NVM Express®

TCP Transport Specification

Revision 1.0

May 18th, 2021

Please send comments to info@nvmexpress.org

NVM Express® TCP Transport Specification is available for download at http://nvmexpress.org. The NVMe TCP Transport Specification revision 1.0 incorporates NVMe-oF ECN001, NVMe-oF ECN002 and TP8011.

SPECIFICATION DISCLAIMER

LEGAL NOTICE:

This NVM Express TCP Transport Specification is proprietary to the NVM Express, Inc. (also referred to as "Company") and/or its successors and assigns.

NOTICE TO USERS WHO ARE NVM EXPRESS, INC. MEMBERS: Members of NVM Express, Inc. have the right to use and implement this NVM Express TCP Transport Specification subject, however, to the Member's continued compliance with the Company's Intellectual Property Policy and Bylaws and the Member's Participation Agreement.

NOTICE TO NON-MEMBERS OF NVM EXPRESS, INC.: If you are not a Member of NVM Express, Inc. and you have obtained a copy of this document, you only have a right to review this document or make reference to or cite this document. Any such references or citations to this document must acknowledge NVM Express, Inc. copyright ownership of this document. The proper copyright citation or reference is as follows: **"© 2021 NVM Express, Inc. ALL RIGHTS RESERVED."** When making any such citations or references to this document you are not permitted to revise, alter, modify, make any derivatives of, or otherwise amend the referenced portion of this document in any way without the prior express written permission of NVM Express, Inc. Nothing contained in this document shall be deemed as granting you any kind of license to implement or use this document or the specification described therein, or any of its contents, either expressly or impliedly, or to any intellectual property owned or controlled by NVM Express, Inc., including, without limitation, any trademarks of NVM Express, Inc.

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.

The NVM Express® design mark is a registered trademark of NVM Express, Inc.

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

Table of Contents

Table of Figures

1 Introduction

1.1 Overview

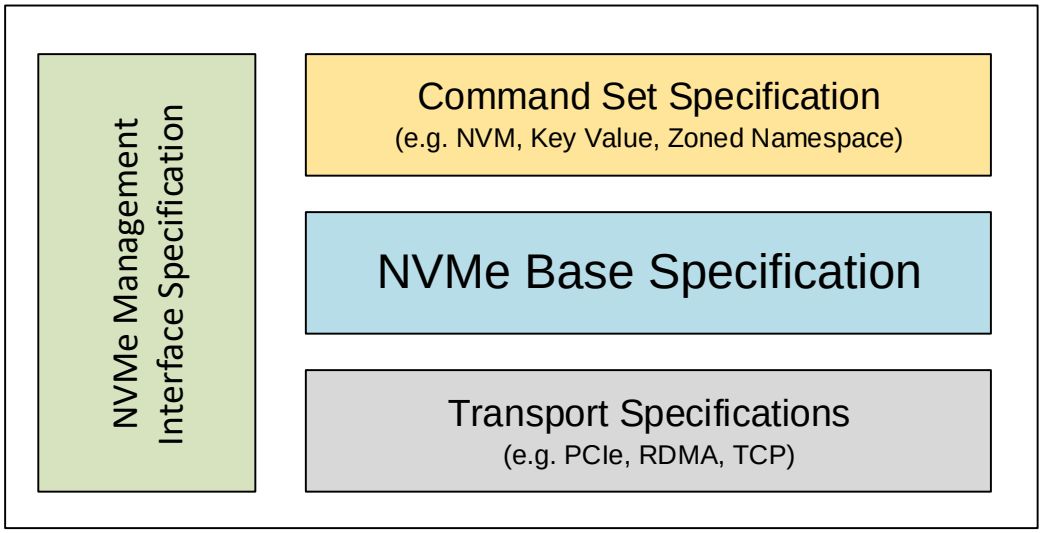
NVM Express® (NVMe®) Base specification defines an interface for host software to communicate with non-volatile memory subsystems over a variety of memory-based transports and message-based transports.

This document defines mappings of extensions defined in the NVMe Base Specification to a specific NVMe Transport: TCP.

1.2 Scope

Figure 1 shows the relationship of the NVM Command Set Specification to other specifications within the NVMe Family of Specifications.

Figure 1: NVMe Family of Specifications



This specification supplements the NVMe Base Specification. This specification defines additional data structures, features, log pages, commands, and/or status values. This specification also defines extensions to existing data structures, features, log pages, commands, and/or status values. This specification defines requirements and behaviors that are specific to the TCP transport. Functionality that is applicable generally to NVMe or that is applicable across multiple NVMe transports is defined in the NVMe Base specification.

If a conflict arises among requirements defined in different specifications, then a lower-numbered specification in the following list shall take precedence over a higher-numbered specification:

1. Non-NVMe specifications
2. NVMe Base specification
3. NVMe transport specifications
4. NVMe I/O command set specifications
5. NVMe-MI specification

1.3 Conventions

This specification conforms to the Conventions section, Keywords section and the Byte, Word, and Dword Relationships section of the NVMe Base Specification.

1.4 Definitions

1.4.1 Established TCP Connection

Two TCP endpoints that have an established full-duplex communication channel between them and are ready for data transfer.

1.4.2 Obsolete

Keyword indicating functionality that was defined in a previous version of this specification and that has been removed from this specification.

1.4.3 TCP

Transmission Communication Protocol

1.4.4 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.

1.4.5 TCP/IP Packet

A TCP segment encapsulated in an Internet Protocol (IP) datagram creating a TCP/IP packet.

1.5 References

NVM Express Base specification, Revision 2.0. Available from http://www.nvmexpress.org.

RFC 1952, P. Deutsch, “GZIP file format specification version 4.3”, May 1996. Available from https://www.ietf.org/rfc.html.

RFC 4648, S. Josefsson, “The Base16, Base32, and Base64 Data Encodings”, October 2006. Available from https://www.ietf.org/rfc.html.

RFC 5869, H. Krawczyk, P. Eronen, “HMAC-based Extract-and-Expand Key Derivation Functions (HKDF)”, May 2010. Available from https://www.ietf.org/rfc.html.

RFC 7296, C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, T. Kivinen, “Internet Key Exchange Protocol Version 2 (IKEv2)”, October 2014. Available from https://www.ietf.org/rfc.html.

RFC 8446, E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, August 2018. Available from https://www.ietf.org/rfc.html.

RFC 8996, K. Moriarty, S. Farrell, “Deprecating TLS 1.0 and TLS 1.1”, March 2021. Available from https://www.ietf.org/rfc.html.

1.6 Acronyms

Figure 2: NVMe/TCP Acronym Descriptions

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

Figure 2: NVMe/TCP Acronym Descriptions

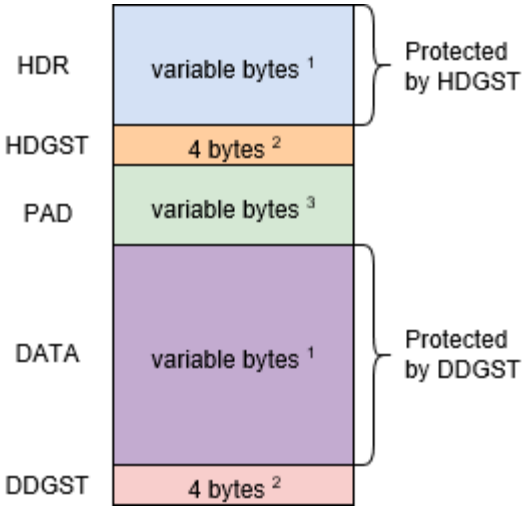
Term	Definition
HPDA	Host PDU Data Alignment
H2C	Host to Controller Direction
H2CTermReq	Host to Controller Terminate Connection Request
ICReq	Initialize Connection Request
ICResp	Initialize Connection Response
PAD	PDU padding bytes (before DATA starts)
PDU	Protocol Data Unit
PFV	Protocol Format Version
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

2 Transport Overview

This section describes how the TCP Transport provides reliable in-order capsule delivery and direct data placement of Admin and I/O command data 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 3, 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 3.6.2).

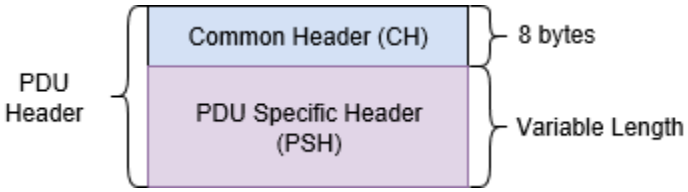
Figure 3: NVMe/TCP PDU Structure



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 3.6.2.1).

Figure 4: NVMe/TCP PDU Header



As shown in Figure 5 and Figure 6, there is no requirement to align NVMe/TCP PDU Headers nor PDU payloads[1] to TCP/IP packet boundaries.

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

Figure 5: Multiple NVMe/TCP PDUs in a Single TCP/IP Packet

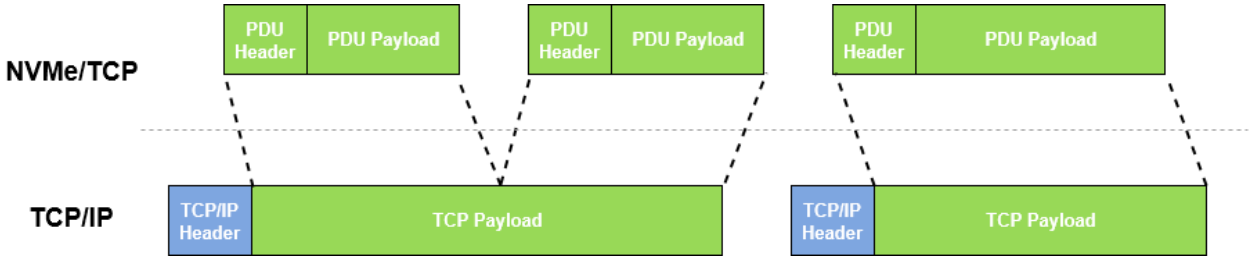
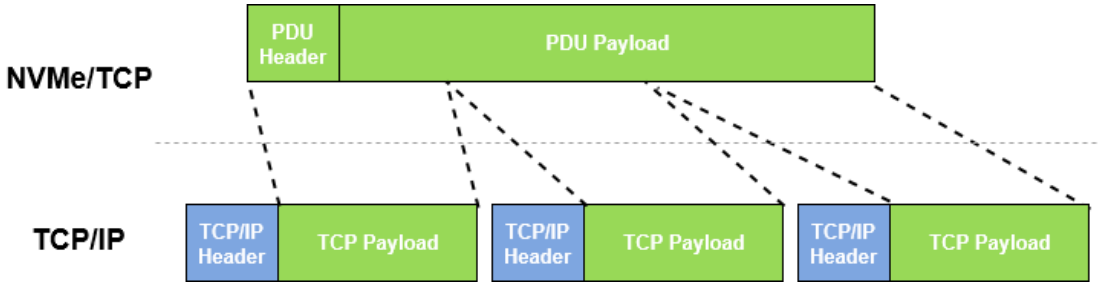


Figure 6: NVMe/TCP PDU Spanning across TCP/IP Packets



The 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.

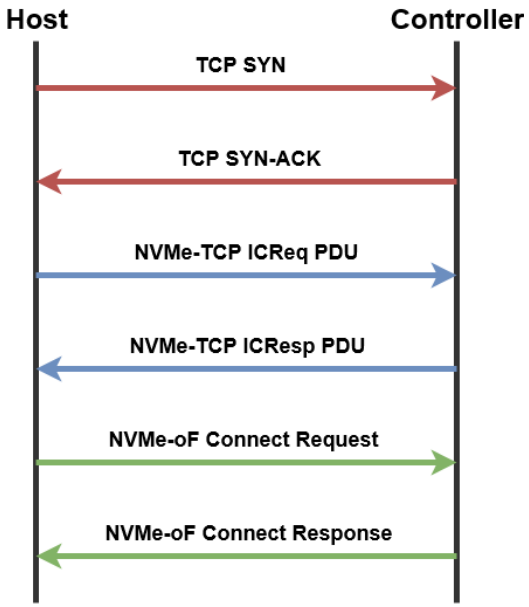
3 Transport Binding

3.1 Setup & Initialization

Figure 7 illustrates the process used to establish an 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, that controller 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 7: 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 3.5). 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 (refer to section 3.5). 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.

3.1.1 Transport Specific Address Subtype

The Discovery Log Entry includes a Transport Specific Address Subtype (TSAS) field that describes TCP connection properties such as whether TLS is used (refer to section 3.6.1.1). The Discovery Log Entry also

includes a Transport Service Identifier (TRSVCID) field that describes the TCP port to use (refer to section 3.1.2).

3.1.2 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.

3.2 Queue Model Instantiation

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.

3.3 Data Transfer Model

The TCP transport is a message-based transport that uses capsules for data transfer as defined in the Capsules and Data Transfer section of the NVMe Base Specification. All NVMe/TCP implementations shall support data transfers using Command Data Buffers (described in section 3.3.2) 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 that the data (and data digest) starting offset 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 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 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. Neither the Host PDU Data Alignment field (HPDA) nor the Controller PDU Data Alignment field shall exceed 128 bytes.

Figure 8 shows an example of an H2CData PDU where the CPDA field (refer to section 3.6.2.3) was set to 0Fh by the controller in the 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 8: Example of 64B PDU DATA Alignment in H2CData PDU

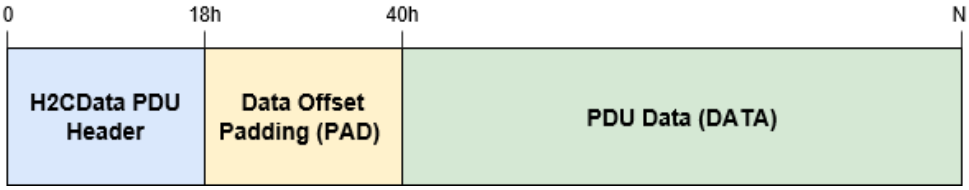
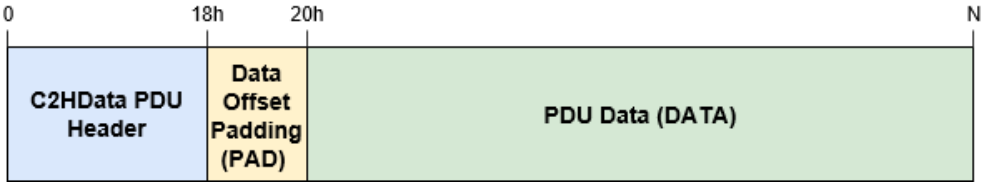


Figure 9 shows an example of a C2HData PDU where the HPDA field (refer to section 3.6.2.2) was set to 03h by the host in the ICReq when the connection was established. The C2HData PDU Header size 24 bytes and header digest is disabled. An alignment of 16 bytes is required, thus the controller inserts 8 bytes of padding in the PAD field.

Figure 9: Example of 64B PDU DATA Alignment in C2HData PDU



The TCP transport defines NVMe/TCP PDU types that are summarized in Figure 10 and defined in section 3.6.2. 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 3.5).

Figure 10: 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	3.6.2.2	**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	3.6.2.3	**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	3.6.2.4	**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	3.6.2.5	**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	3.6.2.6	**Command Capsule:** A PDU sent from a host to a controller to transfer an NVMe over Fabrics Command Capsule
CapsuleResp	0000010b	1b	05h	3.6.2.7	**Response Capsule:** A PDU sent from a controller to a host to transfer an NVMe over Fabrics Response Capsule
H2CData	0000011b	0b	06h	3.6.2.8	**Host to Controller Data:** A PDU sent from a host to a controller to transfer data to the controller
C2HData	0000011b	1b	07h	3.6.2.9	**Controller to Host Data:** A PDU sent from a controller to a host to transfer data to the host

Figure 10: NVMe/TCP PDU Types

PDU Name	Opcode by field		Combined Opcode [2]	Section	PDU Description
	Function (07:01)	PDU Direction [1] (00)			
R2T	0000100b	1b	09h	3.6.2.10	**Ready to Transfer:** A PDU sent from a controller to a host to indicate that the controller 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.

3.3.1 Capsules

The NVMe/TCP transport supports a message-based model. Data transfers are supported via a transport specific data transfer mechanism, described in section 3.3, and optionally via in-capsule data. NVMe/TCP capsule sizes are summarized in Figure 11. 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 8,192 bytes, causing their maximum size to be 8,256 bytes (i.e., 64 bytes + 8,192 bytes). If an Admin or Fabrics Command Capsule requires more than 8,192 bytes of PDU Data to be transferred, then the NVMe/TCP data transfer mechanism described in section 3.3.2.2 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 3.3.2.1.

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 fits in a Command Capsule's PDU Data, then the NVMe/TCP data transfer mechanism described in section 3.3.2.2 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 3.3.2.1. 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 3.3).

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.

Figure 11: NVMe/TCP Capsule Size

Capsule Type	In-Capsule Data Not Supported (bytes)	In-Capsule Data Supported (bytes)
Fabrics and Admin Commands	N/A [1]	64 to 8,256
Fabrics and Admin Responses	16	N/A [2]
I/O Queue Command	64	64 to (IOCCSZ × 16)
I/O Queue Response	16	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.

3.3.1.1 PDU Header and Data Digests

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

The host requests the use of a header digest by setting the HDGST_ENABLE flag in the ICReq PDU. The controller may accept (or reject) the use of a header digest by setting to '1' (or clearing to '0') the HDGST_ENABLE flag in the ICResp PDU. The PDU Header Digest is enabled if HDGST_ENABLE flag is set in both the ICReq and ICResp PDUs. If the 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 the PDU 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 shall have the HDGSTF flag, if defined, cleared to '0' in the PDU Header FLAGS field.

The host requests the use of a data digest by setting the DDGST_ENABLE flag in the ICReq PDU. The controller may accept (or reject) the use of a data digest by setting to '1' (or clearing to '0') the DDGST_ENABLE flag in the ICResp PDU. The PDU Data Digest is enabled if the DDGST_ENABLE flag is set in both the ICReq and ICResp PDUs. If the PDU Data Digest is enabled, then Command Capsule PDUs containing in-capsule data, H2CData PDUs, 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 the PDU Data Digest is not enabled, then these PDUs shall not contain a DDGST field and shall 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 digest 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 3.1).

If a host did not request the use of header digest 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 that ICResp PDU as a fatal transport error (refer to section 3.5) with the Fatal Error Status field set to "Invalid PDU Header Field" and the Additional Error Information field containing the HDGST or DDGST field byte offset.

The HDGST and DDGST are calculated using the CRC32C algorithm (refer to http://www.rfc-editor.org/rfc/rfc3385.txt).

Figure 12: Host to Controller NVMe/TCP PDU Digests

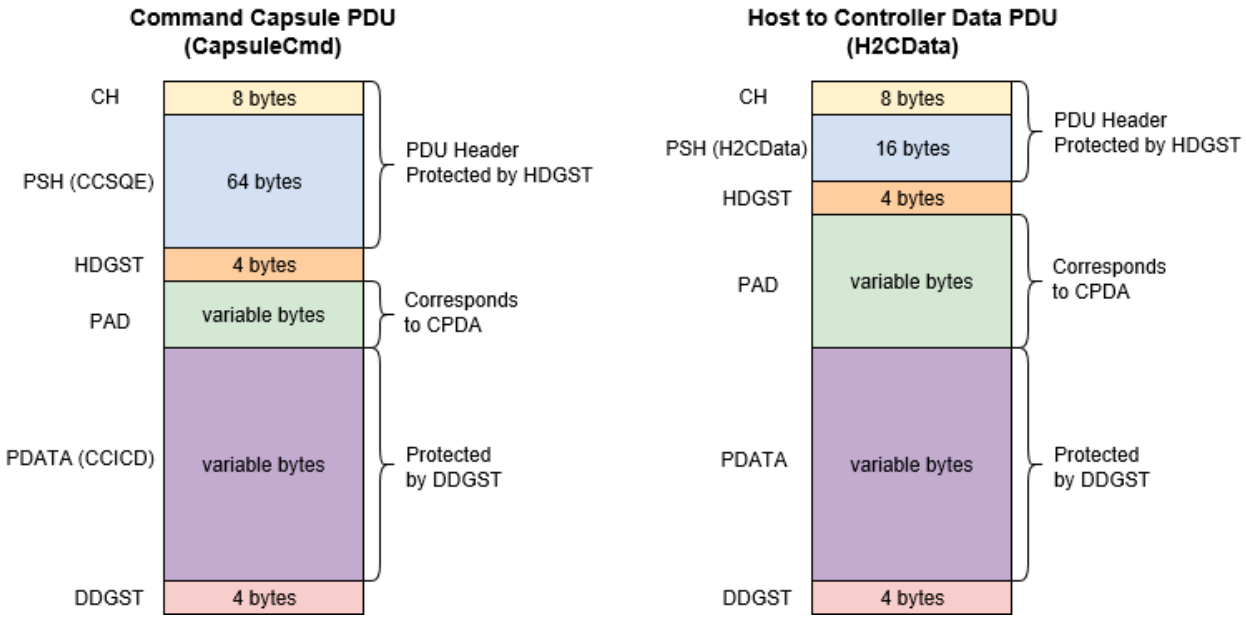
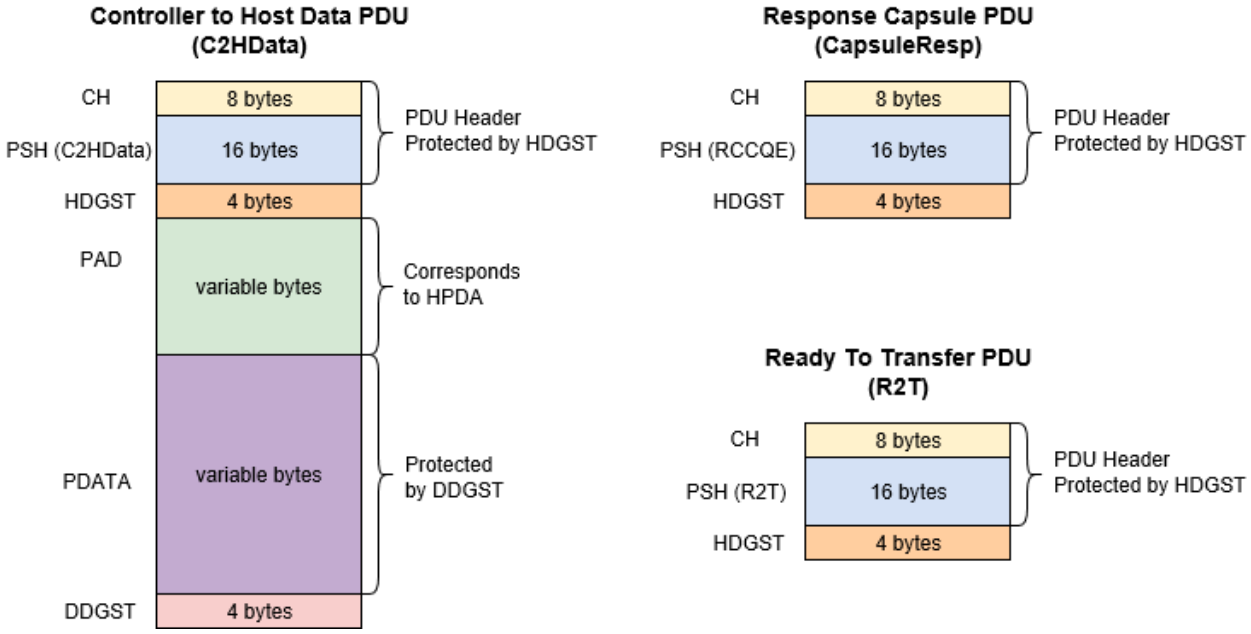


Figure 13: Controller to Host NVMe/TCP PDU Digests



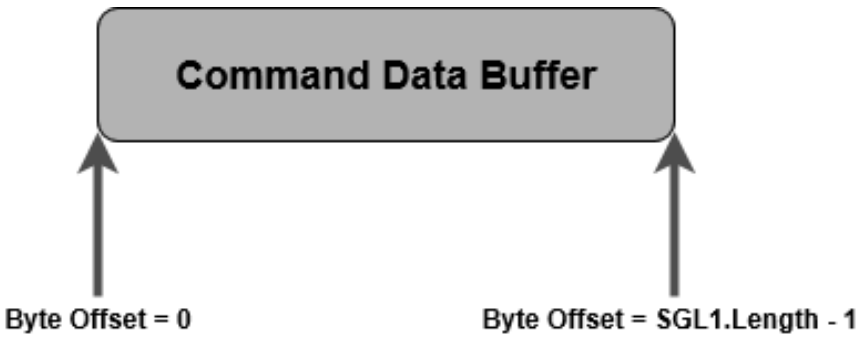
3.3.2 Command Data Buffers and SGLs

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 14).

A 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 14: Command Data Buffer Transport SGL Data Block Descriptor



3.3.2.1 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 3.5).

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 (SQHD) 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.

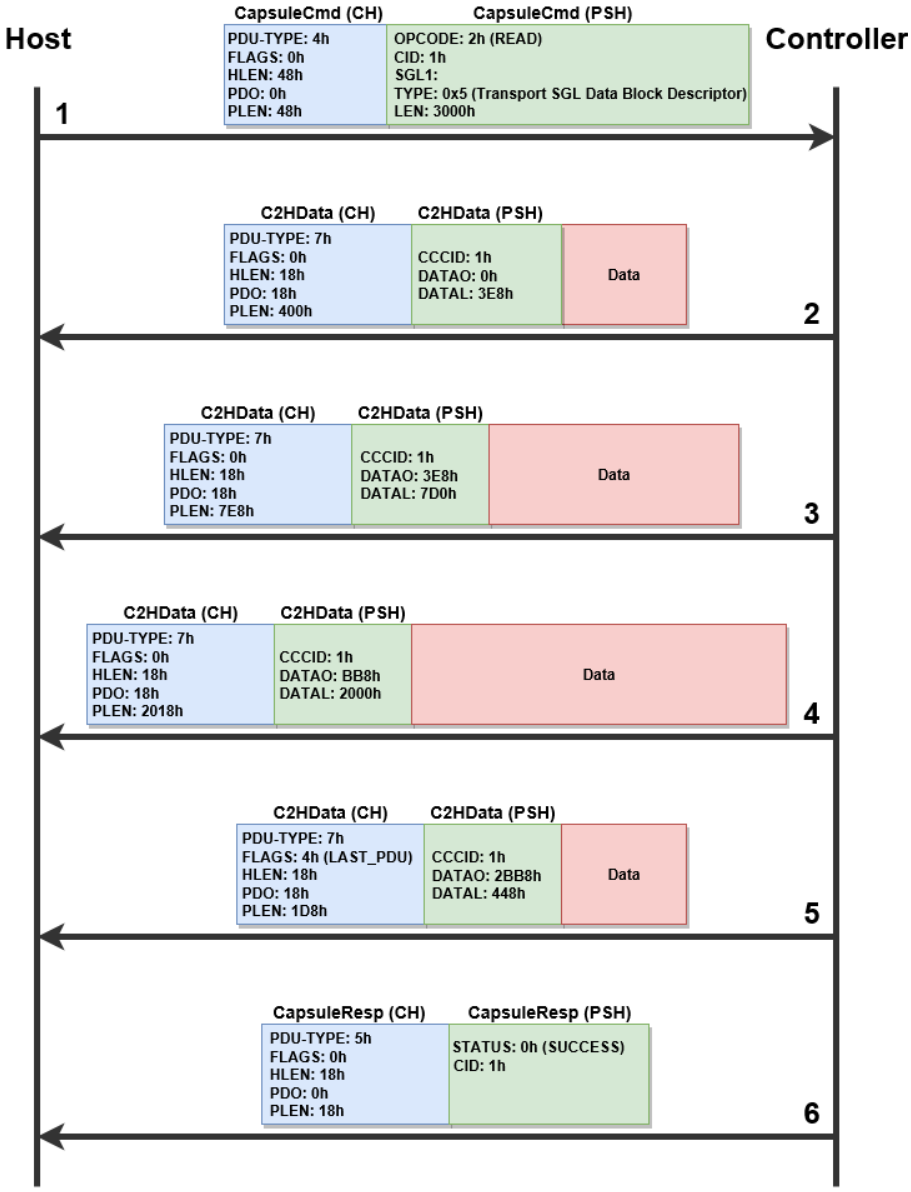
C2HData PDUs for a command shall be sent in-order for a command but may be arbitrarily interleaved with other unrelated PDUs (e.g., other C2HData PDUs transferring data for different commands).

Figure 15 illustrates a command that performs a 12,288 bytes (3000h bytes) controller to host Command Data Buffer transfer:

1. The host sends a Command Capsule PDU (CapsuleCmd) to the controller containing an SQE with a Transport SGL Data Block descriptor with a Sub Type value of 1h 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 1,000 bytes (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 2,000 bytes (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 8,192 bytes (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 1,096 bytes (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 Response Capsule PDU (CapsuleResp) to the host containing a CQE.

Figure 15: Controller to Host Data Transfer Example



3.3.2.2 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 3.3.2.1. 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 specifies the maximum number of outstanding R2T PDUs (MAXR2T) that the host 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 (refer to the Submission Queue Entry section in the NVMe Base Specification) 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, the host transfers the data requested by the controller as indicated by the R2T PDU. This data transfer may be performed using one or more H2CData PDUs. H2CData PDUs 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 a 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 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.

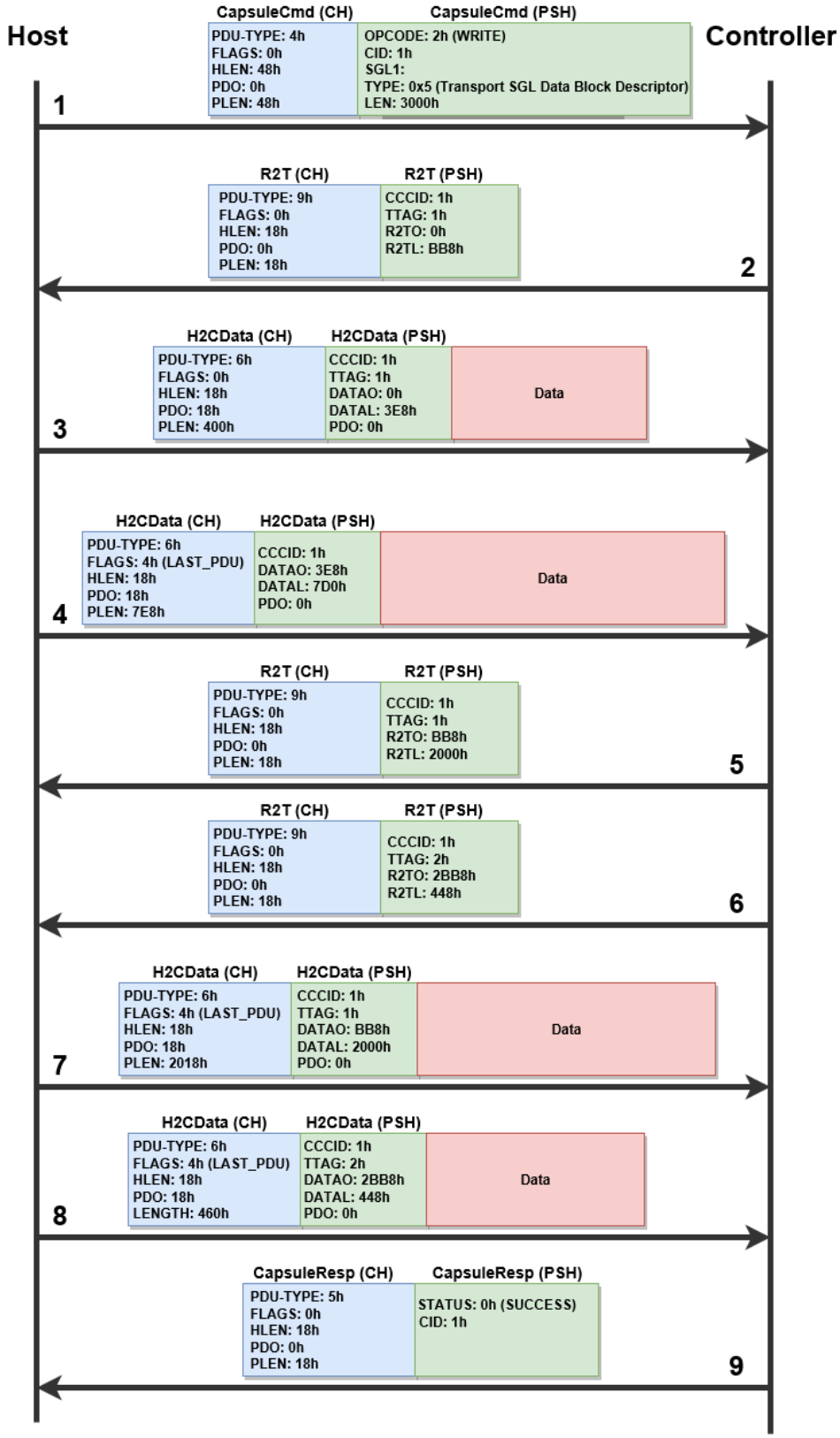
R2T PDUs associated with a specific command shall be serviced in the order received by the host. R2T PDUs associated with different commands received by a host may be serviced in any order. A controller may send an R2T PDU without waiting for a previous R2T data transfer to complete, but shall not exceed 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 16 illustrates a command that performs a 12,288 bytes (3000h bytes) host to controller Command Data Buffer transfer using R2T PDUs:

1. The host sends a Command Capsule PDU (CapsuleCmd) to the controller containing an SQE with a Transport SGL Data Block descriptor with a Sub Type 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 3,000 bytes (BB8h byte) of data with a data offset of zero;
3. When the host receives the R2T PDU, that host sends an H2CData PDU that transfers 1,000 bytes (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;
4. The host sends a subsequent H2CData PDU that transfers 2,000 bytes (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 requests 8,192 bytes (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 requests 1,096 bytes (448h bytes) if the host supports MAXR2T greater than one. The R2TO field is set to 2BB8h and the R2TL field is 448h;
7. The host sends a H2CData PDU that transfers 8,192 bytes (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 1,096 bytes (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 Capsule PDU (CapsuleResp) to the host containing a CQE.

Figure 16: Host to Controller Data Transfer Example



3.4 Keep Alive Model

The NVMe/TCP Transport requires the use of the Keep Alive feature (refer to section 7.12 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 the TCP level keep alive timeout is set to a higher value than the NVMe Keep Alive Timeout to avoid conflicting policies.

3.5 Error Handling Model

The following section defines expectations for both recoverable and non-recoverable error handling on the transport.

3.5.1 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, that controller shall:

1. stop processing any PDUs that arrive on the connection; and
2. send a C2HTermReq PDU (refer to section 3.6.2.4) with:
    a. an appropriate Fatal Error Status field (FES);
    b. additional error information in the Fatal Error Information (FEI) field if applicable; and
    c. the PDU data contents filled 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 seconds, 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 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, that host 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. the PDU data content filled 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 seconds, 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 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 a Transient Transport Error status code (refer to Figure 126 in the NVMe Base Specification).

3.5.2 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. When a host detects a data digest error in a C2HData PDU, that host 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, that controller 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.

3.6 Transport Specific Content

3.6.1 Transport Layer Security

This section describes the Transport Layer Security requirements for the TCP transport.

3.6.1.1 Transport Specific Address Subtype: TLS

The TSAS SECTYPE field defined in Figure 17 describes whether TLS is supported. TLS implementation is optional for NVMe/TCP.

Figure 17: Transport Specific Address Subtype Definition for NVMe/TCP Transport

<table>
<tr><th>Bytes</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 0h (No Security), then the host shall set up 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 NVMe-oF 1.1).</td></tr>
<tr><td>02</td><td>Transport Layer Security (TLS) version 1.3 (refer to RFC 8446) or a subsequent version. The TLS protocol negotiates the version and ciper 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 protocol versions prior to 1.2 shall not be used with NVMe/TCP (refer to section 3.1.1 of RFC 8996). All versions of SSL, the predecessor protocol to TLS, shall not be used with NVMe/TCP. NVMe/TCP implementations that are compliant with version 1.0 of the NVMe TCP Transport Specification and that support TLS shall support TLS 1.3 (refer to RFC 8446).

3.6.1.2 Mandatory and Recommended Cipher Suites

TLS for NVMe/TCP is based on pre-shared key (PSK) authentication. NVMe/TCP implementations that support TLS 1.3 shall support the TLS_AES_128_GCM_SHA256 {13h, 01h} cipher suite and should support the TLS_AES_256_GCM_SHA384 {13h, 02h} cipher suite.

Implementations shall support disabling individual cipher suites. The methods for disabling individual cipher suites are outside the scope of this specification.

For authentication and key exchange, implementations shall support (refer to section 2 of RFC 8446):

- PSK-only authentication; and
- PSK with (EC)DHE (refer to RFC 8446).

PSK with (EC)DHE protects encrypted traffic against compromise of the pre-shared key. Implementations shall support disabling PSK-only authentication. The method for disabling PSK-only authentication is outside the scope of this specification.

The DH exponentials used in an ephemeral DH exchange should not be reused (refer to section 2.12 of RFC 7296 for guidance on DH exponential reuse). Implementations shall not use a DH exponential for multiple protocols (e.g., use the same DH exponential for DH-HMAC-CHAP and TLS).

NVMe/TCP TLS 1.3 implementations shall support the ffdhe3072 group for PSK with DHE and should support the secp384r1 group (refer to section 4.2.7 of RFC 8446). Implementations shall support restricting the DH and ECDH groups offered and accepted. The method for restricting the DH and ECDH groups offered and accepted is outside the scope of this specification.

3.6.1.3 TLS PSK and PSK Identity Derivation

This section uses the following terminology:

- **Configured PSK:** the PSK provided via an administrative interface of the NVMe/TCP entity. The method for configuring a PSK is outside the scope of this specification;
- **Retained PSK:** the PSK stored by the NVMe/TCP entity for use with TLS;
- **Generated PSK:** the PSK generated by an NVMe authentication protocol (e.g., DH-HMAC-CHAP, refer to the DH-HMAC-CHAP Protocol section in the NVMe Base specification); and
- **TLS PSK:** the PSK used by the TLS protocol.

The configured PSK is configured on both involved entities (i.e., host and NVM subsystem). NVM subsystems should support the ability to use a different configured PSK with each host. Hosts should support the ability to use a different configured PSK with each NVM subsystem.

The retained PSK is derived from the configured PSK (refer to section 3.6.1.4). The configured PSK shall be destroyed as soon as the retained PSK is generated and stored. Each NVMe/TCP entity shall support:

- transforming the configured PSK into a retained PSK before it is stored by the NVMe/TCP entity for repeated use with another NVMe/TCP entity; and
- using the configured PSK as a retained PSK.

The method to derive a retained PSK from a configured PSK shall be using the HKDF-Extract and HKDF-Expand-Label operations (refer to RFC 5869 and RFC 8446):

1. PRK = HKDF-Extract(0, Configured PSK); and
2. Retained PSK = HKDF-Expand-Label(PRK, “HostNQN”, NQN_h, Length(Configured PSK)),

where NQN_h in the NQN of the host. The hash function used with HKDF shall be the one specified in the PSK interchange format (refer to section 3.6.1.4). This transform requires that the NVM subsystem knows the NQN of the host with which the configured PSK is used.

The retained PSK or the generated PSK is used to derive the TLS PSK and the related PSK identity that are associated with a TLS 1.3 cipher suite hash function. The result is a {TLS PSK, PSK Identity, Hash} tuple for use with TLS 1.3.

In TLS 1.3 each PSK is identified by the client using a PSK identity. Each PSK is also associated with one hash function that shall be the same as the hash function of the selected cipher suite. For example, the cipher suites TLS_AES_128_GCM_SHA256 and TLS_AES_256_GCM_SHA384 use the SHA-256 and SHA-384 hash functions respectively. A TLS client that offers both cipher suites shall offer two PSKs with different identities, different hash functions, and different key material.

A TLS 1.3 client implementation that only supports sending a single PSK identity during connection setup may be required to connect multiple times in order to negotiate cipher suites with different hash functions.

Some TLS 1.3 server implementations are only able to validate one PSK at a time in the order that they are listed in the TLS 1.3 pre_shared_key extension. As a result, TLS 1.3 client implementations should order their offered PSKs from most desirable to least desirable.

The TLS 1.3 PSK identity used with NVMe/TCP is generated from the NQNs of the host and the controller. The PSK identity is a UTF-8 string constructed as an in-order concatenation of the following elements.

1. A 4-character format specifier “NVMe” in utf-8 encoding;
2. A one-character TLS protocol version indicator:
   - ‘0’ (i.e., U+0030h) indicates TLS 1.3;
3. A one-character PSK type indicator, specifying the used PSK:
   - ‘R’ (i.e., U+0052h) indicates the retained PSK;
   - ‘G’ (i.e., U+0047h) indicates the generated PSK;
4. A two-characters hash specifier, specifying the hash function of the cipher suite associated with this PSK identity:
   - “01” indicates SHA-256 (e.g., for the TLS_AES_128_GCM_SHA256 cipher suite);
   - “02” indicates SHA-384 (e.g., for the TLS_AES_256_GCM_SHA384 cipher suite);
5. A space character (i.e., U+0020h);
6. The NQN of the host (i.e., NQN_h);
7. A space character (i.e., U+0020h);
8. The NQN of the controller (i.e., NQN_c); and
9. A null character (i.e., U+0000h).

For example, host NQN “`nqn.2014-08.org.nvmexpress:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6`” and subsystem NQN “`nqn.2014-08.org.nvmexpress:uuid:36ebf5a9-1df9-47b3-a6d0-e9ba32e428a2`” with the SHA-256 hash function and the retained PSK generate the following PSK identity:

“`NVMe0R01 nqn.2014-08.org.nvmexpress:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6 nqn.2014-08.org.nvmexpress:uuid:36ebf5a9-1df9-47b3-a6d0-e9ba32e428a2`”

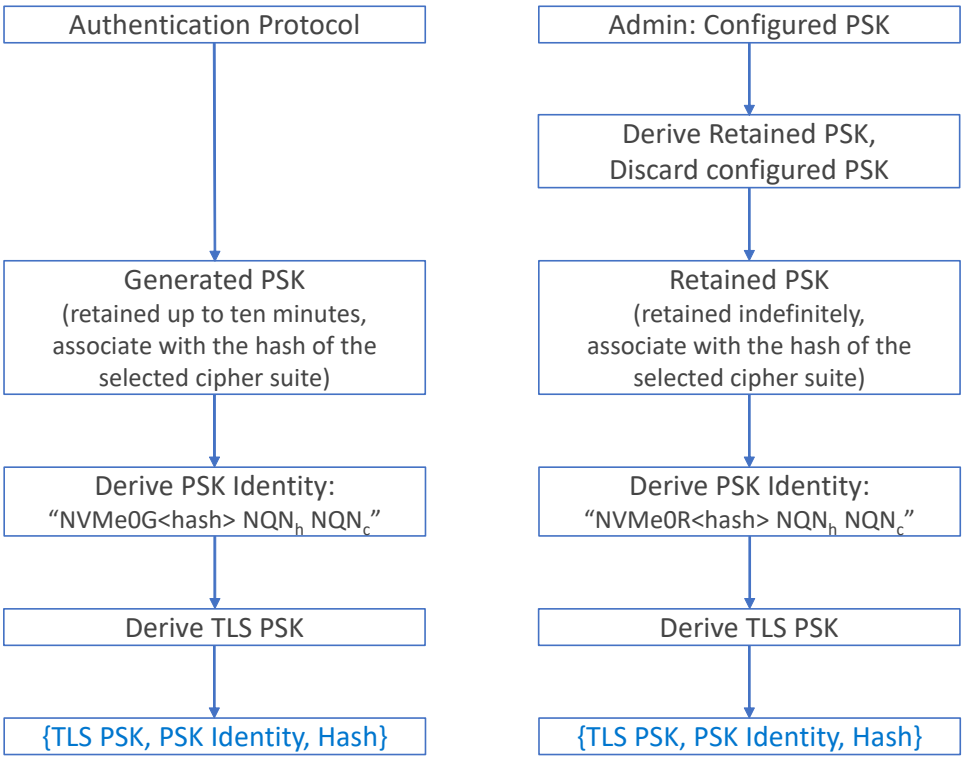
The TLS PSK shall be derived as follows from an input PSK (i.e., either a retained PSK or a generated PSK) and a PSK identity using the HKDF-Extract and HKDF-Expand-Label operations (refer to RFC 5869 and RFC 8446) where the hash function is the one specified by the hash specifier of the PSK identity:

1. PRK = HKDF-Extract(0, Input PSK); and
2. TLS PSK = HKDF-Expand-Label(PRK, “nvme-tls-psk”, PskIdentity, L),

where PskIdentity is the PSK identity and L is the output size in bytes of the hash function (i.e., 32 for SHA-256 and 48 for SHA-384).

The full process to derive the {TLS PSK, PSK Identity, Hash} tuple is shown in Figure 18 for both configured PSK and generated PSK. This process is performed for each supported TLS 1.3 cipher suite hash function.

Figure 18: {TLS PSK, TLS Identity, Hash} Tuple Derivation



3.6.1.4 PSK Reuse

A retained PSK is used between a host and an NVM subsystem to set up TLS secure channels for the Admin Queue and all I/O Queues of each controller associated with that host.

A generated PSK used to set up an Admin Queue TLS secure channel after having performed an authentication transaction between a host and a controller (refer to section 6.5.9) may be reused within the lifetime of the generated PSK to set up additional TLS secure channels for I/O Queues on the same controller. Subsequent creation of an I/O Queue on the same controller requires a different PSK, which may be generated by performing another authentication transaction on that I/O Queue. The resulting generated PSK may be reused within the lifetime of the generated PSK to set up additional TLS secure channels for I/O Queues on the same controller.

A generated PSK shall be discarded when its lifetime expires. The lifetime of a generated PSK shall be ten minutes, unless otherwise configured.

If a retained PSK exists for an Admin Queue or a I/O Queue associated with a generated PSK, then that generated PSK shall not be reused.

3.6.1.5 PSK Interchange Format

In order to facilitate provisioning, management, and interchange (e.g., copy & paste in an administrative configuration tool) of PSKs, all NVMe-oF entities shall support the following ASCII representation of configured PSKs:

```
NVMeTLSkey-1:xx:<Base64 encoded string>:
```

Where:

1. ”NVMeTLSkey-1” indicates this is a version 1 representation of a TLS PSK;
2. ‘:’ is used both as a separator and a terminator;
3. xx indicates the hash function to be used to transform the configured PSK in a retained PSK (refer to section 3.6.1.3), encoded as follows:
   - the two ASCII characters “00” indicate no transform (i.e., the configured PSK is used as a retained PSK);
   - the two ASCII characters “01” indicate SHA-256; and
   - the two ASCII characters “02” indicate SHA-384;

   and
4. The Base64 (refer to RFC 4648) string encodes the configured PSK (32 or 48 bytes binary) followed by the CRC-32 (refer to RFC 1952) of the configured (4 bytes binary).

As an example, the 32-byte configured PSK:

```
5512DBB6_737D0106_F65975B7_73DFB011_FFC344BC_F442E2DD_6D8BC487_0B5D5B03
```

is represented as:

```
NVMeTLSkey-1:01:VRLbtnN9AQb2WXW3c9+wEf/DRLz0QuLdbYvEhwtdWwNf9LrZ:
```

when requested to be transformed to a retained PSK with the SHA-256 hash.

When provided with a configured PSK in this format, NVMe-oF entities shall verify the validity of the provided PSK by computing the CRC-32 value of the PSK and checking the computed value with the provided value. If they do not match, then the PSK shall not be used.

3.6.1.6 TLS Implementations and Use Requirements

NVMe/TCP host and subsystem implementations shall not send or use 0-RTT data as it is subject to replay attacks (refer to Appendix E.5 of RFC 8446).

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.

NVMe/TCP implementations that support TLS shall support disabling the following parameters, using a method outside the scope of this specification:

- each individual cipher suite;
- PSK-only authentication;
- each individual DH group;
- each individual ECDH group.

3.6.2 NVMe/TCP PDUs

This section describes the format of NVME/TCP PDUs.

3.6.2.1 PDU Common Header (CH)

Figure 19: PDU Common Header (CH)

Bytes	Description
00	**PDU-Type:** Specifies the type of PDU. This value is also referred to as the PDU opcode. Refer to Figure 10 for PDU opcodes.
01	**FLAGS:** PDU-TYPE specific flags
02	**Header Length (HLEN):** Length of PDU Header (not including the HDGST) 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.

3.6.2.2 Initialize Connection Request PDU (ICReq)

Figure 20: Initialize Connection Request PDU (ICReq)

<table>
<tr><th>Bytes</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="4">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 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>Definition</th></tr>
<tr><td>7:2</td><td>Reserved</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>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>
</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>

3.6.2.3 Initialize Connection Response PDU (ICResp)

Figure 21: Initialize Connection Response PDU (ICResp)

<table>
<tr><th>Bytes</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 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 that transfer data in addiion to the PDU Header (refer to section 2). This is a 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: Controller PDU header and data digest enable options.
<table>
<tr><th>Bits</th><th>Definition</th></tr>
<tr><td>7:2</td><td>Reserved</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>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>
</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 H2CData PDU in bytes. This value is a multiple of dwords and should be no less than 4,096.</td></tr>
<tr><td>127:16</td><td>Reserved</td></tr>
</table>

3.6.2.4 Host to Controller Terminate Connection Request PDU (H2CTermReq)

Figure 22: Host to Controller Terminate Connection Request PDU (H2CTermReq)

<table>
<tr><th>Bytes</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>00h</td><td>Reserved</td></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: An HDGST error was detected by the host.</td></tr>
<tr><td>04h</td><td>Data Transfer Out of Range: A C2HData PDU 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>07h to FFFFh</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>00</td><td>Reserved</td></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 has an error. If multiple errors exist, then this field indicates the lowest offset that has an error.</td></tr>
<tr><td>02h</td><td>Reserved</td></tr>
<tr><td>03h</td><td>PDU Header Digest: This field indicates the HDGST that was received by the host caused a header digest verification error.</td></tr>
<tr><td>04h to 05h</td><td>Reserved</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 has an error. If multiple errors exist, then this field indicates the lowest offset that has an error.</td></tr>
<tr><td>07h to FFFFh</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>

3.6.2.5 Controller to Host Terminate Connection Request PDU (C2HTermReq)

Figure 23: Controller to Host Terminate Connection Request PDU (C2HTermReq)

<table>
<tr><th>Bytes</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 information.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00h</td><td>Reserved</td></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: An HDGST error was detected by the controller.</td></tr>
<tr><td>04h</td><td>Data Transfer Out of Range: An H2CData PDU 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: An 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>07h to FFFFh</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>00h</td><td>Reserved</td></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 has an error. If multiple errors exist, then this field indicates the lowest offset that has an error.</td></tr>
<tr><td>02h</td><td>Reserved</td></tr>
<tr><td>03h</td><td>PDU Header Digest: This field indicates the HDGST that was received by the controller caused a header digest verification error.</td></tr>
<tr><td>04h to 05h</td><td>Reserved</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 has an error. If multiple errors exist, then this field indicates the lowest offset that has an error.</td></tr>
<tr><td>07h to FFFFh</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 controller while the fatal error was detected.</td></tr>
</table>

3.6.2.6 Command Capsule PDU (CapsuleCmd)

Figure 24: Command Capsule PDU (CapsuleCmd)

<table>
<tr><th>Bytes</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>7:2</td><td>Reserved</td></tr>
<tr><td>1</td><td>DDGSTF: If set to ‘1’, then a valid DDGST value follows the PDU Data.</td></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid HDGST value follows the PDU Header.</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 shall be a multiple of the data alignment specified by the CPDA field in the ICResp PDU (refer to section 3.6.2.3) that was previously sent by the controller on this TCP connection.</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): Command Capsule SQE.</td></tr>
<tr><td>75:72</td><td>HDGST</td><td>HDGST: If HDGSTF is set to ‘1’ in the FLAGS field, this field is present and contains the header digest (refer to section 3.3.1.1).</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 required to achieve the alignment specified by CPDA (refer to section 3.6.2.3).</td></tr>
<tr><td>M - 1:N</td><td>DATA</td><td>NVMe-oF In-Capsule Data (CCICD): This field contains 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 to ‘1’ in the FLAGS field, and the CCICD field is present, then this field contains the data digest (refer to section 3.3.1.1) of the CCICD field (in-capsule data).</td></tr>
</table>

3.6.2.7 Response Capsule PDU (CapsuleResp)

Figure 25: Response Capsule PDU (CapsuleResp)

<table>
<tr><th>Bytes</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>7:1</td><td>Reserved</td></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid HDGST value follows the PDU Header.</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 (refer to section 3.3.1.1).</td></tr>
</table>

3.6.2.8 Host To Controller Data Transfer PDU (H2CData)

Figure 26: Host To Controller Data Transfer PDU (H2CData)

<table>
<tr><th>Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="5">CH</td><td>PDU-Type: 06h</td></tr>
<tr><td>01</td><td>FLAGS:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:3</td><td>Reserved</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>1</td><td>DDGSTF: If set to ‘1’, then a valid DDGST value follows the PDU Data.</td></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid HDGST value follows the PDU Header.</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 shall be a multiple of the data alignment specified by the CPDA field in the ICResp PDU (refer to section ~~3.1.2~~3.6.2.3) that was previously sent by the controller on this TCP connection.</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="4">PSH</td><td>Command Capsule CID (CCCID): This field contains the SQE.CID value of the Command Capsule PDU 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 host.</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 (refer to section 3.3.1.1).</td></tr>
<tr><td>N - 1:N</td><td>PAD</td><td>PAD: The length of this field shall be the necessary number of bytes required to achieve the alignment specified by CPDA (refer to section 3.6.2.3).</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 (refer to section 3.3.1.1).</td></tr>
</table>

3.6.2.9 Controller To Host Data Transfer PDU (C2HData)

Figure 27: Controller To Host Data Transfer PDU (C2HData)

<table>
<tr><th>Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="9">CH</td><td>PDU-Type: 07h</td></tr>
<tr><td>01</td><td>FLAGS:
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:4</td><td>Reserved</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 information and that no Response Capsule PDU is sent by the Controller.</td></tr>
<tr><td>2</td><td>LAST_PDU: If set to ‘1’, indicates the PDU is the last C2HData PDU sent in response to a Command Capsule PDU.</td></tr>
<tr><td>1</td><td>DDGSTF: If set to ‘1’, then a valid DDGST value follows the PDU Data</td></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid HDGST value follows the PDU Header.</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 shall be a multiple of the data alignment specified by the HPDA field in the ICReq PDU (refer to section 3.6.1.1) that was previously sent by the host on this TCP connection.</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 PDU 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 (refer to section 3.3.1.1).</td></tr>
<tr><td>N - 1:N</td><td>PAD</td><td>PAD: If HPDA is set to a non-zero value, then this field is 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 (refer to section 3.3.1.1) of the PDU-Data field.</td></tr>
</table>

3.6.2.10 Ready to Transfer PDU (R2T)

Figure 28: Ready to Transfer PDU (R2T)

<table>
<tr><th>Bytes</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>7:1</td><td>Reserved</td></tr>
<tr><td>0</td><td>HDGSTF: If set to ‘1’, then a valid Header Digest value follows the PDU Header.</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 PDU 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’s 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>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 (refer to section 3.3.1.1).</td></tr>
</table>