Let’s Talk “Fabrics”

J Metz, Ph.D
Twitter: @drjmetz
R&D Engineer, Advanced Storage, Cisco
Board of Directors, SNIA
Board of Directors, NVM Express, Inc.
Board of Directors, FCIA

June 27, 2018
Agenda

- NVMe Refresher
- NVMe-oF Refresher
- “Fabrics!”
  - NVMe/RDMA
  - NVMe/FC
  - NVMe/TCP
- Summary
Agenda

• Special Thanks to:
  • Dave Minturn (Intel)
  • David Black (Dell/EMC)
  • Muli Ben-Yehuda/Kam Eshghi (Lightbits Labs)
  • Fred Knight (NetApp)
  • Craig Carlson (Cavium)
NVMe Refresher
NVM Express (NVMe™) is an open collection of standards and information to fully expose the benefits of non-volatile memory in all types of computing environments from mobile to data center.

**NVM Express Base Specification**

The register interface and command set for PCI Express attached storage with industry standard software available for numerous operating systems. NVMe™ is widely considered the defacto industry standard for PCIe SSDs.

**NVM Express Management Interface (NVMe-MI™) Specification**

The command set and architecture for out of band management of NVM Express storage (i.e., discovering, monitoring, and updating NVMe™ devices using a BMC).

**NVM Express Over Fabrics (NVMe-oF™) Specification**

The extension to NVM Express that enables tunneling the NVM Express command set over additional transports beyond PCIe. NVMe over Fabrics™ extends the benefits of efficient storage architecture at scale in the world’s largest data centers by allowing the same protocol to extend over various networked interfaces.
NVMe…

- Specification for SSD access via PCI Express (PCIe)
- High parallelism and low latency SSD access
- New modern command set with administrative vs. I/O command separation (control path vs. data path)
- Full support for NVMe for all major OS (Linux, Windows, ESX etc.)
NVMe Multi-Queue Interface

- I/O Submission and Completion Queue Pairs are aligned to Host CPU Cores
  - Independent per-queue operations
  - No inter-CPU locks on command Submission or Completion
  - Per Completion Queue Interrupts enables source core interrupt steering
Each Host/Controller pair have an independent set of NVMe queues
- Controllers and queues operate autonomously

NVMe Controllers may be local PCIe or remote Fabric
- Use a common NVMe Queuing Model
NVMe Commands and Completions

- NVMe Commands are sent by the Host to the Controller in Submission Queue Entries (SQE)
  - Separate Admin and IO Commands
  - Three mandatory IO Commands
  - Added two fabric-only Commands
  - Commands may complete out of order

- NVMe Completions are sent by the Controller to the Host in Completion Queue Entries (CQE)
  - Command Id identifies the completed command
  - SQ Head Ptr indicates the consumed SQE slots that are available for posting new SQEs
1. Host Driver enqueues the SQE into the SQ
2. NVMe Controller dequeues SQE
3. NVMe Controller enqueues CQE into the CQ
4. Host Driver dequeues CQE

This queuing functionality is always present... ... but *where* this takes place can differ
1. Host Driver enqueues the SQE in host-memory resident SQ.
1. Host Driver enqueues the SQE in host-memory resident SQ
2. Host Driver notifies controller about new SQE by writing doorbell register
NVMe Queuing on Memory (PCIe)

1. Host Driver enqueues the SQE in host-memory resident SQ
2. Host Driver notifies controller about new SQE by writing doorbell register
3. NVMe Controller dequeues SQE by reading it from the host memory SQ
1. Host Driver enqueues the SQE in host-memory resident SQ
2. Host Driver notifies controller about new SQE by writing doorbell register
3. NVMe Controller dequeues SQE by reading it from the host memory SQ

Data transfer, if applicable, goes here
NVMe Queuing on Memory (PCIe)

1. Host Driver enqueues the SQE in host-memory resident SQ
2. Host Driver notifies controller about new SQE by writing doorbell register
3. NVMe Controller dequeues SQE by reading it from the host memory SQ
4. NVMe Controller enqueues CQE by writing it to host-resident CQ
1. Host Driver enqueues the SQE in host-memory resident SQ
2. Host Driver notifies controller about new SQE by writing doorbell register
3. NVMe Controller dequeues SQE by reading it from the host memory SQ
4. NVMe Controller enqueues CQE by writing it to host-resident CQ
5. Host Driver dequeues CQE
NVMe-oF Refresher
What’s Special About NVMe over Fabrics?

- **Recall:**
  - Multi-queue model
  - Multipathing capabilities built-in

- **Optimized NVMe System**
  - Architecture is the same, regardless of transport
  - Extends efficiencies across fabric
NVMe and NVMe-oF Models

- NVMe is a Memory-Mapped, PCIe Model
- Fabrics is a message-based transport; no shared memory

**NVMe Transports**

- **Memory**
  - Data & Commands/Responses use shared memory
  - Example: **PCI Express**

- **Message**
  - Data & Commands/Responses use capsules
  - Example: **Fibre Channel**

- **Message & Memory**
  - Commands/Responses use capsules
  - Data uses fabric-specific data transfer mechanism
  - Example: **RDMA** (InfiniBand, RoCE, iWARP)

**Fabric Message-Based Transports**

- **Capsule** = Encapsulated NVMe Command/Completion within a transport message
- **Data** = Transport data exchange mechanism (if any)
Key Differences Between NVMe and NVMe-oF

- One-to-one mapping between I/O Submission Queues and I/O Completion Queues
- A controller is associated with only one host at a time
- NVMe over Fabrics does not define an interrupt mechanism that allows a controller to generate a host interrupt
- NVMe over Fabrics does not support PRPs but requires use of SGLs for Admin, I/O, and Fabrics commands
- NVMe over Fabrics does not support Completion Queue flow control
Controller initiates the Read or Write of the NVMe Command Data to/from Host Memory Buffer

Data transfer operations are transport specific; examples

- PCIe Transport: PCIe Read/ PCIe Write Operations
- RDMA Transport: RDMA_READ/ RDMA_WRITE Operations
- TCP Transport: H2CData/C2HData operations

© 2018 Storage Networking Industry Association. All Rights Reserved.
NVMe Command Data Transfers (In-Capsule Data)

- NVMe Command and Command Data sent together in Command Capsule
- Reduces latency by avoiding the Controller having to fetch the data from Host
- SQE SGL Entry will indicate Capsule Offset type address
- Supported in NVMe/FC, and NVMe/TCP (optionally)
Understanding “Fabrics”
What’s Special About NVMe-oF: Bindings

What is a Binding?

“A specification of reliable delivery of data, commands, and responses between a host and an NVM subsystem for an NVMe Transport. The binding may exclude or restrict functionality based on the NVMe Transport’s capabilities”

I.e., it’s the “glue” that links all the pieces above and below (examples):

- SGL Descriptions
- Data placement restrictions
- Data transport capabilities
- Authentication capabilities
Building Networks For A Reason

- You do not need to *and* should not be designing a network that requires a lot of buffering.
- Capacity and over-subscription is *not* a function of the protocol (RDMA, FC, TCP, etc) but of the application I/O requirements.

Data Center Design Goal: Optimizing the balance of end to end fabric latency with the ability to absorb traffic peaks and prevent any associated traffic loss.

Note: Watch for upcoming SNIA webinars!
What is Remote Direct Memory Access (RDMA)?

- RDMA is a host-offload, host-bypass technology that allows an application (including storage) to make data transfers directly to/from another application’s memory space.
- The RDMA-capable Ethernet NICs (RNICs) – not the host – manage reliable connections between source and destination.
- Applications communicate with the RDMA NIC using dedicated Queue Pairs (QPs) and Completion Queues (CQs).
  - Each application can have many QPs and CQs.
  - Each QP has a Send Queue (SQ) and Receive Queue (RQ).
  - Each CQ can be associated with multiple SQs or RQs.

*See also:* “How Ethernet RDMA Protocols iWARP and RoCE Support NVMe over Fabrics”
Benefits of RDMA

- Bypass of system software stack components that processes network traffic
  - For user applications (outer rails), RDMA bypasses the kernel altogether
  - For kernel applications (inner rails), RDMA bypasses the OS stack and the system drivers

- Direct data placement of data from one machine (real or virtual) to another machine – without copies

- Increased bandwidth while lowering latency, jitter, and CPU utilization

- Great for networked storage!
Queues, Capsules, and More Queues
Example of Host Write To Remote Target

- NVMe Host Driver encapsulates the NVMe Submission Queue Entry (including data) into a fabric-neutral Command Capsule and passes it to the NVMe RDMA Transport
- Capsules are placed in Host RNIC RDMA Send Queue and become an RDMA_SEND payload
- Target RNIC at a Fabric Port receives Capsule in an RDMA Receive Queue
- RNIC places the Capsule SQE and data into target host memory
- RNIC signals the RDMA Receive Completion to the target’s NVMe RDMA Transport
- Target processes NVMe Command and Data
- Target encapsulates the NVMe Completion Entry into a fabric-neutral Response Capsule and passes it to NVMe RDMA Transport

Source: SNIA
NVMe Multi-Queue Host Interface Map to RDMA Queue-Pair Model

**Standard (local) NVMe**
- NVMe Submission and Completion Queues are aligned to CPU cores
- No inter-CPU software locks
- Per CQ MSI-X interrupts enable source core interrupt steering

**NVMe Over RDMA Fabric**
- Retains NVMe SQ/CQ CPU alignment
- No inter-CPU software locks
- Source core interrupt steering retained by using RDMA Event Queue MSI-X interrupts
NVMe/FC
Fibre Channel has layers, just like OSI and TCP

- At the top level is the Fibre Channel Protocol (FCP)
- Integrates with upper layer protocols, such as SCSI, FICON, and NVMe

See also: “Introducing FC-NVMe.” FCIA; fibrechannel.org
What Is FCP?

- What’s the difference between FCP and “FCP”? 
  - FCP is a data transfer protocol that carries other upper-level transport protocols (e.g., FICON, SCSI, NVMe)
  - Historically FCP meant SCSI FCP, but other protocols exist now

- NVMe “hooks” into FCP 
  - Seamless transport of NVMe traffic 
  - Allows high performance HBA’s to work with FC-NVMe
The NVMe Command/Response capsules, and for some commands, data transfer, are directly mapped into FCP Information Units (IUs).

A NVMe I/O operation is directly mapped to a Fibre Channel Exchange.
FC-NVMe Information Units (IUs)

1. NVMe Submission Queue Entry (SQE) is mapped to a FCP Command IU

   SQE → FCP Command IU

2. Data to a FCP Data IU

   Data → FCP Command IU(s) → FCP Data IU(s) → FCP Data IU(s)

3. NVMe Completion Queue Entry (CQE) to a FCP Response IU

   CQE → FCP Response IU
Zero Copy

- RDMA is a semantic which encourages more efficient data handling, but you don’t need it to get efficiency.
- FC had zero-copy years before there was RDMA
  - Data is DMA’d straight from HBA to buffers passed to user.
- Difference between RDMA and FC is the APIs
  - RDMA does a lot more to enforce a zero-copy mechanism, but it is not required to use RDMA to get zero-copy.
FCP Transactions

- NVMe-oF using Fibre Channel Transactions look similar to RDMA
  - For Read
    - FCP_DATA from Target
  - For Write
    - Transfer Ready and then DATA to Target
RDMA Transactions

- NVMe-oF over RDMA protocol transactions
  - RDMA Write
  - RDMA Read with RDMA Read Response
NVMe-TCP

- NVMe™ block storage protocol over standard TCP/IP transport
- Enables disaggregation of NVMe™ SSDs without compromising latency and without requiring changes to networking infrastructure
- Independently scale storage & compute to maximize resource utilization and optimize for specific workload requirements
- Maintains NVMe™ model: sub-systems, controllers namespaces, admin queues, data queues
NVMe/TCP in a nutshell

- NVMe-OF Commands sent over standard TCP/IP sockets
- Each NVMe queue pair mapped to a TCP connection
- TCP provides a reliable transport layer for NVMe queueing model
NVMe-TCP Data Path Usage

- Enables NVMe-oF I/O operations in existing IP Datacenter environments
  - Software-only NVMe Host Driver with NVMe-TCP transport

- Provides an NVMe-oF alternative to iSCSI for Storage Systems with PCIe NVMe SSDs
  - More efficient End-to-End NVMe Operations by eliminating SCSI to NVMe translations

- Co-exists with other NVMe-oF transports
  - Transport selection may be based on h/w support and/or policy
NVMe-TCP Control Path Usage

- Enables use of NVMe-oF on Control-Path Networks (example: 1g Ethernet)
- Discovery Service Usage
  - Discovery controllers residing on a common control network that is separate from data-path networks
- NVMe-MI Usage
  - NVMe-MI endpoints on control processors (BMC, ..) with simple IP network stacks
  - NVMe-MI on separate control network

© 2018 Storage Networking Industry Association. All Rights Reserved.
Data transfers supported by:
- Fabric-specific data transfer mechanism
- In-Capsule data (optional)

All NVMe/TCP implementations support data transfers using command data buffers
- In-capsule data transfer is optional
Potential Issues with NVMe/TCP

Absolute latency higher than RDMA?
Head-of-line blocking leading to increased latency?
Delayed acks could increase latency?

Incast could be an issue?

Lack of hardware acceleration?

Only matters if the application cares about latency
Protocol breaks up large transfers
Acks are used to ‘pace the transmission of packets such that TCP is “self-clocking”
Switching network can provide Approximate Fair Drop (AFD) for active switching queue mgmt, and Dynamic Packet Prioritization (DPP) to ensure incast flows are serviced as fast as possible

Not an issue for NVMe/TCP use-cases
TCP Stacks

- The TCP stacks that rely on drops (most common stacks) are the ones that require proper network buffering.
- Newer stacks looking at RTT or other feedback loops to monitor throughput are optimizing for ‘zero buffer’ networks.
  - e.g. DCTCP leverages quantitative feedback to prevent any packet drops.
- Helps to know which stacks you are using.
  - Note: SNIA ESF is working on webinars for this very subject! Stay tuned!
Expect NVMe over TCP standard to be ratified in 2H 2018

- The NVMe-oF 1.1 TCP ballot passed in April 2017
- NVMe Workgroup adding TCP to spec alongside RDMA

Source: Kam Eshghi (Lightbits Labs)
Summary
Summary

• NVMe and NVMe-oF
  • Treats storage like memory, just with permanence
  • Built from the ground up to support a consistent model for NVM interfaces, even across network fabrics
  • No translation to or from another protocol like SCSI (in firmware/software)
  • Inherent parallelism of NVMe multiple I/O Queues is exposed to the host
  • NVMe commands and structures are transferred end-to-end, and architecture is maintained across a range of fabric types