NVM Express™ over Fabrics
Revision 1.0a
July 17, 2018

Please send comments to info@nvmexpress.org

NVM Express™ over Fabrics revision 1.0 is available for download at http://nvmexpress.org. NVM Express™ revision 1.0 was ratified on May 31, 2016. NVM Express over Fabrics revision 1.0a incorporates ECN 001 to ECN 004. Applied the NVM Express trademark and logo usage guidelines.

SPECIFICATION DISCLAIMER

LEGAL NOTICE:

This NVM Express over Fabrics revision 1.0a 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 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 2018 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
info@nvmexpress.org

Table of Contents

1 Introduction

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

The mapping of extensions defined in this document to a specific NVMe Transport are defined in an NVMe Transport binding specification. This document contains an NVMe Transport binding specification for RDMA. 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.

NVMe over Fabrics revision 1.0 is intentionally limited in scope to define essential functionality. Restrictions that may be removed in future revisions include:

- There is a 1:1 mapping of a single Submission Queue to a single Completion Queue. NVMe over Fabrics does not support the mapping of Multiple Submission Queues to a single Completion Queue;
- There is no flow control for completions at the NVM Express layer. Therefore, Completion Queues should be sized to support the maximum number of outstanding commands; and
- Metadata (if supported) is transferred as a contiguous part of logical blocks. Specifically, transferring metadata as a separate buffer of data is not supported.

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. The host increments the Tail entry pointer associated with that Submission Queue as part of submitting a command.

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 Fabrics command;
- If metadata is supported, it 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; and
- 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.

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;

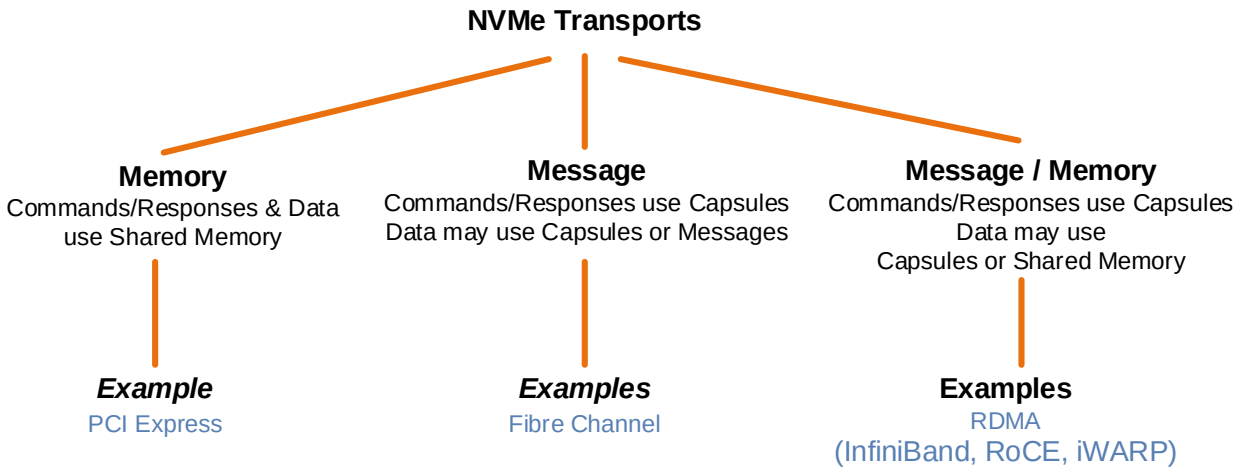
- Support for Enterprise capabilities such as end-to-end data protection;
- Robust error reporting and management capabilities;
- Priority associated with each I/O queue with a well-defined controller queue arbitration mechanism;
- Efficient and streamlined command set;
- Support for multiple namespaces 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.

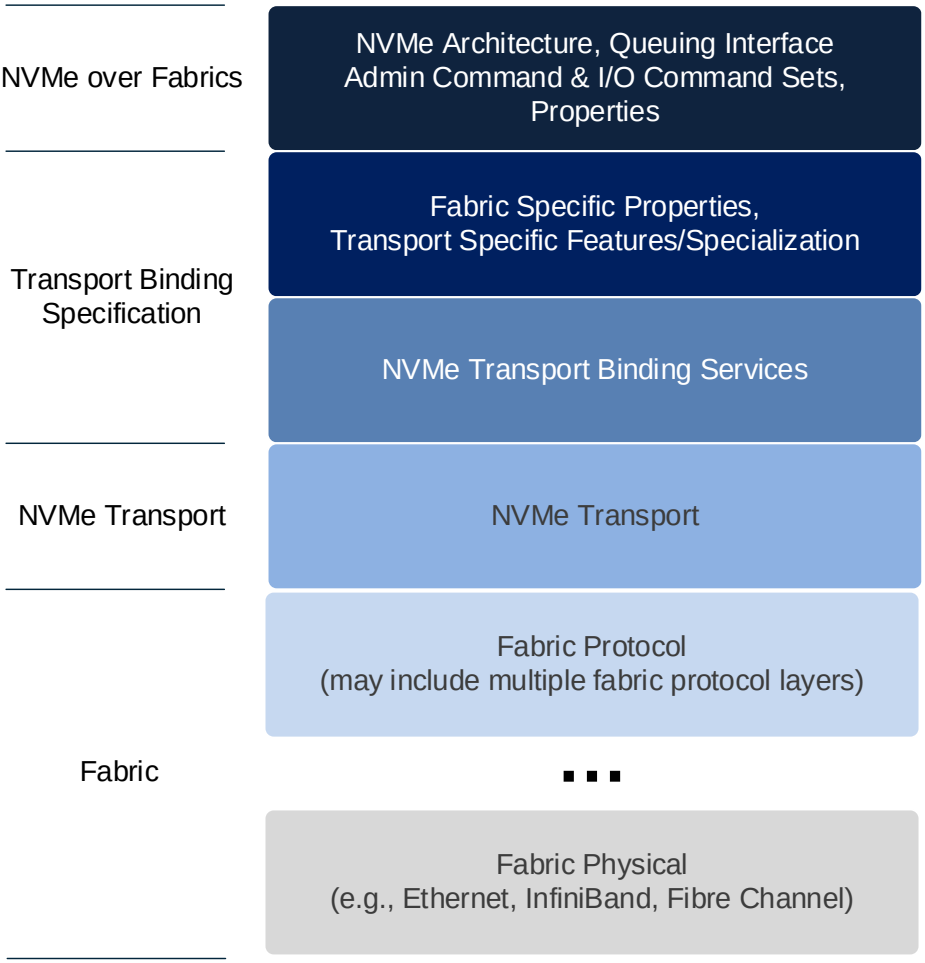
The only memory model NVMe Transport supported by NVMe is PCI Express, as defined in the NVMe Base specification. Message model and message/memory model NVMe Transports are specified in this document.

Figure 1: Taxonomy of Transports



NVMe 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



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 NQN 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 it 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 indicate that it may accept 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.

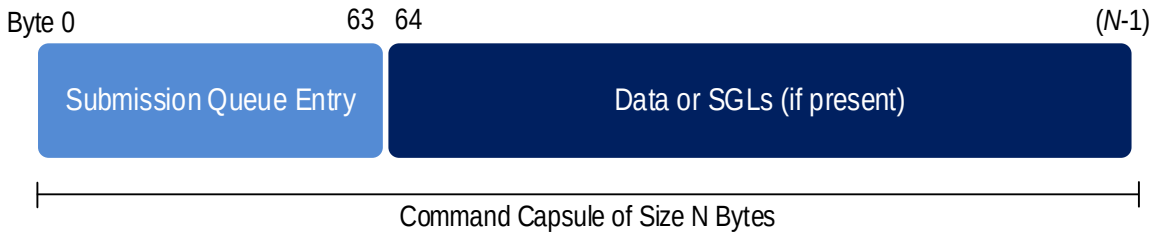
An association exists until the controller is shutdown, a Controller Level Reset, or the NVMe Transport connection is lost between the host and controller for the Admin or any I/O Queue. There is no explicit NVMe command that breaks the NVMe Transport connection between a host and controller. While a controller is associated to a host the controller is busy and no other associations may be made to 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) and may optionally include SGLs or data. A response capsule contains a response (formatted as a Completion Queue Entry) 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.

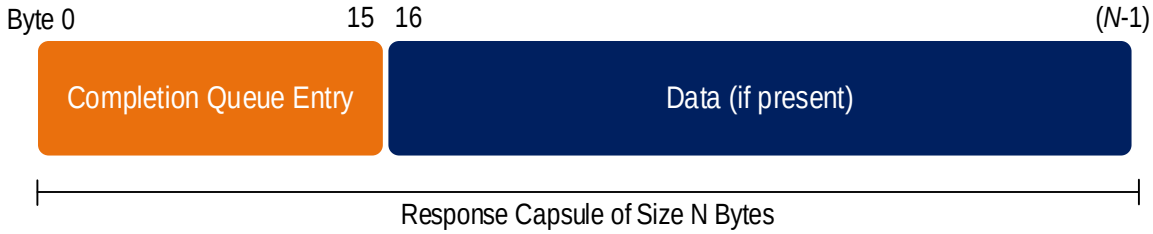
Command capsules are transferred from a host to an NVM subsystem. The SQE contains an Admin command, an I/O command, or a Fabrics command. The minimum size of a command capsule is NVMe Transport binding specific, but shall be at least 64B in size. The maximum size of a command capsule is NVMe Transport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



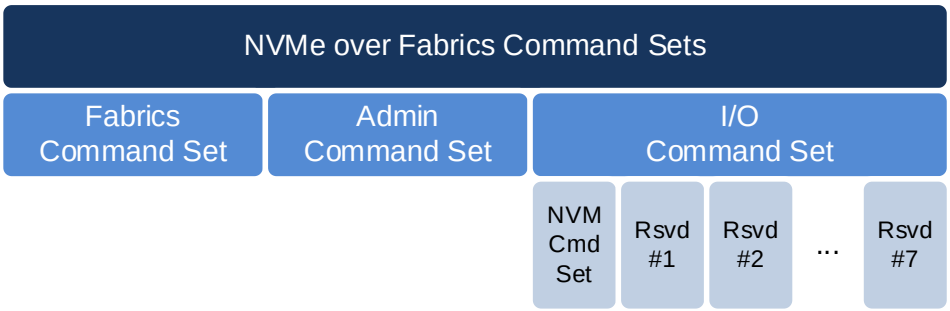
Message and Message/Memory model NVMe Transports require all SGLs sent from the host to the controller be transferred within the command. Message and Message/Memory model NVMe Transports may optionally support the transfer of a portion or all data within the command and response capsules.

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 3.5.1).

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 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. 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 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 it 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 (Fabrics, Admin, or I/O) 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 other than authentication commands until NVMe in-band authentication has completed. Refer to section 6.

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.

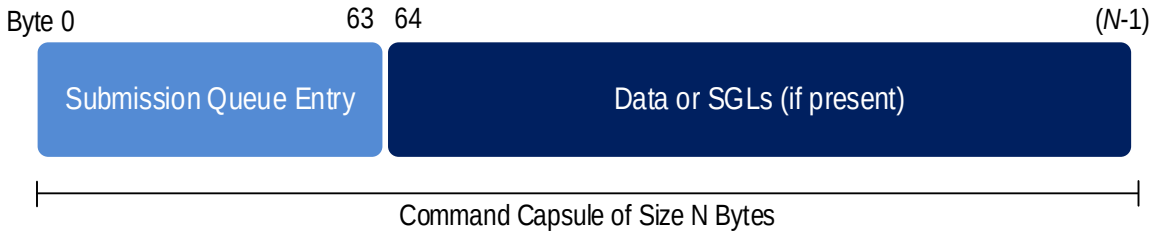
The controller shall support SGL based data transfers for commands on both the Admin Queue and I/O Queues. Data may be transferred within the capsule or through memory transactions based on the underlying NVMe Transport as indicated in the SGL descriptors associated with the command capsule. The SGL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

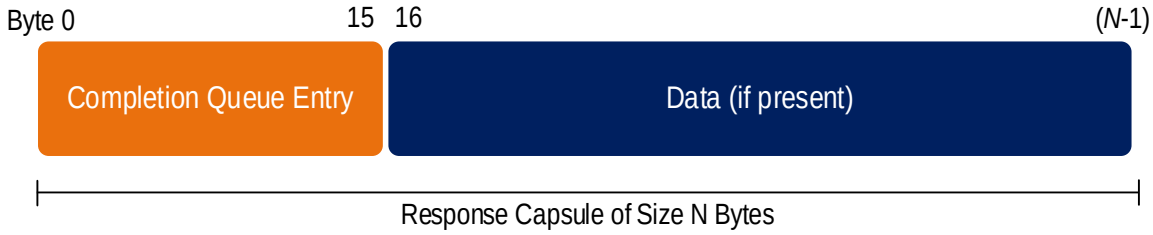
Byte	Description
00	**Opcode (OPC):** Set to 7Fh to indicate a Fabrics command.
01	Reserved
03:02	**Command Identifier (CID):** This field specifies a unique identifier for the command. The identifier shall be unique among all outstanding commands associated with a particular queue.
04	**Fabrics Command Type (FCTYPE):** This field specifies the Fabrics command transferred in the capsule. The Fabrics command types are defined in Figure 14.
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.

Figure 9: Fabrics Response Capsule – Completion Queue Entry Format

Byte	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.
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 associated Fabrics command. **Bit** / **Definition** 15:01 – Status Field as defined in section 4.6.1 of the NVMe Base specification. 00 – Reserved

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 31 of the NVMe Base specification). Refer to Figure 10.

Figure 10: Fabrics Command Specific Status Values

Value	Description	Commands Affected
80h	**Connect Incompatible Format:** The NVM subsystem does not support the Connect Record Format specified by the host.	Connect
81h	**Connect Controller Busy:** The controller is already associated with a host. This value is also returned if there is no available controller.	Connect
82h	**Connect Invalid Parameters:** One or more of the parameters (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 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 BFh	Reserved	
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. If metadata is transferred, it is a contiguous part of the logical block with which it 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.

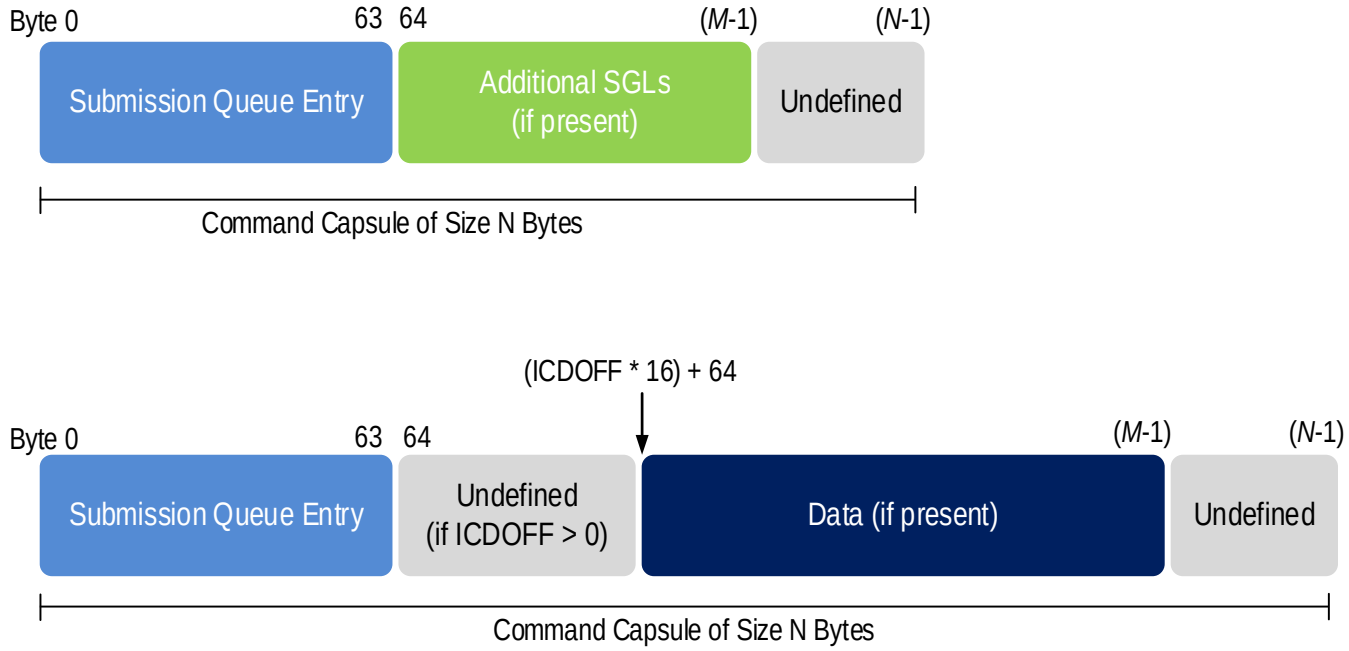
An NVMe Transport binding specification defines the support for data as part of the command capsule. The controller indicates the starting location of data within a command capsule via the In Capsule Data Offset (ICDOFF) field in the Identify Controller data structure.

There are restrictions for SGLs that the host needs to 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

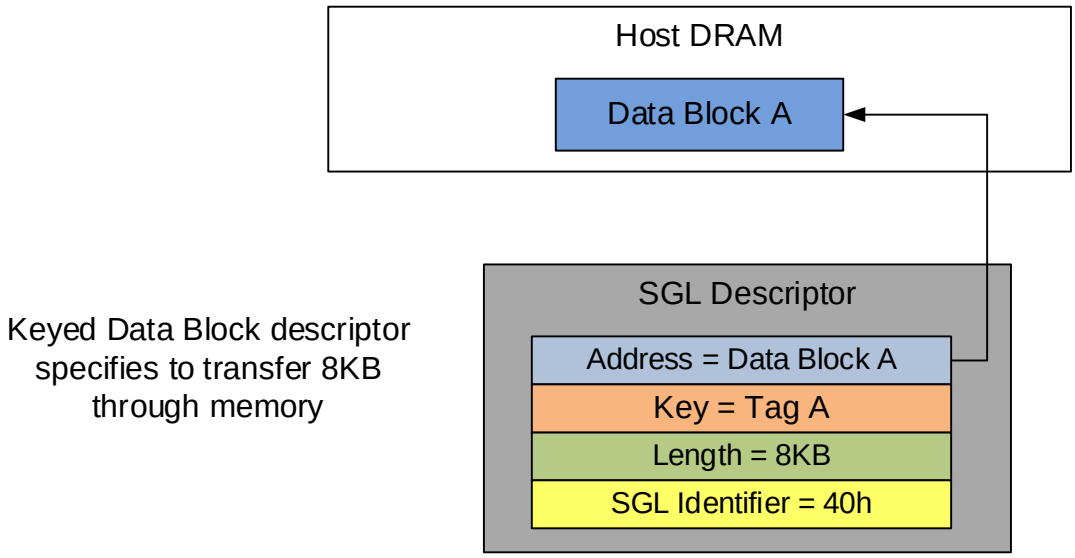


2.3.2 Data Transfer Examples

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.

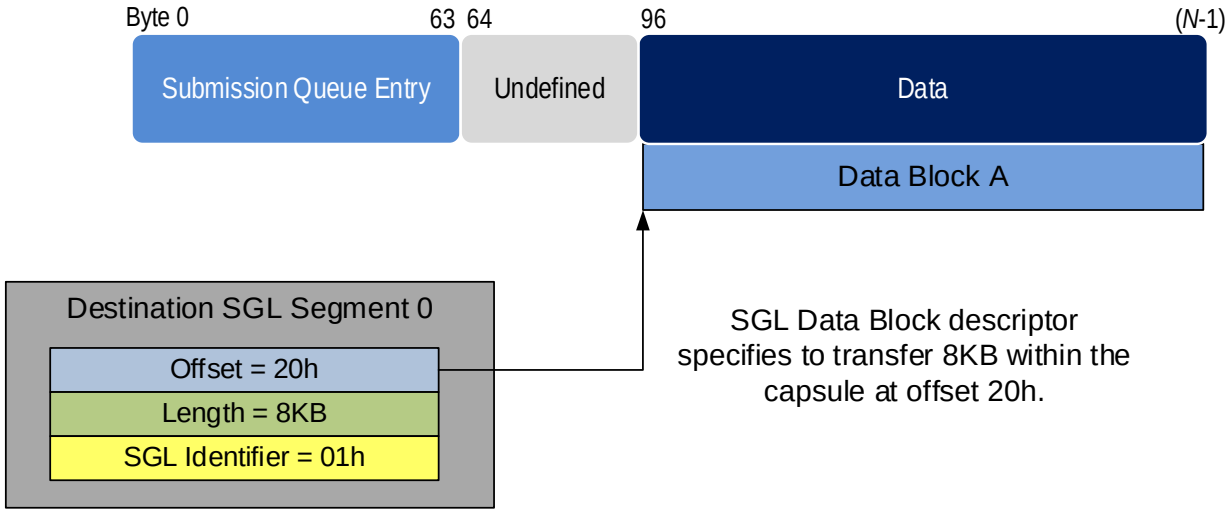
The first example shows an 8KB write where all of the data is transferred via memory transactions. In this case, there is one SGL descriptor that is contained within the Submission Queue Entry at CMD.SGL1. The SGL descriptor is a Keyed SGL Data Block descriptor. If more SGLs were needed to complete the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

Figure 12: SGL Example Using Memory Transactions



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



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, 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).

Each Submission Queue has a Head entry pointer and a Tail entry pointer that are used to manage the queue and determine the number of outstanding capsules. The Head and Tail entry pointers are initialized to zero 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 it adds a capsule to a queue. The controller increments the Head entry pointer when it removes a capsule from the queue.

The Submission Queue is empty when the Head entry pointer equals the Tail entry pointer. A capsule consumer may continue to remove capsules from the queue as long as the empty queue condition is not met.

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 add capsules to a Submission Queue as long as the queue is not full.

Completion Queues do not use Head entry pointers or Tail entry pointers (refer to section 2.4.2).

The definition for 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 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 and 2.4.2).

2.4.1 Submission Queue Flow Control

An NVMe over Fabrics Submission Queue Tail entry pointer is local to the host and is not communicated to the controller. The NVMe Transport is responsible for promptly delivering command capsules to the controller and notifying the controller of capsule arrival in a transport-specific fashion.

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).

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

If the controller detects that the host has submitted a command capsule to a full Submission Queue, then the controller shall stop processing commands and set the Controller Fatal Status (CSTS.CFS) bit to '1' (refer to section 9.5 in the NVMe Base specification).

2.4.2 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 it 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 queue. The host may use this value to size 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: Fabric 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
	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 may be submitted on the Admin Queue. The I/O Queue supports Fabrics commands as specified in this column.
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 it 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

Byte	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
39:24	**SGL Descriptor 1 (SGL1):** This field contains an SGL Data Block or 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

Figure 15 Authentication Receive Command – Submission Queue Entry

Byte	Description
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

Byte	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

Byte	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 an SGL Data Block or 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.

Figure 17: Authentication Send Command – Submission Queue Entry

Byte	Description
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

Byte	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. 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 or a status value of Invalid Field in Command is returned; and
- the NVM subsystem shall return any available controller to the host, indicating the Controller ID allocated 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 it is not allocated to that host, then a status value of Connect Invalid Host is returned.

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 I/O Queue before the controller has been enabled, then a status value of Connect Invalid

Parameters is returned. If the host sends a Connect command specifying a Queue ID for an Admin or I/O Queue which has already been created, then a status value of Command Sequence Error is returned.

If the Host Identifier, Host NQN, NVM Subsystem NQN, and 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, then a status value of Connect Invalid Parameters is returned. If the Host NQN or NVM Subsystem NQN values do not match the values that the NVM subsystem is configured to support, then a status value of Connect Invalid Parameters is returned. If there is a syntax error in the Host NQN or NVM Subsystem NQN value (refer to section 7.9 in the NVMe Base specification), then a status value of Connect Invalid Parameters is returned. If the Host Identifier is cleared to 0h, then a status value of Connect Invalid Parameters is returned.

Figure 19: Connect Command – Submission Queue Entry

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td>Opcode (OPC): Set to 7Fh to indicate a Fabrics command.</td></tr>
<tr><td>01</td><td>Reserved</td></tr>
<tr><td>03:02</td><td>Command Identifier (CID): This field specifies a unique identifier for the command. Refer to the definition in Figure 7.</td></tr>
<tr><td>04</td><td>Fabrics Command Type (FCTYPE): 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>SGL Descriptor 1 (SGL1): This field contains an SGL Data Block or 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>Record Format (RECFMT): Specifies the format of the Connect command capsule. If a new Connect command format is defined, this value is incremented by one. 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 Connect Incompatible Format shall be returned.</td></tr>
<tr><td>43:42</td><td>Queue ID (QID): Specifies the Queue Identifier for the Admin or I/O Queue. The identifier is used for both the Submission and Completion Queue. The Admin Submission and Completion Queue identifier is 0. I/O Submission and Completion Queue identifiers are in the range 1 to 65534.</td></tr>
<tr><td>45:44</td><td>Submission Queue Size (SQSIZE): 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>Connect Attributes (CATTR): This field indicates attributes for the connection.

Bits 7:2 are reserved.

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 4.11 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>
<tr><td>51:48</td><td>Keep Alive Timeout (KATO): This field has the same definition as the Keep Alive Timeout defined in section 5.15.1.14 of the the NVMe Base specification. The controller shall set the Keep Alive Timeout Feature to this value.</td></tr>
<tr><td>63:52</td><td>Reserved</td></tr>
</table>

Figure 20: Connect Command – Data

Byte	Description
15:00	**Host Identifier (HOSTID):** This field has the same definition as the Host Identifier defined in section 5.14.1.16 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 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 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

Byte	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):** 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 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>Byte</th><th>Description</th></tr>
<tr><td>01:00</td><td>Controller ID (CNTLID): 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>Authentication Requirements (AUTHREQ): 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>Bit</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>Byte</th><th>Description</th></tr>
<tr><td>01:00</td><td>Invalid Parameter Offset (IPO): 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>Invalid Attributes (IATTR): 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>Byte</th><th>Description</th></tr>
<tr><td>03:00</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>

3.4 Property Get Command and Response

The Property Get command is used to specify the property value to return to the host (refer to section 3.5.1). The fields for the Property Get command are defined in Figure 23. If an invalid property or invalid offset is specified, then a status value of Invalid Field in Command shall be returned.

Figure 23: Property Get Command

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td>Opcode (OPC): Set to 7Fh to indicate a Fabrics command.</td></tr>
<tr><td>01</td><td>Reserved</td></tr>
<tr><td>03:02</td><td>Command Identifier (CID): This field specifies a unique identifier for the command. Refer to the definition in Figure 7.</td></tr>
<tr><td>04</td><td>Fabrics Command Type (FCTYPE): 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>Attributes (ATTRIB): 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>
<tr><td>43:41</td><td>Reserved</td></tr>
<tr><td>47:44</td><td>Offset (OFST): Specifies the offset to the property to get. Refer to section 3.5.1.</td></tr>
<tr><td>63:48</td><td>Reserved</td></tr>
</table>

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 24.

Figure 24: Property Get Response

Byte	Description
07:00	**Value (VALUE):** Specifes 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.

3.5 Property Set Command and Response

The Property Set command is used to set the value of a property (refer to section 3.5.1). The fields for the Property Set 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 Set Command

Byte	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 00h to indicate a Property Set command.
39:05	Reserved

Figure 25: Property Set Command

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>40</td><td>Attributes (ATTRIB): 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>Offset (OFST): Specifies the offset to the property to set. Refer to section 3.5.1.</td></tr>
<tr><td>55:48</td><td>Value (VALUE): Specifes 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 26.

Figure 26: Property Set Response

Byte	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.

3.5.1 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 27 specifies the properties that are supported by NVMe over Fabrics.

Figure 27: 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 28: Identify Controller Attributes

Byte	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.
2047:1804		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. The host shall specify 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 34) 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, a controller shall abort all commands other than authentication commands with a status of Authentication Required. After the NVMe in-band authentication has been performed successfully, a controller shall abort all authentication commands with a status of Command Sequence Error.

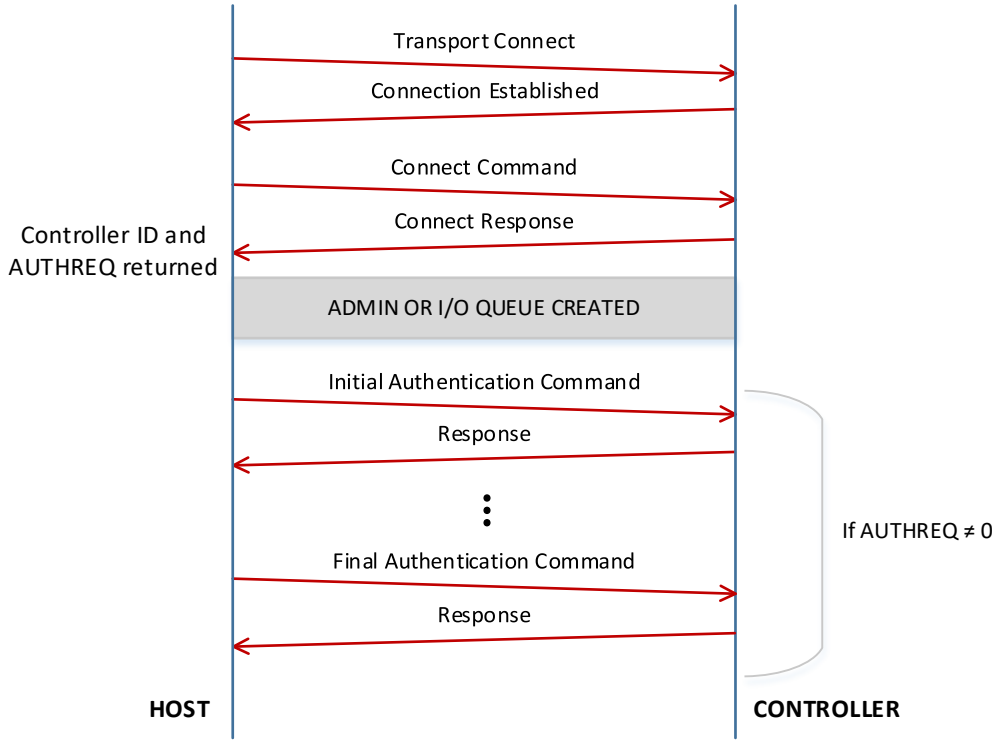
When an Admin Queue is first created, the associated controller is disabled (i.e., CC.EN is initialized to ‘0’). A disabled controller shall abort all commands other than Fabrics commands on the Admin Queue with a status of Command Sequence Error. After the controller is enabled, it shall accept all 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 29 is a ladder diagram that describes the queue creation process for an Admin Queue or an I/O Queue.

Figure 29: Queue Creation Flow



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 it 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 the number of I/O Submission Queues and I/O Completion Queues supported using the Set Features command with the Number of Queues feature identifier; and
8. If the host desires asynchronous notification of optional events, the host should issue a Set Features command specifying the events to enable. If the host desires asynchronous notification of events, the host should submit an appropriate number of Asynchronous Event Request commands. This step may be done at any point after the controller signals it 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 Shutdown

To shutdown the controller, the host should set the Shutdown Notification (CC.SHN) field to 01b to indicate a normal shutdown operation using the Property Set command. After the host specifies a shutdown, the host may either disconnect at the NVMe Transport level or it may choose to poll CSTS.SHST to determine when the shutdown is complete (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 the shutdown.

The CC.EN field is not used to shutdown the controller (it is used for Controller Reset). As part of a shutdown, the CC.EN field is cleared to '0'. After a shutdown has been initiated and while:

- the CC.EN field is cleared to '0' after a shutdown; or
- the CSTS.RDY field is cleared to '0' after a shutdown,

only Fabrics commands are processed by the controller and the Keep Alive timer (if supported) is disabled.

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; and
- The ability to discover controllers that are statically configured.

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

The host uses the well-known Discovery Service NQN (nqn.2014-08.org.nvmexpress.discovery) in the Connect command 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 Discovery Log entries should return addresses on the same fabric as the Discovery Service.

The Keep Alive command is reserved for Discovery controllers. A transport may specify a fixed Discovery controller activity timeout value (e.g., 2 minutes). If no commands are received by 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 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. In NVMe over Fabrics revision 1.0, there is no indication of preferred ports or active/passive port state.

5.1 Discovery Controller Initialization

The initialization process for the Discovery controller is described below:

1. NVMe in-band 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 it 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 30.

Figure 30: 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 31.

Figure 31: 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

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

The Discovery controller shall support the Discovery Log Page. The log pages that a Discovery controller may support are shown in Figure 32.

Figure 32: 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 33.

Figure 33: Discovery Controller – Identify Controller Data Structure

Bytes	O/M[1]	Description
63:00		Reserved
71:64	M	**Firmware Revision (FR):** Refer to Figure 90 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 90 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 90 in the NVMe Base specification.
83:80	M	**Version (VER):** Refer to Figure 90 in the NVMe Base specification.
260:84		Reserved
261	M	**Log Page Attributes (LPA):** Refer to Figure 90 in the NVMe Base specification.
262	M	**Error Log Page Attributes (ELPE):** Refer to Figure 90 in the NVMe Base specification.
513:263		Reserved
515:514	M	**Maximum Outstanding Commands (MAXCMD):** Refer to Figure 90 in the NVMe Base specification.
535:516		Reserved
539:536	M	**SGL Support (SGLS):** Refer to Figure 90 in the NVMe Base specification.
767:540		Reserved
1023:768	M	**NVM Subsystem NVMe Qualified Name (SUBNQN):** Refer to Figure 90 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 is used (if needed) 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 34. The format for the Discovery Log is defined in Figure 35.

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 it 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 it 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 34: Get Log Page – Discovery Log Page Entry

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td>Transport Type (TRTYPE): Specifies the NVMe Transport type.

<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>RDMA Transport (refer to section 7.3)</td></tr>
<tr><td>02</td><td>Fibre Channel Transport (refer to INCITS 540)</td></tr>
<tr><td>03 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>Address Family (ADRFAM): Specifies the address family.

<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>AF_INET: IPV4 address family. Address format specified in IETF RFC 791.</td></tr>
<tr><td>02</td><td>AF_INET: IPV6 address family. Address format specified in IETF RFC 2373</td></tr>
<tr><td>03</td><td>AF_IB: InfiniBand address family.</td></tr>
<tr><td>04</td><td>Fibre Channel address family.</td></tr>
<tr><td>05 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 34: Get Log Page – Discovery Log Page Entry

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>02</td><td>Subsystem Type (SUBTYPE): 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>Transport Requirements (TREQ): Indicates requirements for the NVMe Transport.

Bits 7:2 are reserved.

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>Port ID (PORTID): 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>Controller ID (CNTLID): 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>Admin Max SQ Size (ASQSZ): 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>Transport Service Identifier (TRSVCID): 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>NVM Subsystem Qualified Name (SUBNQN): 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>Transport Address (TRADDR): 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>Transport Specific Address Subtype (TSAS): 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 35: 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 maximum count is exceeded, it wraps to a value of 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 34.
3071:2048	**Discovery Log Entry 1 (DLE1):** Contains the second Discovery Log Entry as defined in Figure 34 (if present).
…	…
(((N + 2) × 1024) - 1): ((N + 1) × 1024)	**Discovery Log Entry N (DLEN):** Contains the Nth Discovery Log Entry as defined in Figure 34 (if present).

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 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 34) 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 level 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 termination of the NVMe Transport connection and end 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 4.10 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.

7.1.1 Submission Queue Head Pointer Update Optimization

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); and
b) is not a Connect 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 7.11 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 for the Admin Queue from the same or another host in order to form a new association.

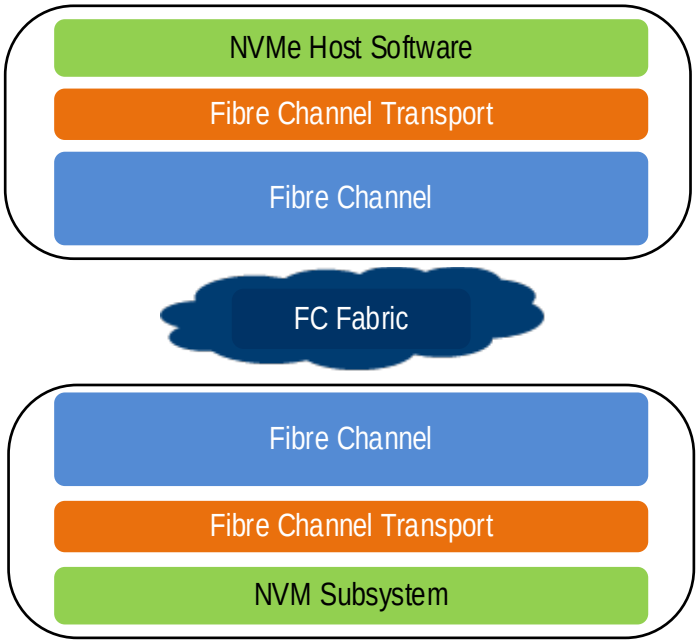
7.2 Transport Capsule and Data Binding: Fibre Channel

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 34 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 36 illustrates the layering of the Fibre Channel Transport within the host and NVM subsystem.

Figure 36: 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 34. Common definitions used for RDMA are defined in Figure 37.

Figure 37: RDMA Definitions

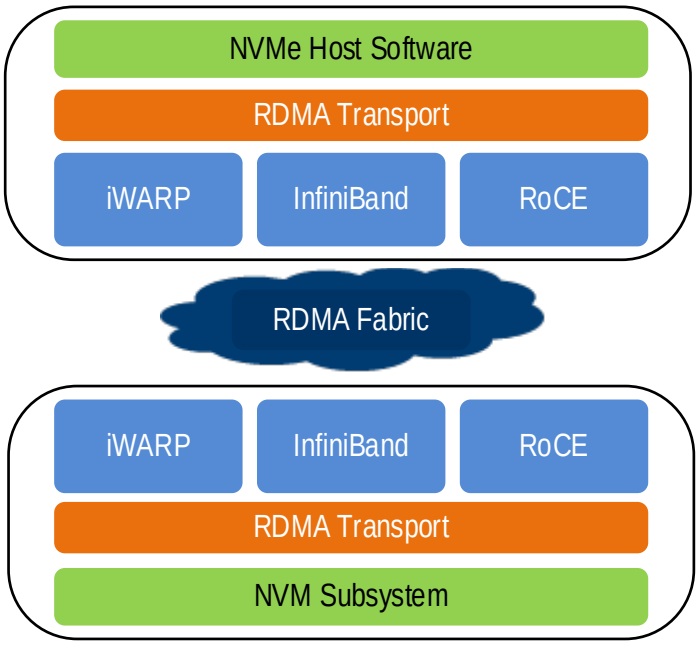
Term	Definition
direct data placement	The use of RDMA_READ or RDMA_WRITE to place data exchanged over the RDMA fabric directly into a host or an NVM subsystem memory buffer as specified in the command.
Established RDMA QP	Two RDMA Queue Pair endpoints that have an established association between them.
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 38 illustrates the layering of the RDMA Transport and common RDMA providers (iWARP, InfiniBand™, and RoCE) within the host and NVM subsystem.

Figure 38: 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 Maximum Data Transfer Size of in-capsule data. 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 39: 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 SGL Data Block 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 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 40.

Figure 40: 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 41.

Figure 41: 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 5.15.1.14 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

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

Figure 42: Transport Specific Address Subtype Definition for RDMA Transport

<table>
<tr><th>Byte</th><th>Description</th></tr>
<tr><td>00</td><td>RDMA QP Service Type: (RDMA_QPTYPE): Specifies the type of RDMA Queue Pair. Valid values are shown in the following table:
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>Reliable Connected</td></tr>
<tr><td>02</td><td>Reliable Datagram</td></tr>
<tr><td>03 to 255</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>01</td><td>RDMA Provider Type: (RDMA_PRTYPE): Specifies the type of RDMA provider. Valid values are shown in the following table:
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>No provider specified</td></tr>
<tr><td>02</td><td>InfiniBand</td></tr>
<tr><td>03</td><td>InfiniBand RoCE</td></tr>
<tr><td>04</td><td>InfiniBand RoCEV2</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>RDMA Connection Management Service: (RDMA_CMS): Specifies the type of RDMA Connection Management Service. Valid values are shown in the following table:
<table>
<tr><th>Value</th><th>Definition</th></tr>
<tr><td>00</td><td>Reserved</td></tr>
<tr><td>01</td><td>RDMA_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>RDMA_PKEY: 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>

7.3.6.2 Discovery Log Page Entry Fields

The RDMA Transport's use of the Discovery Log Page Entry fields shown in Figure 34 is dependent on the value of the RDMA_CMS field. Figure 43 shows the use of these fields. Refer to section 1.5 in the NVMe Base specification for ASCII string format requirements.

Figure 43: 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 34. 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 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 it to the remote-peer using these facilities. The remote-peer uses the host’s IRD value to limit the number of RDMA_READ operations it 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 44: RDMA_CM_REQUEST Private Data Format

Byte	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.
31:08	Reserved

Figure 45: RDMA_CM_ACCEPT Private Data Format

Byte	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 44. RDMA Transports that do not use RNR_RETRY shall set this value to be equal to the value of HSQSIZE specified in Figure 44.
31:04	Reserved

Figure 46: RDMA_CM_REJECT Private Data Format

Byte	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 47.

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 40). 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 47.

Figure 47: 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 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.