LEGAL NOTICE:

LEGAL DISCLAIMER:

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "**AS IS**" BASIS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, NVM EXPRESS, INC. (ALONG WITH THE CONTRIBUTORS TO THIS DOCUMENT) HEREBY DISCLAIM ALL REPRESENTATIONS, WARRANTIES AND/OR COVENANTS, EITHER EXPRESS OR IMPLIED, STATUTORY OR AT COMMON LAW, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, VALIDITY, AND/OR NONINFRINGEMENT.

All product names, trademarks, registered trademarks, and/or servicemarks may be claimed as the property of their respective owners.

NVM Express Workgroup
c/o VTM Group.
3855 SW 153rd Drive
Beaverton, OR 97003 USA
info@nvmexpress.org

NVM Express Technical Proposal for New Feature

Technical Proposal ID	**8000 – NVMe/TCP Transport**
Change Date	**11/13/2018**
Builds on Specification	**NVM Express Over Fabrics, Revision1.0**
References to Ratified Technical Proposals	**TP 4028 Transport/Pathing Related Error Codes**

Technical Proposal Author(s)

Name	**Company**
Sagi Grimberg, Muli Ben-Yehuda	Lightbits Labs
David Minturn, Anthony Knapp	Intel
David Black	Dell EMC
Christoph Hellwig	WD
Chris Petersen, Wei Zhang, Kumar Sundararajan	Facebook
Murali Rajagopal	VmWare
Peter Onufryk	Microsemi
Mike Thompson	Kazan Networks
Dean Scoville, Kamaljit Singh, Amit Radzi, Shai Malin	Marvell
Dinan Gunawardena	SolarFlare
Fredrick Knight	NetApp

This technical proposal adds a new TCP based transport binding definition to NVMe-oF

Revision History

Revision Date	Change Description
2017/07/19	Initial version
2017/07/20	Added Message Formats
2017/07/26	Incorporated Comments, Changed messages to PDU
2017/08/01	Updates based on initial review feedback
2017/08/01	Removed references to Data-In and Data-Out PDUs and replaced them with a single Data-Transfer PDU with Data Direction Bits matching NVMe Data Transfer Bits. Added Data Direction specific padding description.
2017/08/01	Removed last traces of “Message” and “NTCP”
2017/08/14	Replaced single Data Transfer PDU with separate. Added Digest picture. Misc. wording fixes
2017/08/15	Added FLAGS.LAST bit to Data Transfer PDUs.

2017/08/22	Incorporated TLS section
2017/08/28	Added rules to data transfer section
2017/08/29	Incorporated review comments
2017/08/30	Added transport specific errors and completion optimization
2017/08/30	Draft version for initial TWG review
2017/09/20	Add R2T tag to R2T and H2C Data PDUs
2017/09/20	Split DIGEST negotiation flags to connection init request/response to bodies, modified PDU flags to contain DIGEST information.
2017/09/20	Add Maximum host and controller Data offset size restriction.
2017/10/18	Some error status values rework
2017/11/26	Address comments from Mike Thompson Fix a few comments from Sagi in the document Add some more info on Digest negotiation procedure
2017/11/27	Added Figure tbd numbers.
2017/12/6	- Incorporated comments from James Harris and Daniel - Added recommendation on TCP level Keep Alive settings - Added Terminate connection and Terminate connection acknowledge PDUs - Removed ICDDGST - Some more fixups - Updated figure “PDU and Data Ranges”: Changed RTR to R2T - Changed Data offset (command capsule and PDU) upper bound to 127
2018/1/10	- Added TP Dependencies (Editor’s note) - Added Text from PeterO - Reduced Initialize connection request/response to 128 bytes - Mentioned that PDU digest is the 16 LSBits of the CRC32C - Rename Terminate Connection acknowledge to Terminate Connection Response - Clarify that Terminate Connection request is only sent in Connection initialization failure and not normal Connection termination sequence
2018/1/21	- Fixed TP number and Change Date - Enhanced PDU types Table - Enhanced NVMe/TCP PDUs vs. TCP/IP packets Figures - Added editor’s note of dependencies - Sorted out consistent Figure/Table enumeration - Fixed a few section references - Enhanced Table tbd6 (DaveM)
2018/2/14	- Rework Capsules and SGLs sections - Renamed HDOFF/CDOFF to C2HDOFF/H2CDOFF - Renamed RECFMT to PVF (PDU version format) - Replaced NVMe/TCP PDU vs. TCP/IP packets (Kamaljit) - Reworked table 6 - Added explanatory text on SUCCESS flag - Some clarifications on MAXR2T and MAXDADA
2018/2/22	- Updated some figures - Minor typo fixes

2018/2/26	- Added CPH and TSPH descriptions to Transport Overview section and added figure tbd1a. Renamed figure tbd1 to tbd1b. - Replaced figure tbd10 with figures tbd10a, 10b, 10c and 10d and added CPH and TSPH terminology to accompanying text. - Global replacement of PDU digest and PDGST with Header digest and HDGST, respectively. - Global replacement of PDU-HDR with CPH, added TSPH to section describing NVMe/TCP PDUs. - Fixed Controller to Host Data Transfer PDU section heading to say C2HData. - Expanded description of PDU type field to include a description of its components, and its equivalence to the opcode. <Added question about terminology> - Expanded descriptions of C2HDOFF and H2CDOFF fields in Connection Request/Response PDUs. <added questions about field size and padding granularity> - Modified C2H and H2C data transfer examples to include hex values in parentheses alongside the decimal values, to make examples easier to follow; changed DATA OFFSET to DATAO and DATA LEN to DATAL for consistency. (changes by Dean and Kamaljit)
2018/3/15	- Changed CPH to CH and TSPH to PSH - Updated Digest figures - Updated wording in Capsules and Data Transfers - Added PDU header digest generation and verification guidelines (section 7.4.7.1)
2018/3/21	- Added NVMe/TCP definitions - Added paragraph on PDU header digest generation and verification
2018/3/25	- Fixed DATAL,DATAO to R2TL,R2TO in host to controller data transfer - Added PDO (PDU Data Offset) to C2Hdata PDU (like H2Cdata PDU) - Added figures to Data Transfer Sections (Figures tbd5, tbd6)
2018/3/26	- Adjusted opcodes to the correct directions - Updated transport errors reference in table tbd11
2018/4/23	- Changed layout of PDU Common Header - Changed the basic definition of an NVMe/TCP PDU - Made Header digest trail the PDU header (CH+PSH) - Reworked all the PDU layouts (7.4.9) - Reworked the PDU header and data digest section - Removed section on verification and generation of HDGST - Restrict ICDOFF cleared to zero for NVMe/TCP - Changed H2CDOFF/C2HDOFF parameters to Host and Controller PDU Data Alignment (CPDA, HPDA). This now applies to command capsules as well. - Added PAD, PDATA definitions in definitions table - Removed PDU Data Alignment section (now part of H2C/C2H Data transfers) - Minor reword in transport dependent errors section - Phrasing fixes throughout - fixed headings and figures

2018/4/24	- Fixed various comments from Mike Thompson - Moved PAD description to section 7.4.5 and made it generic. - Updated data transfer figures to correct header and opcodes - Updated references throughout
2018/4/25	- Removed summary of data transfer section
2018/4/26	- Added PDU sequence transport specific errors
2018/5/6	- Added ports text from David - Enhanced figure tbd17 Digest error handling
2018/17/5	- Rework transport errors section - Rework Digest error handling section - Match R2TO,R2TL to DATAO,DATAL PDU locations - Made all PDU related length dword aligned for efficient digest calculation - Updated some Figure captions and references
2018/6/5	- Various cleanups (editorial) - Reworked TLS - Reworked Error handling (still WIP)
2018/6/11	- Refined section on Error handling (7.4.7) - Removed TermResp PDU as it can't really work when the connection is out of sync - Reworked language in error cases in sections: Connection Establishment, Data Transfers and PDU digest to refer to section on error handling - Moved R2T PDU format to be the last (for consistency with PDU types table) - Split TermReq PDU to directions (H2CtermReq and C2HtermReq) - Updated TermReq format (field names) - Updated PDU Types table references and opcodes - Updated NVMe/TCP definitions table (got rid of old definitions)
2018/6/13	- Fixed H2C/C2H TermReq PDU formats (maid PDO reserved, and fixed FES field) - Small rewords in Connection Initialization and Data Transfers error handling paragraphs - Fixed figure number in TLS section
2018/6/14	- Added few other transport fatal error statuses and modified the error cases paragraphs - Reword digests section according to feedback from Dean - Bound maximum H2C/C2H TermReq PDU length to protect against corruption in the PDU header (PLEN was corrupted)
2018/7/11	- Phrasing fixes in sections 7.4-7.4.4 (page turner) - Modified figures tbd4, tbd5 - Alphabet order to NVMe/TCP definition table - Modified PDU types table tbd6 format - Added IANA port assignment reference
2018/8/21	- Page Turners fixes (mostly phrasing) - Adding Cavium to the TP authors list
2018/8/29	- Fix Figure tbd3 footnote
2018/8/31	- Added figure for the PDU header structure - Modified figures numbering
2018/9/5	- Modified fatal transport error handling section - Added figure tbd4 on PDU header structure - Modified figure captions

2018/9/12	- Added Conventions section – on Endianess - Fixed Figure tbd10 (offsets in hex) - Fixed wording on transport error handling (section 7.4.7) - Fixed various PDU format descriptions (7.4.10.X) - Augmented TermReq EFI to contain the corrupted header digest for header digest errors. - Fixed bunch of spelling/phrasing comments from Harvey, Peter and others.
2018/10/23	- Minor typo in section 7.4.7 - Changed Figure tbd14 (typo C2Hdata vs. H2Cdata) - Few other phrasing fixes throughout - Changed PSK identity values in TLS negotiation ServerKeyExchange and ClientKeyExchange messages: The host does not need the pdk_identity_hint to learn the subsystem NQN it wants to connect to. Moreover, its not clear what subsystem NQN the TLS server should send when multiple subsystems are implemented. Hence, have the psk_identity in the CliendKeyExchange message contain both the host and the subsystem NQNs separated by the space character
2018/10/30	- Integration
2018/10/31	- Added reference to TP 4018 - Fixed history keep with next
2018/11/1	- Fixed Figure links
2018/11/13	- Ratified

Introduction

NVM Express (NVMe) 1.2.1 and prior revisions define a register level interface for host software to communicate with a non-volatile memory subsystem over PCI Express (NVMe over PCIe). This specification defines extensions to NVMe that enable operation over other interconnects (NVMe over Fabrics). The NVM Express revision 1.2.1 specification is referred to as the NVMe Base specification.

The mapping of extensions defined in this document to a specific NVMe Transport are defined in an NVMe Transport binding specification. This document contains an NVMe Transport binding specification for RDMA and TCP. The NVMe Transport binding specification for Fibre Channel is defined in INCITS 540 Fibre Channel – Non-Volatile Memory Express (FC-NVMe), refer to http://www.incits.org.

Modify Figure 34 as shown below:

Figure 34: Get Log Page – Discovery Log Page Entry

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td>Transport Type (TRTYPE): Specifies the NVMe Transport type.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>RDMA Transport (refer to section 7.3)</td></tr>
<tr><td>02</td><td>Fibre Channel Transport (refer to INCITS 540)</td></tr>
<tr><td>03</td><td>TCP Transport (refer to section 7.4)</td></tr>
<tr><td>~~03~~04-253</td><td>Reserved</td></tr>
<tr><td>254</td><td>Intra-host Transport (i.e., loopback) (NOTE: This is a reserved value for use by host software.)</td></tr>
<tr><td>255</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>01</td><td>Address Family (ADRFAM): Specifies the address family.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>AF_INET: IP~~V~~v4 address family. IPv4address ~~Address~~format syntax specified in section 3.2.2 of IETF RFC 3986~~791~~.</td></tr>
<tr><td>02</td><td>AF_INET6: IP~~V~~v6 address family. IPv6address ~~Address~~format syntax specified in section 3.2.2 of IETF RFC 3986~~2373~~.</td></tr>
<tr><td>03</td><td>AF_IB: InfiniBand address family.</td></tr>
<tr><td>04</td><td>Fibre Channel address family.</td></tr>
<tr><td>05 – 253</td><td>Reserved</td></tr>
<tr><td>254</td><td>Intra-host Transport (i.e., loopback) (NOTE: This is a reserved value for use by host software.)</td></tr>
<tr><td>255</td><td>Reserved</td></tr>
</table></td></tr>
</table>

Modify Section 7.3.6.1 as shown below:

7.3.6.1 Transport Specific Address Subtype and Transport Service Identifier

The Discovery Log Entry includes a Transport Specific Address Subtype (TSAS) field that is defined in Figure 42 for the RDMA Transport.

Figure 42 : Transport Specific Address Subtype Definition for RDMA Transport

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td>RDMA QP Service Type: (RDMA_QPTYPE): Specifies the type of RDMA Queue Pair. Valid values are shown in the following table:
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>Reliable Connected</td></tr>
<tr><td>02</td><td>Reliable Datagram</td></tr>
<tr><td>255:03</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>01</td><td>RDMA Provider Type: (RDMA_PRTYPE): Specifies the type of RDMA provider. Valid values are shown in the following table:
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>No provider specified</td></tr>
<tr><td>02</td><td>InfiniBand</td></tr>
<tr><td>03</td><td>~~InfiniBand~~RoCE (v1)</td></tr>
<tr><td>04</td><td>~~InfiniBand~~RoCE (v2~~V2)~~</td></tr>
<tr><td>05</td><td>iWARP</td></tr>
<tr><td>255:06</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>02</td><td>RDMA Connection Management Service: (RDMA_CMS): Specifies the type of RDMA Connection Management Service. Valid values are shown in the following table:
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>RDMA_IP_CM. Sockets based endpoint addressing. For details on the RDMA IP CM Service, refer to the InfiniBand Trade Association specification Annex A11 or the iWARP specification.</td></tr>
<tr><td>255:02</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>07:03</td><td>Reserved</td></tr>
<tr><td>09:08</td><td>RDMA_PKEY: Specifies the Partition Key when AF_IB (InfiniBand) address family type is used. Otherwise this field is reserved.</td></tr>
<tr><td>255:10</td><td>Reserved</td></tr>
</table>

The contents of the Transport Service Identifier (TRSVCID) field in a Discovery Log Entry for the NVMe/RDMA transport depends on the RDMA Connection Management Service (RDMA_CMS) that is used to establish NVMe/RDMA host-controller communication. The RDMA Internet Protocol Connection Management (RDMA IP CM) service shall be used with the NVMe/RDMA transport.

The following requirements apply to NVMe/RDMA use of the RDMA IP CM service:

- The TRSVCID field in a Discovery Log Entry for NVMe/RDMA shall contain a TCP port number represented in decimal as an ASCII string;
- If the TRSVCID field in a Discovery Log Entry for NVMe/RDMA does not contain a TCP port number represented in decimal as an ASCII string, then the host shall not use the information in that Discovery Log Entry to connect to a controller; and
- Hosts and NVM subsystems that support NVMe/RDMA are required to have IP addresses.

These three requirements apply to all RDMA Provider Types.

The RDMA IP CM service uses the TCP port number indicated by the TRSVCID field as part of establishing NVMe/RDMA host-controller communication. That TCP port number is not used as a component of RDMA protocol endpoint addresses for host-controller communication. RDMA protocol endpoint address establishment and contents are outside the scope of this document.

UDP port 4420 and TCP port 4420 have been assigned by IANA (refer to https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml) for use by NVMe over Fabrics. NVMe/RoCEv2 controllers use UDP port 4420 by default. NVMe/iWARP controllers use TCP port 4420 by default.

Add a new Transport Binding and Data Binding chapter to section 7 of NVMe-oF V1.x specification that defines a TCP based transport

7.4 Transport Capsule and Data Binding: TCP

This section defines the binding of an NVMe implementation that uses the Transport Type of TCP Transport [RFC 793] as defined by Figure 34. Common definitions used for TCP are defined in Figure tbd1.

Figure tbd43: TCP Definitions

Term	Definition
TCP	Transmission Communication Protocol
TCP Segment	TCP accepts data in the form of a data stream and breaks the stream into units. A TCP header is added to a unit creating a TCP segment.
TCP/IP Packet	A TCP segment encapsulated in an Internet Protocol (IP) datagram creating a TCP/IP packet.
Established TCP Connection	Two TCP endpoints that have an established full-duplex communication channel between them and ready for data transfer.

Figure tbd44: NVMe/TCP Definitions

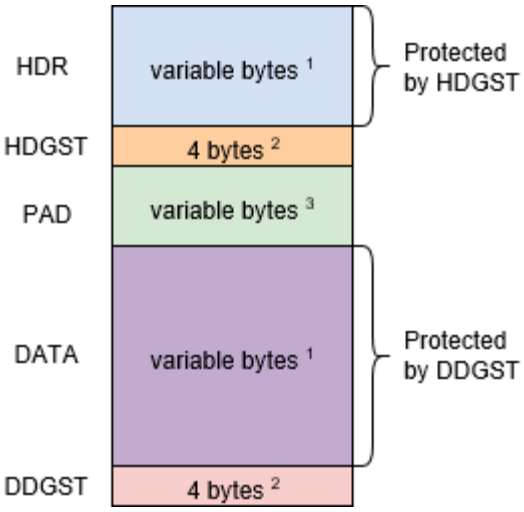
Term	Definition
CH	PDU Common Header
CPDA	Controller PDU Data Alignment
C2H	Controller to Host Direction
C2HTermReq	Controller to Host Terminate Connection Request
DATA	PDU Data
DATAL	H2CData and C2HData PDU Data Length
DATAO	H2CData and C2HData PDU Data Offset
DDGST	PDU Data Digest
HDGST	PDU Header Digest
HDR	PDU Header
HPDA	Host PDU Data Alignment
H2C	Host to Controller Direction
H2CTermReq	Host to Controller Terminate Connection Request
ICReq	Initialize Connection Request
ICResp	Initialize Connection Response
PAD	PDU padding bytes (before DATA starts)
PDU	Protocol Data Unit
PFV	Protocol Format Version
PSH	PDU Specific Header
R2T	Ready to Transfer (PDU)
R2TO	Ready to Transfer PDU Data Offset
R2TL	Ready to Transfer PDU Data Length
TTAG	Transfer Tag

7.4.1 Transport Overview

The TCP transport provides a reliable in-order capsule and data delivery service between a host and an NVM subsystem. While this transport binding is defined in a manner that allows efficient software-only implementations utilizing existing TCP network transport software application interfaces, this binding specification does not preclude hardware-only or hardware-accelerated implementations.

A host and a controller in an NVM subsystem communicate over TCP by exchanging NVMe/TCP Protocol Data Units (NVMe/TCP PDUs). An NVMe/TCP PDU may be used to transfer a capsule, data, or control/status information. As shown in Figure tbd3, an NVMe/TCP PDU consists of five parts. The PDU Header (HDR) is required in all PDUs. The PDU Header Digest (HDGST), the PDU Padding field (PAD), the PDU Data field (DATA), and the Data Digest (DDGST) field are included or omitted based on the PDU type and the values exchanged during connection establishment (refer to section 7.4.10).

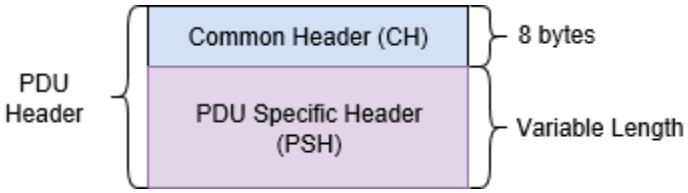
Figure tbd45: NVMe/TCP PDU Structure



1) Length is PDU dependent.
2) Digests (Header and Data) are included only after being enabled during connection establishment.
3) PAD bytes are included only after being communicated during connection establishment.

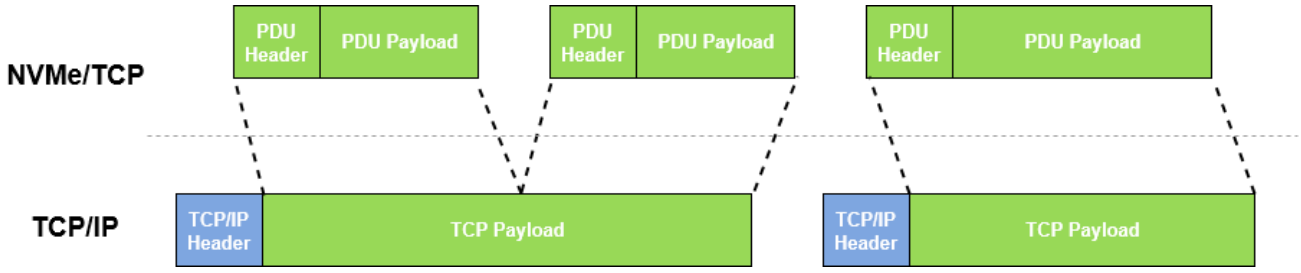
The PDU header (HDR) consists of a PDU common header (CH) which has a fixed length of 8 bytes and a PDU specific header (PSH) which has a variable length (refer to section 7.4.10.1).

Figure tbd46: NVMe/TCP PDU Header



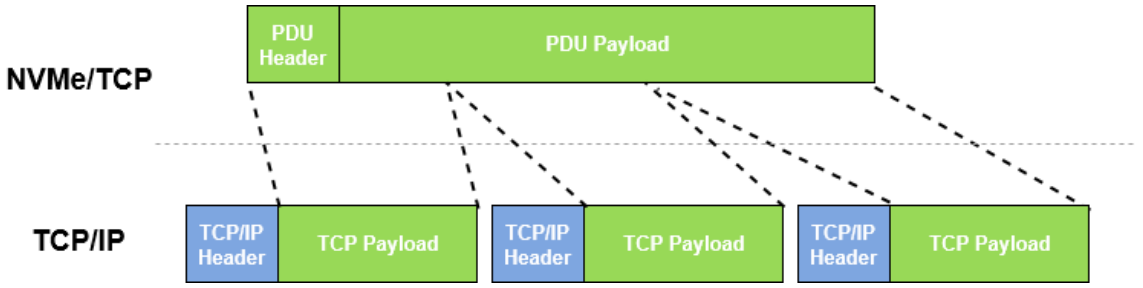
As shown in Figure tbd5 and Figure tbd6, there is no requirement to align NVMe/TCP PDU headers nor PDU payloads [1] to TCP/IP packet boundaries.

Figure tbd47: Multiple NVMe/TCP PDUs in a single TCP/IP packet



[1] PDU payload refers to all the PDU contents other than the PDU header.

Figure tbd48: NVMe/TCP PDU spanning across TCP/IP packets



The TCP transport defines NVMe/TCP PDU types that are summarized in Figure tbd7 and defined in section 7.4.10. Associated with each NVMe/TCP PDU type is a direction that specifies whether the NVMe/TCP PDU is transferred from a host to a controller (H2C) or from a controller to a host (C2H). An NVMe/TCP PDU that is transferred in a direction opposite from that with which it is associated is treated as a fatal transport error (refer to section 7.4.7).

Figure tbd49: NVMe/TCP PDU Types

PDU Name	Opcode by field		Combined Opcode [2]	Section	PDU Description
	Function (07:01)	PDU Direction[1] (00)			
ICReq	0000000b	0b	00h	7.4.10.2	**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	0000000b	1b	01h	7.4.10.3	**Initialize Connection Response:** A PDU sent from a controller to a host to accept a connection request and communicate NVMe/TCP connection parameters
H2CTermReq	0000001b	0b	02h	7.4.10.4	**Host to Controller Terminate Connection Request:** A PDU sent from a host to a controller in response to a fatal transport error
C2HTermReq	0000001b	1b	03h	7.4.10.5	**Controller to Host Terminate Connection Request:** A PDU sent from a controller to a host in response to a fatal transport error
CapsuleCmd	0000010b	0b	04h	7.4.10.6	**Command Capsule:** A PDU sent from a host to a controller to transfer an NVMe over fabrics command capsule
CapsuleResp	0000010b	1b	05h	7.4.10.7	**Response Capsule:** A PDU sent from a controller to a host to transfer an NVMe over fabrics response capsule
H2CData	0000011b	0b	06h	7.4.10.8	**Host to Controller Data:** A PDU sent from a host to a controller to transfer data to the controller

C2HData	0000011b	1b	07h	7.4.10.9	**Controller to Host Data:** A PDU sent from a controller to a host to transfer data to the host
R2T	0000100b	1b	09h	7.4.10.10	**Ready to Transfer:** A PDU sent from a controller to a host to indicate that it is ready to accept data

NOTES:

1. Indicates the opcode encoded direction of the PDU. All PDUs shall follow this convention:
   a. 0b = Host to Controller (H2C); and
   b. 1b = Controller to Host (C2H).
2. Opcodes not listed are reserved.

7.4.1.1 Conventions

NVMe/TCP definition conforms to the byte, word, and dword relationships defined in section 1.8 of the NVMe base specification. This includes specifying all PDU contents in little endian format unless otherwise noted. PDU bytes are transmitted and received in little endian byte order.

7.4.2 Queue Mapping

An NVMe/TCP connection is associated with a single Admin or I/O Submission Queue and Completion Queue pair. Multiplexing two or more Submission Queues or Completion Queues on a single NVMe/TCP connection is not supported. Spanning a single Submission Queue or Completion Queue across two or more NVMe/TCP connections is also not supported.

7.4.3 Capsules

The NVMe/TCP transport supports a message model. Data transfers are supported via a transport specific data transfer mechanism, described in section 7.4.5, and optionally via in-capsule data. NVMe/TCP capsule sizes are summarized in Figure tbd8. The size of capsules is variable when in-capsule data is supported and fixed when in-capsule data is not supported.

The maximum amount of in-capsule data for Fabrics and Admin Commands is 8192 bytes, causing their maximum size to be 8256 bytes (i.e., 64 bytes + 8192 bytes). If an Admin or Fabrics command capsule requires more than 8192 bytes of data to be transferred, then the NVMe/TCP data transfer mechanism described in section 7.4.5.3 shall be used. In-capsule data is not supported for Fabrics and Admin Response capsules, causing their maximum size to be 16 bytes. Response data is transferred using the data transfer mechanism described in section 7.4.5.2.

The maximum I/O Queue Command capsule size is specified by the I/O Queue Command Capsule Supported Size (IOCCSZ) field in the Identify Controller data structure. If more data is required to be transferred than will fit in a command capsule, then the NVMe/TCP data transfer mechanism described in section 7.4.5.3 shall be used. The maximum I/O Queue Response Capsule Supported Size (IORCSZ) is 16 bytes and shall not contain in-capsule data. Response data is transferred using the data transfer mechanism described in section 7.4.5.2. The NVMe/TCP transport does not use ICDOFF to control the in-capsule data offset, thus ICDOFF shall be cleared to '0'. Data alignment is controlled by an NVMe/TCP specific mechanism (refer to section 7.4.5).

When command capsules contain in-capsule data, the capsule ends at the last byte of capsule data. NVMe/TCP capsules shall never contain any undefined data following in-capsule data as is allowed in NVMe over Fabrics in general.

Figure tbd50: NVMe/TCP Capsule Size

Capsule Type	In-Capsule Data Not Supported	In-Capsule Data Supported

Fabrics and Admin Commands	N/A [1]	64 bytes to 8256 bytes
Fabrics and Admin Responses	16 bytes	N/A [2]
I/O Queue Command	64 bytes	64 bytes to (IOCCSZ * 16) bytes
I/O Queue Response	16 bytes	N/A [2]

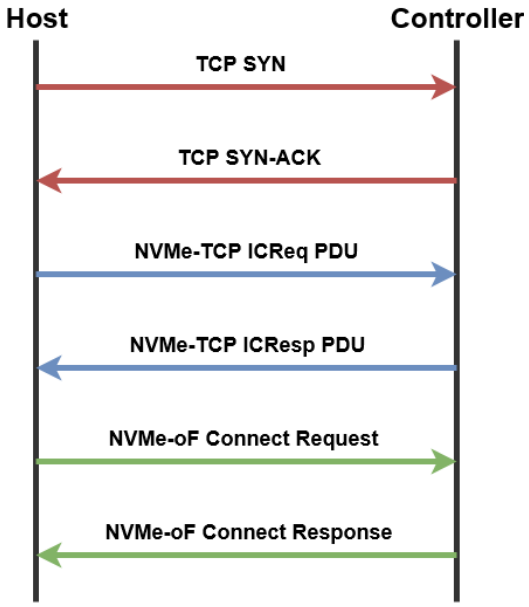
NOTES:
1. NVMe/TCP controllers must support in-capsule data for Fabrics and Admin command capsules.
2. In-capsule data is not supported in response capsules.

7.4.4 Connection Establishment

Figure tbd9 illustrates the process used to establish NVMe/TCP connection. The first step is to establish a TCP connection between a host and a controller. A controller acts as the passive side of the TCP connection and is set to "listen" for host-initiated TCP connection establishment requests.

Once a TCP connection has been established, the host sends an Initialize Connection Request (ICReq) PDU to the controller. When a controller receives an ICReq PDU, it responds with an Initialize Connection Response (ICResp) PDU. The exchange is used to both establish a connection and exchange connection configuration parameters. When a connection is established, the host and controller are ready to exchange capsules and command data. The first capsule exchange is the NVMe-oF Connect Request/Response sequence.

Figure tbd51: NVMe/TCP Queue Establishment Sequence



If a timeout occurs in the process of establishing a connection, then the host shall terminate the connection.

Reception of an ICReq PDU with an invalid field is treated as a fatal transport error with the Fatal Error Status field in the C2HTermReq PDU set to "Invalid PDU header field" and the Additional Error Information field containing the invalid PDU field byte offset (refer to section 7.4.7). Reception of an ICReq PDU with an unsupported parameter is treated as a fatal transport error with the Fatal Error Status field in the C2HTermReq PDU set to "Unsupported Parameter" and the Additional Error Information field containing the unsupported parameter field byte offset.

Reception of an ICResp PDU with an invalid field is treated as a fatal transport error with the Fatal Error Status field in the H2CTermReq PDU set to "Invalid PDU header field" and the Additional Error Information field containing the invalid PDU field byte offset. Reception of an ICResp PDU with an unsupported parameter

is treated as a fatal transport error with the Fatal Error Status field in the H2CTermReq PDU set to "Unsupported Parameter" and the Additional Error Information field containing the unsupported parameter field byte offset.

7.4.5 Data Transfers

All NVMe/TCP implementations shall support data transfers using command data buffers (described in section 7.4.5.1) and may optionally support in-capsule data.

Host and Controller PDU Data is optionally aligned. PDU data alignment is designed to allow the host or controller to guarantee data (and data digest) starting offset to be aligned to some value (usually a cache line). The alignment of data in a PDU is specified by the host and the controller when a connection is established. The Controller to Host PDU Data Alignment (HPDA) field in the ICReq PDU specifies the required alignment of PDU Data (DATA) from the start of the PDU for PDUs that are transferred from the controller to the host. The Host to Controller PDU Data Alignment (CPDA) field in the ICResp PDU specifies the required alignment of PDU Data (DATA) from the start of the PDU for PDUs that are transferred from the host to the controller. An appropriate number of padding bytes shall be inserted by the controller or host in the PAD field to achieve the required alignment. The number of PAD bytes is a function of the required alignment and the size of the PDU header. PDU PAD bytes are considered as reserved bytes and are not protected by HDGST nor by DDGST. The Host and Controller PDU Data Alignment field (HPDA, CPDA) shall not exceed 128 bytes.

Figure tbd10 shows an example of an H2CData PDU where CPDA (refer to section 7.4.10.3) was set to 0Fh by the host in ICResp when the connection was established. The H2CData PDU header size 24 bytes and header digest is disabled. An alignment of 64 bytes is required, thus the host inserts 40 bytes of padding in the PAD field.

Figure tbd52: Example of 64B PDU DATA Alignment in H2CData PDU

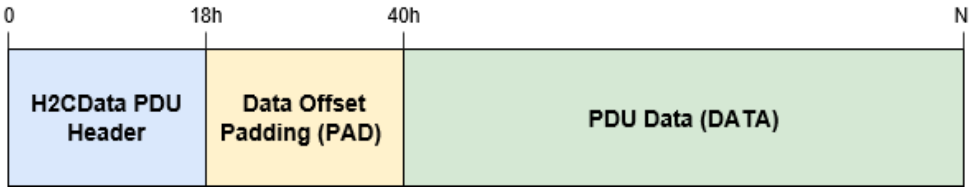
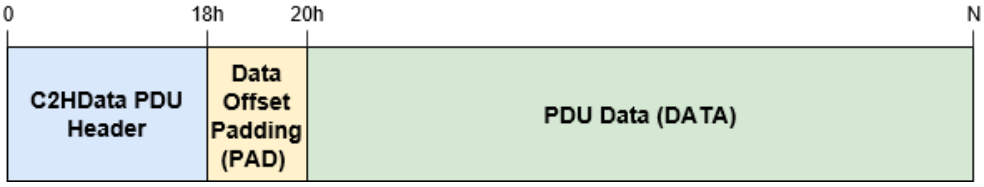


Figure tbd11 shows an example of a C2HData PDU where HPDA (refer to section 7.4.10.2) was set to 03h by the host in ICReq when the connection was established. The C2HData PDU header size 24 bytes and header digest is disabled. An alignment of 16 bytes is required, thus the controller inserts 8 bytes of padding in the PAD field.

Figure tbd53: Example of 16B PDU DATA Alignment in C2HData PDU



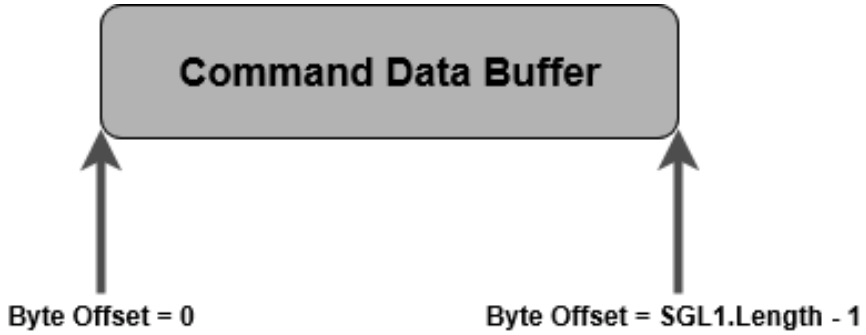
7.4.5.1 Command Data Buffers and SGLs

A Command Data Buffer is associated with commands that require a data transfer beyond what is allowed in a capsule. A Command Data Buffer is a per command buffer in host memory whose size corresponds to the amount of data required to be transferred outside a capsule. It is a logical concept whose actual implementation within a host is outside the scope of this specification. H2CData PDUs transfer data from a Command Data Buffer to a controller while C2HData PDUs transfer data from a controller to a Command Data Buffer.

The NVMe/TCP transport supports exactly one SGL descriptor per command. NVMe/TCP supports two SGL Descriptor types. An SGL Data Block descriptor with an SGL Descriptor Sub Type value of 1h specifies the use of in-capsule data. A Transport SGL Data block descriptor with an SGL Descriptor Sub Type value of Ah is referred to as Command Data Buffer Descriptor and specifies the use of the NVMe/TCP transport specific data transfer mechanism. The length field in the descriptor specifies the size of the Command Data Buffer associated with the command (refer to Figure tbd12).

Command Capsule PDU with an unsupported SGL descriptor type shall be completed by the controller with an "SGL DESCRIPTOR TYPE INVALID" error status set in the response capsule PDU.

Figure tbd54: Command Data Buffer Transport SGL Data Block Descriptor



7.4.5.2 Controller to Host Command Data Buffer Transfers

One or more C2HData PDUs are used to transfer data from a controller to a host. The Data Length (DATAL) field in the PDU specifies the amount of data that is transferred by the PDU while the Data Offset (DATAO) field in the PDU specifies the offset from the start of the Command Data Buffer where the transferred data should be placed. The first C2HData PDU of a command shall start with an offset of zero and subsequent C2HData PDUs for the command shall transfer data sequentially to the end of the data buffer (i.e., the DATAO field in the first C2HData PDU is cleared to 0h and the DATAO field in subsequent C2HData PDUs is equal to the DATAO field plus DATAL field of the previous C2HData PDU).

Reception of a non-contiguous C2HData PDU is treated as a fatal transport error with the Fatal Error Status field in the H2CTermReq PDU set to "PDU Sequence Error" (refer to section 7.4.7).

Reception of a C2HData PDU that is outside the command data buffer range is treated as a fatal transport error with the Fatal Error Status field in the H2CTermReq PDU set to "Data Transfer Out of Range".

Reception of a C2HData PDU with an unknown command capsule identifier (CCCID) is treated as a fatal transport error with the Fatal Error Status field in the H2CTermReq PDU set to "Invalid PDU header field" and the Additional Error Information field containing the CCCID field byte offset.

C2HData PDUs contain a LAST_PDU flag that is set to '1' in the last PDU of a command data transfer and is cleared to '0' in all other C2HData PDUs associated with the command. C2HData PDUs also contain a SUCCESS flag that may be set to '1' in the last C2HData PDU of a command data transfer to indicate that the command has completed successfully. In this case, no Response Capsule is sent by the controller for the command and the host synthesizes a completion queue entry for the command with the Command Specific field and the Status Field both cleared to 0h. If the SUCCESS flag is cleared to '0' in the last C2HData PDU of a command, then the controller shall send a Response Capsule for the command to the host. The SUCCESS flag shall be cleared to '0' in all C2HData PDUs that are not the last C2HData PDU for a command. The SUCCESS flag may be set to '1' in the last C2HData PDU only if the controller supports disabling submission queue head pointer updates.

Reception of a C2HData PDU with the SUCCESS flag set to '1' and the LAST_PDU flag cleared to '0' is treated as a fatal transport error with the Fatal Error Status field in the H2CTermReq PDU set to "Invalid PDU header field" and the Additional Error Information field containing the FLAGS field byte offset.

Reception of an unexpected C2HData PDU is treated as a fatal transport error with the Fatal Error Status field in the H2CTermReq PDU set to “PDU Sequence Error”.

Examples of an unexpected C2HData PDUs are:

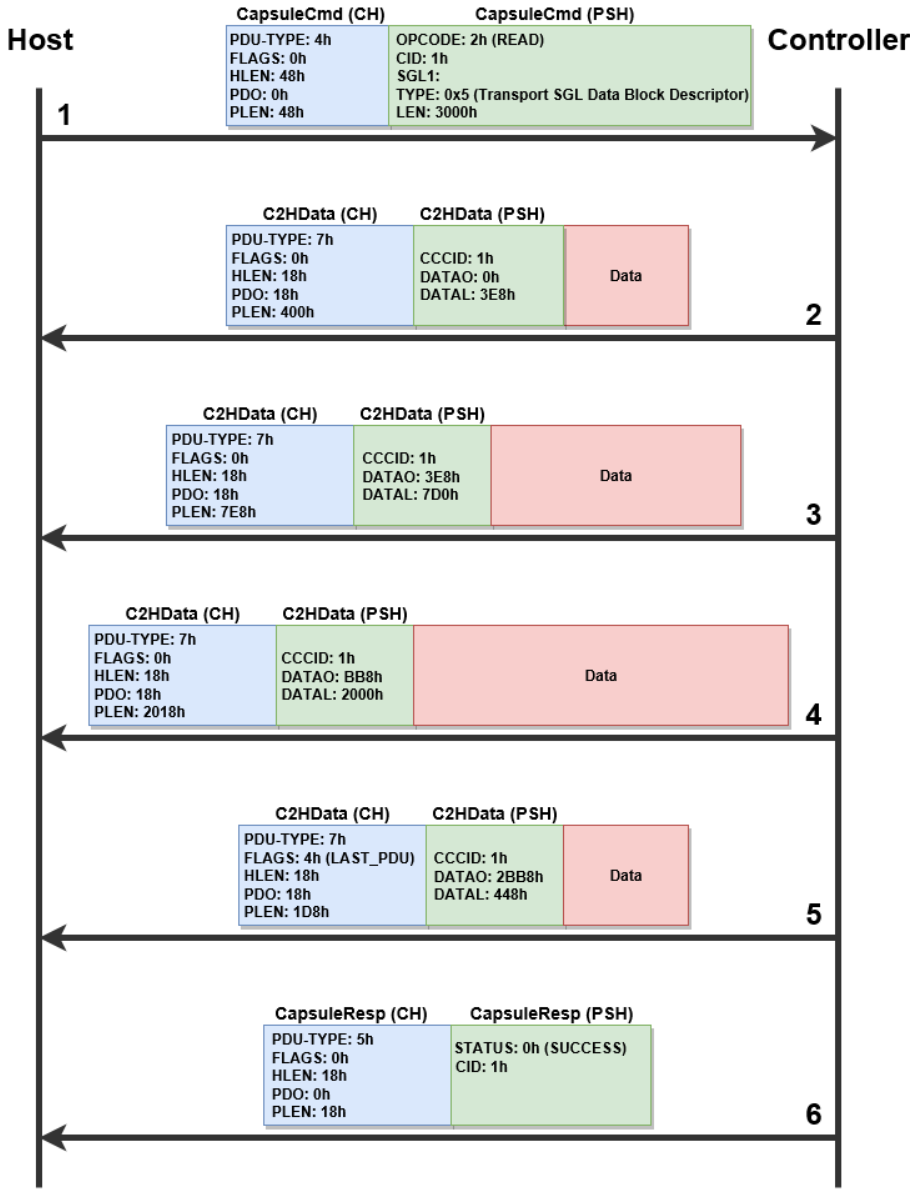
- Reception of a C2HData PDU after reception of a C2HData PDU with the LAST_PDU flag set to ‘1’ and is associated with the same command; and
- Reception of a C2HData PDU associated with a command that does not involve controller to host data transfer.

C2HData PDUs for a command shall be sent in-order for a command but may be arbitrarily interleaved with other unrelated PDUs (e.g., other C2HData PDUs transferring data for different commands).

Figure tbd13 illustrates a command that performs a 12KB (3000h byte) controller to host command data buffer transfer:

1. The host sends a CapsuleCommand PDU to the controller containing an SQE with a Transport SGL Data Block descriptor in SGL1. The Length field in the descriptor has a value of 3000h;
2. The controller processes the command and sends an C2HData PDU that transfers 1000B (3E8h bytes). The Data Offset (DATAO) field is cleared to 0h, the Data Length (DATAL) field is set to 3E8h, and the Last (LAST_PDU) flag is cleared to ‘0’ since this is not the last PDU of the data transfer;
3. The controller sends a subsequent C2HData PDU that transfers 2000B (7D0h bytes). The DATAO field is set to 3E8h, the DATAL field is set to 7D0h, and the LAST_PDU flag is cleared to ‘0’ since this is not the last PDU of the data transfer;
4. The controller sends a subsequent C2HData PDU that transfers 8192B (2000h bytes). The DATAO field is set to BB8h, the DATAL field is set to 2000h, and the LAST_PDU flag is cleared to ‘0’ since this is not the last PDU of the data transfer;
5. The controller sends a subsequent C2HData PDU that transfers 1096B (448h bytes). The DATAO field is set to 2BB8h, the DATAL field is set to 448h, and the LAST_PDU flag is set to ‘1’ since this is the last PDU of the data transfer; and
6. The controller sends a CapsuleResp PDU to the host containing a CQE.

Figure tbd55: Controller to Host Data Transfer Example



7.4.5.3 Host to Controller Command Data Buffer Transfers

Command data buffer transfers from a host to a controller parallel the behavior of command data buffer transfers from a controller to a host described in section 7.4.5.2. They are performed from a host to a controller using one or more H2CData PDUs. The data transferred by the H2CData PDUs starts at the beginning of the command data buffer and continues sequentially to the end of the command data buffer (i.e., the DATAO field in the first H2CData PDU is cleared to 0h and the DATAO field in a subsequent H2CData PDU is equal to the DATAO field plus the DATAL field of the previous H2CData PDU).

Reception of a non-contiguous H2CData PDU is treated as a fatal transport error with the Fatal Error Status field set to “PDU Sequence Error”.

The controller governs the rate of the data transfer from the host to the controller using Ready to Transfer (R2T) PDUs. When a connection is established, the host species the maximum number of outstanding R2T PDUs (MAXR2T) it supports at any point in time for a command. The first R2T PDU of a command shall start

with an offset of zero and subsequent R2T PDUs for the command shall solicit data transfers sequentially to the end of the command data buffer (i.e., the R2TO field in the first R2T PDU is cleared to 0h and the R2TO field in subsequent R2T PDUs shall be equal to the R2TO field plus R2TL field of the previous R2T PDU).

H2CData PDUs are sent in response to R2T PDUs and are never sent without receiving a corresponding R2T PDU. The R2T PDU specifies the command identifier with which it is associated, a transfer tag (TTAG), a data offset (R2TO) from the beginning of the command data buffer, and the transfer data length (R2TL). When an R2T PDU is received by a host, then the host transfers the data requested by the controller in the R2T PDU. This data transfer may be performed using one or more H2CData PDUs. H2CData PDUs must start at the offset specified in the R2T PDU (R2TO) and transfer data contiguously until the data transfer length specified by the R2T PDU (R2TL) is reached (i.e., the DATAO field in the first H2CData PDU is equal to R2TO that specified in the R2T PDU and DATAO field in subsequent PDUs is equal the DATAO field plus DATAL field of the previous H2CData PDU). The H2CData PDU length does not exceed the maximum length communicated by the controller during the connection establishment (MAXH2CDATA).

Reception of a non-contiguous R2T PDU is treated as a fatal transport error with the Fatal Error Status field in the C2HTermReq PDU set to “PDU Sequence Error”.

Reception of a H2CData PDU with data length which exceeds MAXH2CDATA is treated as a fatal transport error with the Fatal Error Status field in the C2HTermReq PDU set to “Data Transfer Limit Exceeded”.

Reception of a H2CData PDU that is outside the range starting from R2TO to R2TO + R2TL is treated as a fatal transport error with the Fatal Error Status field in the C2HTermReq PDU set to “Data Transfer Out of Range”.

The LAST_PDU flag in H2CData PDUs is cleared to ‘0’ in all but the last H2CData PDU that is associated with a single R2T PDU. The LAST_PDU flag is set to ‘1’ in the last H2CData PDU that satisfies a R2T request. All H2CData PDUs used to satisfy data requested by a controller shall have their Transfer Tag (TTAG) field set to the transfer tag specified in the R2T PDU.

Reception of a H2CData PDU with an unknown transfer tag (TTAG) is treated as a fatal transport error with the Fatal Error Status field set to “Invalid PDU header field” and the Additional Error Information field containing the TTAG field byte offset.

Reception of an unexpected H2CData PDU is treated as a fatal transport error with the Fatal Error Status field in the C2HTermReq PDU set to “PDU Sequence Error”.

Examples of an unexpected H2CData PDUs are:

- Reception of a H2CData PDU with the LAST_PDU flag cleared to ‘0’ after reception of a H2CData PDU with the LAST_PDU flag set to ‘1’ and is associated with the same R2T PDU.

R2T PDUs associated with a specific command shall be serviced in the order received by the host. R2T PDUs associated with different commands received by a host may be serviced in any order. A controller may send an R2T PDU without waiting for a previous R2T data transfer to complete, but shall not exceed the maximum number of outstanding R2T PDUs per command supported by the host (MAXR2T).

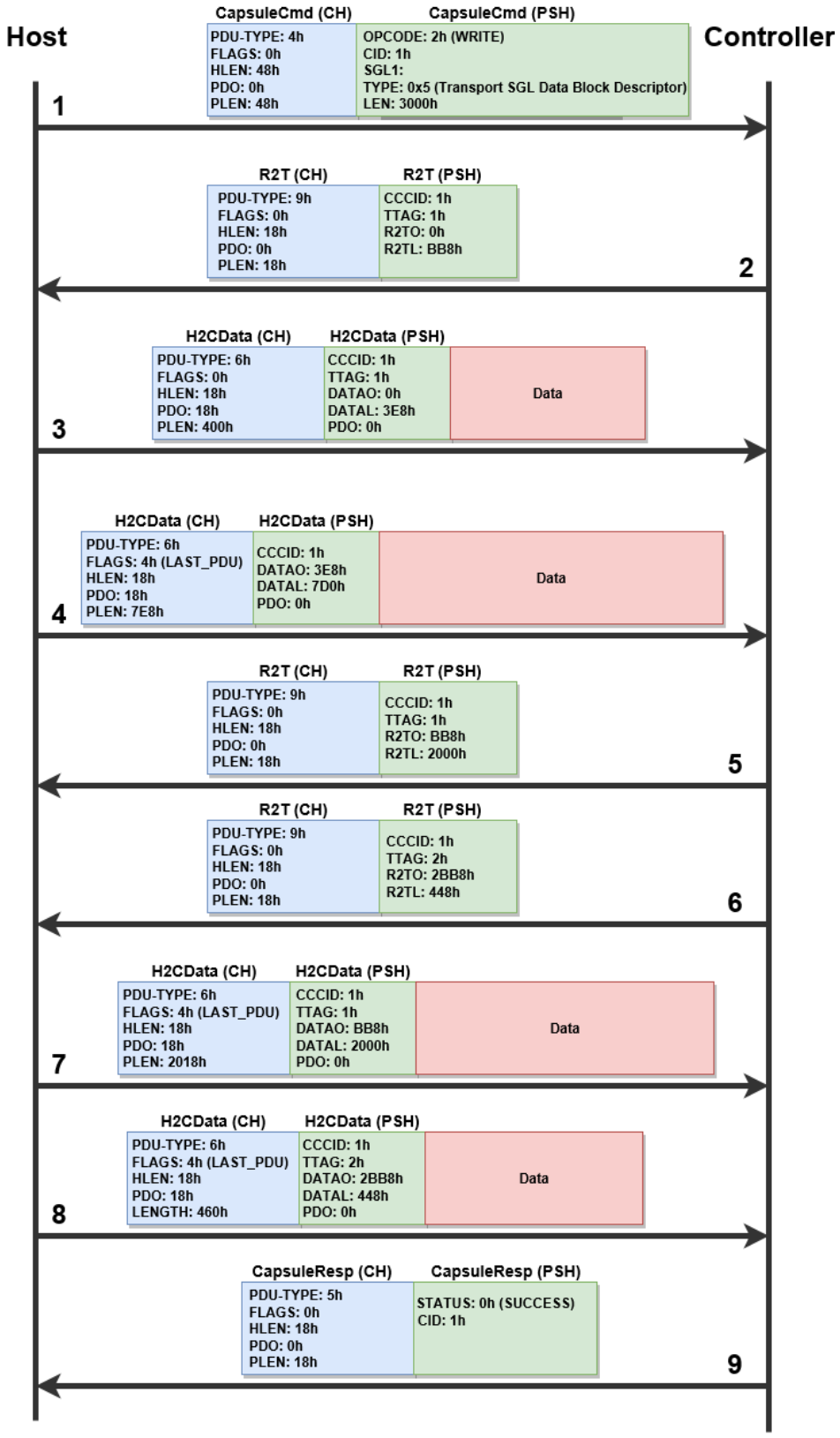
Reception of an R2T PDU that exceeds MAXR2T is treated as a fatal transport error with the Fatal Error Status field in the H2CTermReq PDU set to “R2T Limit Exceeded”.

Figure tbd14 illustrates a command that performs a 12KB (3000h byte) host to controller command data buffer transfer using R2T PDUs.

1. The host sends a Command PDU to the controller containing an SQE with a Transport SGL Data Block descriptor with subtype value of Ah in SGL1. The Length field in the descriptor has a value of 3000h;

2. The controller processes the command and sends an R2T PDU to the host that requests 3000B (BB8h bytes) of data with a data offset of zero;
3. When the host receives the R2T PDU, it sends an H2CData PDU that transfers 1000B (3E8h bytes). The R2T Data Offset (R2TO) field is cleared to 0h, the R2T Data Length (R2TL) field is set to 3E8h, and the Last (LAST_PDU) flag is cleared to '0' since this is not the last PDU of the data transfer;
4. The host sends a subsequent H2CData PDU that transfers 2000B (7D0h bytes). The DATAO field is set to 3E8h, the DATAL field is set to 7D0h, and the LAST_PDU flag is set to '1' since this is the last PDU of the data transfer requested by the R2T PDU;
5. The controller sends a subsequent R2T PDU that transfers 8192B (2000h bytes). The R2TO field is set to BB8h and the R2TL field is set to 2000h;
6. The controller sends a subsequent R2T PDU that transfers 1096B (448h bytes) if the host supports MAXR2T greater than one. The R2TO field is set to 2BB8h and the R2TL field is set to 448h;
7. The host sends a H2CData PDU that transfers 8192B (2000h bytes). The DATAO field is set to BB8h, the DATAL field is set to 2000h, and the LAST_PDU flag is set to '1' since this is the last PDU of the data transfer;
8. The host send a subsequent H2CData PDU that transfers 1096B (448h bytes). The DATAO field is set to 2BB8h, the DATAL field is set to 448h, and the LAST_PDU flag is set to '1' since this is the last PDU of the data transfer; and
9. The controller sends a Response PDU to the host containing a CQE.

Figure tbd56: Host to Controller Data Transfer Example



7.4.6 PDU Header and Data Digests

NVMe/TCP facilitates an optional PDU Header and Data digest. The presence of each digest is negotiated at the connection establishment.

The host requests the use of header digest by setting HDGST_ENABLE flag in the ICReq PDU. The controller may accept (or reject) the use of header digest by setting (or clearing) HDGST_ENABLE flag in the ICResp PDU. Header digest is enabled if HDGST_ENABLE flag is set in both the ICReq and ICResp PDUs. If PDU header digest is enabled, then all the subsequent PDUs transferred in this connection except H2CTermReq and C2HTermReq PDUs shall contain a HDGST field and have the HDGSTF flag set to '1' in the PDU header FLAGS field. If header digest is enabled, the header digest is contained within the HDGST field of the PDU and protects the PDU header. If PDU header digest is not enabled, then all subsequent PDUs shall not contain a HDGST field and have the HDGSTF flag, if defined, cleared to '0' in the PDU header FLAGS field.

The host request the use of data digest by setting DDGST_ENABLE flag in the ICReq PDU. The controller may accept (or reject) the use of data digest by setting (or clearing) DDGST_ENABLE flag in the ICResp PDU. Data digest is enabled if DDGST_ENABLE flag is set in both the ICReq and ICResp PDUs. If data digest is enabled, then all Command Capsule PDUs containing in-capsule data and all H2CData and C2HData PDUs transferred in this connection shall contain a DDGST field and have the DDGSTF flag set to '1' in the PDU header FLAGS field. If PDU data digest is not enabled, then these PDUs shall not contain a DDGST field and have the DDGSTF flag cleared to '0' in the PDU header FLAGS field. If data digest is enabled, the data digest is contained within the DDGST field of the PDU and protects the PDU data.

If a host requests the use of header or data digest in the ICReq PDU, but the use of the digest was not enabled by the controller in the ICResp PDU, then the host may refuse the connection establishment and terminate the NVMe/TCP connection (refer to section 7.4.4).

If a host did not request the use of header or data digest in the ICReq PDU but the use of the digest was enabled by the controller in the ICResp PDU, then the host shall treat it as a fatal transport error with the Fatal Error Status field set to "Invalid PDU header field" and the Additional Error Information field containing the DIGEST field byte offset.

Header and Data digests are calculated using the CRC32C algorithm (refer to http://www.rfc-editor.org/rfc/rfc3385.txt).

Figure tbd57: Host to Controller NVMe/TCP PDU Digests

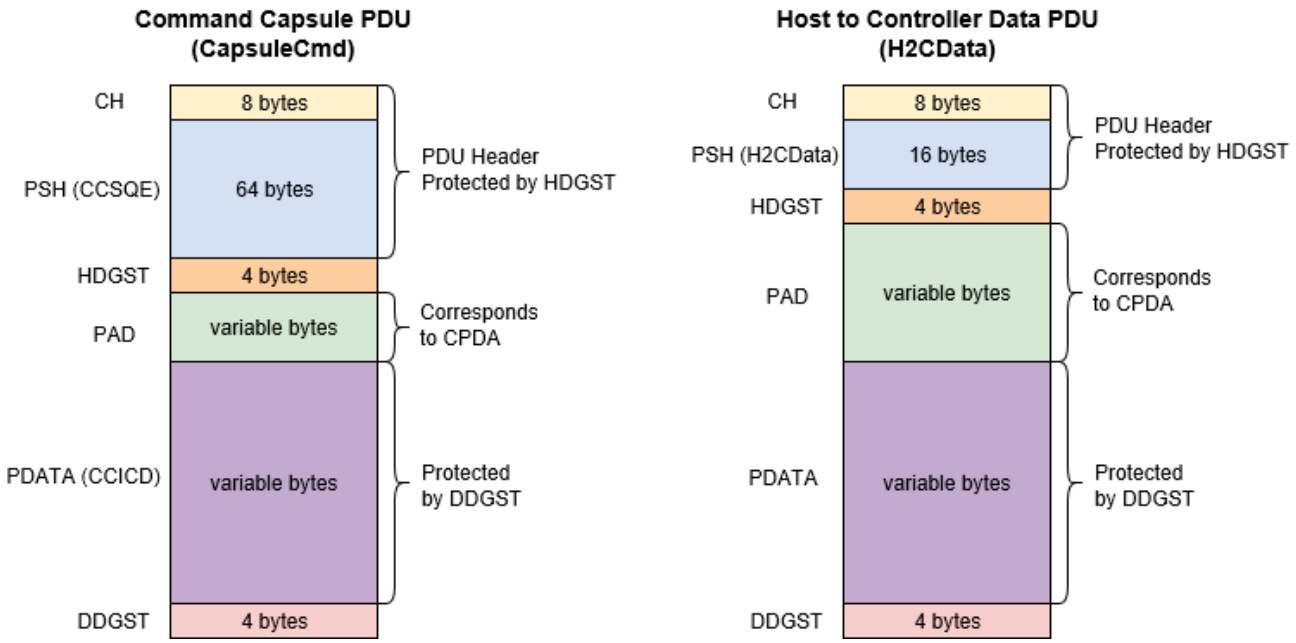
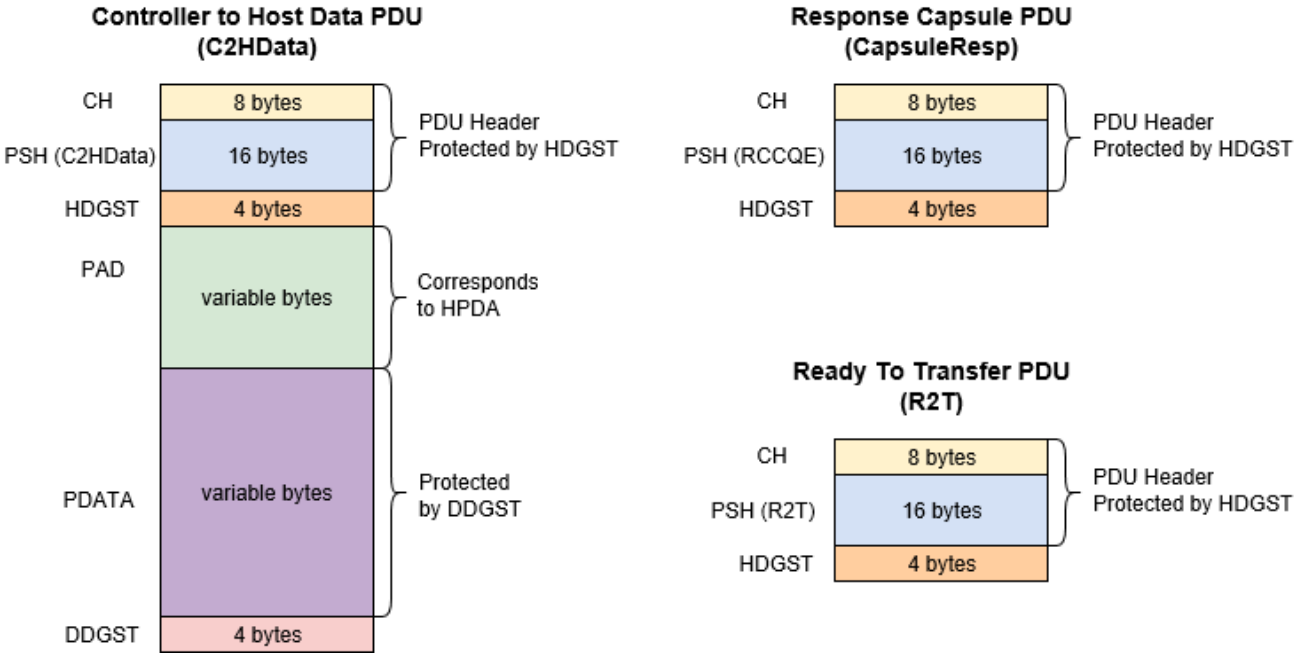


Figure tbd58: Controller to Host NVMe/TCP PDU Digests



7.4.6.1 Digest Error handling

A PDU header digest error impacts TCP byte stream synchronization. When a PDU header digest error is detected, the PDU header fields including the PDU length are not reliable making it impossible to reliably determine the length of the PDU. When a host detects a PDU header digest error, the host shall treat the error as a fatal transport error with the Fatal Error Status field set to “Header Digest Error”. Similarly, when a controller detects a PDU header digest error, the controller shall treat the error as a fatal transport error with the Fatal Error Status field set to “Header Digest Error”.

A PDU data digest error impacts the integrity of PDU data but does not impact the TCP byte stream synchronization. A PDU data digest error detected by the host or the controller is treated as a non-fatal transport error (refer to section 7.4.7). When a host detects a data digest error in a C2HData PDU, it shall continue processing C2HData PDUs associated with the command and when the command processing has completed, if a successful status was returned by the controller, the host shall fail the command with a non-fatal transport error.

When the controller detects a data digest error in a CapsuleCmd PDU with in-capsule data, the controller shall fail the command with a non-fatal transport error. When a controller detects a data digest error in a H2CData PDU, it may continue processing H2CData PDUs associated with the command or terminate the command processing early. When the command processing has completed or terminated, the controller shall fail the command with a non-fatal transport error.

7.4.7 Transport Error Handling

Errors that effect the transport but from which the transport is not able to recover normal operation are fatal errors. NVMe/TCP transport fatal errors are handled by terminating the connection.

When a controller detects a fatal error, it shall:

1. stop processing any PDUs that arrive on the connection; and
2. send a C2HTermReq PDU (refer to section 7.4.10.4) with:
    a. an appropriate Fatal Error Status field (FES);
    b. additional error information in the Fatal Error Information (FEI) field if applicable; and
    c. set the C2HTermReq PDU data with the PDU header that was being processed when the fatal error was detected.

In response to a C2HTermReq PDU, the host shall terminate the connection. If the host does not terminate the connection within 30 s, the controller may terminate the connection. The maximum C2HTermReq PDU data size shall not exceed 128 bytes. The host shall ignore a C2HTermReq PDU with a PDU length (PLEN) that exceeds 152 bytes (24-byte PDU header plus 128-byte PDU data) and shall terminate the connection immediately. Additionally, the host shall ignore a C2HTermReq PDU with PDU length (PLEN) less than 24 bytes and shall terminate the connection immediately. If the controller is unable to send a C2HTermReq PDU, then the controller shall reset the TCP connection.

When a host detects a fatal error, it shall:

1. stop processing any PDUs that arrive on the connection; and
2. send a H2CTermReq PDU with:

    a. an appropriate Fatal Error Status field (FES);
    b. additional error information in the Fatal Error Information (FEI) field if applicable; and
    c. set the H2CTermReq PDU data with the PDU header that was being processed when the fatal error was detected.

In response to a H2CTermReq PDU, the controller shall terminate the connection. If the controller does not terminate the connection within 30 s, the host may terminate the connection. The maximum H2CTermReq PDU data size shall not exceed 128 bytes. The controller shall ignore a H2CTermReq PDU with a PDU length (PLEN) that exceeds 152 bytes (24-byte PDU header plus 128-byte PDU data) and shall terminate the connection immediately. Additionally, the controller shall ignore a H2CTermReq PDU with PDU length (PLEN) less than 24 bytes and shall terminate the connection immediately. If the host is unable to send a H2CTermReq PDU, then the host shall reset the TCP connection.

Errors that effect the transport but from which the transport is able to recover normal operation are non-fatal errors. A non-fatal error may affect one or more commands. These commands are likely to complete successfully if retried. When a non-fatal transport error is detected, affected commands are completed with “Transient Transport Error” status code (refer to Figure 31 in TP 4028a for more information).

7.4.8 Keep Alive

The NVMe/TCP Transport requires the use of the Keep Alive feature (refer to section 7.11 in the NVMe base specification). The NVMe/TCP Transport does not impose any limitations on the minimum and maximum Keep Alive Timeout value. The minimum should be set large enough to account for any transient fabric interconnect failures between the host and controller.

TCP level Keep Alive functionality is not prohibited but it is recommended that TCP level Keep Alive timeout is set to a higher value than the NVMe Keep Alive Timeout to avoid conflicting policies.

7.4.9 Transport Specific Address Subtype and Transport Service Identifier

The Discovery Log Entry includes a Transport Specific Address Subtype (TSAS) field that is defined in Figure tbd17 for the NVMe/TCP Transport. The TSAS field describes TCP connection properties, such as whether TLS is supported.

Figure tbd59: Transport Specific Address Subtype Definition for NVMe/TCP Transport

<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>

TLS implementation is optional for NVMe/TCP.

TLS protocol versions prior to 1.2 shall not be used with NVMe/TCP (refer to section 3.1.1 of RFC 7525). All versions of SSL, the predecessor protocol to TLS, shall not be used with NVMe/TCP. For further discussion, refer to section 3.1.1 of RFC 7525. The NVMe/TCP prohibition on versions of TLS prior to 1.2 is stronger than the requirements in RFC 7525 because NVMe/TCP is a new protocol.

7.4.9.1 Mandatory and Recommended Cipher Suites

TLS for NVMe/TCP is based on pre-shared key (PSK) cipher suites. NVMe/TCP implementations that implement TLS shall support the TLS_PSK_WITH_AES_128_GCM_SHA256 {00h, A8h} cipher suite (refer to RFC 5487), and NVM subsystems should include that cipher suite in the initial set of cipher suites proposed to a host. In addition, the following cipher suites should be supported (refer to RFC 5487):

- TLS_PSK_WITH_AES_256_GCM_SHA384 {00h, A9h} cipher suite;
- TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 {00h, AAh} cipher suite; and
- TLS_DHE_PSK_WITH_AES_256_GCM_SHA384 {00h, ABh} cipher suite.

The _AES_128_ and _AES_256_ cipher suites differ in cryptographic strength (e.g., the _AES_128 cipher suites specify the use of 128-bit AES encryption and the _AES_256_ cipher suites specify the use of 256-bit AES encryption). The _DHE_ cipher suites differ from their non-_DHE_ cipher suite counterparts in the addition of an ephemeral Diffie-Hellman (DH) exchange to protect encrypted traffic against compromise of the pre-shared key (refer to section 6.3 of RFC 7525). The DH keys (also called exponents) used in any ephemeral DH exchange:

- shall be at least 2048 bits in size (refer to section 4.3 of RFC 7525); and
- should not be reused (i.e., used for more than one DH exchange) (refer to section 6.4 of RFC 7525).

The PSK cipher suite framework is described in RFC 4279. NVMe/TCP uses NQNs to identify hosts and NVM subsystems, specifically, in the TLS handshake for a PSK cipher suite:

- The psk_identity field in the ClientKeyExchange message shall contain the host NQN and the subsystem NQN separated by a space (' '=U+0020) character as a UTF-8 string, including the terminating null (00h) character.

The following is an example of the psk_identity field in the ClientKeyExchange message assuming that both the host and the NVM subsystem are using the UUID-based format NVMe Qualified Names:

- *nqn.2014-08.org.nvmexpress:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6 nqn.2014-08.org.nvmexpress:uuid:36ebf5a9-1df9-47b3-a6d0-e9ba32e428a2*.

For interoperability reasons, NVMe/TCP prohibits identity hints in the ServerKeyExchange message. The host is expected to be able to determine which identity and pre-shared key to use with the subsystem based on the NVM subsystem NQN indicated in a corresponding discovery log entry acquired before the client Key Exchange message was sent.

7.4.9.2 TLS Implementations and Use Requirements

The following requirements apply to use of TLS with NVMe/TCP:

- TLS compression shall not be used, as it is not secure (refer to section 3.3 of RFC 7525). This NVMe/TCP prohibition of TLS compression is stronger than the requirements in RFC 7525 because NVMe/TCP is a new protocol;
- If TLS session resumption is supported, the implementation of session resumption shall comply with the requirements in section 3.4 of RFC 7525; and
- If TLS renegotiation is supported, the renegotiation_info extension shall be implemented and used for all renegotiations as described in RFC 5746 (refer to section 3.5 of RFC 7525).

All NVMe/TCP host and subsystem implementations shall be configurable to require that all NVMe/TCP connections use TLS. If a host that supports TLS for NVMe/TCP receives a discovery log entry indicating that the NVM subsystem uses NVMe/TCP and does not support TLS, then the host should nonetheless attempt to establish an NVMe/TCP connection that uses TLS. This requirement applies independent of whether the host is configured to require use of TLS for all NVMe/TCP connections.

7.4.9.3 Transport Service Identifier

TCP port 4420 has been assigned for use by NVMe over Fabrics and TCP port 8009 has been assigned by IANA (refer to https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml) for use by NVMe over Fabrics discovery. TCP port 8009 is the default TCP port for NVMe/TCP discovery controllers. There is no default TCP port for NVMe/TCP I/O controllers, the Transport Service Identifier (TRSVCID) field in the Discovery Log Entry indicates the TCP port to use.

The TCP ports that may be used for NVMe/TCP I/O controllers include TCP port 4420, and the Dynamic and/or Private TCP ports (i.e., ports in the TCP port number range from 49152 to 65535). NVMe/TCP I/O controllers should not use TCP port 8009. TCP port 4420 shall not be used for both NVMe/iWARP and NVMe/TCP at the same IP address on the same network.

The TRSVCID field in a Discovery Log Entry for the NVMe/TCP transport shall contain a TCP port number in decimal representation as an ASCII string. If such a TRSVCID field does not contain a TCP port number in decimal representation as an ASCII string, then the host shall not use the information in that Discovery Log Entry to connect to a controller.

7.4.10 NVMe/TCP PDUs

This section describes the format of NVME/TCP PDUs.

7.4.10.1 PDU Common Header (CH)

Figure tbd60: PDU Common Header (CH)

Byte	Description
00	**PDU-Type:** Specifies the type of PDU. This value is also referred to as the PDU opcode. Refer to Figure tbd7 for PDU opcodes.
01	**FLAGS:** PDU-TYPE specific flags
02	**Header Length (HLEN):** Length of PDU header (not including the Header Digest) in bytes.
03	**PDU Data Offset (PDO):** PDU Data Offset from the start of the PDU in bytes.
07:04	**PDU Length (PLEN):** Total length of PDU (includes CH, PSH, HDGST, PAD, DATA, and DDGST) in bytes.

7.4.10.2 Initialize Connection Request PDU (ICReq)

Figure tbd61: Initialize Connection Request PDU (ICReq)

<table>
<tr><th>Byte</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="5">CH</td><td>PDU-TYPE: 00h</td></tr>
<tr><td>01</td><td>FLAGS: Reserved</td></tr>
<tr><td>02</td><td>HLEN: Fixed length of 128 bytes (80h).</td></tr>
<tr><td>03</td><td>PDO: Reserved</td></tr>
<tr><td>07:04</td><td>PLEN: Fixed length of 128 bytes (80h).</td></tr>
<tr><td>09:08</td><td rowspan="5">PSH</td><td>PDU Format Version (PFV): Specifies the format version of NVMe/TCP PDUs. The format of the record specified in this definition shall be cleared to 0h.</td></tr>
<tr><td>10</td><td>Host PDU Data Alignment (HPDA): Specifies the data alignment for all PDUs transferred from the controller to the host that contain data. This value is 0’s based value in units of dwords and must be a value in the range 0 to 31 (e.g., values 0, 1, and 2 correspond to 4 byte, 8 byte, and 12 byte alignment).</td></tr>
<tr><td>11</td><td>DGST: Host PDU header and Data digest enable options.
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0</td><td>HDGST_ENABLE: If set to ‘1’, the use of header digest is requested by the host for the connection. If cleared to ‘0’, header digest shall not be used for the connection.</td></tr>
<tr><td>1</td><td>DDGST_ENABLE: If set to ‘1’, the use of data digest is requested by the host for the connection. If cleared to ‘0’, data digest shall not be used for the connection.</td></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>15:12</td><td>Maximum Number of Outstanding R2T (MAXR2T): Specifies the maximum number of outstanding R2T PDUs for a command at any point in time on the connection. This is a 0’s based value.</td></tr>
<tr><td>127:16</td><td>Reserved</td></tr>
</table>

7.4.10.3 Initialize Connection Response PDU (ICResp)

Figure tbd62: Initialize Connection Response PDU (ICResp)

<table>
<tr><th>Byte</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="5">CH</td><td>PDU-TYPE: 01h</td></tr>
<tr><td>01</td><td>FLAGS: Reserved</td></tr>
<tr><td>02</td><td>HLEN: Fixed length of 128 bytes (80h).</td></tr>
<tr><td>03</td><td>PDO: Reserved</td></tr>
<tr><td>07:04</td><td>PLEN: Fixed length of 128 bytes (80h).</td></tr>
<tr><td>09:08</td><td rowspan="4">PSH</td><td>PDU Format Version (PFV): Specifies the format version of the NVMe/TCP PDUs. The format of the record specified in this definition shall be cleared to 0h.</td></tr>
<tr><td>10</td><td>Controller PDU Data Alignment (CPDA): Specifies the data alignment for all PDUs transferred from the host to the controller that contain data. This value is 0’s based value in units of dwords in the range 0 to 31 (e.g., values 0, 1, and 2 correspond to 4 byte, 8 byte, and 12 byte alignment).</td></tr>
<tr><td>11</td><td>DGST: PDU and Data Digest enable options.
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0</td><td>HDGST_ENABLE: If set to ‘1’, header digest is used for the connection. If cleared to ‘0’, header digest is not used for the connection.</td></tr>
<tr><td>1</td><td>DDGST_ENABLE: If set to ‘1’, data digest is used for the connection. If cleared to ‘0’, data digest is not used for the connection.</td></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>15:12</td><td>Maximum Host to Controller Data length (MAXH2CDATA): Specifies the maximum number of PDU-Data bytes per H2C Data Transfer PDU in bytes. This value is a multiple of dwords and should be no less than 4096.</td></tr>
<tr><td>127:16</td><td>Reserved</td></tr>
</table>

7.4.10.4 Host to Controller Terminate Connection Request PDU (H2CTermReq)

Figure tbd63: Host to Controller Terminate Connection Request PDU (H2CTermReq)

<table>
<tr><th>Byte</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="5">CH</td><td>PDU-TYPE: 02h</td></tr>
<tr><td>01</td><td>FLAGS: Reserved</td></tr>
<tr><td>02</td><td>HLEN: Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td>PDO: Reserved</td></tr>
<tr><td>07:04</td><td>PLEN: Total length of PDU (including PDU header and DATA) in bytes. This value shall not exceed a limit of 152 bytes.</td></tr>
<tr><td>09:08</td><td rowspan="3">PSH</td><td>Fatal Error Status (FES): Indicates the fatal error information.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>01h</td><td>Invalid PDU Header Field: An Invalid Field in the Transport Header was detected by the host.</td></tr>
<tr><td>02h</td><td>PDU Sequence Error: An unexpected Protocol sequence was detected by the host.</td></tr>
<tr><td>03h</td><td>Header Digest Error: A Header Digest (HDGST) error was detected by the host.</td></tr>
<tr><td>04h</td><td>Data Transfer Out of Range: A C2HData with data offset or data offset plus data length is out of its associated command data buffer range.</td></tr>
<tr><td>05h</td><td>R2T Limit Exceeded: An R2T PDU that exceeds MAXR2T was received by the host.</td></tr>
<tr><td>06h</td><td>Unsupported Parameter: An unsupported parameter was received by the host.</td></tr>
<tr><td>FFFFh:07h, 00h</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>13:10</td><td>Fatal Error Information (FEI): Provides additional information based on the Fatal Error Status field.
<table>
<tr><th>Fatal Error Status</th><th>Contents of Error Specific Information</th></tr>
<tr><td>01h</td><td>PDU Header Field Offset: This field indicates the offset in bytes from the start of the PDU Header to the start of the field that is in error. If multiple errors exist, then this field indicates the lowest offset that is in error.</td></tr>
<tr><td>03h</td><td>PDU Header Digest: This field indicates the header digest that was received by the host and caused a header digest verification error.</td></tr>
<tr><td>06h</td><td>Unsupported Parameter Field Offset: This field indicates the offset in bytes from the start of the PDU Header to the start of the field that is in error. If multiple errors exist, then this field indicates the lowest offset that is in error.</td></tr>
<tr><td>FFFFh:07h, 05h, 04h, 02h, 00h</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>23:14</td><td>Reserved</td></tr>
<tr><td>N - 1:24</td><td>DATA</td><td>Data: This field contains the PDU header that was being processed by the host while the fatal error was detected.</td></tr>
</table>

7.4.10.5 Controller to Host Terminate Connection Request PDU (C2HTermReq)

Figure tbd64: Controller to Host Terminate Connection Request PDU (C2HTermReq)

<table>
<tr><th>Byte</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="5">CH</td><td>PDU-TYPE: 03h</td></tr>
<tr><td>01</td><td>FLAGS: Reserved</td></tr>
<tr><td>02</td><td>HLEN: Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td>PDO: Reserved</td></tr>
<tr><td>07:04</td><td>PLEN: Total length of PDU (including PDU header and DATA) in bytes. This value shall not exceed a limit of 152 bytes.</td></tr>
<tr><td>09:08</td><td rowspan="3">PSH</td><td>Fatal Error Status (FES): Indicates the fatal error status information.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>01h</td><td>Invalid PDU Header Field: An Invalid Field in the Transport Header was detected by the controller.</td></tr>
<tr><td>02h</td><td>PDU Sequence Error: An unexpected Protocol sequence was detected by the controller.</td></tr>
<tr><td>03h</td><td>Header Digest Error: A Header Digest (HDGST) error was detected by the controller.</td></tr>
<tr><td>04h</td><td>Data Transfer Out of Range: A H2CData with data offset or data offset plus data length is out of its associated R2T range.</td></tr>
<tr><td>05h</td><td>Data Transfer Limit Exceeded: A H2CData PDU with data length that exceeds MAXH2CDATA was received by the controller.</td></tr>
<tr><td>06h</td><td>Unsupported Parameter: An unsupported parameter was received by the controller.</td></tr>
<tr><td>FFFFh:07h, 00h</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>13:10</td><td>Fatal Error Information (FEI): Provides additional information based on the Fatal Error Status field.
<table>
<tr><th>Fatal Error Status</th><th>Contents of Error Specific Information</th></tr>
<tr><td>01h</td><td>PDU Header Field Offset: This field indicates the offset in bytes from the start of the Transport Header to the start of the field that is in error. If multiple errors exist, then this field indicates the lowest offset that is in error.</td></tr>
<tr><td>03h</td><td>PDU Header Digest: This field indicates the header digest that was received by the controller and caused a header digest verification error.</td></tr>
<tr><td>06h</td><td>Unsupported Parameter Field Offset: This field indicates the offset in bytes from the start of the PDU Header to the start of the field that is in error. If multiple errors exist, then this field indicates the lowest offset that is in error.</td></tr>
<tr><td>FFFFh:07h,
05h, 04h, 02h, 00h</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>23:14</td><td>Reserved</td></tr>
<tr><td>N - 1:24</td><td>DATA</td><td>Data: This field contains the PDU header that was being processed by the host while the fatal error was detected.</td></tr>
</table>

7.4.10.6 Command Capsule PDU (CapsuleCmd)

Figure tbd65: Command Capsule PDU (CapsuleCmd)

<table>
<tr><th>Byte</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="5">CH</td><td>PDU-TYPE: 04h</td></tr>
<tr><td>01</td><td>FLAGS:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid Header digest value will trail the PDU header.</td></tr>
<tr><td>1</td><td>DDGSTF: If set to ‘1’, then a valid Data digest value will trail the PDU Data.</td></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>02</td><td>HLEN: Fixed length of 72 bytes (48h).</td></tr>
<tr><td>03</td><td>PDO: Data Offset within PDU. This value complies to the CPDA field set by the controller in ICResp PDU (refer to section 7.4.10.3).</td></tr>
<tr><td>07:04</td><td>PLEN: Total length of PDU (including PDU header, HDGST, PAD, DATA, and DDGST) in bytes.</td></tr>
<tr><td>71:08</td><td>PSH</td><td>NVMe-oF Command Capsule SQE (CCSQE): NVMe-oF Command Capsule SQE.</td></tr>
<tr><td>75:72</td><td>HDGST</td><td>HDGST: If HDGSTF is set in the FLAGS field, this field is present and contains the Header digest.</td></tr>
<tr><td>N - 1:76</td><td>PAD</td><td>PAD: If in-capsule data is present, the length of this shall be the necessary number of bytes needed to achieve the alignment specified by CPDA.</td></tr>
<tr><td>M - 1:N</td><td>DATA</td><td>NVMe-oF In-Capsule Data (CCICD): This field will contain the in-capsule data, if any, of the NVMe-oF Command Capsule.</td></tr>
<tr><td>M + 3:M</td><td>DDGST</td><td>Data Digest (DDGST): If DDGSTF is set in the FLAGS field, and the CCICD field is present, this field will contain the Data Digest of the CCICD field (in-capsule data).</td></tr>
</table>

7.4.10.7 Response Capsule PDU (CapsuleResp)

Figure tbd66: Response Capsule PDU (CapsuleResp)

<table>
<tr><th>Byte</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="4">CH</td><td>PDU-TYPE: 05h</td></tr>
<tr><td>01</td><td>FLAGS:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid Header digest value will trail the PDU header.</td></tr>
<tr><td>7:1</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>02</td><td>HLEN: Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td>PDO: Reserved</td></tr>
<tr><td>07:04</td><td>PLEN: Length of PDU Header and HDGST (if present) in bytes.</td></tr>
<tr><td>23:08</td><td>PSH</td><td>NVMe-oF Response Capsule CQE (RCCQE): Response Capsule CQE.</td></tr>
<tr><td>27:24</td><td>HDGST</td><td>HDGST: If HDGSTF is set in the FLAGS field, this field is present and contains the Header digest.</td></tr>
</table>

7.4.10.8 Host To Controller Data Transfer PDU (H2CData)

Figure tbd67: Host To Controller Data Transfer PDU (H2CData)

<table>
<tr><th>Byte</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="8">CH</td><td>PDU-TYPE: 06h</td></tr>
<tr><td rowspan="4">01</td><td>FLAGS:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid Header digest value will trail the PDU header.</td></tr>
<tr><td>1</td><td>DDGSTF: If set to ‘1’, then a valid Data digest value will trail the PDU Data.</td></tr>
<tr><td>2</td><td>LAST_PDU: If set to ‘1’, indicates the PDU is the last in the set of H2CData PDUs that correspond to the same R2T PDU.</td></tr>
<tr><td>7:3</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>02</td><td>HLEN: Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td>PDO: Data Offset within PDU. This value complies to the CPDA field set by the controller in ICResp PDU (refer to section 7.4.9.3).</td></tr>
<tr><td>07:04</td><td>PLEN: Total length of PDU (including PDU header, HDGST, PAD, DATA, and DDGST) in bytes.</td></tr>
<tr><td>09:08</td><td rowspan="5">PSH</td><td>Command Capsule CID (CCCID): This field contains the SQE.CID value of the Command Capsule associated with the command data buffer.</td></tr>
<tr><td>11:10</td><td>Transfer Tag (TTAG): This field contains the Transfer Tag of the corresponding R2T received by the controller.</td></tr>
<tr><td>15:12</td><td>Data Offset (DATAO): Byte offset from start of Command data buffer. This value shall be a multiple of dwords.</td></tr>
<tr><td>19:16</td><td>Data Length (DATAL): PDU DATA field length in bytes. This value shall be a multiple of dwords.</td></tr>
<tr><td>23:20</td><td>Reserved</td></tr>
<tr><td>27:24</td><td>HDGST</td><td>HDGST: If HDGSTF is set in the FLAGS field, this field is present and contains the Header digest.</td></tr>
<tr><td>N - 1:28</td><td>PAD</td><td>PAD: The length of this shall be the necessary number of bytes needed to achieve the alignment specified by CPDA.</td></tr>
<tr><td>M - 1:N</td><td>DATA</td><td>PDU-Data</td></tr>
<tr><td>M + 3:M</td><td>DDGST</td><td>Data Digest (DDGST): If DDGSTF is set in the FLAGS field, this field is present and contains the data digest.</td></tr>
</table>

7.4.10.9 Controller To Host Data Transfer PDU (C2HData)

Figure tbd68: Controller To Host Data Transfer PDU (C2HData)

<table>
<tr><th>Byte</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="8">CH</td><td>PDU-TYPE: 07h</td></tr>
<tr><td rowspan="2">01</td><td>FLAGS:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid Header digest value will trail the PDU header.</td></tr>
<tr><td>1</td><td>DDGSTF: If set to ‘1’, then a valid Data digest value will trail the PDU Data.</td></tr>
<tr><td>2</td><td>LAST_PDU: If set to ‘1’, indicates the PDU is the last in the Command Data transfer series.</td></tr>
<tr><td>3</td><td>SUCCESS: If set to ‘1’, indicates that the command referenced by CCCID was completed successfully with no other needed information and that a response PDU will not be sent by the Controller.</td></tr>
<tr><td>7:4</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>02</td><td>HLEN: Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td>PDO: Data Offset within PDU. This value complies to the HPDA field set by the controller in ICReq PDU (refer to section 7.4.9.2).</td></tr>
<tr><td>07:04</td><td>PLEN: Total length of PDU (including PDU header, HDGST, PAD, DATA, and DDGST) in bytes.</td></tr>
<tr><td>09:08</td><td rowspan="5">PSH</td><td>Command Capsule CID (CCCID): This field contains the SQE.CID value of the Command Capsule associated with the host resident data.</td></tr>
<tr><td>11:10</td><td>Reserved</td></tr>
<tr><td>15:12</td><td>Data Offset (DATAO): Byte offset from start of host resident data. This value shall be dword aligned.</td></tr>
<tr><td>19:16</td><td>Data Length (DATAL): PDU DATA field length in bytes. This value shall be dword aligned.</td></tr>
<tr><td>23:20</td><td>Reserved</td></tr>
<tr><td>27:24</td><td>HDGST</td><td>HDGST: If HDGSTF is set in the FLAGS field, this field is present and contains the Header digest.</td></tr>
<tr><td>N - 1:28</td><td>PAD</td><td>PAD: If HPDA is set to a non-zero value, this field will be padded as specified by HPDA.</td></tr>
<tr><td>M - 1:N</td><td>DATA</td><td>PDU-Data</td></tr>
<tr><td>M + 3:M</td><td>DDGST</td><td>Data Digest (DDGST): Data Digest of PDU-Data field.</td></tr>
</table>

7.4.10.10 Ready to Transfer PDU (R2T)

Figure tbd69: Ready to Transfer PDU (R2T)

<table>
<tr><th>Byte</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="5">CH</td><td>PDU-TYPE: 09h</td></tr>
<tr><td>01</td><td>FLAGS:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid Header digest value will trail the PDU header</td></tr>
<tr><td>7:1</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>02</td><td>HLEN: Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td>PDO: Reserved</td></tr>
<tr><td>07:04</td><td>PLEN: Length of PDU Header and HDGST (if present) in bytes.</td></tr>
<tr><td>09:08</td><td rowspan="5">PSH</td><td>Command Capsule CID (CCCID): This field contains the SQE.CID value of the Command Capsule associated with the host resident data.</td></tr>
<tr><td>11:10</td><td>Transfer Tag (TTAG): This field contains a controller generated tag. The rules of the tag generation are completely up to the controller discretion.</td></tr>
<tr><td>15:12</td><td>Requested Data Offset (R2TO): Byte offset from the start of the host-resident data. This value shall be dword aligned.</td></tr>
<tr><td>19:16</td><td>Requested Data Length (R2TL): Number of bytes of command data buffer requested by the controller. This value shall be dword aligned.</td></tr>
<tr><td>23:20</td><td>Reserved</td></tr>
<tr><td>24:27</td><td>HDGST</td><td>HDGST: If HDGSTF is set in the FLAGS field, this field is present and contains the Header digest.</td></tr>
</table>