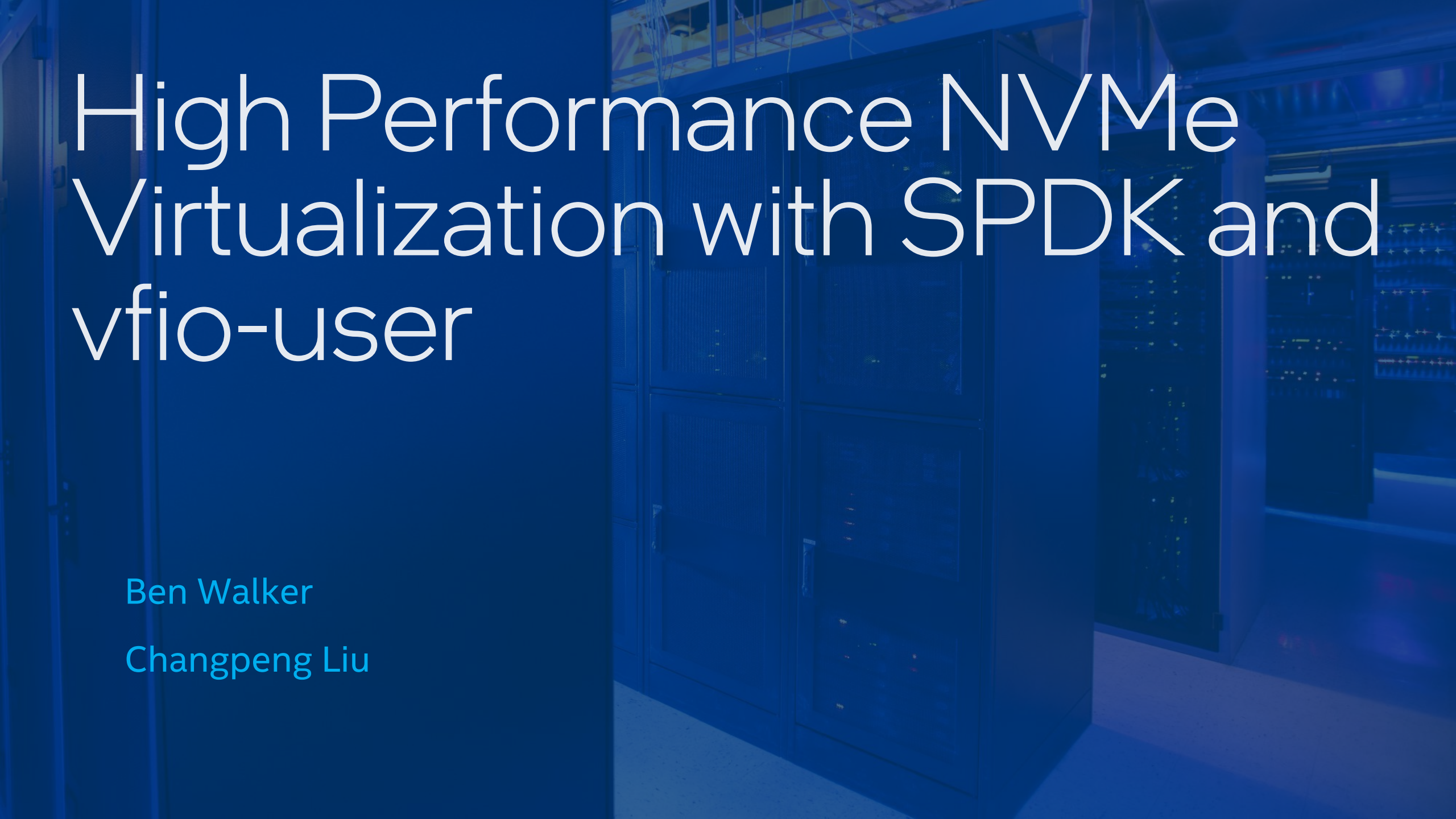
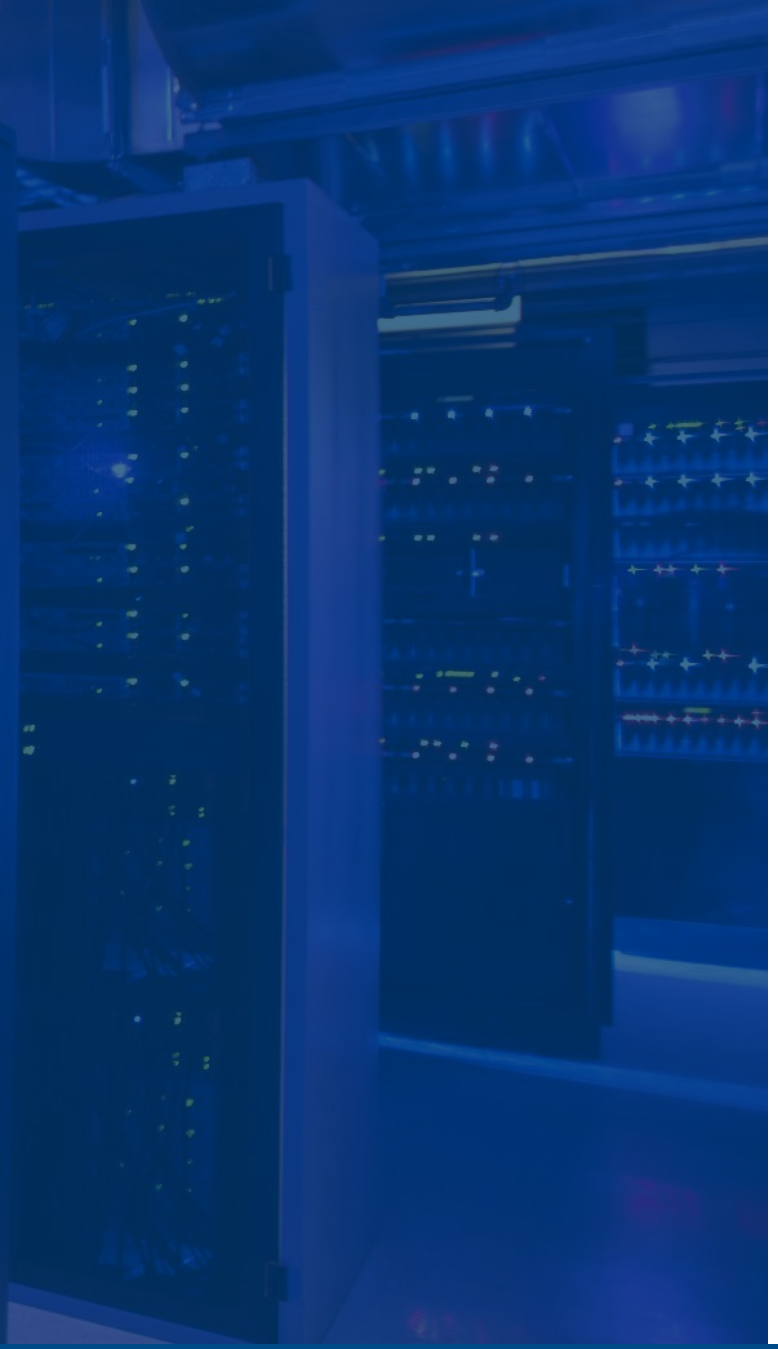


High Performance NVMe Virtualization with SPDK and vfio-user

Ben Walker

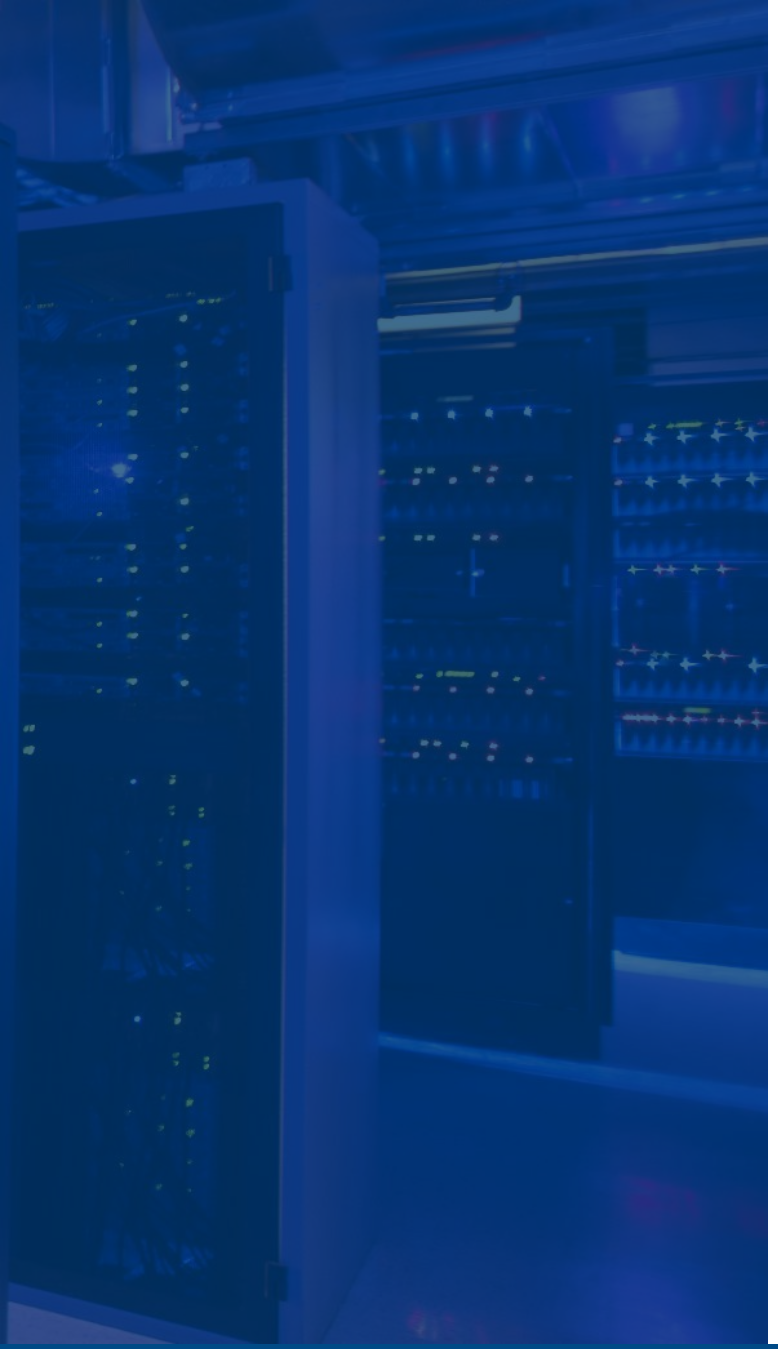
Changpeng Liu



Agenda

1. **Standardization**
   QEMU and libvfio-user
2. **Emulating NVMe devices**
   Re-use of NVMe-oF target
3. **NVMe client library**
   Re-use of NVMe library
4. **Performance**
   vfio-user NVMe vs. vhost-user block

Standardization



Upcoming Talks

1. [libvfio-user: Status Update - Thanos Makatos & John Levon, Nutanix]
   Thursday, September 16 • 15:35 - 16:00

2. [Live Migrating VFIO, vhost-user, and vfio-user Devices - Stefan Hajnoczi, Red Hat]
   Thursday, September 16 • 14:35 - 15:00

Brief Background

Need to emulate device outside VMM

- Performance
- Security
- Stability/resilience
- Device can even run in separate VM

Initially conceived for SPDK

- NVMe device emulation
- But much broader than this use case now!

vfio-user

- Modelled after VFIO ioctls (similar commands/structs)
  1. VFIO commands/structs do exactly what we need
- Similar to vhost-user but not specific to virtio
- VMM agnostic protocol (not tied to QEMU)
  - Commands/messages passed over UNIX domain socket
- Currently under review at qemu-devel
  - “introduce vfio-user protocol specification”
  - https://github.com/nutanix/libvfio-user/blob/master/docs/vfio-user.rst

Emulating NVMe Devices

Approach

NVMe-oF already requires nearly full emulation of an NVMe device

NVMe-oF already has a pluggable transport layer

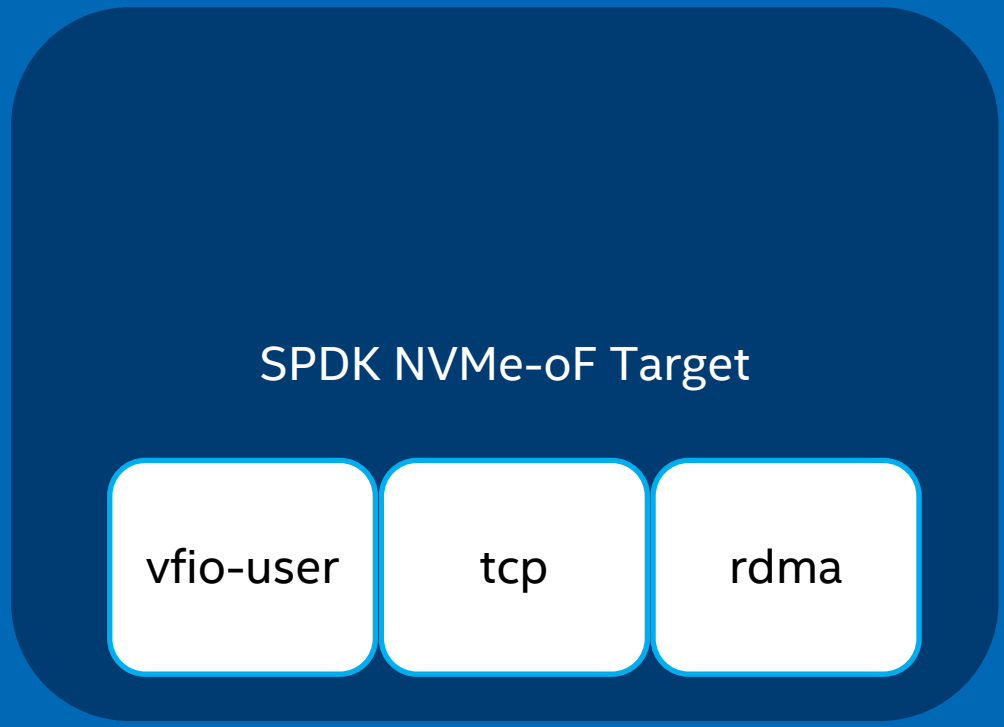
Let's use the NVMe-oF Target!

Let's make a new transport for NVMe-oF

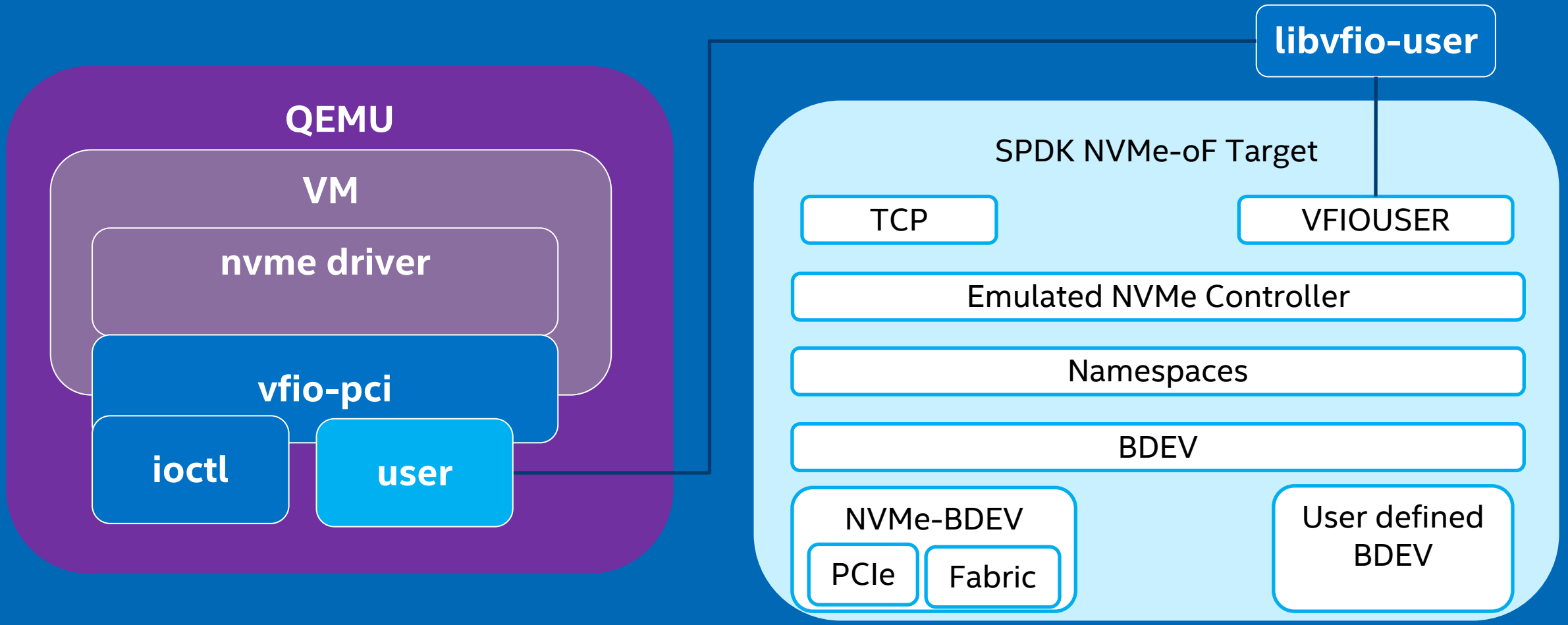
A "shared memory" or "virtualization" transport

But fabrics *is* slightly different than PCIe. Some of the initialization flow is reversed.

- Can we generalize the transport plugin interface to handle this?
- Yes!



Emulating an NVMe device



Challenges

The “listener” concept is different for vfio-user

- Need to “listen” on a Unix domain socket
- Only a single “host” can connect to the subsystem, rather than many
- No need to have an accept poller

Need to generalize concept of listener to “endpoint” in SPDK

- Push accept poller down into the transports. The vfio-user transport just won’t make one.

Challenges

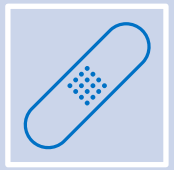
Register reads and writes are very different for PCIe than fabrics	Libvfio-user provides a file descriptor that is signaled when an MMIO operation has occurred	Expand set of allowed Fabrics Property Get/Set commands
• MMIO rather than commands with requests and responses • The set of allowed registers is different	• Create a background thread blocked on that fd • Generate a fake fabrics property get/set command and send to target. For MMIO read, block until response.	• Wider range of registers allowed for PCIe

Challenges

Admin queue creation happens in reversed order compared to real fabrics devices

Real fabrics devices first create an admin queue, then read registers

PCIe devices first read registers, then create an admin queue



Need to create an admin queue as soon as “endpoint” is created so registers can be read

Generate fake admin queue creation command in transport, send to target

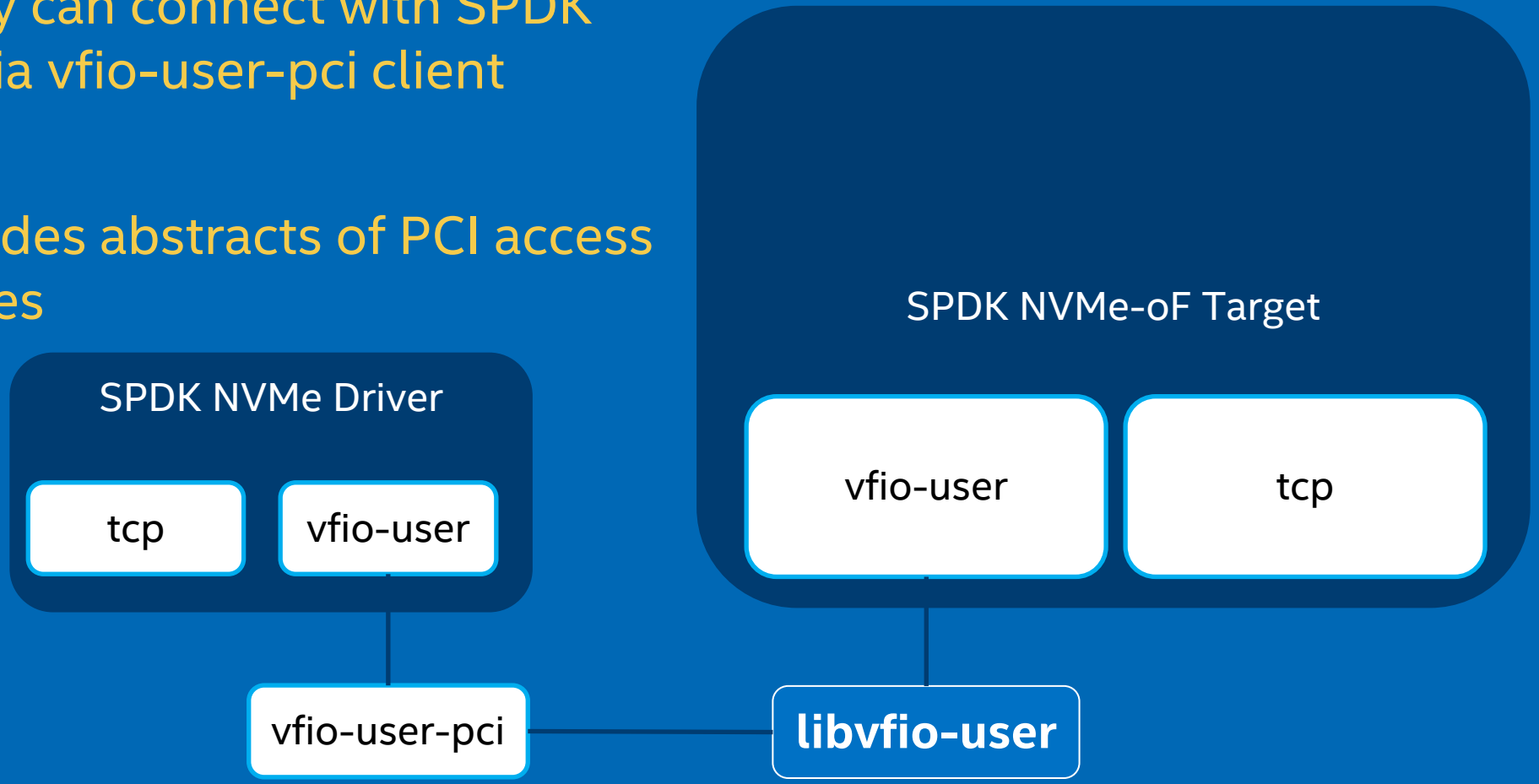
Success!

- Final patch that went into SPDK contained *only* a new transport.
  - No other code changes!
- Generalization is useful for future additional transports we expect to see
  - Running the NVMe-oF target as firmware?
  - QUIC?
- SPDK is a great NVMe emulator
  - Can leverage this to prototype new NVMe features and test from QEMU

NVMe Client Library

NVMe client library with vfio-user transport

- SPDK NVMe library can connect with SPDK NVMe-oF Target via vfio-user-pci client library
- vfio-user-pci provides abstracts of PCI access via socket messages

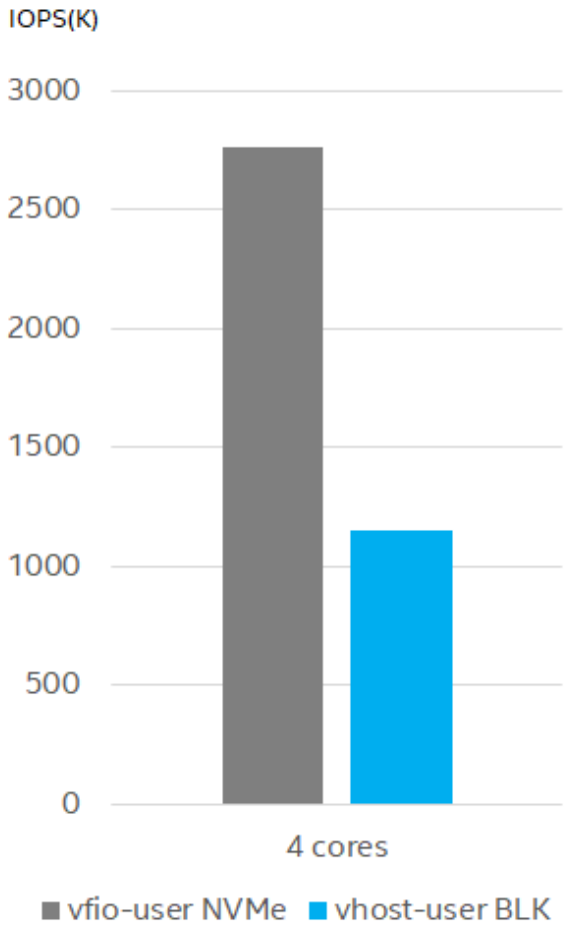


Performance

Vhost and NVMe-oF Target thread model

- Vhost target poller will process IOs in unit of each vhost controller, for multiple IO queues in each controller, all IO queues are processed in the same core context.
- NVMe-oF target poller will process IOs in unit of submission queue of each NVMe controller, for multiple IO queues, submission queues can be processed in different core context.

NULL block devices are used to evaluate the IO processing between VMs and vhost/NVMe-oF target, which means no actual IOs reads from the backend device.



System Configuration: 2 * Intel(R) Xeon(R) Platinum 8180M CPU @ 2.50GHz; 128GB, 2666 DDR4, 6 memory Channels; Bios: HT disabled, Turbo disabled; OS: Fedora 30, kernel 5.6.13-100. VM configuration : 16 vcpus 16GB memory, 16 IO queues; VM OS: Fedora 33, kernel 5.10.8-200, blk-mq enabled; Software: QEMU with vfio-user-pci patch, IO distribution: SPDK, FIO 3.21, io depth=128, numjobs=16, direct=1, block size=4k,randread,total tested data size=400GiB

vfio-user NVMe efficiency

- Chart 1: for the purpose to demo core scaling, total IOPS of 4 VMs increases from 903K,1750K,3500K when using 1,2,4 cores in SPDK.

- Chart 2: for the purpose to compare with vhost-user BLK, vfio-user NVMe gets 984K IOPS while vhost-user BLK gets 1002K IOPS when using 1 core in SPDK and 4 IO queues in VM.

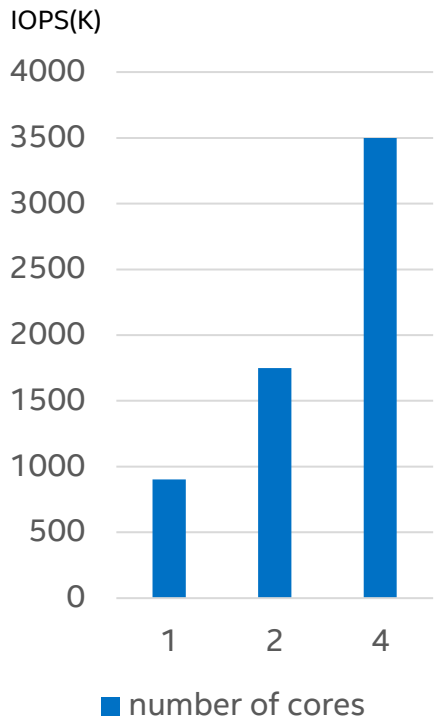


Chart 1: Core scaling for vfio-user NVMe 4VMs

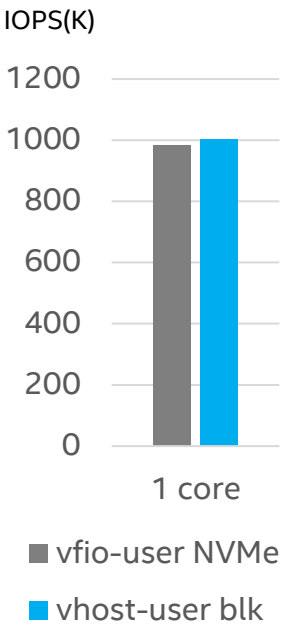


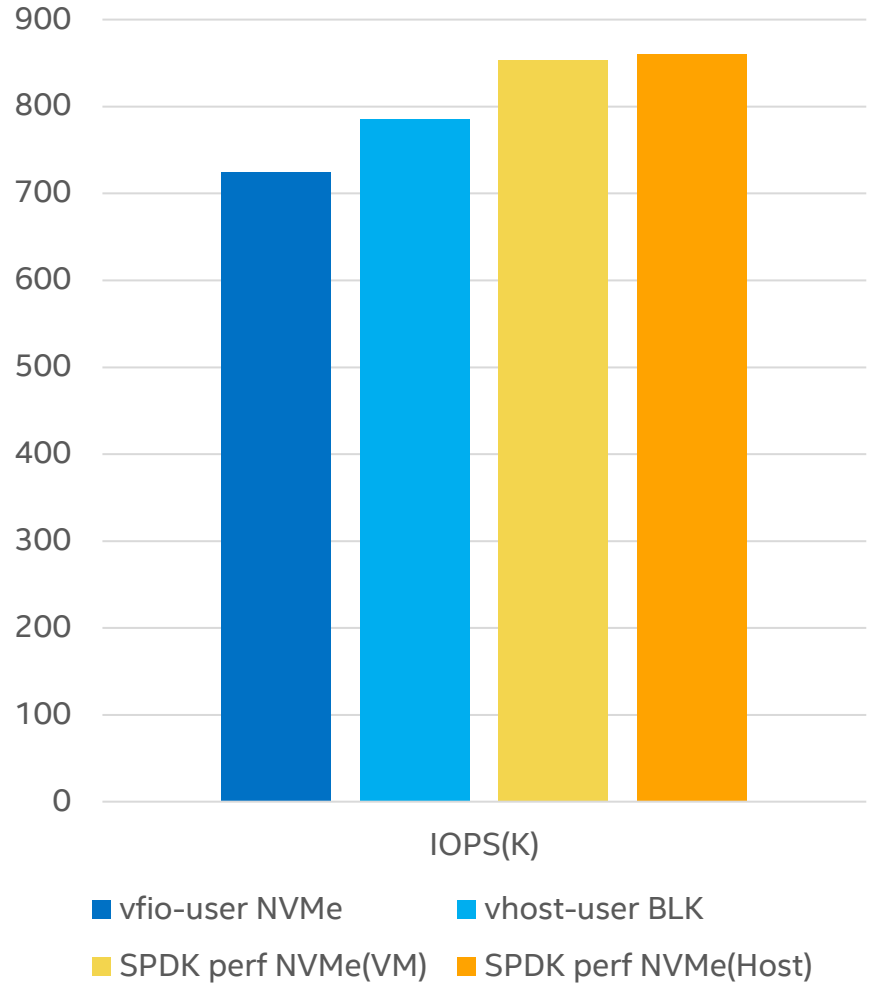
Chart 2: vfio-user NVMe VS. vhost-user BLK 1 VM

NULL BDEVs are used, running FIO inside VM

System Configuration: 2 * Intel(R) Xeon(R) Platinum 8180M CPU @ 2.50GHz; 128GB, 2666 DDR4, 6 memory Channels; Bios: HT disabled, Turbo disabled; OS: Fedora 30, kernel 5.6.13-100. VM configuration : 4 vcpus 8GB memory, 4 IO queues; VM OS: Fedora 33, kernel 5.10.8-200, blk-mq enabled; Software: QEMU with vfio-user-pci patch, IO distribution: SPDK, FIO 3.21, io depth=128, numjobs=4, direct=1, block size=4k,randread,total tested data size=400GiB

vfio-user NVMe efficiency

- Running SPDK NVMe perf tool(QD=128, 4 IO queues using 4 cores) to evaluate the physical NVMe SSD(Intel Optane P5800X, 1.6TB) performance number at first as the base number and gets 860K IOPS.
- Running SPDK NVMe perf tool(same as the above for the workload) inside VM to evaluate the emulated NVMe SSD performance and gets 853K IOPS, we almost can't see the virtualization overhead for this test case.
- Running FIO inside the VM, vfio-user NVMe gets 725K IOPS and vhost-user BLK gets 786K IOPS, vfio-user NVMe can reach about 84% of the physical NVMe base performance number while vhost-user BLK can reach about 91%.



System Configuration: 2 * Intel(R) Xeon(R) Platinum 8180M CPU @ 2.50GHz; 128GB, 2666 DDR4, 6 memory Channels; Intel P5800X 1.6TB, fw: L0310100; Bios: HT disabled, Turbo disabled; OS: Fedora 30, kernel 5.6.13-100. VM configuration : 4 vcpus 8GB memory, 4 IO queues; VM OS: Fedora 33, kernel 5.10.8-200, blk-mq enabled; Software: QEMU with vfio-user-pci patch, IO distribution: SPDK, FIO 3.21, io depth=128, numjobs=4, direct=1, block size=4k,randread,total tested data size=400GiB


intel