NVMe™/TCP: What You Need to Know About the Specification

Sagi Grimberg
Lightbits Labs

Peter Onufryk
Microsemi Corp.

NVMe™ Transport Evolution

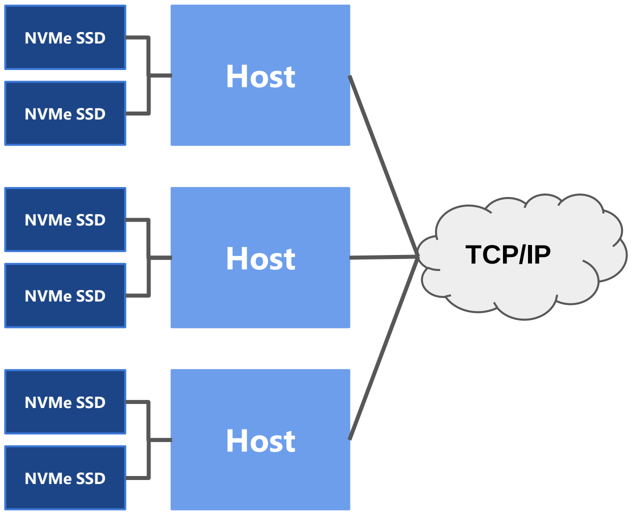
- NVM Express™ (NVMe) standard released in March 2011
  - Architecture, command set, and queueing interface for PCIe SSDs
    - Optimized for direct attached NVM PCIe® SSDs
    - The goal was a single interface that is scalable from client to enterprise
- NVMe™ over Fabrics (NVMe-oF™) standard released in June 2016
  - Extends the architecture, command set, and queueing interface of NVMe to fabric attached storage
  - Two initial transport binding specifications
    - RDMA (Infiniband, RoCE, iWARP)
    - Fibre Channel
- TCP transport binding specification (NVMe™/TCP) specified for NVMe-oF in November of 2018

Why Do We Need Another NVMe™ Transport?

- PCIe®
  - Great for direct attached NVMe SSDs
  - Does not scale well to large topologies
- RDMA (Infiniband, RoCE, iWARP) and Fibre Channel
  - Provides a high degree of scalability, but requires special networks and hardware
- TCP
  - Ubiquitous (does not require special networks or hardware)
  - Scalable allowing large scale deployments and operation over long distances
  - Can provide performance (throughput and latency) that is comparable to direct attached NVMe SSDs

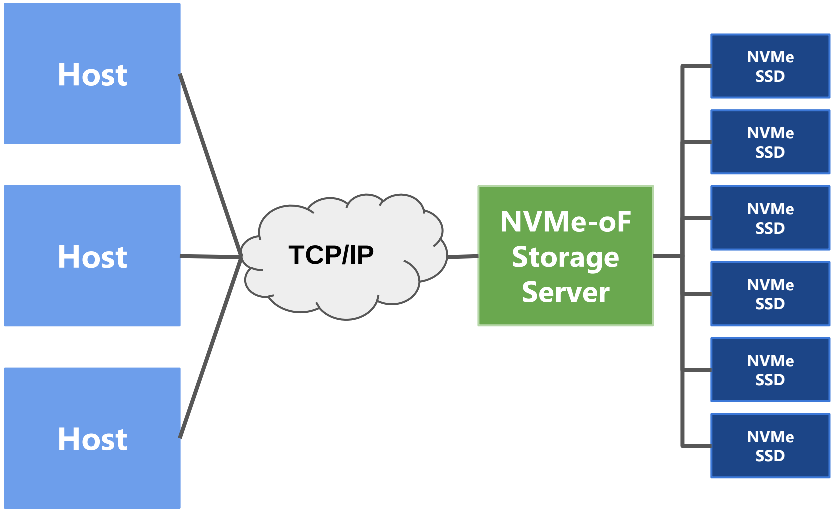
Use Cases for NVMe™/TCP

Direct Attached Architecture



- High Performance and Low latency
- Fixed Storage and Compute Ratios
- Stranded Capacity

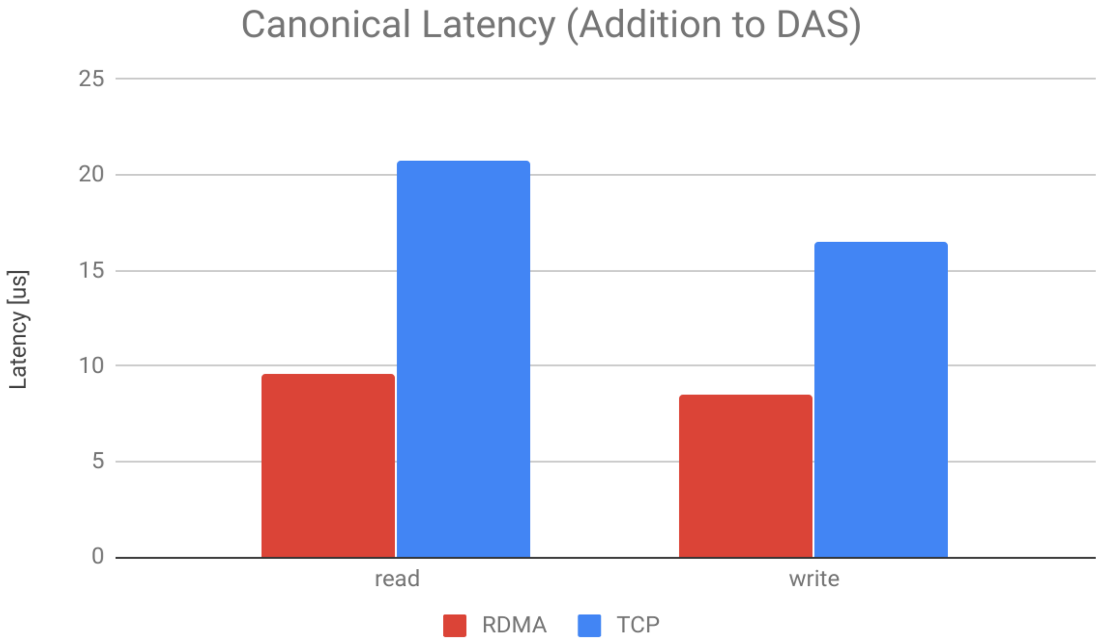
Disaggregated Architecture



- Large Scale Pooled Storage
- Independent Scaling of Storage and Compute
- Near DAS Performance and Latency

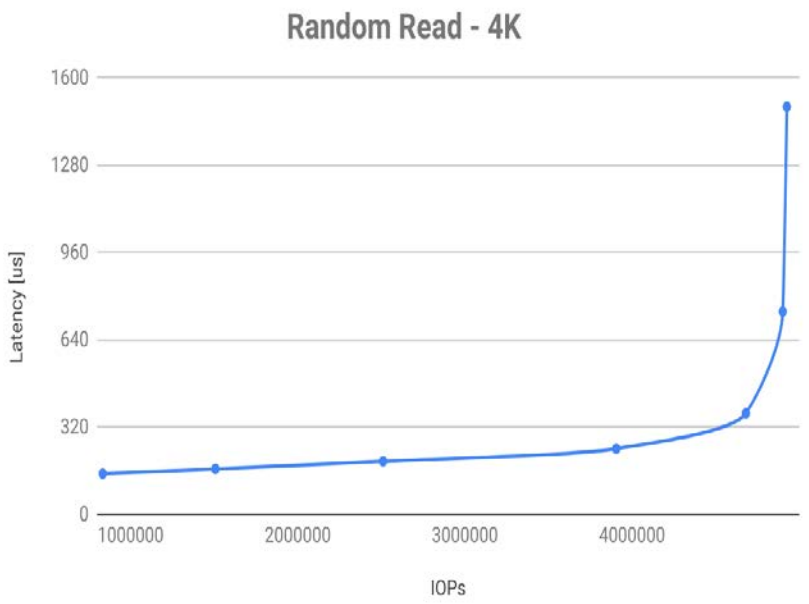
Open Source Performance

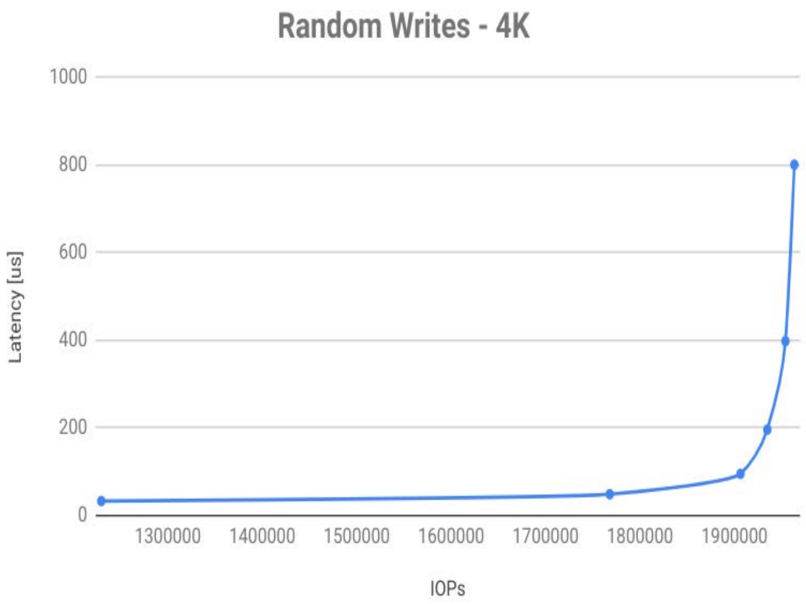
Upstream Linux kernel NVMe™/TCP vs. NVMe/RDMA (added latency over direct attached PCIe® SSD)



Commercial Performance

Software NVMe™/TCP controller performance (IOPs vs. Latency)*

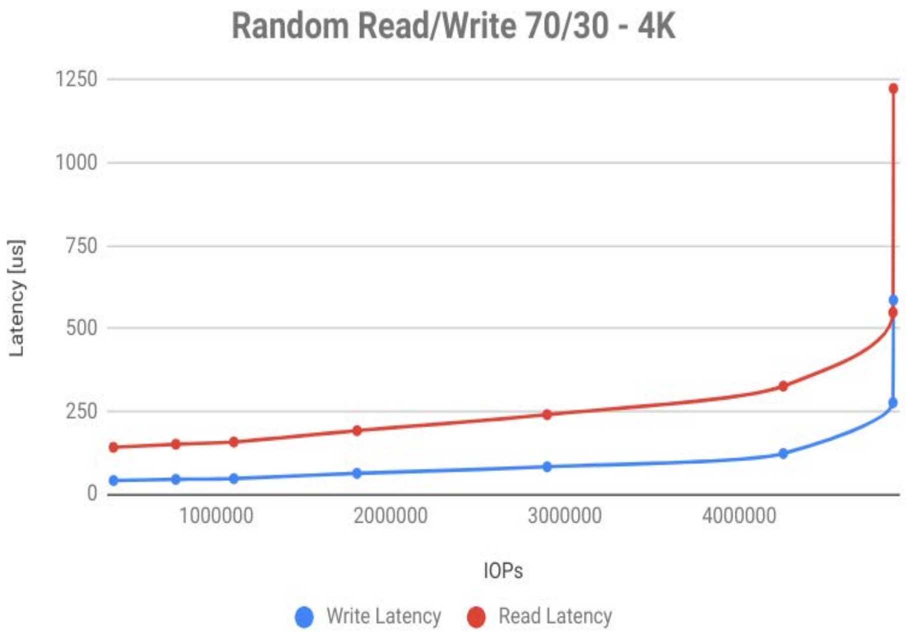


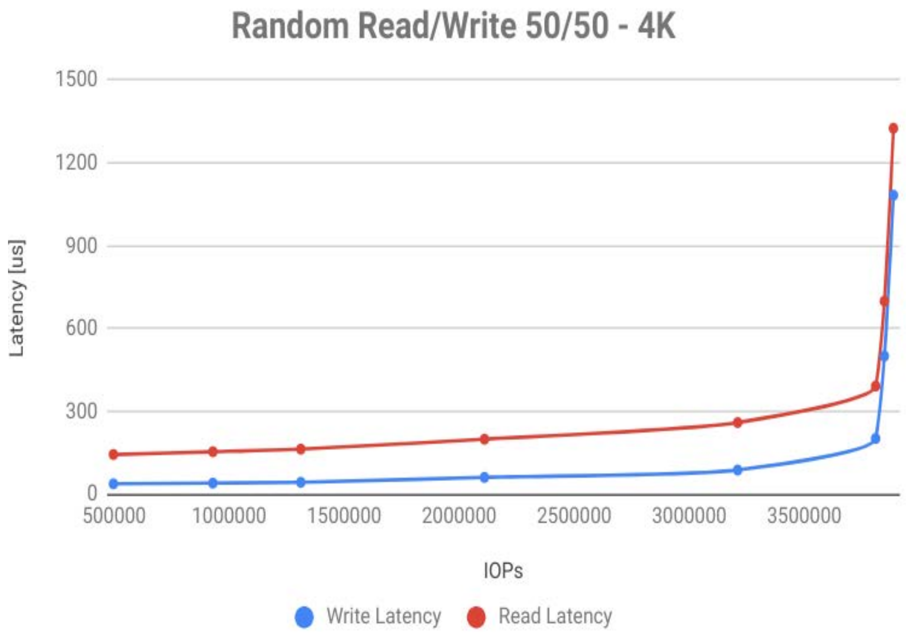


* Commercial single 2U NVM subsystem that implements RAID and compression with 8 attached hosts

Commercial Performance – Mixed Workloads

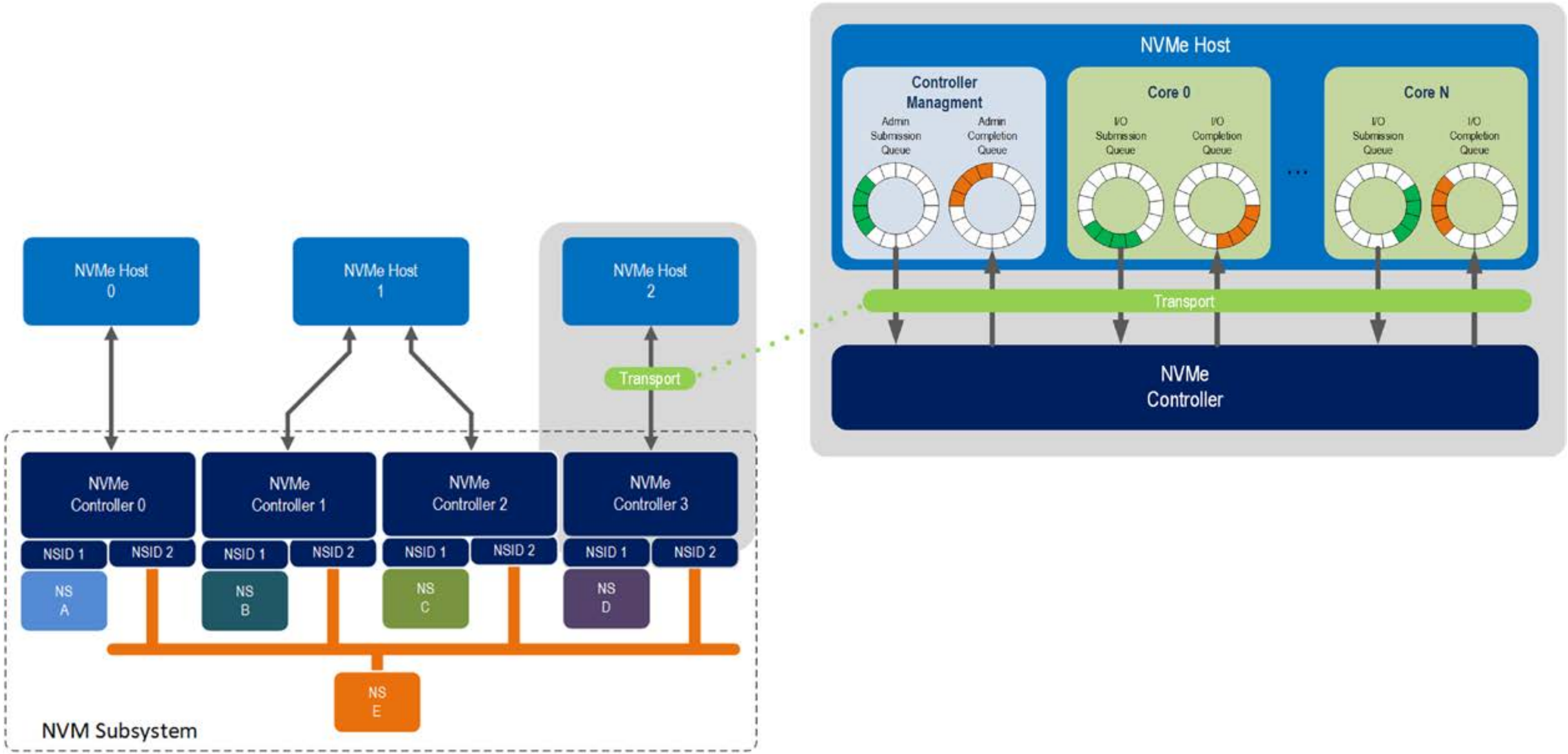
Software NVMe™/TCP Controller performance (IOPs vs. Latency)*



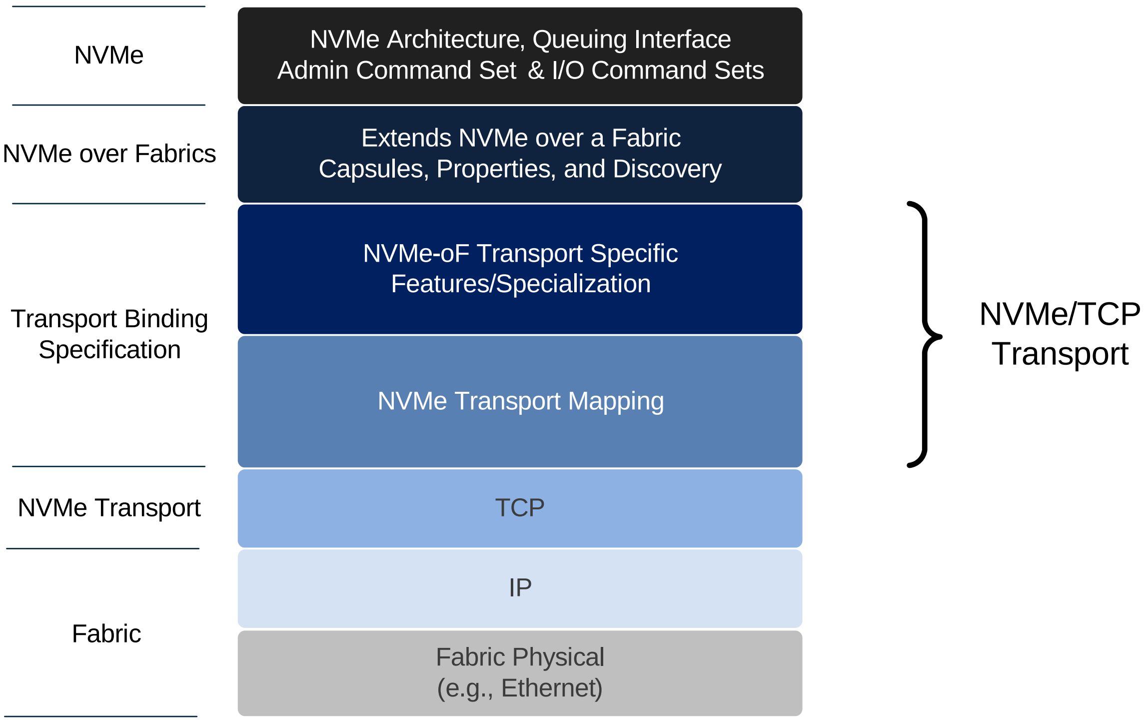


* Commercial single 2U NVM subsystem that implements RAID and compression with 8 attached hosts

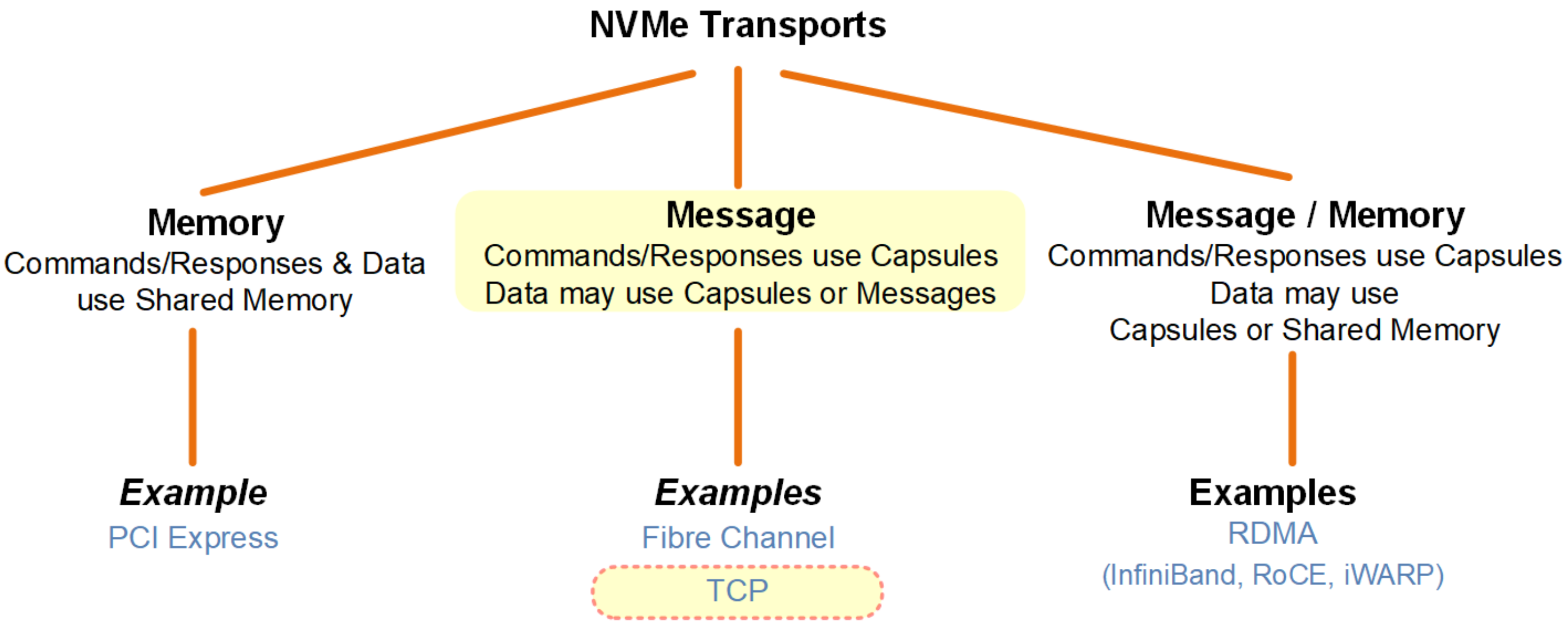
NVMe™ Architecture



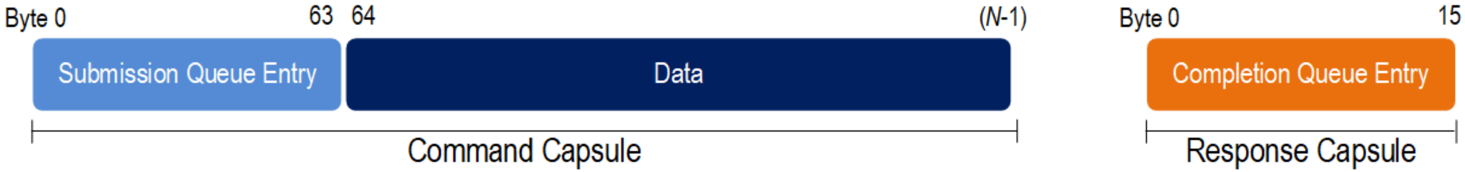
NVMe-oF™ Protocol Layering



NVMe™ Transport Taxonomy



NVMe™/TCP Capsules



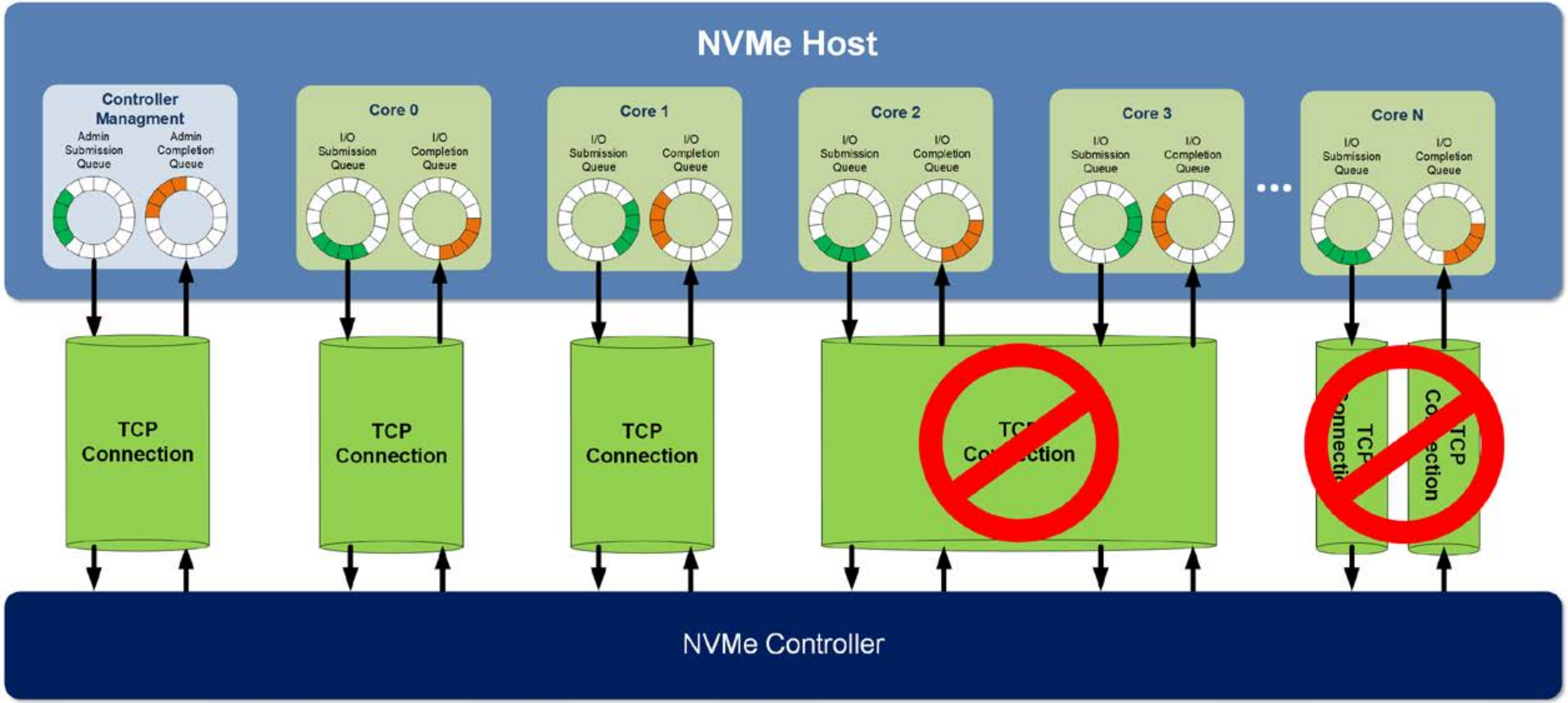
Capsule Type	Capsule Size Without Data	Max Capsule Size With Data
Fabrics or Admin Command	64-bytes	8256 bytes
I/O Command	64 bytes	64 bytes to (IOCCSZ[a] * 16) bytes
Fabrics or Admin Response	16 bytes	Not Applicable
I/O Response	16 bytes	Not Applicable

[a] IOCCSZ – I/O Queue Command Capsule Supported Size (IOCCSZ) field in the Identify Controller data structure defines the maximum I/O command capsule size in units of 16 bytes

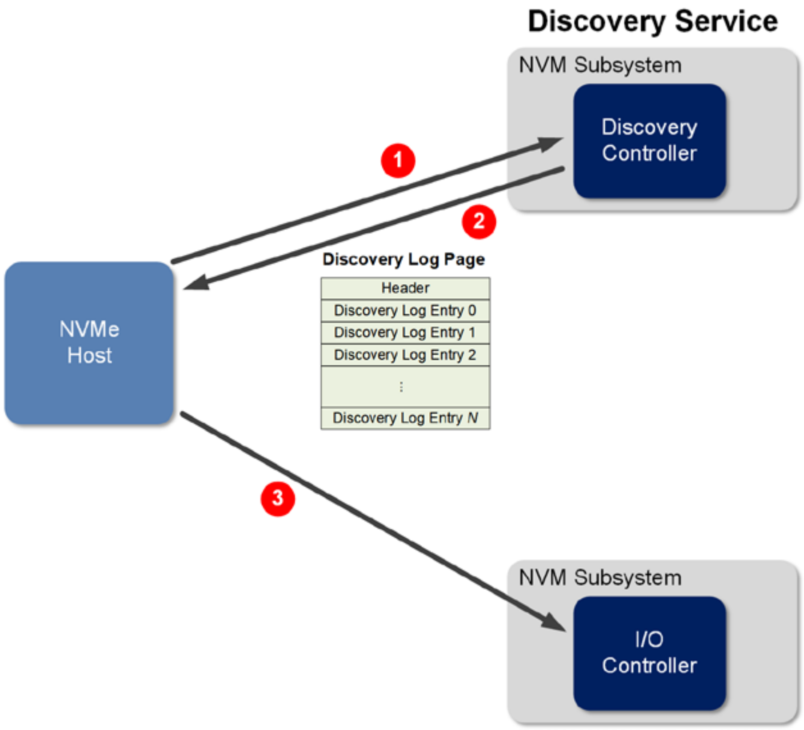
Transmission Control Protocol (TCP) Overview

- Provides a reliable error-free in-order bidirectional byte stream service between a sender and receiver
  - NVMe™/TCP uses this service to send and receive NVMe/TCP Protocol Data Units (PDUs)
- Connection oriented transport
  - Port numbers used to identify sending/receiving application on a host
  - TCP connection defined by source address, source TCP port, destination address, destination TCP port
- TCP uses positive acknowledgement with retransmission and handles
  - Network congestion
  - Packet loss
  - Out-of-order packet delivery

NVMe™/TCP Queue Mapping

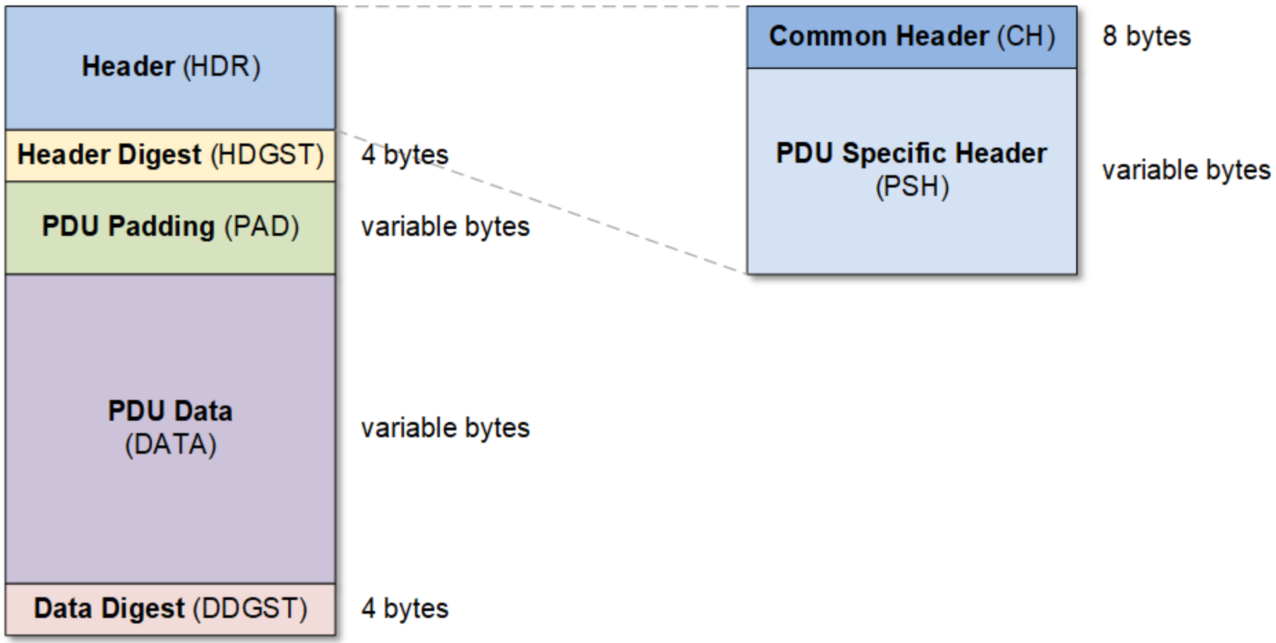


Discovery and Connection

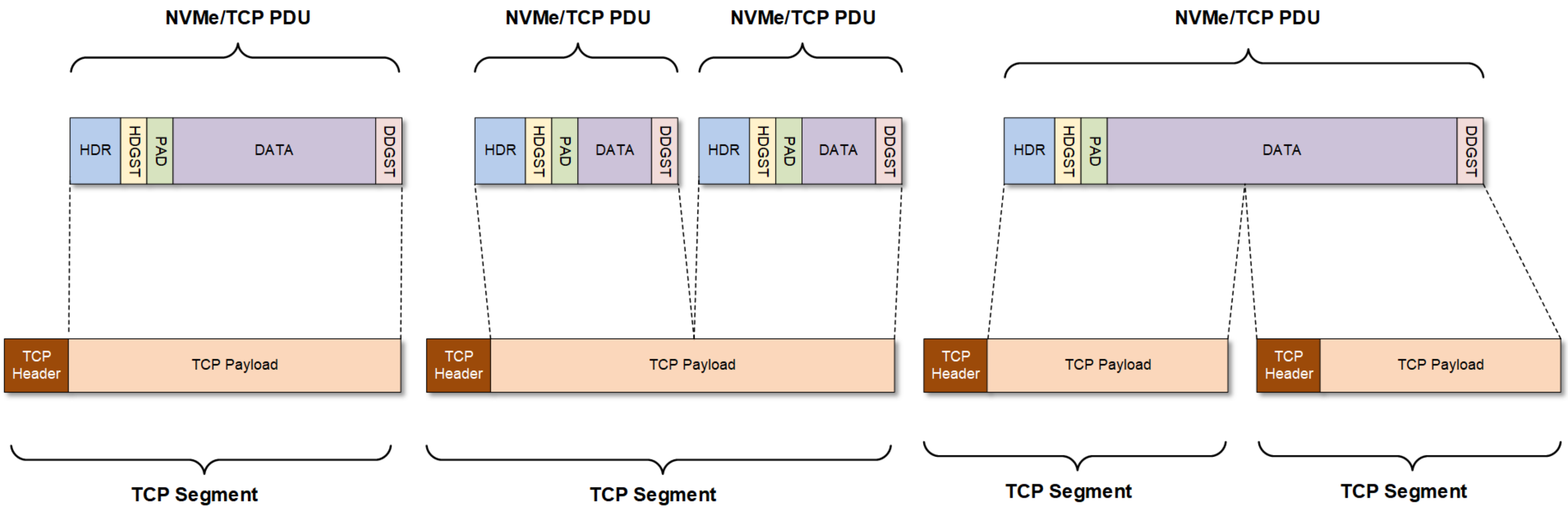


1. Using a preconfigured IP address, NVMe™ hosts connects to a Discovery Controller in a Discovery Service using TCP port 8009
   - Discovery Service is an NVM subsystem that contains Discovery Controllers
   - Discovery Controller is an NVMe Controller with minimal functionality
2. NVMe host reads Discovery Log Page from Discovery Controller
   - Discovery Log Page contains Discovery Log Entries that specify IP address and TCP port number to use to connect to an NVM subsystem
     - Transport Address (TRADDR) field contains IP address
     - Transport Service Identifier (TRSVCID) field contains TCP port number
3. Using information from a Discovery Log Entry, NVMe Host connects to an I/O controller in NVM subsystem
   - Separate TCP connection for Admin queue and for each I/O queue

NVMe™/TCP PDU Structure



Mapping of NVMe™/TCP PDUs to TCP

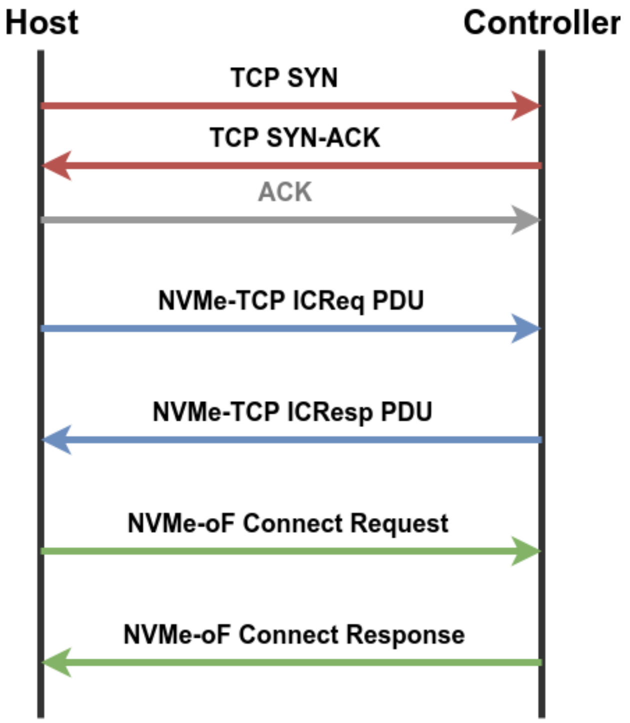


NVMe™/TCP Protocol Data Units (PDUs)

PDU Group	PDU Name	PDU Direction	Description
Initialize Connection	**ICReq**	Host to Controller	**Initialize Connection Request:** A PDU sent from a host to a controller to communicate NVMe/TCP connection parameters and establish an NVMe/TCP connection
	ICResp	Controller to Host	**Initialize Connection Response:** A PDU sent from a controller to a host to accept a connection request and communicate NVMe/TCP connection parameters
Terminate Connection	**H2CTermReq**	Host to Controller	**Host to Controller Terminate Connection Request:** A PDU sent from a host to a controller in response to a fatal transport error
	C2HTermReq	Controller to Host	**Controller to Host Terminate Connection Request:** A PDU sent from a controller to a host in response to a fatal transport error
Capsule Transfer	**CapsuleCmd**	Host to Controller	**Command Capsule:** A PDU sent from a host to a controller to transfer an NVMe over fabrics command capsule
	CapsuleResp	Controller to Host	**Response Capsule:** A PDU sent from a controller to a host to transfer an NVMe over fabrics response capsule
Data Transfer	**H2CData**	Host to Controller	**Host to Controller Data:** A PDU sent from a host to a controller to transfer data to the controller
	C2HData	Controller to Host	**Controller to Host Data:** A PDU sent from a controller to a host to transfer data to the host
	R2T	Controller to Host	**Ready to Transfer:** A PDU sent from a controller to a host to indicate that it is ready to accept data

Connection Establishment

- Stage 1: TCP Connection Establishment
  - General TCP parameters
- Stage 2: NVMe™/TCP Connection Establishment
  - Parameter Negotiation
  - Features Support
- Stage 3: NVMe-oF™ Connection Establishment
  - Controller Binding
  - Queue Sizing

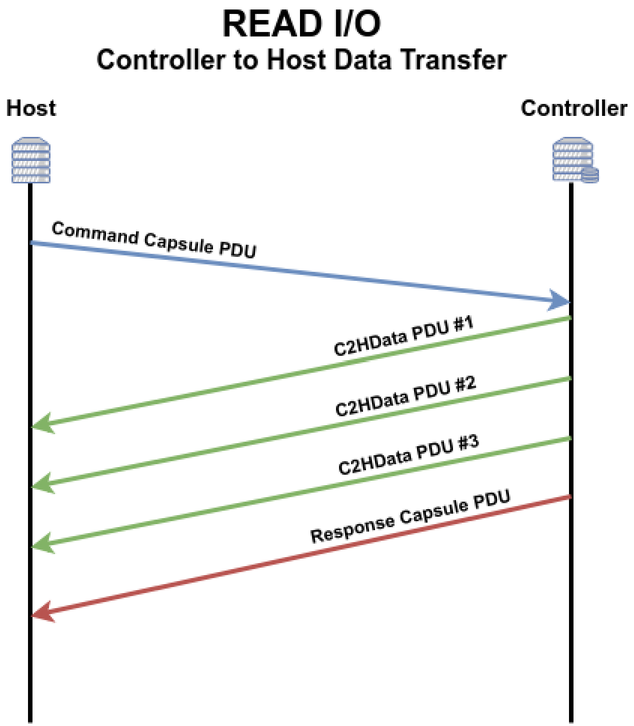


Parameter Negotiation

Section	Role	Attribute	PDU	Description
Protocol Version	Host/Controller	Protocol Format Version (**PFV**)	ICReq/ICResp	Agree on the protocol version supported
Data Alignment	Host	Host PDU Data Alignment (**HPDA**)	ICReq	Host required Data offset alignment from the start of a PDU
	Controller	Controller PDU Data Alignment (**CPDA**)	ICResp	Controller required Data offset alignment from the start of a PDU
Integrity (Digest)	Host/Controller	Required/Supported Header and/or Data Integrity fields (**DGST**)	ICReq/ICResp	Bitmask for Header and/or Data Digest Requested on the connection.
Data Transfer	Host	Maximum Outstanding R2T PDUs (**MAXR2T**)	ICReq	Maximum number of R2T a controller is allowed to send per command
	Controller	Maximum Data Length in a H2CData PDU (**MAXH2CDATA**)	ICResp	Maximum data length a host can send in a single H2CData PDU

Data Transfer – Controller to Host

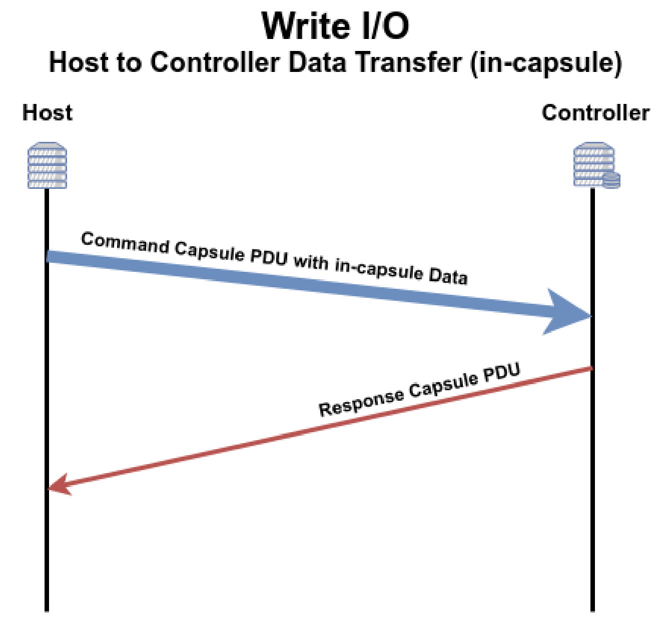
- Host issues a Command Capsule PDU
  - Contains the NVMe™ command
- Controller sends the Data payload to the host
  - Using one or more C2HData PDUs
- Controller sends a Response Casule PDU
  - Usually the NVMe completion entry



Data Transfer – Host to Controller (In-Capsule)

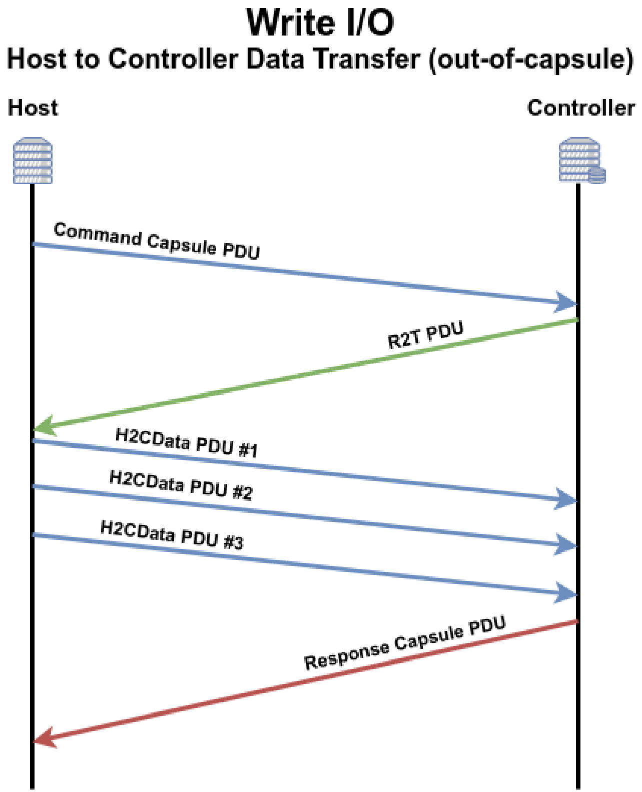
- Host issues a Command Capsule PDU
  - Contains the NVMe™ command
  - Contains in-capsule Data
    - As supported by the Controller

- Controller sends a Response Casule PDU
  - Usually the NVMe completion entry



Data Transfer – Host to Controller (Out-of-Capsule)

- Host issues a Command Capsule PDU
  - Contains the NVMe™ command
- Controller sends a “Ready to Transfer” (R2T) solicitation
  - Host must support at least a single R2T per Command Capsule
- Host sends Data payload for that R2T using one or more H2CData PDUs
- Controller sends a Response Casule PDU
  - Usually the NVMe completion entry

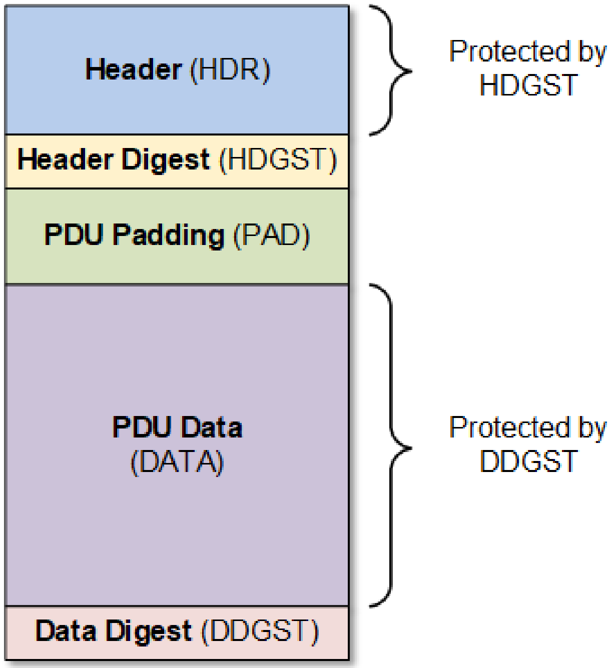


Data Transfer – Rules and Ordering

- Command Capsule Data Delivery must be done in-order
  - No ordering constraints across Command Capsules or NVMe queues
- R2T PDUs must also solicit data in-order
  - Only multiple R2T PDUs within a single Command Capsule
- The last Data PDU in a sequence must set the LAST_PDU flag to indicate the a transfer sequence is complete
- The final C2HData PDU may also set the SUCCESS flag to indicate a successful completion
  - The controller omits a Response Capsule PDU for optimization

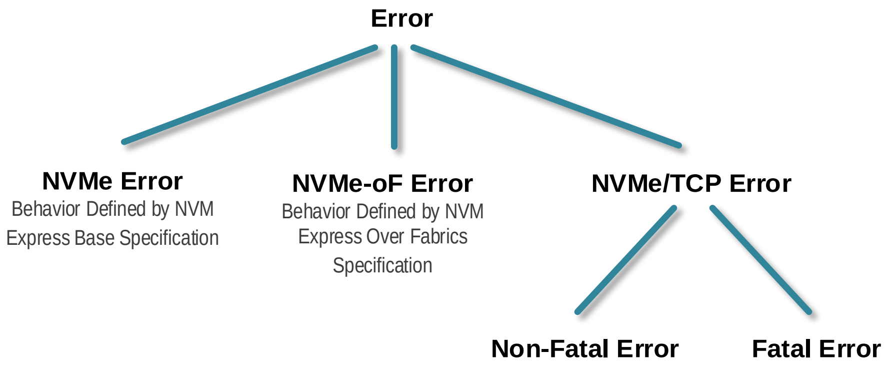
Header and Data Digest

- PDU Data integrity for both header and PDU Data
- Both Header and Data Digests are calculated using CRC32C (http://www.rfc-editor.org/rfc/rfc3385.txt)
- Generated by the sender and verified by the receiver
- Header Digest protects the PDU header it trails
  - Common Header (8 bytes)
  - Type-Specific Header (Variable Size)
- Data Digest protects the PDU Data payload it trails
  - Exists only for PDUs that contain Data payload



NVM EXPRESS®

NVMe™/TCP Errors



- **NVMe/TCP Non-Fatal Error**
  - An error that may affect one or more commands, but from which the transport is able to recover and continue normal operation
  - Commands affected by a non-fatal error are completed with a “Transient Transport Error” status code
- **NVMe/TCP Fatal Error**
  - An error from which the transport is not able to recover and continue normal operation
  - Fatal errors are handled by terminating the NVMe/TCP connection

Non-Fatal Errors

- Actions performed when a host or controller detects a non fatal error
  1. Continue processing associated PDUs that contain Data payload
     - C2HData PDUs that are associated with the same NVMe command
     - H2CData PDUs that are associated with the same R2T PDU
  2. Complete the command with a NVMe™ status code “Transient Transport Error”
  3. Host may retry the command

- Currently the Non-Fatal NVMe™/TCP error is a Data Digest verification error

Fatal Error Behavior

- Actions performed when a host or controller detects a fatal error
  1. Immediately cease processing any further incoming PDUs
  2. Send the appropriate TermReq PDU to the host/controller
  3. Wait a proper timeout for the host/controller to terminate the TCP connection
     - If the timeout expires, then the host/controller actively terminates the TCP connection

- A TermReq PDU contains:
  – Fatal Error Status (FES): Status code indicating the reason for the failure
  – Fatal Error Information (FEI): Specific status information
  – Data Payload: Dump of the faulty PDU Header

Fatal Error Classification

Error Classification	Error Group	Error Description
Corrupted Header	Invalid PDU Header Field	An invalid field detected in the PDU header
	Header Digest Error	Header digest verification failed
Connection Establishment	Unsupported Parameter	Unsupported parameter caused connection to terminate
State Machine / Sequence	PDU Sequence Error	A wrong sequence detected in the command execution
Data Transfer	Data Transfer Out of Range	A PDU with a data payload exceeds the permitted range
	R2T Limit Exceeded	A controller sent more inflight R2T PDUs than the host supports per command.

Transport Layer Security

- An NVM Subsystem can optionally support TLS secured NVMe™/TCP connections
  - TLS version >= 1.2 is supported
  - Discovery
    - TLS capable NVM subsystem discovery is done using the TSAS field in the discovery log page.

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td>Security Type (SECTYPE): Specifies the type of security used by the NVMe/TCP port. If SECTYPE is a value of zero (No Security), then the host is shall setup a normal TCP connection.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>No Security</td></tr>
<tr><td>01</td><td>Transport Layer Security (TLS) version 1.2 (refer to RFC 5246) or a subsequent version. The TLS protocol negotiates the version and cipher suite for each TCP connection.</td></tr>
<tr><td>255:02</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>255:01</td><td>Reserved</td></tr>
</table>

Transport Layer Security

- The TLS handshake psk_identity sent by the host is a combination of the host and NVM subsystem NVMe™ Qualified Names (NQNs) separated by a space (‘ ‘=U+0020) character

  nqn.2014-08.org.nvmexpress::nvm-subsystem-sn-d78432 nqn.2014-08.org.nvmexpress:uuid:36ebf5a9-1df9-47b3-a6d0-e9ba32e428a2

- Mandatory cipher suites
  - TLS_PSK_WITH_AES_128_GCM_SHA256 {00h, A8h}
  - TLS_PSK_WITH_AES_256_GCM_SHA384 {00h, A9h}
  - TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 {00h, AAh}
  - TLS_DHE_PSK_WITH_AES_256_GCM_SHA384 {00h, ABh}

Ecosystem

- Linux kernel support is upstream since v5.0 (both host and NVM subsystem)
  - https://lwn.net/Articles/772556/
  - https://patchwork.kernel.org/patch/10729733/
- SPDK support (both host and NVM subsystem)
  - https://github.com/spdk/spdk/releases
  - https://spdk.io/news/2018/11/15/nvme_tcp/
- NVMe™ compliance program
  - Interoperability testing started at UNH-IOL in the Fall of 2018
  - Formal NVMe compliance testing at UNH-IOL planned to start in the Fall of 2019
- For more information see:
  - https://nvmexpress.org/welcome-nvme-tcp-to-the-nvme-of-family-of-transports/

Summary

- NVMe™/TCP is a new NVMe-oF™ transport
  - NVMe/TCP is specified by TP 8000 (available at www.nvmexpress.org)
  - Since TP 8000 is ratified, NVMe/TCP is officially part of NVMe-oF 1.0 and will be documented as part of the next NVMe-oF specification release
- NVMe/TCP offers a number of benefits
  - Works with any fabric that support TCP/IP
    - Does not require a “storage fabric” or any special hardware
  - Provides near direct attached NAND SSD performance
  - Scalable solution that works within a data center or across the world

nvm EXPRESS®

Architected for Performance