OMG, NPIV!
Virtualizing Fibre Channel with Linux and KVM

Paolo Bonzini, Red Hat
Hannes Reinecke, SuSE

KVM Forum 2017
Outline

• Introduction to Fibre Channel and NPIV
• Fibre Channel and NPIV in Linux and QEMU
• A new NPIV interface for virtual machines
• virtio-scsi 2.0?
What is Fibre Channel?

- High-speed (1-128 Gbps) network interface
- Used to connect storage to server ("SAN")

<table>
<thead>
<tr>
<th>FC-4</th>
<th>Application protocols: FCP (SCSI), FC-NVMe</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC-3</td>
<td>Link services (FC-LS): login, abort, scan...</td>
</tr>
<tr>
<td>FC-2</td>
<td>Signaling protocols (FC-FS): link speed, frame definitions ...</td>
</tr>
<tr>
<td>FC-1</td>
<td>Data link (MAC) layer</td>
</tr>
<tr>
<td>FC-0</td>
<td>PHY layer</td>
</tr>
</tbody>
</table>
Ethernet NIC vs. Fibre channel HBA

- Buffer credits: flow control at the MAC level
- HBAs hide the raw frames from the driver
- IP-address equivalent is dynamic and mostly hidden
- Devices (ports) identified by World Wide Port Name (WWPN) or World Wide Node Name (WWNN)
  - Similar to Ethernet MAC address
  - But: not used for addressing network frames
  - Also used for access control lists (“LUN masking”)
### Fibre channel HBA vs. Ethernet NIC

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>Server</td>
</tr>
<tr>
<td>PLOGI</td>
<td>Port login: prepare communication with a target</td>
</tr>
<tr>
<td>PRLI</td>
<td>Process login: select protocol (SCSI, NVMe, ...), optionally establish connection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAC address</th>
<th>WWPN/WWNN</th>
<th>World Wide Port/Node Name (2x64 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP address</td>
<td>Port ID</td>
<td>24-bit number</td>
</tr>
<tr>
<td>DHCP</td>
<td>FLOGI</td>
<td>Fabric login (usually placed inside switch)</td>
</tr>
<tr>
<td>Zeroconf</td>
<td>Name server</td>
<td>Discover other active devices</td>
</tr>
</tbody>
</table>
FC command format

- FC-4 protocols define commands in terms of sequences and exchanges
- The boundary between HBA firmware and OS driver depends on the h/w
- No equivalent of “tap” interfaces
FC Port addressing

- FC Ports are addressed by WWPN/WWNN or FCID
- Storage arrays associate disks (LUNs) with FC ports
- SCSI command are routed from *initiator* to *target* to LUN
  - Initiator: FC port on the HBA
  - Target: FC port on the storage array
  - LUN: (relative) LUN number on the storage array
FC Port addressing

Node 1

Node 2

SAN

A

WWPN 1a, WWPN 1b

WWPN 2a, WWPN 2b

WWPN 5

B

WWPN 3a

WWPN 3b

WWPN 4a

WWPN 4b
FC Port addressing

- Resource allocation based on FC Ports
- FC Ports are located on FC HBA
- But: VMs have to share FC HBAs
- Resource allocation for VMs not possible
NPIV: N_Port_ID virtualization

- Multiple FC_IDs/WWPNs on the same switch port
  - WWPN/WWNN pair (N_Port_ID) names a vport
  - Each vport is a separate initiator
- Very different from familiar networking concepts
  - No separate hardware (unlike SR-IOV)
  - Similar to Ethernet macvlan
  - Must be supported by the FC HBA
NPIV: N_Port_ID virtualization
NPIV and virtual machines

• Each VM is a separate initiator
  – Different ACLs for each VM
  – Per-VM persistent reservations

• The goal: map each FC port in the guest to an NPIV port on the host.
NPIV in Linux

- FC HBA (ie the PCI Device) can support several FC Ports
  - Each FC Port is represented as an fc_host (visible in /sys/class/fc_host)
  - Each FC NPIV Port is represented as a separate fc_host
- Almost no difference between regular and virtual ports
NPIV in Linux

- FC-HBA
  - FC Port
  - FC NPIV Port
- Linux HBA Driver
- scsi_host
- NPIV scsi_host
- sda
- sdb
- sdc
- sdd
QEMU does not help...

- PCI device assignment
  - Uses the VFIO framework
  - Exposes an entire PCI device to the guest
- Block device emulation
  - Exposes/emulates a single block device
  - virtio-scsi allows SCSI command passthrough
- Neither is a good match for NPIV
  - PCI devices are shared between NPIV ports
  - NPIV ports presents several block devices
NPIV passthrough and KVM

- HBA
- LUN
- PCI
- VFIO
- SCSI
- virtio-scsi
LUN-based NPIV passthrough

• Map all devices from a vport into the guest
• New control command to scan the FC bus
• Handling path failure
  – Use existing hot-plug/hot-unplug infrastructure
  – Or add new virtio-scsi events so that /dev/sdX doesn’t disappear
LUN-based NPIV passthrough

- Assigned NPIV vports do not “feel” like FC
  - Bus rescan in the guest does not map to LUN discovery in the host
  - New LUNs not automatically visible in the VM
- Host can scan LUN for partitions, mount file systems, etc.
Can we do better?

- HBA
- VFIO
- vport
- LUN
- PCI
- SCSI
- ??
- virtio-scsi
Mediated device passthrough

- Based on VFIO
- Introduced for vGPU
- Driver virtualizes itself, and the result is exposed as a PCI device
  - BARs, MSIs, etc. are partly emulated, partly passed-through for performance
  - Typically, the PCI device looks like the parent
- One virtual N_Port per virtual device
Mediated device passthrough

**Advantages:**
- No new guest drivers
- Can be implemented entirely within the driver

**Disadvantages:**
- Specific to each HBA driver
- Cannot stop/start guests across hosts with different HBAs
- Live migration?
What FC looks like

SCSI command

- FCP_CMND_IU
- FCP_DATA_IU
- FCP_RSP_IU

Exchange #1
- FLOGI
- PLOGI
- PRLI
- Exchange #1
- SCN
- Exchange #2
What virtio-scsi looks like

SCSI command

Request buffer

Response buffer

Payload

Request queues

Control queue

Event queue
vhost

- Out-of-process implementation of virtio
  - A vhost-scsi device represents a SCSI target
  - A vhost-net device is connected to a tap device
- The vhost server can be placed closer to the host infrastructure
  - Example: network switches as vhost-user-net servers
  - How to leverage this for NPIV?
Initiator vhost-scsi

• Each vhost-scsi device represents an initiator

• Privileged ioctl to create a new NPIV vport
  - WWPN/WWNN → vport file descriptor
  - vport file descriptor compatible with vhost-scsi

• Host driver converts virtio requests to HBA requests

• Devices on the vport will not be visible on the host
Initiator vhost-scsi

• Advantages:
  – Guests are unaware of the host driver
  – Simpler to handle live migration (in principle)

• Disadvantages:
  – Need to be implemented in each host driver (around a common vhost framework)
  – Guest driver changes likely necessary (path failure etc.)
Live migration

- WWPN/WWNN are unique (per SAN)
- Can log into the SAN only once
- For live migration both instances need to access the same devices at the same time
- Not possible with single WWPN/WWNN
Live migration

Node 1

Node 2

SAN

WWPN 1a, WWPN 1b

WWPN 2a, WWPN 2b

WWPN 5
Live migration

Node 1

Node 2

SAN

WWPN 1a, WWPN 1b

WWPN 2a, WWPN 2b

WWPN 5

WWPN 1a

WWPN 1b

WWPN 2a

WWPN 2b

WWPN 3a

WWPN 3b

WWPN 4a

WWPN 4b
Live migration

- Solution #1: Use “generic” temporary WWPN during migration
- Temporary WWPN has to have access to all devices; potential security issue
- Temporary WWPN has to be scheduled/negotiated between VMs
Live migration

- Solution #2: Use individual temporary WWPNs
- Per VM, so no resource conflict with other VMs
- No security issue as the temporary WWPN only has access to the same devices as the original WWPN
- Additional management overhead; WWPNs have to be created and registered with the storage array
Live migration: multipath to the rescue

- Register two WWPNs for each VM; activate multipathing
- Disconnect the lower WWPN for the source VM during migration, and the higher WWPN for the target VM.
- Both VMs can access the disk; no service interruption
- WWPNs do not need to be re-registered.
Is it better?

- HBA
- vport
- LUN
- PCI
- VFIO
- VFIO mdev
- Initiator vhost-scsi
- SCSI
- virtio-scsi
Can we do even better?

- HBA
- VFIO
- vport
- LUN
- PCI
- VFIO mdev
- FC
- ??
- SCSI
- Initiator vhost-scsi
- virtio-scsi
virtio-scsi 2.0?

• virtio-scsi has a few limitations compared to FCP
  – Hard-coded LUN numbering (8-bit target, 16-bit LUN)
  – One initiator id per virtio-scsi HBA (cannot do “nested NPIV”)
• No support for FC-NVMe
virtio-scsi device addressing

- **virtio-scsi** uses a 64-bit *hierarchical LUN*
  - Fixed format described in the spec
  - Selects both a bus (target) and a device (LUN)

- **FC** uses a 128-bit target (WWNN/WWPN) + 64-bit LUN

- Replace 64-bit LUN with I_T_L nexus id
  - Scan fabric command returns a list of target ids
  - New control commands to map I_T_L nexus
  - Add target id to events
• Emulating NPIV in the VM

• FC NPIV port (in the guest) maps to FC NPIV port on the host

• No field in virtio-scsi to store the initiator WWPN

• Additional control commands required:
  - Create vport on the host
  - Scan vport on the host
Towards virtio-fc?

FCP exchange

- **FCP_CMND_IU**
- **FCP_DATA_IU**
- **FCP_RSP_IU**

virtio-fc request

- **FCP_CMND_IU**
- **FCP_RSP_IU**
- **Payload**
Towards virtio-fc

- HBAs handle only “cooked” FC commands; raw FC frames are not visible
- “Cooked” FC frame format different for each HBA
- Additional abstraction needed
Towards virtio-fc?

FCP exchange
- FCP_CMND_IU
- FCP_DATA_IU
- FCP_RSP_IU

FC-NVMe exchange
- NVMe_CMND_IU
- NVMe_DATA_IU
- NVMe_RSP_IU

virtio-fc request
- Request header
  - FCP_CMND_IU or NVMe_CMND_IU
- FCP_RSP_IU or NVMe_RSP_IU
- Payload
Towards virtio-fc?

- Not a 1:1 mapping – still a “cooked” frame
  - Simplified compared to FCP and FC-NVMe
  - Remember drivers do not even see raw frames
- Reuse FC definitions to avoid obsolescence
  - Support for NVMe from the beginning
  - Overall IU structure
  - Possibly, PLOGI/FLOGI structure too
- Things learnt from virtio-scsi can be reused
Summary

- “Initiator vhost” as the abstraction for NPIV vports
  - Common framework for Linux + driver code
  - Very few changes required in QEMU and libvirt
- Live migration can be handled at the libvirt and/or guest levels
- Could extend virtio-scsi or go with virtio-fc