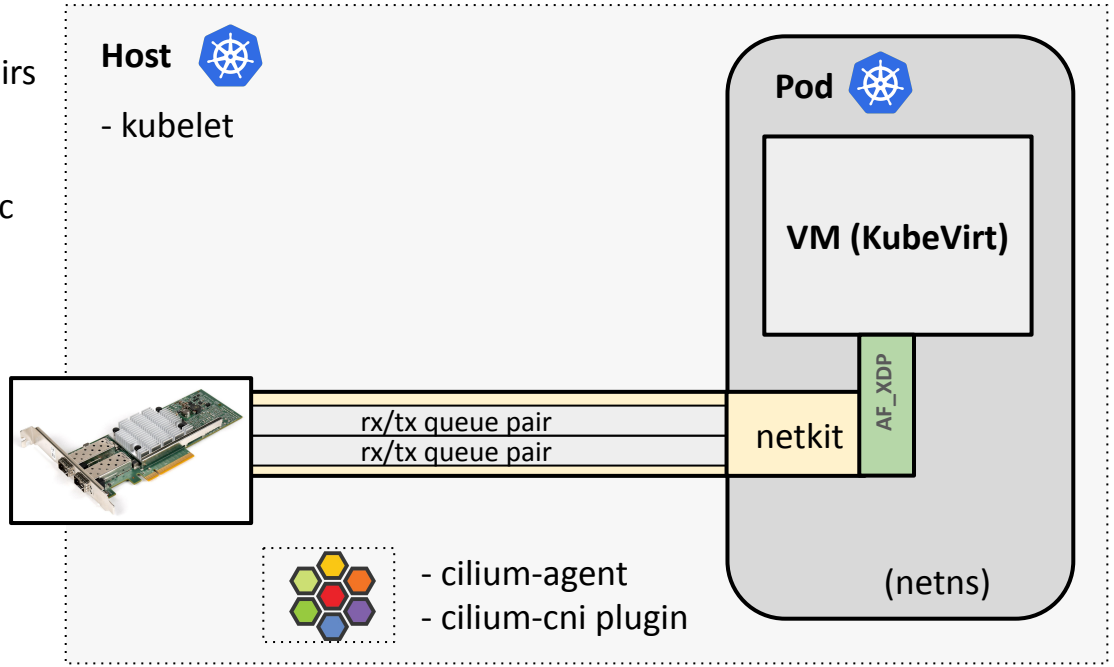
Idea:

- Reserve a set of RX/TX queue pairs from phys dev and bind to netkit

- Set up RSS context to steer traffic to the queue set

- netkit then implements ndo's in order to set up AF_XDP ring buffers for the set of queues

- Allowing native AF_XDP performance & still be able to enforce policy/introspect/etc via BPF controlled from host (no need for SRIOV then)



netkit PoC for AF_XDP

```
# ethtool -i enp10s0f0np0
driver: mlx5_core
version: 6.14.0-rc5+
firmware-version: 22.32.2004 (MT_0000000436)
[...]

# ethtool -l enp10s0f0np0
Channel parameters for enp10s0f0np0:
Pre-set maximums:
RX:             n/a
TX:             n/a
Other:                  n/a
Combined:       16
Current hardware settings:
RX:             n/a
TX:             n/a
Other:                  n/a
Combined:       16
```

Workflow example:

- Control plane (e.g. Cilium) slices NIC to reserve RX/TX queue pairs for netkit / KubeVirt Pod

netkit PoC for AF_XDP

Workflow example:

- Control plane (e.g. Cilium) slices NIC to reserve RX/TX queue pairs for netkit / KubeVirt Pod

```
Reduce default RSS context to just 14 queues for host traffic:

   # ethtool -X enp10s0f0np0 start 0 equal 14
   # ethtool -x enp10s0f0np0

Create new RSS context (1) for 2 queue pairs (14,15) for netkit (aka VM) traffic:

   # ethtool -X enp10s0f0np0 start 14 equal 2 context new
   New RSS context is 1
   # ethtool -x enp10s0f0np0 context 1
```

netkit PoC for AF_XDP

author Maxim Mikityanskiy <maxtram95@gmail.com> 2025-03-19 14:45:08 +0200
committer Jakub Kicinski <kuba@kernel.org> 2025-03-24 13:39:15 -0700
commit 3865bec60683b86d39a5d8d6c34a1d269adaa84c (patch)
tree 38b51f7981b9eb5404339308709da7b44bbc0e33
parent 81273eb87af86d4a43244b553762348e364b2df7 (diff)
download net-3865bec60683b86d39a5d8d6c34a1d269adaa84c.tar.gz

net/mlx5e: Fix ethtool -N flow-type ip4 to RSS context

Workflow example:

- Control plane (e.g. Cilium) slices NIC to reserve RX/TX queue pairs for netkit / KubeVirt Pod
- Once IPAM is done, we set up steering rules for the VM's IPv4/6 address to the new RSS context (small mlx5 fix was needed to make it work)

```
# ethtool --config-ntuple enp10s0f0np0 flow-type ip4 dst-ip 1.1.1.1 context 1
Added rule with ID 1023
```

netkit PoC for AF_XDP

Workflow example:

- Control plane (e.g. Cilium) slices NIC to reserve RX/TX queue pairs for netkit / KubeVirt Pod
- Once IPAM is done, we set up steering rules for the VM's IPv4/6 address to the new RSS context (small mlx5 fix was needed to make it work)
- Cilium's CNI plugin sets up netkit bound with lower device & queue pairs, then moves peer into target netns

```
Add netkit bound to enp10s0f0np0 for queue pairs (14,15):

   # ip link add type netkit lower enp10s0f0np0 14 15

   # ip -d l
   [...]
   8: nk0@nk1: <BROADCAST,MULTICAST,NOARP,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
       link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff promiscuity 0 allmulti 0 minmtu 68 maxmtu 65535
       netkit mode l3 type peer policy forward peer policy forward lower enp10s0f0np0 14 15 numtxqueues 1 numrxqueues 1 gso_max_size
65536 gso_max_segs 65535 tso_max_size 524280 tso_max_segs 65535 gro_max_size 65536 gso_ipv4_max_size 65536 gro_ipv4_max_size 65536
   9: nk1@nk0: <BROADCAST,MULTICAST,NOARP,M-DOWN> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000
       link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff promiscuity 0 allmulti 0 minmtu 68 maxmtu 65535
       netkit mode l3 type primary policy forward peer policy forward lower enp10s0f0np0 14 15 numtxqueues 1 numrxqueues 1 gso_max_size
65536 gso_max_segs 65535 tso_max_size 524280 tso_max_segs 65535 gro_max_size 65536 gso_ipv4_max_size 65536 gro_ipv4_max_size 65536
   [...]
```

netkit PoC for AF_XDP

UAPI extension:

- New lower dev attribute with ifindex and queue pair range to keep it simple

include/uapi/linux/if_link.h

```
@@ -1307,6 +1307,12 @@ enum netkit_scrub {
1307 1307         NETKIT_SCRUB_DEFAULT,
1308 1308 };
1309 1309
     1310 + struct netkit_lowerdev {
     1311 +         __u32 ifindex;
     1312 +         __u32 queue_id_from;
     1313 +         __u32 queue_id_to;
     1314 + };
     1315 +
1310 1316 enum {
1311 1317         IFLA_NETKIT_UNSPEC,
1312 1318         IFLA_NETKIT_PEER_INFO,
@@ -1318,6 +1324,7 @@ enum {
1318 1324         IFLA_NETKIT_PEER_SCRUB,
1319 1325         IFLA_NETKIT_HEADROOM,
1320 1326         IFLA_NETKIT_TAILROOM,
     1327 +         IFLA_NETKIT_LOWERDEV,
1321 1328         __IFLA_NETKIT_MAX,
1322 1329 };
1323 1330 #define IFLA_NETKIT_MAX (__IFLA_NETKIT_MAX - 1)
```

netkit PoC for AF_XDP

ndo_bpf:

- Only allows the setup of xsk pools
- No actual XDP prog attachment

```
173  + static int netkit_xdp(struct net_device *dev, struct netdev_bpf *xdp)
174  + {
175  +         struct netkit *nk = netkit_priv(dev);
176  +
177  +         if (!nk->lower || nk->primary)
178  +                 return -EOPNOTSUPP;
179  +
180  +         switch (xdp->command) {
181  +         case XDP_SETUP_XSK_POOL:
182  +                 if (!netkit_queue_valid(dev, xdp->xsk.queue_id))
183  +                         return -EPERM;
184  +                 break;
185  +         case XDP_SETUP_PROG:
186  +                 return -EPERM;
187  +         default:
188  +                 return -EINVAL;
189  +         }
190  +
191  +         return nk->lower->dev->netdev_ops->ndo_bpf(nk->lower->dev, xdp);
192  + }
```

netkit PoC for AF_XDP

ndo_xsk_wakeup:

- Simple pass-through to lower device for the set of queues

```
194 + static int netkit_xsk_wakeup(struct net_device *dev, u32 queue_id, u32 flags)
195 + {
196 +         struct netkit *nk = netkit_priv(dev);
197 +
198 +         if (!nk->lower || nk->primary)
199 +                 return -EOPNOTSUPP;
200 +         if (!netkit_queue_valid(dev, queue_id))
201 +                 return -EPERM;
202 +
203 +         return nk->lower->dev->netdev_ops->ndo_xsk_wakeup(nk->lower->dev,
204 +                                                           queue_id, flags);
205 + }
```

netkit PoC for AF_XDP

3 new ndo's needed for custom pool binding:

- netkit implements these to bind pool to lower device

```
1646  +        struct xsk_buff_pool *  (*ndo_xsk_get_pool_from_qid)(struct net_device *dev,
1647  +                                                             u16 queue_id);
1648  +        int                     (*ndo_xsk_reg_pool_at_qid)(struct net_device *dev,
1649  +                                                           struct xsk_buff_pool *pool,
1650  +                                                           u16 queue_id);
1651  +        void                    (*ndo_xsk_clear_pool_at_qid)(struct net_device *dev,
1652  +                                                             u16 queue_id);
```

netkit PoC for AF_XDP

Example ndo for pool registration:

- The get/clear pool callbacks follow similar model to manage pool association with lower device

```
224 + static int netkit_xsk_reg_pool_at_qid(struct net_device *dev,
225 +                                       struct xsk_buff_pool *pool,
226 +                                       u16 queue_id)
227 + {
228 +         struct netkit *nk = netkit_priv(dev);
229 +
230 +         if (!nk->lower || nk->primary)
231 +                 return -EOPNOTSUPP;
232 +         if (!netkit_queue_valid(dev, queue_id))
233 +                 return -EPERM;
234 +         if (queue_id >= max_t(unsigned int,
235 +                               nk->lower->dev->real_num_rx_queues,
236 +                               nk->lower->dev->real_num_tx_queues))
237 +                 return -EINVAL;
238 +         if (xsk_get_pool_from_qid(nk->lower->dev, queue_id))
239 +                 return -EBUSY;
240 +
241 +         pool->netdev = nk->lower->dev;
242 +         pool->queue_id = queue_id;
243 +
244 +         if (queue_id < nk->lower->dev->real_num_rx_queues)
245 +                 nk->lower->dev->_rx[queue_id].pool = pool;
246 +         if (queue_id < nk->lower->dev->real_num_tx_queues)
247 +                 nk->lower->dev->_tx[queue_id].pool = pool;
248 +         return 0;
249 + }
```

Future work: qemu

Attachment handling:

```
In a general case QEMU will need CAP_NET_ADMIN and CAP_SYS_ADMIN
or CAP_BPF capabilities in order to load default XSK/XDP programs to
the network interface and configure BPF maps.  It is possible, however,
to run with no capabilities.  For that to work, an external process
with enough capabilities will need to pre-load default XSK program,
create AF_XDP sockets and pass their file descriptors to QEMU process
on startup via 'sock-fds' option.  Network backend will need to be
configured with 'inhibit=on' to avoid loading of the program.
QEMU will need 32 MB of locked memory (RLIMIT_MEMLOCK) per queue
or CAP_IPC_LOCK.
```

Short-term: ‘map-path=/xyz’ would be useful in future to bind mount bpf map into KubeVirt Pod . Mid term: Do we even need XDP prog?

Future work: XDP API redesign

XDP API overhaul:

- XDP bpf_mprog conversion
  - Unlocking multi-user attachments in general for native XDP
- Per-queue/queue set XDP bpf_mprog
  - Different programs for different queues (or RSS context)
- AF_XDP receive & transmit-side hook
  - Policy enforcement and introspection for VM traffic for RX and TX side

Same UAPI as tcx, additional parameter: queue_id or rss_context (default 0)?
Flagged as NETDEV_XDP_ACT_MPROG capability (initial support mlx5/ice?)
Old netlink API can route the request through the new API if capability set
Deprecate generic XDP along the way?

Future work: netkit

Depends on XDP API overhaul work:

- Similar to netkit/tcx programs, allow XDP program attachment from the primary netkit device for all AF_XDP ingress/egress traffic
- netkit peer device cannot manage attachments

Future work: netkit

Depends on XDP API overhaul work:

- Similar to netkit/tcx programs, allow XDP program attachment from the primary netkit device for all AF_XDP ingress/egress traffic
- netkit peer device cannot manage attachments

Even further out in future (netkit as phys NIC slice):

- Full reservation of NIC queues also outside of AF_XDP
- BPF still controls ingress/egress traffic into Pod on peer device, managed via primary netkit device
- Fully dedicated NIC queues in particular for latency sensitive applications

Future work: AF_XDP performance features

From qemu commit, native AF_XDP vs copy/skb mode:

```
iperf3 result:
 TCP stream      : 19.1 Gbps

In skb mode the same setup shows much lower performance, similar to
the setup where pair of physical NICs is replaced with veth pair:

iperf3 result:
  TCP stream      : 9 Gbps

Results in skb mode or over the veth are close to results of a tap
backend with vhost=on and disabled segmentation offloading bridged
with a NIC.
```

Future work: AF_XDP performance features

From qemu commit, current limitations:

```
There are also a few kernel limitations.  AF_XDP sockets do not
support any kinds of checksum or segmentation offloading.  Buffers
are limited to a page size (4K), i.e. MTU is limited.  Multi-buffer
support implementation for AF_XDP is in progress, but not ready yet.
Also, transmission in all non-zero-copy modes is synchronous, i.e.
done in a syscall.  That doesn't allow high packet rates on virtual
interfaces.
```

Thanks! Questions?

netkit PoC: https://github.com/cilium/linux/commits/pr/netkit-xdp/