NVM Express^{TM} over Fabrics

Revision 1.1

October 22, 2019

Please send comments to info@nvmexpress.org

NVM Express[TM] over Fabrics revision 1.1 is available for download at http://nvmexpress.org. NVM Express over Fabrics revision 1.1 incorporates NVM Express[TM] over Fabrics revision 1.0, ratified on June 5, 2016, along with NVM Express[TM] revision 1.0 ECNs 001 to ECN 006, TP 8000, TP 8001, TP 8002, TP 8005, and TP 8008 (refer to https://nvmexpress.org/changes-in-nvme-over-fabrics-revision-1-1 for details). Applied the NVM Express trademark and logo usage guidelines.

SPECIFICATION DISCLAIMER

LEGAL NOTICE:

This NVM Express over Fabrics revision 1.1 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 over Fabrics revision 1.1 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: **"© 2007 to 2019 NVM Express, Inc. ALL RIGHTS RESERVED."** When making any such citations or references to this document you are not permitted to revise, alter, modify, make any derivatives of, or otherwise amend the referenced portion of this document in any way without the prior express written permission of NVM Express, Inc. Nothing contained in this document shall be deemed as granting you any kind of license to implement or use this document or the specification described therein, or any of its contents, either expressly or impliedly, or to any intellectual property owned or controlled by NVM Express, Inc., including, without limitation, any trademarks of NVM Express, Inc.

LEGAL DISCLAIMER:

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

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

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

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

Table of Contents

1 Introduction

NVM Express™ (NVMe™) Base Specification revision 1.4 and prior revisions define a register level interface for host software to communicate with a non-volatile memory subsystem over PCI Express™ (NVMe™ over PCIe™). This specification defines extensions to NVMe that enable operation over other interconnects (NVMe™ over Fabrics). The NVM Express Base Specification revision 1.4 is referred to as the NVMe Base specification.

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

1.1 Scope

This specification defines extensions to the NVMe interface that enable operation over a fabric other than PCI Express (PCIe). This specification supplements the NVMe Base specification.

1.2 Outside of Scope

Functionality that is applicable only to NVMe over PCIe or to both NVMe over PCIe and NVMe over Fabrics is defined in the NVMe Base specification.

This specification defines requirements and behaviors that are implementation agnostic. The implementation of these requirements and behaviors are outside the scope of this specification. For example, an NVM subsystem that follows this specification may be implemented by an SSD that attaches directly to a fabric, a device that translates between a fabric and a PCIe NVMe SSD, or software running on a general purpose server.

Other published specifications referred to in this document, even if required for compliance, are outside the scope of this specification; this includes published specifications for fabrics and other technologies referred to by this document or any NVMe Transport binding specification.

1.3 Conventions

NVMe over Fabrics definition conforms to the byte, word, and dword relationships defined in section 1.8 of the NVMe Base specification. This includes specifying all data in little endian format unless otherwise noted.

A Discovery controller is a type of controller that supports minimal functionality required for the Discovery Log Page to be retrieved. The use of the term “controller” in the NVMe Base specification and in this document refers to requirements that apply to all controllers or specifically to a controller that may expose namespaces. When a requirement applies to only a Discovery controller, the specification includes the complete term “Discovery controller”.

1.4 Definitions

1.4.1 association

An exclusive communication relationship between a particular controller and a particular host that encompasses the Admin Queue and all I/O Queues of that controller.

1.4.2 authentication commands

Used to refer to Fabrics Authentication Send or Authentication Receive commands.

1.4.3 capsule

An NVMe unit of information exchange used in NVMe over Fabrics. A capsule contains a command or response and may optionally contain command/response data and SGLs.

1.4.4 Discovery controller

A controller that supports minimal functionality and only implements the required features that allow the Discovery Log Page to be retrieved. A Discovery controller does not implement I/O Queues or expose namespaces.

1.4.5 Discovery Service

An NVM subsystem that supports Discovery controllers only. A Discovery Service shall not support a controller that exposes namespaces.

1.4.6 dynamic controller

The controller is allocated on demand with no state (e.g., Feature settings) preserved from prior associations.

1.4.7 fabric (network fabric)

A network topology in which nodes pass data to each other.

1.4.8 NVMe Transport

A protocol layer that provides reliable delivery of data, commands, and responses between a host and an NVM subsystem. The NVMe Transport layer is layered on top of the fabric. It is independent of the fabric physical interconnect and low level fabric protocol layers.

1.4.9 NVMe Transport binding specification

A specification of reliable delivery of data, commands, and responses between a host and an NVM subsystem for an NVMe Transport. The binding may exclude or restrict functionality based on the NVMe Transport’s capabilities.

1.4.10 port (NVM subsystem port)

An NVMe over Fabrics protocol interface between an NVM subsystem and a fabric. An NVM subsystem port is a collection of one or more physical fabric interfaces that together act as a single interface.

1.4.11 Port ID

A 16-bit identifier that is associated with an NVM subsystem port.

1.4.12 property

The generalization of memory mapped controller registers defined for NVMe over PCIe. Properties are used to configure low level controller attributes and obtain low level controller status.

1.4.13 static controller

The controller is pre-existing with a specific Controller ID and its state (e.g., Feature settings) is preserved from prior associations.

1.4.14 command submission

A command is submitted when a host adds a capsule to a Submission Queue.

1.4.15 physical fabric interface (physical ports)

A physical connection between an NVM subsystem and a fabric.

1.5 Theory of Operation

NVMe over Fabrics builds on the architecture, command sets, and queueing interface defined in the NVMe Base specification. A central goal of NVMe over Fabrics is to maintain consistency with the base definition and only deviate where necessary in order to support general fabrics. Extensions defined by NVMe over Fabrics include:

- The use of capsules for commands, responses, and optionally for data transfers;
- The extension of Scatter Gather Lists (SGLs) to support in-capsule data as well as NVMe Transports that utilize a key/offset memory addressing architecture;
- Extensions to the queueing model that enable the use of underlying capabilities provided by some NVMe Transports;
- A method for a host to establish a connection to a controller's Admin or I/O Queue within a specific NVM subsystem. This method includes an authentication procedure that may be used to authenticate the host and controller identities;
- The generalization of controller PCI Express Memory Mapped I/O (MMIO) registers to properties that may be accessed by a host over any type of NVMe Transport; and
- A discovery mechanism for a host to determine which NVM subsystems may be accessed.

NVMe over Fabrics has the following differences from the NVMe Base specification:

- There is a one-to-one mapping between I/O Submission Queues and I/O Completion Queues. NVMe over Fabrics does not support multiple I/O Submission Queues being mapped to a single I/O Completion Queue;
- NVMe over Fabrics does not define an interrupt mechanism that allows a controller to generate a host interrupt. It is the responsibility of the host fabric interface (e.g., Host Bus Adapter) to generate host interrupts;
- NVMe over Fabrics does not use the Create I/O Completion Queue, Create I/O Submission Queue, Delete I/O Completion Queue, and Delete I/O Submission Queue commands. NVMe over Fabrics does not use the Admin Submission Queue Base Address (ASQ), Admin Completion Queue Base Address (ACQ), and Admin Queue Attributes (AQA) properties (i.e., registers in PCI Express). Queues are created using the Connect command (refer to section 3.3);
- NVMe over Fabrics uses the Disconnect command (refer to section 3.4) to delete an I/O Submission Queue and corresponding I/O Completion Queue;
- Metadata, if supported, shall be transferred as a contiguous part of the logical block. NVMe over Fabrics does not support transferring metadata from a separate buffer;
- NVMe over Fabrics does not support PRPs but requires use of SGLs for Admin, I/O, and Fabrics commands. This differs from NVMe over PCIe where SGLs are not supported for Admin commands and are optional for I/O commands;
- NVMe over Fabrics does not support Completion Queue flow control. This requires that the host ensures there are available Completion Queue slots before submitting new commands; and
- NVMe over Fabrics allows Submission Queue flow control to be disabled if the host and controller agree to disable it. If Submission Queue flow control is disabled, the host is required to ensure that there are available Submission Queue slots before submitting new commands.

While differences exist between NVMe over Fabrics and NVMe over PCIe implementations, both implement the same architecture and command sets. The shared characteristics include:

- A scalable host controller interface;
- Optimized command submission and completion paths;
- Support for parallel operation that supports up to (64K – 1) I/O queues per controller with up to (64K – 1) outstanding commands per I/O queue;
- Support for Enterprise capabilities such as end-to-end data protection;
- Robust error reporting and management capabilities;
- Priority associated with each I/O queue with a well-defined controller queue arbitration mechanism;

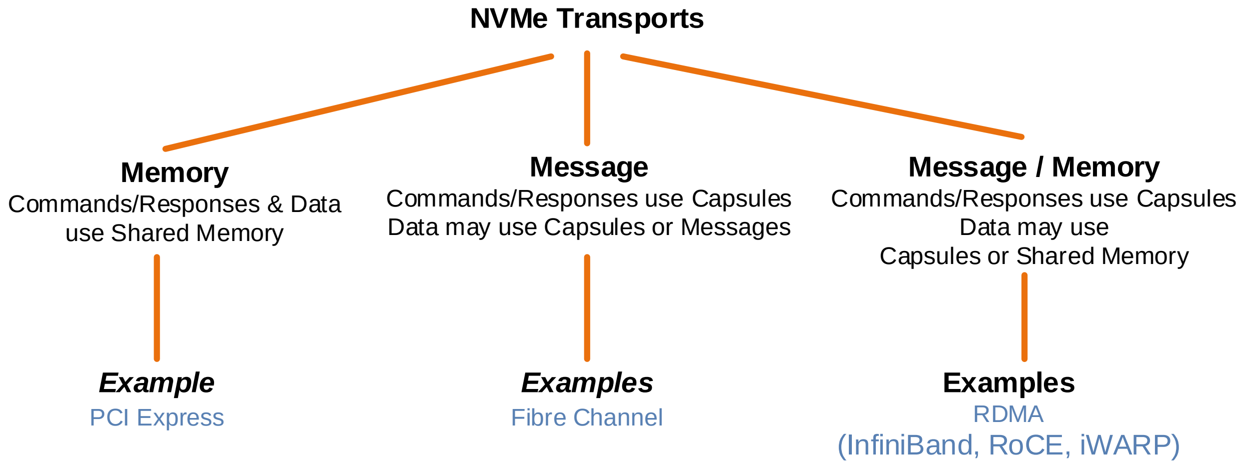
- Efficient and streamlined command set;
- Support for multiple namespaces and namespace management; and
- Support for multi-path I/O and namespace sharing.

1.5.1 Fabrics and Transports

NVMe over Fabrics requires the underlying NVMe Transport to provide reliable NVMe command and data delivery. An NVMe Transport is an abstract protocol layer independent of any physical interconnect properties. A taxonomy of NVMe Transports along with examples is shown in Figure 1. An NVMe Transport may expose a memory model, a message model, or a combination of the two. A memory model is one in which commands, responses and data are transferred between fabric nodes by performing explicit memory read and write operations while a message model is one in which only messages containing command capsules, response capsules, and data are sent between fabric nodes. A message/memory model uses a combination of messages and explicit memory read and write operations to transfer command capsules, response capsules and data between fabric nodes. Data may optionally be included in command capsules and response capsules.

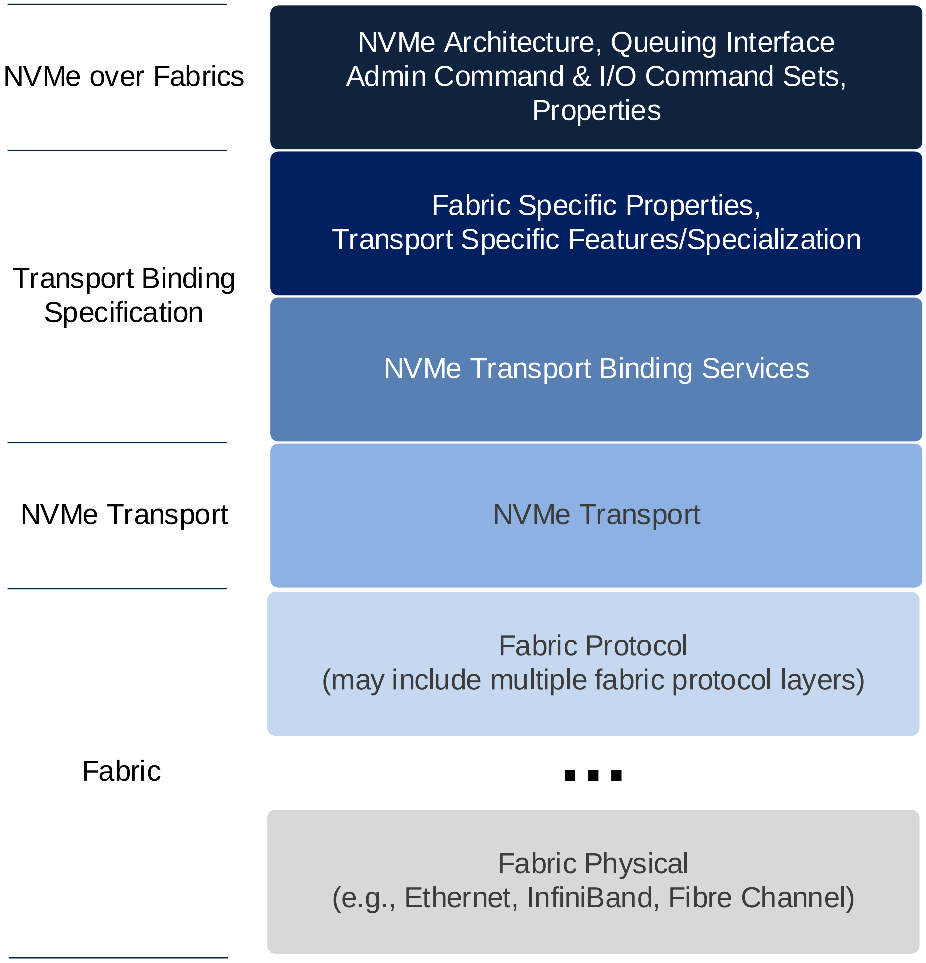
The only memory model NVMe Transport supported by NVMe is PCI Express, as defined in the NVMe Base specification. Message model and message/memory model NVMe Transports are specified in this document.

Figure 1: Taxonomy of Transports


NVMe Transports
Memory
Commands/Responses & Data use Shared Memory
Message
Commands/Responses use Capsules
Data may use Capsules or Messages
Message / Memory
Commands/Responses use Capsules
Data may use
Capsules or Shared Memory
Example
PCI Express
Examples
Fibre Channel
Examples
RDMA
(InfiniBand, RoCE, iWARP)

NVMe over Fabrics utilizes the protocol layering shown in Figure 2. This specification defines core aspects of the architecture that are independent of the NVMe Transport. An NVMe Transport binding specification is used to describe any NVMe Transport specific specialization as well as how the services required by the NVMe interface are mapped onto the corresponding NVMe Transport. The native fabric communication services and other functionality used by the NVMe interface and NVMe Transports (e.g., the Fabric Protocol and Fabric Physical layers in Figure 2) are outside the scope of this specification.

Figure 2: NVMe over Fabrics Layering


NVMe over Fabrics
NVMe Architecture, Queuing Interface Admin Command & I/O Command Sets, Properties
Transport Binding Specification
Fabric Specific Properties, Transport Specific Features/Specialization
NVMe Transport Binding Services
NVMe Transport
NVMe Transport
Fabric Protocol (may include multiple fabric protocol layers)
Fabric
Fabric Physical (e.g., Ethernet, InfiniBand, Fibre Channel)

1.5.2 NVM Subsystem

NVMe over Fabrics builds on the NVM subsystem architecture defined in the NVMe Base specification. An NVM subsystem presents a collection of one to (64K - 16) controllers which are used to access namespaces. The controllers may be associated with hosts through one to 64K NVM subsystem ports.

An NVM subsystem port (port) is a protocol interface between an NVM subsystem and a fabric. An NVM subsystem port is a collection of one or more physical fabric interfaces that together act as a single protocol interface. When link aggregation (e.g., Ethernet) is used, the physical ports for the group of aggregated links constitute a single NVM subsystem port.

An NVM subsystem contains one or more NVM subsystem ports.

Each NVM subsystem port has a 16-bit port identifier (Port ID). An NVM subsystem port is identified by the NVM Subsystem NVMe Qualified Name (NQN) (refer to section 7.9 in the NVMe Base specification) and Port ID. The ports of an NVM subsystem may support different NVMe Transports. An NVM subsystem port may support multiple NVMe Transports if more than one NVMe Transport binding specifications exist for the underlying fabric (e.g., an NVM subsystem port identified by a Port ID may support both iWARP and RoCE). An NVM subsystem implementation may bind specific controllers to specific ports or allow the flexible allocation of controllers between ports, however, once connected, each specific controller is bound to a single NVM subsystem port.

A controller is associated with exactly one host at a time. NVMe over Fabrics allows multiple hosts to connect to different controllers in the NVM subsystem through the same port. All other aspects of NVMe over Fabrics multi-path I/O and namespace sharing are equivalent to that defined in the NVMe Base specification.

An NVM subsystem may optionally include a non-volatile storage medium, and an interface between the controller(s) of the NVM subsystem and the non-volatile storage medium. Controllers expose this non-volatile storage medium to hosts through namespaces. An NVM subsystem is not required to have the same namespaces attached to all controllers. An NVM subsystem may support controllers that expose namespaces or Discovery controllers; but an NVM subsystem does not expose a mix of controller types. A Discovery Service is an NVM subsystem that exposes Discovery controllers only.

An association is established between a host and a controller when the host connects to a controller's Admin Queue using the Fabrics Connect command (refer to section 3.3). Within the Connect command, the host specifies the Host NQN, NVM Subsystem NQN, Host Identifier, and may request a specific Controller ID or may request a connection to any available controller. A controller has only one association at a time.

The NVM subsystem may support a dynamic or static controller model. In a dynamic controller model, the controller is allocated by the NVM subsystem on demand with no state (e.g., Feature settings) preserved from prior associations. In a static controller model, the host may request a particular controller based on the Controller ID where state (e.g., Feature settings) is preserved from prior associations. Refer to section 4.2.

While an association exists between a host and a controller, only that host may establish connections with I/O Queues of that controller by presenting the same Host NQN, Host Identifier, NVM Subsystem NQN and Controller ID in subsequent Connect command(s) using the same NVM subsystem port, NVMe Transport type, and NVMe Transport address.

An association between a host and controller is terminated if:

- the controller is shutdown as described in section 4.6;
- a Controller Level Reset occurs;
- the NVMe Transport connection is lost between the host and controller for the Admin Queue; or
- an NVMe Transport connection is lost between the host and controller for any I/O Queue and the host or controller does not support individual I/O Queue deletion (refer to section 1.5.9).

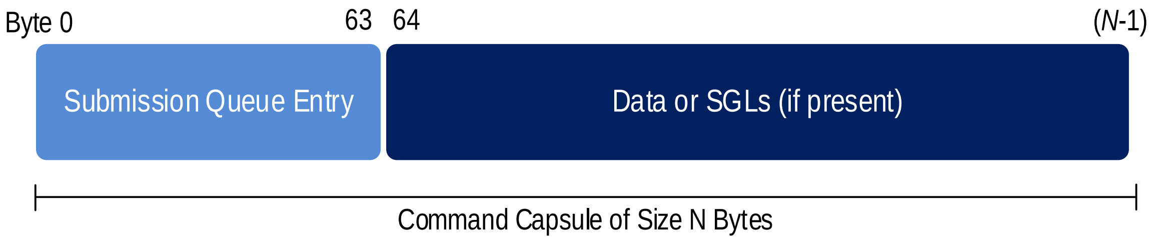
There is no explicit NVMe command that breaks the NVMe Transport association between a host and controller. The Disconnect command (refer to section 3.4) provides a method to delete an NVMe I/O Queue (refer to section 1.5.9). While a controller is associated with a host, that controller is busy, and no other associations may be made with that controller.

1.5.3 Capsules and Data Transfer

A capsule is an NVMe unit of information exchange used in NVMe over Fabrics. A capsule may be classified as a command capsule or a response capsule. A command capsule contains a command (formatted as a Submission Queue Entry (SQE)) and may optionally include SGLs or data. A response capsule contains a response (formatted as a Completion Queue Entry (CQE)) and may optionally include data. Data refers to any data transferred at an NVMe layer between a host and an NVM subsystem (e.g., logical block data or a data structure associated with a command). A capsule is independent of any underlying NVMe Transport unit (e.g., packet, message, or frame and associated headers and footers) and may consist of multiple such units.

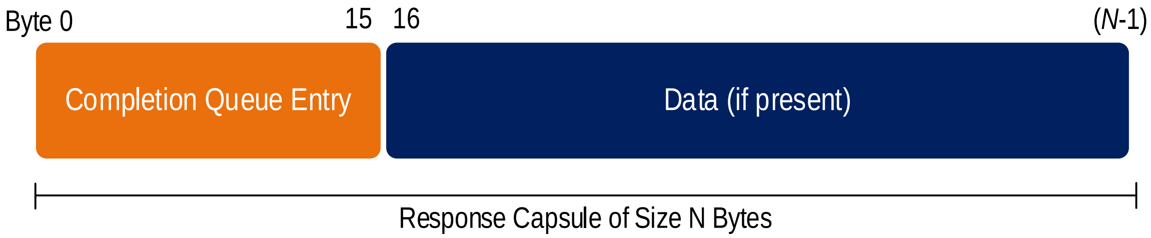
Command capsules are transferred from a host to an NVM subsystem. The SQE contains an Admin command, an I/O command, or a Fabrics command. The minimum size of a command capsule is NVMe Transport binding specific, but shall be at least 64B in size. The maximum size of a command capsule is NVMe Transport binding specific. The format of a command capsule is shown in Figure 3.

Figure 3: Command Capsule Format



Response capsules are transferred from an NVM subsystem to a host. The CQE is associated with a previously issued Admin command, I/O command, or Fabrics command. The size of a response capsule is NVMe Transport binding specific, but shall be at least 16B in size. The maximum size of a response capsule is NVMe Transport binding specific. The format of a response capsule is shown in Figure 4.

Figure 4: Response Capsule Format



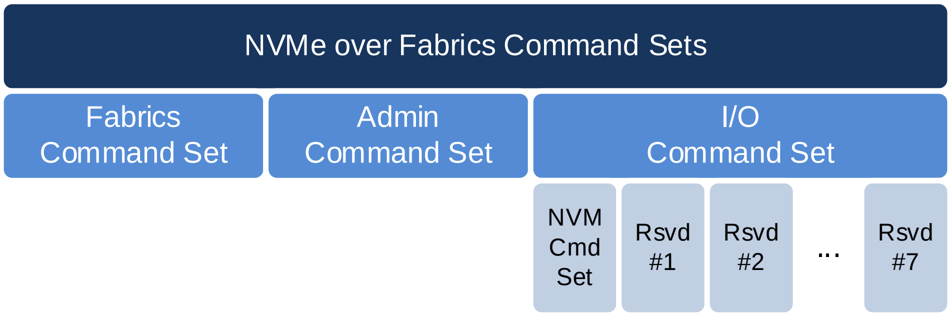
Message and Message/Memory model NVMe Transports require all SGLs sent from the host to the controller be transferred within the command. Message and Message/Memory model NVMe Transports may optionally support the transfer of a portion or all data within the command and response capsules.

NVMe over PCIe commands use PRPs and optionally SGLs to specify data transfer memory regions. NVMe over Fabrics requires SGLs for all commands (Fabrics, Admin, and I/O). An SGL may specify the placement of data within a capsule or the information required to transfer data using an NVMe Transport specific data transfer mechanism (e.g., via memory transfers as in RDMA). The NVMe Transport binding specification defines the SGLs used by a particular NVMe Transport and any capsule SGL and data placement restrictions.

1.5.4 Command Sets

As shown in Figure 5, NVMe over Fabrics supports three Command Sets. The Fabrics Command Set is NVMe over Fabrics specific. The Admin and I/O Command Sets are defined by the NVMe Base specification.

Figure 5: NVMe over Fabrics Command Sets



Fabrics Command Set commands are used for operations specific to NVMe over Fabrics including establishing a connection, NVMe in-band authentication, and to get or set a property. All Fabrics commands may be submitted on the Admin Queue and some Fabrics commands may also be submitted on an I/O Queue. Unlike Admin and I/O commands, Fabrics commands are processed by a controller regardless of whether the controller is enabled (i.e., regardless of the state of CC.EN).

This specification assumes an SQE size of 64B and CQE size of 16B as defined by the NVM Command Set.

1.5.5 Properties

Properties are the NVMe over Fabrics analog of memory mapped NVMe controller registers defined for NVMe over PCIe. Properties are used to configure a subset of controller attributes and obtain a subset of status.

A host may obtain the value of a property by using the Property Get command and may modify the value of a property by using the Property Set command that are part of the Fabrics Command Set. Properties are only accessible via the Admin Queue.

Some controller registers or fields are specific to PCIe functionality (e.g., Submission Queue Doorbell registers) and are not used in NVMe over Fabrics. As a result, a subset of controller registers and fields defined in the NVMe Base specification map to properties in NVMe over Fabrics (refer to section 3.6.1).

1.5.6 Discovery

NVMe over Fabrics defines a discovery mechanism that a host may use to determine the NVM subsystems the host may access. A Discovery controller supports minimal functionality for providing Discovery Logs. A Discovery controller may support notification of Discovery Log changes using Asynchronous Events. A Discovery controller does not implement I/O Queues or expose namespaces. A Discovery Service is an NVM subsystem that exposes only Discovery controllers.

The Discovery Log Page provided by a Discovery controller contains one or more entries. Each entry specifies information necessary for the host to connect to an NVM subsystem via an NVMe Transport. An entry may specify an NVM subsystem that exposes namespaces that the host may access, or a referral to another Discovery Service. The maximum referral depth supported is eight levels.

The method that a host uses to obtain the information necessary to connect to the initial Discovery Service is implementation specific. This information may be determined using a host configuration file, a hypervisor or OS property or some other mechanism.

1.5.7 Connection

NVMe over Fabrics uses the Connect command (refer to section 3.3) to create controller Admin or I/O Queues. The creation of an Admin Queue establishes an association between a host and the corresponding controller. NVMe over Fabrics does not support the Admin Submission Queue Base Address (ASQ), Admin Completion Queue Base Address (ACQ), and Admin Queue Attributes (AQA) properties as all information necessary to establish an Admin Queue is contained in the Connect command. NVMe over Fabrics does not support the Admin commands associated with I/O Queue creation and deletion (Create I/O Completion Queue, Create I/O Submission Queue, Delete I/O Completion Queue, Delete I/O Submission Queue) defined in the NVMe Base specification.

An NVMe Transport connection is established between a host and an NVM subsystem prior to the transfer of any capsules or data. The mechanism used to establish an NVMe Transport connection is NVMe Transport specific and defined by the corresponding NVMe Transport binding specification. The NVMe Transport may require a separate NVMe Transport connection for each Admin or I/O Queue or may utilize the same NVMe Transport connection for all Admin and I/O Queues associated with a particular controller. An NVMe Transport may also require that NVMe layer information be passed between the host and controller in the process of establishing an NVMe Transport connection (e.g., exchange queue size to appropriately size send and receive buffers).

The Connect command specifies the Queue ID and type (Admin or I/O), the size of the Submission and Completion Queues, queue attributes, Host NQN, NVM Subsystem NQN, and Host Identifier. The Connect command may specify a particular controller if the NVM subsystem supports a static controller model. The Connect response indicates whether the connection was successfully established as well as whether NVMe in-band authentication is required.

The Connect command is submitted to the same Admin Queue or I/O Queue that the Connect command creates. The underlying NVMe Transport connection that is used for that queue is created first and the Connect command and response capsules are sent over that NVMe Transport connection. The Connect command shall be sent once to a queue.

When a Connect command successfully completes, the corresponding Submission and Completion Queues are created. If NVMe in-band authentication is required as indicated in the Connect response, then NVMe in-band authentication shall be performed before the queues may be used to perform other Fabrics, Admin, or I/O commands. Once a Connect command for an Admin Queue has completed successfully (and NVMe in-band authentication if required has succeeded), only Fabrics commands may be submitted until the controller is ready (CSTS.RDY = 1). Both Fabrics commands and Admin commands may be submitted to the Admin Queue while the controller is ready. A Connect command for an I/O Queue may be submitted only after the controller is ready. Once a Connect command for an I/O Queue has completed successfully (and NVMe in-band authentication, if required, has succeeded), I/O commands may be submitted to the queue.

The Connect response contains the controller ID allocated to the host. All subsequent Connect commands that create an I/O Queue with that controller shall be from the same host, utilize the same NVMe Transport, and have the same Host Identifier, Host NQN, and NVM Subsystem NQN; if any of these conditions are not met, then the Connect command fails.

1.5.8 Authentication

NVMe over Fabrics supports both fabric secure channel (that includes authentication) and NVMe in-band authentication. An NVM subsystem may require a host to use fabric secure channel, NVMe in-band authentication, or both. The Discovery Service indicates if fabric secure channel shall be used for an NVM subsystem. The Connect response indicates if NVMe in-band authentication shall be used with that controller.

A controller associated with an NVM subsystem that requires a fabric secure channel shall not accept any commands (i.e., Fabrics commands, Admin commands, or I/O commands) on an NVMe Transport until a secure channel is established. Following a Connect command, a controller that requires NVMe in-band authentication shall not accept any commands on the queue created by that Connect command other than authentication commands until NVMe in-band authentication has completed. Refer to section 6.

1.5.9 I/O Queue Deletion

NVMe over Fabrics deletes an individual I/O Queue and may delete the associated NVMe Transport connection as a result of:

- the exchange of a Disconnect command and response (refer to section 3.4) between a host and controller; or
- the detection and processing of a transport error on an NVMe Transport connection.

The host indicates support for the deletion of an individual I/O Queue by setting bit 3 to '1' in the CATTR field in the Connect command (refer to Figure 19) used to create the Admin Queue. The controller indicates support for the deletion of an individual I/O Queue by setting bit 0 to '1' in the OFCS field in the Identify Controller Attributes region of the Identify Controller data structure (refer to Figure 30).

If both the host and the controller support deletion of an individual I/O Queue, then the termination of an individual I/O Queue impacts only that I/O Queue (i.e., the association and all other I/O Queues and their associated NVMe Transport connections are not impacted). If either the host or the controller does not support deletion of an individual I/O Queue, then the deletion of an individual I/O Queue or the termination of an NVMe Transport connection causes the association to be terminated.

NVMe over Fabrics uses the Disconnect command to delete an Individual I/O Queue. This command is sent on the I/O Submission Queue to be deleted and affects only that I/O Submission Queue and its associated I/O Completion Queue (i.e., other I/O Queues are not affected). To delete an I/O Queue, the NVMe Transport connection for that I/O Queue is used. If all Queues associated with an NVMe transport connection are deleted, then the NVMe Transport connection may be deleted after completion of the Disconnect command. Actions necessary to delete the NVMe transport connection are transport specific. The association between the host and the controller is not affected.

If a Disconnect command returns a status other than success, the host may delete an I/O Queue using other methods including:

- waiting a vendor specific amount of time and retry the Disconnect command;
- deleting the NVMe Transport connection (note: this may impact other I/O Queues);
- performing a Controller Level Reset (note: this impacts other I/O Queues); or
- ending the host to controller association.

If the transport requires a separate NVMe Transport connection for each Admin and I/O Queue (refer to section 1.5.7), then the host should not delete an NVMe Transport connection until after:

- a Disconnect command has been submitted to the I/O Submission Queue; and
- the response for that Disconnect command has been received by the host on the corresponding I/O Completion Queue or a vendor specific timeout (refer to section 7.1.2) has occurred while waiting for that response.

If the transport requires a separate NVMe Transport connection for each Admin and I/O Queue, then the controller should not delete an NVMe Transport connection until after:

- a Disconnect command has been received on the I/O Submission Queue and processed by the controller;
- the responses for commands received by the controller on that I/O Submission Queue prior to receiving the Disconnect command have been sent to the host on the corresponding I/O Completion Queue; and
- the resulting response for that Disconnect command has been sent to the host on the corresponding I/O Completion queue (i.e., this response is the last response sent). It is recommended that the controller delay destroying the NVMe Transport connection to allow time for the Disconnect command response to be received by the host (e.g., a transport specific event occurs or a transport specific time period elapses).

If the transport utilizes the same NVMe Transport connection for all Admin and I/O Queues associated with a particular controller (refer to section 1.5.7), then the deletion of an individual I/O Queue has no impact on the NVMe Transport connection.

A Disconnect command is the last I/O Submission Queue entry processed by the controller for an I/O Queue. Controller processing of the Disconnect command completes or aborts all commands on the I/O Queue on which the Disconnect command was received. The controller determines whether to complete or abort each of those commands.

The response to the Disconnect command is the last I/O Completion Queue entry processed by the host for an I/O Queue. To avoid command aborts the host should wait for outstanding commands on an I/O Queue to complete before sending the Disconnect command.

If the controller detects an NVMe Transport connection loss, then the controller shall stop processing all commands received on I/O Queues associated with that NVMe Transport connection. Until the controller detects an NVMe Transport connection loss or sends a successful completion for a Disconnect command, outstanding commands may continue being processed by the controller.

If the host detects an NVMe Transport connection loss before the responses are received for all outstanding commands submitted to the associated I/O Queue, then there is no further information available to the host about the state of those commands (e.g., each individual outstanding command may have been completed or aborted by the controller).

If an NVMe Transport connection is lost as a result of an NVMe Transport error, then before performing recovery actions related to commands sent on I/O queues associated with that NVMe Transport connection, the host should wait for at least the longer of:

- the NVMe Keep Alive timeout; or
- the underlying fabric transport timeout, if any.

2 Capsules and Data Transfers

This section describes capsules and data transfer mechanisms. These mechanisms are used for Fabrics commands, Admin commands, and I/O commands.

A capsule is an NVMe unit of information exchanged between a host and a controller. A capsule may contain commands, responses, SGLs, and/or data. The data may include logical block data and metadata that is transferred as a contiguous part of the logical block, and data structures associated with the command.

The capsule size for the Admin Queue commands and responses is fixed and defined in the NVMe Transport binding specification. The controller indicates in the Identify Controller data structure the capsule command and response sizes that the host shall use with I/O commands.

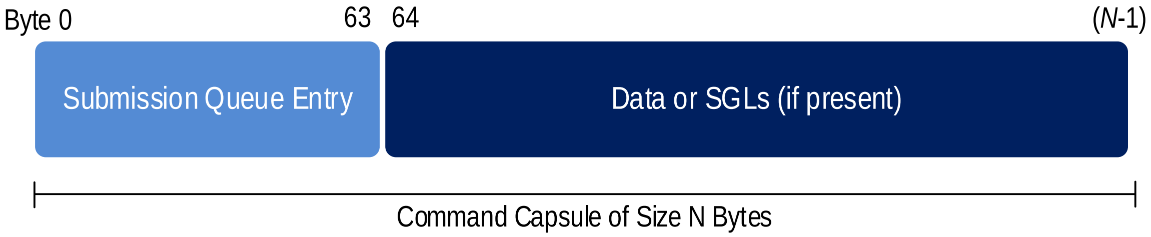
The controller shall support SGL based data transfers for commands on both the Admin Queue and I/O Queues. Data may be transferred within the capsule or through memory transactions based on the underlying NVMe Transport as indicated in the SGL descriptors associated with the command capsule. The SGL types supported by an NVMe Transport are specified in the NVMe Transport binding specification.

The value of unused and not reserved capsule fields (e.g., the capsule is larger than the command / response and associated data) is undefined and shall not be interpreted by the recipient.

2.1 Command Capsules

A command capsule is sent from a host to a controller. It contains a Submission Queue Entry (SQE) and may optionally contain data or SGLs. The SQE is 64 bytes in size and contains the Admin command, I/O command, or Fabrics command to be executed.

Figure 6: Command Capsule



The Command Identifier field in the SQE shall be unique among all outstanding commands associated with that queue. If there is data or additional SGLs to be transferred within the capsule, then the SGL descriptor in the SQE contains a Data Block, Segment Descriptor, or Last Segment Descriptor specifying an appropriate Offset address. The definition for the Submission Queue Entry when the command is a Fabrics command is defined in Figure 7. The definition for the Submission Queue Entry when the command is an Admin or I/O command is defined in section 4.2 of the NVMe Base specification, where the Metadata Pointer field is reserved.

Figure 7: Fabrics Command Capsule – Submission Queue Entry Format

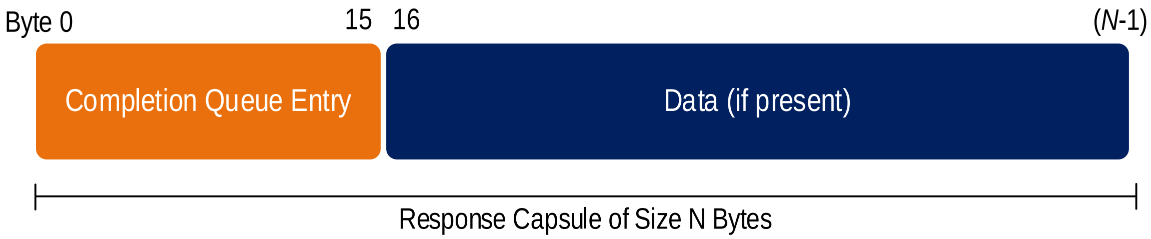
Bytes	Description
00	**Opcode (OPC):** Set to 7Fh to indicate a Fabrics command.
01	Reserved
03:02	**Command Identifier (CID):** This field specifies a unique identifier for the command. The identifier shall be unique among all outstanding commands associated with a particular queue.
04	**Fabrics Command Type (FCTYPE):** This field specifies the Fabrics command transferred in the capsule. The Fabrics command types are defined in Figure 14. If this field is set to a reserved value, the command should be aborted with a status code of Invalid Field in Command.
39:05	Reserved
63:40	**Fabrics Command Type Specific:** This field is Fabrics command type specific.

2.2 Response Capsules

A response capsule is sent from the NVM subsystem to the host. It contains a Completion Queue Entry (CQE) and may optionally contain data. The CQE is the completion entry associated with a previously issued command capsule.

If a command requests data and the SGL in the associated command capsule specifies a Data Block descriptor with an Offset, the data is included in the response capsule. If the SGL(s) in the command capsule specify a region in host memory, then data is transferred via memory transactions.

Figure 8: Response Capsule



The Completion Queue Entry is 16 bytes in size and contains a two byte status field.

The definition for the Completion Queue Entry for a Fabrics command is defined in Figure 9. The definition for the Completion Queue Entry when the command is an Admin or I/O command is defined in section 4.6 of the NVMe Base specification, where the SQ Identifier and Phase Tag fields are reserved because they are not used in NVMe over Fabrics. Use of the SQ Head Pointer (SQHD) field depends on whether SQ flow control is disabled for the queue pair, refer to section 2.4 and to section 3.3.

Figure 9: Fabrics Response Capsule – Completion Queue Entry Format

Bytes	Description
07:00	The definition of this field is Fabrics response type specific.
09:08	**SQ Head Pointer (SQHD):** Indicates the current Submission Queue Head pointer for the associated Submission Queue[1].
11:10	Reserved
13:12	**Command Identifier (CID):** Indicates the identifier of the command that is being completed.

Figure 9: Fabrics Response Capsule – Completion Queue Entry Format

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>15:14</td><td><b>Status (STS):</b> Specifies status for the associated Fabrics command.
<table><tr><th>Bits</th><th>Definition</th></tr><tr><td>15:01</td><td>Status Field as defined in section 4.6.1 of the NVMe Base specification.</td></tr><tr><td>00</td><td>Reserved</td></tr></table></td></tr>
<tr><td colspan="2">NOTES:
1. The SQHD field is reserved if SQ flow control is disabled for the queue pair, refer to section 2.4 and to section 3.3.</td></tr>
</table>

2.2.1 Status Values

Fabrics commands use the status for commands defined in the NVMe Base specification. The Status Field defined in section 4.6.1 defines the status for Fabrics, Admin, and I/O commands.

Fabrics commands use an allocation of command specific status values from 80h to BFh (refer to Figure 126 of the NVMe Base specification). Refer to Figure 10.

Figure 10: Fabrics Command Specific Status Values

Value	Description	Commands Affected
80h	**Incompatible Format:** The NVM subsystem does not support the record format specified by the host.	Connect, Disconnect
81h	**Controller Busy:** The controller is already associated with a host (Connect command). This value is also returned if there is no available controller (Connect command). The controller is not able to disconnect the I/O Queue at the current time (Disconnect command).	Connect, Disconnect
82h	**Connect Invalid Parameters:** One or more of the command parameters (e.g., Host NQN, Subsystem NQN, Host Identifier, Controller ID, Queue ID) specified are not valid.	Connect
83h	**Connect Restart Discovery:** The NVM subsystem requested is not available. The host should restart the discovery process.	Connect
84h	**Connect Invalid Host:** The host is not allowed to establish an association to any controller in the NVM subsystem or the host is not allowed to establish an association to the specified controller.	Connect
85h	**Invalid Queue Type:** The command was sent on the wrong queue type (e.g., a Disconnect command was sent on the Admin queue).	Disconnect
86h to 8Fh	Reserved	
90h	**Discover Restart:** The snapshot of the records is now invalid or out of date. The host should re-read the Discovery Log Page.	Get Log Page
91h	**Authentication Required:** NVMe in-band authentication is required and the queue has not yet been authenticated.	NOTE 1
92h to AFh	Reserved	
B0h to BFh	**Transport Specific:** The status values in this range are NVMe Transport specific. Refer to the appropriate NVMe Transport binding specification for the definition of these status values.	
NOTES: 1. All commands other than Connect, Authenticate Send, and Authenticate Receive.		

2.3 Data Transfers

Data may be transferred within capsules or by memory transfers. SGLs are used to specify the location of data. Metadata, if transferred, is a contiguous part of the logical block with which that metadata is associated. The SGL descriptor(s) (refer to section 4.4 in the NVMe Base specification) specify whether the command’s data is transferred through memory or within the capsule. The capsule may contain either SGLs or data (not a mixture of both) following the SQE. If additional SGLs are required, then the SGLs are included in the capsule immediately after the SQE. If an invalid offset is specified in an SGL descriptor, then a status value of SGL Offset Invalid shall be returned.

SGLs shall be supported within a capsule. The NVMe Transport binding specification defines the SGL Descriptor Types and Sub Types that are supported for the corresponding NVMe Transport. The NVMe Transport binding specification also specifies if SGLs may be supported in host memory.

2.3.1 Data and SGL Locations within a Command Capsule

The Submission Queue Entry within the command capsule includes one SGL entry. If there are additional SGL entries to be transferred in the command capsule, then those entries shall be contiguous and located immediately after the Submission Queue Entry.

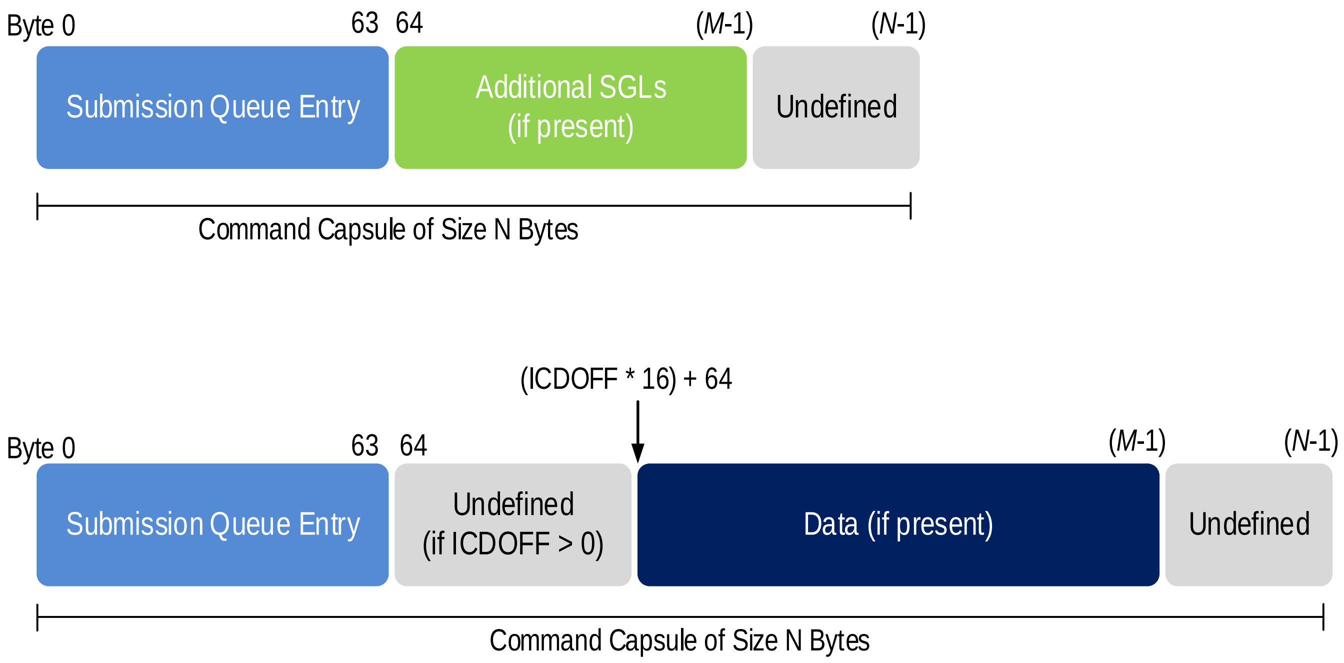
An NVMe Transport binding specification defines the support for data as part of the command capsule. The controller indicates the starting location of data within a command capsule via the In Capsule Data Offset (ICDOFF) field in the Identify Controller data structure.

There are restrictions for SGLs that the host should follow:

- If ICDOFF is a non-zero value, then all the SGL descriptors following the Submission Queue Entry shall not have a total size greater than (ICDOFF * 16); and
- the host shall not place more SGL Data Block or Keyed SGL Data Block descriptors within a capsule than the maximum indicated in the Identify Controller data structure.

The host shall start data (if present) in command capsules at byte offset (ICDOFF * 16) from the end of the Submission Queue Entry.

Figure 11: Data and SGL Locations within a Command Capsule

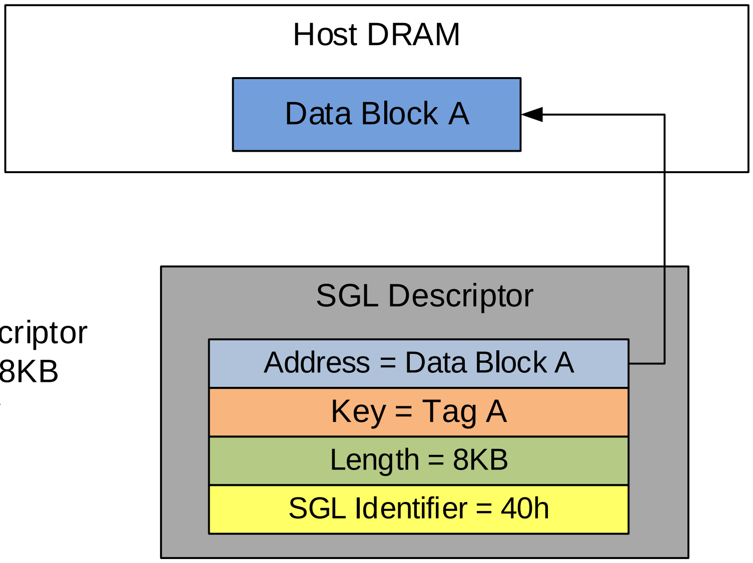


2.3.2 Data Transfer Examples

The data transfer examples in this section show SGL examples for a Write command where data is transferred via a memory transaction or within the capsule. The SGL may use a key as part of the data transfer depending on the requirements of the NVMe Transport used.

The first example shows an 8KB write where all of the data is transferred via memory transactions. In this case, there is one SGL descriptor that is contained within the Submission Queue Entry at CMD.SGL1. The SGL descriptor is a Keyed SGL Data Block descriptor. If more SGLs are required to complete the command, the additional SGLs are contained in the command capsule.

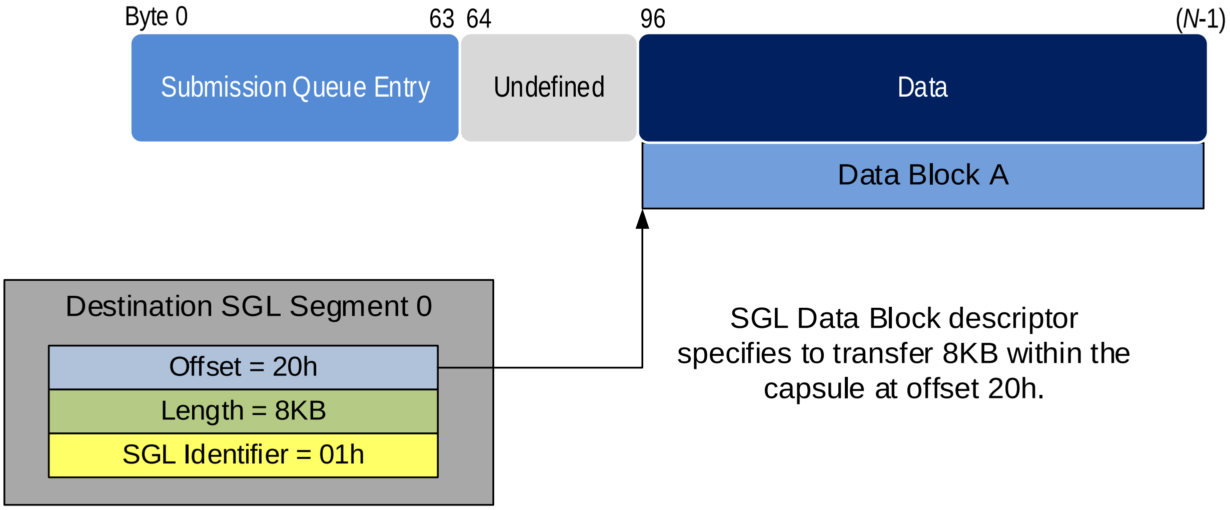
Figure 12: SGL Example Using Memory Transactions



Keyed Data Block descriptor specifies to transfer 8KB through memory

The second example shows an 8KB write where all of the data is transferred within the capsule. In this case, the SGL descriptor is an SGL Data Block descriptor specifying an Offset of 20h based on an ICDOFF value of 2h.

Figure 13: SGL Example Using In Capsule Data Transfer


Byte 0
63
64
96
(N-1)
Submission Queue Entry
Undefined
Data
Data Block A
Destination SGL Segment 0
Offset = 20h
Length = 8KB
SGL Identifier = 01h
SGL Data Block descriptor specifies to transfer 8KB within the capsule at offset 20h.

2.4 Submission Queue and Completion Queue Definition

NVMe over Fabrics Submission Queues and Completion Queues are message-based (refer to Figure 1) in contrast to NVMe over PCIe memory-based queues (refer to section 4.1 in NVMe Base specification), Doorbells are not used by NVMe over Fabrics. In this section and sections 2.4.1, 2.4.2, and 2.4.3, the terms Submission Queue, Completion Queue and queue refer to NVMe over Fabrics queues unless explicitly stated otherwise.

For NVMe over Fabrics, a queue is a unidirectional communication channel that is used to send capsules between a host and a controller. A host uses Submission Queues to send command capsules (refer to section 2.1) to a controller. A controller uses Completion Queues to send response capsules (refer to section 2.2) to a host. Submission and Completion Queues are created in pairs using the Connect command (refer to section 1.5.7).

The NVMe Transport is responsible for delivering command capsules to the controller and notifying the controller of capsule arrival in a transport-specific fashion.

Altering a command capsule between host submission to the Submission Queue and transport delivery of that capsule to the controller results in undefined behavior.

The queue attributes of Queue Size, Queue Identifier, and Queue Priority are defined in sections 4.1.3, 4.1.4, and 4.1.5 of the NVMe Base specification.

NVMe Transports are not required to provide any additional end-to-end flow control. Specific NVMe Transports may require low level flow control for congestion avoidance and reliability; any such additional NVMe Transport flow control is outside the scope of this specification.

Flow control differs for Submission Queues and Completion Queues (refer to sections 2.4.1, 2.4.2, and 2.4.3).

2.4.1 Submission Queue Flow Control Negotiation

Use of Submission Queue (SQ) flow control is negotiated for each queue pair by the Connect command and the controller response to the Connect command. SQ flow control shall be used unless it is disabled as a result of that negotiation. If SQ flow control is disabled, then the SQHD field is reserved in all Fabrics response capsules for that queue pair after the response to the Connect command (i.e., in all subsequent response capsules for that queue pair, the controller shall clear the SQHD field to 0h and the host should ignore the SQHD field).

If the host requests that SQ flow control be disabled for a queue pair, then the host should size each Submission Queue to support the maximum number of commands that the host could have outstanding at one time for that Submission Queue.

The maximum size of the Admin Submission Queue is specified in the Admin Max SQ Size (ASQSZ) field of the Discovery Log entry for the NVM subsystem (refer to section 5.3).

The maximum size of an I/O Submission Queue is specified in the Maximum Queue Entries Supported (MQES) field of the Controller Capabilities (CAP) property (refer to section 1.5.5) for the controller.

The Maximum Outstanding Commands (MAXCMD) value in the Identify Controller data structure indicates the maximum number of commands that the controller processes at one time for a particular I/O Queue. The host may use this value to size I/O Submission Queues and optimize the number of commands submitted at one time per queue to achieve the best performance.

If SQ flow control is disabled, then the host should limit the number of outstanding commands for a queue pair to be less than the size of the Submission Queue. If the controller detects that the number of outstanding commands for a queue pair is greater than or equal to the size of the Submission Queue, then the controller shall:

a) stop processing commands and set the Controller Fatal Status (CSTS.CFS) bit to ‘1’ (refer to section 10.5 in the NVMe Base specification); and
b) terminate the NVMe Transport connection and end the association between the host and the controller.

2.4.2 Submission Queue Flow Control

This section applies only to Submission Queues that use SQ flow control.

The Submission Queue has a Head entry pointer and a Tail entry pointer that are used to manage the queue and determine the number of Submission Queue capsules available to the host for new submissions. The Head and Tail entry pointers are initialized to 0h when a queue is created. All arithmetic operations and comparisons on entry pointers are performed modulo the queue size with queue wrap conditions taken into account. The host increments the Tail entry pointer when the host adds a capsule to a queue. The controller increments the Head entry pointer when that controller removes a capsule from the queue.

The NVMe over Fabrics Submission Queue Head entry pointer is maintained by the controller and is communicated to the host in the SQHD field of Completion Queue Entries. The host uses the received SQHD values for Submission Queue management (e.g., to determine whether the Submission Queue is full).

The NVMe over Fabrics Submission Queue Tail entry pointer is local to the host and is not communicated to the controller.

The Submission Queue is full when the Head entry pointer equals one more than the Tail entry pointer (i.e., incrementing the Tail entry pointer has caused it to wrap around to just behind the Head entry pointer). A full Submission Queue contains one less capsule than the queue size. A host may continue to submit commands to a Submission Queue as long as the queue is not full.

If the controller detects that the host has submitted a command capsule to a full Submission Queue, then the controller shall:

a) stop processing commands and set the Controller Fatal Status (CSTS.CFS) bit to ‘1’ (refer to section 10.5 in the NVMe Base specification); and
b) terminate the NVMe Transport connection and end the association between the host and the controller.

The Submission Queue is empty when the Head entry pointer equals the Tail entry pointer.

2.4.3 Completion Queue Flow Control Considerations

Completion Queue flow control is not used in NVMe over Fabrics. NVMe over Fabrics Completion Queues do not use either Head entry pointers or Tail entry pointers.

The host should size each Completion Queue to support the maximum number of commands that the host could have outstanding at one time for a particular Submission Queue. The Completion Queue size may be larger than the size of the corresponding Submission Queue to accommodate responses for commands that are being processed by the controller in addition to responses for commands that are still in the Submission Queue.

If the size of a Completion Queue is too small for the number of outstanding commands and the controller submits a response capsule to a full Completion Queue, then the results are undefined.

The Maximum Outstanding Commands (MAXCMD) value in the Identify Controller data structure indicates the maximum number of commands that the controller processes at one time for a particular I/O Queue. The host may use this value to size I/O Completion Queues and optimize the number of commands submitted at one time per queue to achieve the best performance.

Altering a response capsule between controller submission to the Completion Queue and transport delivery of that capsule to the host results in undefined behavior.

3 Commands

Fabrics commands are used to create queues and initialize a controller. Fabrics commands have an Opcode field of 7Fh. Fabrics commands are processed regardless of the state of controller enable (CC.EN). The Fabrics command capsule is defined in section 2.1 and the Fabrics response capsule and status is defined in section 2.2.

Figure 14: Fabrics Command Types

Command Type by Field			Combined Command Type[2]	O/M[1]	I/O Queue[3]	Command
(07)	**(06:02)**	**(01:00)**				
Generic Command	**Function**	**Data Transfer[4]**				
0b	000 00b	00b	00h	M	No	Property Set
0b	000 00b	01b	01h	M	Yes	Connect[5]
0b	000 01b	00b	04h	M	No	Property Get
0b	000 01b	01b	05h	O	Yes	Authentication Send
0b	000 01b	10b	06h	O	Yes	Authentication Receive
0b	000 10b	00b	08h	O	Yes	Disconnect
	Vendor Specific					
1b	na	na	C0h to FFh	O		Vendor specific

NOTES:

1. O/M definition: O = Optional, M = Mandatory.
2. Opcodes not listed are reserved.
3. All Fabrics commands, other than the Disconnect command, may be submitted on the Admin Queue. The I/O Queue supports Fabrics commands as specified in this column. If a Fabrics command that is not supported on an I/O Queue is sent on an I/O Queue, that command shall be aborted with a status code of Invalid Field in Command.
4. 00b = no data transfer; 01b = host to controller; 10b = controller to host; 11b = reserved
5. The Connect command is submitted and completed on the same queue that the Connect command creates. Refer to section 1.5.7.

3.1 Authentication Receive Command and Response

The Authentication Receive command transfers the status and data result of one or more Authentication Send commands that were previously submitted to the controller.

The association between an Authentication Receive command and previous Authentication Send commands is dependent on the Security Protocol. The format of the data to be transferred is dependent on the Security Protocol. Refer to SPC-4 for Security Protocol details.

Authentication Receive commands return the appropriate data corresponding to an Authentication Send command as defined by the rules of the Security Protocol. The Authentication Receive command data shall not be retained if there is a loss of communication between the controller and host, or if a Controller Level Reset occurs.

Figure 15 Authentication Receive Command – Submission Queue Entry

Bytes	Description
00	**Opcode (OPC):** Set to 7Fh to indicate a Fabrics command.
01	Reserved
03:02	**Command Identifier (CID):** This field specifies a unique identifier for the command. Refer to the definition in Figure 7.
04	**Fabrics Command Type (FCTYPE):** Set to 06h to indicate an Authentication Receive command.
23:05	Reserved

Figure 15 Authentication Receive Command – Submission Queue Entry

Bytes	Description
39:24	**SGL Descriptor 1 (SGL1):** This field contains a Transport SGL Data Block descriptor or a Keyed SGL Data Block descriptor that describes the entire data transfer. Refer to section 4.4 of the NVMe Base specification for the definition of SGL descriptors.
40	Reserved
41	**SP Specific 0 (SPSP0):** The value of this field contains bits 07:00 of the Security Protocol Specific field as defined in SPC-4.
42	**SP Specific 1 (SPSP1):** The value of this field contains bits 15:08 of the Security Protocol Specific field as defined in SPC-4.
43	**Security Protocol (SECP):** This field specifies the security protocol as defined in SPC-4. The controller shall fail the command with Invalid Parameter indicated if a reserved value of the Security Protocol is specified.
47:44	**Allocation Length (AL):** The value of this field is specific to the Security Protocol as defined in SPC-4 where INC_512 is cleared to ‘0’.
63:48	Reserved

Figure 16: Authentication Receive Response

Bytes	Description
07:00	Reserved
09:08	**SQ Head Pointer (SQHD):** Indicates the current Submission Queue Head pointer for the associated Submission Queue.
11:10	Reserved
13:12	**Command Identifier (CID):** Indicates the identifier of the command that is being completed.
15:14	**Status (STS):** Specifies status for the command.

3.2 Authentication Send Command and Response

The Authentication Send command is used to transfer security protocol data to the controller. The data structure transferred as part of this command contains security protocol specific commands to be performed by the controller. The data structure may contain data or parameters associated with the security protocol specific commands. Status and data that is to be returned to the host for the security protocol specific commands submitted by an Authentication Send command are retrieved with the Authentication Receive command defined in section 3.1.

The association between an Authentication Send command and subsequent Authentication Receive commands is Security Protocol field dependent as defined in SPC-4.

Figure 17: Authentication Send Command – Submission Queue Entry

Bytes	Description
00	**Opcode (OPC):** Set to 7Fh to indicate a Fabrics command.
01	Reserved
03:02	**Command Identifier (CID):** This field specifies a unique identifier for the command. Refer to the definition in Figure 7.
04	**Fabrics Command Type (FCTYPE):** Set to 05h to indicate an Authentication Send command.
23:05	Reserved
39:24	**SGL Descriptor 1 (SGL1):** This field contains a Transport SGL Data Block descriptor or a Keyed SGL Data Block descriptor that describes the entire data transfer. Refer to section 4.4 of the NVMe Base specification for the definition of SGL descriptors.
40	Reserved
41	**SP Specific 0 (SPSP0):** The value of this field contains bits 07:00 of the Security Protocol Specific field as defined in SPC-4.

Figure 17: Authentication Send Command – Submission Queue Entry

Bytes	Description
42	**SP Specific 1 (SPSP1):** The value of this field contains bits 15:08 of the Security Protocol Specific field as defined in SPC-4.
43	**Security Protocol (SECP):** This field specifies the security protocol as defined in SPC-4. The controller shall fail the command with Invalid Parameter indicated if a reserved value of the Security Protocol is specified.
47:44	**Transfer Length (TL):** The value of this field is specific to the Security Protocol as defined in SPC-4 where INC_512 is cleared to '0'.
63:48	Reserved

Figure 18: Authentication Send Response

Bytes	Description
07:00	Reserved
09:08	**SQ Head Pointer (SQHD):** Indicates the current Submission Queue Head pointer for the associated Submission Queue.
11:10	Reserved
13:12	**Command Identifier (CID):** Indicates the identifier of the command that is being completed.
15:14	**Status (STS):** Specifies status for the command.

3.3 Connect Command and Response

The Connect command is used to create a Submission and Completion Queue pair for an Admin Queue or an I/O Queue. If the Admin Queue is specified, then the Connect command establishes an association between a host and a controller. The fields for the Submission Queue Entry are defined in Figure 19 and the fields for the data portion are defined in Figure 20.

A host that uses a single Host NQN may employ multiple Host Identifiers to designate elements of the host that access an NVM subsystem independently of each other (e.g., physical or logical partitions of the host). Alternatively, a host may employ multiple Host NQN values to cause each element to be treated as a separate host by an NVM subsystem.

The NVM subsystem shall not allocate a Controller ID in the range FFF0h to FFFFh as a valid Controller ID on completion of a Connect command. If the host is not allowed to establish an association to any controller in the NVM subsystem, then a status of Connect Invalid Host is returned.

If the NVM subsystem supports the dynamic controller model, then:

- the Controller ID of FFFFh shall be specified as the Controller ID in a Connect command for the Admin Queue. If the controller ID is not set to FFFFh, then a status value of Connect Invalid Parameters is returned;
- the NVM subsystem shall allocate any available controller to the host; and
- return that allocated Controller ID in the Connect response.

If the NVM subsystem supports the static controller model, then:

- The host may request a specific controller in a Connect command for the Admin Queue. If the host is not allowed to establish an association to the specified controller, then a status of Connect Invalid Host is returned;
- The Controller ID of FFFEh on the Admin Queue specifies that any Controller ID may be allocated and returned in the Connect response; and
- If the host specifies a Controller ID value of FFFFh for the Admin Queue, then a status value of Connect Invalid Parameters is returned.

The NVM subsystem may allocate specific controllers to particular hosts. If a host requests a controller that is not allocated to that host, then a status value of Connect Invalid Host is returned. The mechanism for allocating specific controllers to particular hosts is outside the scope of this specification.

The host shall establish an association with a controller and enable the controller before establishing a connection with an I/O Queue of the controller. If the host sends a Connect command specifying a Queue ID for an Admin Queue or I/O Queue which has already been created, then a status value of Command Sequence Error is returned.

A status of Connect Invalid Parameters is returned for a Connect command if:

- the host sends a Connect command to create an I/O Queue while the controller is disabled;
- the Host Identifier, Host NQN, NVM Subsystem NQN, and the Controller ID values specified for an I/O Queue are not the same as the values specified for the associated Admin Queue in which the association between the host and controller was established;
- the Host NQN or NVM Subsystem NQN values do not match the values that the NVM subsystem is configured to support;
- there is a syntax error in the Host NQN or NVM Subsystem NQN value (refer to section 7.9 in the NVMe Base specification); or
- the Host Identifier is cleared to 0h.

If the NVMe Subsystem Port, NVMe Transport Type or NVMe Transport Address used by the NVMe Transport (refer to section 1.5.1) are not the same as the values used for the associated Admin Queue in which the association between the host and controller was established, then it is possible that the Connect command is not received by an NVM subsystem. If the Connect command is received by an NVM subsystem, then:

- the NVM subsystem that receives the command may not be the same NVM subsystem to which the association between the host and controller was established (i.e., the NVMe Transport Type and NVMe Transport Address are unique to an NVM Subsystem Port); and
- the values of the NVM Subsystem NQN or Controller ID may not be valid at that NVM Subsystem Port (e.g., the NVM Subsystem NQN may specify a different NVM subsystem than the one that received that Connect command, or the Controller ID may specify a controller that is already bound to a different NVM Subsystem Port).

If this situation occurs and a status value is returned for the Connect command, then that status value is Connect Invalid Parameters. There is no requirement that such a Connect command be received by an NVM subsystem (e.g., if the NVMe Transport Address is not a valid transport address, or is the address of a fabric endpoint that does not support NVMe over Fabrics, then the resulting error, if any, is specific to the fabric).

Submission Queue (SQ) flow control based on the SQ Head Pointer (SQHD) field in Fabrics response capsules (refer to section 2.2) shall be supported by all hosts and controllers. Use of SQ flow control is negotiated by the Connect command and response. A host requests that SQ flow control be disabled by setting bit 2 of the Connect Attributes field to ‘1’ in a Connect command. A controller that agrees to disable SQ flow control shall set the SQHD field to FFFFh in the response to that Connect command. A controller that does not agree to disable SQ flow control shall set the SQHD field to a value other than FFFFh in the response to that Connect command.

If the Connect command did not request that SQ flow control be disabled, then the controller shall not set the SQHD field to FFFFh in the response to that Connect command.

SQ flow control is disabled and shall not be used for a created queue pair only if:

a) bit 2 is set to ‘1’ in the Connect Attributes field of the Connect command that creates the queue pair; and
b) the SQHD field is set to FFFFh in the response to that Connect command.

If SQ flow control is disabled, then the SQHD field is reserved in Fabrics response capsules for all command completions on that queue pair after the response that completes the Connect command.

SQ flow control is enabled and shall be used for a created queue pair if:

a) bit 2 is cleared to ‘0’ in the Connect Attributes field of the Connect command that creates the queue pair; or
b) the SQHD field is not set to FFFFh in the response to that Connect command.

If SQ flow control is enabled, then the controller shall use the SQHD field in Fabrics response capsules for all command completions on that queue pair, except for command completions that omit the SQHD value due to use of the SQHD pointer update optimization described in section 7.1.1.

Figure 19: Connect Command – Submission Queue Entry

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>00</td><td><b>Opcode (OPC):</b> Set to 7Fh to indicate a Fabrics command.</td></tr>
<tr><td>01</td><td>Reserved</td></tr>
<tr><td>03:02</td><td><b>Command Identifier (CID):</b> This field specifies a unique identifier for the command. Refer to the definition in Figure 7.</td></tr>
<tr><td>04</td><td><b>Fabrics Command Type (FCTYPE):</b> Set to 01h to indicate a Connect command.</td></tr>
<tr><td>23:05</td><td>Reserved</td></tr>
<tr><td>39:24</td><td><b>SGL Descriptor 1 (SGL1):</b> This field contains a Transport SGL Data Block descriptor or a Keyed SGL Data Block descriptor that describes the entire data transfer. Refer to section 4.4 of the NVMe Base specification for the definition of SGL descriptors.</td></tr>
<tr><td>41:40</td><td><b>Record Format (RECFMT):</b> Specifies the format of the Connect command capsule. The format of the record specified in this definition shall be 0h. If the NVM subsystem does not support the value specified, then a status value of Incompatible Format shall be returned.</td></tr>
<tr><td>43:42</td><td><b>Queue ID (QID):</b> Specifies the Queue Identifier for the Admin Queue or I/O Queue to be created. The identifier is used for both the Submission and Completion Queue. The identifier for the Admin Submission Queue and Completion Queue is 0h. The identifier for an I/O Submission and Completion Queue is in the range 1 to 65,534.</td></tr>
<tr><td>45:44</td><td><b>Submission Queue Size (SQSIZE):</b> This field indicates the size of the Submission Queue to be created. If the size is 0h or larger than the controller supports, then a status value of Connect Invalid Parameters shall be returned. The maximum size of the Admin Submission Queue is specified in the Discovery Log entry for the NVM subsystem. Refer to section 4.1.3 of the NVMe Base specification. This is a 0’s based value.</td></tr>
<tr><td>46</td><td><b>Connect Attributes (CATTR):</b> This field indicates attributes for the connection.
Bits 7:4 are reserved.
Bit 3 indicates support for deleting individual I/O Queues. If this bit is set to ‘1’, then the host supports the deletion of individual I/O Queues. If this bit is cleared to ‘0’, then the host does not support the deletion of individual I/O Queues.
Bit 2 if set to ‘1’, then the host is requesting that SQ flow control be disabled. If cleared to ‘0’, then SQ flow control shall not be disabled.
Bits 1:0 indicate the priority class to use for commands within this Submission Queue. This field is only used when the weighted round robin with urgent priority class is the arbitration mechanism selected, the field is ignored if weighted round robin with urgent priority class is not used. Refer to section 4.13 of the NVMe Base specification. This field is only valid for I/O Queues. It shall be set to 00b for Admin Queue connections.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00b</td><td>Urgent</td></tr>
<tr><td>01b</td><td>High</td></tr>
<tr><td>10b</td><td>Medium</td></tr>
<tr><td>11b</td><td>Low</td></tr>
</table></td></tr>
<tr><td>47</td><td>Reserved</td></tr>
</table>

Figure 19: Connect Command – Submission Queue Entry

Bytes	Description
51:48	**Keep Alive Timeout (KATO):** In the Connect command for the Admin Queue, this field has the same definition as the Keep Alive Timeout (Keep Alive Timer) defined in section 5.21.1.15 of the NVMe Base specification. Upon successful completion of the Connect command the controller shall enable and activate the Keep Alive timer as described in section 7.12 (Keep Alive) of the NVMe Base specification. In the Connect command for an I/O Queue, this field is reserved.
63:52	Reserved

Figure 20: Connect Command – Data

Bytes	Description
15:00	**Host Identifier (HOSTID):** This field has the same definition as the Host Identifier defined in section 5.21.1.26 (Host Identifier) of the the NVMe Base specification. The controller shall set the Host Identifer Feature to this value.
17:16	**Controller ID (CNTLID):** Specifies the controller ID requested. This field corresponds to the Controller ID (CNTLID) value returned in the Identify Controller data structure for a particular controller. If the NVM subsystem uses the dynamic controller model, then the value shall be FFFFh for the Admin Queue and any available controller may be returned. If the NVM subsystem uses the static controller model and the value is FFFEh for the Admin Queue, then any available controller may be returned.
255:18	Reserved
511:256	**NVM Subsystem NVMe Qualified Name (SUBNQN):** NVMe Qualified Name (NQN) that uniquely identifies the NVM subsystem. Refer to section 7.9 (NVMe Qualified Names) of the NVMe Base specification.
767:512	**Host NVMe Qualified Name (HOSTNQN):** NVMe Qualified Name (NQN) that uniquely identifies the host. Refer to section 7.9 (NVMe Qualified Names) of the NVMe Base specification.
1023:768	Reserved

The Connect response provides status for the Connect command. If a connection is established, then the Controller ID allocated to the host is returned. The Connect response is defined in Figure 21.

For a Connect command that fails the controller shall not:

- return a status value of Invalid Field in Command; and
- add an entry to the Error Information Log.

Figure 21: Connect Response

Bytes	Description
03:00	**Status Code Specific:** The value is dependent on the status returned. Refer to Figure 22.
07:04	Reserved
09:08	**SQ Head Pointer (SQHD):** If the Connect command requested that SQ flow control be disabled, then a value of FFFFh in this field indicates that SQ flow control is disabled for the created queue pair. Otherwise, this field indicates the current Submission Queue Head pointer for the associated Submission Queue and also indicates that SQ flow control is enabled for the created queue pair.
11:10	Reserved
13:12	**Command Identifier (CID):** Indicates the identifier of the command that is being completed.
15:14	**Status (STS):** Specifies status for the command. Refer to section 2.2.1 for values specific to the Connect command.

Figure 22: Connect Response – Dword 0 Value Based on Status Code

<table>
<tr><th>Status Code</th><th>Definition of Dword 0</th></tr>
<tr><td>Successful Completion</td><td>
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>01:00</td><td><b>Controller ID (CNTLID):</b> Specifies the controller ID allocated to the host. If a particular controller was specified in the CNTLID field of the Connect command, then this field shall contain the same value.</td></tr>
<tr><td>03:02</td><td><b>Authentication Requirements (AUTHREQ):</b> Specifies the NVMe in-band authentication requirements. The field is bit significant. If no bit is set to ‘1’, then NVMe in-band authentication is not required. If a bit is set to ‘1’, then NVMe in-band authentication using the specified protocol is acceptable. If a bit is cleared to ‘0’, then NVMe in-band authentication shall not use the specified protocol.
<table>
<tr><th>Bits</th><th>Definition</th></tr>
<tr><td>15:01</td><td>Reserved</td></tr>
<tr><td>00</td><td>TCG Security Protocols (refer to TCG Storage Interface Interactions Specification [SIIS])</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>Connect Invalid Parameters</td><td>
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>01:00</td><td><b>Invalid Parameter Offset (IPO):</b> If an invalid parameter is reported, then this field specifies the offset in bytes to the invalid parameter from the start of the SQE or the data.</td></tr>
<tr><td>02</td><td><b>Invalid Attributes (IATTR):</b> Specifies attributes of the invalid field parameter.
Bits 7:1 are reserved.
Bit 0 if cleared to ‘0’, then the invalid parameter is specified from the start of the SQE. Bit 0 if set to ‘1’, then the invalid parameter is specified from the start of the data.</td></tr>
<tr><td>03</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>All Other Status Values</td><td>
<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>03:00</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>

3.4 Disconnect Command and Response

The Disconnect command is used to delete the I/O Queue on which the command is submitted. If a Disconnect command is submitted on an Admin Queue, then the controller shall abort the command with a status of Invalid Queue Type. If the controller is not able to delete the I/O Queue, then the controller shall abort the command with a status of Controller Busy. The fields for the Submission Queue Entry are defined in Figure 23.

The NVMe Transport connection is not deleted upon issuance of a Disconnect command; the host and controller may delete the NVMe Transport connection and associated resources after all NVMe Queues (I/O Queues and/or Admin Queue) associated with that NVMe Transport connection have been deleted (refer to section 1.5.9 and section 4.5).

The Completion Queue entry for the Disconnect command shall be the last entry submitted to the I/O Queue Completion queue by the controller (i.e., no completion queue entries shall be submitted to the I/O Queue Completion Queue after the Completion Queue entry for the Disconnect command). The controller shall not perform command processing for any command on an I/O queue after sending the Completion Queue entry for the Disconnect command.

The host should not submit commands to an I/O Submission Queue after the submission of a Disconnect command to that I/O Submission Queue; submitting commands to an I/O Queue after a Disconnect command is submitted to that I/O Queue results in undefined behavior.

Figure 23: Disconnect Command and Response

Bytes	Description
00	**Opcode (OPC):** Set to 7Fh to indicate a Fabrics command.
01	Reserved
03:02	**Command Identifier (CID):** This field specifies a unique identifier for the command. Refer to the definition in Figure 7.
04	**Fabrics Command Type (FCTYPE):** Set to 08h to indicate a Disconnect command.
23:05	Reserved
39:24	**SGL Descriptor 1 (SGL1):** This field is reserved, as there is no data transferred by this command.
41:40	**Record Format (RECFMT):** Specifies the format of the Disconnect command capsule. The format of the record specified in this definition shall be 0h. If the NVM subsystem does not support the value specified, then a status value of Incompatible Format shall be returned.
63:48	Reserved

The Disconnect response provides status for the Disconnect command. The Disconnect response is defined in Figure 24.

Figure 24: Disconnect Response

Bytes	Description
07:00	Reserved
09:08	**SQ Head Pointer (SQHD):** Indicates the current Submission Queue Head pointer for the associated Submission Queue.
11:10	Reserved
13:12	**Command Identifier (CID):** Indicates the identifier of the command that is being completed.
15:14	**Status (STS):** Specifies status for the command. Refer to section 2.2.1 for values specific to the Disconnect command.

3.5 Property Get Command and Response

The Property Get command is used to specify the property value to return to the host (refer to section 3.6.1). The fields for the Property Get command are defined in Figure 25. If an invalid property or invalid offset is specified, then a status value of Invalid Field in Command shall be returned.

Figure 25: Property Get Command

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>00</td><td><b>Opcode (OPC):</b> Set to 7Fh to indicate a Fabrics command.</td></tr>
<tr><td>01</td><td>Reserved</td></tr>
<tr><td>03:02</td><td><b>Command Identifier (CID):</b> This field specifies a unique identifier for the command. Refer to the definition in Figure 7.</td></tr>
<tr><td>04</td><td><b>Fabrics Command Type (FCTYPE):</b> Set to 04h to indicate a Property Get command.</td></tr>
<tr><td>39:05</td><td>Reserved</td></tr>
<tr><td>40</td><td><b>Attributes (ATTRIB):</b> Specifies attributes for the Property Get command.

Bits 7:3 are reserved.

Bits 2:0 specifies the size of the property to return. Valid values are shown in the table below.

<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>000b</td><td>4 bytes</td></tr>
<tr><td>001b</td><td>8 bytes</td></tr>
<tr><td>010b to 111b</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>

Figure 25: Property Get Command

Bytes	Description
43:41	Reserved
47:44	**Offset (OFST):** Specifies the offset to the property to get. Refer to section 3.6.1.
63:48	Reserved

The Property Get response is used to return the value of the property requested to the host. The Property Get response is defined in Figure 26.

Figure 26: Property Get Response

Bytes	Description
07:00	**Value (VALUE):** Specifies the value returned for the property if the Property Get command is successful. If the size of the property is four bytes, then the value is specified in bytes 03:00 and bytes 07:04 are reserved.
09:08	**SQ Head Pointer (SQHD):** Indicates the current Submission Queue Head pointer for the associated Submission Queue.
11:10	Reserved
13:12	**Command Identifier (CID):** Indicates the identifier of the command that is being completed.
15:14	**Status (STS):** Specifies status for the command.

3.6 Property Set Command and Response

The Property Set command is used to set the value of a property (refer to section 3.6.1). The fields for the Property Set command are defined in Figure 27. If an invalid property or invalid offset is specified, then a status value of Invalid Field in Command shall be returned.

Figure 27: Property Set Command

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>00</td><td><b>Opcode (OPC):</b> Set to 7Fh to indicate a Fabrics command.</td></tr>
<tr><td>01</td><td>Reserved</td></tr>
<tr><td>03:02</td><td><b>Command Identifier (CID):</b> This field specifies a unique identifier for the command. Refer to the definition in Figure 7.</td></tr>
<tr><td>04</td><td><b>Fabrics Command Type (FCTYPE):</b> Set to 00h to indicate a Property Set command.</td></tr>
<tr><td>39:05</td><td>Reserved</td></tr>
<tr><td>40</td><td><b>Attributes (ATTRIB):</b> Specifies attributes for the Property Set command.
Bits 7:3 are reserved.
Bits 2:0 specifies the size of the property to update. Valid values are shown in the table below.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>000b</td><td>4 bytes</td></tr>
<tr><td>001b</td><td>8 bytes</td></tr>
<tr><td>010b to 111b</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>43:41</td><td>Reserved</td></tr>
<tr><td>47:44</td><td><b>Offset (OFST):</b> Specifies the offset to the property to set. Refer to section 3.6.1.</td></tr>
<tr><td>55:48</td><td><b>Value (VALUE):</b> Specifies the value used to update the property. If the size of the property is four bytes, then the value is specified in bytes 51:48 and bytes 55:52 are reserved.</td></tr>
<tr><td>63:56</td><td>Reserved</td></tr>
</table>

The Property Set response provides status for the Property Set command. The Property Set response is defined in Figure 28.

Figure 28: Property Set Response

Bytes	Description
07:00	Reserved
09:08	**SQ Head Pointer (SQHD):** Indicates the current Submission Queue Head pointer for the associated Submission Queue.
11:10	Reserved
13:12	**Command Identifier (CID):** Indicates the identifier of the command that is being completed.
15:14	**Status (STS):** Specifies status for the command.

3.6.1 Property Definitions

A property is a dword, or qword attribute of a controller. The attribute may have read, write, or read/write access. The host shall access properties in their native width with an offset that is at the beginning of the property. All reserved properties and all reserved bits within properties are read-only and return 0h when read. Properties may be read with the Property Get command and may be written with the Property Set command.

The property address range from 0h to FFFh is reserved for functionality that is equivalent to the register functionality defined for NVMe over PCIe. The property address range from 1000h to 12FFh is reserved for definition by NVMe over Fabrics. The doorbell registers defined for NVMe over PCIe are not supported in NVMe over Fabrics.

Figure 29 specifies the properties that are supported by NVMe over Fabrics.

Figure 29: Property Definitions

Start	End	Symbol	Description
00h	07h	CAP	Controller Capabilities Note: CAP.DSTRD shall be set to fixed value 0h. Note: CAP.CQR shall be set to fixed value 1h.
08h	0Bh	VS	Version
0Ch	0Fh	INTMS	Reserved
10h	13h	INTMC	Reserved
14h	17h	CC	Controller Configuration
18h	1Bh	Reserved	Reserved
1Ch	1Fh	CSTS	Controller Status
20h	23h	NSSR	NVM Subsystem Reset (Optional)
24h	27h	AQA	Reserved
28h	2Fh	ASQ	Reserved
30h	37h	ACQ	Reserved
38h	3Bh	CMBLOC	Reserved
3Ch	3Fh	CMBSZ	Reserved
40h	EFFh	Reserved	Reserved
F00h	FFFh	Reserved	Command Set Specific
1000h	12FFh	Reserved	Reserved for Fabrics definition
			Vendor Specific (Optional)

4 Controller Architecture

NVMe over Fabrics utilizes the same controller architecture as that defined in the NVMe Base specification. This includes using Submission and Completion Queues to execute commands between a host and a controller.

4.1 Identify Controller Data Structure Enhancements

This section defines Identify Controller fields that are specific to NVMe over Fabrics.

Figure 30: Identify Controller Attributes

Bytes	O/M[1]	Description
1795:1792	M	**I/O Queue Command Capsule Supported Size (IOCCSZ):** This field defines the maximum I/O command capsule size in 16 byte units. The minimum value that shall be indicated is 4 corresponding to 64 bytes.
1799:1796	M	**I/O Queue Response Capsule Supported Size (IORCSZ):** This field defines the maximum I/O response capsule size in 16 byte units. The minimum value that shall be indicated is 1 corresponding to 16 bytes.
1801:1800	M	**In Capsule Data Offset (ICDOFF):** This field defines the offset where data starts within a capsule. This value is applicable to I/O Queues only (the Admin Queue shall use a value of 0h). The value is specified in 16 byte units. The offset is from the end of the Submission Queue Entry within the command capsule (starting at 64 bytes in the command capsule). The minimum value is 0 and the maximum value is FFFFh.
1802	M	**Fabrics Controller Attributes (FCATT):** This field indicates attributes of the controller that are specific to NVMe over Fabrics. Bits 7:1 are reserved. Bit 0 if cleared to ‘0’, then the NVM subsystem uses a dynamic controller model. Bit 0 if set to ‘1’, then the NVM subsystem uses a static controller model.
1803	M	**Maximum SGL Data Block Descriptors (MSDBD):** This field indicates the maximum number of SGL Data Block or Keyed SGL Data Block descriptors that a host is allowed to place in a capsule. A value of 0h indicates no limit.
1805:1804	M	**Optional Fabric Commands Support (OFCS):** Indicate whether the controller supports optional fabric commands. Bits 15:1 are reserved. Bit 0 if cleared to ‘0’ then the controller does not support the Disconnect command. Bit 0 if set to ‘1’ then the controller supports the Disconnect command and deletion of individual I/O Queues.
2047:1806		Reserved
NOTES: 1. O/M definition: O = Optional, M = Mandatory.		

4.2 Controller Model

The NVM subsystem may support a dynamic or static controller model. All controllers in the NVM subsystem shall follow the same controller model. A Discovery Controller shall support the dynamic controller model.

In a dynamic controller model, the controller is allocated by the NVM subsystem on demand. In this model, all controllers allocated to a specific host have the same state at the time the association is established, including attached namespaces and Feature settings. Changes to a controller (e.g., attached namespaces, feature settings) after the association is established do not impact other dynamic controllers. ANA state (refer to section 8.20 in the NVMe Base specification) describes a relationship between a controller and a namespace and is not persistent controller state. The host shall specifies a Controller ID of FFFFh when using the Fabrics Connect command (refer to section 3.3) to establish an association with an NVM subsystem using the dynamic controller model.

In a static controller model, controllers that may be allocated to a particular host may have different state at the time the association is established. The controllers within an NVM subsystem are distinguished by their Controller ID. The state that persists across associations is any state that persists across a Controller Level Reset. In a static controller model, different controllers may present different Feature settings or namespace attachments to the same host. The NVM subsystem may allocate particular controllers to specific hosts.

While allocation of static controllers to hosts are expected to be durable (so that hosts can expect to form associations to the same controllers repeatedly (e.g., after each host reboot)), the NVM subsystem may remove the host allocation of a controller that is not in use at any time for implementation specific reasons (e.g., controller resource reclamation, subsystem reconfiguration).

The controller ID values returned in the Discover Log entries indicate whether an NVM subsystem supports the dynamic or static controller model. The controller ID value of FFFFh is a special value used for NVM subsystems that support the dynamic controller model indicating that any available controller may be returned. The controller ID value of FFFEh is a special value used for NVM subsystems that support the static controller model indicating that any available controller may be returned. An NVM subsystem supports the dynamic controller model if Discovery Log entries use the Controller ID value of FFFFh. An NVM subsystem supports the static controller model if Discovery Log entries use a Controller ID value that is less than FFFFh. The Identify Controller data structure also indicates whether an NVM subsystem is dynamic or static.

If an NVM subsystem is dynamic, then multiple Discovery Log Page entries (refer to Figure 38) with the Controller ID set to FFFFh may be returned for that NVM subsystem (e.g., to indicate multiple NVM subsystem ports) in the Discovery Log. If an NVM subsystem is static, then multiple Discovery Log Page entries that indicate different Controller ID values may be returned for that NVM subsystem in the Discovery Log. If an NVM subsystem that is static includes any Discovery Log entries that indicate a Controller ID of FFFEh, then the host should remember the Controller ID returned from the Fabrics Connect command and re-use the allocated Controller ID for future associations to that particular controller.

4.3 Queue Initialization and Queue State

When a Connect command successfully completes, the corresponding Admin Submission and Completion Queue or I/O Submission and Completion Queues are created. If the host sends a Connect command specifying the Queue ID of a queue which already exists, then a status value of Command Sequence Error is returned.

The Authentication Requirements (AUTHREQ) field in the Connect response indicates if NVMe in-band authentication is required. If AUTHREQ is cleared to zero, the created queue is ready for use after the Connect command completes successfully. If AUTHREQ is set to a non-zero value, the created queue is ready for use after NVMe in-band authentication has been performed successfully using the Authentication Send and Authentication Receive Fabrics commands.

If a controller requires or is undergoing NVMe in-band authentication for a queue pair, then a controller shall abort all commands received on that queue other than authentication commands with a status of Authentication Required. After the NVMe in-band authentication has been performed successfully on a queue, then a controller shall abort all authentication commands on that queue with a status of Command Sequence Error.

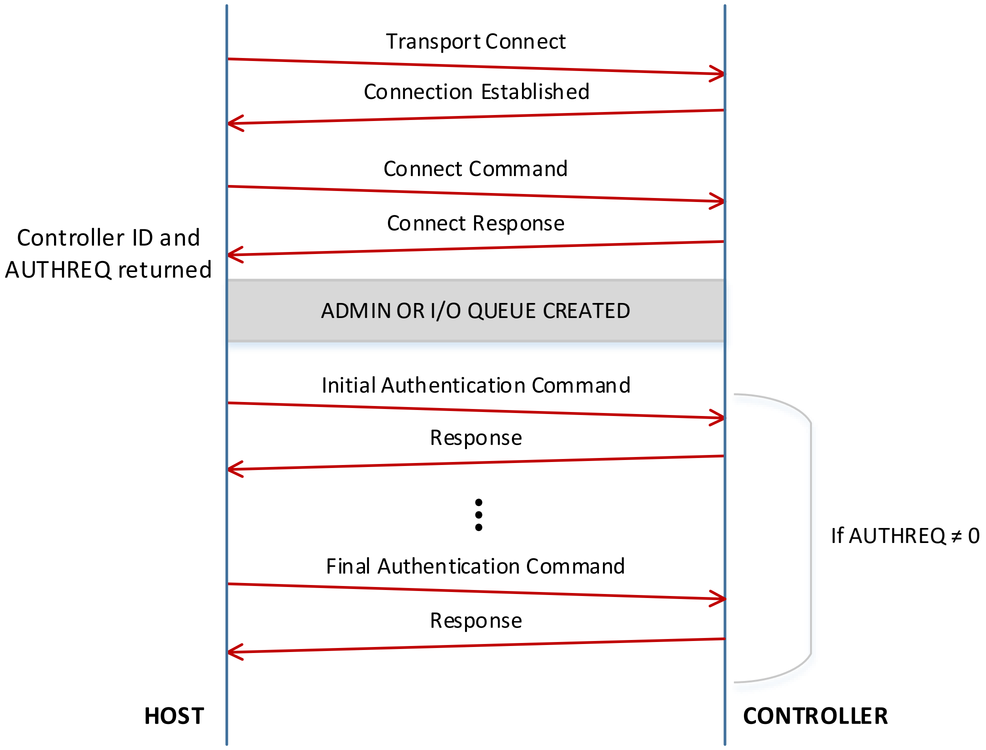
When an Admin Queue is first created, the associated controller is disabled (i.e., CC.EN is initialized to '0'). A disabled controller shall abort all commands other than Fabrics commands on the Admin Queue with a status of Command Sequence Error. After the controller is enabled, it shall accept all supported Admin commands in addition to Fabrics commands.

A created I/O queue shall abort all commands with a status of Command Sequence Error if the associated controller is disabled.

4.4 Initialization

The host selects the NVM subsystem with which to create a host to controller association. The host first establishes an NVMe Transport connection with the NVM subsystem. Next the host forms an association with a controller and creates the Admin Queue using the Fabrics Connect command. Finally, the host configures the controller and creates I/O Queues. Figure 31 is a ladder diagram that describes the queue creation process for an Admin Queue or an I/O Queue.

Figure 31: Queue Creation Flow


Transport Connect
Connection Established
Connect Command
Connect Response
Controller ID and AUTHREQ returned
ADMIN OR I/O QUEUE CREATED
Initial Authentication Command
Response
Final Authentication Command
Response
If AUTHREQ ≠ 0
HOST
CONTROLLER

The controller initialization steps after an association is established are described below. For determining capabilities or configuring properties, the host uses the Property Get and Property Set commands, respectively.

1. NVMe in-band authentication is performed if required (refer to section 6.2);
2. The host determines the controller capabilities;
3. The host configures controller settings. Specific settings include:
   a. The arbitration mechanism should be selected in CC.AMS;
   b. The memory page size should be initialized in CC.MPS; and
   c. The I/O Command Set that is to be used should be selected in CC.CSS;

4. The controller should be enabled by setting CC.EN to ‘1’;
5. The host should wait for the controller to indicate the controller is ready to process commands. The controller is ready to process commands when CSTS.RDY is set to ‘1’;
6. The host should determine the configuration of the controller by issuing the Identify command, specifying the Controller data structure. The host should then determine the configuration of each namespace by issuing the Identify command for each namespace, specifying the Namespace data structure;
7. The host should determine:
    a. the maximum I/O Queue size using CAP.MQES; and
    b. the number of I/O Submission Queues and I/O Completion Queues supported using the response from the Set Features command with the Number of Queues feature identifier;
8. The host should use the Connect command (refer to section 3.3) to create I/O Submission and Completion Queue pairs; and
9. To enable asynchronous notification of optional events, the host should issue a Set Features command specifying the events to enable. The host may submit one or more Asynchronous Event Request commands to be notified of asynchronous events as described by section 5.2 in the NVMe Base specification. This step may be done at any point after the controller signals that the controller is ready (i.e., CSTS.RDY is set to ‘1’).

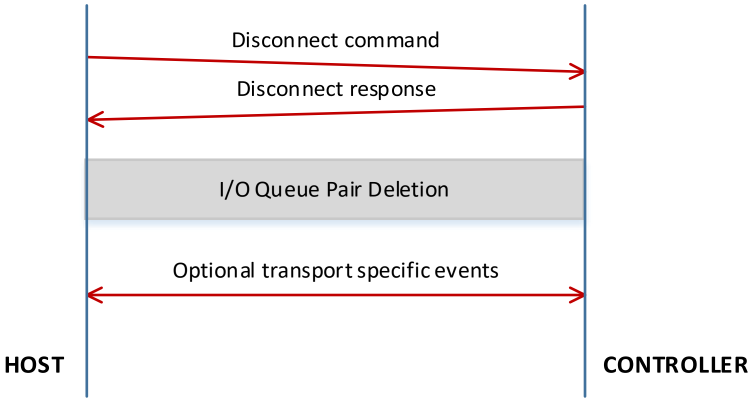
The association may be removed if step 4 (set CC.EN to ‘1’) is not completed within 2 minutes after establishing the association.

4.5 I/O Queue Deletion

Individual I/O Queues may be deleted by the host. Admin Queue deletion is not possible. Deletion of the Admin Queue requires termination of the entire association between the host and controller.

The host selects the I/O Queue to delete. The Disconnect command is used to request deletion of both the I/O Submission Queue and the I/O Completion Queue. Outstanding commands on the specified I/O Queue are completed or aborted by the controller before the Disconnect command is completed by the controller. The successful completion of the Disconnect command indicates an implicit completion status of Command Aborted due to SQ Deletion for any outstanding commands on that I/O Queue for which a completion queue entry has not been returned by the controller. Upon successful completion of the Disconnect command, the host and controller may release their respective transport resources and their queue resources that are dedicated to that I/O Queue. The host and controller may also use timers to determine when transport resources should be released or may retrain transport resources (e.g., an NVMe Transport connection) for reuse. Figure 32 is a ladder diagram that describes the queue deletion process for an I/O Queue where the transport resources dedicated to that queue, including an NVMe Transport connection, are released after the I/O Queue is deleted.

Figure 32 I/O Queue Deletion Flow



Sequence of events:

1) The host submits a Disconnect command to the I/O Queue that is to be deleted.
2) The controller processes the Disconnect command. As part of this processing, the controller completes or aborts all commands on this I/O Queue other than the Disconnect command.
3) The controller sends the Disconnect command response (i.e., completion) to the host.
4) Upon receipt of the Disconnect command completion, the host deletes its NVMe resources for the I/O Queue.
5) The host may send an optional transport-specific event to the controller to indicate that the host has processed the Disconnect command completion.
6) The controller deletes its NVMe resources for this I/O Queue.
7) The host removes the NVMe Transport connection and releases associated transport resources if the NVMe Transport connection is not associated with any other NVMe Queues.
8) The controller removes the NVMe Transport connection and releases associated transport resources if the NVMe Transport connection is not associated with any other NVMe Queues.

4.6 Shutdown

To initiate a shutdown of a controller, the host should use the Property Set command (refer to section 3.6) to set the Shutdown Notification (CC.SHN) field to:

- 01b to initiate a normal shutdown operation; or
- 10b to initiate an abrupt shutdown.

After the host initiates a shutdown, the host may either disconnect at the NVMe Transport level or the host may choose to poll CSTS.SHST to determine when the shutdown is complete (i.e., the controller should not initiate a disconnect at the NVMe Transport level). It is an implementation choice whether the host aborts all outstanding commands prior to initiating the shutdown.

The CC.EN field is not used to shutdown the controller (i.e., it is used for Controller Reset).

From the time a shutdown is initiated until:

- a Controller Level Reset occurs; or
- the controller, if dynamic, is removed from the NVM subsystem,

the controller shall:

- process only Fabrics commands (refer to Figure 14); and
- disable the Keep Alive timer, if supported.

After CC.EN transitions to ‘0’ (due to shutdown or reset), the association between the host and controller shall be preserved for at least 2 minutes. After this time, the association may be removed if the controller has not been re-enabled.

5 Discovery Service

NVMe over Fabrics defines a discovery mechanism that a host uses to determine the NVM subsystems that expose namespaces that the host may access. The Discovery Service provides a host with the following capabilities:

- The ability to discover a list of NVM subsystems with namespaces that are accessible to the host;
- The ability to discover multiple paths to an NVM subsystem;
- The ability to discover controllers that are statically configured;
- The optional ability to establish explicit persistent connections to the Discovery controller; and
- The optional ability to receive Asynchronous Event Notifications from the Discovery controller.

A Discovery Service is an NVM subsystem that supports only Discovery controllers. A Discovery controller supports minimal functionality and only implements features related to Discovery Log Pages and does not implement I/O Queues or expose namespaces. The functionality supported by the Discovery controller is defined in section 5.4.

The host uses the well-known Discovery Service NQN (nqn.2014-08.org.nvmexpress.discovery) in the Connect command (refer to section 3.3) to a Discovery Service. The method that a host uses to obtain the NVMe Transport information necessary to connect to the well-known Discovery Service is implementation specific.

The Discovery Log Page provided by a Discovery controller contains one or more entries. Each entry specifies information necessary for the host to connect to an NVM subsystem. An entry may be associated with an NVM subsystem that exposes namespaces or a referral to another Discovery Service. There are no ordering requirements for log page entries within the Discovery Log Page.

Discovery controller(s) may provide different log page contents depending on the Host NQN provided (e.g., different NVM subsystems may be accessible to different hosts). The set of Discovery Log entries should include all applicable addresses on the same fabric as the Discovery Service and may include addresses on other fabrics.

Discovery controllers that support explicit persistent connections shall support both Asynchronous Event Request and Keep Alive commands (refer to NVMe Base specification sections 5.2 and 5.16 respectively). A host requests an explicit persistent connection to a Discovery controller and Asynchronous Event Notifications from the Discovery controller on that persistent connection by specifying a non-zero Keep Alive Timer value in the Connect command. If the Connect command specifies a non-zero Keep Alive Timer value and the Discovery controller does not support Asynchronous Events, then the Discovery controller shall return a status value of Connect Invalid Parameters (refer to Figure 22) for the Connect command. Discovery controllers shall indicate support for Discovery Log Change Notifications in the Identify Controller Data Structure (refer to Figure 37).

Discovery controllers that do not support explicit persistent connections shall not support Keep Alive commands and may use a fixed Discovery controller activity timeout value (e.g., 2 minutes). If no commands are received by such a Discovery controller within that time period, the controller may perform the actions for Keep Alive Timer expiration defined in section 7.1.2.

A Discovery controller shall not support the Disconnect command.

A Discovery Log Page with multiple entries for the same NVM subsystem indicates that there are multiple fabric paths to the NVM subsystem, and/or that multiple static controllers may share a fabric path. The host may use this information to form multiple associations to controllers within an NVM subsystem.

Multiple entries for the same NVM subsystem with different Port ID values indicates that the resulting NVMe Transport connections are independent with respect to NVM subsystem port hardware failures. A host that uses a single association should pick the first / best record to attach to an NVM subsystem. A host that uses multiple associations should choose different ports.

A transport specific method may exist to indicate changes to a Discovery controller.

5.1 Discovery Controller Initialization

The initialization process for Discovery controllers is described in Figure 33.

Figure 33: Discovery Controller Initialization process flow

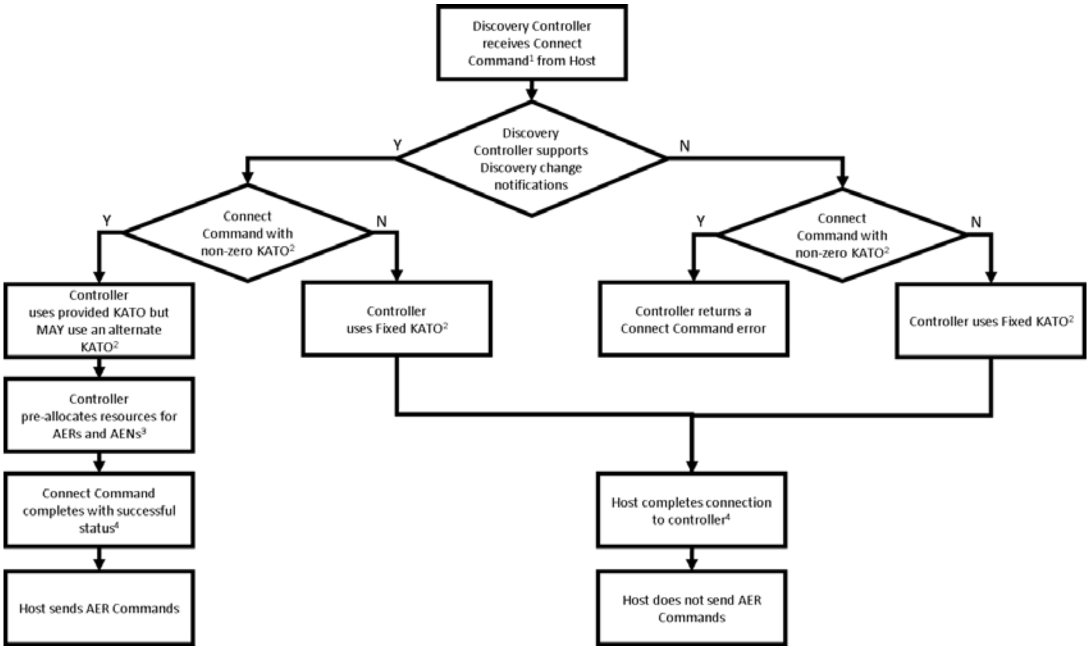

Discovery Controller receives Connect Command[1] from Host
Discovery Controller supports Discovery change notifications
Y
N
Connect Command with non-zero KATO[2]
Y
N
Connect Command with non-zero KATO[2]
Y
N
Controller uses provided KATO but MAY use an alternate KATO[2]
Controller uses Fixed KATO[2]
Controller returns a Connect Command error
Controller uses Fixed KATO[2]
Controller pre-allocates resources for AERs and AENs[3]
Connect Command completes with successful status[4]
Host sends AER Commands
Host completes connection to controller[4]
Host does not send AER Commands

Figure 33 references:

1. Refer to section 3.3;
2. Refer to NVMe Base specification Asynchronous Event Request command in section 5.2;
3. Refer to NVMe Base specification Keep Alive command in section 5.16; and
4. Refer to the following steps in this section.

After the Connect Command completes with a status of Successful Completion, the host performs the following steps:

1. NVMe authentication is performed if required (refer to section 6.2);
2. The host determines the controller's capabilities by reading the Controller Capabilities property;
3. The host configures the controller's settings by writing the Controller Configuration property, including setting CC.EN to '1' to enable command processing;
4. The host waits for the controller to indicate that the controller is ready to process commands. The controller is ready to process commands when CSTS.RDY is set to '1' in the Controller Status property; and
5. The host determines the features and capabilities of the controller by issuing the Identify command, specifying the Controller data structure.

After initializing the Discovery controller, the host reads the Discovery Log Page. Refer to section 5.3.

5.2 Discovery Controller Properties and Command Support

The supported properties for a Discovery controller are defined in Figure 34.

Figure 34: Discovery Controller – Properties

Start	End	Symbol	Description
00h	07h	CAP	Controller Capabilities CAP.MPSMAX shall be cleared to 0h. CAP.MPSMIN shall be cleared to 0h. CAP.CSS shall be set to 1h. CAP.NSSRS shall be cleared to 0h. CAP.AMS shall be cleared to 0h. CAP.CQR shall be set to 1b.
08h	0Bh	VS	Version
0Ch	0Fh	INTMS	Reserved
10h	13h	INTMC	Reserved
14h	17h	CC	Controller Configuration CC.IOCQES is reserved. CC.IOSQES is reserved. CC.AMS shall be cleared to 0h. CC.MPS shall be cleared to 0h. CC.CSS shall be cleared to 0h.
18h	1Bh	Reserved	Reserved
1Ch	1Fh	CSTS	Controller Status
20h	23h	NSSR	Reserved
24h	27h	AQA	Reserved
28h	2Fh	ASQ	Reserved
30h	37h	ACQ	Reserved
38h	3Bh	CMBLOC	Reserved
3Ch	3Fh	CMBSZ	Reserved
40h	FFFh	Reserved	Reserved
			Vendor Specific (Optional)

A Discovery Controller supports all mandatory Fabrics commands. A Discovery controller supports a subset of Admin commands shown in Figure 35.

Figure 35: Discovery Controller – Admin Commands

Opcode by Field			Combined Opcode[2]	O/M[1]	Namespace Identifier Used[3]	Command
(07)	(06:02)	(01:00)				
Generic Command	Function	Data Transfer[4]				
0b	000 00b	10b	02h	M	n/a	Get Log Page
0b	000 01b	10b	06h	M	n/a	Identify
0b	000 10b	01b	09h	NOTE 5	n/a	Set Features
0b	000 10b	10b	0Ah	NOTE 5	n/a	Get Features
0b	000 11b	00b	0Ch	NOTE 5	n/a	Asynchronous Event Request
0b	001 10b	00b	18h	NOTE 5	n/a	Keep Alive

NOTES:
1. O/M definition: O = Optional, M = Mandatory.
2. Opcodes not listed are reserved.
3. The Namespace Identifier field (CDW1.NSID) is reserved for Discovery controllers.
4. 00b = no data transfer; 01b = host to controller; 10b = controller to host; 11b = bidirectional
5. For Discovery controllers that do not support explicit persistent connections, the command is reserved. For Discovery controllers that support explicit persistent connections, the command is mandatory.

The Discovery controller shall support the Discovery Log Page. The log pages that a Discovery controller may support are shown in Figure 36.

Figure 36: Discovery Controller – Log Page Identifiers

Log Identifier	O/M[1]	Description
00h		Reserved
01h	O	Error Information
02h to 6Fh		Reserved
70h	M	Discovery
71h to BFh		Reserved
C0h to FFh		Vendor specific
NOTES: 1. O/M definition: O = Optional, M = Mandatory.		

The Discovery controller shall support the Identify command with a CNS value of 01h (Identify Controller data structure); all other CNS values are reserved. The Identify Controller data structure returned when CNS is 01h is defined in Figure 37.

Figure 37: Discovery Controller – Identify Controller Data Structure

Bytes	O/M[1]	Description
63:00		Reserved
71:64	M	**Firmware Revision (FR):** Refer to Figure 247 in the NVMe Base specification. Note: The firmware revision is only accessible via this mechanism as the Firmware Information log page is not supported.
76:72		Reserved
77	M	**Maximum Data Transfer Size (MDTS):** Refer to Figure 247 in the NVMe Base specification. Note: The CAP.MPSMIN and CAP.MPSMAX values are fixed at 0h. Thus, this field is reported in units of 4KB.
79:78	M	**Controller ID (CNTLID):** Refer to Figure 247 in the NVMe Base specification.
83:80	M	**Version (VER):** Refer to Figure 247 in the NVMe Base specification.
91:84		Reserved
95:92	M	**Optional Asynchronous Events Supported (OAES):** This field indicates the optional asynchronous events supported by the controller. A controller shall not send optional asynchronous events before they are enabled by host software. Bit 31 is set to ‘1’ if the controller supports sending Discovery Log Change Notifications. If cleared to ‘0’, then the controller does not support the Discovery Log Change Notification events. Bits 30:0 are reserved.
260:96		Reserved
261	M	**Log Page Attributes (LPA):** Refer to Figure 247 in the NVMe Base specification.
262	M	**Error Log Page Attributes (ELPE):** Refer to Figure 247 in the NVMe Base specification.
513:263		Reserved
515:514	M	**Maximum Outstanding Commands (MAXCMD):** Refer to Figure 247 in the NVMe Base specification.
535:516		Reserved
539:536	M	**SGL Support (SGLS):** Refer to Figure 247 in the NVMe Base specification.
767:540		Reserved
1023:768	M	**NVM Subsystem NVMe Qualified Name (SUBNQN):** Refer to Figure 247 in the NVMe Base specification.
2047:1024		Reserved for Discovery controller specific fields.
3071:2048		Reserved
4095:3072	O	Vendor Specific
NOTES: 1. O/M definition: O = Optional, M = Mandatory.		

5.3 Discovery Log Page (Log Identifier 70h)

The Discovery Log Page shall only be supported by Discovery controllers. The Discovery Log Page shall not be supported by controllers that expose namespaces for NVMe over PCIe or NVMe over Fabrics. The Discovery Log Page provides an inventory of NVM subsystems with which a host may attempt to form an association. The Discovery Log may be specific to the host requesting the log. The Discovery Log page is persistent across power cycles.

The Log Page Offset may be used to retrieve specific records. The number of records is returned in the header of the log page. The format for a Discovery Log Page entry is defined in Figure 38. The format for the Discovery Log is defined in Figure 39.

A single Get Log Page command used to read the Discovery Log Page shall be atomic. If the host reads the Discovery Log Page using multiple Get Log Page commands the host should ensure that there has not been a change in the contents of the data. The host should read the Discovery Log Page contents in order (i.e., with increasing Log Page Offset values) and then re-read the Generation Counter after the entire log page is transferred. If the Generation Counter does not match the original value read, the host should discard the log page read as the entries may be inconsistent. If the log page contents change during this command sequence, the controller may return a status of Discover Restart.

Every record indicates via the SUBTYPE field if that record is referring to another Discovery Service or if the record indicates an NVM subsystem composed of controllers that may expose namespaces. A referral to another Discovery Service is a mechanism to find additional NVM subsystems that contain controllers that may expose namespaces. Referrals shall not be deeper than eight levels.

If an NVM subsystem supports the dynamic controller model, then all entries for that NVM subsystem shall have the Controller ID field set to FFFFh. For a particular NVM subsystem port and NVMe Transport address in an NVM subsystem, there shall be no more than one entry with the Controller ID field set to:

- FFFFh if that NVM subsystem supports the dynamic controller model; or
- FFFEh if that NVM subsystem supports the static controller model.

Figure 38: Get Log Page – Discovery Log Page Entry

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>00</td><td><b>Transport Type (TRTYPE):</b> Specifies the NVMe Transport type.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>RDMA Transport (refer to section 7.3)</td></tr>
<tr><td>02</td><td>Fibre Channel Transport (refer to INCITS 540)</td></tr>
<tr><td>03</td><td>TCP Transport (refer to section 7.4)</td></tr>
<tr><td>04 to 253</td><td>Reserved</td></tr>
<tr><td>254</td><td>Intra-host Transport (i.e., loopback) (NOTE: This is a reserved value for use by host software.)</td></tr>
<tr><td>255</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>01</td><td><b>Address Family (ADRFAM):</b> Specifies the address family.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td><b>AF_INET:</b> IPv4 address family. IPv4address format syntax specified in section 3.2.2 of IETF RFC 3986.</td></tr>
<tr><td>02</td><td><b>AF_INET6:</b> IPv6 address family. IPv6address format syntax specified in section 3.2.2 of IETF RFC 3986.</td></tr>
<tr><td>03</td><td><b>AF_IB:</b> InfiniBand address family.</td></tr>
<tr><td>04</td><td>Fibre Channel address family.</td></tr>
<tr><td>05 to 253</td><td>Reserved</td></tr>
<tr><td>254</td><td>Intra-host Transport (i.e., loopback) (NOTE: This is a reserved value for use by host software.)</td></tr>
<tr><td>255</td><td>Reserved</td></tr>
</table></td></tr>
</table>

Figure 38: Get Log Page – Discovery Log Page Entry

<table>
<tr><th>Bytes</th><th>Description</th></tr>
<tr><td>02</td><td><b>Subsystem Type (SUBTYPE):</b> Specifies the type of the NVM subsystem that is indicated in this entry.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved.</td></tr>
<tr><td>01</td><td>The entry describes a referral to another Discovery Service composed of Discovery controllers for additional records.</td></tr>
<tr><td>02</td><td>The entry describes an NVM subsystem that is not associated with Discovery controllers and whose controllers may have attached namespaces.</td></tr>
<tr><td>03 to 255</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>03</td><td><b>Transport Requirements (TREQ):</b> Indicates requirements for the NVMe Transport.
Bits 7:3 are reserved.
Bit 2 if set to ‘1’ indicates that the controller is capable of disabling SQ flow control. A controller that is capable of disabling SQ flow control may accept or reject a host request to disable SQ flow control. If cleared to ‘0’, then the controller requires use of SQ flow control.
Bits 1:0 indicate whether connections shall be made over a fabric secure channel.
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00b</td><td>Not specified</td></tr>
<tr><td>01b</td><td>Required</td></tr>
<tr><td>10b</td><td>Not required</td></tr>
<tr><td>11b</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>05:04</td><td><b>Port ID (PORTID):</b> Specifies a particular NVM subsystem port. Different NVMe Transports or address families may utilize the same Port ID value (e.g., a Port ID may support both iWARP and RoCE).</td></tr>
<tr><td>07:06</td><td><b>Controller ID (CNTLID):</b> Specifies the controller ID. If the NVM subsystem uses a dynamic controller model, then this field shall be set to FFFFh. If the NVM subsystem uses a static controller model, then this field may be set to a specific controller ID (values 0h to FFEFh are valid). If the NVM subsystem uses a static controller model and the value indicated is FFFEh, then the host should remember the Controller ID returned as part of the Fabrics Connect command in order to re-establish an association in the future with the same controller.</td></tr>
<tr><td>09:08</td><td><b>Admin Max SQ Size (ASQSZ):</b> Specifies the maximum size of an Admin Submission Queue. This applies to all controllers in the NVM subsystem. The value shall be a minimum of 32 entries.</td></tr>
<tr><td>31:10</td><td>Reserved</td></tr>
<tr><td>63:32</td><td><b>Transport Service Identifier (TRSVCID):</b> Specifies the NVMe Transport service identifier as an ASCII string. The NVMe Transport service identifier is specified by the associated NVMe Transport binding specification.</td></tr>
<tr><td>255:64</td><td>Reserved</td></tr>
<tr><td>511:256</td><td><b>NVM Subsystem Qualified Name (SUBNQN):</b> NVMe Qualified Name (NQN) that uniquely identifies the NVM subsystem. Refer to section 7.9 of the NVMe Base specification. For a Discovery Service, the value returned shall be the well-known Discovery Service NQN (nqn.2014-08.org.nvmexpress.discovery).</td></tr>
<tr><td>767:512</td><td><b>Transport Address (TRADDR):</b> Specifies the address of the NVM subsystem that may be used for a Connect command as an ASCII string. The Address Family field describes the reference for parsing this field. Refer to section 1.5 of the NVMe Base specification for ASCII string requirements. For the definition of this field, refer to the appropriate NVMe Transport binding specification.</td></tr>
<tr><td>1023:768</td><td><b>Transport Specific Address Subtype (TSAS):</b> Specifies NVMe Transport specific information about the address. For the definition of this field, refer to the appropriate NVMe Transport binding specification.</td></tr>
</table>

Figure 39: Get Log Page – Discovery Log

Bytes	Description
07:00	**Generation Counter (GENCTR):** Indicates the version of the discovery information, starting at a value of 0h. For each change in the Discovery Log, this counter is incremented by one. If the value of this field is FFFFFFFF_FFFFFFFFh, then the field shall be cleared to 0h when incremented (i.e., rolls over to 0h).
15:08	**Number of Records (NUMREC):** Indicates the number of records contained in the log.
17:16	**Record Format (RECFMT):** Specifies the format of the Discovery Log. If a new format is defined, this value is incremented by one. The format of the record specified in this definition shall be 0h.
1023:18	Reserved
2047:1024	**Discovery Log Entry 0 (DLE0):** Contains the first Discovery Log Entry as defined in Figure 38.
3071:2048	**Discovery Log Entry 1 (DLE1):** Contains the second Discovery Log Entry as defined in Figure 38 (if present).
…	…
(((N + 2) × 1024) - 1): ((N + 1) × 1024)	**Discovery Log Entry N (DLEN):** Contains the Nth Discovery Log Entry as defined in Figure 38 (if present).

5.4 Discovery Controller Features and Command Support

These features indicate the attributes of a Discovery controller (refer to Figure 40). This is optional information not required for proper behavior of the system (refer to Figure 271 in the NVMe Base specification).

Figure 40: Set Features Identifier

Feature Identifier	O/M[2]	Persistent Across Power Cycle and Reset[1]	Uses Memory Buffer for Attributes	Description
00h to 0Ah				Reserved
0Bh	O	No	No	Asynchronous Event Configuration
0Ch to 0Eh				Reserved
0Fh	O	No	No	Keep Alive Timer
10h to BFh				Reserved
C0h to FFh				Vendor Specific[3]

NOTES:
1. This column is only valid if the feature is not saveable (refer to section 7.8 in the NVMe Base specification). If the feature is saveable, then this column is not used and any feature may be configured to be saved across power cycles and reset.
2. O/M definition: O = Optional, M = Mandatory.
3. The behavior of a controller in response to a vendor specific Feature Identifier is vendor specific.

5.4.1 Asynchronous Event Configuration (Feature Identifier 0Bh), (Optional)

Discovery controllers that support Asynchronous Event Notifications shall implement the Get Features and Set Features commands. A Discovery controller shall enable Asynchronous Discovery Log Event Notifications, if a non-zero KATO value is received in the Connect command (refer to section 3.3) sent to that controller.

Figure 41 defines Discovery controller Asynchronous Event Notifications.

Figure 41: Asynchronous Event Configuration – Command Dword 11

Bits	Description
31	**Discovery Log Page Change Notification:** This bit indicates that the Discovery controller reports Discovery Log Page Change Notifications. If set to ‘1’, the Discovery controller shall send a notification if Discovery Log Page changes occur.
30:00	Reserved

5.5 Discovery Controller Asynchronous Event Information – Requests and Notifications

If Discovery controllers detect events about which a host has requested notification, then the Discovery controller shall send an Asynchronous Event with the:

- Asynchronous Event Type field set to Notice (i.e., 2h);
- Log Page Identifier field set to Discovery (i.e., 70h); and
- Asynchronous Event Information field set as defined in Figure 39.

Figure 42: Asynchronous Event Information – Notice

Value	Description
F0h	**Discovery Log Page Change:** A change has occurred to one or more of the Discovery Log Pages. The host should submit a Get Log Page command to receive updated Discovery Log Pages.
F1h to FFh	Reserved for future NVMe-oF Asynchronous Event Notifications

5.5.1 Discovery Log Page Change Asynchronous Event Notification (Event Information F0h)

When a Discovery controller updates Discovery Log Page(s), the Discovery controller shall send a Discovery Log Page Change Asynchronous Event notification to each host that has requested asynchronous event notifications of this type.

6 Authentication

NVMe over Fabrics supports both fabric secure channel (that includes authentication) and NVMe in-band authentication. Fabric authentication is part of establishing a fabric secure channel via an NVMe Transport specific protocol that provides authentication, encryption, and integrity checking (e.g., IPsec; see RFC 4301). NVMe in-band authentication is performed immediately after a Connect command (refer to section 3.3) succeeds using the Authentication Send and Authentication Receive commands (refer to section 3) to tunnel authentication protocol commands between the host and the controller.

Enrollment of the host and controller in an authentication mechanism, including provisioning of authentication credentials to the host and controller, is outside the scope of this specification.

If both fabric secure channel and NVMe in-band authentication are used, the identities for these two instances of authentication may differ for the same NVMe Transport connection. For example, if an iWARP NVMe Transport is used with IPsec as the fabric secure channel technology, the IPsec identities for authentication are associated with the IP network (e.g., DNS host name or IP address), whereas NVMe in-band authentication uses NVMe identities (i.e., Host NQNs). The NVMe Transport binding specification may provide further guidance and requirements on the relationship between these two identities, but determination of which NVMe Transport identities are authorized to be used with which NVMe identities is part of the security policy for the deployed subsystem.

6.1 Fabric Secure Channel

The Transport Requirements field in the Fabrics Discovery Log Page Entry (refer to Figure 38) indicates whether a fabric secure channel shall be used for an NVMe Transport connection to an NVM subsystem. The secure channel mechanism is specific to the type of fabric.

If establishment of a secure channel fails or a secure channel is not established when required by the controller, the resulting errors are fabric-specific and may not be reported to the NVMe layer on the host. Such errors may result in the controller being inaccessible to the host via the NVMe Transport connection on which the authentication failure occurred.

An NVM subsystem that requires use of a fabric secure channel (i.e., as indicated by the TREQ field in the associated Discovery Log entry) shall not allow capsules to be transferred until a secure channel has been established for the NVMe Transport connection.

All Discovery Log Page Entries for an NVM subsystem should report the same value of TREQ to each host. Discovery Log Page Entries for an NVM subsystem may report different values of TREQ to different hosts.

6.2 NVMe In-band Authentication

The Authentication Requirements (AUTHREQ) field in the Connect response capsule (refer to Figure 21) indicates whether NVMe in-band authentication is required.

If one or more of the bits in the AUTHREQ field are set to ‘1’, then the controller requires that the host authenticate on each queue via one of the indicated security protocols in order to proceed with Fabrics, Admin, and I/O commands. Authentication success is defined by the specific security protocol that is used for authentication. If any command other than Connect, Authentication Send, or Authentication Receive is received prior to authentication success, then the controller shall abort the command with Authentication Required status.

If all bits in the AUTHREQ field are cleared to ‘0’, then the controller does not require the host to authenticate, and the subsystem shall not abort any command with a status value of Authentication Required.

A controller shall report the same value of AUTHREQ in the Connect response capsules sent by all of that controller’s queues. All controllers in an NVM subsystem should report the same value of AUTHREQ.

If NVMe in-band authentication succeeds, then:

1) any supported commands for the associated queue type may be processed; and
2) if an Authentication Send or an Authentication Receive command is received, then that command shall be aborted with a status value of Command Sequence Error.

6.2.1 NVMe In-band Authentication Protocol-Specific Requirements

Authentication requirements for security commands are based on the security protocol indicated by the SECP field in the command.

6.2.1.1 NVMe In-band Authentication Requirements for the TCG Security Protocols

For the TCG Security Protocols (i.e., bit 00 is set to ‘1’ in the AUTHREQ field), security commands specifying security protocol values 01h to 06h do not require authentication when used for NVMe in-band authentication. When used for other purposes, in-band authentication of these commands is required. The TCG Storage Interface Interactions Specification (SIIS) and associated specifications specify the subset of the TCG security protocols used for NVMe in-band authentication.

7 Transport Definition

7.1 Transport Requirements

This section defines requirements that all NVMe Transports that support an NVMe over Fabrics implementation shall meet.

The NVMe Transport may support NVMe Transport error detection and report errors to the NVMe layer in command status values. The controller may record NVMe Transport specific errors in the Error Information Log. Transport errors that cause loss of a message or loss of data in a way that the low-level NVMe Transport cannot replay or recover should cause:

- the deletion of the individual I/O Queues (refer to section 4.5) and the associated NVMe Transport connection on which that NVMe Transport level error occurred; or
- termination of the NVMe Transport connection and end ending of the association between the host and controller.

The NVMe Transport shall provide reliable delivery of capsules between a host and NVM subsystem (and allocated controller) over each connection. The NVMe Transport may deliver command capsules in any order on each queue except for I/O commands that are part of fused operations (refer to section 4.12 of the NVMe Base specification).

For command capsules that are part of fused operations for I/O commands, the NVMe Transport:

a) shall deliver the first and second command capsules for each fused operation to the queue in-order; and
b) shall not deliver any other command capsule for the same Submission Queue between delivery of the two command capsules for a fused operation.

The NVMe Transport shall provide reliable delivery of response capsules from an NVMe subsystem to a host over each connection. The NVMe Transport shall deliver response capsules that include an SQ Head Pointer (SQHD) value to the host in-order; this includes all Connect response capsules and all Disconnect response capsules.

7.1.1 Submission Queue Head Pointer Update Optimization

This optimization does not apply to queue pairs for which Submission Queue (SQ) flow control is disabled, as the SQHD field is reserved if SQ flow control is disabled, refer to section 2.4 and to section 3.3.

The NVMe Transport may omit transmission of the SQHD value for a response capsule that:

a) contains a Generic Command status (i.e., Status Code Type 0h) indicating successful completion of a command (i.e., Status Code 00h);
b) is not a Connect response capsule; and
c) is not a Disconnect response capsule.

If a new SQHD value is not received in a response capsule, the host continues to use its previous SQHD value. Thus, at the NVMe layer there is a logical progression of SQHD values despite the fact that the NVMe Transport may not actually transfer the SQHD value in each response capsule.

The NVMe Transport may deliver response capsules that do not contain an SQHD value to the host in any order. The applicable NVMe Transport binding specification defines how presence versus absence of an SQHD value in a response capsule is indicated by the NVMe Transport.

Periodic SQHD updates at the host are required to avoid Submission Queue (SQ) starvation as SQHD value transmission in responses is the only means of releasing SQ slots for host reuse.

An NVMe Transport may transmit an SQHD value in every response capsule. If an NVMe Transport does not transmit an SQHD value in every response capsule, then an SQHD value should be transmitted periodically (e.g., in at least one of every n response capsules on a CQ, where n is 10% of the size of the associated SQ) or more often. An SQHD value should always be transmitted if 90% or more of the slots in the associated SQ are occupied at the subsystem.

7.1.2 Keep Alive

The Keep Alive feature is defined in section 7.12 of the NVMe Base specification. The Transport binding specification indicates whether the Keep Alive feature is required.

The controller shall treat a Keep Alive Timeout in the same manner as connection loss. If the Keep Alive feature is in use and the timer expires, then the controller shall:

- stop processing commands and set the Controller Fatal Status (CSTS.CFS) bit to ‘1’;
- terminate the NVMe Transport connection; and
- break the host to controller association.

After completing these steps, a controller may accept a Connect command (refer to section 3.3) for the Admin Queue from the same or another host in order to form a new association.

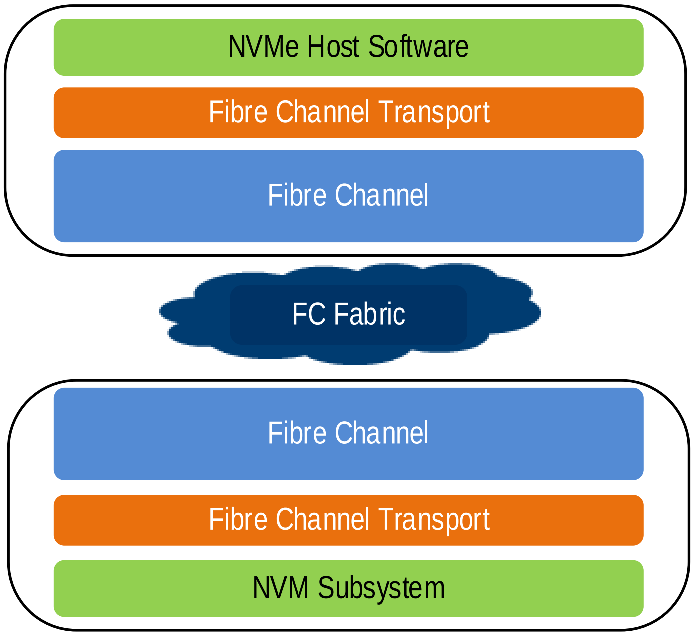
7.2 Transport Capsule and Data Binding: Fibre Channel

The Fibre Channel Technical Committee (ANSI/INCITS TC T11) has defined a transport binding for NVMe over Fabrics. The Fibre Channel Transport maps NVMe capsules onto Fibre Channel frames using the NVMe over FC protocol (FC-NVMe).

The binding of an NVMe implementation using the Transport Type of Fibre Channel Transport as defined in Figure 38 is specified in INCITS 540 Fibre Channel – Non-Volatile Memory Express (FC-NVMe). See http://www.t11.org for more information on the Fibre Channel Technical Committee and http://www.incits.org for information on how to purchase Fibre Channel standards.

The diagram in Figure 43 illustrates the layering of the Fibre Channel Transport within the host and NVM subsystem.

Figure 43: Fibre Channel Transport Protocol Layers



7.3 Transport Capsule and Data Binding: RDMA

This section defines the binding of an NVMe implementation that uses the Transport Type of RDMA Transport as defined in Figure 38. Common definitions used for RDMA are defined in Figure 44.

Figure 44: RDMA Definitions

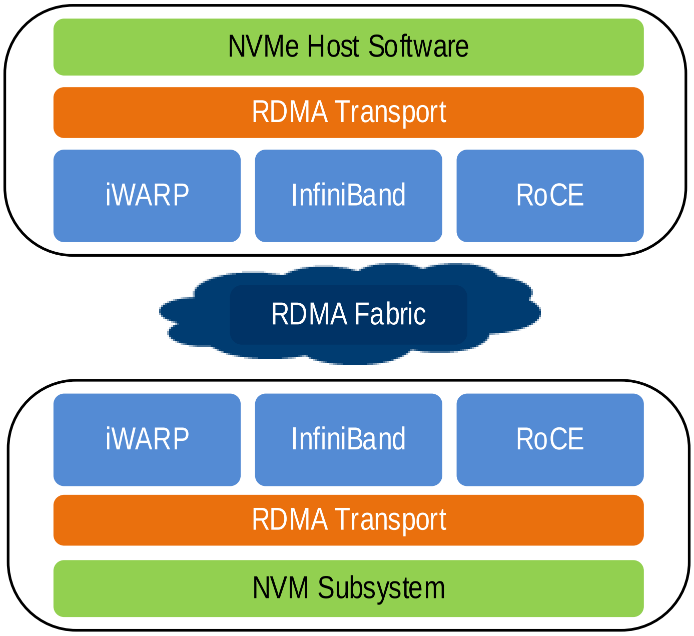
Term	Definition
direct data placement	The use of RDMA_READ or RDMA_WRITE to place data exchanged over the RDMA fabric directly into a host or an NVM subsystem memory buffer as specified in the command.
Established RDMA QP	Two RDMA Queue Pair endpoints that have an established association between them.
Host Memory Buffer Address	The RDMA Memory Key, byte offset, and byte length, identify a host memory buffer within an RDMA Memory Region or Memory Window.
in-order delivery	The use of RDMA_SEND to deliver capsules over the RDMA fabric in the same order that the capsules were submitted to the RDMA Transport for transmission for a given Submission or Completion Queue.
InfiniBand™ R_Key	Term used to desribed a remote Memory Region or Window in InfiniBand™ RDMA implementations.
InfiniBand™ RDMA	InfiniBand™ Trade Association definition of RDMA. Refer to www.infinibandta.org.
iWARP RDMA	IETF standard definition of RDMA. Refer to https://tools.ietf.org/html/rfc5040.
iWARP STag	Term used to describe a local or remote Memory Region or Window in iWARP RDMA implementations.
RDMA_LOCAL_INVALIDATE	RDMA operation used to invalidate the local system’s memory key.
RDMA Memory Key (RKEY)	Component of the Host Memory Buffer Address that associates a host buffer with an RDMA Memory Region or Memory Window. For the RDMA Transport, this is either an iWARP STag or InfiniBand™ R_KEY.
RDMA Memory Region	A range of host memory that has been registered with a host-resident RDMA device.
RDMA Memory Window	A range of host memory residing within a registered RDMA Memory Region.
RDMA NIC (RNIC)	RDMA enabled network adapter.
RDMA Queue Pair (QP)	RDMA communication endpoint that consists of a Send Queue and Receive Queue.
RDMA_READ	RDMA operation used to read from the remote system’s memory buffer to the local system’s memory buffer.
RDMA_SEND	RDMA operation used to send a message from the local peer’s QP Send Queue to the remote peer’s QP Receive Queue or Shared Receive Queue.
RDMA_SEND_INVALIDATE	RDMA operation used to perform the RDMA_SEND operation and invalidate the memory Key on the remote system.
RDMA_WRITE	RDMA operation used to write memory buffer(s) from the local system’s memory to the remote system’s memory buffer(s).
RDMA Verbs	Common functional definition and implementation of the RDMA operational programming model between applications and RDMA providers. Applications are the consumers of RDMA operations and RDMA providers are the various implementations of RDMA operations, such as InfiniBand™, iWARP, RoCE, etc. Refer to https://tools.ietf.org/html/draft-hilland-rddp-verbs-00.
Reliable Connected QP	Two RDMA Queue Pair endpoints connected with reliable in-order communications of RDMA messages and data.
Reliable Datagram QP	Two or more Queue Pair endpoints using a reliable datagram channel to exchange RDMA messages and data.
RoCE and RoCEv2 RDMA	RDMA over Converged Ethernet definition. Refer to www/infinibandta.org.

7.3.1 Transport Overview

The RDMA Transport provides reliable in-order capsule delivery and direct data placement of Admin and I/O command data through use of the RDMA reliable QP modes (Reliable Connected and Reliable Datagram). Use of RDMA unreliable QP modes is not supported. Refer to the RDMA specifications and RFCs for a description of RDMA QP modes.

The RDMA Transport is RDMA Provider agnostic. The diagram in Figure 45 illustrates the layering of the RDMA Transport and common RDMA providers (iWARP, InfiniBand™, and RoCE) within the host and NVM subsystem.

Figure 45: RDMA Transport Protocol Layers



The RDMA Transport uses a common set of RDMA operations to facilitate the exchange of command capsules, response capsules, and data. These operations are RDMA_SEND, RDMA_SEND_INVALIDATE, RDMA_READ, and RDMA_WRITE. The RDMA Transport uses RDMA buffer registration and invalidation operations to facilitate the use of host or NVM subsystem resident buffers for the exchange of data and metadata for Admin and I/O commands.

In some host and NVM subsystem implementations, the interface between the RDMA Transport and the RDMA Providers is defined by an implementation of RDMA Verbs. When applicable, the Host and NVM subsystem RDMA Transport implementations should use the common RDMA Verbs software interfaces in order for the RDMA Transport layer to be RDMA provider agnostic.

7.3.2 Capsules and SGLs

The capsule size for Fabrics commands are fixed in size regardless of whether commands are submitted on an Admin Queue or an I/O Queue. The command capsule size is 64 bytes and the response capsule size is 16 bytes.

The capsule sizes for the Admin Queue are fixed in size. The command capsule size is 64 bytes and the response capsule size is 16 bytes. In-capsule data is not supported for the Admin Queue.

Command capsules for I/O commands sent on I/O Queues may contain up to the I/O command capsule size reported at the I/O Queue Command Capsule Supported Size (IOCCSZ) field multiple by 16. The response capsule size shall be 16 bytes and shall not contain in-capsule data.

The RDMA Transport facilitates the use of separate locations for SGLs and data (refer to section 2.3.1).

Figure 46: RDMA Capsule Size and SGL Mapping

Capsule Type	Capsule Size	SGL Type
Fabrics and Admin Commands	64 bytes	Host-resident data buffer only.
Fabrics and Admin Responses	16 bytes	n/a
I/O Queue Command	(IOCCSZ * 16) bytes	Host-resident data buffer or in-capsule data.
I/O Queue Response	16 bytes	n/a

Admin command data is transferred using host-resident data buffers specified in Keyed SGL Data Block descriptor entries. I/O command data is transferred using host-resident data buffers specified in Keyed SGL Data Block descriptor entries or within the capsule. The RDMA Transport supports the SGL Data Block with Sub Type Offset, SGL Last Segment with Sub Type Offset, and Keyed SGL Data Block descriptors only. The RDMA Transport does not support SGLs in host memory; all SGLs shall be contained in the command capsule. Fabrics and Admin commands have one Transport SGL Data Block descriptor or Keyed SGL Data Block descriptor (i.e., there are no SGL descriptors following the Submission Queue Entry). I/O commands may have more than one SGL descriptor.

The controller shall set bits 01:00 of the SGLS field to 01b in the Identify Controller data structure (refer to Figure 247 in the NVMe Base specification) (i.e., the controller shall support SGLs and impose no alignment or granularity requirements for data blocks).

There are SGL Descriptor Sub Type values that are specific to RDMA operation as defined in Figure 47.

Figure 47: SGL Sub Types Specific to RDMA

Descriptor Types	Sub Type	Sub Type Description
All	Ah to Eh	Reserved
Keyed SGL Data Block (4h)	Fh	**Invalidate Key:** The host uses this Sub Type to specify that the controller should remotely invalidate the RKEY. If the controller does not support remote invalidate, then this Sub Type is ignored.

7.3.3 Queue Mapping

A single I/O Submission Queue and I/O Completion Queue pair shall be mapped to a single RDMA Queue Pair. Multiplexing multiple I/O Submission Queues and I/O Completion Queues onto a single RDMA Connected Queue pair is not supported. Spanning a single I/O Submission Queue and I/O Completion Queue pair across multiple RDMA Queue pairs is not supported.

7.3.4 Capsule and Data Exchange

Capsule exchanges are performed using the RDMA_SEND or RDMA_SEND_INVALIDATE operations. The RDMA Transport at the host uses RDMA_SEND to transmit command capsules. The RDMA Transport at the controller uses RDMA_SEND or RDMA_SEND_INVALIDATE to transmit response capsules and optionally invalidate a host memory key. An RDMA_SEND contains at most one command or response capsule.

All RDMA_READ and RDMA_WRITE operations are initiated by the controller. Data transfers from a controller to a host are performed using the RDMA_WRITE operation. Data transfers from a host buffer to a controller buffer are performed using the RDMA_READ operation. Data for an I/O command may also be exchanged from the host to the controller using a single RDMA_SEND operation that contains the command capsule and in-capsule data within the RDMA_SEND message payload.

Host memory buffer addresses are communicated in the command's SGL entries. Host memory buffer addresses are represented as the combination of a memory key, offset, and length. The host is responsible for allocating the memory buffers, registering the keys, and constructing the SGL entries with the associated memory buffer address.

To ensure proper command data to command completion ordering, all RDMA_WRITE data transfer operations for a command shall be submitted onto the same RDMA QP prior to submitting the associated response capsule onto the same RDMA QP.

The detailed flow of the command sequences using RDMA operations is shown in Figure 48.

Figure 48: Command Sequence Using RDMA Operations

Data Transfer Direction	Command Sequence Using RDMA Operations
No data to transfer	• The host transmits the command capsule to the controller using an RDMA_SEND operation. • Command action completed by the controller. • The controller transmits the response capsule to the host using an RDMA_SEND operation. • The same RDMA QP shall be used for both RDMA_SEND operations.
Controller to host	• The host transmits the command capsule to the controller using an RDMA_SEND operation. The capsule contains or points to SGL(s) required for the data transfer. • The controller uses RDMA_WRITE operation(s) to transfer data from the controller to the host. Each RDMA_WRITE is associated with one keyed remote host memory buffer (SGL) and one or more local controller memory buffer(s). • The controller transmits the response capsule to the host using an RDMA_SEND or (optionally) RDMA_SEND_INVALIDATE operation. • The same RDMA QP shall be used for the RDMA_WRITE operation(s) and the RDMA_SEND operation.
Host to controller	• The host transmits the command capsule to the controller using an RDMA_SEND operation. The capsule contains or points to SGL(s) required for the data transfer. The capsule may contain in-capsule data, which is pointed to by an offset address in an SGL within the capsule. • For host-resident command data, the controller uses RDMA_READ operations to transfer the data from the host to the controller. Each RDMA_READ is associated with one keyed remote memory buffer (SGL) and one or more local memory buffer(s). • The controller transmits the response capsule to the host using an RDMA_SEND or (optionally) RDMA_SEND_INVALIDATE operation. • The same RDMA QP shall be used for the RDMA_READ operation(s) and the RDMA_SEND operation.

7.3.5 Keep Alive Settings

Keep Alive functionality is not supported by all RDMA provider types at the RDMA Transport layer. As a result, the RDMA Transport requires the use of the Keep Alive Feature (refer to section 5.21.1.15 in the NVMe Base specification). It is recommended that any RDMA provider level functionality be disabled to avoid redundant and conflicting policies.

The RDMA 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.

7.3.6 Setup and Initialization

7.3.6.1 Transport Specific Address Subtype and Transport Service Identifier

The Discovery Log Entry includes a Transport Specific Address Subtype (TSAS) field that is defined in Figure 49 for the RDMA Transport.

Figure 49: Transport Specific Address Subtype Definition for RDMA Transport

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

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

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

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

These three requirements apply to all RDMA Provider Types.

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

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

7.3.6.2 Discovery Log Page Entry Fields

The RDMA Transport's use of the Discovery Log Page Entry fields shown in Figure 38 is dependent on the value of the RDMA_CMS field. Figure 50 shows the use of these fields. Refer to section 1.5 in the NVMe Base specification for ASCII string format requirements.

Figure 50: RDMA Transport Discovery Log Page Entry Usage by RDMA_CMS Value

<table>
<tr><th>RDMA_CMS Value</th><th>Discovery Log Page Entry Field</th><th>Discover Log Page Entry Field Valuese</th></tr>
<tr><td rowspan="3">RDMA_CM</td><td>ADRFAM</td><td>IPV4, IPV6, AF_IB</td></tr>
<tr><td>TR_ADDR</td><td>The value of this field is dependent on the value of the ADRFAM Field. IPv4 and IPv6 address format is described in Figure 38. AF_IB address format is an InfiniBand™ GUID in ASCII string format.</td></tr>
<tr><td>TRSVCID</td><td>The value of this field is dependent on the value of the TR_ADDR field as shown.
<table>
<tr><th>TR_ADDR Value</th><th>TRSVCID Value (ASCII string)</th></tr>
<tr><td>IPv4 or IPv6</td><td>TCP or UDP port number: decimal number represented as an ASCII string with no leading zeroes.</td></tr>
<tr><td>AF_IB</td><td>InfiniBand™ Service ID represented as 16 case insensitive ASCII hexadecimal digits ('0'-'9', 'a'-'f', 'A-F) in groups of 4 separated by hyphens ('-'), e.g., dead-beef-0123-3210</td></tr>
</table></td></tr>
</table>

7.3.6.3 Disabling Submission Queue Flow Control

Hosts and controllers that use the RDMA Transport may disable Submission Queue flow control as part of creating a Submission Queue and Completion Queue pair, refer to section 2.4 and to section 3.3.

7.3.6.4 Fabric Dependent Settings

As part of the RDMA QP creation, the host and remote-peer exchange the RDMA_READ resources. This exchange is typically facilitated by the RDMA Provider.

The host RDMA Transport sets the Inbound RDMA Read (IRD) value and passes that value to the remote-peer using these facilities. The remote-peer uses the host's IRD value to limit the number of RDMA_READ operations that the remote-peer issues to the host. Exceeding this limit may result in an RDMA QP error. The remote-peer returns an IRD value of zero to indicate that any host initiated RDMA_READ operations results in an RDMA QP error.

The host RDMA Transport sets the value of RNR_RETRY based on the RDMA provider capability. If the RDMA provider supports RNR_RETRY, then the host should set RNR_RETRY to infinite (value of 7). If the RDMA provider does not support RNR_RETRY, then the host should set the RNR_RETRY to zero. The remote-peer shall match the host RNR_RETRY setting or fail the QP creation.

Use of RDMA Private Data for the exchange of NVMe parameters:

Figure 51: RDMA_CM_REQUEST Private Data Format

Bytes	Description
01:00	**Record Format (RECFMT):** Specifies the format of the RDMA Private Data. If a new format is defined, this value is incremented by one. The format of the record specified in this definition shall be 0h.
03:02	**Queue ID (QID):** Refer to the Queue ID definition in the Connect command in Figure 19.
05:04	**RDMA QP Host Receive Queue Size (HRQSIZE):** This field indicates the number of RDMA QP receive queue entries allocated by the host's RDMA Transport for capsule reception.
07:06	**RDMA QP Host Send Queue Size (HSQSIZE):** This field indicates the number of RDMA QP send queue entries allocated by the host's RDMA transport for capsule transmission. The value shall be set to the Submission Queue Size (SQSIZE). Refer to the SQSIZE definition in the Connect command in Figure 19. This is a 0's based value.
09:08	**Controller ID (CNTLID):** This field is used to indicate the Controller ID if the RDMA QP is for an I/O Queue. If the QID field is cleared to 0h (i.e., the RDMA QP is for an Admin Queue), this field shall be cleared to 0h. If the QID field is non-zero, this field should be set to the value of the Controller ID associated with the creation of this queue.
31:10	Reserved

Figure 52: RDMA_CM_ACCEPT Private Data Format

Bytes	Description
01:00	**Record Format (RECFMT):** Specifies the format of the RDMA Private Data. If a new format is defined, this value is incremented by one. The format of the record specified in this definition shall be 0h.
03:02	**RDMA QP Controller Receive Queue Size (CRQSIZE):** This field indicates the number of RDMA QP receive queue entries allocated by the controller's RDMA Transport for capsule reception. RDMA Transports that use RNR_RETRY flow control may set this entry to be less than or equal to the value of HSQSIZE specified in Figure 51. RDMA Transports that do not use RNR_RETRY shall set this value to be equal to the value of HSQSIZE specified in Figure 51.
31:04	Reserved

Figure 53: RDMA_CM_REJECT Private Data Format

Bytes	Description
01:00	**Record Format (RECFMT):** Specifies the format of the RDMA Private Data. If a new format is defined, this value is incremented by one. The format of the record specified in this definition shall be 0h.
03:02	**Status (STS):** Specifies status for the associated RDMA_CM_REQUEST that is paired with this reject response. The valid status values are specified in Figure 54.

7.3.7 Key Management

An RDMA Memory Key (RKEY) is an identifier that associates a data buffer in host memory with a registered "remote access enabled" RDMA memory region or memory window on an RDMA NIC (RNIC) attached to a host. The host NVMe RDMA Transport software manages the creation of RKEYs, association of the RKEYs to data buffers, insertion of RKEYs into SGL entries, and invalidation of the RKEYs upon completion of Fabrics, Admin, or I/O commands. Refer to section 4.4 in the NVMe Base specification for the definition of the Keyed SGL Data Block descriptor.

Commands that require data transfers between a host memory buffer and the controller shall use SGLs that contain a full RDMA host address tuple consisting of an RKEY, Offset, and Length. The host NVMe RDMA Transport is responsible for allocating this tuple by registering the associated data buffers with the appropriate RNIC to produce the RKEY and then subsequently inserting the RKEY, Offset, and Length into the SGL entries associated with the command. The same RKEY may be used in multiple SGL entries associated with the same Fabrics, Admin, or I/O command. The RKEY shall be invalidated only after all RDMA_READ or RDMA_WRITE operations have been completed that use the RKEY.

The host RDMA Transport software selects one of two methods to invalidate the RKEY: local invalidate or remote invalidate. To indicate a remote invalidate, the host sets the Sub Type field in the Keyed SGL Data Block descriptor to Fh (refer to Figure 47). If the controller RDMA Transport does not support remote invalidate, then the host's request for remote invalidation is ignored.

The controller RDMA Transport may or may not honor the remote invalidate request. If honored, the controller RDMA Transport invalidates the RKEY by using the RDMA_SEND_INVALIDATE operation to return the capsule response. If the command capsule contains multiple SGL entries with the remote invalidate bit set, the controller RDMA Transport shall only invalidate the RKEY in the last Keyed SGL Data Block descriptor.

The host RDMA Transport shall check the RDMA receive completion to determine if the RDMA_SEND_INVALIDATE was received and checks the value of the RKEY that was invalidated. If the controller RDMA Transport did not invalidate the RKEY as requested, the host is responsible for invalidating the RKEY using a local invalidate operation.

7.3.8 Error Handling

7.3.8.1 RDMA Transport Errors

Errors detected by the RDMA Transport may result in the termination of any command capsule, response capsule, or data transfer operations and may result in the tear down of the RDMA QP(s). The RDMA Transport may detect errors that are not directly associated with a capsule or data transfer operation (e.g., tear down of the RDMA QP due to connection loss, data corruption, or protection error). In the case of a RDMA QP tear down, the RDMA Transport is responsible for terminating the RDMA QP, freeing up any NVMe Transport resources, and then informing the NVMe layer about the termination and the associated cause.

Errors detected by the RDMA Transport during RDMA QP establishment are handled within the RDMA Transport and are not reported to the NVMe layer. These errors are described in Figure 54.

Figure 54: RDMA Transport Errors

Value	Description
1h	**RDMA Invalid Private Data Length:** The host sent an incorrect private data size.
2h	**RDMA Invalid RECFMT:** The host sent an invalid RECFMT.
3h	**RDMA Invalid QID:** The host sent an invalid QID.
4h	**RDMA Invalid HSQSIZE**: The host sent an invalid HSQSIZE.
5h	**RDMA Invalid HRQSIZE:** The host sent an invalid HRQSIZE.
6h	**RDMA No Resources**: The controller-side RDMA transport is unable to create the RDMA QP due to lack of resources.
7h	**RDMA Invalid IRD:** The host sent an invalid IRD value.
8h	**RDMA Invalid ORD:** The host sent an invalid ORD value.
9h	**RDMA Invalid CNTLID:** The host sent an invalid CNTLID value.
Ah to FFh	Reserved

7.3.8.2 RDMA Provider Errors

Errors detected by the RDMA Provider are communicated to the local NVMe RDMA Transport through implementation specific interfaces (e.g., RDMA Verbs). RDMA Providers may have facilities to communicate errors to the RDMA QP remote peer. Details of these facilities or their use is outside the scope of this specification. Details on the types of errors and their associated identification encoding is contained within the RDMA Provider specifications.

7.4 Transport Capsule and Data Binding: TCP

This section defines the binding of an NVMe implementation that uses the Transport Type of TCP Transport [RFC 793] as defined by Figure 38. Common definitions used for TCP are defined in Figure 55.

Figure 55: TCP Definitions

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

Figure 56: NVMe/TCP Definitions

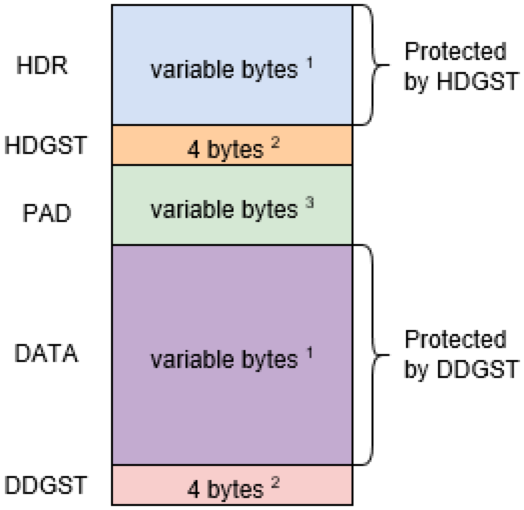
Term	Definition
CH	PDU Common Header
CPDA	Controller PDU Data Alignment
C2H	Controller to Host Direction
C2HTermReq	Controller to Host Terminate Connection Request
DATA	PDU Data
DATAL	H2CData and C2HData PDU Data Length
DATAO	H2CData and C2HData PDU Data Offset
DDGST	PDU Data Digest
HDGST	PDU Header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

7.4.1 Transport Overview

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

A host and a controller in an NVM subsystem communicate over TCP by exchanging NVMe/TCP Protocol Data Units (NVMe/TCP PDUs). An NVMe/TCP PDU may be used to transfer a capsule, data, or control/status information. As shown in Figure 57, an NVMe/TCP PDU consists of five parts. The PDU Header (HDR) is required in all PDUs. The PDU Header Digest (HDGST), the PDU Padding field (PAD), the PDU Data field (DATA), and the Data Digest (DDGST) field are included or omitted based on the PDU type and the values exchanged during connection establishment (refer to section 7.4.10).

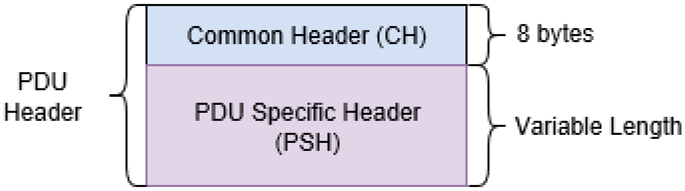
Figure 57: NVMe/TCP PDU Structure



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

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

Figure 58: NVMe/TCP PDU Header



As shown in Figure 59 and Figure 60, 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 59: Multiple NVMe/TCP PDUs in a single TCP/IP packet

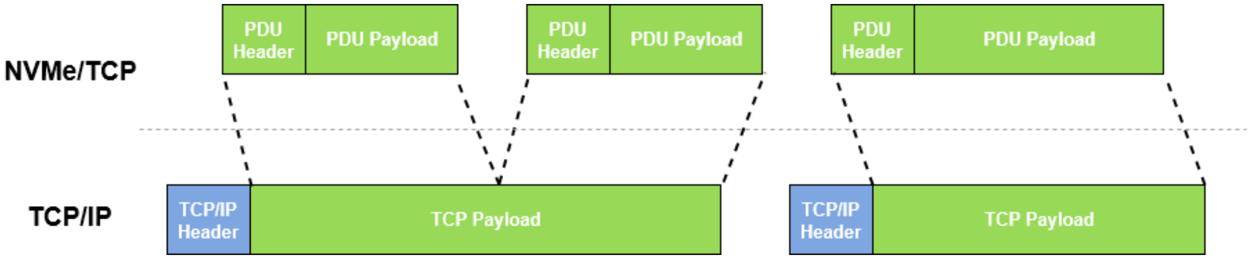
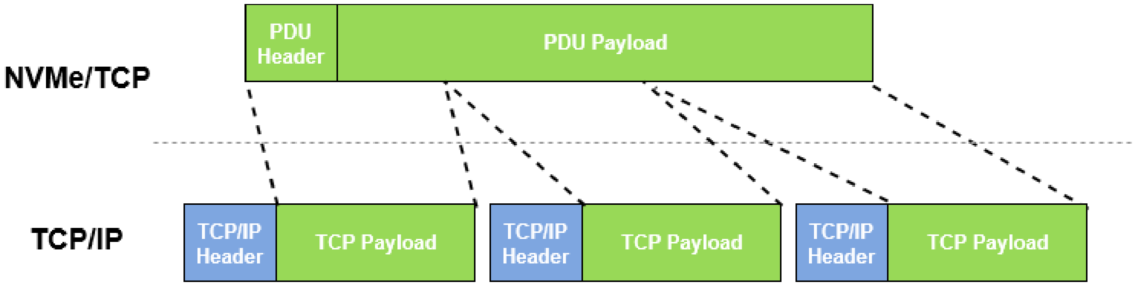


Figure 60: NVMe/TCP PDU spanning across TCP/IP packets



The TCP transport defines NVMe/TCP PDU types that are summarized in Figure 61 and defined in section 7.4.10. Associated with each NVMe/TCP PDU type is a direction that specifies whether the NVMe/TCP PDU is transferred from a host to a controller (H2C) or from a controller to a host (C2H). An NVMe/TCP PDU that is transferred in a direction opposite from that with which it is associated is treated as a fatal transport error (refer to section 7.4.7).

Figure 61: NVMe/TCP PDU Types

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

Figure 61: NVMe/TCP PDU Types

PDU Name	Opcode by field		Combined Opcode [2]	Section	PDU Description
	Function (07:01)	PDU Direction [1] (00)			
H2CData	0000011b	0b	06h	7.4.10.8	**Host to Controller Data:** A PDU sent from a host to a controller to transfer data to the controller
C2HData	0000011b	1b	07h	7.4.10.9	**Controller to Host Data:** A PDU sent from a controller to a host to transfer data to the host
R2T	0000100b	1b	09h	7.4.10.10	**Ready to Transfer:** A PDU sent from a controller to a host to indicate that the controller is ready to accept data

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

7.4.1.1 Conventions

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

7.4.2 Queue Mapping

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

7.4.3 Capsules

The NVMe/TCP transport supports a message model. Data transfers are supported via a transport specific data transfer mechanism, described in section 7.4.5, and optionally via in-capsule data. NVMe/TCP capsule sizes are summarized in Figure 62. 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 data to be transferred, then the NVMe/TCP data transfer mechanism described in section 7.4.5.3 shall be used. In-capsule data is not supported for Fabrics and Admin Response capsules, causing their maximum size to be 16 bytes. Response data is transferred using the data transfer mechanism described in section 7.4.5.2.

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

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

Figure 62: NVMe/TCP Capsule Size

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

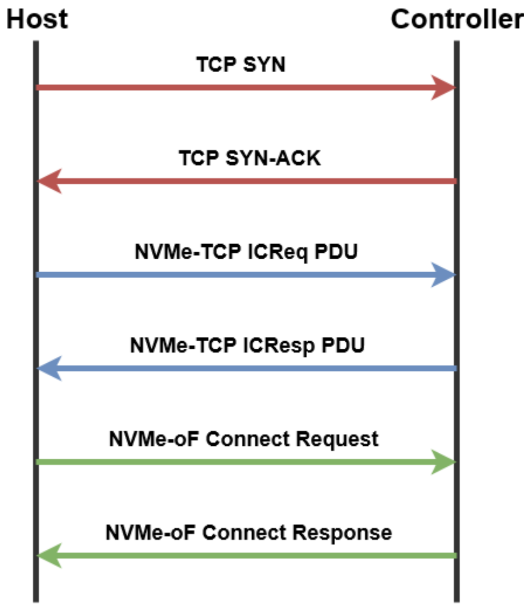
NOTES:
1. NVMe/TCP controllers must support in-capsule data for Fabrics and Admin command capsules.
2. In-capsule data is not supported in response capsules.

7.4.4 Connection Establishment

Figure 63 illustrates the process used to establish NVMe/TCP connection. The first step is to establish a TCP connection between a host and a controller. A controller acts as the passive side of the TCP connection and is set to “listen” for host-initiated TCP connection establishment requests.

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

Figure 63: NVMe/TCP Queue Establishment Sequence



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

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

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

7.4.5 Data Transfers

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

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

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

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

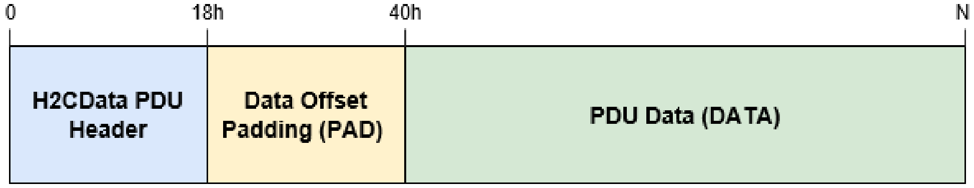
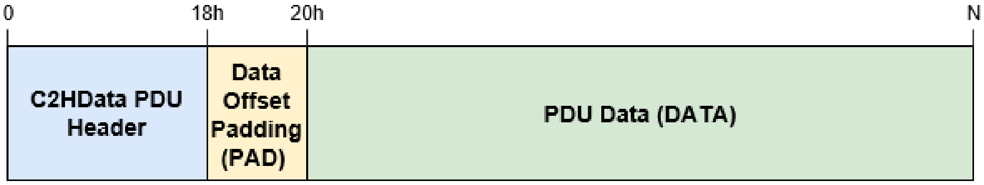


Figure 65 shows an example of a C2HData PDU where HPDA (refer to section 7.4.10.2) was set to 03h by the host in ICReq when the connection was established. The C2HData PDU header size 24 bytes and header digest is disabled. An alignment of 16 bytes is required, thus the controller inserts 8 bytes of padding in the PAD field.

Figure 65: Example of 64B PDU DATA Alignment in H2CData PDU



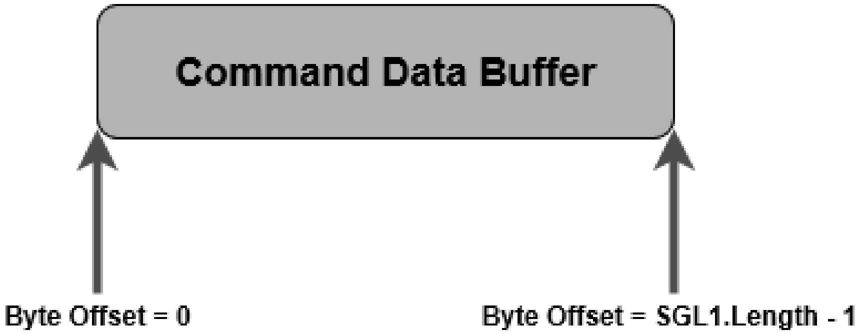
7.4.5.1 Command Data Buffers and SGLs

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

The NVMe/TCP transport supports exactly one SGL descriptor per command. NVMe/TCP supports two SGL Descriptor types. An SGL Data Block descriptor with an SGL Descriptor Sub Type value of 1h specifies the use of in-capsule data. A Transport SGL Data block descriptor with an SGL Descriptor Sub Type value of Ah is referred to as Command Data Buffer Descriptor and specifies the use of the NVMe/TCP transport specific data transfer mechanism. The length field in the descriptor specifies the size of the Command Data Buffer associated with the command (refer to Figure 66).

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 66: Command Data Buffer Transport SGL Data Block Descriptor



7.4.5.2 Controller to Host Command Data Buffer Transfers

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

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

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

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

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

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

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

Examples of an unexpected C2HData PDUs are:

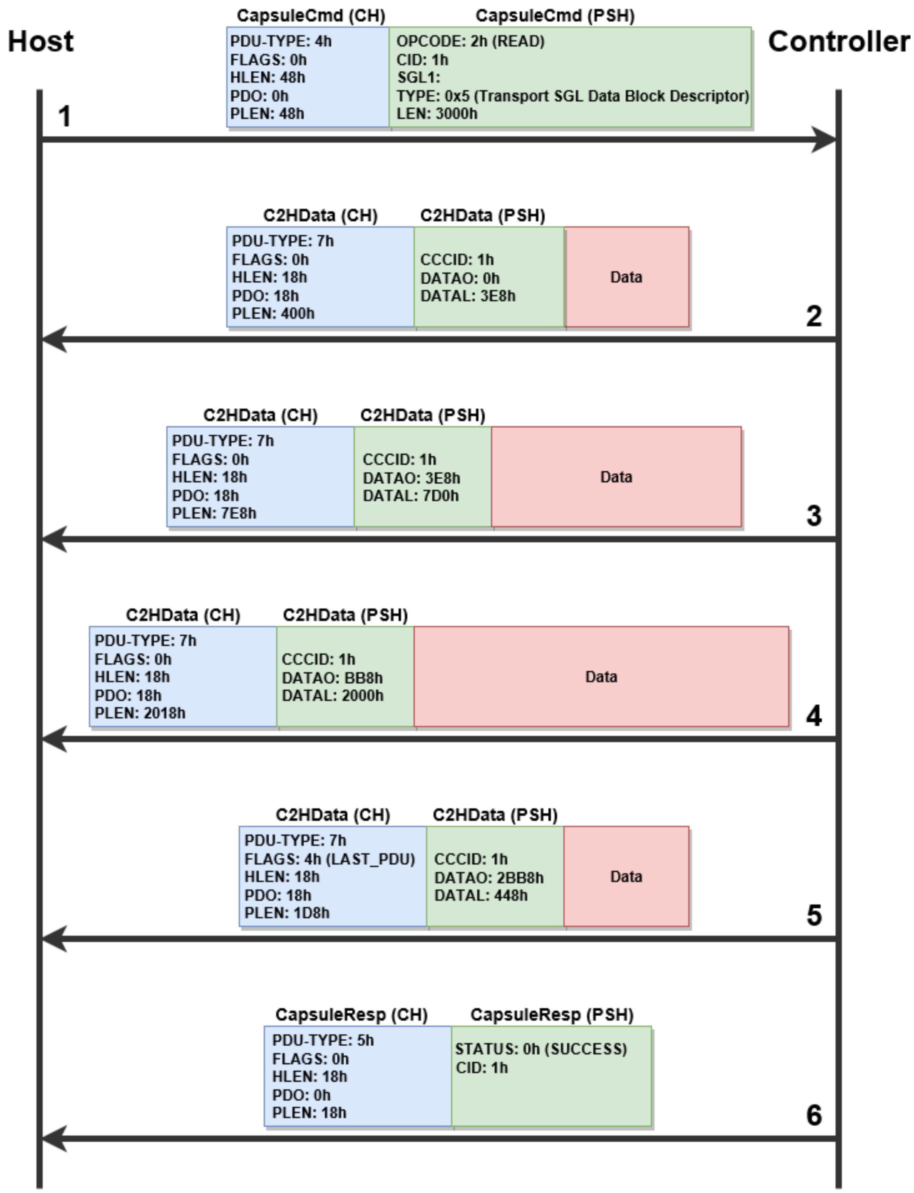
- Reception of a C2HData PDU after reception of a C2HData PDU with the LAST_PDU flag set to ‘1’ and is associated with the same command; and
- Reception of a C2HData PDU associated with a command that does not involve controller to host data transfer.

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

Figure 67 illustrates a command that performs a 12,288 bytes (3000h bytes) controller to host command data buffer transfer:

1. The host sends a CapsuleCmd PDU to the controller containing an SQE with a Transport SGL Data Block descriptor in SGL1. The Length field in the descriptor has a value of 3000h;
2. The controller processes the command and sends an C2HData PDU that transfers 1,000 bytes (3E8h bytes). The Data Offset (DATAO) field is cleared to 0h, the Data Length (DATAL) field is set to 3E8h, and the Last (LAST_PDU) flag is cleared to ‘0’ since this is not the last PDU of the data transfer;
3. The controller sends a subsequent C2HData PDU that transfers 2,000 bytes (7D0h bytes). The DATAO field is set to 3E8h, the DATAL field is set to 7D0h, and the LAST_PDU flag is cleared to ‘0’ since this is not the last PDU of the data transfer;
4. The controller sends a subsequent C2HData PDU that transfers 8,192 bytes (2000h bytes). The DATAO field is set to BB8h, the DATAL field is set to 2000h, and the LAST_PDU flag is cleared to ‘0’ since this is not the last PDU of the data transfer;
5. The controller sends a subsequent C2HData PDU that transfers 1,096 bytes (448h bytes). The DATAO field is set to 2BB8h, the DATAL field is set to 448h, and the LAST_PDU flag is set to ‘1’ since this is the last PDU of the data transfer; and
6. The controller sends a CapsuleResp PDU to the host containing a CQE.

Figure 67: Controller to Host Data Transfer Example



7.4.5.3 Host to Controller Command Data Buffer Transfers

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

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

The controller governs the rate of the data transfer from the host to the controller using Ready to Transfer (R2T) PDUs. When a connection is established, the host 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 with which it is associated, a transfer tag (TTAG), a data offset (R2TO) from the beginning of the command data buffer, and the transfer data length (R2TL). When an R2T PDU is received by a host, then the host transfers the data requested by the controller in the R2T PDU. This data transfer may be performed using one or more H2CData PDUs. H2CData PDUs must start at the offset specified in the R2T PDU (R2TO) and transfer data contiguously until the data transfer length specified by the R2T PDU (R2TL) is reached (i.e., the DATAO field in the first H2CData PDU is equal to R2TO that specified in the R2T PDU and DATAO field in subsequent PDUs is equal the DATAO field plus DATAL field of the previous H2CData PDU). The H2CData PDU length does not exceed the maximum length communicated by the controller during the connection establishment (MAXH2CDATA).

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

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

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

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

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

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

Examples of an unexpected H2CData PDUs are:

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

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

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

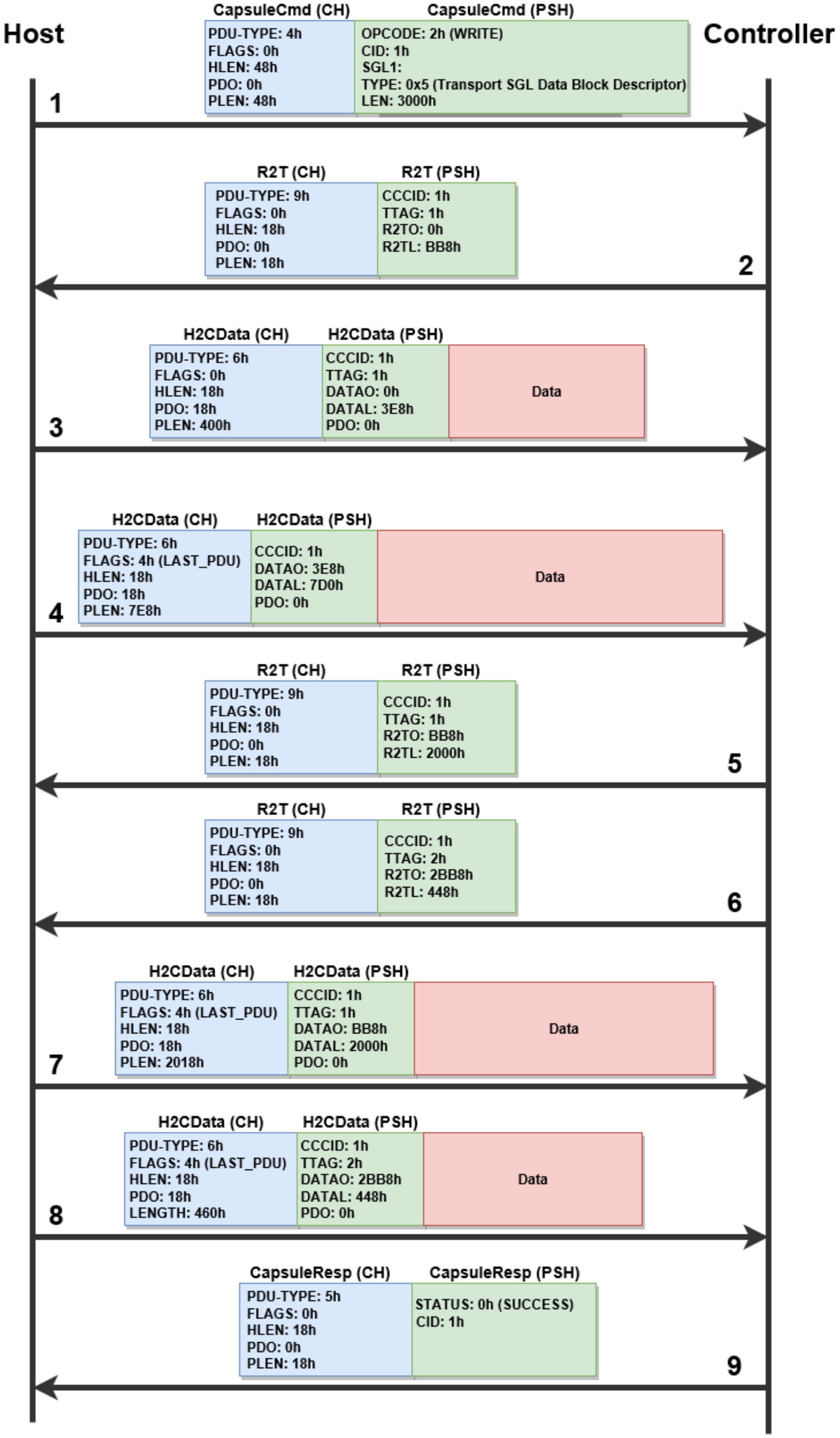
Figure 68 illustrates a command that performs a 12,288 bytes (3000h bytes) host to controller command data buffer transfer using R2T PDUs:

1. The host sends a Command PDU to the controller containing an SQE with a Transport SGL Data Block descriptor with subtype value of Ah in SGL1. The Length field in the descriptor has a value of 3000h;
2. The controller processes the command and sends an R2T PDU to the host that requests 3,000 byte (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 R2T Data Offset (R2TO) field is cleared to 0h, the R2T Data Length (R2TL) field

is set to 3E8h, and the Last (LAST_PDU) flag is cleared to ‘0’ since this is not the last PDU of the data transfer;

4. The host sends a subsequent H2CData PDU that transfers 2,000 bytes (7D0h bytes). The DATAO field is set to 3E8h, the DATAL field is set to 7D0h, and the LAST_PDU flag is set to ‘1’ since this is the last PDU of the data transfer requested by the R2T PDU;
5. The controller sends a subsequent R2T PDU that transfers 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 transfers 1,096 bytes (448h bytes) if the host supports MAXR2T greater than one. The R2TO field is set to 2BB8h and the R2TL field is 448h;
7. The host sends a H2CData PDU that transfers 8,192 bytes (2000h bytes). The DATAO field is set to BB8h, the DATAL field is set to 2000h, and the LAST_PDU flag is set to ‘1’ since this is the last PDU of the data transfer;
8. The host send a subsequent H2CData PDU that transfers 1,096 bytes (448h bytes). The DATAO field is set to 2BB8h, the DATAL field is set to 448h, and the LAST_PDU flag is set to ‘1’ since this is the last PDU of the data transfer; and
9. The controller sends a Response PDU to the host containing a CQE.

Figure 68: Host to Controller Data Transfer Example



7.4.6 PDU Header and Data Digests

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

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

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

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

If a host did not request the use of header or data digest in the ICReq PDU but the use of the digest was enabled by the controller in the ICResp PDU, then the host shall treat that ICResp PDU as a fatal transport error (refer to section 7.4.7) with the Fatal Error Status field set to “Invalid PDU header field” and the Additional Error Information field containing the DIGEST field byte offset.

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

Figure 69: Host to Controller NVMe/TCP PDU Digests

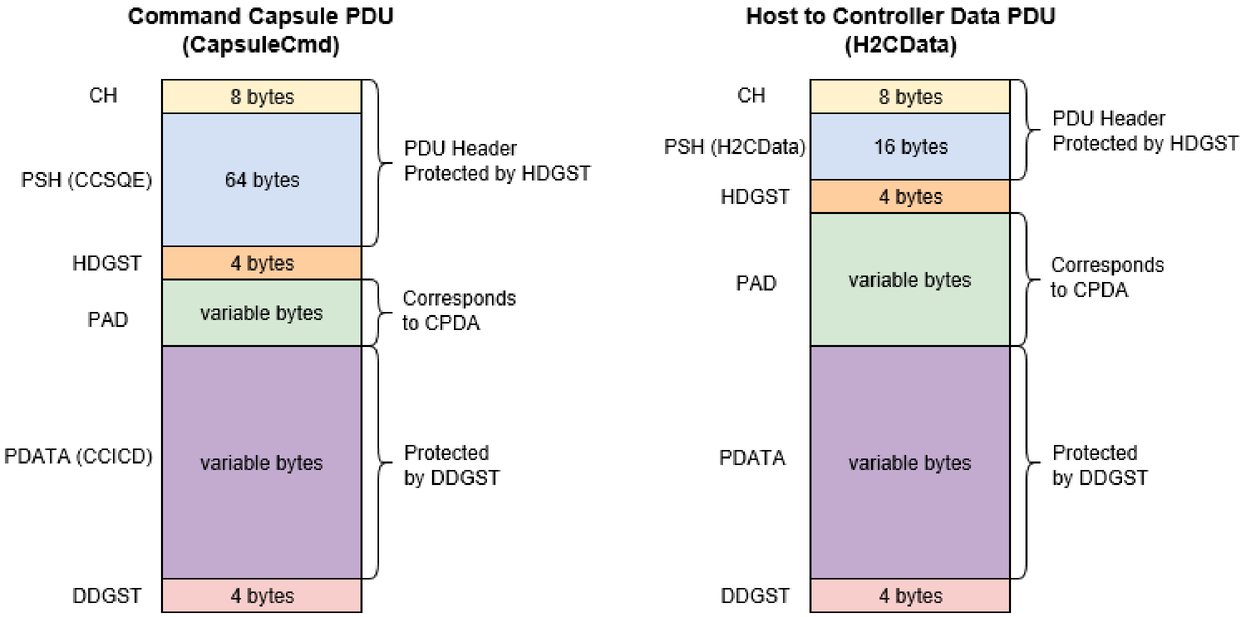
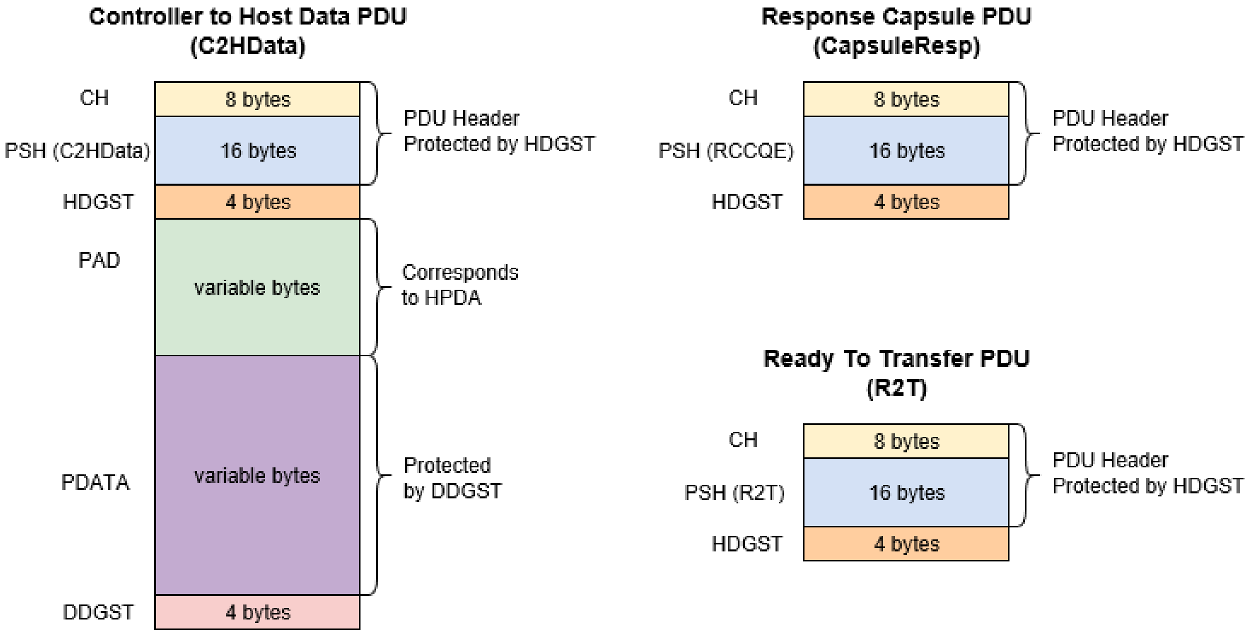


Figure 70: Controller to Host NVMe/TCP PDU Digests



7.4.6.1 Digest Error handling

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

A PDU data digest error impacts the integrity of PDU data but does not impact the TCP byte stream synchronization. A PDU data digest error detected by the host or the controller is treated as a non-fatal transport error (refer to section 7.4.7). When a host detects a data digest error in a C2HData PDU, 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.

7.4.7 Transport Error Handling

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

When a controller detects a fatal error, that controller shall:

1. stop processing any PDUs that arrive on the connection; and

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

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

When a host detects a fatal error, that host shall:

1. stop processing any PDUs that arrive on the connection; and
2. send a H2CTermReq PDU with:
    a. an appropriate Fatal Error Status field (FES);
    b. additional error information in the Fatal Error Information (FEI) field if applicable; and
    c. set the H2CTermReq PDU data with the PDU header that was being processed when the fatal error was detected.

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

Errors that effect the transport but from which the transport is able to recover normal operation are non-fatal errors. A non-fatal error may affect one or more commands. These commands are likely to complete successfully if retried. When a non-fatal transport error is detected, affected commands are completed with Transient Transport Error status code (refer to Figure 126 in the NVMe Base specification).

7.4.8 Keep Alive

The NVMe/TCP Transport requires the use of the Keep Alive feature (refer to section 7.12 in the NVMe base specification). The NVMe/TCP Transport does not impose any limitations on the minimum and maximum Keep Alive Timeout value. The minimum should be set large enough to account for any transient fabric interconnect failures between the host and controller.

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

7.4.9 Transport Specific Address Subtype and Transport Service Identifier

The Discovery Log Entry includes a Transport Specific Address Subtype (TSAS) field that is defined in Figure 71 for the NVMe/TCP Transport. The TSAS field describes TCP connection properties, such as whether TLS is supported.

Figure 71: Transport Specific Address Subtype Definition for NVMe/TCP Transport

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

TLS implementation is optional for NVMe/TCP.

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

7.4.9.1 Mandatory and Recommended Cipher Suites

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

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

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

10. shall be at least 2,048 bits in size (refer to section 4.3 of RFC 7525); and
11. should not be reused (i.e., used for more than one DH exchange) (refer to section 6.4 of RFC 7525).

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

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

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

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

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

7.4.9.2 TLS Implementations and Use Requirements

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

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

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

7.4.9.3 Transport Service Identifier

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

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

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

7.4.10 NVMe/TCP PDUs

This section describes the format of NVME/TCP PDUs.

7.4.10.1 PDU Common Header (CH)

Figure 72: PDU Common Header (CH)

Bytes	Description
00	**PDU-Type:** Specifies the type of PDU. This value is also referred to as the PDU opcode. Refer to Figure 61 for PDU opcodes.
01	**FLAGS:** PDU-TYPE specific flags
02	**Header Length (HLEN):** Length of PDU header (not including the Header Digest) in bytes.
03	**PDU Data Offset (PDO):** PDU Data Offset from the start of the PDU in bytes.
07:04	**PDU Length (PLEN):** Total length of PDU (includes CH, PSH, HDGST, PAD, DATA, and DDGST) in bytes.

7.4.10.2 Initialize Connection Request PDU (ICReq)

Figure 73: 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><b>PDU-Type:</b> 00h</td></tr>
<tr><td>01</td><td><b>FLAGS:</b> Reserved</td></tr>
<tr><td>02</td><td><b>HLEN:</b> Fixed length of 128 bytes (80h).</td></tr>
<tr><td>03</td><td><b>PDO:</b> Reserved</td></tr>
<tr><td>07:04</td><td><b>PLEN:</b> Fixed length of 128 bytes (80h).</td></tr>
<tr><td>09:08</td><td rowspan="5">PSH</td><td><b>PDU Format Version (PFV):</b> 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><b>Host PDU Data Alignment (HPDA):</b> Specifies the data alignment for all PDUs transferred from the controller to the host that contain data. This value is 0’s based value in units of dwords and must be a value in the range 0 to 31 (e.g., values 0, 1, and 2 correspond to 4 byte, 8 byte, and 12 byte alignment).</td></tr>
<tr><td>11</td><td><b>DGST:</b> Host PDU header and Data digest enable options.
<table>
<tr><th>Bits</th><th>Definition</th></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
<tr><td>1</td><td><b>DDGST_ENABLE:</b> If set to ‘1’, the use of data digest is requested by the host for the connection. If cleared to ‘0’, data digest shall not be used for the connection.</td></tr>
<tr><td>0</td><td><b>HDGST_ENABLE:</b> If set to ‘1’, the use of header digest is requested by the host for the connection. If cleared to ‘0’, header digest shall not be used for the connection.</td></tr>
</table></td></tr>
<tr><td>15:12</td><td><b>Maximum Number of Outstanding R2T (MAXR2T):</b> Specifies the maximum number of outstanding R2T PDUs for a command at any point in time on the connection. This is a 0’s based value.</td></tr>
<tr><td>127:16</td><td>Reserved</td></tr>
</table>

7.4.10.3 Initialize Connection Response PDU (ICResp)

Figure 74: 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><b>PDU-Type:</b> 01h</td></tr>
<tr><td>01</td><td><b>FLAGS:</b> Reserved</td></tr>
<tr><td>02</td><td><b>HLEN:</b> Fixed length of 128 bytes (80h).</td></tr>
<tr><td>03</td><td><b>PDO:</b> Reserved</td></tr>
<tr><td>07:04</td><td><b>PLEN:</b> Fixed length of 128 bytes (80h).</td></tr>
<tr><td>09:08</td><td rowspan="5">PSH</td><td><b>PDU Format Version (PFV):</b> 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><b>Controller PDU Data Alignment (CPDA):</b> Specifies the data alignment for all PDUs transferred from the host to the controller that contain data. This value is 0's based value in units of dwords in the range 0 to 31 (e.g., values 0, 1, and 2 correspond to 4 byte, 8 byte, and 12 byte alignment).</td></tr>
<tr><td>11</td><td><b>DGST:</b> PDU and Data Digest enable options.
<table>
<tr><th>Bits</th><th>Definition</th></tr>
<tr><td>7:2</td><td>Reserved</td></tr>
<tr><td>1</td><td><b>DDGST_ENABLE:</b> If set to '1', data digest is used for the connection. If cleared to '0', data digest is not used for the connection.</td></tr>
<tr><td>0</td><td><b>HDGST_ENABLE:</b> If set to '1', header digest is used for the connection. If cleared to '0', header digest is not used for the connection.</td></tr>
</table></td></tr>
<tr><td>15:12</td><td><b>Maximum Host to Controller Data length (MAXH2CDATA):</b> Specifies the maximum number of PDU-Data bytes per H2C Data Transfer PDU in bytes. This value is a multiple of dwords and should be no less than 4,096.</td></tr>
<tr><td>127:16</td><td>Reserved</td></tr>
</table>

7.4.10.4 Host to Controller Terminate Connection Request PDU (H2CTermReq)

Figure 75: 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><b>PDU-Type:</b> 02h</td></tr>
<tr><td>01</td><td><b>FLAGS:</b> Reserved</td></tr>
<tr><td>02</td><td><b>HLEN:</b> Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td><b>PDO:</b> Reserved</td></tr>
<tr><td>07:04</td><td><b>PLEN:</b> 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>PSH</td><td><b>Fatal Error Status (FES):</b> Indicates the fatal error information.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00h</td><td>Reserved</td></tr>
<tr><td>01h</td><td><b>Invalid PDU Header Field:</b> An Invalid Field in the Transport Header was detected by the host.</td></tr>
<tr><td>02h</td><td><b>PDU Sequence Error:</b> An unexpected Protocol sequence was detected by the host.</td></tr>
<tr><td>03h</td><td><b>Header Digest Error:</b> A Header Digest (HDGST) error was detected by the host.</td></tr>
<tr><td>04h</td><td><b>Data Transfer Out of Range:</b> A C2HData with data offset or data offset plus data length is out of its associated command data buffer range.</td></tr>
<tr><td>05h</td><td><b>R2T Limit Exceeded:</b> An R2T PDU that exceeds MAXR2T was received by the host.</td></tr>
<tr><td>06h</td><td><b>Unsupported Parameter:</b> An unsupported parameter was received by the host.</td></tr>
<tr><td>07h to FFFFh</td><td>Reserved</td></tr>
</table></td></tr>
</table>

Figure 75: Host to Controller Terminate Connection Request PDU (H2CTermReq)

<table>
<tr><th>Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>13:10</td><td></td><td><b>Fatal Error Information (FEI):</b> 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><b>PDU Header Field Offset:</b> This field indicates the offset in bytes from the start of the PDU Header to the start of the field that is in error. If multiple errors exist, then this field indicates the lowest offset that is in error.</td></tr>
<tr><td>02h</td><td>Reserved</td></tr>
<tr><td>03h</td><td><b>PDU Header Digest:</b> This field indicates the header digest that was received by the host and caused a header digest verification error.</td></tr>
<tr><td>04h to 05h</td><td>Reserved</td></tr>
<tr><td>06h</td><td><b>Unsupported Parameter Field Offset:</b> This field indicates the offset in bytes from the start of the PDU Header to the start of the field that is in error. If multiple errors exist, then this field indicates the lowest offset that is in error.</td></tr>
<tr><td>07h to FFFFh</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>23:14</td><td></td><td>Reserved</td></tr>
<tr><td>N - 1:24</td><td>DATA</td><td><b>Data:</b> This field contains the PDU header that was being processed by the host while the fatal error was detected.</td></tr>
</table>

7.4.10.5 Controller to Host Terminate Connection Request PDU (C2HTermReq)

Figure 76: 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>CH</td><td><b>PDU-Type:</b> 03h</td></tr>
<tr><td>01</td><td></td><td><b>FLAGS:</b> Reserved</td></tr>
<tr><td>02</td><td></td><td><b>HLEN:</b> Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td></td><td><b>PDO:</b> Reserved</td></tr>
<tr><td>07:04</td><td></td><td><b>PLEN:</b> 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>PSH</td><td><b>Fatal Error Status (FES):</b> Indicates the fatal error information.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>00h</td><td>Reserved</td></tr>
<tr><td>01h</td><td><b>Invalid PDU Header Field:</b> An Invalid Field in the Transport Header was detected by the controller.</td></tr>
<tr><td>02h</td><td><b>PDU Sequence Error:</b> An unexpected Protocol sequence was detected by the controller.</td></tr>
<tr><td>03h</td><td><b>Header Digest Error:</b> A Header Digest (HDGST) error was detected by the controller.</td></tr>
<tr><td>04h</td><td><b>Data Transfer Out of Range:</b> A H2CData with data offset or data offset plus data length is out of its associated R2T range.</td></tr>
<tr><td>05h</td><td><b>Data Transfer Limit Exceeded:</b> A H2CData PDU with data length that exceeds MAXH2CDATA was received by the controller.</td></tr>
<tr><td>06h</td><td><b>Unsupported Parameter:</b> An unsupported parameter was received by the controller.</td></tr>
<tr><td>07h to FFFFh</td><td>Reserved</td></tr>
</table></td></tr>
</table>

Figure 76: Controller to Host Terminate Connection Request PDU (C2HTermReq)

<table>
<tr><th>Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>13:10</td><td rowspan="2"></td><td><b>Fatal Error Information (FEI):</b> 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><b>PDU Header Field Offset:</b> This field indicates the offset in bytes from the start of the Transport Header to the start of the field that is in error. If multiple errors exist, then this field indicates the lowest offset that is in error.</td></tr>
<tr><td>02h</td><td>Reserved</td></tr>
<tr><td>03h</td><td><b>PDU Header Digest:</b> This field indicates the header digest that was received by the controller and caused a header digest verification error.</td></tr>
<tr><td>04h to 05h</td><td>Reserved</td></tr>
<tr><td>06h</td><td><b>Unsupported Parameter Field Offset:</b> This field indicates the offset in bytes from the start of the PDU Header to the start of the field that is in error. If multiple errors exist, then this field indicates the lowest offset that is in error.</td></tr>
<tr><td>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><b>Data:</b> This field contains the PDU header that caused the error.</td></tr>
</table>

7.4.10.6 Command Capsule PDU (CapsuleCmd)

Figure 77: Command Capsule PDU (CapsuleCmd)

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

7.4.10.7 Response Capsule PDU (CapsuleResp)

Figure 78: Response Capsule PDU (CapsuleResp)

<table>
<tr><th>Bytes</th><th>PDU Section</th><th>Description</th></tr>
<tr><td>00</td><td rowspan="5">CH</td><td><b>PDU-Type:</b> 05h</td></tr>
<tr><td>01</td><td><b>FLAGS:</b>
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>7:1</td><td>Reserved</td></tr>
<tr><td>0</td><td><b>HDGSTF:</b> If set to ‘1’, then a valid Header digest value follows the PDU header.</td></tr>
</table></td></tr>
<tr><td>02</td><td><b>HLEN:</b> Fixed length of 24 bytes (18h).</td></tr>
<tr><td>03</td><td><b>PDO:</b> Reserved</td></tr>
<tr><td>07:04</td><td><b>PLEN:</b> Length of PDU Header and HDGST (if present) in bytes.</td></tr>
<tr><td>23:08</td><td>PSH</td><td><b>NVMe-oF Response Capsule CQE (RCCQE):</b> Response Capsule CQE.</td></tr>
<tr><td>27:24</td><td>HDGST</td><td><b>HDGST:</b> If HDGSTF is set in the FLAGS field, this field is present and contains the Header digest.</td></tr>
</table>

7.4.10.8 Host To Controller Data Transfer PDU (H2CData)

Figure 79: Host To Controller Data Transfer PDU (H2CData)

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

7.4.10.9 Controller To Host Data Transfer PDU (C2HData)

Figure 80: Controller To Host Data Transfer PDU (C2HData)

Bytes	PDU Section	Description
00	CH	**PDU-Type:** 07h
01		**FLAGS:** **Bits** – **Description** 7:4 – Reserved 3 – **SUCCESS:** If set to ‘1’, indicates that the command referenced by CCCID was completed successfully with no other information and that no response PDU is sent by the Controller. 2 – **LAST_PDU:** If set to ‘1’, indicates the PDU is the last in the Command Data transfer series. 1 – **DDGSTF:** If set to ‘1’, then a valid Data digest value follows the PDU Da 0 – **HDGSTF:** If set to ‘1’, then a valid Header digest value follows the PDU header.
02		**HLEN:** Fixed length of 24 bytes (18h).
03		**PDO:** Data Offset within PDU. This value complies to the HPDA field set by the controller in ICReq PDU (refer to section 7.4.9.2).
07:04		**PLEN:** Total length of PDU (including PDU header, HDGST, PAD, DATA, and DDGST) in bytes.
09:08	PSH	**Command Capsule CID (CCCID):** This field contains the SQE.CID value of the Command Capsule associated with the host resident data.
11:10		Reserved
15:12		**Data Offset (DATAO):** Byte offset from start of host resident data. This value shall be dword aligned.
19:16		**Data Length (DATAL):** PDU DATA field length in bytes. This value shall be dword aligned.
23:20		Reserved
27:24	HDGST	**HDGST:** If HDGSTF is set in the FLAGS field, this field is present and contains the Header digest.
N - 1:N	PAD	**PAD:** If HPDA is set to a non-zero value, then this field is padded as specified by HPDA.
M - 1:N	DATA	**PDU-Data**
M + 3:M	DDGST	**Data Digest (DDGST):** Data Digest of PDU-Data field.

7.4.10.10 Ready to Transfer PDU (R2T)

Figure 81: Ready to Transfer PDU (R2T)

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