NVMe Over Fabrics Support in Linux

Christoph Hellwig

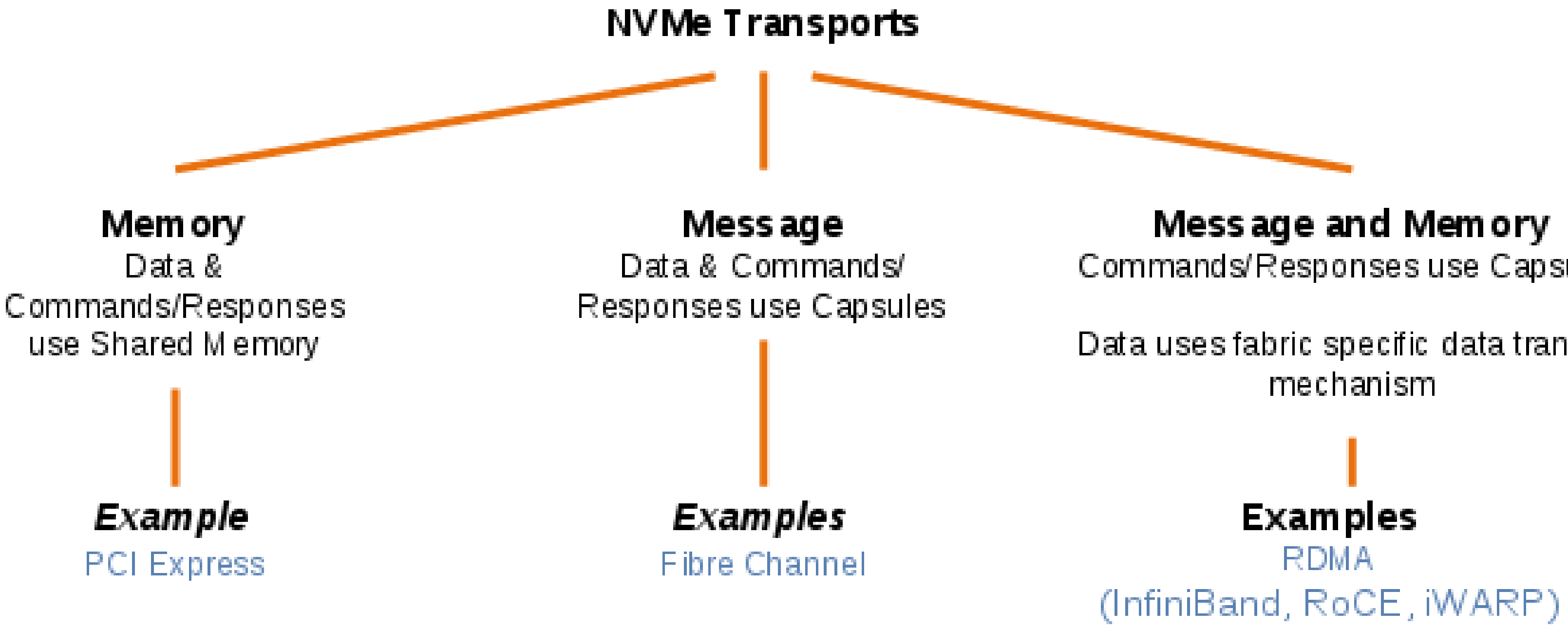
Introduction to NVMe

- NVM Express (NVMe) originally was a vendor-independent interface for PCIe storage devices (usually Flash)
- NVMe uses a command set that gets sent to multiple queues (one per CPU in the best case)
- NVMe creates these queues in host memory and uses PCIe MMIO transactions to communicate them with the device

NVMe over Fabrics

- Is a way to send NVMe commands over networking protocols (“Fabrics”). E.g.
  - RDMA (Infiniband, iWarp, RoCE, ..)
  - Fibre Channel
- At this point still worded as an add-on to the NVMe spec and not fully integrated with the PCIe version.

NVMe Transports



NVMe Transports

Memory
Data & Commands/Responses use Shared Memory

Example
PCI Express

Message
Data & Commands/ Responses use Capsules

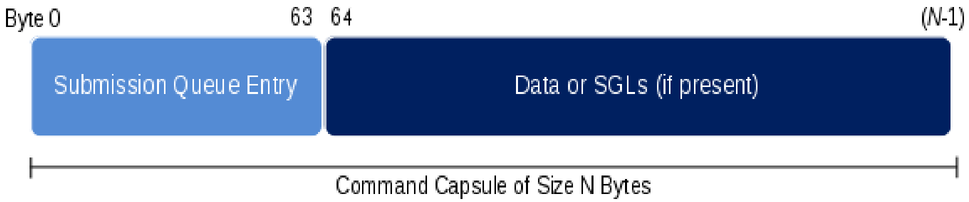
Examples
Fibre Channel

Message and Memory
Commands/Responses use Capsules

Data uses fabric specific data transfer mechanism

Examples
RDMA
(InfiniBand, RoCE, iWARP)

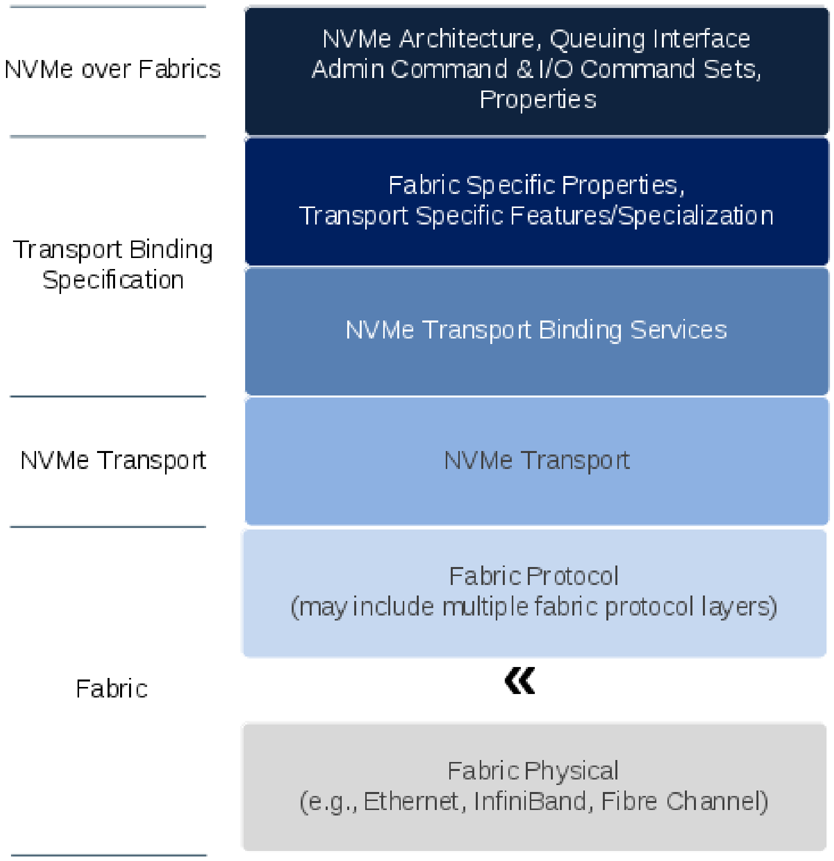
Capsules



Each Capsule sends the NVMe submission queue entry (aka command) plus an optional payload

- Shared memory queues are replaced by capsules
- The queue concept is moved to the transport
- The submission queue entry itself also needs changes as PRPs or simple SGLs don’t work for the Fabrics transports

NVMe over Fabrics layering



NVMe over Fabrics

NVMe Architecture, Queuing Interface
Admin Command & I/O Command Sets,
Properties

Transport Binding Specification

Fabric Specific Properties,
Transport Specific Features/Specialization

NVMe Transport Binding Services

NVMe Transport

NVMe Transport

Fabric

Fabric Protocol
(may include multiple fabric protocol layers)

«

Fabric Physical
(e.g., Ethernet, InfiniBand, Fibre Channel)

Fabrics Commands

- NVMe traditionally uses MMIO registers for initialization
- NVMe over Fabrics instead adds new “Fabrics” commands to create queues and get or set properties:
  - **Connect**
  - **Property Set**
  - **Property Get**

Discovery

- NVMe traditionally uses the PCIe bus for enumeration, on Fabrics we need a way to find available NVMe controllers:
  - New concept of a discovery controller

NVMe over RDMA

- Uses RDMA technologies using IB Verbs to transport NVMe packets
- Uses RDMA/CM to establish connections
- Normal I/O path is to register the memory on the host (client) and perform RDMA READ/WRITE operations from/to it on the target.
- Also allows inline data in the command submission

NVMe over Fabrics in Linux

- Initially there were at least two implementations: Intel (+ a few others) and HGST.
- Initial HGST prototype:
  - simply tunnel NVMe commands over the existing SRP protocol
  - Then tried to accommodate the existing draft spec where possible
  - Where not possible, change the spec.

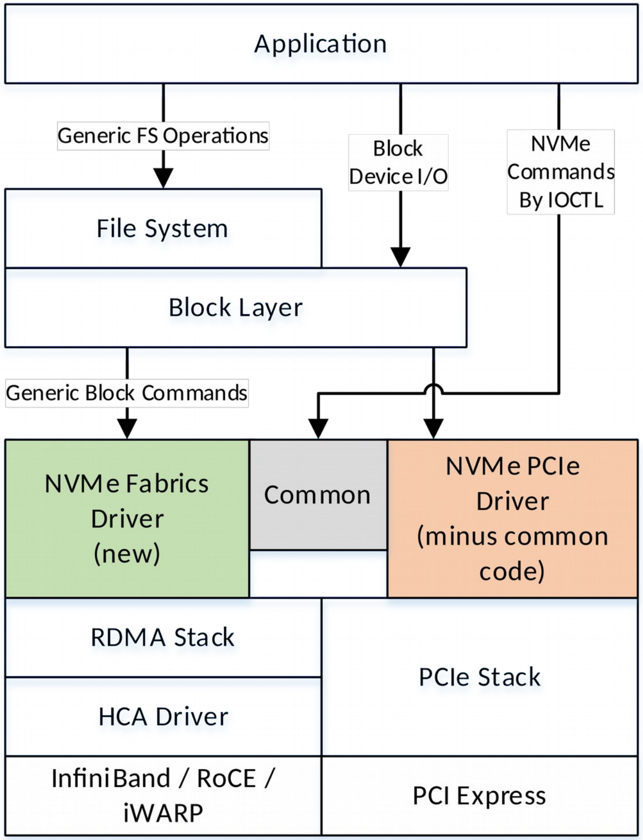
NVMe Linux Fabrics Driver WG

- In 2015 a new working group of the NVM Express organization was created to merge the different Linux development streams.
- Multiple dozen members, with more than a handful actively contributing and even more testing the code base
- Tried to follow Linux-style development as much as possible:
  - Private git repository
  - Mailing list

NVMe Linux Driver

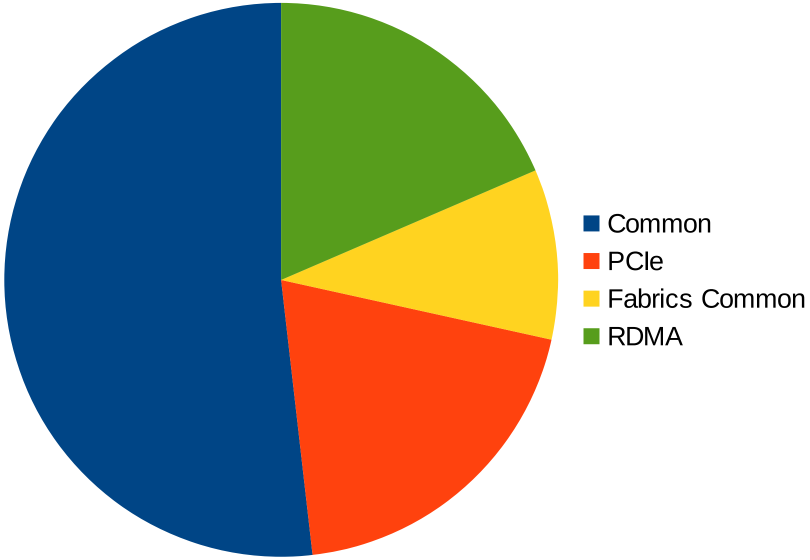
- Even before the release of the spec we started splitting the existing Linux NVMe driver into a common and a PCIe specific part:
  - Use struct request passthrough for NVMe command (similar to SCSI)
  - Separate data structures into common and PCIe
  - Add struct nvme_ctrl_ops
  - And move the code of course

NVMe over Fabrics Host Driver



- The new Fabric drivers uses the existing common code
- Additional it is split into a small common fabrics library and the actual transport driver
- The transport driver is in control of the actual I/O path (no additional indirections for the fast path)
- Existing user space APIs of the PCIe driver are all also supported when using Fabrics
- Uses new sub-command of the existing **nvme-cli** tool to connect to remote controllers

NVMe Linux Host Driver now



- Most code is shared for the different transports
- Transport drivers are fairly small (~2000 lines of code)

NVMe Target

- Supports implementing NVMe controllers in the Linux kernel
  - Initially just NVMe over Fabrics
  - Adding real PCIe support (e.g. using vhost) could be done later
- Split into a generic target and transport drivers:
  - RDMA
  - Loop (for local testing)

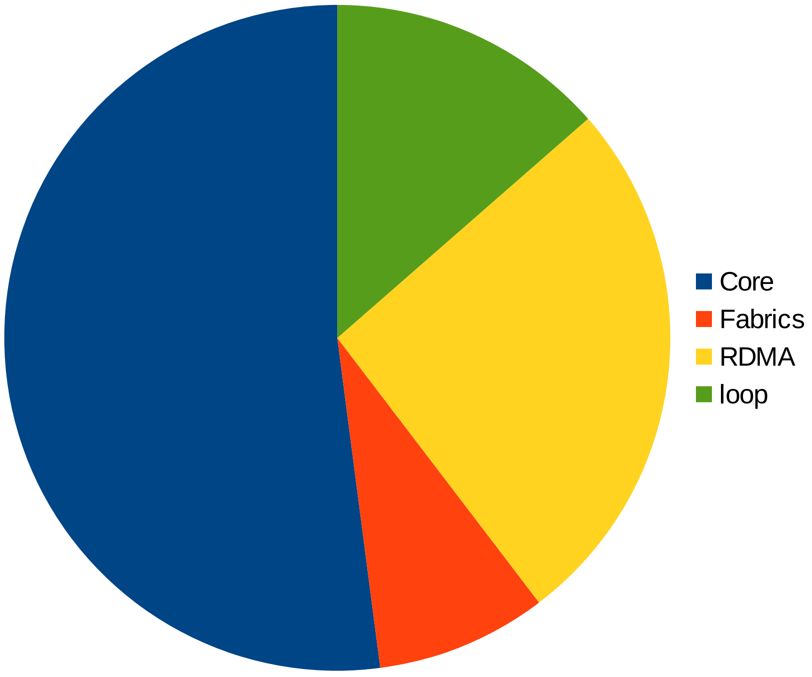
NVMe Target

- The NVMe target can use any Linux block device (NVMe, SCSI, SATA, ramdisk, virtio)
  - Uses the block layer to communicate with the device
  - Early experiments with NVMe command passthrough not continued

NVMe Target

- Initially implemented the bare minimum of required NVMe commands:
  - READ, WRITE, FLUSH + admin command
  - We now also support DSM (aka discard)
  - More functionality (e.g. Persistent Reservations is planned)

NVMe Target

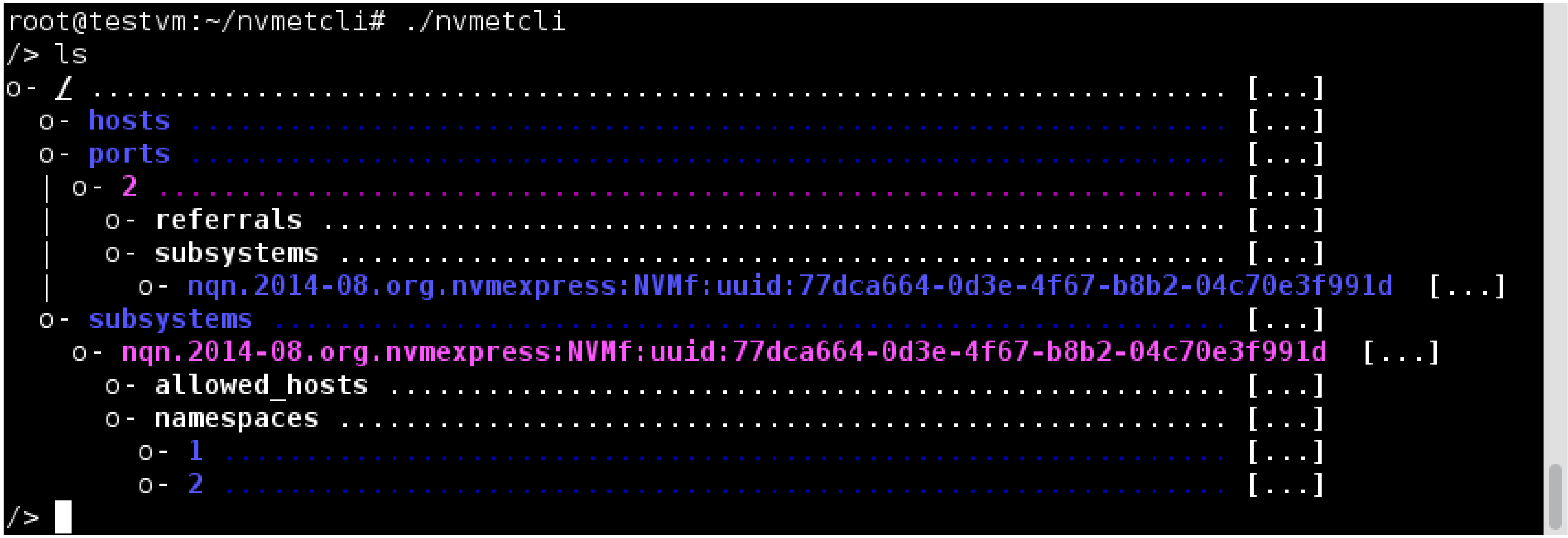


- Again most code is in the core
- The whole core (~ 3000 lines of code) is smaller than many SCSI target transport drivers
- We agressively tried offloading code to common libraries (e.g. RDMA R/W API, configfs improvements) and will continue to do so for new features (e.g. Persistent Reservations)

NVMe Target – configuration

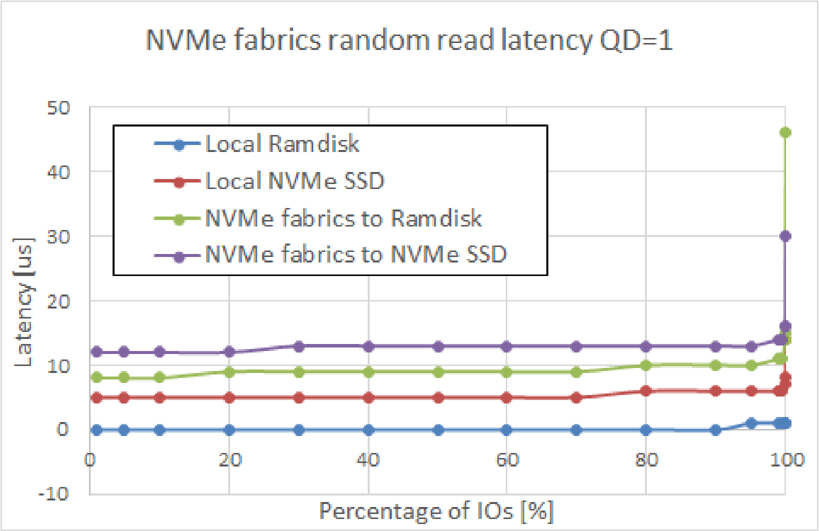
- Uses a configfs interface to let user space tools configure the tool.
  - Simpler and more integrated than the SCSI target
- The prime user space tool is called nvmetcli and is written in python
  - Allows interactive configuration using a console interface
  - Allows saving configurations into json format and restoring them

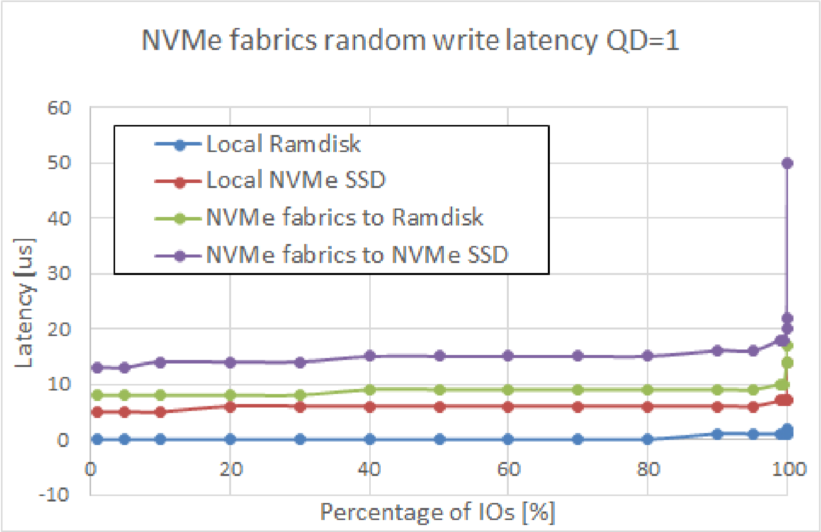
Nvmetcli



```
root@testvm:~/nvmetcli# ./nvmetcli
/> ls
o- / ..................................................................... [...]
  o- hosts ............................................................... [...]
  o- ports ............................................................... [...]
  | o- 2 ................................................................. [...]
  |   o- referrals ....................................................... [...]
  |   o- subsystems ...................................................... [...]
  |     o- nqn.2014-08.org.nvmexpress:NVMf:uuid:77dca664-0d3e-4f67-b8b2-04c70e3f991d  [...]
  o- subsystems .......................................................... [...]
    o- nqn.2014-08.org.nvmexpress:NVMf:uuid:77dca664-0d3e-4f67-b8b2-04c70e3f991d  [...]
      o- allowed_hosts ................................................... [...]
      o- namespaces ...................................................... [...]
        o- 1 ............................................................. [...]
        o- 2 ............................................................. [...]
/>
```

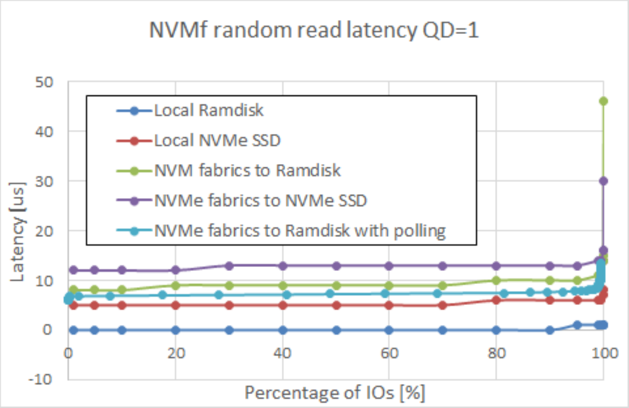
Initial Performance Measurements

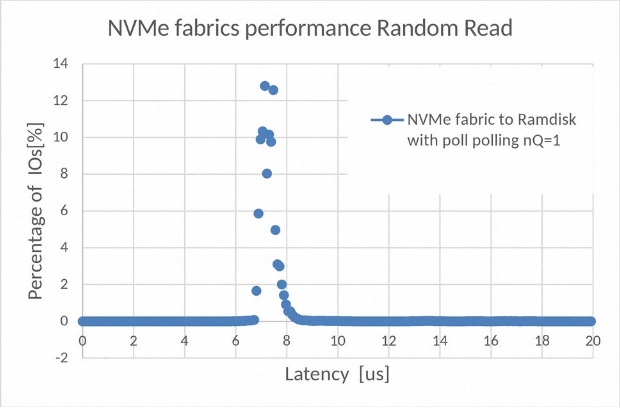




- 13us latency for QD=1 random reads
  - Sub-10us network contribution

Performance Measurements (2)





- Polling allows for sub-7us added latency

Status

- All code mentioned is in the block maintainer tree and should be merged in Linux 4.8
- Fibre Channel support for both the host and target will be submitted soon
- The updated nvme-cli with Fabrics support and nvmetcli need to get into Distributions

Links

- Block layer git tree with NVMe over Fabrics support:
  - http://git.kernel.dk/cgit/linux-block/log/?h=for-next
- Nvme-cli repository:
  - http://github.com/linux-nvme/nvme-cli/
- Nvmetcli repository:
  - http://git.infradead.org/users/hch/nvmetcli.git

Questions?