Vhost dataplane in Qemu

Jason Wang
Red Hat
Agenda

- History & Evolution of vhost
- Issues
- Vhost dataplane
- TODO
Userspace Qemu networking

Qemu

mainloop in IOThread

virtio

VCPU

VCPU

KVM

userspace

kernel

tap fd

vcpu fd

vcpu fd

TAP

OVS

eth0

NIC
Userspace qemu networking is slow

- **Limitation of both qemu and backend**
  - Run inside mainloop
    - No real multiqueue
    - No dedicated thread, No busy polling
  - Extra data copy to internal buffer
  - TAP
    - syscall to send/receive message
  - IRQ/ioexit is slow
    - VCPU needs to be blocked
    - Slow path
  - No burst/bulking
Vhost kernel

- qemu
- vcpu
- ioctl()/read()/write()
- kick(ioeventfd)
- notify(irqfd)
- TAP
- ioctl()
- sendmsg()
- recvmsg()
- virtio
- eBPF
- TAP
- OVS
- eth0
- NIC
- userspace
- kernel
Vhost user

- qemu
- vcpu
- virtio
- Vhost
- OVS
- DRV
- iommu
- vfio
- NIC
- userspace
- kernel
- irqfd
- ioeventfd
- AF_UNIX
- mmap()
- ioctl()
VHost

• Offload dataplane to another process
  - kthread or userspace process

• A set of API that did
  - Features Negotiation
  - MEM Table
  - Dirty page logging
  - Virtqueues setting
  - Endianess
  - Device specific

• An API transport
  - ioctl()
So far so good?
How hard for adding a new feature

- Formalization in Virtio Specification
- Codes in qemu userspace virtio-net backend
- Vhost protocol extension:
  - Vhost-kernel (uapi), vhost-user (has its own spec)
  - Versions, feature negotiations, compatibility
- Vhost support codes in qemu (user and kernel)
- Features (bugs) duplicated everywhere:
  - vhost_net, dpdk, TAP, macvtap, OVS, VPP, qemu
Even if we manage to do this
Device IOTLB

qemu

vIOMMU

IOTLB miss

IOTLB update

vhost backend

Device IOTLB

slow or even unreliable
Minor impact for static mapping
Poor performance for dynamic mapping
Issue

Datapath needs information from control path. But vhost control path is not designed for high performance.
Receive Side Scaling

VCPU0  VCPU1  VCPU2  VCPU3

qemu  cvq  MSI-X

Indirection table?

RSS

Network backend

More kinds of steering policy?

q1  q2

1

vhost  algo

AF_UNIX
Networking backend is transparent to qemu in the case of vhost-user. Net specific request through vhost-user.
Migration compatibility

Host 1
- Guest
  - qemu
  - vhost backend with EVENT_IDX

Host 2
- Guest
  - qemu
  - vhost backend without EVENT_IDX

migrate
Issue

Though features was negotiated during startup. Backend needs to implement each feature for providing migration compatibility.
Attack surface

qemu

untrusted userspace drv
VFIO
IOMMU DRV

guest

kernel

vfio

vIOMMU

Device IOTLB

vhost backend

MEM_TABLE

ATS request

ATS reply

can protect malicious guest userspace driver. but not malicious vhost-backend.
We don’t want to trust vhost-user backend
But we share (almost) all memory to it!
Issues with external vhost process

- **Complexity in Engineering**
  - Hard to be extended, duplicated codes (bugs) in many places

- **Performance is not always good**
  - Datapath can not be offloaded completely

- **Visibility of networking backend**
  - Re-invent wheels in vhost-user protocol

- **Divergence of protocol between vhost-kernel and vhost-user**
  - Workarounds, how to deal with the 3rd vhost transport?

- **Increasing of attack surface**
Vhost dataplane = Vhost through qemu IOThread

- **Vhost IOThreads**
  - Datapath in vhost IOThread
    - Hide VM state from backends
    - Function call for state accessing, better vIOMMU
    - Decouple vitio out of backends
  - Full functional features through control vq
  - Fast address translation (vhost memory table)
  - Copy inside qemu
  - Drivers for various backends
  - Multiqueue
Vhost dataplane

Qemu - Vhost IOThread

- vhost protocol
- virtqueue manipulation helpers *
- IOThreads

vhost dataplane API

Drivers

- netmap
- dpdk
- AF_XDP/AF_PACKET
- vhost_net/TAP
- mdev/zerocopy
- virtio-user
- shared memory
- ...

Drivers
Inline driver
Multi-process cooperation

Qemu
- iotread
- vcpu
- thread
- vIOMMU

Vhost dataplane
- vhost
- virtio
- virtio-net drv
- rte_ring
- ring pmd
- mempool

OVS-dpdk
- port

Guest
- port
Vhost friendly networking backend

• **Generic inline networking functions:**
  - TX/RX, Multiqueue, QOS, GSO, steering …

• **Secure and efficient IPC**
  - No knowledge of virtio
  - Stable ABI

• **Programmability for userspace defined polices**

• **Do we have something existed?**
  - AF_XDP?
## External vs vhost-dataplane

<table>
<thead>
<tr>
<th></th>
<th>Remote Dataplane</th>
<th>Vhost-Dataplane</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VM metadata access</strong></td>
<td>Slow, inter process communication</td>
<td>Fast, function call</td>
</tr>
<tr>
<td><strong>New feature development</strong></td>
<td>Hard, New types of IPCs</td>
<td>Easy, limited to qemu (or programmibility from backend)</td>
</tr>
<tr>
<td><strong>Compatibility</strong></td>
<td>Complex, extra works on the backend</td>
<td>Easy, limited to qemu</td>
</tr>
<tr>
<td><strong>New backend integration</strong></td>
<td>Hard, need to know all about virtio</td>
<td>Easy, no need to know virtio</td>
</tr>
<tr>
<td><strong>Attack surface</strong></td>
<td>Increased</td>
<td>Limited to qemu</td>
</tr>
<tr>
<td><strong>Backend visibility</strong></td>
<td>May be transparent</td>
<td>Visible</td>
</tr>
</tbody>
</table>
Virtio-net = virtio + networking

• **Vhost dataplane**
  - Virtio functions in vhost IOThread
  - Networking functions in the backend

• **Limitation**
  - More cores for multi process cooperation
  - The ideal networking backends does not exist in real world
    • invent one?
  - ...
Status & TODO

• Status
  – prototype
    • Basic IOThreads / Virtqueue helpers
    • TAP drive
      – -device virtio-net-pci,netdev=vd0 -netdev vhost-dp,id=vd0,driver=tap-driver0 -object vhost-dp-tap,id=tap-driver0
  – RFC sent in next few months

• TODO
  – Dpdk static linking
  – vIOMMU, Multiqueue
  – Benchmarking
Thanks