VIRTIO: VHOST DATA PATH ACCELERATION TORWARDS NFV CLOUD

CUNMING LIANG, Intel

Agenda

- Towards NFV Cloud
  - Background & Motivation
- vHost Data Path Acceleration
  - Intro
  - Design
  - Impl
- Summary & Future Work

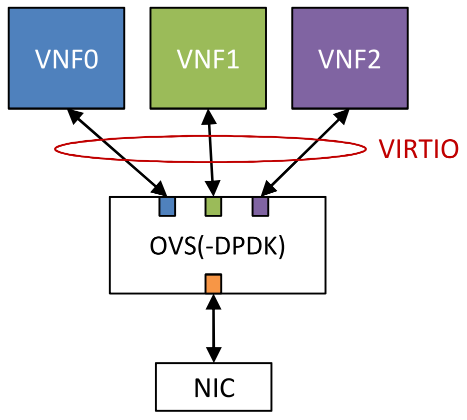
Towards NFV Cloud

- VIRTIO is a well recognized by Cloud
- DPDK promotes its Perf. into NFV Level
- New accelerators comes, what's the SW impact on I/O virtualization?

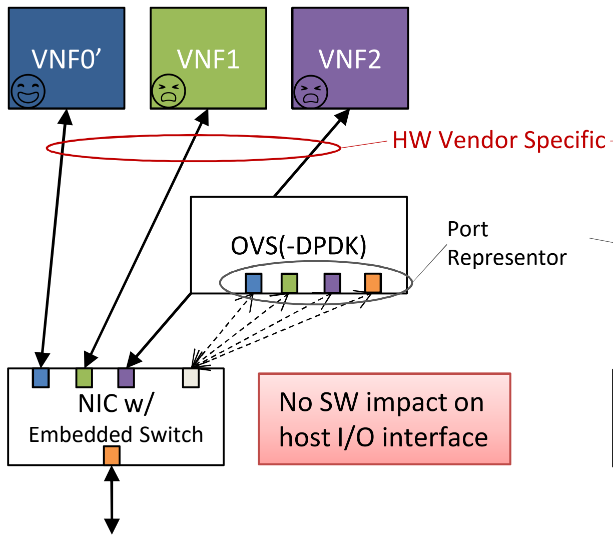
Native I/O Perf. by SR-IOV device PT
Faster simple forwarding by 'cache'
Remains historical gaps of cloudlization
- Stock VM and SW vSwitch fallback
- Cross-platform Live-migration

vDPA: Balanced Perf. and Cloudlization
- Device Pass-thru Like Performance
- Hypervisor native I/O
- Live-migration Friendly
- Stock vSwitch/VMs Support

GOAL

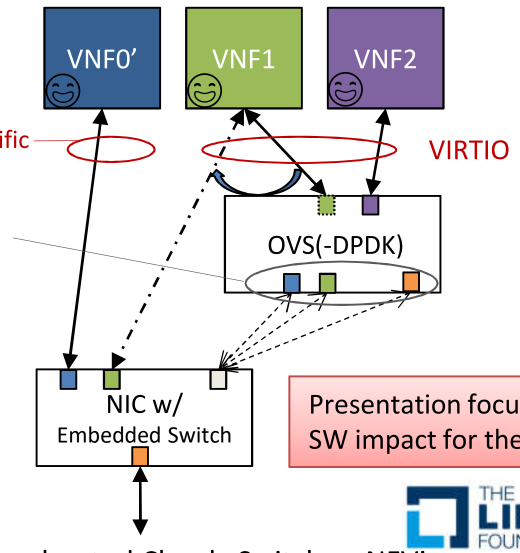


Cloud vSwitch as NFVi



No SW impact on host I/O interface

Accelerated vSwitch as NFVi



Presentation focus on SW impact for the goal

Accelerated Cloud vSwitch as NFVi

vDPA Intro

What is vDPA

- As a VMM native device, PV hasn't shared the I/O VT benefits
  - PV device was born with **cloud-lization characters**,
  - But it's **lack of performance towards NFV cloud**.
- vHost Data Path Acceleration is a framework offering virtualization infrastructure for VRING capable device
  - Decompose DP/CP of VIRTIO device
  - DP pass-thru for VRING capable device
  - CP remains to be emulated, but backed by a DP capable device
  - VRING capable device has ability to ENQ/DEQ VRING and recognize VRING format according to VIRTIO Spec.

	PV	**Dev Pass-thru**
VMM	Aware	Unaware
Performance	~Cloud Qualified	~NFV Qualified
Direct I/O	N/A(SW Relay)	IOMMU/SMMU
I/O Bus VT	N/A	SR-IOV, SIOV
CPU Utilization	Variable	Zero
SW framework	Emulated device w/ backend Impl.	kvm-pci, vfio-{pci|mdev}
Cloud-lization	- LM friendly - SW fallback - SW vswitch native	- Tricky LM - N/A - N/A

Why not device pass-thru for VIRTIO

Statement

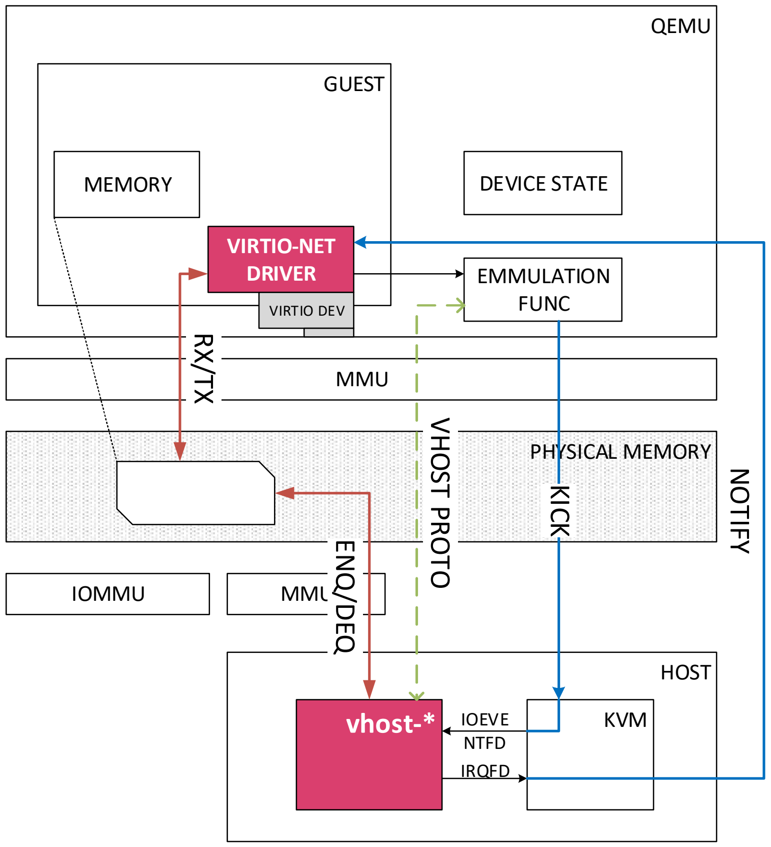
- VIRTIO is a SW Spec. continuous evolution
- Unlikely forcing HW to follow ‘uniform’ device definition

Disadvantage

- Inherits all device pass-thru properties
  - “All or nothing” offload, SW fallback in the guest (bonding)
  - Framework limitation to support live-migration in general use
- Becomes VIRTIO Spec. version specific
  - e.g. 0.95 PIO, 1.0 MMIO, etc.
- Lose the benefit of decomposed frontend/backend device framework
  - Diverse backend adaption

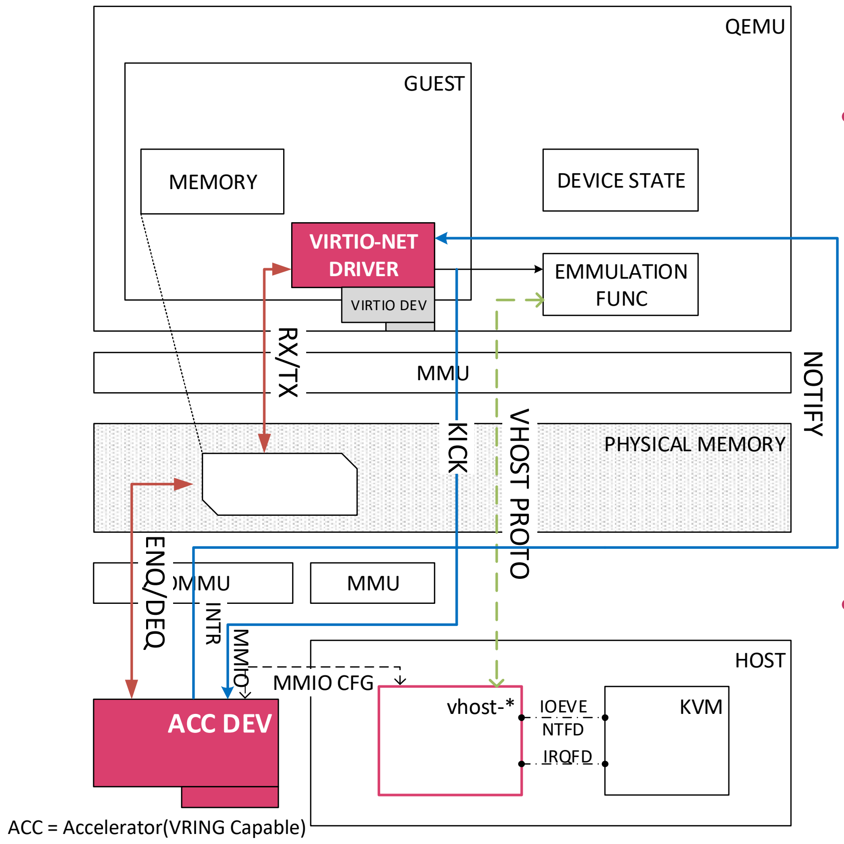
vDPA Design

VIRTIO Anatomy



- PCI CSR Trapped
- Device-specific register trapped (PIO/MMIO)
- Emulation backed by backend adapter via VHOST PROTO
- Packet I/O via Shared memory
- Interrupt via IRQFD
- Doorbell via IOEVENTFD
- Diverse VHOST backend adaption

Data Path Pass-thru

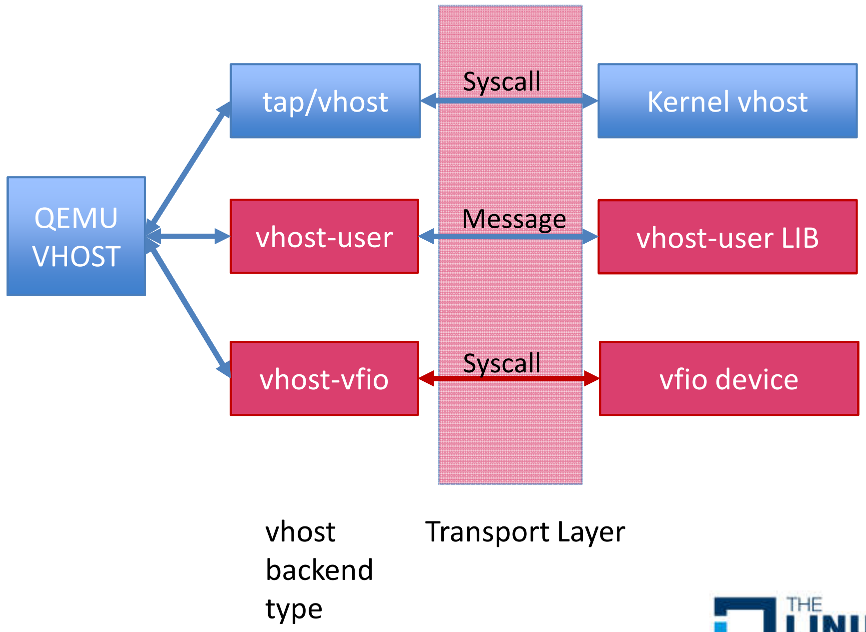


ACC = Accelerator(VRING Capable)

- Decomposed VRING Data Path on ACC
  - DMA Enq/Deq VRING via IOMMU
  - Interrupt Notification
    - VFIO INTR eventfd associate with IRQFD
    - IRQFD as token for irq_bypass Prod/Cons
    - Leverage existing posted-interrupt support
  - Doorbell Kick
    - SW Relayed IOEVENTFD to trigger doorbell (PIO)
    - Add guest physical memory slot for doorbell direct mapping (MMIO)
- ACC needs a device framework
  - Leverage user space driver by vhost-user
  - vhost-net won't directly associate with driver

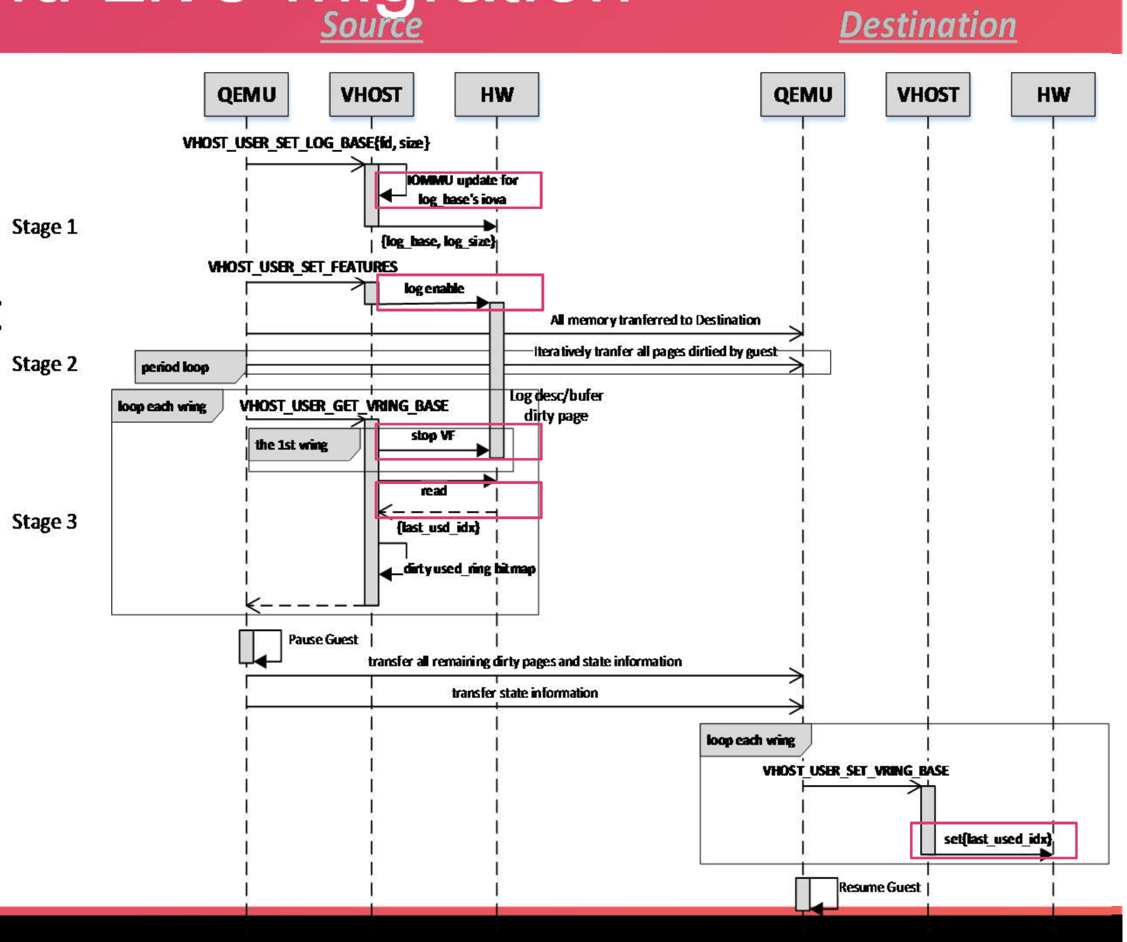
Control Path Emulation

- VIRTIO PIO/MMIO trap to QEMU
- Emulation Call → VHOST Req.
- VHOST Req. go thru transport channel via different backend
- User space backend
  - Feature message extension
- Kernel space backend
  - Add a new transport channel for VFIO (mediated) device
  - Define transport layout for data path relevant request



Cross vhost Backend Live-migration

- Live-migration Friendly
- Consistent vhost transport message sequence interact with QEMU live-migration
- Cross vhost backend LM
  - netdev for virtio-net-pci
    - tap w/ vhost=on/off
    - vhost-user
    - vhost-vfio (+)



Source
Destination
QEMU
VHOST
HW
QEMU
VHOST
HW
Stage 1
Stage 2
Stage 3
VHOST_USER_SET_LOG_BASE{fd, size}
IOMMU update for log_base's iova
{log_base, log_size}
VHOST_USER_SET_FEATURES
log enable
All memory tranferred to Destination
Iteratively tranfer all pages dirtied by guest
period loop
loop each vring
VHOST_USER_GET_VRING_BASE
Log desc/bufer dirty page
the 1st vring
stop VF
read
{last_usd_idx}
dirty used_ring bitmap
Pause Guest
transfer all remaining dirty pages and state information
transfer state information
loop each vring
VHOST_USER_SET_VRING_BASE
set{last_used_idx}
Resume Guest

vDPA Implementation

Construct vDPA via VFIO

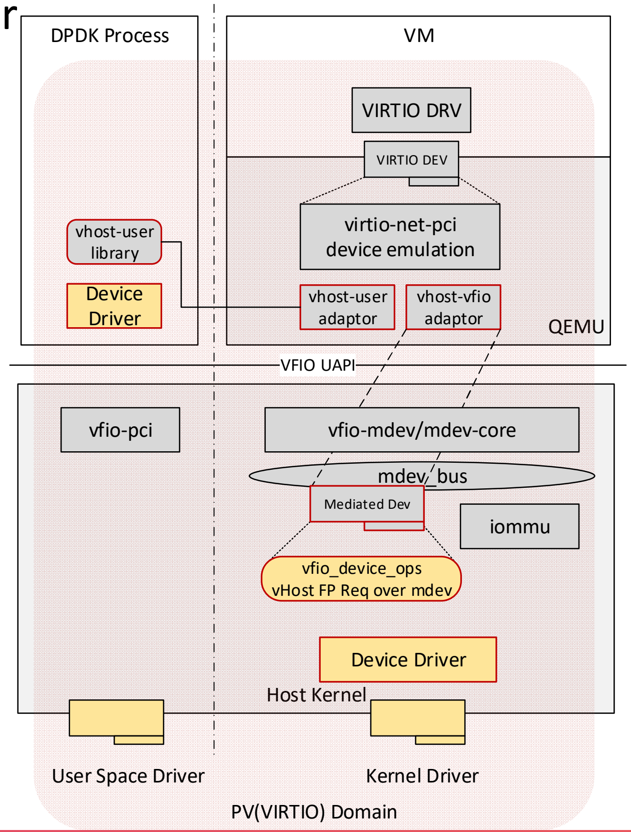
#1 QEMU for User Space Driver

vhost-user adapter

- New protocol message extension -- F_VFIO
- SLAVE Request to handover vfio group fd and notify meta data
- vhost-user adapter to map doorbell

Dependence

- Leverage user space device framework (DPDK)



#2 QEMU for Kernel Driver

vhost-vfio adapter

- New netdev as vhost backend
- Reuse QEMU VFIO interface
- VFIO device as vhost request transport layer
- Leverage vfio/mdev framework

Dependence

- mdev_bus IOMMU support
- Single mdev per VF instance in Kernel

QEMU Changes for User Space Driver

-- #1 vhost-user extension

- New Protocol Feature -- VHOST_USER_PROTOCOL_F_VFIO
- Slave Request
  - Meta Data Update: VFIO Group FD, Notify Info
  - Actions: Enable/Disable ACC
- VFIO Group FD
  - Associate VFIO group fd with kvm_device_fd
  - Update GSI routing
- Notify Info
  - Represent for doorbell info (in page boundary)
  - Add guest physical memory slot

QEMU Changes for Kernel Driver
-- #2 vhost-vfio

- New netdev for virtio-net-pci
  - '-chardev vfio,id=vfio0,sysfsdev=/sys/bus/mdev/devices/$UUID \
  - -netdev vhost-vfio,id=net0,chardev=vfio0 -device virtio-net-pci,netdev=net0'
- VFIO device based vhost transport layer
  - vhost request over vfio_device_ops(read, write)
  - data path relevant request: feature, vring, doorbell, log
- Construct context for data path accelerator
  - Leverage QEMU KVM/VFIO interface
  - Memory region mapping for DMA
  - Add guest physical memory slot for doorbell
  - Interrupt/IRQFD via VFIO device ioctl CMD
- Don't expect other host applications to use the device so far

Relevant Dependence

-- #2 vhost-vfio

- Kernel
  - Leverage VFIO mediated device framework
  - Add IOMMU support for mdev-bus
  - VRING capable device driver to register as mdev
    - Singleton mode only, 1:1 BDF(Bus, Device, Function) with mdev

Summary

- Hypervisor Native I/O
  - virtio-net-pci
- Stock vSwitch/VMs Support
  - Transparent to frontend
- Device Pass-thru Like Performance
  - Data path pass-thru
- Live-migration Friendly
  - Cross vhost backend live-migration

Future Work

- Collect feedback
- Send out RFC patches to Kernel, Qemu and DPDK
- Upstream current Impl. together w/ other relevant patches
- Continue to enable VRING incompatible device
  - other non-VRING accelerators
  - new VRING version

Acknowledgment

- Tiwei Bie
- Jianfeng Tan
- Dan Daly
- Zhihong Wang
- Xiao Wang
- Heqing Zhu
- Kevin Tian
- Rashmin N Patal
- Edwin Verplanke

Thanks!

Q&A

Contacts:

cunming.liang@intel.com

KVM
FORUM

THE LINUX FOUNDATION