Mallikarjun Chadalapaka
HP
John Hufferd
IBM
Julian Satran
IBM
Hemal Shah
Intel

Expires April 2007

Datamover Architecture for iSCSI (DA)

Status of this Memo

Abstract

iSCSI is a SCSI transport protocol that maps the SCSI family of application protocols onto TCP/IP. Datamover Architecture

for iSCSI (DA) defines an abstract model in which the movement of data between iSCSI end nodes is logically separated from the rest of the iSCSI protocol in order to allow iSCSI to adapt to innovations available in new IP transports. *While DA defines the architectural functions required of the class of Datamover protocols, it does not define any specific Datamover protocols. Each such* Datamover protocol*, to be defined in a separate document,* provides a reliable transport for all iSCSI PDUs, but actually moves the data required for certain iSCSI PDUs without involving the remote iSCSI layer itself. This document begins with an introduction of a few new abstractions, defines a layered architecture for iSCSI and Datamover protocols, and then models the interactions within an iSCSI end node between the iSCSI layer and the Datamover layer that happen in order to transparently perform remote data movement within an IP fabric. It is intended that this definition would help map iSCSI to generic RDMA-capable IP fabrics in the future comprising TCP, SCTP, and possibly other underlying network transport layers *such as InfiniBand*.

Table of Contents

Table of Figures

1 Definitions and acronyms

1.1 Definitions

I/O Buffer – A buffer that is used in a SCSI Read or Write operation so SCSI data may be sent from or received into that buffer.

Datamover protocol – A Datamover protocol is a data transfer wire protocol for iSCSI that meets the requirements stated in section 6.

Datamover layer – A Datamover layer is a protocol layer within an end node that implements the Datamover protocol.

Datamover-assisted - An iSCSI connection is said to be “Datamover-assisted” when a Datamover layer is enabled for moving control and data information on that iSCSI connection.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

1.2 Acronyms

Acronym	Definition
DA	Datamover Architecture for iSCSI
DDP	Direct Data Placement Protocol
DI	Datamover Interface
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
I/O	Input - Output
IP	Internet Protocol
iSCSI	Internet SCSI
iSER	iSCSI Extensions for RDMA

ITT	Initiator Task Tag
LO	Leading Only
MPA	Marker PDU Aligned Framing for TCP
PDU	Protocol Data Unit
RDDP	Remote Direct Data Placement
RDMA	Remote Direct Memory Access
R2T	Ready To Transfer
R2TSN	Ready To Transfer Sequence Number
RDMA	Remote Direct Memory Access
RDMAP	Remote Direct Memory Access Protocol
RFC	Request For Comments
SAM	SCSI Architecture Model
SCSI	Small Computer Systems Interface
SN	Sequence Number
SNACK	Selective Negative Acknowledgment - also Sequence Number Acknowledgement for data
TCP	Transmission Control Protocol
TTT	Target Transfer Tag

2 Motivation

2.1 Intent

There are relatively new standard protocols that enable Remote Direct Memory Access (RDMA) and Remote Direct Data Placement (RDDP) technologies to work over IP fabrics. The principal value proposition of these technologies is that they enable one end node to place data in the final intended buffer on the remote end node, thus eliminating the data copy that traditionally happens in the receive path to move the data to the final buffer. The data copy avoidance in turn eliminates unnecessary memory bandwidth consumption, substantially decreases the reassembly buffer size requirements, and preserves CPU cycles that would otherwise be spent in copying.

The iSCSI specification ([RFC3720]) defines a very detailed data transfer model that employs SCSI Data-In PDUs, SCSI Data-Out PDUs, and R2T PDUs, in addition to the SCSI Command and SCSI Response PDUs that respectively create and conclude the task context for the data transfer. In the traditional iSCSI model, the iSCSI protocol layer plays the central role in pacing the data transfer and carrying out the ensuing data transfer itself. An alternative architecture would be for iSCSI to delegate a large part of this data transfer role to a separate protocol layer exclusively designed to move data, which in turn is possibly aided by a data movement and placement technology such as RDMA.

If iSCSI were operating in such RDMA environments, iSCSI would be shielded from the low-level data transfer mechanics but would only be privy to the conclusion of the requested data transfer Thus, there would be an effective "off-loading" of the work that an iSCSI protocol layer is expected to perform, compared to today's iSCSI end nodes. For such RDMA environments, it is highly desirable that there be a standard architecture to separate the data movement part of the iSCSI protocol definition from the rest of the iSCSI functionality. This architecture precisely defines what a Datamover layer is and also describes the model of interactions between the iSCSI layer and the Datamover layer (section 6). In order to satisfy this need, this document presents a Datamover Architecture for iSCSI(DA) and also summarizes a reasonable model for interactions between the iSCSI layer and the Datamover layer for each of the iSCSI

PDUs that are defined in [RFC3720]. Note that while DA is motivated by the advent of RDMA over TCP/IP technology, the architecture is not dependent on RDMA in its design. DA is intended to be a generic architectural framework for allowing different types of Datamovers based on different types of RDMA and transport protocols. Adoption of this model will help iSCSI proliferate into more environments.

2.2 Interpretation of Requirements

This draft introduces certain architectural abstractions and builds an abstract functional interface model between iSCSI and Datamover protocol layers based on those abstractions. This architectural style is motivated by the following desires:

a) Provide guidance to Datamover protocol designers with respect to the functional boundary between iSCSI and the Datamover protocols. This guidance is critical since a significant part of the [RFC3720] protocol definition is left unchanged by *DA architecture* and the iSCSI notions from [RFC3720] (e.g., tasks, ITTs) are leveraged by the Datamover protocol.

b) Aid existing iSCSI implementations to rapidly adapt to *DA architecture*, largely by leveraging the architectural abstractions also into implementation constructs – e.g., functions, APIs, modules.

However, note that *DA a*rchitecture does not intend to impose any implementation specifics per se. When a DA architectural concept (e.g., Operational Primitive) is described as mandatory ("MUST") or recommended ("SHOULD") of a layer (iSCSI or Datamover) in this document, the intent is that an implementation respectively MUST or SHOULD produce the same protocol action as what the model describes. Specifically, no implementation compliance in terms of names, modules or API arguments etc. is implied by this Architecture by such use of [RFC2119] terms, only a functional compliance is sought.

3 Architectural layering of iSCSI and Datamover layers

Figure 1 illustrates an example of the architectural layering of iSCSI and Datamover layers, in conjunction with a TCP/IP implementation of RDMAP/DDP layers in an iSCSI end node. Note that RDMAP/DDP/MPA, and TCP protocol layers are shown here only as an example and in reality, DA is completely oblivious to protocol layers below the Datamover layer. The RDMAP/DDP/MPA protocol stack provides a generic transport service with direct data placement. There is no need to tailor the implementation of this protocol stack to the specific ULP to benefit from these services.

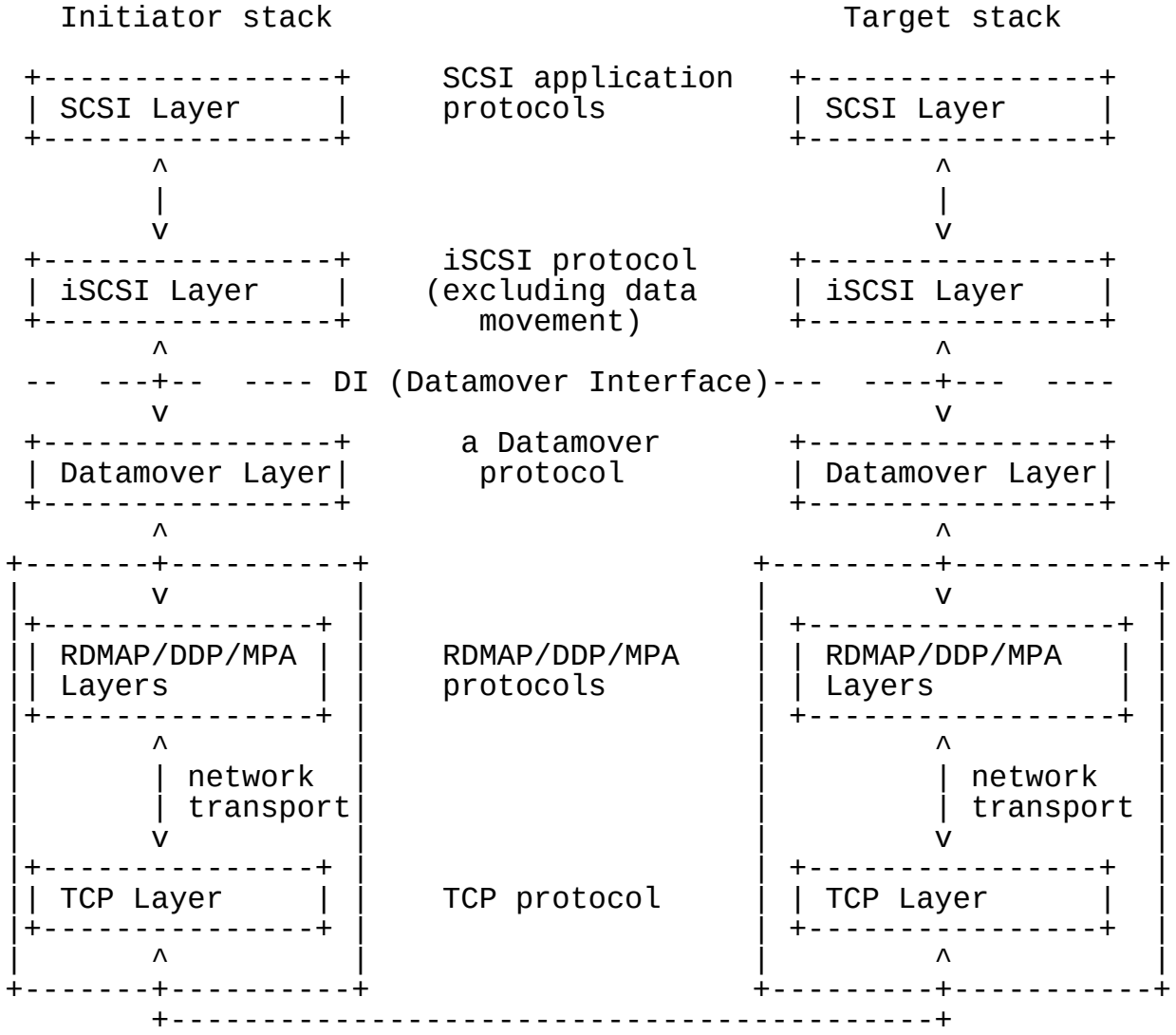

Figure 1 Datamover Architecture diagram, with the RDMAP example

The scope of this document is limited to:

1. Defining the notion of a Datamover layer and a Datamover protocol (section 6),

2. Defining the functionality distribution between the iSCSI layer and the Datamover layer along with the communication model between the two (Operational Primitives), and,

3. Modeling the interactions between the blocks labeled as "iSCSI Layer" and "Datamover Layer" in Figure 1 – i.e. defining the interface labeled as “DI” in the figure - for each defined iSCSI PDU, based on the Operational Primitives.

4 Design Overview

This document discusses and defines a model for interactions between the iSCSI layer and a "Datamover layer" (see section 6) operating within an iSCSI end node, presumably communicating with one or more iSCSI end nodes with similar layering. The model for interactions for handling different iSCSI operations is called the “Datamover Interface” (DI, section 10), while the architecture itself is called “Datamover Architecture for iSCSI” (DA). It is likely that the architecture will have implications on the Datamover wire protocols as DA places certain requirements and functionality expectations on the Datamover layer. However, this document itself neither defines any new wire protocol for the Datamover layer, nor any potential modifications to the iSCSI wire protocol to employ the Datamover layer. The scope of this document is strictly limited to specifying the architectural framework and the minimally required interactions that happen within an iSCSI end node to leverage the Datamover layer.

The design ideas behind DA can be summarized thus –

1) DA defines an abstract functional interface model of iSCSI layer’s interactions with a Datamover layer below – i.e. DA models the interactions between the logical “bottom” interface of iSCSI and the logical “top” interface of a Datamover.

2) DA guides the wire protocol for a Datamover layer by defining the iSCSI knowledge that the Datamover layer may utilize in its protocol definition (as an example, this draft completely limits the notion of “iSCSI session” to the iSCSI layer).

3) DA is designed to allow implementing the Datamover layer either in hardware or in software.

4) DA is not a wire protocol spec, but an architecture that also models the interactions between iSCSI and Datamover layers operating within an iSCSI end node.

5) DA by design seeks to model the iSCSI-Datamover interactions in a way that the modeling is independent of the specifics of either a particular iSCSI revision, or a specific instantiation of a Datamover layer.

6) DA introduces and relies on the notion of a defined set of Operational Primitives (could be seen as entry point definitions in implementation terms) provided by each layer to the other to carry out the request-response interactions.

7) DA is intended to allow Datamover protocol definitions with minimal changes to existing iSCSI implementations.

8) DA is designed to allow the iSCSI layer to completely rely on the Datamover layer for all the data transport needs.

9) DA models the architecturally required minimal interactions between an operational iSCSI layer and a Datamover layer to realize the iSCSI-transparent data movement. There may be several other interactions in a typical implementation in order to bootstrap a Datamover layer (or an iSCSI layer) into operation, and they are outside the scope of this document.

Note that in summary, DA is architected to support many different Datamover protocols operating under the iSCSI layer. One such example of a Datamover protocol is iSER ([iSER]).

5 Architectural Concepts

5.1 iSCSI PDU types

This section defines the iSCSI PDU classification terminology, as defined and used in this document. Out of the set of legal iSCSI PDUs defined in [RFC3720], as we will see in section 5.1.1, the iSCSI layer does not request a SCSI Data-Out PDU carrying solicited data for transmission across the Datamover Interface per this architecture. For this reason, the SCSI Data-Out PDU carrying solicited data is excluded in the iSCSI PDU classification we introduce in this section (for SCSI Data-Out PDUs for unsolicited Data, see section 5.1.2). The rest of the legal iSCSI PDUs that may be exchanged across the Datamover Interface are defined to consist of two classes:

1) iSCSI data-type PDUs

2) iSCSI control-type PDUs

5.1.1 iSCSI data-type PDUs

An iSCSI data-type PDU is defined as an iSCSI PDU that causes data transfer, transparent to the remote iSCSI layer, to take place between the peer iSCSI nodes on a full feature phase iSCSI connection. A data-type PDU, when requested for transmission by the sender iSCSI layer, results in the associated data transfer without the participation of the remote iSCSI layer, i.e. the PDU itself is not delivered as-is to the remote iSCSI layer. The following iSCSI PDUs constitute the set of iSCSI data-type PDUs –

1) SCSI Data-In PDU

2) R2T PDU

In an iSCSI end node structured as an iSCSI layer and a Datamover layer as defined in this document, the solicitation for Data-out (i.e. R2T PDU) is not delivered to the initiator iSCSI layer, per the definition of an iSCSI data-type PDU. The data transfer is instead performed via the mechanisms known to the Datamover layer (e.g. RDMA Read). This in turn implies that a SCSI Data-Out PDU for solicited data is never requested for transmission across the Datamover Interface at the initiator.

5.1.2 iSCSI control-type PDUs

Any iSCSI PDU that is not an iSCSI data-type PDU and also not a solicited SCSI Data-out PDU is defined as an iSCSI control-type PDU. Specifically, it is to be noted that SCSI Data-Out PDUs for unsolicited Data are defined as iSCSI control-type PDUs.

5.2 Data_Descriptor

A Data_Descriptor is an information element that describes an iSCSI/SCSI data buffer, provided by the iSCSI layer to its local Datamover layer or by the Datamover layer to its local iSCSI layer for identifying the data associated respectively with the requested or completed operation.

In implementation terms, a Data_Descriptor may be a scatter-gather list describing a local buffer, the exact structure of which is subject to the constraints imposed by the operating environment on the local iSCSI node.

5.3 Connection_Handle

A Connection_Handle is an information element that identifies the particular iSCSI connection for which an inbound or outbound iSCSI PDU is intended. A connection handle is unique for a given pair of an iSCSI layer instance and a Datamover layer instance. The Connection_Handle qualifier is used in all invocations of any Operational Primitive for connection identification.

Note that the Connection_Handle is conceptually different from the Connection Identifier (CID) defined by the iSCSI specification. While the CID is a unique identifier of an iSCSI connection within an iSCSI session, the uniqueness of the Connection_Handle extends to the entire iSCSI layer instance coupled with the Datamover layer instance, across possibly multiple iSCSI sessions.

In implementation terms, a Connection_Handle could be an opaque identifier exchanged between the iSCSI layer and the Datamover layer at the connection login time. One may also consider it to be similar in scope of uniqueness to a socket identifier. The exact structure and modalities of exchange of a Connection_Handle between the two layers is implementation-specific.

5.4 Operational Primitive

An Operational Primitive, in this document, is an abstract functional interface procedure that requests another layer to perform a specific action on the requestor's behalf or notifies the other layer of some event. The Datamover Interface between an iSCSI layer instance and a Datamover layer instance within an iSCSI end node uses a set of Operational Primitives to define the functional interface between the two layers. Note that not every invocation of an Operational Primitive may elicit a response from the requested layer. This document describes the types of Operational Primitives that are implicitly required and provided by the iSCSI protocol layer as defined in [RFC3720], and the semantics of these Primitives.

Note that ownership of buffers and data structures is likely to be exchanged between the iSCSI layer and its local Datamover layer in invoking the Operational Primitives defined in this architecture. The buffer management details, including how buffers are allocated and released, are implementation-specific and thus are outside the scope of this document.

Each Operational Primitive invocation needs a certain "information context" (e.g., Connection_Handle) for performing the specific action being requested of it. The required information context is described in this document by a listing of "qualifiers" on each invocation - in the style of function call arguments. No implementation specific is however implied in this notation. The "qualifiers" of any Operational Primitive invocation specified in this document thus represent the mandatory information context that the Operational Primitive invocation MUST consider in performing the action. While the qualifiers are required, the method of realizing the qualifiers (passed synchronously with

invocation, or retrieved from task context, or retrieved from shared memory etc.) is really up to the implementations.

When an Operational Primitive implementation is described as mandatory ("MUST") or recommended ("SHOULD") of a layer (iSCSI or Datamover) in this document, the intent is that an implementation respectively MUST or SHOULD produce the same protocol action as what the model describes.

5.5 Transport Connection

The term “Transport Connection” is used in this document as a generic term to represent the end-to-end logical connection as defined by the underlying reliable transport protocol. For this revision of this document, a Transport Connection means only a TCP connection.

6 Datamover layer and Datamover protocol

This section introduces the notion of a "Datamover layer" and “Datamover protocol” as meant in this document, and defines the requirements on a Datamover protocol.

A Datamover layer is the implementation component that realizes a Datamover protocol functionality in an iSCSI-capable end node, in communicating with other iSCSI end nodes with similar capabilities. More specifically, a “Datamover layer” MUST provide the following functionality and the “Datamover protocol” MUST consist of the wire protocol required to realize the following functionality –

1) guarantee that all the necessary data transfers take place when the local iSCSI layer requests transmitting a command (in order to complete a SCSI command, for an initiator),or sending/receiving an iSCSI data sequence (in order to complete part of a SCSI command, for a target).

2) transport an iSCSI control-type PDU as-is to the peer Datamover layer when requested to do so by the local iSCSI layer.

3) provide notification and delivery to the iSCSI layer upon arrival of an iSCSI control-type PDU.

4) provide an initiator-to-target data acknowledgement of SCSI read data back to the target iSCSI layer, when requested.

5) provide an asynchronous notification upon completion of a requested data transfer operation that moved data without involving the iSCSI layer.

6) place the SCSI data into the I/O buffers or pick up the SCSI data for transmission out of the data buffers that the iSCSI layer had requested to be used for a SCSI I/O.

7) provide an error-free (i.e. must have at least the same level of assurance of data integrity as the CRC32C iSCSI data digest), reliable, in-order delivery transport mechanism over IP networks in performing the data transfer, and asynchronously notify the iSCSI layer upon iSCSI connection termination.

Note that this architecture expects that each compliant Datamover protocol will define the precise means of satisfying the requirements specified in this section.

In order to meet the functional requirements listed in this section, certain Datamover protocols may require pre-posted buffers from the local iSCSI protocol layer via mechanisms outside the scope of this document and in some implementations, the absence of such buffers may result in a connection failure. Datamover protocols may also realize these functional requirements via methods not explicitly listed in this document.

7 Functional Overview

This section presents an overview of the functional interactions between the iSCSI layer and the Datamover layer as intended by this Architecture.

7.1 Startup

The iSCSI Login Phase on an iSCSI connection occurs as defined in [RFC3720]. The Architecture assumes that at the end of the Login Phase, both the initiator and target, if they had so decided, transition the connection to being Datamover-assisted. The precise means of how an iSCSI initiator and an iSCSI target agree on having the connection Datamover-assisted is defined by the Datamover protocol. The only architectural requirement is that all iSCSI interactions in the iSCSI Full Feature Phase MUST be Datamover-assisted subject to the prior agreement, meaning that Datamover protocol is in the iSCSI-to-iSCSI communication path below the iSCSI layer on either side as shown in Figure 1. DA defines the Enable_Datamover Operational Primitive (section 8.6) to bring about this transition to a Datamover-assisted connection.

The Architecture also assumes that the Datamover layer may require a certain number of opaque local resources for making a connection Datamover-assisted. DA thus defines the Allocate_Connection_Resources Operational Primitive (section 8.4) to model this interaction. This Primitive is intended to be invoked on each side once the two sides decide (as previously noted) to have the connection Datamover-assisted. The expected sequence of Primitive invocations is depicted in Figure 2 and Figure 3 in section 16.2. Figure 4, Figure 5, and Figure 6 illustrate how the Primitives may be employed to deal with various legal login outcomes.

7.2 Full Feature Phase

All iSCSI peer communication in the Full Feature Phase happens through the Datamover layers if the iSCSI connection is Datamover-assisted. The Architecture assumes that a Datamover layer may require a certain number of opaque local resources for each new iSCSI task. In the normal course of execution, these task-level resources in the Datamover layer

are assumed to be transparently allocated on each task initiation and deallocated on the conclusion of each task as appropriate. In exception scenarios however – in scenarios that do not yield a SCSI Response for each task such as ABORT TASK operation – the Architecture assumes that the Datamover layer needs to be notified of the individual task terminations to aid its task-level resource management. DA thus defines the Deallocate_Task_Resources Operational Primitive (section 8.9) to model this task-resource management. In specifying the ITT qualifier for the Deallocate_Task_Resources Primitive, the Architecture further assumes that the Datamover layer tracks its opaque task-level local resources by the iSCSI ITT. DA also defines Send_Control (section 8.1), Put_Data (section 8.2), Get_Data (section 8.3), Data_Completion_Notify(section 9.3), Data_ACK_Notify (section 9.4), and Control_Notify (section 9.1) Operational Primitives to model the various Full Feature Phase interactions.

Figure 9, Figure 10, and Figure 11 in section 16.2 show some Full Feature Phase interactions – SCSI Write task, SCSI Read task, and a SCSI Read Data acknowledgement respectively. Figure 12 in section 16.2 illustrates how an ABORT TASK operation can be modeled leading to deterministic resource cleanup on the Datamover layer.

7.3 Wrapup

Once an iSCSI connection becomes Datamover-assisted, the connection continues in that state till the end of the Full Feature Phase, i.e. the termination of the connection. The Architecture assumes that when a connection is normally logged out, the Datamover layer needs to be notified so that its connection-level opaque resources (see section 7.1) may now be freed up. DA thus defines a Connection_Terminate Operational Primitive (section 8.7) to model this interaction. The Architecture further assumes that when a connection termination happens without iSCSI layer's involvement (e.g., TCP RST), the Datamover layer is capable of locally cleaning up its task-level and connection-level resources before notifying the iSCSI layer of the fact. DA thus defines the Connection_Terminate_Notify Operational Primitive (section 9.2) to model this interaction.

Figure 7 and Figure 8 in section 16.2 illustrate the interactions between the iSCSI and Datamover layers in normal and unexpected connection termination scenarios.

8 Operational Primitives provided by the Datamover layer

While the iSCSI specification itself does not have a notion of Operational Primitives, any iSCSI layer implementing the iSCSI specification functionally requires the following Operational Primitives from its Datamover layer. Thus, any Datamover protocol compliant with this architecture MUST implement the Operational Primitives described in this section. These Operational Primitives are invoked by the iSCSI layer as appropriate. Unless otherwise stated, all the following Operational Primitives may be used both on the initiator side and the target side. In general programming terminology, this set of Operational Primitives may be construed as “down calls”.

1) Send_Control
2) Put_Data
3) Get_Data
4) Allocate_Connection_Resources
5) Deallocate_Connection_Resources
6) Enable_Datamover
7) Connection_Terminate
8) Notice_Key_Values
9) Deallocate_Task_Resources

8.1 Send_Control

Input qualifiers: Connection_Handle, iSCSI PDU-specific qualifiers

Return Results: Not specified.

An iSCSI layer requests its local Datamover layer to transmit an iSCSI control-type PDU to the peer iSCSI layer operating in the remote iSCSI node by this Operational Primitive. The Datamover layer performs the requested operation, and may add its own protocol headers in doing so. The iSCSI layer MUST

NOT invoke the Send_Control Operational Primitive on an iSCSI connection that is not yet Datamover-assisted.

An initiator iSCSI layer requesting the transfer of a SCSI command PDU or a target iSCSI layer requesting the transfer of a SCSI response PDU are examples of invoking the Send_Control Operational Primitive. As section 10.3.1 illustrates later on, the iSCSI PDU-specific qualifiers in this example are: BHS and AHS, DataDescriptorOut, DataDescriptorIn, ImmediateDataSize, and UnsolicitedDataSize

8.2 Put_Data

Input qualifiers: Connection_Handle, contents of a SCSI Data-In PDU header, Data_Descriptor, Notify_Enable

Return Results: Not specified.

An iSCSI layer requests its local Datamover layer to transmit the data identified by the Data_Descriptor for the SCSI Data-In PDU to the peer iSCSI layer on the remote iSCSI node by this Operational Primitive. The Datamover layer performs the operation by using its own protocol means, completely transparent to the remote iSCSI layer. The iSCSI layer MUST NOT invoke the Put_Data Operational Primitive on an iSCSI connection that is not yet Datamover-assisted.

The Notify_Enable qualifier is used to request the local Datamover layer to generate or to not generate the eventual local completion notification to the iSCSI layer for this Put_Data invocation. For detailed semantics of this qualifier, see section 9.3.

A Put_Data Primitive may only be invoked by an iSCSI layer on the target to its local Datamover layer.

A target iSCSI layer requesting the transfer of an iSCSI read data sequence (also known as a read burst) is an example of invoking the Put_Data Operational Primitive.

8.3 Get_Data

Input qualifiers: Connection_Handle, contents of an R2T PDU, Data_Descriptor, Notify_Enable

Return Results: Not specified.

An iSCSI layer requests its local Datamover layer to retrieve certain data identified by the R2T PDU from the peer iSCSI layer on the remote iSCSI node into the buffer identified by the Data_Descriptor by invoking this Operational Primitive. The Datamover layer performs the operation by using its own protocol means, completely transparent to the remote iSCSI layer. The iSCSI layer MUST NOT invoke the Get_Data Operational Primitive on an iSCSI connection that is not yet Datamover-assisted.

The Notify_Enable qualifier is used to request the local Datamover layer to generate or to not generate the eventual local completion notification to the iSCSI layer for this Get_Data invocation. For detailed semantics of this qualifier, see section 9.3.

A Get_Data Primitive may only be invoked by an iSCSI layer on the target to its local Datamover layer.

A target iSCSI layer requesting the transfer of an iSCSI write data sequence (also known as a write burst) is an example of invoking the Get_Data Operational Primitive.

8.4 Allocate_Connection_Resources

Input qualifiers: Connection_Handle[, Resource_Descriptor]

Return Results: Status.

By invoking this Operational Primitive, an iSCSI layer requests its local Datamover layer to perform all the Datamover-specific resource allocations required for the full feature phase of an iSCSI connection. The Connection_Handle

identifies the connection the iSCSI layer is requesting the resource allocation for in order to eventually transition the connection to be a Datamover-assisted iSCSI connection. Note that the Datamover layer however does not allocate any Datamover-specific task-level resources upon invocation of this Primitive.

An iSCSI layer, in addition, optionally specifies the implementation-specific resource requirements for the iSCSI connection to the Datamover layer, by passing an input qualifier called Resource_Descriptor. The exact structure of a Resource_Descriptor is implementation-dependent, and hence structurally opaque to DA.

A return result of Status=success means that the Allocate_Connection_Resources invocation corresponding to that Connection_Handle succeeded. If an Allocate_Connection_Resources invocation is made for a Connection_Handle for which an earlier invocation succeeded, the return Status must be success and the request will be ignored by the Datamover layer. A return result of Status=failure means that the Allocate_Connection_Resources invocation corresponding to that Connection_Handle failed. There MUST NOT be more than one Allocate_Connection_Resources Primitive invocation outstanding for a given Connection_Handle at any time.

The iSCSI layer must invoke the Allocate_Connection_Resources Primitive before the invocation of the Enable_Datamover Primitive.

8.5 Deallocate_Connection_Resources

Input qualifiers: Connection_Handle

Return Results: Not specified.

By invoking this Operational Primitive, an iSCSI layer requests its local Datamover layer to deallocate all the Datamover-specific resources that may have been allocated earlier for the Transport Connection identified by the Connection_Handle. The iSCSI layer may invoke this

Operational Primitive when the Datamover-specific resources associated with the Connection_Handle are no longer necessary (such as the Login failure of the corresponding iSCSI connection).

8.6 Enable_Datamover

Input qualifiers: Connection_Handle, Transport_Connection_Descriptor [, Final_Login_Response_PDU]

Return Results: Not specified.

By invoking this Operational Primitive, an iSCSI layer requests its local Datamover layer to assist all further iSCSI exchanges on the iSCSI connection (i.e. to make the connection Datamover-assisted) identified by the Connection_Handle, for which the Datamover-specific resource allocation was earlier made. The iSCSI layer MUST NOT invoke the Enable_Datamover Operational Primitive for an iSCSI connection unless there was a corresponding prior resource allocation.

The Final_Login_Response_PDU input qualifier is applicable only for a target, and contains the final Login Response that concludes the iSCSI Login phase and which must be sent as a byte stream as expected by the initiator iSCSI layer. When this qualifier is used, the target-Datamover layer MUST transmit this final Login Response before Datamover assistance is enabled for the Transport Connection.

The iSCSI layer identifies the specific Transport Connection associated with the Connection_Handle to the Datamover layer by specifying the Transport_Connection_Descriptor. The exact structure of this Descriptor is implementation-dependent.

8.7 Connection_Terminate

Input qualifiers: Connection_Handle

Return Results: Not specified.

By invoking this Operational Primitive, an iSCSI layer requests its local Datamover layer to terminate the Transport Connection and deallocate all the connection and task resources associated with the Connection_Handle. When this Operational Primitive invocation returns to the iSCSI layer, the iSCSI layer may assume the full ownership of all the iSCSI-level resources, e.g. I/O Buffers, associated with the connection. This Operational Primitive may be invoked only with a valid Connection_Handle and the Transport Connection associated with the Connection_Handle must already be Datamover-assisted.

8.8 Notice_Key_Values

Input qualifiers: Connection_Handle, Number of keys, a list of Key-Value pairs

Return Results: Not specified.

By invoking this Operational Primitive, an iSCSI layer requests its local Datamover layer to take note of the negotiated values of the listed keys for the Transport Connection. This Operational Primitive may be invoked only with a valid Connection_Handle and the Key-Value pairs MUST be the current values that were successfully agreed upon by the iSCSI peers for the connection. The Datamover layer may use the values of the keys to aid the Datamover operation as it deems appropriate. The specific keys to be passed in as input qualifiers and the point(s) in time this Operational Primitive is invoked are implementation-dependent.

8.9 Deallocate_Task_Resources

Input qualifiers: Connection_Handle, ITT

Return Results: Not specified.

By invoking this Operational Primitive, an iSCSI layer requests its local Datamover Layer to deallocate all Datamover-specific resources that earlier may have been allocated for the task identified by the ITT qualifier. The iSCSI layer uses this Operational Primitive during exception processing when one or more active tasks are to be terminated without corresponding SCSI Response PDUs. This Primitive MUST be invoked for each active task terminated without a SCSI Response PDU. This Primitive MUST NOT be invoked by the

iSCSI layer when a SCSI Response PDU normally concludes a task. When a SCSI Response PDU normally concludes a task (even if the SCSI Status was not a success), the Datamover layer is assumed to have automatically deallocated all Datamover-specific task resources for that task. Refer to section 7.2 for a related discussion on the Architectural assumptions on the task-level Datamover resource management, especially with respect to when the resources are assumed to be allocated.

9 Operational Primitives provided by the iSCSI layer

While the iSCSI specification itself does not have a notion of Operational Primitives, any iSCSI layer implementing the iSCSI specification would have to provide the following Operational Primitives to its local Datamover layer. Thus, any iSCSI protocol implementation compliant with this architecture MUST implement the Operational Primitives described in this section. These Operational Primitives are invoked by the Datamover layer as appropriate and when the iSCSI connection is Datamover-assisted. Unless otherwise stated, all the following Operational Primitives may be used both on the initiator side and the target side. In general programming terminology, this set of Operational Primitives may be construed as “up calls”.

1) Control_Notify
2) Connection_Terminate_Notify
3) Data_Completion_Notify
4) Data_ACK_Notify

9.1 Control_Notify

Input qualifiers: Connection_Handle, an iSCSI control-type PDU.

Return Results: Not specified.

A Datamover layer notifies its local iSCSI layer, via this Operational Primitive, of the arrival of an iSCSI control-type PDU from the peer Datamover layer on the remote iSCSI node. The iSCSI layer processes the control-type PDU as defined in [RFC3720].

A target iSCSI layer being notified of the arrival of a SCSI Command is an example of invoking the Control_Notify Operational Primitive.

Note that implementations may choose to describe the “iSCSI control-type PDU” qualifier in this notification using a Data_Descriptor (section 5.2) and not necessarily one contiguous buffer.

9.2 Connection_Terminate_Notify

Input qualifiers: Connection_Handle

Return Results: Not specified.

A Datamover layer notifies its local iSCSI layer on an unsolicited termination or failure of an iSCSI connection providing the Connection_Handle associated with the iSCSI Connection. The iSCSI Layer MUST consider the Connection_Handle to be invalid upon being so notified. The iSCSI layer processes the connection termination as defined in [RFC3720]. The Datamover layer MUST deallocate the connection and task resources associated with the terminated connection before notifying the iSCSI layer of the termination via this Operational Primitive.

A target iSCSI layer being notified *of an ungraceful connection termination by the Datamover layer when the underlying Transport Connection is torn down due to receiving a* TCP RESET is *one* example when the Connection_Terminate_Notify Operational Primitive is invoked.

9.3 Data_Completion_Notify

Input qualifiers: Connection_Handle, ITT, SN

Return Results: Not specified.

A Datamover layer notifies its local iSCSI layer on completing the retrieval of the data or upon sending the data, as requested in a prior iSCSI data-type PDU, from/to the peer Datamover layer on the remote iSCSI node via this Operational Primitive. The iSCSI layer processes the operation as defined in [RFC3720].

SN may be either the DataSN associated with the SCSI Data-In PDU or R2TSN associated with the R2T PDU depending on the SCSI operation. Note that, for targets, a TTT (see [RFC3720]) could have been specified instead of an SN. However, the considered choice was to leave the SN to be the qualifier for two reasons – a) it is generic and applicable to initiators and targets as well as Data-in and Data-out, and b) having both SN and TTT qualifiers for the notification was considered onerous on the Datamover layer, in terms of state maintenance for each completion notification. The implication of this choice is that iSCSI target implementations will have to adapt to using the ITT-SN tuple in associating the solicited data to the appropriate task, rather than the ITT-TTT tuple for doing the same.

If Notify_Enable was set in either a Put_Data or a Get_Data invocation, the Datamover layer MUST invoke the Data_Completion_Notify Operational Primitive upon completing that requested data transfer. If the Notify_Enable was cleared in either a Put_Data or a Get_Data invocation, the Datamover layer MUST NOT invoke the Data_Completion_Notify Operational Primitive upon completing that requested data transfer.

A Data_Completion_Notify invocation serves to notify the iSCSI layer of the Put_Data or Get_Data completion respectively. As earlier noted in sections 8.2 and 8.3, specific Datamover protocol definitions may restrict the usage scope of Put_Data and Get_Data, and thus implicitly the usage scope of Data_Completion_Notify.

A target iSCSI layer being notified of the retrieval of a write data sequence is an example of invoking the Data_Completion_Notify Operational Primitive.

9.4 Data_ACK_Notify

Input qualifiers: Connection_Handle, ITT, DataSN

Return Results: Not specified.

A target Datamover layer notifies its local iSCSI layer of the arrival of a previously requested data acknowledgement from the peer Datamover layer on the remote (initiator) iSCSI node via this Operational Primitive. The iSCSI layer processes the data acknowledgement notification as defined in [RFC3720].

A target iSCSI layer being notified of the arrival of a data acknowledgement for a certain SCSI Read data PDU is the only example of invoking the Data_ACK_Notify Operational Primitive.

10 Datamover Interface (DI)

10.1 Overview

This chapter describes the interactions model between iSCSI and Datamover layers when the iSCSI connection is Datamover-assisted so the iSCSI layer may carry out the following -

- send iSCSI data-type PDUs and exchange iSCSI control-type PDUs, and
- handle asynchronous notifications such as completion of data sequence transfer, and connection failure.

This chapter relies on the notion of Operational Primitives (section 5.4) to define DI.

10.2 Interactions for handling asynchronous notifications

10.2.1 Connection termination

As stated in section 9.2, the Datamover layer notifies the iSCSI layer of a failed or terminated connection via the Connection_Terminate_Notify Operational Primitive. The iSCSI layer MUST consider the connection as unusable upon the invocation of this Primitive and handle the connection termination as specified in [RFC3720].

10.2.2 Data transfer completion

As stated in section 9.3, the Datamover layer notifies the iSCSI layer of a completed data transfer operation via the Data_Completion_Notify Operational Primitive. The iSCSI layer processes the transfer completion as specified in [RFC3720].

10.2.2.1 Completion of a requested SCSI Data transfer

The Datamover layer, to notify the iSCSI layer of the completion of a requested iSCSI data-type PDU transfer, uses the Data_Completion_Notify Operational Primitive with the following input qualifiers.

a) Connection_Handle

b) ITT: Initiator Task Tag semantics as defined in [RFC3720]

c) SN: DataSN for a SCSI Data-in/Data-out PDU, and R2TSN for an iSCSI R2T PDU. The semantics for both types of sequence numbers are as defined in [RFC3720].

The rationale for choosing SN is explained in section 9.3.

Every invocation of the Data_Completion_Notify Operational Primitive MUST be preceded by an invocation of the Put_Data or Get_Data Operational Primitive with the Notify_Enable qualifier set by the iSCSI layer at an earlier point in time.

10.2.3 Data acknowledgement

[RFC3720] allows the iSCSI targets to optionally solicit data acknowledgement from the initiator for one or more Data-in PDUs, via setting of the A-bit on a Data-in PDU. The Data_ACK_Notify Operational Primitive with the following input qualifiers is used by the target Datamover layer to notify the local iSCSI layer of the arrival of data acknowledgement of a previously solicited iSCSI read data acknowledgement. This Operational Primitive thus is applicable only to iSCSI targets.

a) Connection_Handle

b) ITT: Initiator Task Tag semantics as defined in [RFC3720]

c) DataSN: of the next SCSI Data-in PDU which immediately follows the SCSI Data-in PDU with the A-bit set to which this notification corresponds, with semantics as defined in [RFC3720].

Every invocation of the Data_ACK_Notify Operational Primitive MUST be preceded by an invocation of the Put_Data Operational Primitive by the iSCSI target layer with the A-bit set to 1 at an earlier point in time.

10.3 Interactions for sending an iSCSI PDU

This section discusses the interactions model for sending each of the iSCSI PDUs defined in [RFC3720]. A Connection_Handle (see section 5.3) is assumed to qualify each of these interactions so that the Datamover layer can route it to the appropriate Transport Connection. The qualifying Connection_Handle is not explicitly listed in the subsequent sections.

Note that the defined list of input qualifiers represents the semantically required set for the Datamover layer to consider in implementing the Primitive in each interaction described in this section (see section 5.4 for an elaboration). Implementations may choose to deduce the qualifiers in ways that are optimized for the implementation specifics. Two examples of this are:

1. For SCSI Command (section 10.3.1), deducing the ImmediateDataSize input qualifier from the DataSegmentLength field of the SCSI Command PDU.

2. For SCSI Data-Out (section 10.3.5.1), deducing the DataDescriptorOut input qualifier from the associated SCSI Command invocation qualifiers (assuming such state is maintained) in conjunction with BHS fields of the SCSI Data-out PDU.

10.3.1 SCSI Command

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a SCSI Command PDU.

a) BHS and AHS, if any, of the SCSI Command PDU as defined in [RFC3720]

b) DataDescriptorOut: that defines the I/O Buffer meant for Data-out for the entire command, in the case of a write or bidirectional command

c) DataDescriptorIn: that defines the I/O Buffer meant for Data-in for the entire command, in the case of a read or bidirectional command

d) ImmediateDataSize: that defines the number of octets of immediate unsolicited data for a write/bidirectional command

e) UnsolicitedDataSize: that defines the number of octets of immediate and non-immediate unsolicited data for a write/bidirectional command.

10.3.2 SCSI Response

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a SCSI Response PDU.

a) BHS of the SCSI Response PDU as defined in [RFC3720]

b) DataDescriptorStatus: that defines the iSCSI buffer which contains the sense and response information for the command

10.3.3 Task Management Function Request

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a Task Management Function Request PDU.

a) BHS of the Task Management Function Request PDU as defined in [RFC3720]

b) DataDescriptorOut: that defines the I/O Buffer meant for Data-out for the entire command, in the case of a write or bidirectional command (Only valid if Function="TASK REASSIGN" – [RFC3720])

c) DataDescriptorIn: that defines the I/O Buffer meant for Data-in for the entire command, in the case of a read or bidirectional command (Only valid if Function="TASK REASSIGN" - [RFC3720])

10.3.4 Task Management Function Response

The Send_Control Operational Primitive with the following input qualifier is used for requesting the transmission of a Task Management Function Response PDU.

a) BHS of the Task Management Function Response PDU as defined in [RFC3720]

10.3.5 SCSI Data-out & SCSI Data-in

10.3.5.1 SCSI Data-out

The Send_Control Operational Primitive with the following input qualifiers is used by the initiator iSCSI layer for requesting the transmission of a SCSI Data-out PDU carrying the non-immediate unsolicited data.

a) BHS of the SCSI Data-out PDU as defined in [RFC3720]

b) DataDescriptorOut: that defines the I/O Buffer with the Data-out to be carried in the iSCSI data segment of the PDU

10.3.5.2 SCSI Data-in

The Put_Data Operational Primitive with the following input qualifiers is used by the target iSCSI layer for requesting the transmission of the data carried by a SCSI Data-in PDU.

a) BHS of the SCSI Data-in PDU as defined in [RFC3720]

b) DataDescriptorIn: that defines the I/O Buffer with the Data-in being requested for transmission

10.3.6 Ready To Transfer (R2T)

The Get_Data Operational Primitive with the following input qualifiers is used by the target iSCSI layer for requesting

the retrieval of the data as specified by the semantic content of an R2T PDU.

a) BHS of the Ready To Transfer PDU as defined in [RFC3720]

b) DataDescriptorOut: that defines the I/O Buffer for the Data-out being requested for retrieval

10.3.7 Asynchronous Message

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of an Asynchronous Message PDU.

a) BHS of the Asynchronous Message PDU as defined in [RFC3720]

b) DataDescriptorSense: that defines an iSCSI buffer which contains the sense and iSCSI Event information.

10.3.8 Text Request

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a Text Request PDU.

a) BHS of the Text Request PDU as defined in [RFC3720]

b) DataDescriptorTextOut: that defines the iSCSI Text Request buffer

10.3.9 Text Response

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a Text Response PDU.

a) BHS of the Text Response PDU as defined in [RFC3720]

b) DataDescriptorTextIn: that defines the iSCSI Text Response buffer

10.3.10 Login Request

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a Login Request PDU.

a) BHS of the Login Request PDU as defined in [RFC3720]

b) DataDescriptorLoginRequest: that defines the iSCSI Login Request buffer

Note that specific Datamover protocols may choose to disallow the standard DA Primitives from being used for the iSCSI Login phase. When used in conjunction with such Datamover protocols, an attempt to send a Login Request via the Send_Control Operational Primitive invocation is clearly an error scenario, as the Login Request PDU is being sent while the connection is in the iSCSI full feature phase. It is outside the scope of this document to specify the resulting implementation behavior in this case - [RFC3720] already defines the error handling for this error scenario.

10.3.11 Login Response

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a Login Response PDU.

a) BHS of the Login Response PDU as defined in [RFC3720]

b) DataDescriptorLoginResponse: that defines the iSCSI Login Response buffer

Note that specific Datamover protocols may choose to disallow the standard DA Primitives from being used for the iSCSI Login phase. When used in conjunction with such Datamover protocols, an attempt to send a Login Response via the Send_Control Operational Primitive invocation is clearly an error scenario, as the Login Response PDU is being sent while in the iSCSI full feature phase. It is outside the scope of this document to specify the resulting implementation behavior in this case - [RFC3720] already defines the error handling for this error scenario.

10.3.12 Logout Command

The Send_Control Operational Primitive with the following input qualifier is used for requesting the transmission of a Logout Command PDU.

a) BHS of the Logout Command PDU as defined in [RFC3720]

10.3.13 Logout Response

The Send_Control Operational Primitive with the following input qualifier is used for requesting the transmission of a Logout Response PDU.

a) BHS of the Logout Response PDU as defined in [RFC3720]

10.3.14 SNACK Request

The Send_Control Operational Primitive with the following input qualifier is used for requesting the transmission of a SNACK Request PDU.

a) BHS of the SNACK Request PDU as defined in [RFC3720]

10.3.15 Reject

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a Reject PDU.

a) BHS of the Reject PDU as defined in [RFC3720]

b) DataDescriptorReject: that defines the iSCSI Reject buffer

10.3.16 NOP-Out

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a NOP-Out PDU.

a) BHS of the NOP-Out PDU as defined in [RFC3720]

b) DataDescriptorNOPOut: that defines the iSCSI Ping data buffer

10.3.17 NOP-In

The Send_Control Operational Primitive with the following input qualifiers is used for requesting the transmission of a NOP-In PDU.

a) BHS of the NOP-In PDU as defined in [RFC3720]

b) DataDescriptorNOPIn: that defines the iSCSI Return Ping data buffer

10.4 Interactions for receiving an iSCSI PDU

The only PDUs that are received by an iSCSI layer operating on a Datamover layer are the iSCSI control-type PDUs. The

Datamover layer delivers the iSCSI control-type PDUs as they arrive, qualifying each with the Connection_Handle (see section 5.3) that identifies the iSCSI connection the PDU is meant for. The subsequent processing of the iSCSI control-type PDUs proceeds as defined in [RFC3720].

10.4.1 General Control-type PDU notification

This sub-section describes the general mechanics applicable to several control-type PDUs. The following sub-sections note additional considerations for control-type PDUs not covered in this sub-section.

The Control_Notify Operational Primitive is used for notifying the arrival of the following iSCSI control-type PDUs: SCSI Command, SCSI Response, Task Management Function Request, Task Management Function Response, Asynchronous Message, Text Request, Text Response, Logout command, Logout Response, SNACK, Reject, NOP-Out, NOP-In.

10.4.2 SCSI Data Transfer PDUs

10.4.2.1 SCSI Data-out

The Control_Notify Operational Primitive is used for notifying the iSCSI layer of the arrival of a SCSI Data-out PDU carrying the non-immediate unsolicited data. Note however that the solicited SCSI Data-out arriving on the target is not notified to the iSCSI layer using the Control_Notify Primitive because the solicited SCSI Data-out was not sent by the initiator iSCSI layer as control-type PDUs.

10.4.2.2 SCSI Data-in

The arrival of the SCSI Data-in is not notified to the iSCSI layer by the Datamover layer at the initiator, because SCSI Data-in is an iSCSI data-type PDU (see section 5.1). The iSCSI layer at the initiator however may infer the arrival of the SCSI Data-in when it receives a subsequent notification of the SCSI Response PDU via a Control_Notify invocation.

While this document does not contemplate the possibility of a Data-in PDU being received at the initiator iSCSI layer, specific Datamover protocols may define how to deal with an unexpected inbound SCSI Data-in PDU that may result in the initiator iSCSI layer receiving the Data-in PDU. This document leaves the details of handling this error scenario to the specific Datamover protocols, so each may define the appropriate error handling specific to the Datamover environment.

10.4.2.3 Ready To Transfer (R2T)

Because an R2T PDU is an iSCSI data-type PDU (see section 5.1) that is not delivered as-is to the initiator iSCSI layer, the arrival of an R2T PDU is not notified to the iSCSI layer by the Datamover layer. When an iSCSI node sends an R2T PDU to its local Datamover layer, the local and remote Datamover layers transparently bring about the data transfer requested by the R2T PDU.

While this document does not contemplate the possibility of an R2T PDU being received at the initiator iSCSI layer, specific Datamover protocols may define how to deal with an unexpected inbound R2T PDU that may result in the initiator iSCSI layer receiving the R2T PDU. This document leaves the details of handling this error scenario to the specific Datamover protocols, so each may define the appropriate error handling specific to the Datamover environment.

10.4.3 Login Request

The Control_Notify Operational Primitive is used for notifying the target iSCSI layer of the arrival of a Login Request PDU. Note that specific Datamover protocols may choose to disallow the standard DA Primitives from being used for the iSCSI Login phase. When used in conjunction with such Datamover protocols, the arrival of a Login Request necessitating the Control_Notify Operational Primitive invocation is clearly an error scenario, as the Login Request PDU is arriving in the iSCSI full feature phase. It is outside the scope of this document to specify the resulting

implementation behavior in this case - [RFC3720] already defines the error handling in this error scenario.

10.4.4 Login Response

The Control_Notify Operational Primitive is used for notifying the initiator iSCSI layer of the arrival of a Login Response PDU. Note that specific Datamover protocols may choose to disallow the standard DA Primitives from being used for the iSCSI Login phase. When used in conjunction with such Datamover protocols, the arrival of a Login Response necessitating the Control_Notify Operational Primitive invocation is clearly an error scenario, as the Login Response PDU is arriving in the iSCSI full feature phase. It is outside the scope of this document to specify the resulting implementation behavior in this case - [RFC3720] already defines the error handling in this error scenario.

11 Security Considerations

11.1 Architectural Considerations

DA enables compliant iSCSI implementations to realize a control and data separation in the way they interact with their Datamover protocols. Note however that this separation does not imply a separation in transport mediums between control traffic and data traffic - basic iSCSI architecture with respect to tasks and PDU relationships to tasks remains unchanged. [RFC3720] defines several MUST requirements on ordering relationships across control and data for a given task besides a mandatory deterministic task allegiance model - DA does not change this basic architecture (DA has a normative reference on [RFC3720]) nor allow any additional flexibility in compliance in this area. To summarize, sending bulk data transfers (prompted by Put_Data and Get_Data Primitive invocations) on a different transport medium would be as ill-advised as sending just the Data-out/Data-in PDUs on a different TCP connection in RFC 3720-based iSCSI implementations. Consequently, all the iSCSI-related security text in [RFC3723] is directly applicable to a DA-enabled iSCSI implementation.

Another area with security implications is the Datamover connection resource management model which DA defines – particularly the Allocate_Connection_Resources Primitive. An inadvertent realization of this model could leave an iSCSI implementation exposed to denial of service attacks. As Figure 2 and Figure 3 in section 16.2 illustrate, the most effective countermeasure to this potential attack consists of performing the Datamover resource allocation when the iSCSI layer is sufficiently far along in the iSCSI Login Phase that it is reasonably certain that the peer side is not an attacker. In particular, if the Login Phase includes a SecurityNegotiation stage, an iSCSI end node MUST defer the Datamover connection resource allocation (i.e. invoking the Allocate_Connection_Resources Primitive) to the LoginOperationalNegotiation stage ([RFC3720]) so that the resource allocation happens post-authentication. This considerably minimizes the potential for a denial of service attack.

11.2 Wire Protocol Considerations

In view of the fact that the DA architecture itself does not define any new wire protocol nor propose modifications to the existing protocols, there are no additional wire protocol security considerations in employing DA itself. However, a DA-compliant iSCSI implementation MUST comply with all the iSCSI-related requirements stipulated in [RFC3723] and [RFC3720]. Note further that in realizing DA, each Datamover protocol must define and elaborate as appropriate on any additional security considerations resulting from the use of that Datamover protocol.

All Datamover protocol designers are strongly recommended to refer to [RDDPSEC] for the types of security issues to consider. While [RDDPSEC] elaborates on the security considerations applicable to an RDDP-based Datamover ([iSER]), the document is representative of the type of analysis of resource exhaustion and the application of countermeasures that needs to be done for any Datamover protocol.

12 IANA Considerations

DA *architecture* does not have any IANA considerations.

13 References and Bibliography

13.1 Normative References

[RFC3720] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)", RFC 3720, April 2004.

[RFC3723] B. Aboba, J. Tseng, J. Walker, V. Rangan, F. Travostino, "Securing Block Storage Protocols over IP", RFC 3723, April 2004.

[RFC2119] S. Bradner, “Key words for use in RFCs to Indicate Requirement Levels”, March 1997.

13.2 Informative References

[DDP] H. Shah et al., "Direct Data Placement over Reliable Transports", IETF Internet Draft draft-ietf-rddp-ddp-06.txt (work in progress), June 2006.

[iSER] M. Ko et al., “iSCSI Extensions for RDMA”, IETF Internet Draft draft-ietf-ips-iser-03.txt (work in progress), April 2005.

[RDDPSEC] J. Pinkerton et al., "DDP/RDMAP Security", IETF Internet Draft draft-ietf-rddp-security-07.txt (work in progress), April 2005

14 Authors' Addresses

Mallikarjun Chadalapaka
Hewlett-Packard Company
8000 Foothills Blvd.
Roseville, CA 95747-5668, USA
Phone: +1-916-785-5621
E-mail: cbm@rose.hp.com

John L. Hufferd
IBM
San Jose CA, USA
Phone: +1-408-256-0403
E-mail: hufferd@us.ibm.com

Julian Satran
IBM, Haifa Research Lab
Haifa University Campus - Mount Carmel
Haifa 31905, Israel
Phone +972-4-829-6264
E-mail: Julian_Satran@il.ibm.com

Hemal Shah
Intel Corporation
MS PTL1
1501 South Mopac Expressway, #400
Austin, TX 78746 USA
Phone: +1 (512) 732-3963
Email: hemal.shah@intel.com

Comments may be sent to Mallikarjun Chadalapaka.

15 Acknowledgements

The IP Storage (ips) Working Group in the Transport Area of IETF has been responsible for defining the iSCSI protocol (apart from a host of other relevant IP Storage protocols). The authors are grateful to the entire working group, whose work allowed this document to build on the concepts and details of the iSCSI protocol.

In addition, the following individuals had reviewed and contributed to the improvement of this document. The authors are grateful for their contribution.

John Carrier
Adaptec, Inc.
691 S. Milpitas Blvd., Milpitas, CA 95035 USA
Phone: +1 (360) 378-8526
Email: john_carrier@adaptec.com

Hari Ghadia
Adaptec, Inc.
691 S. Milpitas Blvd., Milpitas, CA 95035 USA
Phone: +1 (408) 957-5608
Email: hari_ghadia@adaptec.com

Hari Mudaliar
Adaptec, Inc.
691 S. Milpitas Blvd., Milpitas, CA 95035 USA
Phone: +1 (408) 957-6012
Email: hari_mudaliar@adaptec.com

Patricia Thaler
Agilent Technologies, Inc.
1101 Creekside Ridge Drive, #100, M/S-RG10,
Roseville, CA 95678
Phone: +1-916-788-5662
email: pat_thaler@agilent.com

Uri Elzur
Broadcom Corporation

16215 Alton Parkway, Irvine, CA 92619-7013 USA
Phone: +1 (949) 585-6432
Email: Uri@Broadcom.com

Mike Penna
Broadcom Corporation
16215 Alton Parkway,Irvine, CA 92619-7013 USA
Phone: +1 (949) 926-7149
Email: MPenna@Broadcom.com

David Black
EMC Corporation
176 South St., Hopkinton, MA 01748, USA
Phone: +1 (508) 293-7953
Email: black_david@emc.com

Ted Compton
EMC Corporation
Research Triangle Park, NC 27709, USA
Phone: +1-919-248-6075
Email: compton_ted@emc.com

Dwight Barron
Hewlett-Packard Company
20555 SH 249, Houston, TX 77070-2698 USA
Phone: +1 (281) 514-2769
Email: Dwight.Barron@Hp.com

Paul R. Culley
Hewlett-Packard Company
20555 SH 249, Houston, TX 77070-2698 USA
Phone: +1 (281) 514-5543
Email: paul.culley@hp.com

Dave Garcia
Hewlett-Packard Company
19333 Vallco Parkway, Cupertino, Ca. 95014 USA
Phone: +1 (408) 285-6116
Email: dave.garcia@hp.com

Randy Haagens
Hewlett-Packard Company
8000 Foothills Blvd, MS 5668, Roseville CA
Phone: +1-916-785-4578
email: randy_haagens@hp.com

Jeff Hilland
Hewlett-Packard Company
20555 SH 249, Houston, Tx. 77070-2698 USA
Phone: +1 (281) 514-9489
Email: jeff.hilland@hp.com

Mike Krause
Hewlett-Packard Company, 43LN
19410 Homestead Road, Cupertino, CA 95014 USA
Phone: +1 (408) 447-3191
Email: krause@cup.hp.com

Jim Wendt
Hewlett-Packard Company
8000 Foothills Blvd, MS 5668, Roseville CA
Phone: +1-916-785-5198
email: jim_wendt@hp.com

Mike Ko
IBM
650 Harry Rd, San Jose, CA 95120
Phone: +1 (408) 927-2085
Email: mako@us.ibm.com

Renato Recio
IBM Corporation
11501 Burnett Road, Austin, TX 78758 USA
Phone: +1 (512) 838-1365
Email: recio@us.ibm.com

Howard C. Herbert
Intel Corporation
MS CH7-404,5000 West Chandler Blvd., Chandler, AZ 85226 USA
Phone: +1 (480) 554-3116
Email: howard.c.herbert@intel.com

Dave Minturn
Intel Corporation
MS JF1-210, 5200 North East Elam Young Parkway
Hillsboro, OR 97124 USA
Phone: +1 (503) 712-4106
Email: dave.b.minturn@intel.com

James Pinkerton
Microsoft Corporation
One Microsoft Way, Redmond, WA 98052 USA
Phone: +1 (425) 705-5442
Email: jpink@microsoft.com

Tom Talpey
Network Appliance
375 Totten Pond Road, Waltham, MA 02451 USA
Phone: +1 (781) 768-5329
EMail: thomas.talpey@netapp.com

16 Appendix

16.1 Design considerations for a Datamover protocol

This section discusses the specific considerations for RDMA-based and RDDP-based Datamover protocols.

a) Note that the modeling of interactions for SCSI Data-Out (section 10.3.5.1) is only used for unsolicited data transfer.

b) The modeling of interactions for SNACK (section 10.3.14, and section 10.4.1) is not expected to be used given that one of the design requirements on the Datamover is that it "guarantees an error-free, reliable, in-order transport mechanism" (section 6). The interactions for sending and receiving a SNACK are nevertheless modeled in this document because the receiving iSCSI layer can deterministically deal with an inadvertent SNACK. This also shows the DA designers' intent that DI is not meant to filter certain types of PDUs.

c) The onus is on a reliable Datamover (per requirements stated in section 6) to realize end-to-end data acknowledgements via Datamover-specific means. In view of this, even data-ACK-type SNACKs are unnecessary to be used. Consequently, an initiator may never request sending a SNACK Request in this model assuming that the proactive (timeout-driven) SNACK functionality is turned off in the legacy iSCSI code.

d) Note that the current DA model for bootstrapping a Connection_Handle into service – i.e. associating a new iSCSI connection with a Connection_Handle – clearly implies that the iSCSI connection must already be in full feature phase when the Datamover layer comes into the stack. This further implies that the iSCSI login phase must be carried out in the traditional "Byte streaming mode" with no assistance or involvement from the Datamover layer.

16.2 Examples of Datamover interactions

The figures described in this section provide some examples of the usage of Operational Primitives in interactions between the iSCSI layer and the Datamover layer. The following abbreviations are used in this section.

Avail – Available

Abted - Aborted

Buf – I/O Buffer

Cmd – Command

Compl – Complete

Conn - Connection

Ctrl_Ntfy – Control_Notify

Dal_Tk_Res – Deallocate_Task_Resources

Data_Cmp_Nfy – Data_Completion_Notify

Data_ACK_Nfy – Data_ACK_Notify

DM – Datamover

Imm - Immediate

Snd_Ctrl – Send_Control

Msg – Message

Resp – Response

Sol – Solicited

TMF Req – Task Management Function Request

TMF Res – Task Management Function Response

Trans – Transfer

Unsol – Unsolicited

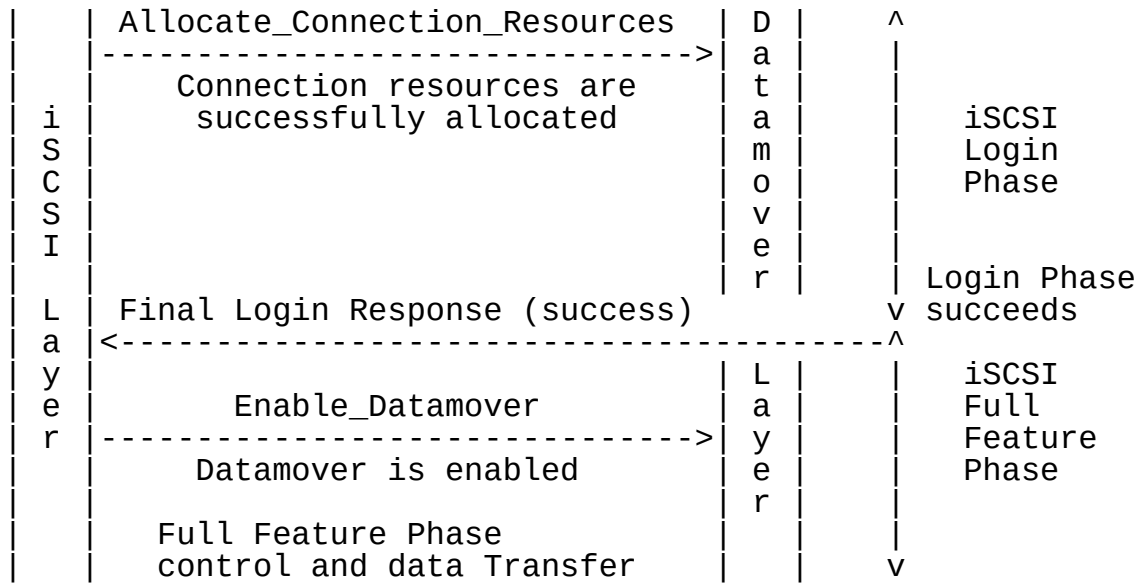

Figure 2 A successful iSCSI login on initiator

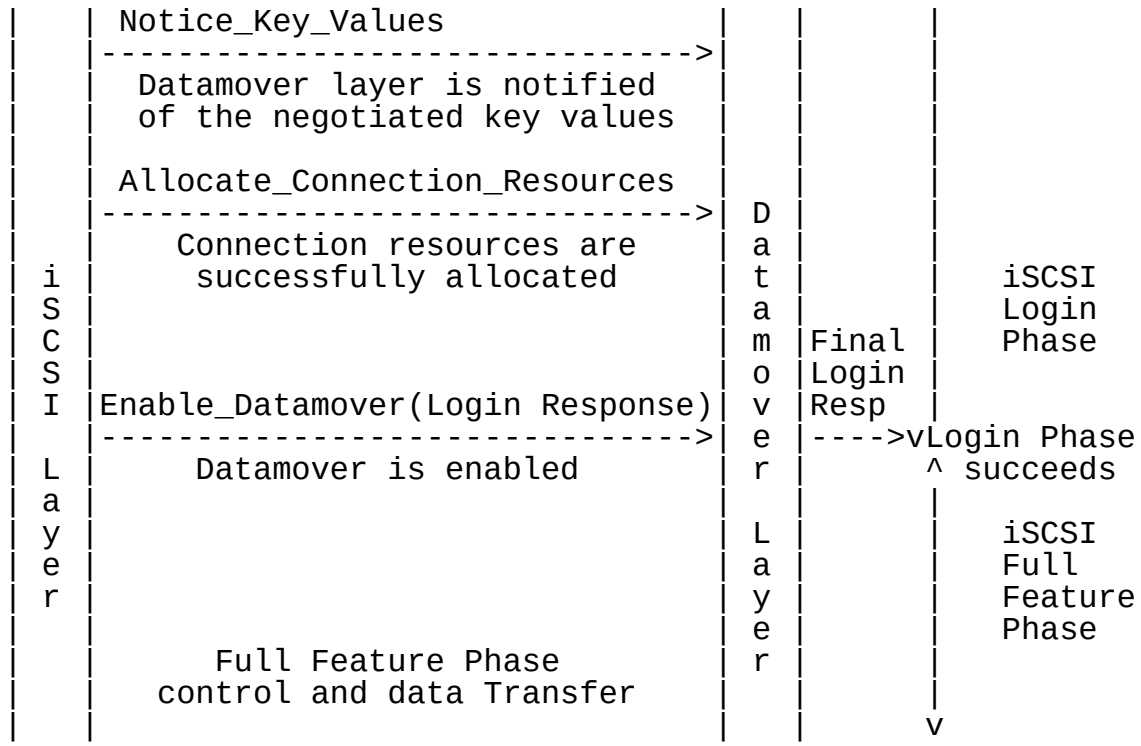

Figure 3 A successful iSCSI login on target

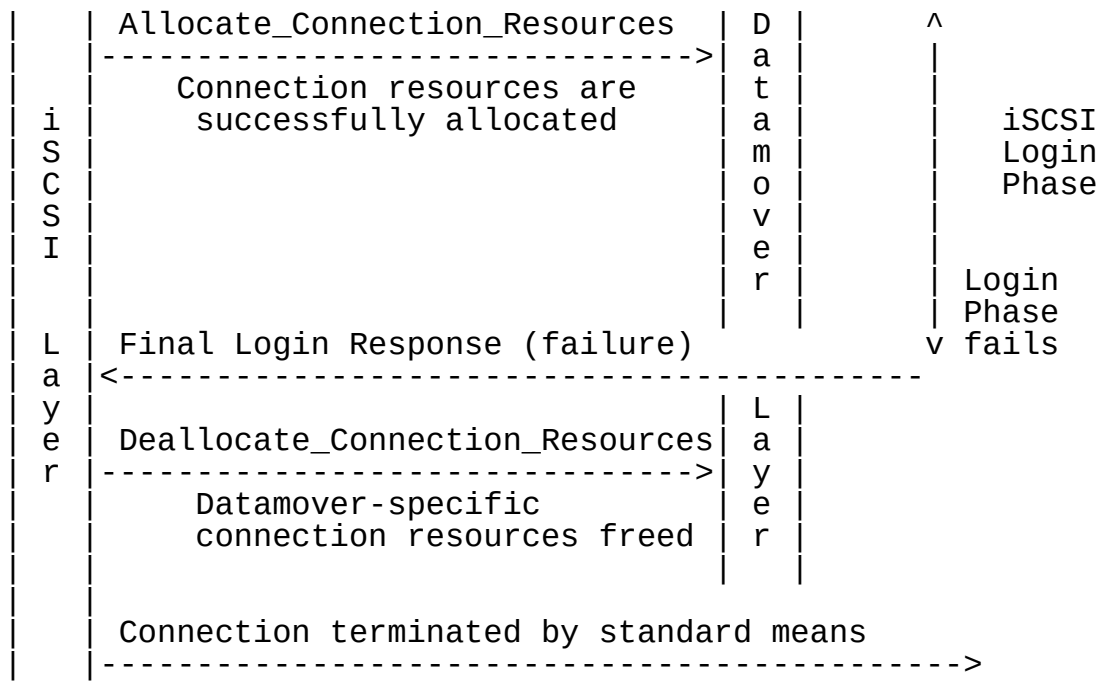

Figure 4 A failed iSCSI login on initiator

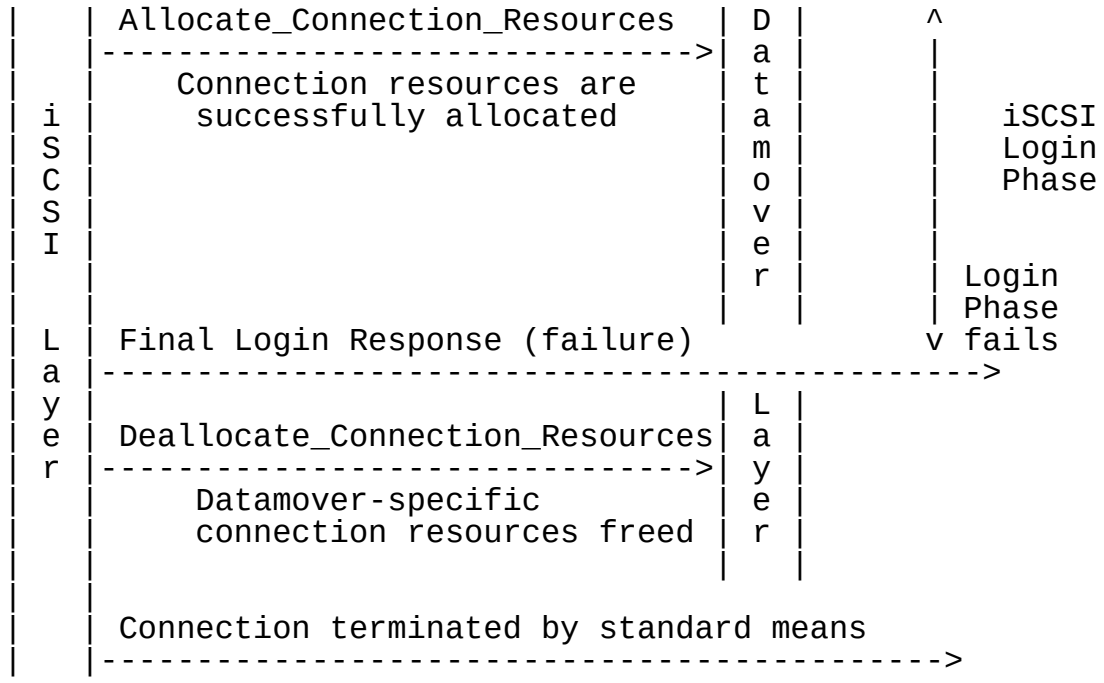

Figure 5 A failed iSCSI login on target

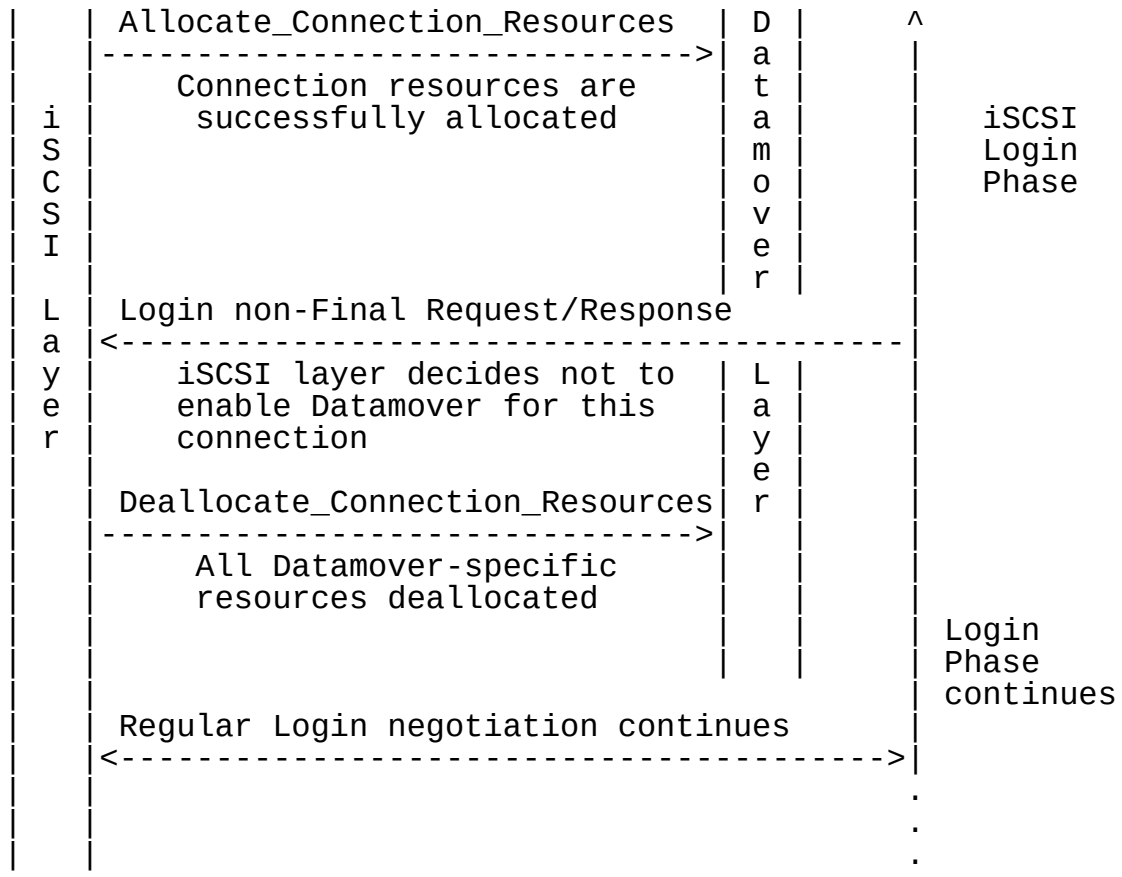

Figure 6 iSCSI does not enable the Datamover

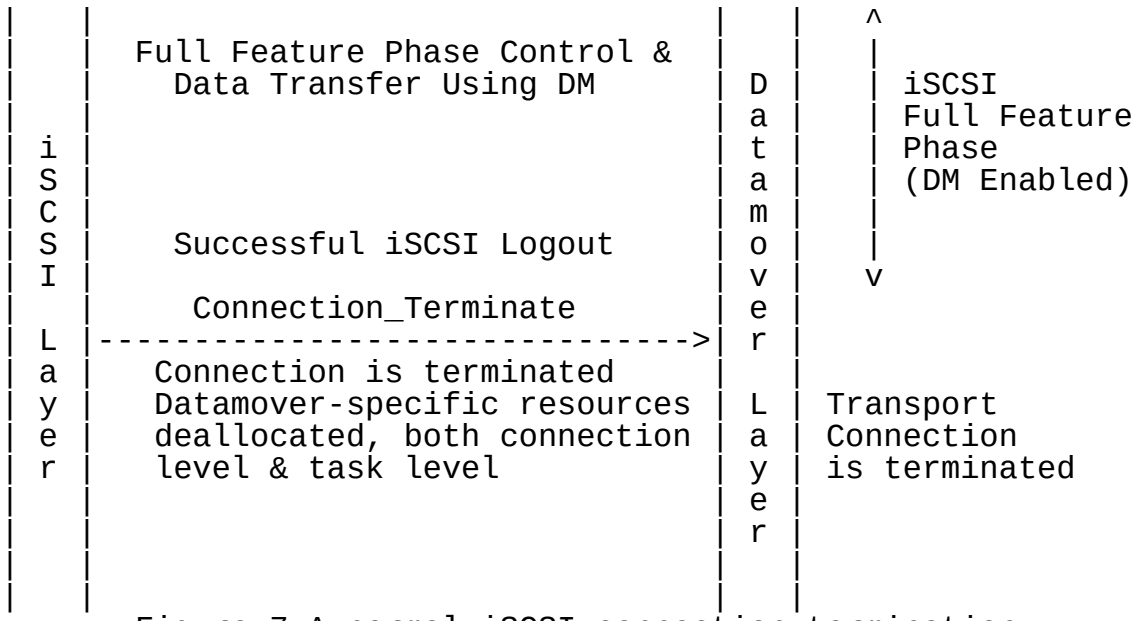

Figure 7 A normal iSCSI connection termination

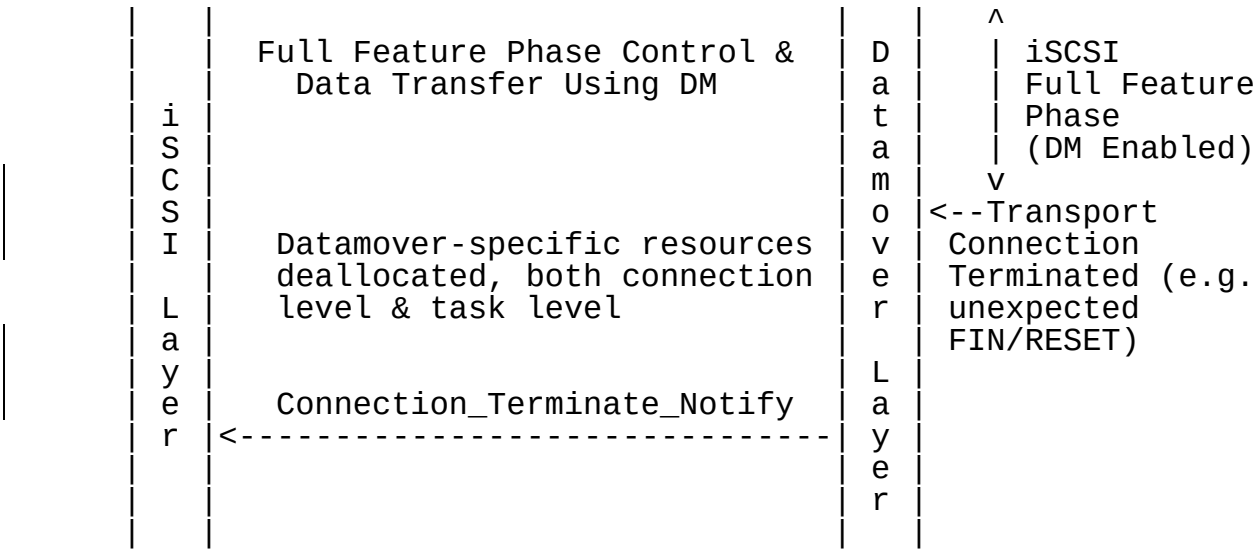

Figure 8 An abnormal iSCSI connection termination

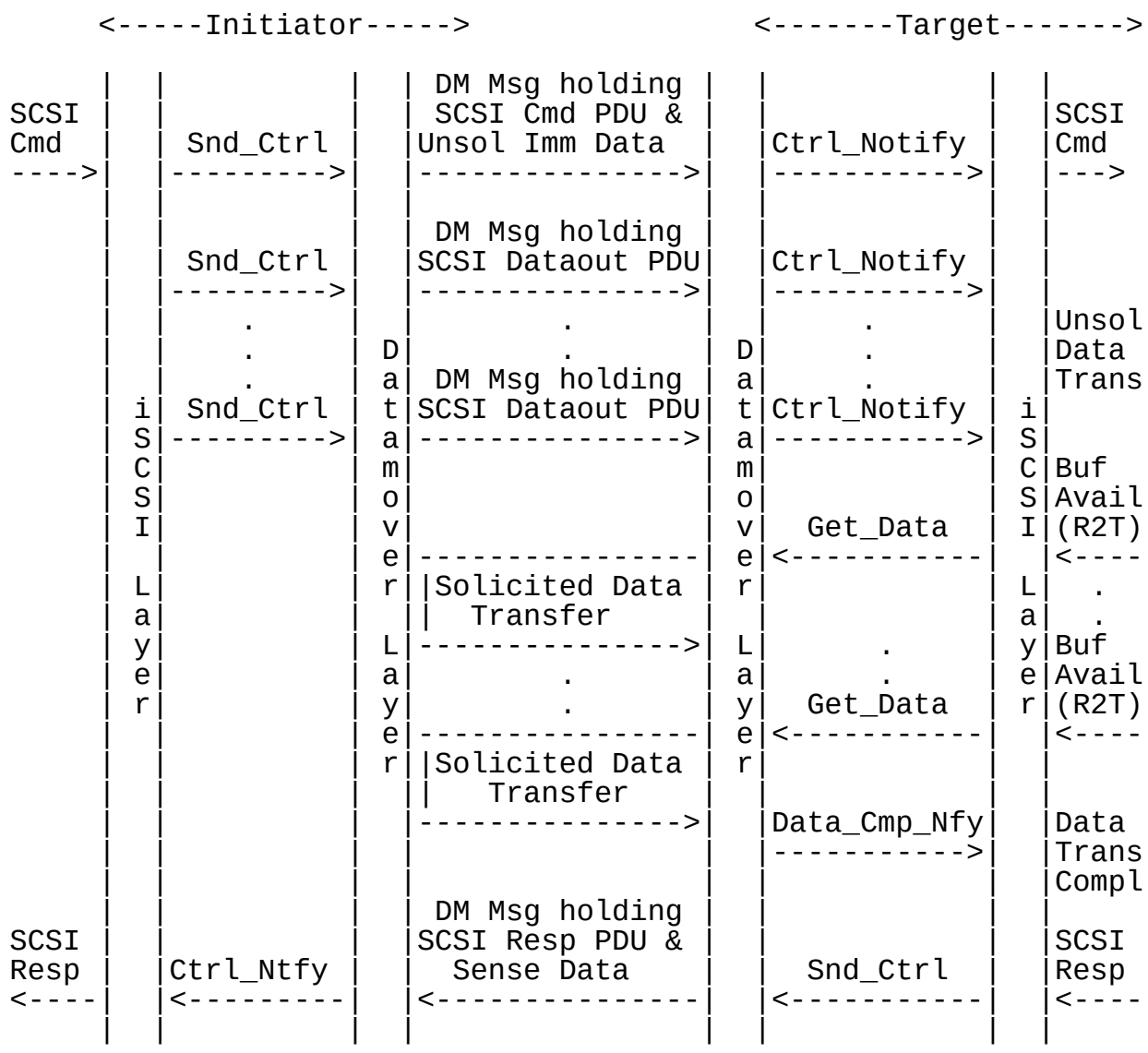

Figure 9 A SCSI Write data transfer

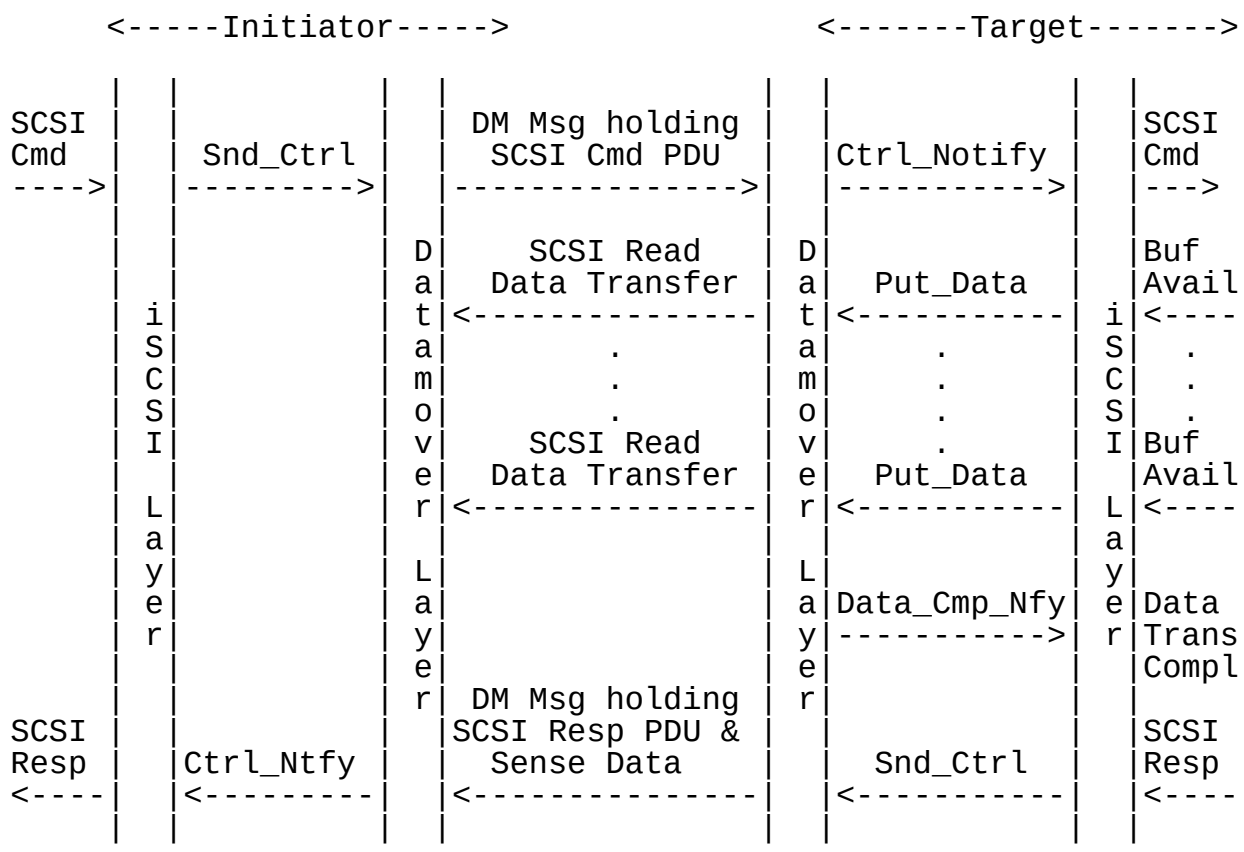

Figure 10 A SCSI Read data transfer

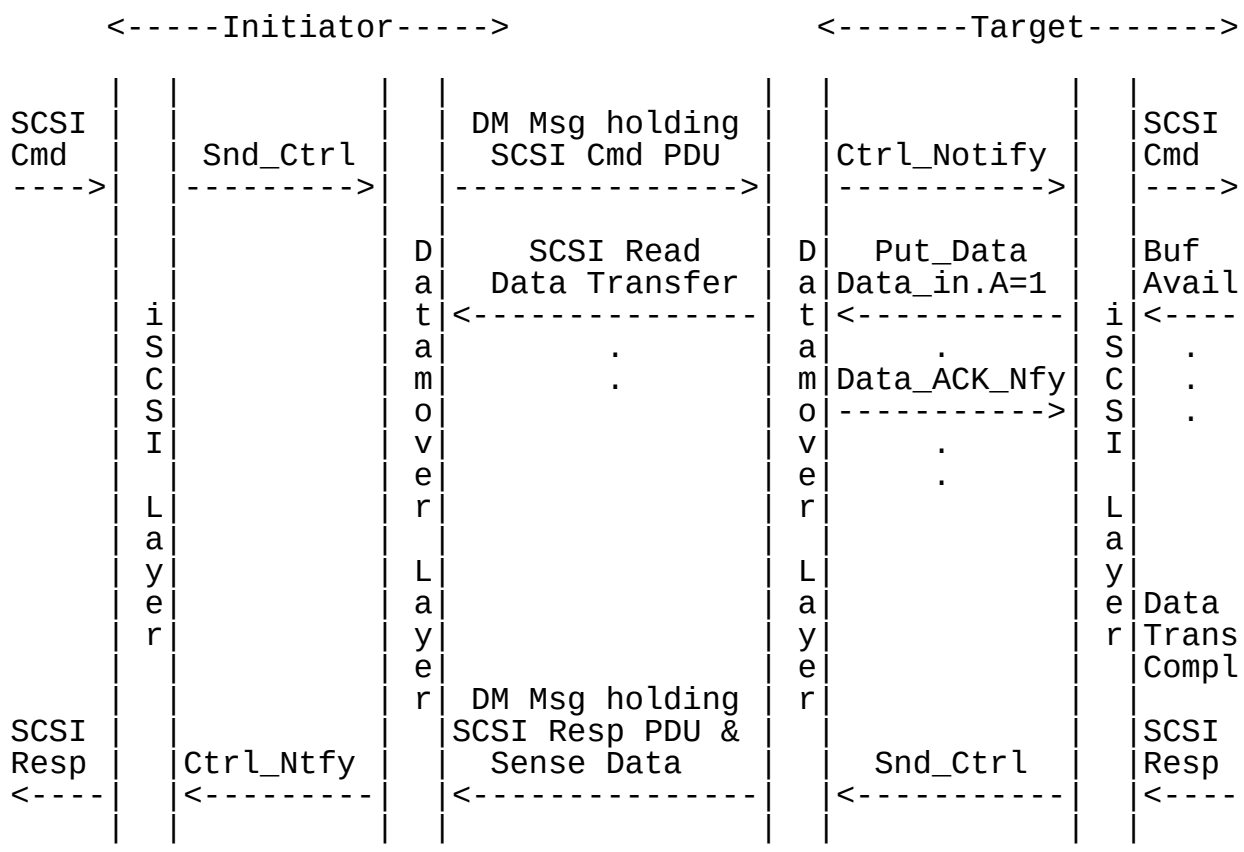

Figure 11 A SCSI Read data acknowledgement

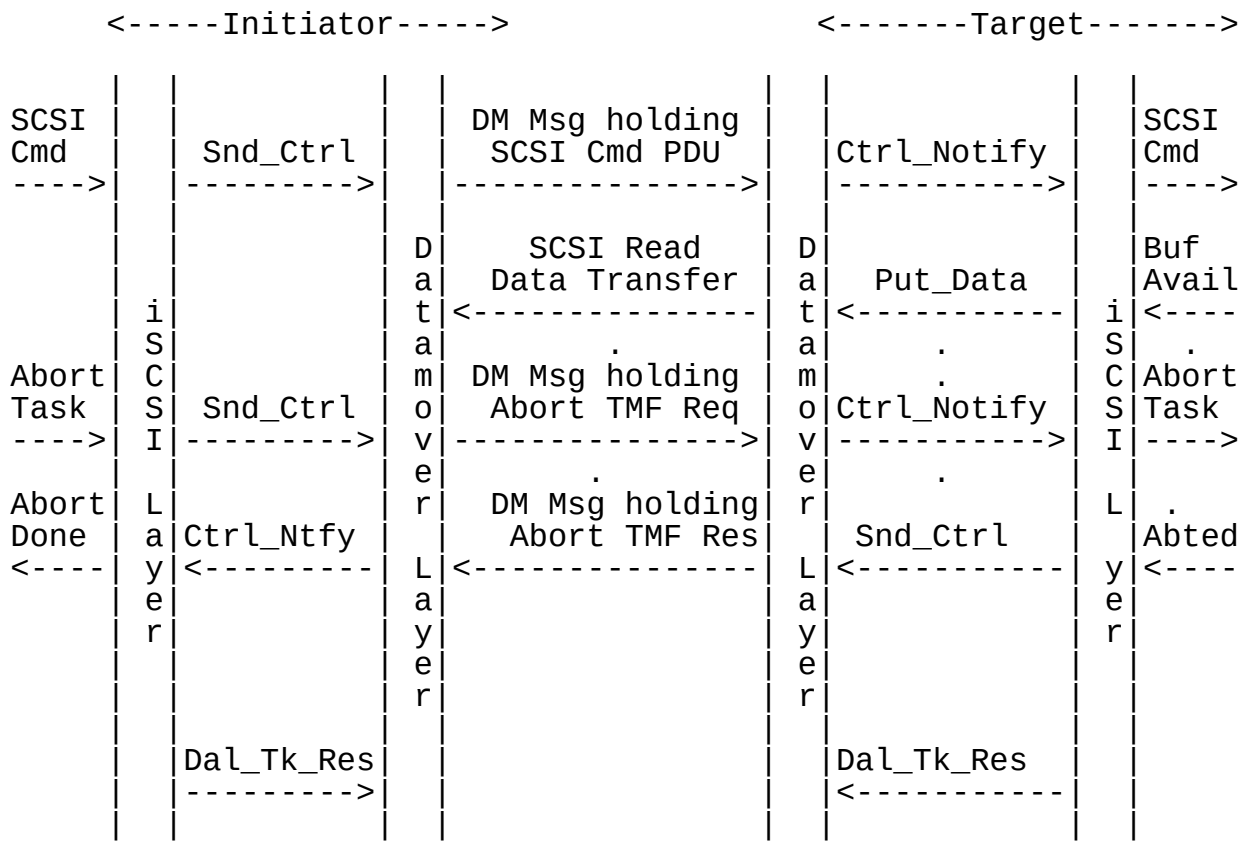

Figure 12 Task resource cleanup on abort

17 Full Copyright Statement

18 Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.