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)

Canonical Latency (Addition to DAS)

Latency [μs]

- RDMA
- TCP

read       write
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
- NVMe over Fabrics
  - Extends NVMe over a Fabric Capsules, Properties, and Discovery
  - NVMe-oF Transport Specific Features/Specialization
  - NVMe Transport Mapping
  - TCP
  - IP
- Fabric
  - Fabric Physical (e.g., Ethernet)
- NVMe/TCP Transport
NVMe™ Transport Taxonomy

NVMe Transports

- Memory
  - Commands/Responses & Data use Shared Memory
    - Example
      - PCI Express

- Message
  - Commands/Responses use Capsules
    - Data may use Capsules or Messages
    - Examples
      - Fibre Channel
        - TCP

- Message / Memory
  - Commands/Responses use Capsules
    - Data may use Capsules or Shared Memory
    - Examples
      - RDMA
        - (InfiniBand, RoCE, iWARP)
**NVMe™/TCP Capsules**

<table>
<thead>
<tr>
<th>Capsule Type</th>
<th>Capsule Size Without Data</th>
<th>Max Capsule Size With Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fabrics or Admin Command</td>
<td>64-bytes</td>
<td>8256 bytes</td>
</tr>
<tr>
<td>I/O Command</td>
<td>64 bytes</td>
<td>64 bytes to (IOCCSZ(^a) * 16) bytes</td>
</tr>
<tr>
<td>Fabrics or Admin Response</td>
<td>16 bytes</td>
<td>Not Applicable</td>
</tr>
<tr>
<td>I/O Response</td>
<td>16 bytes</td>
<td>Not Applicable</td>
</tr>
</tbody>
</table>

\(^a\) 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
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

- Header (HDR)
- Header Digest (HDGST)
- PDU Padding (PAD)
- PDU Data (DATA)
- Data Digest (DDGST)

Common Header (CH)

- PDU Specific Header (PSH)
- 8 bytes
- variable bytes

- 4 bytes
- variable bytes
- variable bytes
- 4 bytes
Mapping of NVMe™/TCP PDUs to TCP
# NVMe™/TCP Protocol Data Units (PDUs)

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

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

<table>
<thead>
<tr>
<th>Error Classification</th>
<th>Error Group</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corrupted Header</td>
<td>Invalid PDU Header Field</td>
<td>An invalid field detected in the PDU header</td>
</tr>
<tr>
<td></td>
<td>Header Digest Error</td>
<td>Header digest verification failed</td>
</tr>
<tr>
<td>Connection Establishment</td>
<td>Unsupported Parameter</td>
<td>Unsupported parameter caused connection to terminate</td>
</tr>
<tr>
<td>State Machine / Sequence</td>
<td>PDU Sequence Error</td>
<td>A wrong sequence detected in the command execution</td>
</tr>
<tr>
<td>Data Transfer</td>
<td>Data Transfer Out of Range</td>
<td>A PDU with a data payload exceeds the permitted range</td>
</tr>
<tr>
<td></td>
<td>R2T Limit Exceeded</td>
<td>A controller sent more inflight R2T PDUs than the host supports per command.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td><strong>Security Type (SECTYPE):</strong> 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.</td>
</tr>
<tr>
<td></td>
<td><strong>Value</strong></td>
</tr>
<tr>
<td></td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>01</td>
</tr>
<tr>
<td>255:01</td>
<td>Reserved</td>
</tr>
<tr>
<td>255:02</td>
<td>Reserved</td>
</tr>
</tbody>
</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

- 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:
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