NVM Express®

TCP Transport Specification

Revision 1.1

August 5th, 2024

Please send comments to info@nvmexpress.org

NVM Express® TCP Transport Specification is available for download at https://nvmexpress.org. The NVMe TCP Transport Specification, Revision 1.1 incorporates NVMe TCP Transport Specification, Revision 1.0 and ECN001, ECN102, ECN105, ECN108, ECN110, ECN115, ECN116, ECN117, ECN118, ECN120, TP4129, TP6036, TP8010a, TP8012, TP8018, TP8025, and TP8026 (refer to https://nvmexpress.org/changes-in-nvme-revision-2-1 for details).

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: **"© 2008 to 2024 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.
PCIe® is a registered trademark of PCI-SIG.

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

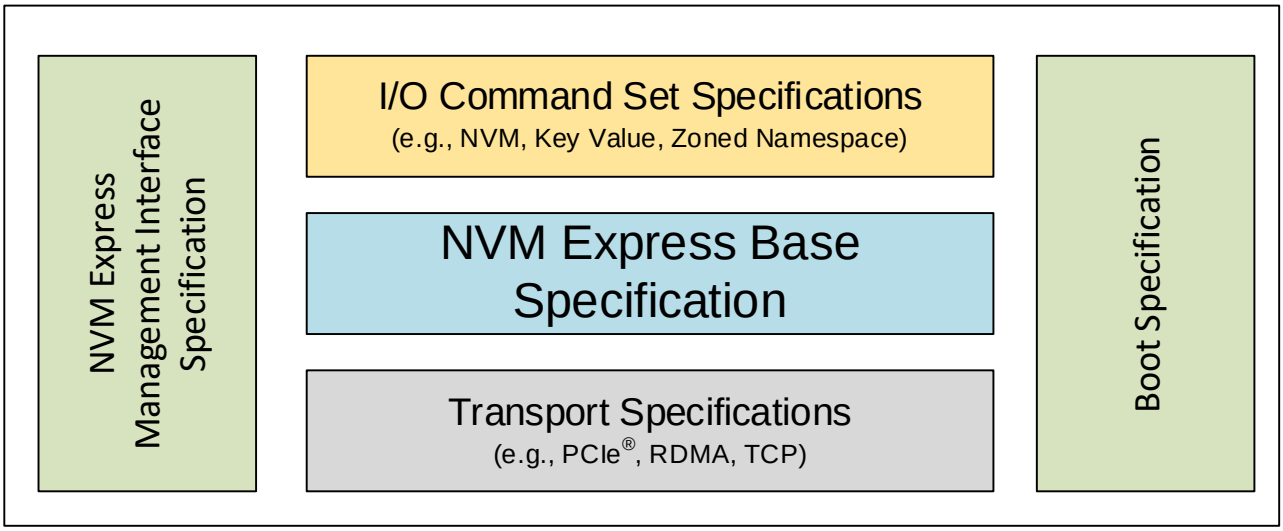
NVM Express® (NVMe®) Base Specification defines an interface for host software to communicate a with non-volatile memory subsystem (NVM subsystem) over a variety of memory-based transports and message-based transports.

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

1.2 Scope

Figure 1 shows the relationship of the NVM Express® TCP Transport Specification to other specifications within the NVMe Family of Specifications.

Figure 1: NVMe Family of Specifications



This specification supplements the NVM Express 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 NVM Express 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. NVM Express Base specification
3. NVMe transport specifications
4. NVMe I/O command set specifications
5. NVM Express Management Interface Specification
6. NVM Express® Boot Specification

1.3 Conventions

This specification conforms to the Conventions section, Keywords section and the Byte, Word, and Dword Relationships section of the NVM Express 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 Control 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

CNSSP 15 “USE OF PUBLIC STANDARDS FOR SECURE INFORMATION SHARING”, 20 October 2016. Available from https://www.cnss.gov/CNSS/issuances/Policies.cfm.

NVM Express® Base Specification, Revision 2.1. Available from https://www.nvmexpress.org.

NVM Express over Fabrics Specification, Revision 1.1a. Available from https://www.nvmexpress.org.

> Note: the obsoleted NVM Express over Fabrics Specification is included only for its requirements for TLS version 1.2, refer to section 3.6.1.

RFC 1952, P. Deutsch, “GZIP file format specification version 4.3”, May 1996. Available from https://www.rfc-editor.org/rfc/rfc1952.

RFC 4648, S. Josefsson, “The Base16, Base32, and Base64 Data Encodings”, October 2006. Available from https://www.rfc-editor.org/rfc/rfc4648.

RFC 5869, H. Krawczyk, P. Eronen, “HMAC-based Extract-and-Expand Key Derivation Functions (HKDF)”, May 2010. Available from https://www.rfc-editor.org/rfc/rfc5869.

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.rfc-editor.org/rfc/rfc7296.

RFC 7301, S. Friedl, A. Popov, A. Langley, E. Stephan, “Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension”, July 2014. Available from https://www.rfc-editor.org/rfc/rfc7301.

RFC 8446, E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3”, August 2018. Available from https://www.rfc-editor.org/rfc/rfc8446.

RFC 8996, K. Moriarty, S. Farrell, “Deprecating TLS 1.0 and TLS 1.1”, March 2021. Available from https://www.rfc-editor.org/rfc/rfc8996.

1.6 Acronyms

Figure 2: NVMe/TCP Acronym Descriptions

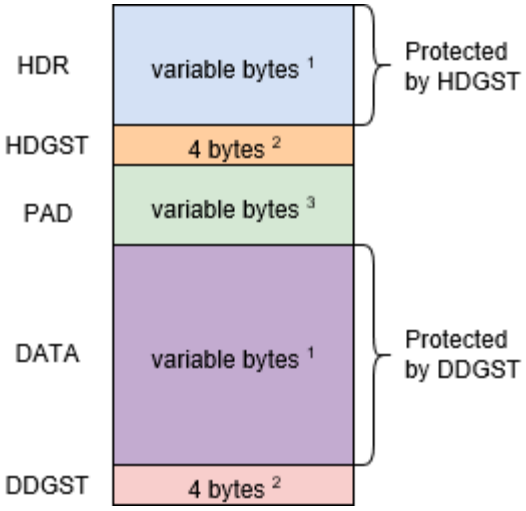
Term	Definition
CH	PDU Common Header
CPDA	Controller PDU Data Alignment
C2H	Controller to Host Direction
C2HtermReq	Controller to Host Terminate Connection Request
DATA	PDU Data
DATAL	H2Cdata and C2Hdata PDU Data Length
DATAO	H2Cdata and C2Hdata PDU Data Offset
DDGST	PDU Data Digest
HDGST	PDU Header Digest
HDR	PDU Header
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
KDReq	Kickstart Discovery Request
KDResp	Kickstart Discovery 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 3.6.1.7).

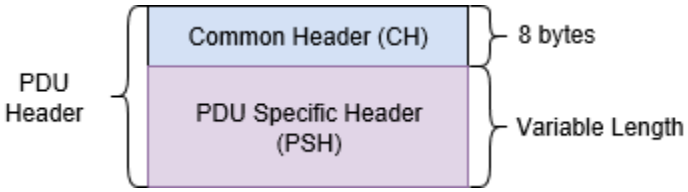
Figure 3: NVMe/TCP PDU Structure



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

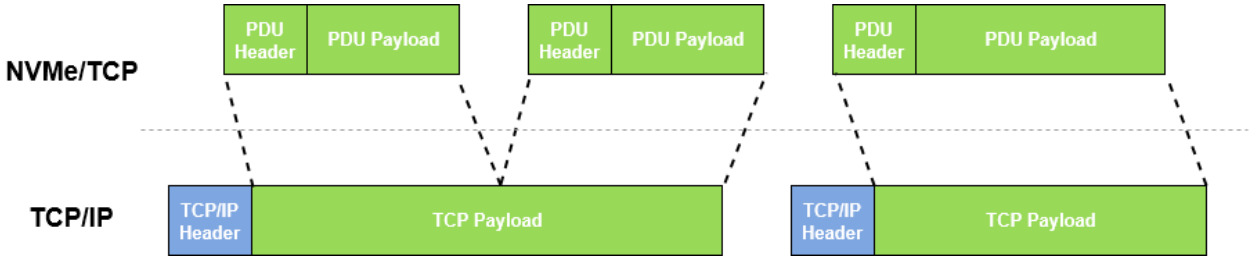
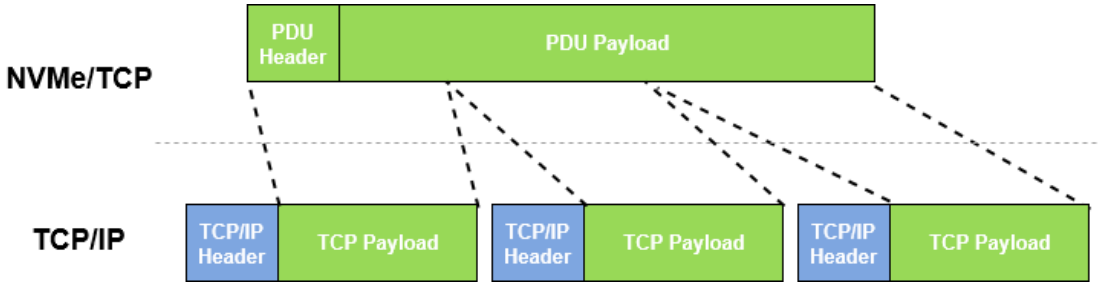


Figure 6: NVMe/TCP PDU Spanning across TCP/IP Packets



The NVMe/TCP definition conforms to the byte, word, and dword relationships defined in the Conventions section of the NVM Express 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.

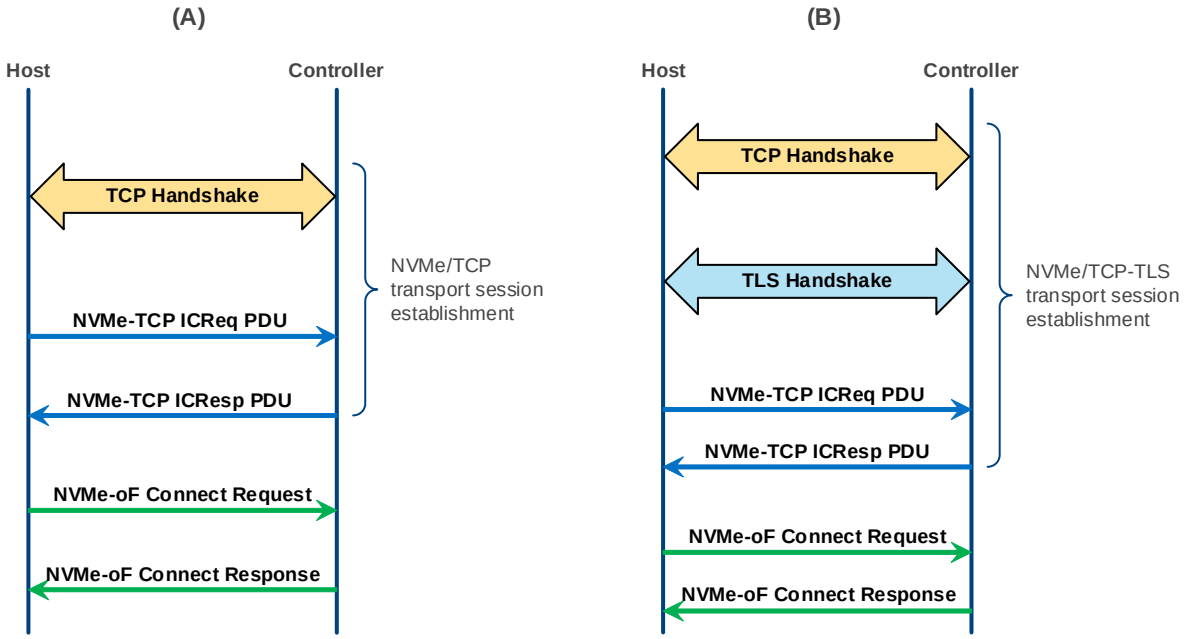
3 Transport Binding

3.1 Setup & Initialization

Figure 7 illustrates the process used to establish an NVMe/TCP connection, with or without TLS. To establish an NVMe/TCP connection without TLS, the first step is to establish a TCP connection between a host and a controller, as shown in Figure 7 (A). To establish an NVMe/TCP connection with TLS (i.e., an NVMe/TCP-TLS connection), the first step is to establish a TCP connection between a host and a controller followed by a TLS handshake, as shown in Figure 7 (B). 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 or TLS 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. If the Kickstart Discovery Connection (KDCONN) bit is cleared to ‘0’ in the ICReq PDU, then a non-kickstart discovery NVMe/TCP connection is established, and the first capsule exchange is the NVMe-oF Connect request/response sequence. The NVMe/TCP PDUs listed in Figure 10 except for KDReq (refer to section 3.6.2.11) and KDResp (refer to section 3.6.2.12) may be exchanged over non-kickstart discovery NVMe/TCP connections. If the KDCONN bit is set to ‘1’ in the ICReq PDU, then a kickstart discovery (refer to the Kickstart Discovery Pull Registration Requests section in the NVM Express Base Specification) NVMe/TCP connection is established, and only KDReq NVMe/TCP PDUs and KDResp NVMe/TCP PDUs shall be exchanged over that kickstart discovery NVMe/TCP connection.

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 Page 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 Page 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 NVM Express 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 NVM Express 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 Page 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 Page 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 Page 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 NVM Express 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 for non-kickstart discovery NVMe/TCP connections (refer to the Kickstart Discovery Pull Registration Requests section in the NVM Express Base Specification). 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.

Host and controller PDU Data alignment is not used for kickstart discovery NVMe/TCP connections.

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

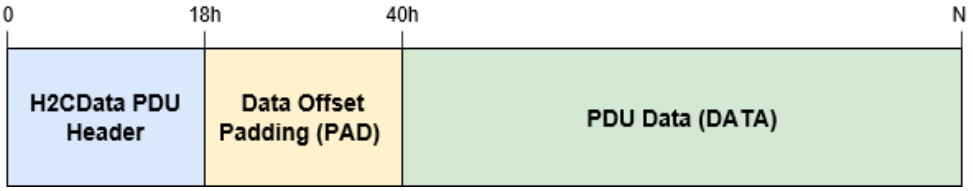
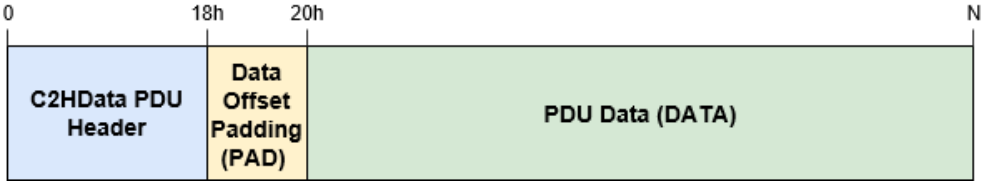


Figure 9 shows an example of a C2HData PDU where the HPDA field (refer to section 3.6.2.2) was set to 03h by the host in the ICReq when the connection was established. The C2HData PDU Header size 24 bytes and header digest is disabled. An alignment of 16 bytes is required, thus the controller inserts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

Figure 9: Example of 16B PDU DATA Alignment in C2HData PDU



The TCP transport defines NVMe/TCP PDU types that are summarized in Figure 10 and defined in section 3.6.1.7. 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 that PDU 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

Figure 10: NVMe/TCP PDU Types

PDU Name	Opcode by field		Combined Opcode [2]	Section	PDU Description
	Function (07:01)	PDU Direction [1] (00)			
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 a Fabrics Command Capsule
CapsuleResp	0000010b	1b	05h	3.6.2.7	**Response Capsule:** A PDU sent from a controller to a host to transfer a 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
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
KDReq	0000101b	0b	0Ah	3.6.2.11	**Kickstart Discovery Request:** A PDU sent from a DDC to a CDC to request a pull registration and communicate connection parameters to be used during the subsequent pull registration (refer to the Kickstart Discovery Pull Registration Requests section in the NVM Express Base Specification).
KDResp	0000101b	1b	0Bh	3.6.2.12	**Kickstart Discovery Response:** A PDU sent from a CDC to a DDC to accept a pull registration request and connection parameters (refer to the Kickstart Discovery Pull Registration Requests section in the NVM Express Base Specification).

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 the ICDOFF field in the Identify Controller data structure to control the in-capsule data offset, thus the ICDOFF field 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 bit in the ICReq PDU. The controller may accept the use of a header digest by setting the HDGST_ENABLE bit to '1' in the ICResp PDU. The controller may reject the use of a header digest by clearing the HDGST_ENABLE bit to '0' in the ICResp PDU. The PDU Header Digest is enabled if HDGST_ENABLE bit is set to '1' in both the ICReq PDU and the ICResp PDU. If the PDU Header Digest is enabled, then all the subsequent PDUs transferred in this connection except a H2CTermReq PDU and a C2HTermReq PDU shall contain a valid HDGST field and the HDGSTF bit, if defined, shall be 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 the HDGSTF bit, if defined, shall be cleared to '0' in the PDU Header FLAGS field.

If the PDU Header Digest is enabled and a subsequent PDU transferred in this connection other than a H2CTermReq PDU or a C2HTermReq PDU has the HDGSTF bit, if defined, cleared to '0', then the receiver shall treat that PDU as if a fatal transport error (refer to section 3.5.1) has occurred. If the PDU Header Digest is disabled and a subsequent PDU is received with the HDGSTF bit, if defined, set to '1', then the receiver shall treat that PDU as if a fatal transport error has occurred.

The host requests the use of a data digest by setting the DDGST_ENABLE bit in the ICReq PDU. The controller may accept the use of a data digest by setting the DDGST_ENABLE bit to '1' in the ICResp PDU. The controller may reject the use of a data digest by clearing the DDGST_ENABLE bit to '0' in the ICResp PDU. The PDU Data Digest is enabled if the DDGST_ENABLE bit is set to '1' in both the ICReq PDU and the ICResp PDU. 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 valid DDGST field and the DDGSTF bit, if defined, shall be 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 the DDGSTF bit, if defined, shall be 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 the PDU Data Digest is enabled and a Command Capsule PDU containing in-capsule data, a H2CData PDU, or a C2HData PDU transferred in this connection has the DDGSTF bit, if defined, cleared to '0', then the receiver shall treat that PDU as if a fatal transport error (refer to section 3.5.1) has occurred. If the PDU

Data Digest is disabled and any of these PDUs are received with the DDGSTF bit, if defined, set to '1', then the receiver shall treat that PDU as if a fatal transport error has occurred.

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

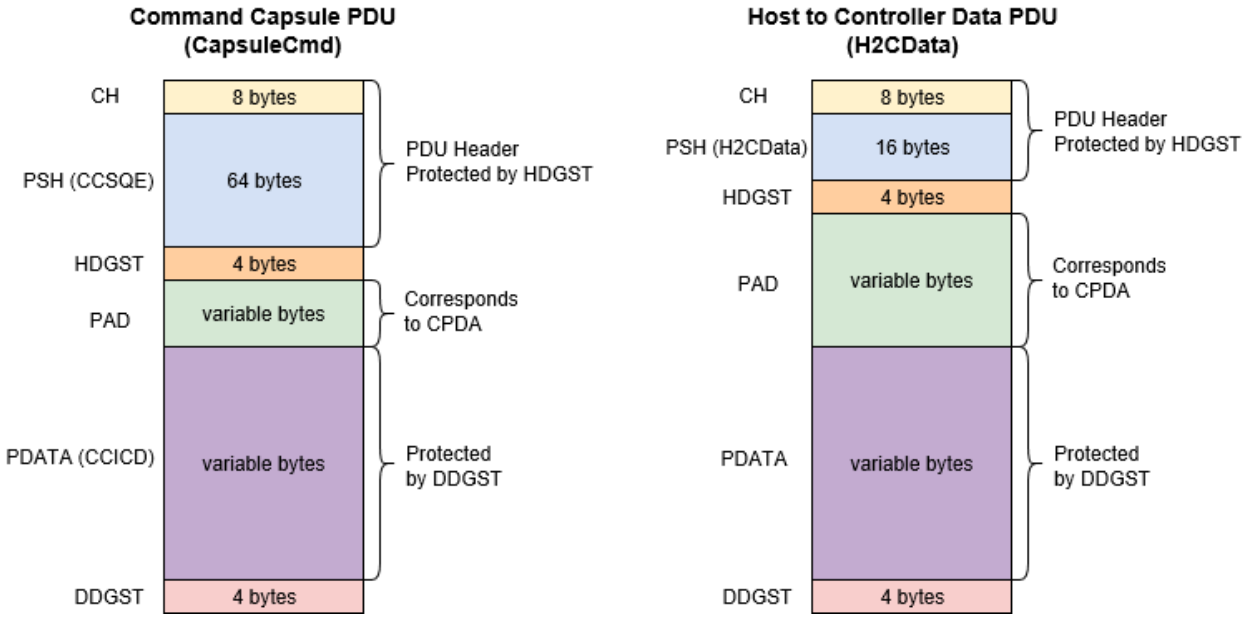
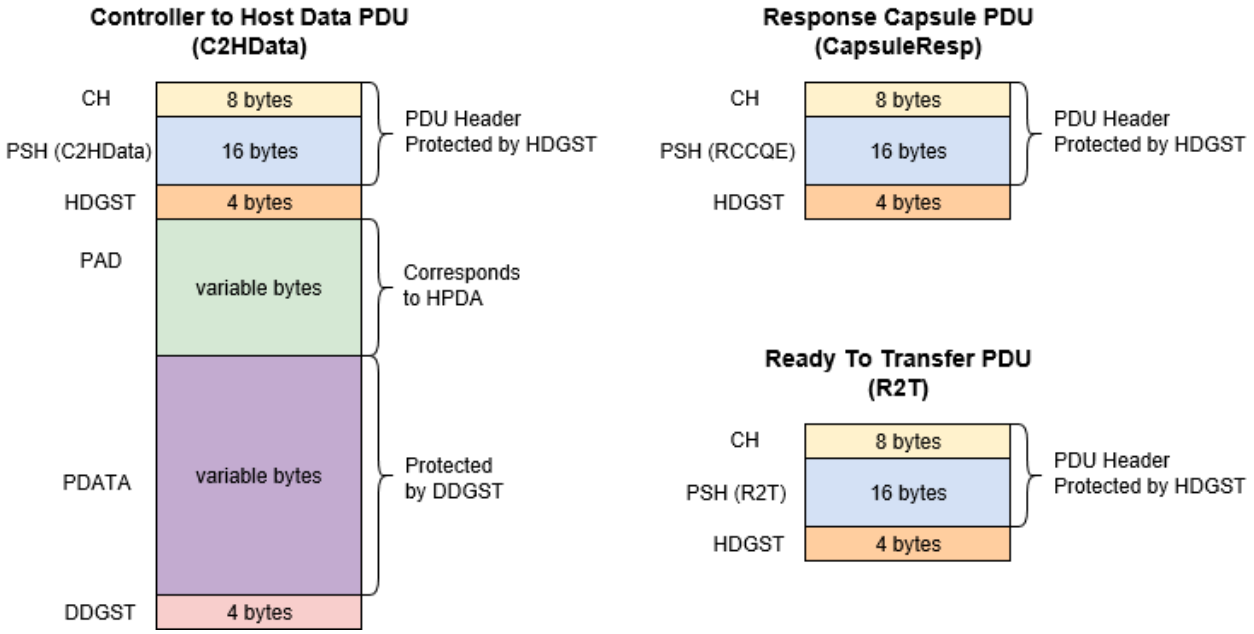


Figure 13: Controller to Host NVMe/TCP PDU Digests



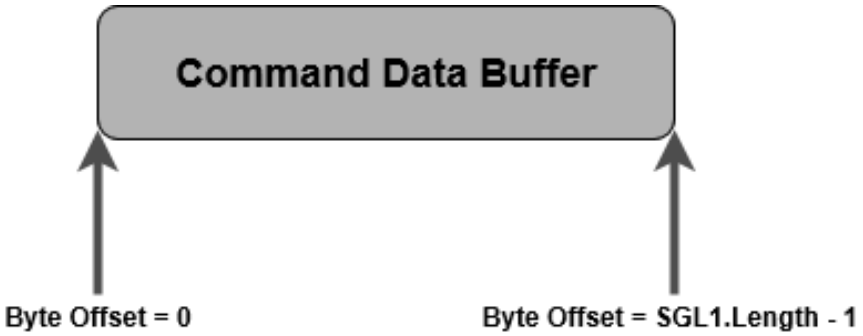
3.3.2 Command Data Buffers and SGLs

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. The Command Data Buffer 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 field value of 1h specifies the use of in-capsule data. A Transport SGL Data block descriptor with an SGL Descriptor Sub Type field 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 bit 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 bit 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 bit 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 bit shall be cleared to ‘0’ in all C2HData PDUs that are not the last C2HData PDU for a command. The SUCCESS bit 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 bit set to ‘1’ and the LAST_PDU bit 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 bit 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.

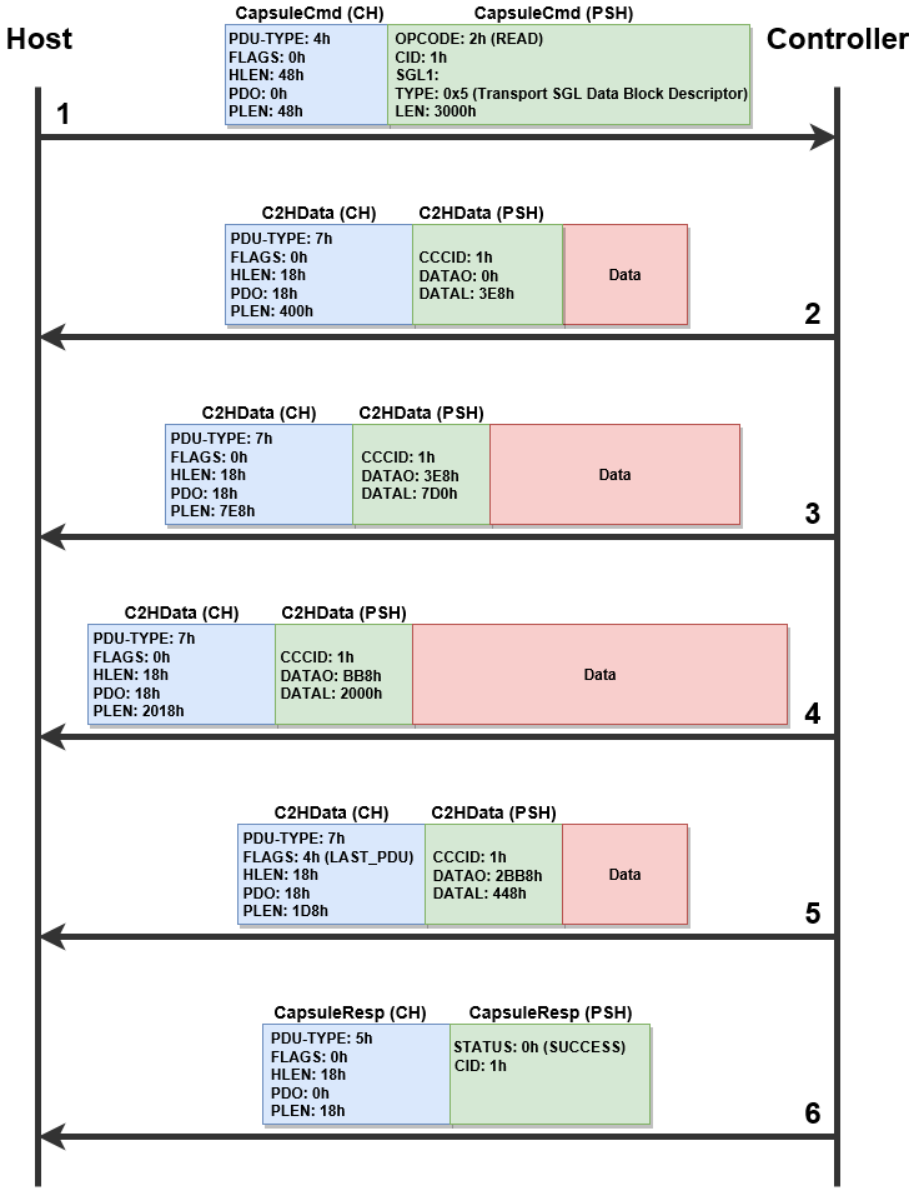
C2HData PDUs for a command shall be sent in-order for a command but may be arbitrarily interleaved with other unrelated PDUs (e.g., other C2HData PDUs transferring data for different commands).

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 SGL Descriptor Sub Type field 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 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) bit 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 bit 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 bit 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 bit 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. Command Data Buffer transfers from a

host to a controller are performed 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 NVM Express Base Specification) with which that R2T PDU is associated, a transfer tag in the TTAG field, a data offset in the R2TO field from the beginning of the Command Data Buffer, and the transfer data length in the R2TL field. 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 R2TO field of the R2T PDU and transfer data contiguously until the data transfer length specified by the R2TL field of the R2T PDU is reached (i.e., the DATAO field in the first H2CData PDU is equal to the value of the R2TO field 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 as indicated by the MAXH2CDATA field.

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 the value of the R2TO field to the sum of the R2TO field and the R2TL field 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 bit 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 bit 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 value in the TTAG field 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 bit cleared to ‘0’ after reception of a H2CData PDU with the LAST_PDU bit 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

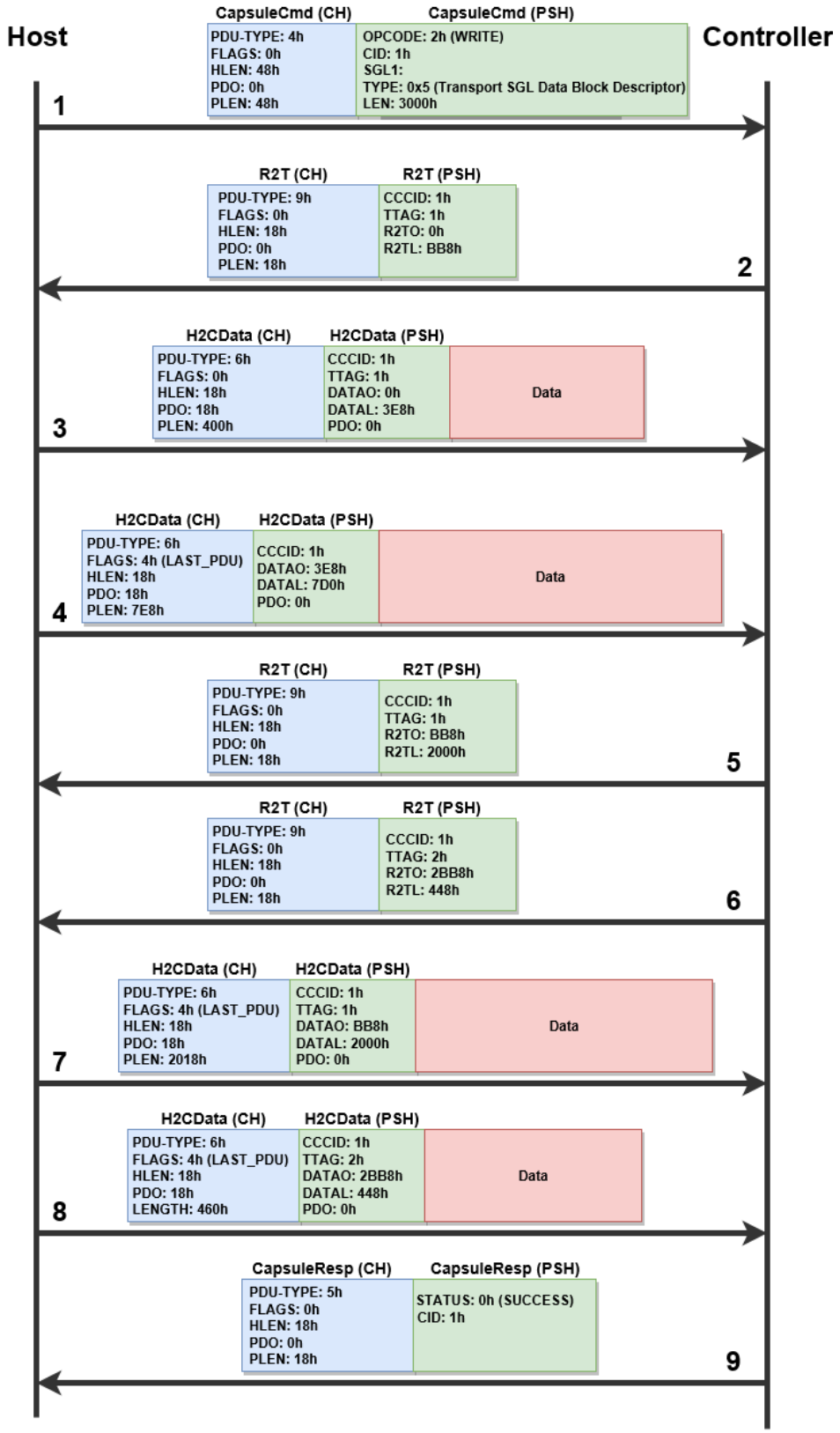
may send an R2T PDU without waiting for a previous R2T data transfer to complete, but shall not exceed the maximum number of outstanding R2T PDUs per command supported by the host (refer to the MAXR2T field).

Reception of an R2T PDU that results in the number of outstanding R2T PDUs for a command exceeding the maximum value specified by the MAXR2T field in the ICReq PDU 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 SGL Descriptor Sub Type field 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) bit 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 bit 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 a value in the MAXR2T field greater than 1h. 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 bit 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 bit 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 Timer feature (refer to the Keep Alive section in the NVM Express 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 (FES) field;
    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 a PDU Length (PLEN) field 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 (FES) field;
    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 a PLEN field 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.

The host and controller may detect connection loss at different times (refer to the Communication Loss Handling section in the NVM Express Base Specification). The host should not assume the controller has detected connection loss if there is a loss of a TCP connection detected by the host (e.g., due to host-side failures or an intermediary such as a firewall sending a TCP RST). The host should treat the reception of a C2HTermReq PDU as an indication that the controller has initiated clean-up of the NVMe Transport 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 the Status Code – Generic Command Status Values figure in the NVM Express 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 (TLS) requirements for the TCP transport. Requirements associated with Post-Quantum Cryptography are outside the scope of this specification.

3.6.1.1 Transport Specific Address Subtype: TLS

As specified in section 3.1.1, a Discovery Log Page Entry for the TCP transport includes a Transport Specific Address Subtype (TSAS) field. The TSAS field for the TCP transport is defined in Figure 17 and contains a SECTYPE field that 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 supported by the NVMe/TCP port. The SECTYPE value of 0h (i.e., No Security) specifies that TLS is not supported.
<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 (the obsolete NVMe over Fabrics Specification describes requirements for TLS version 1.2). TLS version 1.2 should not be used with NVMe/TCP.</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 cipher suite for each TCP connection.</td></tr>
<tr><td>255:03</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. Implementation and use of the TLS_AES_256_GCM_SHA384 cipher suite may be necessary to meet requirements of security policies that are not defined by NVM Express (e.g., CNSA 1.0 as specified in CNSSP 15, Annex B (the NSA-Approved Commercial National Security Algorithm (CNSA) 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 PSK with (EC)DHE (refer to RFC 8446).

PSK with (EC)DHE protects encrypted traffic against compromise of the pre-shared key. Implementations that support PSK-only authentication 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 secp384r1 group (refer to section 4.2.7 of RFC 8446) for PSK with ECDHE. 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.

If TLS 1.2 is supported in addition to TLS 1.3, then using the same PSK with both TLS 1.2 and TLS 1.3 is prohibited.

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 NVM Express 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 storing that retained PSK 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. In this case, the client implementation should connect first with the most preferred PSK. For the cipher suites that are supported, the PSKs should be used in the following order of the associated cipher suite:

1. TLS_AES_256_GCM_SHA384; and
2. TLS_AES_128_GCM_SHA256.

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 preferred to least preferred. For the cipher suites that are supported, the PSKs should be listed in the following order of the associated cipher suite:

1. TLS_AES_256_GCM_SHA384; and
2. TLS_AES_128_GCM_SHA256.

The TLS 1.3 PSK identity used with NVMe/TCP is generated from the NQNs of the host and the NVM subsystem that contains the controller and from a digest of the PSK to which the identity is associated. 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, as shown in Figure 18;

Figure 18: TLS protocol version indicator

Value	Definition
‘0’ (i.e., U+0030h)	Obsolete. Refer to NVM Express TCP Transport Specification 1.0.
‘1’ (i.e., U+0031h)	Indicates TLS 1.3 with PSK digest in the PSK identity
All other values	Reserved

3. A one-character PSK type indicator, specifying the used PSK, as shown in Figure 19;

Figure 19: PSK type indicator

Value	Definition
‘R’ (i.e., U+0052h)	Indicates a retained PSK
‘G’ (i.e., U+0047h)	Indicates a generated PSK
All other values	Reserved

4. A two-character hash specifier, specifying the hash function of the cipher suite associated with this PSK identity, as shown in Figure 20;

Figure 20: Hash function indicator

Value	Definition
“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)
All other values	Reserved

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 NVM subsystem that contains the controller (i.e., NQN_c);
9. A space character (i.e., U+0020h);
10. The PSK digest; and
11. A null character (i.e., U+0000h).

The PSK digest shall be computed by encoding in Base64 (refer to RFC 4648) the result of the application of the HMAC function using the hash function specified in element 4 of the above in-order concatenation with the PSK as the HMAC key to the concatenation of:

- the NQN of the host (i.e., NQN_h) not including the null terminator;
- a space character;
- the NQN of the NVM subsystem (i.e., NQN_c) not including the null terminator;
- a space character; and
- the seventeen ASCII characters “NVMe-over-Fabrics”,

(i.e., <PSK digest> = Base64(HMAC(PSK, NQN_h || “ “ || NQN_c || “ “ || ”NVMe-over-Fabrics”))).

The length of the PSK digest depends on the hash function used as follows:

- If the SHA-256 hash function is used, the resulting PSK digest is 44 characters long; or
- If the SHA-384 hash function is used, the resulting PSK digest is 64 characters long.

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:

“NVMe1R01 nqn.2014-08.org.nvmexpress:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6 nqn.2014-08.org.nvmexpress:uuid:36ebf5a9-1df9-47b3-a6d0-e9ba32e428a2 <PSK digest>”

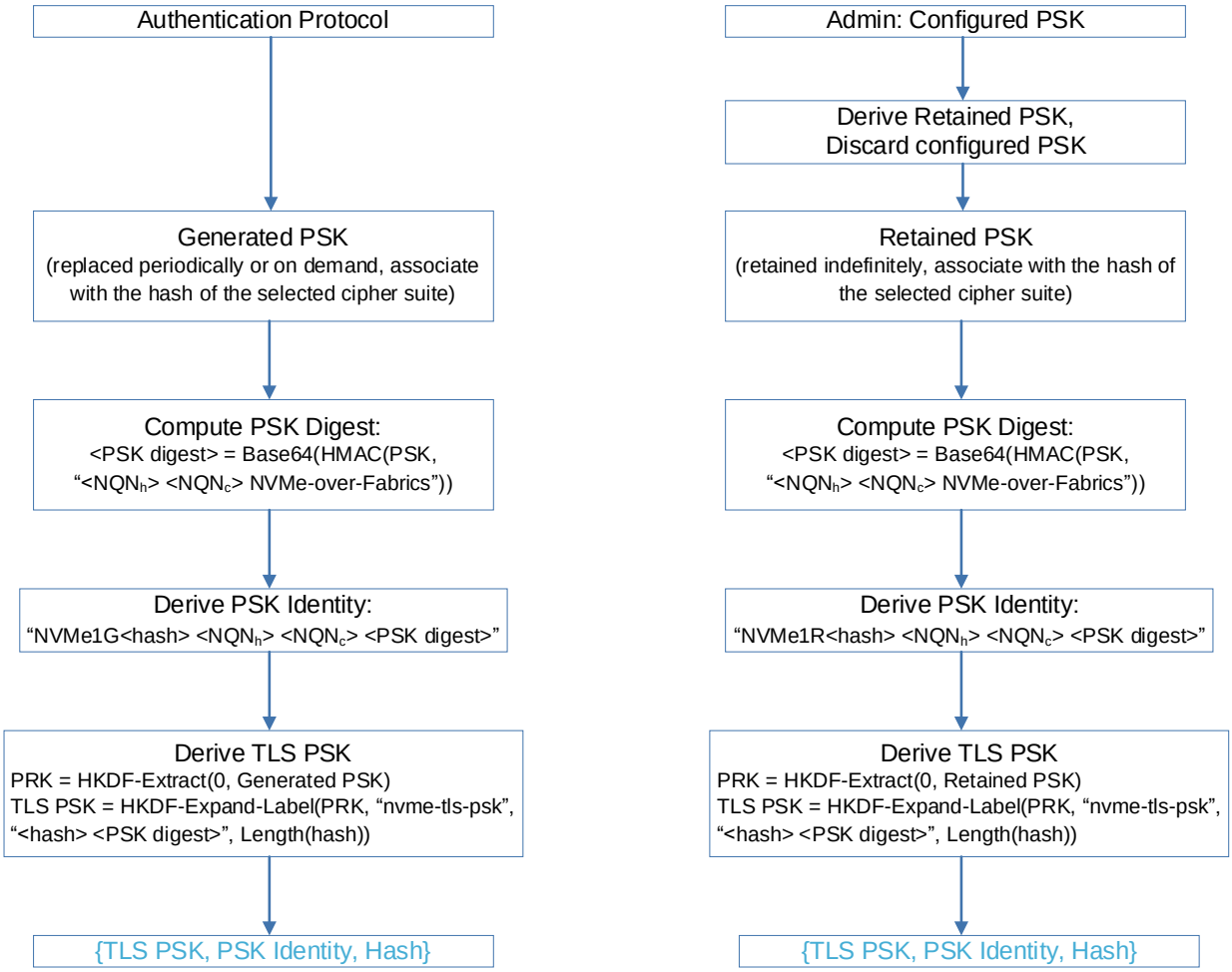
The TLS PSK shall be derived as follows from an input PSK (i.e., either a retained PSK or a generated PSK) and a PSK identity using the HKDF-Extract and HKDF-Expand-Label operations (refer to RFC 5869 and RFC 8446) where the hash function is the one specified by the hash specifier of the PSK identity:

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

where Context is the hash identifier indicated in the PSK identity concatenated to a space character and to the Base64 PSK digest (i.e., “<hash> <PSK digest>”) 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 21 for both configured PSK and generated PSK. This process is performed for each supported TLS 1.3 cipher suite hash function.

Figure 21: {TLS PSK, TLS Identity, Hash} Tuple Derivation



The TLS PSK derivation specified in the NVM Express TCP Transport Specification Revision 1.0 is based on a PSK identity having "0" as TLS protocol version indicator and is not compatible with the process specified in this specification. The NVM Express TCP Transport Specification Revision 1.0 specification of the TLS PSK derivation is ambiguous if the associated PSK identity length exceeds 256 characters, hence interoperability problems may arise if the associated PSK identity is longer than 256 characters.

3.6.1.4 PSK Use

If a retained PSK is available for a host and an NVM subsystem, then that retained PSK and its related PSK Identity shall be used between that host and that NVM subsystem to set up all TLS secure channels for the Admin Queue and all I/O Queues of each controller associated with that host and the use of any generated PSK between that host and that subsystem is prohibited.

If a retained PSK is not available for a host and an NVM subsystem, then one or more generated PSKs shall be used between that host and that NVM subsystem to set up any TLS secure channels.

A generated PSK and its related PSK identity are derived by performing an authentication transaction with the SC_C field in the AUTH_Negotiate message set to NEWTLSPSK (refer to the AUTH_Negotiate Message section of the NVM Express Base Specification) on an Admin Queue over a TCP channel without TLS. Once that authentication transaction is completed, the NVMe/TCP transport connection of that Admin Queue shall be disconnected (refer to the Secure Channel Concatenation section of the NVM Express

Base Specification). The generated {PSK, PSK Identity} pair may be used to set up TLS secure channels for subsequent Admin and I/O queues, as specified in section 3.1.

Once the TLS secure channel for the Admin Queue of an association has been set up with a generated {PSK, PSK Identity} pair, that generated {PSK, PSK Identity} pair should be replaced periodically (e.g., every hour) or on demand by performing a reauthentication with the SC_C field in the AUTH_Negotiate message set to REPLACETLSPSK (refer to the AUTH_Negotiate Message section of the NVM Express Base Specification) over the Admin Queue of that association. The most recently generated PSK, if any, is the generated PSK associated with that Admin Queue.

The TLS secure channels for each I/O Queue of an association shall be set up using the generated PSK that is associated with the Admin Queue of that association. TLS secure channels already set up with a generated PSK shall not be affected by a PSK replacement.

A generated {PSK, PSK Identity} pair may be used to set up the TLS secure channels for Admin Queues of additional associations between the same host and NVM subsystem. In this case that generated {PSK, PSK Identity} pair should be replaced periodically (e.g., every hour) or on demand by performing a reauthentication with the SC_C field in the AUTH_Negotiate message set to REPLACETLSPSK (refer to the AUTH_Negotiate Message section of the NVM Express Base Specification) over the Admin Queue of the association having the smallest Controller ID among the associations using that {PSK, PSK Identity} pair.

The process of generating a PSK on an Admin Queue over a TCP channel without TLS, disconnecting that Admin Queue transport connection, and setting up an Admin Queue over a TLS secure channel established using that generated PSK is called TLS concatenation.

A host specifies the PSK identity of the PSK to use to set up a TLS secure channel with a subsystem in the ClientHello message of the TLS handshake. If that PSK, either retained or generated, is not available on the subsystem, then:

- the TLS 1.3 handshake shall be aborted with an unknown_psk_identity alert; and
- the TCP connection shall be closed.

If the host was attempting to set up with a generated PSK the TLS secure channel for an I/O Queue belonging to an existing secured association (i.e., an association whose Admin Queue was set up over a TLS secure channel), then to set up that TLS secure channel the host shall:

- first replace the association's {PSK, PSK Identity} pair by performing a reauthentication with the SC_C field in the AUTH_Negotiate message set to REPLACETLSPSK (refer to the AUTH_Negotiate Message section of the NVM Express Base Specification) over the Admin Queue of that association; and
- then attempt again to set up the TLS secure channel for that I/O Queue.

If the host was attempting to set up with a generated PSK the TLS secure channel for the Admin Queue of a new association, then to set up that TLS secure channel the host shall:

- first perform an authentication transaction with the SC_C field in the AUTH_Negotiate message set to NEWTLSPSK (refer to the AUTH_Negotiate Message section of the NVM Express Base Specification) over an unsecured channel to generate a new {PSK, PSK Identity} pair useable for that new association; and
- then attempt again to set up the TLS secure channel for that Admin Queue.

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 PSK (4 bytes binary).

As an example, the 32-byte configured PSK:

```
5512DBB6 737D0106 F65975B7 73DFB011 FFC344BC F442E2DD 6D8BC487 0B5D5B03h
```

has the CRC-32 D9BAF45Fh, that is represented in little endian format (i.e., 5FF4BAD9h) for concatenation to the configured PSK, resulting in the following:

```
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 implementations shall not send or use 0-RTT data to avoid replay attacks (refer to Appendix E.5 of RFC 8446).

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, if supported;
- Each individual DH group; and
- each individual ECDH group.

TLS concatenation is prohibited over a TLS 1.3 session that has been established using a PSK as specified in section 3.6.1. The controller failure response to a host request for such TLS concatenation is specified in the AUTH_Negotiate section of the NVM Express Base Specification. In contrast, replacing a TLS PSK for such a TLS 1.3 session is permitted (refer to section 3.6.1.4).

TLS Application-Layer Protocol Negotiation enables negotiation of the application-level protocol to be used with TLS in situations where multiple application protocols are supported on the same TCP or UDP port (refer to RFC 7301). NVMe/TCP is not expected to be deployed as one of multiple protocols that are

supported on the same TCP port and no ALPN Protocol ID has been registered for NVMe/TCP. For these reasons, ALPN shall not be used with TLS for NVMe/TCP.

TLS 1.3 implementations for NVMe/TCP are not required to support TLS session resumption (refer to section 2.2 of RFC 8446), and TLS session resumption should not be used with NVMe/TCP.

3.6.1.6.1 TLS Channel Binding

If a Connect command (refer to the NVM Express Base Specification) is sent over a TLS 1.3 session established using a PSK as specified in section 3.6.1, then the controller shall check whether:

- the NVM Subsystem NVMe Qualified Name (SUBNQN) field of the Connect command is the same as the NQN of the controller (i.e., NQN_c) in the identity of that PSK; and
- the Host NVMe Qualified Name (HOSTNQN) field of the Connect command is the same as the NQN of the host (i.e., NQN_h) in the identity of that PSK.

If either the SUBNQN check or the HOSTNQN check fails during processing of the Connect command (i.e., the compared NQNs are not the same), then the controller shall abort that Connect command with a status code of Connect Invalid Parameters (refer to the Connect Command and Response section of the NVM Express Base specification).

3.6.1.6.2 Unexpected Data During TLS 1.3 Handshake

A TLS 1.3 handshake is initiated by the host upon completing the TCP handshake at the start of a TCP connection (refer to section 3.1). If the host initiates such a TLS 1.3 handshake, then the host and controller are prohibited from sending any other data instead of initiating or performing the TLS 1.3 handshake, as further specified in this section. This prohibition does not apply to TCP connections that do not use TLS.

A host that initiates the TLS 1.3 handshake at the start of a TCP connection shall not send any other data before initiating the TLS 1.3 handshake. If a controller receives data that is prohibited by this requirement, then the controller shall:

- immediately close the TCP connection without attempting to determine whether the host initiated the TLS 1.3 handshake after sending the prohibited data;
- treat the prohibited data as opaque data; and
- not process that prohibited data (e.g., not process that data as NVMe/TCP PDUs).

A controller shall not send any data on a TCP connection until the controller receives data from the host and determines whether that data indicates that the host has initiated a TLS 1.3 handshake. If a host receives data that is prohibited by this requirement, then the host shall:

- abort the TLS 1.3 handshake with an unexpected_message alert if the host has already initiated the TLS 1.3 handshake;
- close the TCP connection regardless of whether or not the host has initiated the TLS 1.3 handshake; and
- treat the prohibited data as opaque data and not process that data (e.g., as if that data consisted of one or more NVMe/TCP PDUs).

A host shall not send any other data (e.g., NVMe/TCP PDUs) while a TLS 1.3 handshake is in progress. If a controller receives any other data (e.g., NVMe/TCP PDUs) while a TLS 1.3 handshake is in progress (i.e., before receiving the TLS 1.3 Finished message), then receiving that data (e.g., NVMe/TCP PDUs) shall cause the TLS 1.3 handshake to be aborted with an appropriate error alert (e.g., decode_error or unexpected_message), as specified in section 6.2 of RFC 8446, and shall cause the TCP connection to be closed. That received data shall be treated as opaque data and shall not be processed (e.g., as if that data consisted of one or more NVMe/TCP PDUs).

A controller shall not send any other data (e.g., NVMe/TCP PDUs) while a TLS 1.3 handshake is in progress. If a host receives any other data (e.g., NVMe/TCP PDUs) while a TLS 1.3 handshake is in progress (i.e., before receiving the TLS 1.3 Finished message), then receiving that data (e.g., NVMe/TCP PDUs) shall cause the TLS 1.3 handshake to be aborted with an appropriate error alert (e.g., decode_error or unexpected_message), as specified in section 6.2 of RFC 8446, and shall cause the TCP connection to

be closed. That received data shall be treated as opaque data and shall not be processed (e.g., as if that data consisted of one or more NVMe/TCP PDUs).

All data sent after completion of the TLS 1.3 handshake (i.e., after sending the TLS 1.3 Finished message) shall be encrypted (refer to section 4.4.4 of RFC 8446). If a host or controller receives unencrypted data after completion of the TLS 1.3 handshake, then receiving that data shall cause the TLS 1.3 session to be aborted with an appropriate alert (e.g., decode_error or unexpected_message), as specified in section 6.2 of RFC 8446, and shall cause the TCP connection to be closed. That unencrypted data shall be treated as opaque data and shall not be processed (e.g., as if that data consisted of one or more NVMe/TCP PDUs).

3.6.1.7 TLS Configuration for TCP Connection Initiation

NVMe/TCP implementations that support TLS shall be able to be provisioned with a configured PSK for each remote entity with which a TLS connection may be established.

This section uses the term configuration to indicate an NVMe/TCP internal state that controls the use of TLS. That internal state may or may not be externally configurable for a host or NVM subsystem. For an NVMe/TCP implementation that supports TLS, a TLS configuration may apply to:

- a single connection;
- a group of connections; or
- all connections.

NVMe/TCP implementations that support TLS shall support configuring use of TLS with TCP connection initiation using one or more of the configurations shown in Figure 22.

Figure 22: TLS Configurations

Configuration	Definition
TLS Disabled	Only TCP connections without TLS with a remote entity are allowed
TLS Permitted	TCP connections with and without TLS with a remote entity are allowed
TLS Required	Only TCP connections with TLS with a remote entity are allowed

NVMe/TCP hosts that support TLS shall behave as shown in Figure 23 when establishing NVMe/TCP connections with an NVM subsystem.

Figure 23: Host TLS Behavior

Host Configuration	Action
TLS Disabled	Do not initiate TCP connections with TLS.
TLS Permitted	If the SECTYPE field in the TSAS field in the Discovery Log Page Entry for the remote entity is not cleared to 0h, then initiate TCP connections with TLS, irrespective of the value of the TSC field in that Discovery Log Page Entry. If establishing any TCP connection with TLS fails and the TSC field in that Discovery Log Page Entry is not set to 01b (i.e., Required), the host may fall back to initiate TCP connections without TLS. If the SECTYPE field in the TSAS field in the Discovery Log Page Entry for the remote entity is cleared to 0h and the TSC field is not set to 01b (i.e., Required), then initiate TCP connections without TLS. If the SECTYPE field in the TSAS field in the Discovery Log Page Entry for the remote entity is cleared to 0h and the TSC field is set to 01b (i.e., Required), then that Discovery Log Page Entry is inconsistent and TCP connections without TLS may or may not be initiated. If no Discovery Log Page Entry has been retrieved for the remote entity, then TCP connections with or without TLS may be initiated.
TLS Required	Initiate TCP connections with TLS.

NVM subsystems that support TLS with NVMe/TCP shall behave as defined in Figure 24 when establishing NVMe/TCP connections with a host.

Figure 24: NVM Subsystem TLS Behavior

Subsystem Configuration	Description
TLS Disabled	Close the TCP connection if a TLS handshake is initiated upon completion of the TCP handshake
TLS Permitted	Continue all TCP connections whether or not a TLS handshake is initiated upon completion of the TCP handshake
TLS Required	Close the TCP connection if a TLS handshake is not initiated upon completion of the TCP handshake

For example, consider a host that supports TLS configured with TLS Disabled and an NVM subsystem that supports TLS configured with TLS Permitted. As defined in Figure 23, the host initiates a TCP connection without TLS. As defined in Figure 24, the subsystem accepts the TCP connection.

3.6.2 NVMe/TCP PDUs

This section describes the format of NVME/TCP PDUs.

3.6.2.1 PDU Common Header (CH)

Figure 25: PDU Common Header (CH)

Bytes	Description
00	**PDU Type (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 (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 26: 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 (PDU-Type): 00h</td></tr>
<tr><td>01</td><td>Flags (FLAGS):
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7</td><td>Kickstart Discovery Connection (KDCONN): If this bit is set to ‘1’, then a kickstart discovery (refer to the Kickstart Discovery Pull Registration Requests section in the NVM Express Base Specification) NVMe/TCP connection is established, as described in section 3.1. If this bit is cleared to ‘0’, then a non-kickstart discovery NVMe/TCP connection is established, as described in section 3.1.</td></tr>
<tr><td>6:0</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>02</td><td>Header Length (HLEN): Fixed length of 128 bytes (80h).</td></tr>
<tr><td>03</td><td>PDU Data Offset (PDO): Reserved</td></tr>
<tr><td>07:04</td><td>PDU Length (PLEN): Fixed length of 128 bytes (80h).</td></tr>
<tr><td>09:08</td><td rowspan="2">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>
</table>

Figure 26: Initialize Connection Request PDU (ICReq)

<table>
<tr><th>Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>11</td><td rowspan="3"></td><td>Digest (DGST): Host PDU header and data digest enable options.
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
<tr><td>1</td><td>PDU Data Digest Enable (DDGST_ENABLE): If this bit is set to ‘1’, then the use of data digest is requested by the host for the connection. If this bit is cleared to ‘0’, then the data digest shall not be used for the connection.</td></tr>
<tr><td>0</td><td>PDU Header Digest Enable (HDGST_ENABLE): If this bit is set to ‘1’, then the use of header digest is requested by the host for the connection. If this bit is cleared to ‘0’, then the 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 27: 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 (PDU-Type): 01h</td></tr>
<tr><td>01</td><td>Flags (FLAGS): Reserved</td></tr>
<tr><td>02</td><td>Header Length (HLEN): Fixed length of 128 bytes (80h).</td></tr>
<tr><td>03</td><td>PDU Data Offset (PDO): Reserved</td></tr>
<tr><td>07:04</td><td>PDU Length (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>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>Digest (DGST): Controller PDU header and data digest enable options.
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
<tr><td>1</td><td>Data Digest Enable (DDGST_ENABLE): If this bit is set to ‘1’, then the data digest is used for the connection. If this bit is cleared to ‘0’, then the data digest is not used for the connection.</td></tr>
<tr><td>0</td><td>Header Digest Enable (HDGST_ENABLE): If this bit is set to ‘1’, then the header digest is used for the connection. If this bit is cleared to ‘0’, then the 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 28: 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 (PDU-Type): 02h</td></tr>
<tr><td>01</td><td>Flags (FLAGS): Reserved</td></tr>
<tr><td>02</td><td>Header Length (HLEN): Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td>PDU Data Offset (PDO): Reserved</td></tr>
<tr><td>07:04</td><td>PDU Length (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>Definition</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 (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 29: 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 (PDU-Type): 03h</td></tr>
<tr><td>01</td><td>Flags (FLAGS): Reserved</td></tr>
<tr><td>02</td><td>Header Length (HLEN): Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td>PDU Data Offset (PDO): Reserved</td></tr>
<tr><td>07:04</td><td>PDU Length (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>Definition</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 (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 30: Command Capsule PDU (CapsuleCmd)

<table>
<tr><th colspan="2">Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td colspan="2">00</td><td rowspan="5">CH</td><td>PDU Type (PDU-Type): 04h</td></tr>
<tr><td colspan="2">01</td><td>Flags (FLAGS):
<table><tr><th>Bits</th><th>Description</th></tr><tr><td>7:2</td><td>Reserved</td></tr><tr><td>1</td><td>PDU Data Digest Flag (DDGSTF): If this bit is set to ‘1’, then the DDGST field follows the PDU Data and contains a valid value. If this bit is cleared to ‘0’, then the DDGST field is not present.</td></tr><tr><td>0</td><td>PDU Header Digest Flag (HDGSTF): If this bit is set to ‘1’, then the HDGST field follows the PDU Header and contains a valid value. If this bit is cleared to ‘0’, then the HDGST field is not present.</td></tr></table></td></tr>
<tr><td colspan="2">02</td><td>Header Length (HLEN): Fixed length of 72 bytes (i.e., 48h).</td></tr>
<tr><td colspan="2">03</td><td>PDU Data Offset (PDO): Data offset within PDU (i.e., the offset from byte 0 to the CCICD field; the value of ‘N’). 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 colspan="2">07:04</td><td>PDU Length (PLEN): Total length of PDU (including CH, PSH, HDGST, PAD, DATA, and DDGST) in bytes.</td></tr>
<tr><td colspan="2">71:08</td><td>PSH</td><td>NVMe-oF Command Capsule SQE (CCSQE): Command Capsule SQE.</td></tr>
<tr><td>HDGSTF=1</td><td>HDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>75:72</td><td>Not present</td><td>HDGST</td><td>PDU Header Digest (HDGST): If the HDGSTF bit is set to ‘1’ in the FLAGS field, this field is present and contains a valid header digest (refer to section 3.3.1.1). If the HDGSTF bit is cleared to ‘0’, then this field is not present.</td></tr>
<tr><td>N - 1:76</td><td>N - 1:72</td><td>PAD</td><td>Pad (PAD): If in-capsule data is present, the length of this field shall be the necessary number of bytes required to achieve the alignment specified by the CPDA field (refer to section 3.6.2.3).</td></tr>
<tr><td colspan="2">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>DDGSTF=1</td><td>DDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>M + 3:M</td><td>Not present</td><td>DDGST</td><td>PDU Data Digest (DDGST): If the DDGSTF bit 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 (i.e., the in-capsule data). If the DDGSTF bit is cleared to ‘0’, then this field is not present.</td></tr>
</table>

3.6.2.7 Response Capsule PDU (CapsuleResp)

Figure 31: 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 (PDU-Type): 05h</td></tr>
<tr><td>01</td><td>Flags (FLAGS):
<table><tr><th>Bits</th><th>Description</th></tr><tr><td>7:1</td><td>Reserved</td></tr><tr><td>0</td><td>PDU Header Digest Flag (HDGSTF): If this bit is set to ‘1’, then a valid HDGST value follows the PDU Header. If this bit is cleared to ‘0’, then the HDGST field is not present.</td></tr></table></td></tr>
<tr><td>02</td><td>Header Length (HLEN): Fixed length of 24 bytes (i.e., 18h).</td></tr>
<tr><td>03</td><td>PDU Data Offset (PDO): Reserved</td></tr>
<tr><td>07:04</td><td>PDU Length (PLEN): Length of CH, PSH, and HDGST, if present, in bytes.</td></tr>
</table>

Figure 31: Response Capsule PDU (CapsuleResp)

Bytes		PDU Section	Description
23:08		PSH	**NVMe-oF Response Capsule CQE (RCCQE):** Response Capsule CQE.
HDGSTF=1	HDGSTF=0		
27:24	Not present	HDGST	**PDU Header Digest (HDGST):** If the HDGSTF bit is set to '1' in the FLAGS field, this field is present and contains a valid header digest (refer to section 3.3.1.1). If the HDGSTF bit is cleared to '0', then this field is not present.

3.6.2.8 Host To Controller Data Transfer PDU (H2CData)

Figure 32: Host To Controller Data Transfer PDU (H2CData)

<table>
<tr><th colspan="2">Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td colspan="2">00</td><td rowspan="5">CH</td><td>PDU Type (PDU-Type): 06h</td></tr>
<tr><td colspan="2">01</td><td>Flags (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 (LAST_PDU): If this bit is 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>PDU Data Digest Flag (DDGSTF): If this bit is set to '1', then the DDGST field follows the PDU Data and contains a valid value. If this bit is cleared to '0', then the DDGST field is not present.</td></tr>
<tr><td>0</td><td>PDU Header Digest Flag (HDGSTF): If this bit is set to '1', then the HDGST field follows the PDU Header and contains a valid value. If this bit is cleared to '0', then the HDGST field is not present.</td></tr>
</table></td></tr>
<tr><td colspan="2">02</td><td>Header Length (HLEN): Fixed length of 24 bytes (i.e., 18h).</td></tr>
<tr><td colspan="2">03</td><td>PDU Data Offset (PDO): Data Offset within PDU (i.e., the offset from byte 0 to the PDU-Data field; the value of 'N'). 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 colspan="2">07:04</td><td>PDU Length (PLEN): Total length of PDU (including CH, PSH, HDGST, PAD, DATA, and DDGST) in bytes.</td></tr>
<tr><td colspan="2">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 Command Data Buffer.</td></tr>
<tr><td colspan="2">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 colspan="2">15:12</td><td>Data Offset (DATAO): Byte offset from start of Command Data Buffer to the first byte to transfer. This value shall be a multiple of dwords.</td></tr>
<tr><td colspan="2">19:16</td><td>Data Length (DATAL): PDU-Data field length in bytes (i.e., the value of M-N). This value shall be a multiple of dwords.</td></tr>
<tr><td colspan="2">23:20</td><td>Reserved</td></tr>
<tr><td>HDGSTF=1</td><td>HDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>27:24</td><td>Not present</td><td>HDGST</td><td>PDU Header Digest (HDGST): If the HDGSTF bit is set to '1' in the FLAGS field, this field is present and contains a valid header digest (refer to section 3.3.1.1). If the HDGSTF bit is cleared to '0', then this field is not present.</td></tr>
<tr><td>N - 1:28</td><td>N - 1:24</td><td>PAD</td><td>Pad (PAD): The length of this field shall be the necessary number of bytes required to achieve the alignment specified by the CPDA field (refer to section 3.6.2.3).</td></tr>
<tr><td colspan="2">M - 1:N</td><td>DATA</td><td>PDU Data (PDU-Data): This field contains the contents of the Command Data Buffer being transferred. The length of this field is a multiple of dwords.</td></tr>
<tr><td>DDGSTF=1</td><td>DDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>M + 3:M</td><td>Not present</td><td>DDGST</td><td>Data Digest (DDGST): If the DDGSTF bit is set to '1' in the FLAGS field, this field is present and contains a valid data digest (refer to section 3.3.1.1). If the DDGSTF bit is cleared to '0', then this field is not present.</td></tr>
</table>

3.6.2.9 Controller To Host Data Transfer PDU (C2HData)

Figure 33: Controller To Host Data Transfer PDU (C2HData)

<table>
<tr><th colspan="2">Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td colspan="2">00</td><td rowspan="5">CH</td><td>PDU Type (PDU-Type): 07h</td></tr>
<tr><td colspan="2">01</td><td>Flags (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 (SCSS): If this bit is set to ‘1’, then 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 (LAST_PDU): If this bit is set to ‘1’, then the PDU is the last C2HData PDU sent in response to a Command Capsule PDU.</td></tr>
<tr><td>1</td><td>PDU Data Digest Flag (DDGSTF): If this bit is set to ‘1’, then the DDGST field follows the PDU Data and contains a valid value. If this bit is cleared to ‘0’, then the DDGST field is not present.</td></tr>
<tr><td>0</td><td>PDU Header Digest Flag (HDGSTF): If this bit is set to ‘1’, then the HDGST field follows the PDU Header and contains a valid value. If this bit is cleared to ‘0’, then the HDGST field is not present.</td></tr>
</table></td></tr>
<tr><td colspan="2">02</td><td>Header Length (HLEN): Fixed length of 24 bytes (i.e., 18h).</td></tr>
<tr><td colspan="2">03</td><td>PDU Data Offset (PDO): Data offset within PDU (i.e., the offset from byte 0 to the PDU-Data field; the value of ‘N’). This value shall be a multiple of the data alignment specified by the HPDA field in the ICReq PDU (refer to section 3.6.2.2) that was previously sent by the host on this TCP connection.</td></tr>
<tr><td colspan="2">07:04</td><td>PDU Length (PLEN): Total length of PDU (i.e., including CH, PSH, HDGST, PAD, DATA, and DDGST) in bytes.</td></tr>
<tr><td colspan="2">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 colspan="2">11:10</td><td>Reserved</td></tr>
<tr><td colspan="2">15:12</td><td>Data Offset (DATAO): Byte offset from start of host-resident data to the first byte to transfer. This value shall be dword aligned.</td></tr>
<tr><td colspan="2">19:16</td><td>Data Length (DATAL): PDU-Data field length in bytes (i.e., the value of M-N). This value shall be dword aligned.</td></tr>
<tr><td colspan="2">23:20</td><td>Reserved</td></tr>
<tr><td>HDGSTF=1</td><td>HDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>27:24</td><td>Not present</td><td>HDGST</td><td>PDU Header Digest (HDGST): If the HDGSTF bit is set to ‘1’ in the FLAGS field, this field is present and contains a valid header digest (refer to section 3.3.1.1). If the HDGSTF bit is cleared to ‘0’, then this field is not present.</td></tr>
<tr><td>N - 1:28</td><td>N - 1:24</td><td>PAD</td><td>Pad (PAD): If the HPDA field (refer to section 3.6.2.2) is set to a non-zero value, then the length of this field shall be the necessary number of bytes required to achieve the alignment specified by the HPDA field.</td></tr>
<tr><td colspan="2">M - 1:N</td><td>DATA</td><td>PDU Data (PDU-Data): This field contains the host-resident data being transferred. The length of this field is a multiple of dwords.</td></tr>
<tr><td>DDGSTF=1</td><td>DDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>M + 3:M</td><td>Not present</td><td>DDGST</td><td>PDU Data Digest (DDGST): If the DDGSTF bit is set to ‘1’ in the FLAGS field, this field is present and contains a valid data digest (refer to section 3.3.1.1) of the PDU-Data field. If the DDGSTF bit is cleared to ‘0’, then this field is not present.</td></tr>
</table>

3.6.2.10 Ready to Transfer PDU (R2T)

Figure 34: Ready to Transfer PDU (R2T)

<table>
<tr><th colspan="2">Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td colspan="2">00</td><td rowspan="4">CH</td><td>PDU Type (PDU-Type): 09h</td></tr>
<tr><td colspan="2">01</td><td>Flags (FLAGS):
<table><tr><th>Bits</th><th>Description</th></tr><tr><td>7:1</td><td>Reserved</td></tr><tr><td>0</td><td>PDU Header Digest Flag (HDGSTF): If this bit is set to ‘1’, then the HDGST field follows the PDU Header and contains a valid value. If this bit is cleared to ‘0’, then the HDGST field is not present.</td></tr></table></td></tr>
<tr><td colspan="2">02</td><td>Header Length (HLEN): Fixed length of 24 bytes (i.e., 18h).</td></tr>
<tr><td colspan="2">03</td><td>PDU Data Offset (PDO): Reserved</td></tr>
<tr><td colspan="2">07:04</td><td rowspan="6">PSH</td><td>PDU Length (PLEN): Length of CH, PSH, and HDGST, if present, in bytes.</td></tr>
<tr><td colspan="2">09:08</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 colspan="2">11:10</td><td>Transfer Tag (TTAG): This field contains a controller generated tag. The rules of the tag generation are outside the scope of this specification.</td></tr>
<tr><td colspan="2">15:12</td><td>Requested Data Offset (R2TO): Byte offset from the start of the host-resident data to the first byte to transfer. This value shall be dword aligned.</td></tr>
<tr><td colspan="2">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 colspan="2">23:20</td><td>Reserved</td></tr>
<tr><td>HDGSTF=1</td><td>HDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>27:24</td><td>Not present</td><td>HDGST</td><td>PDU Header Digest (HDGST): If the HDGSTF bit is set to ‘1’ in the FLAGS field, this field is present and contains a valid header digest (refer to section 3.3.1.1). If the HDGSTF bit is cleared to ‘0’, then this field is not present.</td></tr>
</table>

3.6.2.11 Kickstart Discovery Request PDU (KDReq)

Figure 35: Kickstart Discovery Request PDU (KDReq)

<table>
<tr><th colspan="2">Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td colspan="2">00</td><td rowspan="5">CH</td><td>PDU Type (PDU-Type): 0Ah</td></tr>
<tr><td colspan="2">01</td><td>Flags (FLAGS):
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
<tr><td>1</td><td>PDU Data Digest Flag (DDGSTF): If this bit is set to ‘1’, then the DDGST field follows the PDU Data and contains a valid value. If this bit is cleared to ‘0’, then the DDGST field is not present.</td></tr>
<tr><td>0</td><td>PDU Header Digest Flag (HDGSTF): If this bit is set to ‘1’, then the HDGST field follows the PDU Header and contains a valid value. If this bit is cleared to ‘0’, then the HDGST field is not present.</td></tr>
</table></td></tr>
<tr><td colspan="2">02</td><td>Header Length (HLEN): Fixed length of 12 bytes (i.e., Ch).</td></tr>
<tr><td colspan="2">03</td><td>PDU Data Offset (PDO): Obsolete</td></tr>
<tr><td colspan="2">07:04</td><td>PDU Length (PLEN): Variable length in bytes.
If HDGST and DDGST are both not present, then the length will be (NUMKR * 290 bytes) + 12 bytes.
If only HDGST or only DDGST is present, then the length will be (NUMKR * 290 bytes) + 16 bytes.
If HDGST and DDGST are both present, then the length will be (NUMKR * 290 bytes) + 20 bytes.</td></tr>
<tr><td colspan="2">09:08</td><td rowspan="2">PSH</td><td>Number of Kickstart Records (NUMKR): This field specifies the number kickstart records included in the PDU DATA field.</td></tr>
<tr><td colspan="2">11:10</td><td>Number of Discovery Information Entries (NUMDIE): This field specifies the maximum number of discovery information entires that the DDC is expected to return if a pull registration is requested. This field shall be cleared to 0h if a pull de-registration is being requested. Refer to the Pull Registrations and Pull De-Registrations section in the NVM Express Base Specification.</td></tr>
<tr><td>HDGSTF=1</td><td>HDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>15:12</td><td>Not present</td><td>HDGST</td><td>PDU Header Digest (HDGST): If the HDGSTF bit is set to ‘1’ in the FLAGS field, this field is valid and contains the header digest (refer to section 3.3.1.1). If the HDGSTF bit is cleared to ‘0’, then this field is not present.</td></tr>
<tr><td>305:16</td><td>301:12</td><td rowspan="3">DATA</td><td>Kickstart Record 0: This field specifies the first kickstart record as defined in Figure 36.</td></tr>
<tr><td>…</td><td>…</td><td>…</td></tr>
<tr><td>(((N + 1) * 290) – 1) + 16:(N * 290) + 16</td><td>(((N + 1) * 290) – 1) + 12:(N * 290) + 12</td><td>Kickstart Record N: This field specifies the last kickstart record as defined in Figure 36 (if present). N is defied as the value in the NUMKR field minus 1.</td></tr>
<tr><td>DDGSTF=1</td><td>DDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>M + 3:M</td><td>Not present</td><td>DDGST</td><td>PDU Data Digest (DDGST): If the DDGSTF bit is set to ‘1’ in the FLAGS field, this field is valid and contains the data digest (refer to section 3.3.1.1). If the DDGSTF bit is cleared to ‘0’, then this field is not present.</td></tr>
</table>

Figure 36: Kickstart Record

Bytes	Description
00	**Transport Type (TRTYPE):** This field specifies the transport type as defined in the Transport Type (TRTYPE) field in the Discovery Log Page Entry data structure in the NVM Express Base Specification.
01	**Address Family (ADRFAM):** This field specifies the address family as defined in the Address Family (ADRFAM) field in the Discovery Log Page Entry data structure in the NVM Express Base Specification.

Figure 36: Kickstart Record

Bytes	Description
33:02	**Transport Service Identifier (TRSVCID):** This field specifies the NVMe Transport service identifier as an ASCII string as defined in the Transport Service Identifier (TRSVCID) field of the Discovery Log Page Entry data structure in the NVM Express Base Specification.
289:34	**Transport Address (TRADDR):** This field specifies the address of the DDC that may be used for a Connect command as an ASCII string as defined in the Transport Address (TRADDR) field of the Discovery Log Page Entry data structure in the NVM Express Base Specification.

3.6.2.12 Kickstart Discovery Response PDU (KDResp)

Figure 37: Kickstart Discovery Response PDU (KDResp)

<table>
<tr><th colspan="2">Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td colspan="2">00</td><td rowspan="5">CH</td><td>PDU Type (PDU-Type): 0Bh</td></tr>
<tr><td colspan="2">01</td><td>Flags (FLAGS):
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
<tr><td>1</td><td>PDU Data Digest Flag (DDGSTF): If this bit is set to ‘1’, then the DDGST field follows the PDU Data and contains a valid value. If this bit is cleared to ‘0’, then the DDGST field is not present.</td></tr>
<tr><td>0</td><td>PDU Header Digest Flag (HDGSTF): If this bit is set to ‘1’, then the HDGST field follows the PDU Header and contains a valid value. If this bit is cleared to ‘0’, then the HDGST field is not present.</td></tr>
</table></td></tr>
<tr><td colspan="2">02</td><td>Header Length (HLEN): Fixed length of 10 bytes (i.e., Ah).</td></tr>
<tr><td colspan="2">03</td><td>PDU Data Offset (PDO): Obsolete</td></tr>
<tr><td colspan="2">07:04</td><td>PDU Length (PLEN): Fixed length of 274 bytes (i.e., 112h).</td></tr>
<tr><td colspan="2">08</td><td rowspan="2">PSH</td><td>Kickstart Status (KSSTAT):
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
<tr><td>1</td><td>Failure (FAILURE): If this bit is set to ‘1’, then the CDC shall not perform a pull registration due to the reason indicated in the Failure Reason (FAILRSN) field.</td></tr>
<tr><td>0</td><td>Success (SUCESS): If this bit is set to ‘1’, then the CDC shall perform a pull registration.</td></tr>
</table></td></tr>
<tr><td colspan="2">09</td><td>Failure Reason (FAILRSN):
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved</td></tr>
<tr><td>6</td><td>Insufficient Discovery Resources (IDR)</td></tr>
<tr><td>5</td><td>TRSVCID does not match TRTYPE (TDNMT)</td></tr>
<tr><td>4</td><td>TRADDR does not match ADRFAM (TDNMA)</td></tr>
<tr><td>3</td><td>Invalid ADRFAM (INVA)</td></tr>
<tr><td>2</td><td>Invalid TRTYPE (INVT)</td></tr>
<tr><td>1</td><td>No additional information (NAI)</td></tr>
<tr><td>0</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>HDGSTF=1</td><td>HDGSTF=0</td><td colspan="2"></td></tr>
<tr><td>13:10</td><td>Not present</td><td>HDGST</td><td>PDU Header Digest (HDGST): If the HDGSTF bit is set to ‘1’ in the FLAGS field, this field is valid and contains the header digest (refer to section 3.3.1.1). If the HDGSTF bit is cleared to ‘0’, then this field is not present.</td></tr>
</table>

Figure 37: Kickstart Discovery Response PDU (KDResp)

Bytes		PDU Section	Description
269:14	265:10	DATA	**CDC NVMe Qualified Name (CDCNQN):** This field indicates the NVMe Qualified Name (NQN) that uniquely identifies the CDC. Refer to the NVMe Qualified Names section in the NVM Express Base Specification for the formatting requirements of NQNs.
DDGSTF=1	DDGSTF=0		
273:270	Not present	DDGST	**PDU Data Digest (DDGST):** If the DDGSTF bit is set to '1' in the FLAGS field, this field is valid and contains the data digest (refer to section 3.3.1.1). If the DDGSTF bit is cleared to '0', then this field is not present.