Network Working Group
Request for Comments: 3720
Category: Standards Track

J. Satran
K. Meth
IBM
C. Sapuntzakis
Cisco Systems
M. Chadalapaka
Hewlett-Packard Co.
E. Zeidner
IBM

April 2004

Internet Small Computer Systems Interface (iSCSI)

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

Abstract

This document describes a transport protocol for Internet Small Computer Systems Interface (iSCSI) that works on top of TCP. The iSCSI protocol aims to be fully compliant with the standardized SCSI architecture model.

SCSI is a popular family of protocols that enable systems to communicate with I/O devices, especially storage devices. SCSI protocols are request/response application protocols with a common standardized architecture model and basic command set, as well as standardized command sets for different device classes (disks, tapes, media-changers etc.).

As system interconnects move from the classical bus structure to a network structure, SCSI has to be mapped to network transport protocols. IP networks now meet the performance requirements of fast system interconnects and as such are good candidates to "carry" SCSI.

Docker EX1014
Page 1 of 257

Table of Contents

1. Introduction. 9
2. Definitions and Acronyms. 10
   2.1. Definitions. 10
   2.2. Acronyms 14
   2.3. Conventions. 16
      2.3.1. Word Rule. 16
      2.3.2. Half-Word Rule 17
      2.3.3. Byte Rule. 17
3. Overview. 17
   3.1. SCSI Concepts. 17
   3.2. iSCSI Concepts and Functional Overview 18
      3.2.1. Layers and Sessions. 19
      3.2.2. Ordering and iSCSI Numbering 19
         3.2.2.1. Command Numbering and Acknowledging 20
         3.2.2.2. Response/Status Numbering and Acknowledging 23
         3.2.2.3. Data Sequencing 24
      3.2.3. iSCSI Login. 24
      3.2.4. iSCSI Full Feature Phase 25
         3.2.4.1. Command Connection Allegiance . . 26
         3.2.4.2. Data Transfer Overview. 27
         3.2.4.3. Tags and Integrity Checks 28
         3.2.4.4. Task Management 28
      3.2.5. iSCSI Connection Termination 29
      3.2.6. iSCSI Names. 29
         3.2.6.1. iSCSI Name Properties 30
         3.2.6.2. iSCSI Name Encoding 31
         3.2.6.3. iSCSI Name Structure. 32
            3.2.6.3.1. Type "iqn." (iSCSI Qualified Name) . . . 32
            3.2.6.3.2. Type "eui." (IEEE EUI-64 format). . . . 34
      3.2.7. Persistent State 34
      3.2.8. Message Synchronization and Steering 35
         3.2.8.1. Sync/Steering and iSCSI PDU Length 36
   3.3. iSCSI Session Types. 36
   3.4. SCSI to iSCSI Concepts Mapping Model 37
      3.4.1. iSCSI Architecture Model 37
      3.4.2. SCSI Architecture Model. 39
      3.4.3. Consequences of the Model. 41
         3.4.3.1. I_T Nexus State 42
   3.5. Request/Response Summary 42
      3.5.1. Request/Response Types Carrying SCSI Payload 43
         3.5.1.1. SCSI-Command 43

Docker EX1014
Page 2 of 257

3.5.1.2. SCSI-Response 43
3.5.1.3. Task Management Function Request. 44
3.5.1.4. Task Management Function Response 44
3.5.1.5. SCSI Data-Out and SCSI Data-In. . 44
3.5.1.6. Ready To Transfer (R2T) 45
3.5.2. Requests/Responses carrying SCSI and iSCSI Payload. 46
3.5.2.1. Asynchronous Message. 46
3.5.3. Requests/Responses Carrying iSCSI Only Payload. 46
3.5.3.1. Text Request and Text Response. . 46
3.5.3.2. Login Request and Login Response. 47
3.5.3.3. Logout Request and Response . . . 47
3.5.3.4. SNACK Request 48
3.5.3.5. Reject. 48
3.5.3.6. NOP-Out Request and NOP-In Response 48
4. SCSI Mode Parameters for iSCSI. 48
5. Login and Full Feature Phase Negotiation. 48
5.1. Text Format. 50
5.2. Text Mode Negotiation. 53
5.2.1. List negotiations. 56
5.2.2. Simple-value Negotiations. 56
5.3. Login Phase. 57
5.3.1. Login Phase Start. 60
5.3.2. iSCSI Security Negotiation 62
5.3.3. Operational Parameter Negotiation During the Login Phase. 63
5.3.4. Connection Reinstatement 64
5.3.5. Session Reinstatement, Closure, and Timeout. 64
5 5.3.5.1. Loss of Nexus Notification. 65
5.3.6. Session Continuation and Failure 65
5.4. Operational Parameter Negotiation Outside the Login Phase. 66
6. iSCSI Error Handling and Recovery 67
6.1. Overview 67
6.1.1. Background 67
6.1.2. Goals. 67
6.1.3. Protocol Features and State Expectations . . 68
6.1.4. Recovery Classes 69
6.1.4.1. Recovery Within-command 69
6.1.4.2. Recovery Within-connection. . . . 70
6.1.4.3. Connection Recovery 71
6.1.4.4. Session Recovery. 72
6.1.5. Error Recovery Hierarchy 72
6.2. Retry and Reassign in Recovery 74
6.2.1. Usage of Retry 74

Docker EX1014
Page 3 of 257

6.2.2. Allegiance Reassignment. 75
6.3. Usage Of Reject PDU in Recovery. 76
6.4. Connection Timeout Management. 76
6.4.1. Timeouts on Transport Exception Events . . . 77
6.4.2. Timeouts on Planned Decommissioning. 77
6.5. Implicit Termination of Tasks. 77
6.6. Format Errors. 78
6.7. Digest Errors. 78
6.8. Sequence Errors. 80
6.9. SCSI Timeouts. 81
6.10. Negotiation Failures 81
6.11. Protocol Errors. 82
6.12. Connection Failures. 82
6.13. Session Errors 83
7. State Transitions 84
7.1. Standard Connection State Diagrams 84
7.1.1. State Descriptions for Initiators and Targets. 84
7.1.2. State Transition Descriptions for Initiators and Targets. 85
7.1.3. Standard Connection State Diagram for an Initiator. 88
7.1.4. Standard Connection State Diagram for a Target 90
7.2. Connection Cleanup State Diagram for Initiators and Targets. 92
7.2.1. State Descriptions for Initiators and Targets. 94
7.2.2. State Transition Descriptions for Initiators and Targets. 94
7.3. Session State Diagrams 95
7.3.1. Session State Diagram for an Initiator . . . 95
7.3.2. Session State Diagram for a Target 96
7.3.3. State Descriptions for Initiators and Targets. 97
7.3.4. State Transition Descriptions for Initiators and Targets. 98
8. Security Considerations 99
8.1. iSCSI Security Mechanisms. 100
8.2. In-band Initiator-Target Authentication. 100
8.2.1. CHAP Considerations. 101
8.2.2. SRP Considerations 103
8.3. IPsec. 104
8.3.1. Data Integrity and Authentication. 104
8.3.2. Confidentiality. 105
8.3.3. Policy, Security Associations, and Cryptographic Key Management 105
9. Notes to Implementers 106

Docker EX1014
Page 4 of 257

9.1. Multiple Network Adapters. 106
9.1.1. Conservative Reuse of ISIDs. 107
9.1.2. iSCSI Name, ISID, and TPGT Use 107
9.2. Autosense and Auto Contingent Allegiance (ACA) 109
9.3. iSCSI Timeouts 109
9.4. Command Retry and Cleaning Old Command Instances . . . 110
9.5. Synch and Steering Layer and Performance 110
9.6. Considerations for State-dependent Devices and Long-lasting SCSI Operations 111
9.6.1. Determining the Proper ErrorRecoveryLevel. . 112
10. iSCSI PDU Formats 112
10.1. iSCSI PDU Length and Padding 113
10.2. PDU Template, Header, and Opcodes. 113
10.2.1. Basic Header Segment (BHS) 114
10.2.1.1. I 115
10.2.1.2. Opcode. 115
10.2.1.3. Final (F) bit 116
10.2.1.4. Opcode-specific Fields. 116
10.2.1.5. TotalAHSLength. 116
10.2.1.6. DataSegmentLength 116
10.2.1.7. LUN 116
10.2.1.8. Initiator Task Tag. 117
10.2.2. Additional Header Segment (AHS) 117
10.2.2.1. AHSType 117
10.2.2.2. AHSLength 117
10.2.2.3. Extended CDB AHS. 118
10.2.2.4. Bidirectional Expected Read-Data Length AHS. 118
10.2.3. Header Digest and Data Digest. 118
10.2.4. Data Segment 119
10.3. SCSI Command 119
10.3.1. Flags and Task Attributes (byte 1) 120
10.3.2. CmdSN - Command Sequence Number. 120
10.3.3. ExpStatSN. 120
10.3.4. Expected Data Transfer Length. 121
10.3.5. CDB - SCSI Command Descriptor Block. 121
10.3.6. Data Segment - Command Data. 121
10.4. SCSI Response. 122
10.4.1. Flags (byte 1) 123
10.4.2. Status 123
10.4.3. Response 124
10.4.4. SNACK Tag. 125
10.4.5. Residual Count 125
10.4.6. Bidirectional Read Residual Count. 125
10.4.7. Data Segment - Sense and Response Data Segment. 125
10.4.7.1. SenseLength 126
10.4.7.2. Sense Data. 126

Docker EX1014
Page 5 of 257

10.4.8. ExpDataSN. 127
10.4.9. StatSN - Status Sequence Number. 127
10.4.10. ExpCmdSN - Next Expected CmdSN from this Initiator. 128
10.4.11. MaxCmdSN - Maximum CmdSN from this Initiator 128
10.5. Task Management Function Request 129
10.5.1. Function 129
10.5.2. TotalAHSLength and DataSegmentLength 132
10.5.3. LUN. 132
10.5.4. Referenced Task Tag. 132
10.5.5. RefCmdSN 132
10.5.6. ExpDataSN. 133
10.6. Task Management Function Response. 134
10.6.1. Response 134
10.6.2. Task Management Actions on Task Sets 136
10.6.3. TotalAHSLength and DataSegmentLength 137
10.7. SCSI Data-Out & SCSI Data-In 137
10.7.1. F (Final) Bit. 139
10.7.2. A (Acknowledge) Bit. 139
10.7.3. Flags (byte 1) 140
10.7.4. Target Transfer Tag and LUN. 140
10.7.5. DataSN 141
10.7.6. Buffer Offset. 141
10.7.7. DataSegmentLength. 141
10.8. Ready To Transfer (R2T). 142
10.8.1. TotalAHSLength and DataSegmentLength 143
10.8.2. R2TSN. 143
10.8.3. StatSN 144
10.8.4. Desired Data Transfer Length and Buffer Offset 144
10.8.5. Target Transfer Tag. 144
10.9. Asynchronous Message 145
10.9.1. AsyncEvent 146
10.9.2. AsyncVCode 147
10.9.3. LUN. 147
10.9.4. Sense Data and iSCSI Event Data. 148
10.9.4.1. SenseLength 148
10.10. Text Request 149
10.10.1. F (Final) Bit. 150
10.10.2. C (Continue) Bit 150
10.10.3. Initiator Task Tag 150
10.10.4. Target Transfer Tag. 150
10.10.5. Text 151
10.11. Text Response. 152
10.11.1. F (Final) Bit. 152
10.11.2. C (Continue) Bit 153
10.11.3. Initiator Task Tag 153
10.11.4. Target Transfer Tag. 153

Docker EX1014
Page 6 of 257

10.11.5. StatSN 154
10.11.6. Text Response Data 154
10.12. Login Request. 154
10.12.1. T (Transit) Bit. 155
10.12.2. C (Continue) Bit 155
10.12.3. CSG and NSG. 156
10.12.4. Version. 156
10.12.4.1. Version-max. 156
10.12.4.2. Version-min. 156
10.12.5. ISID 157
10.12.6. TSIH 158
10.12.7. Connection ID - CID. 158
10.12.8. CmdSN. 159
10.12.9. ExpStatSN. 159
10.12.10. Login Parameters 159
10.13. Login Response 160
10.13.1. Version-max. 160
10.13.2. Version-active 161
10.13.3. TSIH 161
10.13.4. StatSN 161
10.13.5. Status-Class and Status-Detail 161
10.13.6. T (Transit) Bit. 164
10.13.7. C (Continue) Bit 164
10.13.8. Login Parameters 164
10.14. Logout Request 165
10.14.1. Reason Code. 167
10.14.2. TotalAHSLength and DataSegmentLength 168
10.14.3. CID. 168
10.14.4. ExpStatSN. 168
10.14.5. Implicit termination of tasks. 168
10.15. Logout Response. 169
10.15.1. Response 170
10.15.2. TotalAHSLength and DataSegmentLength 170
10.15.3. Time2Wait. 170
10.15.4. Time2Retain. 170
10.16. SNACK Request. 171
10.16.1. Type 172
10.16.2. Data Acknowledgement 173
10.16.3. Resegmentation 173
10.16.4. Initiator Task Tag 174
10.16.5. Target Transfer Tag or SNACK Tag 174
10.16.6. BegRun 174
10.16.7. RunLength. 174
10.17. Reject 175
10.17.1. Reason 176
10.17.2. DataSN/R2TSN 177
10.17.3. StatSN, ExpCmdSN and MaxCmdSN. 177
10.17.4. Complete Header of Bad PDU 177

Docker EX1014
Page 7 of 257

10.18. NOP-Out. 178
10.18.1. Initiator Task Tag 179
10.18.2. Target Transfer Tag. 179
10.18.3. Ping Data. 179
10.19. NOP-In 180
10.19.1. Target Transfer Tag. 181
10.19.2. StatSN 181
10.19.3. LUN. 181
11. iSCSI Security Text Keys and Authentication Methods 181
11.1. AuthMethod 182
11.1.1. Kerberos 184
11.1.2. Simple Public-Key Mechanism (SPKM) 184
11.1.3. Secure Remote Password (SRP) 185
11.1.4. Challenge Handshake Authentication Protocol (CHAP) 186
12. Login/Text Operational Text Keys. 187
12.1. HeaderDigest and DataDigest. 188
12.2. MaxConnections 190
12.3. SendTargets. 191
12.4. TargetName 191
12.5. InitiatorName. 192
12.6. TargetAlias. 192
12.7. InitiatorAlias 193
12.8. TargetAddress. 193
12.9. TargetPortalGroupTag 194
12.10. InitialR2T 194
12.11. ImmediateData. 195
12.12. MaxRecvDataSegmentLength 196
12.13. MaxBurstLength 196
12.14. FirstBurstLength 197
12.15. DefaultTime2Wait 197
12.16. DefaultTime2Retain 198
12.17. MaxOutstandingR2T. 198
12.18. DataPDUInOrder 198
12.19. DataSequenceInOrder. 199
12.20. ErrorRecoveryLevel 199
12.21. SessionType. 200
12.22. The Private or Public Extension Key Format 200
13. IANA Considerations 201
13.1. Naming Requirements. 203
13.2. Mechanism Specification Requirements 203
13.3. Publication Requirements 203
13.4. Security Requirements. 203
13.5. Registration Procedure 204
13.5.1. Present the iSCSI extension item to the Community. 204
13.5.2. iSCSI extension item review and IESG approval 204

Docker EX1014
Page 8 of 257

13.5.3. IANA Registration. 204
13.5.4. Standard iSCSI extension item-label format . 204
13.6. IANA Procedures for Registering iSCSI extension items. 205
References. 205
Appendix A. Sync and Steering with Fixed Interval Markers 209
A.1. Markers At Fixed Intervals 209
A.2. Initial Marker-less Interval 210
A.3. Negotiation. 210
A.3.1. OFMarker, IFMarker 210
A.3.2. OFMarkInt, IFMarkInt 211
Appendix B. Examples 212
B.1. Read Operation Example 212
B.2. Write Operation Example. 213
B.3. R2TSN/DataSN Use Examples. 214
B.4. CRC Examples 217
Appendix C. Login Phase Examples 219
Appendix D. SendTargets Operation. 229
Appendix E. Algorithmic Presentation of Error Recovery Classes . 233
E.1. General Data Structure and Procedure Description . . . 233
E.2. Within-command Error Recovery Algorithms 234
E.2.1. Procedure Descriptions 234
E.2.2. Initiator Algorithms 235
E.2.3. Target Algorithms. 237
E.3. Within-connection Recovery Algorithms. 240
E.3.1. Procedure Descriptions 240
E.3.2. Initiator Algorithms 241
E.3.3. Target Algorithms. 243
E.4. Connection Recovery Algorithms 243
E.4.1. Procedure Descriptions 243
E.4.2. Initiator Algorithms 244
E.4.3. Target Algorithms. 246
Appendix F. Clearing Effects of Various Events on Targets. . . . 249
F.1. Clearing Effects on iSCSI Objects. 249
F.2. Clearing Effects on SCSI Objects 253
Acknowledgements. 254
Authors' Addresses. 256
Full Copyright Statement. 257

1. Introduction

The Small Computer Systems Interface (SCSI) is a popular family of protocols for communicating with I/O devices, especially storage devices. SCSI is a client-server architecture. Clients of a SCSI interface are called "initiators". Initiators issue SCSI "commands" to request services from components, logical units of a server known as a "target". A "SCSI transport" maps the client-server SCSI protocol to a specific interconnect. An Initiator is one endpoint of a SCSI transport and a target is the other endpoint.

Docker EX1014
Page 9 of 257

The SCSI protocol has been mapped over various transports, including Parallel SCSI, IPI, IEEE-1394 (firewire) and Fibre Channel. These transports are I/O specific and have limited distance capabilities.

The iSCSI protocol defined in this document describes a means of transporting SCSI packets over TCP/IP (see [RFC791], [RFC793], [RFC1035], [RFC1122]), providing for an interoperable solution which can take advantage of existing Internet infrastructure, Internet management facilities, and address distance limitations.

2. Definitions and Acronyms

2.1. Definitions

- Alias: An alias string can also be associated with an iSCSI Node. The alias allows an organization to associate a user-friendly string with the iSCSI Name. However, the alias string is not a substitute for the iSCSI Name.

- CID (Connection ID): Connections within a session are identified by a connection ID. It is a unique ID for this connection within the session for the initiator. It is generated by the initiator and presented to the target during login requests and during logouts that close connections.

- Connection: A connection is a TCP connection. Communication between the initiator and target occurs over one or more TCP connections. The TCP connections carry control messages, SCSI commands, parameters, and data within iSCSI Protocol Data Units (iSCSI PDUs).

- iSCSI Device: A SCSI Device using an iSCSI service delivery subsystem. Service Delivery Subsystem is defined by [SAM2] as a transport mechanism for SCSI commands and responses.

- iSCSI Initiator Name: The iSCSI Initiator Name specifies the worldwide unique name of the initiator.

- iSCSI Initiator Node: The "initiator". The word "initiator" has been appropriately qualified as either a port or a device in the rest of the document when the context is ambiguous. All unqualified usages of "initiator" refer to an initiator port (or device) depending on the context.

- iSCSI Layer: This layer builds/receives iSCSI PDUs and relays/receives them to/from one or more TCP connections that form an initiator-target "session".

Docker EX1014
Page 10 of 257

- iSCSI Name: The name of an iSCSI initiator or iSCSI target.

- iSCSI Node: The iSCSI Node represents a single iSCSI initiator or iSCSI target. There are one or more iSCSI Nodes within a Network Entity. The iSCSI Node is accessible via one or more Network Portals. An iSCSI Node is identified by its iSCSI Name. The separation of the iSCSI Name from the addresses used by and for the iSCSI Node allows multiple iSCSI Nodes to use the same address, and the same iSCSI Node to use multiple addresses.

- iSCSI Target Name: The iSCSI Target Name specifies the worldwide unique name of the target.

- iSCSI Target Node: The "target".

- iSCSI Task: An iSCSI task is an iSCSI request for which a response is expected.

- iSCSI Transfer Direction: The iSCSI transfer direction is defined with regard to the initiator. Outbound or outgoing transfers are transfers from the initiator to the target, while inbound or incoming transfers are from the target to the initiator.

- ISID: The initiator part of the Session Identifier. It is explicitly specified by the initiator during Login.

- I_T nexus: According to [SAM2], the I_T nexus is a relationship between a SCSI Initiator Port and a SCSI Target Port. For iSCSI, this relationship is a session, defined as a relationship between an iSCSI Initiator's end of the session (SCSI Initiator Port) and the iSCSI Target's Portal Group. The I_T nexus can be identified by the conjunction of the SCSI port names; that is, the I_T nexus identifier is the tuple (iSCSI Initiator Name + ',i,'+ ISID, iSCSI Target Name + ',t,'+ Portal Group Tag).

- Network Entity: The Network Entity represents a device or gateway that is accessible from the IP network. A Network Entity must have one or more Network Portals, each of which can be used to gain access to the IP network by some iSCSI Nodes contained in that Network Entity.

- Network Portal: The Network Portal is a component of a Network Entity that has a TCP/IP network address and that may be used by an iSCSI Node within that Network Entity for the connection(s) within one of its iSCSI sessions. A Network Portal in an initiator is identified by its IP address. A Network Portal in a target is identified by its IP address and its listening TCP port.

Docker EX1014
Page 11 of 257

- Originator: In a negotiation or exchange, the party that initiates the negotiation or exchange.

- PDU (Protocol Data Unit): The initiator and target divide their communications into messages. The term "iSCSI protocol data unit" (iSCSI PDU) is used for these messages.

- Portal Groups: iSCSI supports multiple connections within the same session; some implementations will have the ability to combine connections in a session across multiple Network Portals. A Portal Group defines a set of Network Portals within an iSCSI Network Entity that collectively supports the capability of coordinating a session with connections spanning these portals. Not all Network Portals within a Portal Group need participate in every session connected through that Portal Group. One or more Portal Groups may provide access to an iSCSI Node. Each Network Portal, as utilized by a given iSCSI Node, belongs to exactly one portal group within that node.

- Portal Group Tag: This 16-bit quantity identifies a Portal Group within an iSCSI Node. All Network Portals with the same portal group tag in the context of a given iSCSI Node are in the same Portal Group.

- Recovery R2T: An R2T generated by a target upon detecting the loss of one or more Data-Out PDUs through one of the following means: a digest error, a sequence error, or a sequence reception timeout. A recovery R2T carries the next unused R2TSN, but requests all or part of the data burst that an earlier R2T (with a lower R2TSN) had already requested.

- Responder: In a negotiation or exchange, the party that responds to the originator of the negotiation or exchange.

- SCSI Device: This is the SAM2 term for an entity that contains one or more SCSI ports that are connected to a service delivery subsystem and supports a SCSI application protocol. For example, a SCSI Initiator Device contains one or more SCSI Initiator Ports and zero or more application clients. A Target Device contains one or more SCSI Target Ports and one or more device servers and associated logical units. For iSCSI, the SCSI Device is the component within an iSCSI Node that provides the SCSI functionality. As such, there can be at most, one SCSI Device within a given iSCSI Node. Access to the SCSI Device can only be achieved in an iSCSI normal operational session. The SCSI Device Name is defined to be the iSCSI Name of the node.

Docker EX1014
Page 12 of 257

- SCSI Layer: This builds/receives SCSI CDBs (Command Descriptor Blocks) and relays/receives them with the remaining command execute [SAM2] parameters to/from the iSCSI Layer.

- Session: The group of TCP connections that link an initiator with a target form a session (loosely equivalent to a SCSI I-T nexus). TCP connections can be added and removed from a session. Across all connections within a session, an initiator sees one and the same target.

- SCSI Initiator Port: This maps to the endpoint of an iSCSI normal operational session. An iSCSI normal operational session is negotiated through the login process between an iSCSI initiator node and an iSCSI target node. At successful completion of this process, a SCSI Initiator Port is created within the SCSI Initiator Device. The SCSI Initiator Port Name and SCSI Initiator Port Identifier are both defined to be the iSCSI Initiator Name together with (a) a label that identifies it as an initiator port name/identifier and (b) the ISID portion of the session identifier.

- SCSI Port: This is the SAM2 term for an entity in a SCSI Device that provides the SCSI functionality to interface with a service delivery subsystem. For iSCSI, the definition of the SCSI Initiator Port and the SCSI Target Port are different.

- SCSI Port Name: A name made up as UTF-8 [RFC2279] characters and includes the iSCSI Name + 'i' or 't' + ISID or Portal Group Tag.

- SCSI Target Port: This maps to an iSCSI Target Portal Group.

- SCSI Target Port Name and SCSI Target Port Identifier: These are both defined to be the iSCSI Target Name together with (a) a label that identifies it as a target port name/identifier and (b) the portal group tag.

- SSID (Session ID): A session between an iSCSI initiator and an iSCSI target is defined by a session ID that is a tuple composed of an initiator part (ISID) and a target part (Target Portal Group Tag). The ISID is explicitly specified by the initiator at session establishment. The Target Portal Group Tag is implied by the initiator through the selection of the TCP endpoint at connection establishment. The TargetPortalGroupTag key must also be returned by the target as a confirmation during connection establishment when TargetName is given.

- Target Portal Group Tag: A numerical identifier (16-bit) for an iSCSI Target Portal Group.

Docker EX1014
Page 13 of 257

- TSIH (Target Session Identifying Handle): A target assigned tag for a session with a specific named initiator. The target generates it during session establishment. Its internal format and content are not defined by this protocol, except for the value 0 that is reserved and used by the initiator to indicate a new session. It is given to the target during additional connection establishment for the same session.

2.2. Acronyms

Acronym	Definition
3DES	Triple Data Encryption Standard
ACA	Auto Contingent Allegiance
AEN	Asynchronous Event Notification
AES	Advanced Encryption Standard
AH	Additional Header (not the IPsec AH!)
AHS	Additional Header Segment
API	Application Programming Interface
ASC	Additional Sense Code
ASCII	American Standard Code for Information Interchange
ASCQ	Additional Sense Code Qualifier
BHS	Basic Header Segment
CBC	Cipher Block Chaining
CD	Compact Disk
CDB	Command Descriptor Block
CHAP	Challenge Handshake Authentication Protocol
CID	Connection ID
CO	Connection Only
CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
CSG	Current Stage
CSM	Connection State Machine
DES	Data Encryption Standard
DNS	Domain Name Server
DOI	Domain of Interpretation
DVD	Digital Versatile Disk
ESP	Encapsulating Security Payload
EUI	Extended Unique Identifier
FFP	Full Feature Phase
FFPO	Full Feature Phase Only
FIM	Fixed Interval Marker
Gbps	Gigabits per Second
HBA	Host Bus Adapter
HMAC	Hashed Message Authentication Code
I_T	Initiator_Target
I_T_L	Initiator_Target_LUN
IANA	Internet Assigned Numbers Authority

ID	Identifier
IDN	Internationalized Domain Name
IEEE	Institute of Electrical & Electronics Engineers
IETF	Internet Engineering Task Force
IKE	Internet Key Exchange
I/O	Input - Output
IO	Initialize Only
IP	Internet Protocol
IPsec	Internet Protocol Security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IQN	iSCSI Qualified Name
ISID	Initiator Session ID
ITN	iSCSI Target Name
ITT	Initiator Task Tag
KRB5	Kerberos V5
LFL	Lower Functional Layer
LTDS	Logical-Text-Data-Segment
LO	Leading Only
LU	Logical Unit
LUN	Logical Unit Number
MAC	Message Authentication Codes
NA	Not Applicable
NIC	Network Interface Card
NOP	No Operation
NSG	Next Stage
OS	Operating System
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
R2T	Ready To Transfer
R2TSN	Ready To Transfer Sequence Number
RDMA	Remote Direct Memory Access
RFC	Request For Comments
SAM	SCSI Architecture Model
SAM2	SCSI Architecture Model - 2
SAN	Storage Area Network
SCSI	Small Computer Systems Interface
SN	Sequence Number
SNACK	Selective Negative Acknowledgment - also Sequence Number Acknowledgement for data
SPKM	Simple Public-Key Mechanism
SRP	Secure Remote Password
SSID	Session ID
SW	Session Wide
TCB	Task Control Block
TCP	Transmission Control Protocol
TPGT	Target Portal Group Tag
TSIH	Target Session Identifying Handle

TTT	Target Transfer Tag
UFL	Upper Functional Layer
ULP	Upper Level Protocol
URN	Uniform Resource Names [RFC2396]
UTF	Universal Transformation Format
WG	Working Group

2.3. Conventions

In examples, "I->" and "T->" show iSCSI PDUs sent by the initiator and target respectively.

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 BCP 14 [RFC2119].

iSCSI messages - PDUs - are represented by diagrams as in the following example:

```
 Byte/     0       |       1       |       2       |       3       |
    /              |               |               |               |
   |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
   +---------------+---------------+---------------+---------------+
  0| Basic Header Segment (BHS)                                    |
   +---------------+---------------+---------------+---------------+
 ----------
  +|                                                               |
   +---------------+---------------+---------------+---------------+
```

The diagrams include byte and bit numbering.

The following representation and ordering rules are observed in this document:

- Word Rule
- Half-word Rule
- Byte Rule

2.3.1. Word Rule

A word holds four consecutive bytes. Whenever a word has numeric content, it is considered an unsigned number in base 2 positional representation with the lowest numbered byte (e.g., byte 0) bit 0 representing 2**31 and bit 1 representing 2**30 through lowest numbered byte + 3 (e.g., byte 3) bit 7 representing 2**0.

Decimal and hexadecimal representation of word values map this representation to decimal or hexadecimal positional notation.

2.3.2. Half-Word Rule

A half-word holds two consecutive bytes. Whenever a half-word has numeric content it is considered an unsigned number in base 2 positional representation with the lowest numbered byte (e.g., byte 0), bit 0 representing 2**15 and bit 1 representing 2**14 through lowest numbered byte + 1 (e.g., byte 1), bit 7 representing 2**0.

Decimal and hexadecimal representation of half-word values map this representation to decimal or hexadecimal positional notation.

2.3.3. Byte Rule

For every PDU, bytes are sent and received in increasing numbered order (network order).

Whenever a byte has numerical content, it is considered an unsigned number in base 2 positional representation with bit 0 representing 2**7 and bit 1 representing 2**6 through bit 7 representing 2**0.

3. Overview

3.1. SCSI Concepts

The SCSI Architecture Model-2 [SAM2] describes in detail the architecture of the SCSI family of I/O protocols. This section provides a brief background of the SCSI architecture and is intended to familiarize readers with its terminology.

At the highest level, SCSI is a family of interfaces for requesting services from I/O devices, including hard drives, tape drives, CD and DVD drives, printers, and scanners. In SCSI terminology, an individual I/O device is called a "logical unit" (LU).

SCSI is a client-server architecture. Clients of a SCSI interface are called "initiators". Initiators issue SCSI "commands" to request services from components, logical units, of a server known as a "target". The "device server" on the logical unit accepts SCSI commands and processes them.

A "SCSI transport" maps the client-server SCSI protocol to a specific interconnect. Initiators are one endpoint of a SCSI transport. The "target" is the other endpoint. A target can contain multiple Logical Units (LUs). Each Logical Unit has an address within a target called a Logical Unit Number (LUN).

A SCSI task is a SCSI command or possibly a linked set of SCSI commands. Some LUs support multiple pending (queued) tasks, but the

Docker EX1014
Page 17 of 257

queue of tasks is managed by the logical unit. The target uses an initiator provided "task tag" to distinguish between tasks. Only one command in a task can be outstanding at any given time.

Each SCSI command results in an optional data phase and a required response phase. In the data phase, information can travel from the initiator to target (e.g., WRITE), target to initiator (e.g., READ), or in both directions. In the response phase, the target returns the final status of the operation, including any errors.

Command Descriptor Blocks (CDB) are the data structures used to contain the command parameters that an initiator sends to a target. The CDB content and structure is defined by [SAM2] and device-type specific SCSI standards.

3.2. iSCSI Concepts and Functional Overview

The iSCSI protocol is a mapping of the SCSI remote procedure invocation model (see [SAM2]) over the TCP protocol. SCSI commands are carried by iSCSI requests and SCSI responses and status are carried by iSCSI responses. iSCSI also uses the request response mechanism for iSCSI protocol mechanisms.

For the remainder of this document, the terms "initiator" and "target" refer to "iSCSI initiator node" and "iSCSI target node", respectively (see Section 3.4.1 iSCSI Architecture Model) unless otherwise qualified.

In keeping with similar protocols, the initiator and target divide their communications into messages. This document uses the term "iSCSI protocol data unit" (iSCSI PDU) for these messages.

For performance reasons, iSCSI allows a "phase-collapse". A command and its associated data may be shipped together from initiator to target, and data and responses may be shipped together from targets.

The iSCSI transfer direction is defined with respect to the initiator. Outbound or outgoing transfers are transfers from an initiator to a target, while inbound or incoming transfers are from a target to an initiator.

An iSCSI task is an iSCSI request for which a response is expected.

In this document "iSCSI request", "iSCSI command", request, or (unqualified) command have the same meaning. Also, unless otherwise specified, status, response, or numbered response have the same meaning.

Docker EX1014
Page 18 of 257

3.2.1. Layers and Sessions

The following conceptual layering model is used to specify initiator and target actions and the way in which they relate to transmitted and received Protocol Data Units:

a) the SCSI layer builds/receives SCSI CDBs (Command Descriptor Blocks) and passes/receives them with the remaining command execute parameters ([SAM2]) to/from

b) the iSCSI layer that builds/receives iSCSI PDUs and relays/receives them to/from one or more TCP connections; the group of connections form an initiator-target "session".

Communication between the initiator and target occurs over one or more TCP connections. The TCP connections carry control messages, SCSI commands, parameters, and data within iSCSI Protocol Data Units (iSCSI PDUs). The group of TCP connections that link an initiator with a target form a session (loosely equivalent to a SCSI I_T nexus, see Section 3.4.2 SCSI Architecture Model). A session is defined by a session ID that is composed of an initiator part and a target part. TCP connections can be added and removed from a session. Each connection within a session is identified by a connection ID (CID).

Across all connections within a session, an initiator sees one "target image". All target identifying elements, such as LUN, are the same. A target also sees one "initiator image" across all connections within a session. Initiator identifying elements, such as the Initiator Task Tag, are global across the session regardless of the connection on which they are sent or received.

iSCSI targets and initiators MUST support at least one TCP connection and MAY support several connections in a session. For error recovery purposes, targets and initiators that support a single active connection in a session SHOULD support two connections during recovery.

3.2.2. Ordering and iSCSI Numbering

iSCSI uses Command and Status numbering schemes and a Data sequencing scheme.

Command numbering is session-wide and is used for ordered command delivery over multiple connections. It can also be used as a mechanism for command flow control over a session.

Docker EX1014
Page 19 of 257

Status numbering is per connection and is used to enable missing status detection and recovery in the presence of transient or permanent communication errors.

Data sequencing is per command or part of a command (R2T triggered sequence) and is used to detect missing data and/or R2T PDUs due to header digest errors.

Typically, fields in the iSCSI PDUs communicate the Sequence Numbers between the initiator and target. During periods when traffic on a connection is unidirectional, iSCSI NOP-Out/In PDUs may be utilized to synchronize the command and status ordering counters of the target and initiator.

The iSCSI session abstraction is equivalent to the SCSI I_T nexus, and the iSCSI session provides an ordered command delivery from the SCSI initiator to the SCSI target. For detailed design considerations that led to the iSCSI session model as it is defined here and how it relates the SCSI command ordering features defined in SCSI specifications to the iSCSI concepts see [CORD].

3.2.2.1. Command Numbering and Acknowledging

iSCSI performs ordered command delivery within a session. All commands (initiator-to-target PDUs) in transit from the initiator to the target are numbered.

iSCSI considers a task to be instantiated on the target in response to every request issued by the initiator. A set of task management operations including abort and reassign (see Section 10.5 Task Management Function Request) may be performed on any iSCSI task.

Some iSCSI tasks are SCSI tasks, and many SCSI activities are related to a SCSI task ([SAM2]). In all cases, the task is identified by the Initiator Task Tag for the life of the task.

The command number is carried by the iSCSI PDU as CmdSN (Command Sequence Number). The numbering is session-wide. Outgoing iSCSI PDUs carry this number. The iSCSI initiator allocates CmdSNs with a 32-bit unsigned counter (modulo 2**32). Comparisons and arithmetic on CmdSN use Serial Number Arithmetic as defined in [RFC1982] where SERIAL_BITS = 32.

Commands meant for immediate delivery are marked with an immediate delivery flag; they MUST also carry the current CmdSN. CmdSN does not advance after a command marked for immediate delivery is sent.

Command numbering starts with the first login request on the first connection of a session (the leading login on the leading connection) and command numbers are incremented by 1 for every non-immediate command issued afterwards.

If immediate delivery is used with task management commands, these commands may reach the target before the tasks on which they are supposed to act. However their CmdSN serves as a marker of their position in the stream of commands. The initiator and target must ensure that the task management commands act as specified by [SAM2]. For example, both commands and responses appear as if delivered in order. Whenever CmdSN for an outgoing PDU is not specified by an explicit rule, CmdSN will carry the current value of the local CmdSN variable (see later in this section).

The means by which an implementation decides to mark a PDU for immediate delivery or by which iSCSI decides by itself to mark a PDU for immediate delivery are beyond the scope of this document.

The number of commands used for immediate delivery is not limited and their delivery for execution is not acknowledged through the numbering scheme. Immediate commands MAY be rejected by the iSCSI target layer due to a lack of resources. An iSCSI target MUST be able to handle at least one immediate task management command and one immediate non-task-management iSCSI command per connection at any time.

In this document, delivery for execution means delivery to the SCSI execution engine or an iSCSI protocol specific execution engine (e.g., for text requests with public or private extension keys involving an execution component). With the exception of the commands marked for immediate delivery, the iSCSI target layer MUST deliver the commands for execution in the order specified by CmdSN. Commands marked for immediate delivery may be delivered by the iSCSI target layer for execution as soon as detected. iSCSI may avoid delivering some commands to the SCSI target layer if required by a prior SCSI or iSCSI action (e.g., CLEAR TASK SET Task Management request received before all the commands on which it was supposed to act).

On any connection, the iSCSI initiator MUST send the commands in increasing order of CmdSN, except for commands that are retransmitted due to digest error recovery and connection recovery.

For the numbering mechanism, the initiator and target maintain the following three variables for each session:

Docker EX1014
Page 21 of 257

- CmdSN - the current command Sequence Number, advanced by 1 on each command shipped except for commands marked for immediate delivery. CmdSN always contains the number to be assigned to the next Command PDU.
- ExpCmdSN - the next expected command by the target. The target acknowledges all commands up to, but not including, this number. The initiator treats all commands with CmdSN less than ExpCmdSN as acknowledged. The target iSCSI layer sets the ExpCmdSN to the largest non-immediate CmdSN that it can deliver for execution plus 1 (no holes in the CmdSN sequence).
- MaxCmdSN - the maximum number to be shipped. The queuing capacity of the receiving iSCSI layer is MaxCmdSN - ExpCmdSN + 1.

The initiator's ExpCmdSN and MaxCmdSN are derived from target-to-initiator PDU fields. Comparisons and arithmetic on ExpCmdSN and MaxCmdSN MUST use Serial Number Arithmetic as defined in [RFC1982] where SERIAL_BITS = 32.

The target MUST NOT transmit a MaxCmdSN that is less than ExpCmdSN-1. For non-immediate commands, the CmdSN field can take any value from ExpCmdSN to MaxCmdSN inclusive. The target MUST silently ignore any non-immediate command outside of this range or non-immediate duplicates within the range. The CmdSN carried by immediate commands may lie outside the ExpCmdSN to MaxCmdSN range. For example, if the initiator has previously sent a non-immediate command carrying the CmdSN equal to MaxCmdSN, the target window is closed. For group task management commands issued as immediate commands, CmdSN indicates the scope of the group action (e.g., on ABORT TASK SET indicates which commands are aborted).

MaxCmdSN and ExpCmdSN fields are processed by the initiator as follows:

- If the PDU MaxCmdSN is less than the PDU ExpCmdSN-1 (in Serial Arithmetic Sense), they are both ignored.
- If the PDU MaxCmdSN is greater than the local MaxCmdSN (in Serial Arithmetic Sense), it updates the local MaxCmdSN; otherwise, it is ignored.
- If the PDU ExpCmdSN is greater than the local ExpCmdSN (in Serial Arithmetic Sense), it updates the local ExpCmdSN; otherwise, it is ignored.

This sequence is required because updates may arrive out of order (e.g., the updates are sent on different TCP connections).

iSCSI initiators and targets MUST support the command numbering scheme.

A numbered iSCSI request will not change its allocated CmdSN, regardless of the number of times and circumstances in which it is reissued (see Section 6.2.1 Usage of Retry). At the target, CmdSN is only relevant when the command has not created any state related to its execution (execution state); afterwards, CmdSN becomes irrelevant. Testing for the execution state (represented by identifying the Initiator Task Tag) MUST precede any other action at the target. If no execution state is found, it is followed by ordering and delivery. If an execution state is found, it is followed by delivery.

If an initiator issues a command retry for a command with CmdSN R on a connection when the session CmdSN value is Q, it MUST NOT advance the CmdSN past R + 2**31 -1 unless the connection is no longer operational (i.e., it has returned to the FREE state, see Section 7.1.3 Standard Connection State Diagram for an Initiator), the connection has been reinstated (see Section 5.3.4 Connection Reinstatement), or a non-immediate command with CmdSN equal or greater than Q was issued subsequent to the command retry on the same connection and the reception of that command is acknowledged by the target (see Section 9.4 Command Retry and Cleaning Old Command Instances).

A target MUST NOT issue a command response or Data-In PDU with status before acknowledging the command. However, the acknowledgement can be included in the response or Data-In PDU.

3.2.2.2. Response/Status Numbering and Acknowledging

Responses in transit from the target to the initiator are numbered. The StatSN (Status Sequence Number) is used for this purpose. StatSN is a counter maintained per connection. ExpStatSN is used by the initiator to acknowledge status. The status sequence number space is 32-bit unsigned-integers and the arithmetic operations are the regular mod(2**32) arithmetic.

Status numbering starts with the Login response to the first Login request of the connection. The Login response includes an initial value for status numbering (any initial value is valid).

To enable command recovery, the target MAY maintain enough state information for data and status recovery after a connection failure. A target doing so can safely discard all of the state information maintained for recovery of a command after the delivery of the status for the command (numbered StatSN) is acknowledged through ExpStatSN.

A large absolute difference between StatSN and ExpStatSN may indicate a failed connection. Initiators MUST undertake recovery actions if

Docker EX1014
Page 23 of 257

the difference is greater than an implementation defined constant that MUST NOT exceed 2**31-1.

Initiators and Targets MUST support the response-numbering scheme.

3.2.2.3. Data Sequencing

Data and R2T PDUs transferred as part of some command execution MUST be sequenced. The DataSN field is used for data sequencing. For input (read) data PDUs, DataSN starts with 0 for the first data PDU of an input command and advances by 1 for each subsequent data PDU. For output data PDUs, DataSN starts with 0 for the first data PDU of a sequence (the initial unsolicited sequence or any data PDU sequence issued to satisfy an R2T) and advances by 1 for each subsequent data PDU. R2Ts are also sequenced per command. For example, the first R2T has an R2TSN of 0 and advances by 1 for each subsequent R2T. For bidirectional commands, the target uses the DataSN/R2TSN to sequence Data-In and R2T PDUs in one continuous sequence (undifferentiated). Unlike command and status, data PDUs and R2Ts are not acknowledged by a field in regular outgoing PDUs. Data-In PDUs can be acknowledged on demand by a special form of the SNACK PDU. Data and R2T PDUs are implicitly acknowledged by status for the command. The DataSN/R2TSN field enables the initiator to detect missing data or R2T PDUs.

For any read or bidirectional command, a target MUST issue less than 2**32 combined R2T and Data-In PDUs. Any output data sequence MUST contain less than 2**32 Data-Out PDUs.

3.2.3. iSCSI Login

The purpose of the iSCSI login is to enable a TCP connection for iSCSI use, authentication of the parties, negotiation of the session’s parameters and marking of the connection as belonging to an iSCSI session.

A session is used to identify to a target all the connections with a given initiator that belong to the same I_T nexus. (For more details on how a session relates to an I_T nexus, see Section 3.4.2 SCSI Architecture Model).

The targets listen on a well-known TCP port or other TCP port for incoming connections. The initiator begins the login process by connecting to one of these TCP ports.

As part of the login process, the initiator and target SHOULD authenticate each other and MAY set a security association protocol for the session. This can occur in many different ways and is subject to negotiation.

Docker EX1014
Page 24 of 257

To protect the TCP connection, an IPsec security association MAY be established before the Login request. For information on using IPsec security for iSCSI see Chapter 8 and [RFC3723].

The iSCSI Login Phase is carried through Login requests and responses. Once suitable authentication has occurred and operational parameters have been set, the session transitions to the Full Feature Phase and the initiator may start to send SCSI commands. The security policy for whether, and by what means, a target chooses to authorize an initiator is beyond the scope of this document. For a more detailed description of the Login Phase, see Chapter 5.

The login PDU includes the ISID part of the session ID (SSID). The target portal group that services the login is implied by the selection of the connection endpoint. For a new session, the TSIH is zero. As part of the response, the target generates a TSIH.

During session establishment, the target identifies the SCSI initiator port (the "I" in the "I_T nexus") through the value pair (InitiatorName, ISID). We describe InitiatorName later in this section. Any persistent state (e.g., persistent reservations) on the target that is associated with a SCSI initiator port is identified based on this value pair. Any state associated with the SCSI target port (the "T" in the "I_T nexus") is identified externally by the TargetName and portal group tag (see Section 3.4.1 iSCSI Architecture Model). ISID is subject to reuse restrictions because it is used to identify a persistent state (see Section 3.4.3 Consequences of the Model).

Before the Full Feature Phase is established, only Login Request and Login Response PDUs are allowed. Login requests and responses MUST be used exclusively during Login. On any connection, the login phase MUST immediately follow TCP connection establishment and a subsequent Login Phase MUST NOT occur before tearing down a connection.

A target receiving any PDU except a Login request before the Login phase is started MUST immediately terminate the connection on which the PDU was received. Once the Login phase has started, if the target receives any PDU except a Login request, it MUST send a Login reject (with Status "invalid during login") and then disconnect. If the initiator receives any PDU except a Login response, it MUST immediately terminate the connection.

3.2.4. iSCSI Full Feature Phase

Once the initiator is authorized to do so, the iSCSI session is in the iSCSI Full Feature Phase. A session is in Full Feature Phase after successfully finishing the Login Phase on the first (leading)

Docker EX1014
Page 25 of 257

connection of a session. A connection is in Full Feature Phase if the session is in Full Feature Phase and the connection login has completed successfully. An iSCSI connection is not in Full Feature Phase

a) when it does not have an established transport connection,

OR

b) when it has a valid transport connection, but a successful login was not performed or the connection is currently logged out.

In a normal Full Feature Phase, the initiator may send SCSI commands and data to the various LUs on the target by encapsulating them in iSCSI PDUs that go over the established iSCSI session.

3.2.4.1. Command Connection Allegiance

For any iSCSI request issued over a TCP connection, the corresponding response and/or other related PDU(s) MUST be sent over the same connection. We call this "connection allegiance". If the original connection fails before the command is completed, the connection allegiance of the command may be explicitly reassigned to a different transport connection as described in detail in Section 6.2 Retry and Reassign in Recovery.

Thus, if an initiator issues a READ command, the target MUST send the requested data, if any, followed by the status to the initiator over the same TCP connection that was used to deliver the SCSI command. If an initiator issues a WRITE command, the initiator MUST send the data, if any, for that command over the same TCP connection that was used to deliver the SCSI command. The target MUST return Ready To Transfer (R2T), if any, and the status over the same TCP connection that was used to deliver the SCSI command. Retransmission requests (SNACK PDUs) and the data and status that they generate MUST also use the same connection.

However, consecutive commands that are part of a SCSI linked command-chain task (see [SAM2]) MAY use different connections. Connection allegiance is strictly per-command and not per-task. During the iSCSI Full Feature Phase, the initiator and target MAY interleave unrelated SCSI commands, their SCSI Data, and responses over the session.

3.2.4.2. Data Transfer Overview

Outgoing SCSI data (initiator to target user data or command parameters) is sent as either solicited data or unsolicited data. Solicited data are sent in response to R2T PDUs. Unsolicited data can be sent as part of an iSCSI command PDU ("immediate data") or in separate iSCSI data PDUs.

Immediate data are assumed to originate at offset 0 in the initiator SCSI write-buffer (outgoing data buffer). All other Data PDUs have the buffer offset set explicitly in the PDU header.

An initiator may send unsolicited data up to FirstBurstLength as immediate (up to the negotiated maximum PDU length), in a separate PDU sequence or both. All subsequent data MUST be solicited. The maximum length of an individual data PDU or the immediate-part of the first unsolicited burst MAY be negotiated at login.

The maximum amount of unsolicited data that can be sent with a command is negotiated at login through the FirstBurstLength key. A target MAY separately enable immediate data (through the ImmediateData key) without enabling the more general (separate data PDUs) form of unsolicited data (through the InitialR2T key).

Unsolicited data on write are meant to reduce the effect of latency on throughput (no R2T is needed to start sending data). In addition, immediate data is meant to reduce the protocol overhead (both bandwidth and execution time).

An iSCSI initiator MAY choose not to send unsolicited data, only immediate data or FirstBurstLength bytes of unsolicited data with a command. If any non-immediate unsolicited data is sent, the total unsolicited data MUST be either FirstBurstLength, or all of the data if the total amount is less than the FirstBurstLength.

It is considered an error for an initiator to send unsolicited data PDUs to a target that operates in R2T mode (only solicited data are allowed). It is also an error for an initiator to send more unsolicited data, whether immediate or as separate PDUs, than FirstBurstLength.

An initiator MUST honor an R2T data request for a valid outstanding command (i.e., carrying a valid Initiator Task Tag) and deliver all the requested data provided the command is supposed to deliver outgoing data and the R2T specifies data within the command bounds. The initiator action is unspecified for receiving an R2T request that specifies data, all or part, outside of the bounds of the command.

Docker EX1014
Page 27 of 257

A target SHOULD NOT silently discard data and then request retransmission through R2T. Initiators SHOULD NOT keep track of the data transferred to or from the target (scoreboarding). SCSI targets perform residual count calculation to check how much data was actually transferred to or from the device by a command. This may differ from the amount the initiator sent and/or received for reasons such as retransmissions and errors. Read or bidirectional commands implicitly solicit the transmission of the entire amount of data covered by the command. SCSI data packets are matched to their corresponding SCSI commands by using tags specified in the protocol.

In addition, iSCSI initiators and targets MUST enforce some ordering rules. When unsolicited data is used, the order of the unsolicited data on each connection MUST match the order in which the commands on that connection are sent. Command and unsolicited data PDUs may be interleaved on a single connection as long as the ordering requirements of each are maintained (e.g., command N+1 MAY be sent before the unsolicited Data-Out PDUs for command N, but the unsolicited Data-Out PDUs for command N MUST precede the unsolicited Data-Out PDUs of command N+1). A target that receives data out of order MAY terminate the session.

3.2.4.3. Tags and Integrity Checks

Initiator tags for pending commands are unique initiator-wide for a session. Target tags are not strictly specified by the protocol. It is assumed that target tags are used by the target to tag (alone or in combination with the LUN) the solicited data. Target tags are generated by the target and "echoed" by the initiator. These mechanisms are designed to accomplish efficient data delivery along with a large degree of control over the data flow.

As the Initiator Task Tag is used to identify a task during its execution, the iSCSI initiator and target MUST verify that all other fields used in task-related PDUs have values that are consistent with the values used at the task instantiation based on the Initiator Task Tag (e.g., the LUN used in an R2T PDU MUST be the same as the one used in the SCSI command PDU used to instantiate the task). Using inconsistent field values is considered a protocol error.

3.2.4.4. Task Management

SCSI task management assumes that individual tasks and task groups can be aborted solely based on the task tags (for individual tasks) or the timing of the task management command (for task groups), and that the task management action is executed synchronously - i.e., no message involving an aborted task will be seen by the SCSI initiator after receiving the task management response. In iSCSI initiators

Docker EX1014
Page 28 of 257

and targets interact asynchronously over several connections. iSCSI specifies the protocol mechanism and implementation requirements needed to present a synchronous view while using an asynchronous infrastructure.

3.2.5. iSCSI Connection Termination

An iSCSI connection may be terminated by use of a transport connection shutdown or a transport reset. Transport reset is assumed to be an exceptional event.

Graceful TCP connection shutdowns are done by sending TCP FINs. A graceful transport connection shutdown SHOULD only be initiated by either party when the connection is not in iSCSI Full Feature Phase. A target MAY terminate a Full Feature Phase connection on internal exception events, but it SHOULD announce the fact through an Asynchronous Message PDU. Connection termination with outstanding commands may require recovery actions.

If a connection is terminated while in Full Feature Phase, connection cleanup (see section 7) is required prior to recovery. By doing connection cleanup before starting recovery, the initiator and target will avoid receiving stale PDUs after recovery.

3.2.6. iSCSI Names

Both targets and initiators require names for the purpose of identification. In addition, names enable iSCSI storage resources to be managed regardless of location (address). An iSCSI node name is also the SCSI device name of an iSCSI device. The iSCSI name of a SCSI device is the principal object used in authentication of targets to initiators and initiators to targets. This name is also used to identify and manage iSCSI storage resources.

iSCSI names must be unique within the operational domain of the end user. However, because the operational domain of an IP network is potentially worldwide, the iSCSI name formats are architected to be worldwide unique. To assist naming authorities in the construction of worldwide unique names, iSCSI provides two name formats for different types of naming authorities.

iSCSI names are associated with iSCSI nodes, and not iSCSI network adapter cards, to ensure that the replacement of network adapter cards does not require reconfiguration of all SCSI and iSCSI resource allocation information.

Docker EX1014
Page 29 of 257

Some SCSI commands require that protocol-specific identifiers be communicated within SCSI CDBs. See Section 3.4.2 SCSI Architecture Model for the definition of the SCSI port name/identifier for iSCSI ports.

An initiator may discover the iSCSI Target Names to which it has access, along with their addresses, using the SendTargets text request, or other techniques discussed in [RFC3721].

3.2.6.1. iSCSI Name Properties

Each iSCSI node, whether an initiator or target, MUST have an iSCSI name.

Initiators and targets MUST support the receipt of iSCSI names of up to the maximum length of 223 bytes.

The initiator MUST present both its iSCSI Initiator Name and the iSCSI Target Name to which it wishes to connect in the first login request of a new session or connection. The only exception is if a discovery session (see Section 2.3 iSCSI Session Types) is to be established. In this case, the iSCSI Initiator Name is still required, but the iSCSI Target Name MAY be omitted.

iSCSI names have the following properties:

a) iSCSI names are globally unique. No two initiators or targets can have the same name.
b) iSCSI names are permanent. An iSCSI initiator node or target node has the same name for its lifetime.
c) iSCSI names do not imply a location or address. An iSCSI initiator or target can move, or have multiple addresses. A change of address does not imply a change of name.
d) iSCSI names do not rely on a central name broker; the naming authority is distributed.
e) iSCSI names support integration with existing unique naming schemes.
f) iSCSI names rely on existing naming authorities. iSCSI does not create any new naming authority.

The encoding of an iSCSI name has the following properties:

a) iSCSI names have the same encoding method regardless of the underlying protocols.
b) iSCSI names are relatively simple to compare. The algorithm for comparing two iSCSI names for equivalence does not rely on an external server.

Docker EX1014
Page 30 of 257

c) iSCSI names are composed only of displayable characters. iSCSI names allow the use of international character sets but are not case sensitive. No whitespace characters are used in iSCSI names.
d) iSCSI names may be transported using both binary and ASCII-based protocols.

An iSCSI name really names a logical software entity, and is not tied to a port or other hardware that can be changed. For instance, an initiator name should name the iSCSI initiator node, not a particular NIC or HBA. When multiple NICs are used, they should generally all present the same iSCSI initiator name to the targets, because they are simply paths to the same SCSI layer. In most operating systems, the named entity is the operating system image.

Similarly, a target name should not be tied to hardware interfaces that can be changed. A target name should identify the logical target and must be the same for the target regardless of the physical portion being addressed. This assists iSCSI initiators in determining that the two targets it has discovered are really two paths to the same target.

The iSCSI name is designed to fulfill the functional requirements for Uniform Resource Names (URN) [RFC1737]. For example, it is required that the name have a global scope, be independent of address or location, and be persistent and globally unique. Names must be extensible and scalable with the use of naming authorities. The name encoding should be both human and machine readable. See [RFC1737] for further requirements.

3.2.6.2. iSCSI Name Encoding

An iSCSI name MUST be a UTF-8 encoding of a string of Unicode characters with the following properties:

- It is in Normalization Form C (see "Unicode Normalization Forms" [UNICODE]).
- It only contains characters allowed by the output of the iSCSI stringprep template (described in [RFC3722]).
- The following characters are used for formatting iSCSI names:

  - dash ('-'=U+002d)
  - dot ('.'=U+002e)
  - colon (':'=U+003a)

- The UTF-8 encoding of the name is not larger than 223 bytes.

The stringprep process is described in [RFC3454]; iSCSI's use of the stringprep process is described in [RFC3722]. Stringprep is a method designed by the Internationalized Domain Name (IDN) working group to translate human-typed strings into a format that can be compared as opaque strings. Strings MUST NOT include punctuation, spacing, diacritical marks, or other characters that could get in the way of readability. The stringprep process also converts strings into equivalent strings of lower-case characters.

The stringprep process does not need to be implemented if the names are only generated using numeric and lower-case (any character set) alphabetic characters.

Once iSCSI names encoded in UTF-8 are "normalized" they may be safely compared byte-for-byte.

3.2.6.3. iSCSI Name Structure

An iSCSI name consists of two parts--a type designator followed by a unique name string.

The iSCSI name does not define any new naming authorities. Instead, it supports two existing ways of designating naming authorities: an iSCSI-Qualified Name, using domain names to identify a naming authority, and the EUI format, where the IEEE Registration Authority assists in the formation of worldwide unique names (EUI-64 format).

The type designator strings currently defined are:

```
iqn.       - iSCSI Qualified name
eui.       - Remainder of the string is an IEEE EUI-64
             identifier, in ASCII-encoded hexadecimal.
```

These two naming authority designators were considered sufficient at the time of writing this document. The creation of additional naming type designators for iSCSI may be considered by the IETF and detailed in separate RFCs.

3.2.6.3.1. Type "iqn." (iSCSI Qualified Name)

This iSCSI name type can be used by any organization that owns a domain name. This naming format is useful when an end user or service provider wishes to assign iSCSI names for targets and/or initiators.

To generate names of this type, the person or organization generating the name must own a registered domain name. This domain name does not have to be active, and does not have to resolve to an address; it

Docker EX1014
Page 32 of 257

just needs to be reserved to prevent others from generating iSCSI names using the same domain name.

Since a domain name can expire, be acquired by another entity, or may be used to generate iSCSI names by both owners, the domain name must be additionally qualified by a date during which the naming authority owned the domain name. For this reason, a date code is provided as part of the "iqn." format.

The iSCSI qualified name string consists of:

- The string "iqn.", used to distinguish these names from "eui." formatted names.
- A date code, in yyyy-mm format. This date MUST be a date during which the naming authority owned the domain name used in this format, and SHOULD be the first month in which the domain name was owned by this naming authority at 00:01 GMT of the first day of the month. This date code uses the Gregorian calendar. All four digits in the year must be present. Both digits of the month must be present, with January == "01" and December == "12". The dash must be included.
- A dot "."
- The reversed domain name of the naming authority (person or organization) creating this iSCSI name.
- An optional, colon (:) prefixed, string within the character set and length boundaries that the owner of the domain name deems appropriate. This may contain product types, serial numbers, host identifiers, or software keys (e.g., it may include colons to separate organization boundaries). With the exception of the colon prefix, the owner of the domain name can assign everything after the reversed domain name as desired. It is the responsibility of the entity that is the naming authority to ensure that the iSCSI names it assigns are worldwide unique. For example, "Example Storage Arrays, Inc.", might own the domain name "example.com".

The following are examples of iSCSI qualified names that might be generated by "EXAMPLE Storage Arrays, Inc."

```
                 Naming     String defined by
   Type  Date     Auth      "example.com" naming authority
  +--++-----+ +---------+ +--------------------------------+
  |  ||     | |         | |                                |

  iqn.2001-04.com.example:storage:diskarrays-sn-a8675309
  iqn.2001-04.com.example
  iqn.2001-04.com.example:storage.tape1.sys1.xyz
  iqn.2001-04.com.example:storage.disk2.sys1.xyz
```

3.2.6.3.2. Type "eui." (IEEE EUI-64 format)

The IEEE Registration Authority provides a service for assigning globally unique identifiers [EUI]. The EUI-64 format is used to build a global identifier in other network protocols. For example, Fibre Channel defines a method of encoding it into a WorldWideName. For more information on registering for EUI identifiers, see [OUI].

The format is "eui." followed by an EUI-64 identifier (16 ASCII-encoded hexadecimal digits).

Example iSCSI name:

```
Type  EUI-64 identifier (ASCII-encoded hexadecimal)
+--++--------------+
|  ||              |
eui.02004567A425678D
```

The IEEE EUI-64 iSCSI name format might be used when a manufacturer is already registered with the IEEE Registration Authority and uses EUI-64 formatted worldwide unique names for its products.

More examples of name construction are discussed in [RFC3721].

3.2.7. Persistent State

iSCSI does not require any persistent state maintenance across sessions. However, in some cases, SCSI requires persistent identification of the SCSI initiator port name (See Section 3.4.2 SCSI Architecture Model and Section 3.4.3 Consequences of the Model).

iSCSI sessions do not persist through power cycles and boot operations.

All iSCSI session and connection parameters are re-initialized upon session and connection creation.

Commands persist beyond connection termination if the session persists and command recovery within the session is supported. However, when a connection is dropped, command execution, as perceived by iSCSI (i.e., involving iSCSI protocol exchanges for the affected task), is suspended until a new allegiance is established by the ’task reassign’ task management function. (See Section 10.5 Task Management Function Request.)

Docker EX1014
Page 34 of 257

3.2.8. Message Synchronization and Steering

iSCSI presents a mapping of the SCSI protocol onto TCP. This encapsulation is accomplished by sending iSCSI PDUs of varying lengths. Unfortunately, TCP does not have a built-in mechanism for signaling message boundaries at the TCP layer. iSCSI overcomes this obstacle by placing the message length in the iSCSI message header. This serves to delineate the end of the current message as well as the beginning of the next message.

In situations where IP packets are delivered in order from the network, iSCSI message framing is not an issue and messages are processed one after the other. In the presence of IP packet reordering (i.e., frames being dropped), legacy TCP implementations store the "out of order" TCP segments in temporary buffers until the missing TCP segments arrive, upon which the data must be copied to the application buffers. In iSCSI, it is desirable to steer the SCSI data within these out of order TCP segments into the pre-allocated SCSI buffers rather than store them in temporary buffers. This decreases the need for dedicated reassembly buffers as well as the latency and bandwidth related to extra copies.

Relying solely on the "message length" information from the iSCSI message header may make it impossible to find iSCSI message boundaries in subsequent TCP segments due to the loss of a TCP segment that contains the iSCSI message length. The missing TCP segment(s) must be received before any of the following segments can be steered to the correct SCSI buffers (due to the inability to determine the iSCSI message boundaries). Since these segments cannot be steered to the correct location, they must be saved in temporary buffers that must then be copied to the SCSI buffers.

Different schemes can be used to recover synchronization. To make these schemes work, iSCSI implementations have to make sure that the appropriate protocol layers are provided with enough information to implement a synchronization and/or data steering mechanism. One of these schemes is detailed in Appendix A. - Sync and Steering with Fixed Interval Markers -.

The Fixed Interval Markers (FIM) scheme works by inserting markers in the payload stream at fixed intervals that contain the offset for the start of the next iSCSI PDU.

Under normal circumstances (no PDU loss or data reception out of order), iSCSI data steering can be accomplished by using the identifying tag and the data offset fields in the iSCSI header in addition to the TCP sequence number from the TCP header. The

Docker EX1014
Page 35 of 257

identifying tag helps associate the PDU with a SCSI buffer address while the data offset and TCP sequence number are used to determine the offset within the buffer.

When the part of the TCP data stream containing an iSCSI PDU header is delayed or lost, markers may be used to minimize the damage as follows:

- Markers indicate where the next iSCSI PDU starts and enable continued processing when iSCSI headers have to be dropped due to data errors discovered at the iSCSI level (e.g., iSCSI header CRC errors).

- Markers help minimize the amount of data that has to be kept by the TCP/iSCSI layer while waiting for a late TCP packet arrival or recovery, because later they might help find iSCSI PDU headers and use the information contained in those to steer data to SCSI buffers.

3.2.8.1. Sync/Steering and iSCSI PDU Length

When a large iSCSI message is sent, the TCP segment(s) that contain the iSCSI header may be lost. The remaining TCP segment(s), up to the next iSCSI message, must be buffered (in temporary buffers) because the iSCSI header that indicates to which SCSI buffers the data are to be steered was lost. To minimize the amount of buffering, it is recommended that the iSCSI PDU length be restricted to a small value (perhaps a few TCP segments in length). During login, each end of the iSCSI session specifies the maximum iSCSI PDU length it will accept.

3.3. iSCSI Session Types

iSCSI defines two types of sessions:

a) Normal operational session - an unrestricted session.
b) Discovery-session - a session only opened for target discovery. The target MUST ONLY accept text requests with the SendTargets key and a logout request with the reason "close the session". All other requests MUST be rejected.

The session type is defined during login with the key=value parameter in the login command.

3.4. SCSI to iSCSI Concepts Mapping Model

The following diagram shows an example of how multiple iSCSI Nodes (targets in this case) can coexist within the same Network Entity and can share Network Portals (IP addresses and TCP ports). Other more complex configurations are also possible. For detailed descriptions of the components of these diagrams, see Section 3.4.1 iSCSI Architecture Model.

```
+-----------------------------------+
|  Network Entity (iSCSI Client)    |
|                                   |
|         +-------------+           |
|         | iSCSI Node  |           |
|         | (Initiator) |           |
|         +-------------+           |
|            |       |              |
| +--------------+ +--------------+ |
| |Network Portal| |Network Portal| |
| |   10.1.30.4  | |   10.1.40.6  | |
+-+--------------+-+--------------+-+
         |               |
         |  IP Networks  |
         |               |
+-+--------------+-+--------------+-+
| |Network Portal| |Network Portal| |
| |  10.1.30.21  | |   10.1.40.3  | |
| | TCP Port 3260| | TCP Port 3260| |
| +--------------+ +--------------+ |
|        |               |          |
|        -----------------          |
|           |         |             |
|  +-------------+ +--------------+ |
|  | iSCSI Node  | | iSCSI Node   | |
|  |  (Target)   | |  (Target)    | |
|  +-------------+ +--------------+ |
|                                   |
|   Network Entity (iSCSI Server)   |
+-----------------------------------+
```

3.4.1. iSCSI Architecture Model

This section describes the part of the iSCSI architecture model that has the most bearing on the relationship between iSCSI and the SCSI Architecture Model.

a) Network Entity - represents a device or gateway that is accessible from the IP network. A Network Entity must have one or more Network Portals (see item d), each of which can be used by some iSCSI Nodes (see item (b)) contained in that Network Entity to gain access to the IP network.

b) iSCSI Node - represents a single iSCSI initiator or iSCSI target. There are one or more iSCSI Nodes within a Network Entity. The iSCSI Node is accessible via one or more Network Portals (see item d). An iSCSI Node is identified by its iSCSI Name (see Section 3.2.6 iSCSI Names and Chapter 12). The separation of the iSCSI Name from the addresses used by and for the iSCSI node allows multiple iSCSI nodes to use the same addresses, and the same iSCSI node to use multiple addresses.

c) An alias string may also be associated with an iSCSI Node. The alias allows an organization to associate a user friendly string with the iSCSI Name. However, the alias string is not a substitute for the iSCSI Name.

d) Network Portal - a component of a Network Entity that has a TCP/IP network address and that may be used by an iSCSI Node within that Network Entity for the connection(s) within one of its iSCSI sessions. In an initiator, it is identified by its IP address. In a target, it is identified by its IP address and its listening TCP port.

e) Portal Groups - iSCSI supports multiple connections within the same session; some implementations will have the ability to combine connections in a session across multiple Network Portals. A Portal Group defines a set of Network Portals within an iSCSI Node that collectively supports the capability of coordinating a session with connections that span these portals. Not all Network Portals within a Portal Group need to participate in every session connected through that Portal Group. One or more Portal Groups may provide access to an iSCSI Node. Each Network Portal, as utilized by a given iSCSI Node, belongs to exactly one portal group within that node. Portal Groups are identified within an iSCSI Node by a portal group tag, a simple unsigned-integer between 0 and 65535 (see Section 12.3 SendTargets). All Network Portals with the same portal group tag in the context of a given iSCSI Node are in the same Portal Group.

Both iSCSI Initiators and iSCSI Targets have portal groups, though only the iSCSI Target Portal Groups are used directly in the iSCSI protocol (e.g., in SendTargets). For references to the initiator Portal Groups, see Section 9.1.1 Conservative Reuse of ISIDs.

f) Portals within a Portal Group should support similar session parameters, because they may participate in a common session.

The following diagram shows an example of one such configuration on a target and how a session that shares Network Portals within a Portal Group may be established.

```
  ----------------------------IP Network---------------------
         |                |                    |
    +----|---------------|-----+         +----|---------+
    | +---------+  +---------+ |         | +---------+  |
    | | Network |  | Network | |         | | Network |  |
    | | Portal  |  | Portal  | |         | | Portal  |  |
    | +--|------+  +---------+ |         | +---------+  |
    |    |                |     |         |    |         |
    |    |    Portal      |     |         |    | Portal  |
    |    |    Group 1     |     |         |    | Group 2 |
    +--------------------------+         +--------------+
         |                |                    |
+--------|---------------|--------------------|--------------------+
|        |                |                    |                    |
|  +----------------------------+  +-----------------------------+ |
|  | iSCSI Session (Target side)|  | iSCSI Session (Target side) | |
|  |                            |  |                             | |
|  |       (TSIH = 56)          |  |       (TSIH = 48)           | |
|  +----------------------------+  +-----------------------------+ |
|                                                                  |
|                     iSCSI Target Node                            |
|             (within Network Entity, not shown)                   |
+------------------------------------------------------------------+
```

3.4.2. SCSI Architecture Model

This section describes the relationship between the SCSI Architecture Model [SAM2] and the constructs of the SCSI device, SCSI port and I_T nexus, and the iSCSI constructs described in Section 3.4.1 iSCSI Architecture Model.

This relationship implies implementation requirements in order to conform to the SAM2 model and other SCSI operational functions. These requirements are detailed in Section 3.4.3 Consequences of the Model.

The following list outlines mappings of SCSI architectural elements to iSCSI.

a) SCSI Device - the SAM2 term for an entity that contains one or more SCSI ports that are connected to a service delivery subsystem and supports a SCSI application protocol. For example, a SCSI Initiator Device contains one or more SCSI Initiator Ports and zero or more application clients. A SCSI Target Device contains one or more SCSI Target Ports and one or more logical units. For iSCSI, the SCSI Device is the component within an iSCSI Node that provides the SCSI functionality. As such, there can be one SCSI Device, at most, within an iSCSI Node. Access to the SCSI Device can only be achieved in an iSCSI normal operational session (see Section 3.3 iSCSI Session Types). The SCSI Device Name is defined to be the iSCSI Name of the node and MUST be used in the iSCSI protocol.

b) SCSI Port - the SAM2 term for an entity in a SCSI Device that provides the SCSI functionality to interface with a service delivery subsystem or transport. For iSCSI, the definition of SCSI Initiator Port and SCSI Target Port are different.

   SCSI Initiator Port: This maps to one endpoint of an iSCSI normal operational session (see Section 3.3 iSCSI Session Types). An iSCSI normal operational session is negotiated through the login process between an iSCSI initiator node and an iSCSI target node. At successful completion of this process, a SCSI Initiator Port is created within the SCSI Initiator Device. The SCSI Initiator Port Name and SCSI Initiator Port Identifier are both defined to be the iSCSI Initiator Name together with (a) a label that identifies it as an initiator port name/identifier and (b) the ISID portion of the session identifier.

   SCSI Target Port: This maps to an iSCSI Target Portal Group. The SCSI Target Port Name and the SCSI Target Port Identifier are both defined to be the iSCSI Target Name together with (a) a label that identifies it as a target port name/identifier and (b) the portal group tag.

   The SCSI Port Name MUST be used in iSCSI. When used in SCSI parameter data, the SCSI port name MUST be encoded as:
   - The iSCSI Name in UTF-8 format, followed by
   - a comma separator (1 byte), followed by
   - the ASCII character 'i' (for SCSI Initiator Port) or the ASCII character 't' (for SCSI Target Port) (1 byte), followed by

- a comma separator (1 byte), followed by
- a text encoding as a hex-constant (see Section 5.1 Text Format) of the ISID (for SCSI initiator port) or the portal group tag (for SCSI target port) including the initial 0X or 0x and the terminating null (15 bytes).

The ASCII character 'i' or 't' is the label that identifies this port as either a SCSI Initiator Port or a SCSI Target Port.

c) I_T nexus - a relationship between a SCSI Initiator Port and a SCSI Target Port, according to [SAM2]. For iSCSI, this relationship is a session, defined as a relationship between an iSCSI Initiator's end of the session (SCSI Initiator Port) and the iSCSI Target's Portal Group. The I_T nexus can be identified by the conjunction of the SCSI port names or by the iSCSI session identifier SSID. iSCSI defines the I_T nexus identifier to be the tuple (iSCSI Initiator Name + 'i' + ISID, iSCSI Target Name + 't' + Portal Group Tag).

NOTE: The I_T nexus identifier is not equal to the session identifier (SSID).

3.4.3. Consequences of the Model

This section describes implementation and behavioral requirements that result from the mapping of SCSI constructs to the iSCSI constructs defined above. Between a given SCSI initiator port and a given SCSI target port, only one I_T nexus (session) can exist. No more than one nexus relationship (parallel nexus) is allowed by [SAM2]. Therefore, at any given time, only one session can exist between a given iSCSI initiator node and an iSCSI target node, with the same session identifier (SSID).

These assumptions lead to the following conclusions and requirements:

ISID RULE: Between a given iSCSI Initiator and iSCSI Target Portal Group (SCSI target port), there can only be one session with a given value for ISID that identifies the SCSI initiator port. See Section 10.12.5 ISID.

The structure of the ISID that contains a naming authority component (see Section 10.12.5 ISID and [RFC3721]) provides a mechanism to facilitate compliance with the ISID rule. (See Section 9.1.1 Conservative Reuse of ISIDs.)

Docker EX1014
Page 41 of 257

The iSCSI Initiator Node should manage the assignment of ISIDs prior to session initiation. The "ISID RULE" does not preclude the use of the same ISID from the same iSCSI Initiator with different Target Portal Groups on the same iSCSI target or on other iSCSI targets (see Section 9.1.1 Conservative Reuse of ISIDs). Allowing this would be analogous to a single SCSI Initiator Port having relationships (nexus) with multiple SCSI target ports on the same SCSI target device or SCSI target ports on other SCSI target devices. It is also possible to have multiple sessions with different ISIDs to the same Target Portal Group. Each such session would be considered to be with a different initiator even when the sessions originate from the same initiator device. The same ISID may be used by a different iSCSI initiator because it is the iSCSI Name together with the ISID that identifies the SCSI Initiator Port.

NOTE: A consequence of the ISID RULE and the specification for the I_T nexus identifier is that two nexus with the same identifier should never exist at the same time.

TSIH RULE: The iSCSI Target selects a non-zero value for the TSIH at session creation (when an initiator presents a 0 value at Login). After being selected, the same TSIH value MUST be used whenever the initiator or target refers to the session and a TSIH is required.

3.4.3.1. I_T Nexus State

Certain nexus relationships contain an explicit state (e.g., initiator-specific mode pages) that may need to be preserved by the device server [SAM2] in a logical unit through changes or failures in the iSCSI layer (e.g., session failures). In order for that state to be restored, the iSCSI initiator should reestablish its session (re-login) to the same Target Portal Group using the previous ISID. That is, it should perform session recovery as described in Chapter 6. This is because the SCSI initiator port identifier and the SCSI target port identifier (or relative target port) form the datum that the SCSI logical unit device server uses to identify the I_T nexus.

3.5. Request/Response Summary

This section lists and briefly describes all the iSCSI PDU types (request and responses).

All iSCSI PDUs are built as a set of one or more header segments (basic and auxiliary) and zero or one data segments. The header group and the data segment may each be followed by a CRC (digest).

The basic header segment has a fixed length of 48 bytes.

Docker EX1014
Page 42 of 257

3.5.1. Request/Response Types Carrying SCSI Payload

3.5.1.1. SCSI-Command

This request carries the SCSI CDB and all the other SCSI execute command procedure call (see [SAM2]) IN arguments such as task attributes, Expected Data Transfer Length for one or both transfer directions (the latter for bidirectional commands), and Task Tag (as part of the I_T_L_x nexus). The I_T_L nexus is derived by the initiator and target from the LUN field in the request and the I_T nexus is implicit in the session identification.

In addition, the SCSI-command PDU carries information required for the proper operation of the iSCSI protocol - the command sequence number (CmdSN) for the session and the expected status number (ExpStatSN) for the connection.

All or part of the SCSI output (write) data associated with the SCSI command may be sent as part of the SCSI-Command PDU as a data segment.

3.5.1.2. SCSI-Response

The SCSI-Response carries all the SCSI execute-command procedure call (see [SAM2]) OUT arguments and the SCSI execute-command procedure call return value.

The SCSI-Response contains the residual counts from the operation, if any, an indication of whether the counts represent an overflow or an underflow, and the SCSI status if the status is valid or a response code (a non-zero return value for the execute-command procedure call) if the status is not valid.

For a valid status that indicates that the command has been processed, but resulted in an exception (e.g., a SCSI CHECK CONDITION), the PDU data segment contains the associated sense data. The use of Autosense ([SAM2]) is REQUIRED by iSCSI.

Some data segment content may also be associated (in the data segment) with a non-zero response code.

In addition, the SCSI-Response PDU carries information required for the proper operation of the iSCSI protocol:

- The number of Data-In PDUs that a target has sent (to enable the initiator to check that all have arrived).
- StatSN - the Status Sequence Number on this connection.

Docker EX1014
Page 43 of 257

- ExpCmdSN - the next Expected Command Sequence Number at the target.
- MaxCmdSN - the maximum CmdSN acceptable at the target from this initiator.

3.5.1.3 Task Management Function Request

The Task Management function request provides an initiator with a way to explicitly control the execution of one or more SCSI Tasks or iSCSI functions. The PDU carries a function identifier (which task management function to perform) and enough information to unequivocally identify the task or task-set on which to perform the action, even if the task(s) to act upon has not yet arrived or has been discarded due to an error.

The referenced tag identifies an individual task if the function refers to an individual task.

The I_T_L nexus identifies task sets. In iSCSI the I_T_L nexus is identified by the LUN and the session identification (the session identifies an I_T nexus).

For task sets, the CmdSN of the Task Management function request helps identify the tasks upon which to act, namely all tasks associated with a LUN and having a CmdSN preceding the Task Management function request CmdSN.

For a Task Management function, the coordination between responses to the tasks affected and the Task Management function response is done by the target.

3.5.1.4. Task Management Function Response

The Task Management function response carries an indication of function completion for a Task Management function request including how it was completed (response and qualifier) and additional information for failure responses.

After the Task Management response indicates Task Management function completion, the initiator will not receive any additional responses from the affected tasks.

3.5.1.5. SCSI Data-Out and SCSI Data-In

SCSI Data-Out and SCSI Data-In are the main vehicles by which SCSI data payload is carried between initiator and target. Data payload is associated with a specific SCSI command through the Initiator Task Tag. For target convenience, outgoing solicited data also carries a

Target Transfer Tag (copied from R2T) and the LUN. Each PDU contains the payload length and the data offset relative to the buffer address contained in the SCSI execute command procedure call.

In each direction, the data transfer is split into "sequences". An end-of-sequence is indicated by the F bit.

An outgoing sequence is either unsolicited (only the first sequence can be unsolicited) or consists of all the Data-Out PDUs sent in response to an R2T.

Input sequences are built to enable the direction switching for bidirectional commands.

For input, the target may request positive acknowledgement of input data. This is limited to sessions that support error recovery and is implemented through the A bit in the SCSI Data-In PDU header.

Data-In and Data-Out PDUs also carry the DataSN to enable the initiator and target to detect missing PDUs (discarded due to an error).

In addition, StatSN is carried by the Data-In PDUs.

To enable a SCSI command to be processed while involving a minimum number of messages, the last SCSI Data-In PDU passed for a command may also contain the status if the status indicates termination with no exceptions (no sense or response involved).

3.5.1.6. Ready To Transfer (R2T)

R2T is the mechanism by which the SCSI target "requests" the initiator for output data. R2T specifies to the initiator the offset of the requested data relative to the buffer address from the execute command procedure call and the length of the solicited data.

To help the SCSI target associate the resulting Data-Out with an R2T, the R2T carries a Target Transfer Tag that will be copied by the initiator in the solicited SCSI Data-Out PDUs. There are no protocol specific requirements with regard to the value of these tags, but it is assumed that together with the LUN, they will enable the target to associate data with an R2T.

Docker EX1014
Page 45 of 257

R2T also carries information required for proper operation of the iSCSI protocol, such as:

- R2TSN (to enable an initiator to detect a missing R2T)
- StatSN
- ExpCmdSN
- MaxCmdSN

3.5.2. Requests/Responses carrying SCSI and iSCSI Payload

3.5.2.1. Asynchronous Message

Asynchronous Messages are used to carry SCSI asynchronous events (AEN) and iSCSI asynchronous messages.

When carrying an AEN, the event details are reported as sense data in the data segment.

3.5.3. Requests/Responses Carrying iSCSI Only Payload

3.5.3.1. Text Request and Text Response

Text requests and responses are designed as a parameter negotiation vehicle and as a vehicle for future extension.

In the data segment, Text Requests/Responses carry text information using a simple "key=value" syntax.

Text Request/Responses may form extended sequences using the same Initiator Task Tag. The initiator uses the F (Final) flag bit in the text request header to indicate its readiness to terminate a sequence. The target uses the F (Final) flag bit in the text response header to indicate its consent to sequence termination.

Text Request and Responses also use the Target Transfer Tag to indicate continuation of an operation or a new beginning. A target that wishes to continue an operation will set the Target Transfer Tag in a Text Response to a value different from the default 0xffffffff. An initiator willing to continue will copy this value into the Target Transfer Tag of the next Text Request. If the initiator wants to restart the current target negotiation (start fresh) will set the Target Transfer Tag to 0xffffffff.

Although a complete exchange is always started by the initiator, specific parameter negotiations may be initiated by the initiator or target.

3.5.3.2. Login Request and Login Response

Login Requests and Responses are used exclusively during the Login Phase of each connection to set up the session and connection parameters. (The Login Phase consists of a sequence of login requests and responses carrying the same Initiator Task Tag.)

A connection is identified by an arbitrarily selected connection-ID (CID) that is unique within a session.

Similar to the Text Requests and Responses, Login Requests/Responses carry key=value text information with a simple syntax in the data segment.

The Login Phase proceeds through several stages (security negotiation, operational parameter negotiation) that are selected with two binary coded fields in the header -- the "current stage" (CSG) and the "next stage" (NSG) with the appearance of the latter being signaled by the "transit" flag (T).

The first Login Phase of a session plays a special role, called the leading login, which determines some header fields (e.g., the version number, the maximum number of connections, and the session identification).

The CmdSN initial value is also set by the leading login.

StatSN for each connection is initiated by the connection login.

A login request may indicate an implied logout (cleanup) of the connection to be logged in (a connection restart) by using the same Connection ID (CID) as an existing connection, as well as the same session identifying elements of the session to which the old connection was associated.

3.5.3.3. Logout Request and Response

Logout Requests and Responses are used for the orderly closing of connections for recovery or maintenance. The logout request may be issued following a target prompt (through an asynchronous message) or at an initiators initiative. When issued on the connection to be logged out, no other request may follow it.

The Logout Response indicates that the connection or session cleanup is completed and no other responses will arrive on the connection (if received on the logging out connection). In addition, the Logout Response indicates how long the target will continue to hold resources for recovery (e.g., command execution that continues on a

Docker EX1014
Page 47 of 257

new connection) in the text key Time2Retain and how long the initiator must wait before proceeding with recovery in the text key Time2Wait.

3.5.3.4. SNACK Request

With the SNACK Request, the initiator requests retransmission of numbered-responses or data from the target. A single SNACK request covers a contiguous set of missing items, called a run, of a given type of items. The type is indicated in a type field in the PDU header. The run is composed of an initial item (StatSN, DataSN, R2TSN) and the number of missed Status, Data, or R2T PDUs. For long Data-In sequences, the target may request (at predefined minimum intervals) a positive acknowledgement for the data sent. A SNACK request with a type field that indicates ACK and the number of Data-In PDUs acknowledged conveys this positive acknowledgement.

3.5.3.5. Reject

Reject enables the target to report an iSCSI error condition (e.g., protocol, unsupported option) that uses a Reason field in the PDU header and includes the complete header of the bad PDU in the Reject PDU data segment.

3.5.3.6. NOP-Out Request and NOP-In Response

This request/response pair may be used by an initiator and target as a "ping" mechanism to verify that a connection/session is still active and all of its components are operational. Such a ping may be triggered by the initiator or target. The triggering party indicates that it wants a reply by setting a value different from the default 0xffffffff in the corresponding Initiator/Target Transfer Tag.

NOP-In/NOP-Out may also be used "unidirectional" to convey to the initiator/target command, status or data counter values when there is no other "carrier" and there is a need to update the initiator/target.

4. SCSI Mode Parameters for iSCSI

There are no iSCSI specific mode pages.

5. Login and Full Feature Phase Negotiation

iSCSI parameters are negotiated at session or connection establishment by using Login Requests and Responses (see Section 3.2.3 iSCSI Login) and during the Full Feature Phase (Section 3.2.4 iSCSI Full Feature Phase) by using Text Requests and Responses. In

both cases the mechanism used is an exchange of iSCSI-text-key=value pairs. For brevity iSCSI-text-keys are called just keys in the rest of this document.

Keys are either declarative or require negotiation and the key description indicates if the key is declarative or requires negotiation.

For the declarative keys, the declaring party sets a value for the key. The key specification indicates if the key can be declared by the initiator, target or both.

For the keys that require negotiation one of the parties (the proposing party) proposes a value or set of values by including the key=value in the data part of a Login or Text Request or Response PDUs. The other party (the accepting party) makes a selection based on the value or list of values proposed and includes the selected value in a key=value in the data part of one of the following Login or Text Response or Request PDUs. For most of the keys both the initiator and target can be proposing parties.

The login process proceeds in two stages - the security negotiation stage and the operational parameter negotiation stage. Both stages are optional but at least one of them has to be present to enable the setting of some mandatory parameters.

If present, the security negotiation stage precedes the operational parameter negotiation stage.

Progression from stage to stage is controlled by the T (Transition) bit in the Login Request/Response PDU header. Through the T bit set to 1, the initiator indicates that it would like to transition. The target agrees to the transition (and selects the next stage) when ready. A field in the Login PDU header indicates the current stage (CSG) and during transition, another field indicates the next stage (NSG) proposed (initiator) and selected (target).

The text negotiation process is used to negotiate or declare operational parameters. The negotiation process is controlled by the F (final) bit in the PDU header. During text negotiations, the F bit is used by the initiator to indicate that it is ready to finish the negotiation and by the Target to acquiesce the end of negotiation.

Since some key=value pairs may not fit entirely in a single PDU, the C (continuation) bit is used (both in Login and Text) to indicate that "more follows".

Docker EX1014
Page 49 of 257

The text negotiation uses an additional mechanism by which a target may deliver larger amounts of data to an enquiring initiator. The target sets a Target Task Tag to be used as a bookmark that when returned by the initiator, means "go on". If reset to a "neutral value", it means "forget about the rest".

This chapter details types of keys and values used, the syntax rules for parameter formation, and the negotiation schemes to be used with different types of parameters.

5.1. Text Format

The initiator and target send a set of key=value pairs encoded in UTF-8 Unicode. All the text keys and text values specified in this document are to be presented and interpreted in the case in which they appear in this document. They are case sensitive.

The following character symbols are used in this document for text items (the hexadecimal values represent Unicode code points):

```
(a-z, A-Z) - letters
(0-9) - digits
" "  (0x20) - space
"."  (0x2e) - dot
"-"  (0x2d) - minus
"+"  (0x2b) - plus
"@"  (0x40) - commercial at
"_"  (0x5f) - underscore
"="  (0x3d) - equal
":"  (0x3a) - colon
"/"  (0x2f) - solidus or slash
"["  (0x5b) - left bracket
"]"  (0x5d) - right bracket
null (0x00) - null separator
","  (0x2c) - comma
"~"  (0x7e) - tilde
```

Key=value pairs may span PDU boundaries. An initiator or target that sends partial key=value text within a PDU indicates that more text follows by setting the C bit in the Text or Login Request or Text or Login Response to 1. Data segments in a series of PDUs that have the C bit set to 1 and end with a PDU that have the C bit set to 0, or include a single PDU that has the C bit set to 0, have to be considered as forming a single logical-text-data-segment (LTDS).

Every key=value pair, including the last or only pair in a LTDS, MUST be followed by one null (0x00) delimiter.

A key-name is whatever precedes the first "=" in the key=value pair. The term key is used frequently in this document in place of key-name.

A value is whatever follows the first "=" in the key=value pair up to the end of the key=value pair, but not including the null delimiter.

The following definitions will be used in the rest of this document:

standard-label: A string of one or more characters that consist of letters, digits, dot, minus, plus, commercial at, or underscore. A standard-label MUST begin with a capital letter and must not exceed 63 characters.

key-name: A standard-label.

text-value: A string of zero or more characters that consist of letters, digits, dot, minus, plus, commercial at, underscore, slash, left bracket, right bracket, or colon.

iSCSI-name-value: A string of one or more characters that consist of minus, dot, colon, or any character allowed by the output of the iSCSI string-prep template as specified in [RFC3722] (see also Section 3.2.6.2 iSCSI Name Encoding).

iSCSI-local-name-value: A UTF-8 string; no null characters are allowed in the string. This encoding is to be used for localized (internationalized) aliases.

boolean-value: The string "Yes" or "No".

hex-constant: A hexadecimal constant encoded as a string that starts with "0x" or "0X" followed by one or more digits or the letters a, b, c, d, e, f, A, B, C, D, E, or F. Hex-constants are used to encode numerical values or binary strings. When used to encode numerical values, the excessive use of leading 0 digits is discouraged. The string following 0X (or 0x) represents a base16 number that starts with the most significant base16 digit, followed by all other digits in decreasing order of significance and ending with the least-significant base16 digit. When used to encode binary strings, hexadecimal constants have an implicit byte-length that includes four bits for every hexadecimal digit of the constant, including leading zeroes. For example, a hex-constant of n hexadecimal digits has a byte-length of (the integer part of) (n+1)/2.

decimal-constant: An unsigned decimal number with the digit 0 or a string of one or more digits that start with a non-zero digit. Decimal-constants are used to encode numerical values or binary strings. Decimal constants can only be used to encode binary strings if the string length is explicitly specified. There is no implicit length for decimal strings. Decimal-constant MUST NOT be used for parameter values if the values can be equal or greater than 2**64 (numerical) or for binary strings that can be longer than 64 bits.

base64-constant: base64 constant encoded as a string that starts with "0b" or "0B" followed by 1 or more digits or letters or plus or slash or equal. The encoding is done according to [RFC2045] and each character, except equal, represents a base64 digit or a 6-bit binary string. Base64-constants are used to encode numerical-values or binary strings. When used to encode numerical values, the excessive use of leading 0 digits (encoded as A) is discouraged. The string following 0B (or 0b) represents a base64 number that starts with the most significant base64 digit, followed by all other digits in decreasing order of significance and ending with the least-significant base64 digit; the least significant base64 digit may be optionally followed by pad digits (encoded as equal) that are not considered as part of the number. When used to encode binary strings, base64-constants have an implicit
byte-length that includes six bits for every character of the constant, excluding trailing equals (i.e., a base64-constant of n base64 characters excluding the trailing equals has a byte-length of ((the integer part of) (n*3/4)). Correctly encoded base64 strings cannot have n values of 1, 5 ... k*4+1.

numerical-value: An unsigned integer always less than 2**64 encoded as a decimal-constant or a hex-constant. Unsigned integer arithmetic applies to numerical-values.

large-numerical-value: An unsigned integer that can be larger than or equal to 2**64 encoded as a hex constant, or base64-constant. Unsigned integer arithmetic applies to large-numeric-values.

numeric-range: Two numerical-values separated by a tilde where the value to the right of tilde must not be lower than the value to the left.

regular-binary-value: A binary string not longer than 64 bits encoded as a decimal constant, hex constant, or base64-constant. The length of the string is either specified by the key definition or is the implicit byte-length of the encoded string.

large-binary-value: A binary string longer than 64 bits encoded as a hex-constant or base64-constant. The length of the string is either specified by the key definition or is the implicit byte-length of the encoded string.

binary-value: A regular-binary-value or a large-binary-value. Operations on binary values are key specific.

simple-value: Text-value, iSCSI-name-value, boolean-value, numeric-value, a numeric-range, or a binary-value.

list-of-values: A sequence of text-values separated by a comma.

If not otherwise specified, the maximum length of a simple-value (not its encoded representation) is 255 bytes, not including the delimiter (comma or zero byte).

Any iSCSI target or initiator MUST support receiving at least 8192 bytes of key=value data in a negotiation sequence. When proposing or accepting authentication methods that explicitly require support for very long authentication items, the initiator and target MUST support receiving of at least 64 kilobytes of key=value data (see Appendix 11.1.2 - Simple Public-Key Mechanism (SPKM) - that require support for public key certificates).

5.2. Text Mode Negotiation

During login, and thereafter, some session or connection parameters are either declared or negotiated through an exchange of textual information.

The initiator starts the negotiation and/or declaration through a Text or Login Request and indicates when it is ready for completion (by setting the F bit to 1 and keeping it to 1 in a Text Request or the T bit in the Login Request). As negotiation text may span PDU boundaries, a Text or Login Request or Text or Login Response PDU that has the C bit set to 1 MUST NOT have the F/T bit set to 1.

A target receiving a Text or Login Request with the C bit set to 1 MUST answer with a Text or Login Response with no data segment (DataSegmentLength 0). An initiator receiving a Text or Login Response with the C bit set to 1 MUST answer with a Text or Login Request with no data segment (DataSegmentLength 0).

A target or initiator SHOULD NOT use a Text or Login Response or Text or Login Request with no data segment (DataSegmentLength 0) unless explicitly required by a general or a key-specific negotiation rule.

The format of a declaration is:

```
Declarer-> <key>=<valuex>
```

The general format of text negotiation is:

```
Proposer-> <key>=<valuex>
Acceptor-> <key>={<valuey>|NotUnderstood|Irrelevant|Reject}
```

Thus a declaration is a one-way textual exchange while a negotiation is a two-way exchange.

The proposer or declarer can either be the initiator or the target, and the acceptor can either be the target or initiator, respectively. Targets are not limited to respond to key=value pairs as proposed by the initiator. The target may propose key=value pairs of its own.

All negotiations are explicit (i.e., the result MUST only be based on newly exchanged or declared values). There are no implicit proposals. If a proposal is not made, then a reply cannot be expected. Conservative design also requires that default values should not be relied upon when use of some other value has serious consequences.

The value proposed or declared can be a numerical-value, a numerical-range defined by lower and upper values with both integers separated by a tilde, a binary value, a text-value, an iSCSI-name-value, an iSCSI-local-name-value, a boolean-value (Yes or No), or a list of comma separated text-values. A range, a large-numerical-value, an iSCSI-name-value and an iSCSI-local-name-value MAY ONLY be used if it is explicitly allowed. An accepted value can be a numerical-value, a large-numerical-value, a text-value, or a boolean-value.

If a specific key is not relevant for the current negotiation, the acceptor may answer with the constant "Irrelevant" for all types of negotiation. However the negotiation is not considered as failed if the answer is "Irrelevant". The "Irrelevant" answer is meant for those cases in which several keys are presented by a proposing party but the selection made by the acceptor for one of the keys makes other keys irrelevant. The following example illustrates the use of "Irrelevant":

```
I->T OFMarker=Yes,OFMarkInt=2048~8192
T->I OFMarker=No,OFMarkInt=Irrelevant

I->T X#vkey1=(bla,alb,None),X#vkey2=(bla,alb)
T->I X#vkey1=None,X#vkey2=Irrelevant
```


Docker EX1014
Page 54 of 257

Any key not understood by the acceptor may be ignored by the acceptor without affecting the basic function. However, the answer for a key not understood MUST be key=NotUnderstood.

The constants "None", "Reject", "Irrelevant", and "NotUnderstood" are reserved and MUST ONLY be used as described here. Violation of this rule is a protocol error (in particular the use of "Reject", "Irrelevant", and "NotUnderstood" as proposed values).

Reject or Irrelevant are legitimate negotiation options where allowed but their excessive use is discouraged. A negotiation is considered complete when the acceptor has sent the key value pair even if the value is "Reject", "Irrelevant", or "NotUnderstood. Sending the key again would be a re-negotiation and is forbidden for many keys.

If the acceptor sends "Reject" as an answer the negotiated key is left at its current value (or default if no value was set). If the current value is not acceptable to the proposer on the connection or to the session it is sent, the proposer MAY choose to terminate the connection or session.

All keys in this document, except for the X extension formats, MUST be supported by iSCSI initiators and targets when used as specified here. If used as specified, these keys MUST NOT be answered with NotUnderstood.

Implementers may introduce new keys by prefixing them with "X-", followed by their (reversed) domain name, or with new keys registered with IANA prefixing them with X#. For example, the entity owning the domain example.com can issue:

```
X-com.example.bar.foo.do_something=3
```

or a new registered key may be used as in:

```
X#SuperCalyPhraGilistic=Yes
```

Implementers MAY also introduce new values, but ONLY for new keys or authentication methods (see Section 11 iSCSI Security Text Keys and Authentication Methods), or digests (see Section 12.1 HeaderDigest and DataDigest).

Whenever parameter action or acceptance is dependent on other parameters, the dependency rules and parameter sequence must be specified with the parameters.

Docker EX1014
Page 55 of 257

In the Login Phase (see Section 5.3 Login Phase), every stage is a separate negotiation. In the FullFeaturePhase, a Text Request Response sequence is a negotiation. Negotiations MUST be handled as atomic operations. For example, all negotiated values go into effect after the negotiation concludes in agreement or are ignored if the negotiation fails.

Some parameters may be subject to integrity rules (e.g., parameter-x must not exceed parameter-y or parameter-u not 1 implies parameter-v be Yes). Whenever required, integrity rules are specified with the keys. Checking for compliance with the integrity rule must only be performed after all the parameters are available (the existent and the newly negotiated). An iSCSI target MUST perform integrity checking before the new parameters take effect. An initiator MAY perform integrity checking.

An iSCSI initiator or target MAY terminate a negotiation that does not end within a reasonable time or number of exchanges.

5.2.1. List negotiations

In list negotiation, the originator sends a list of values (which may include "None") in its order of preference.

The responding party MUST respond with the same key and the first value that it supports (and is allowed to use for the specific originator) selected from the originator list.

The constant "None" MUST always be used to indicate a missing function. However, "None" is only a valid selection if it is explicitly proposed.

If an acceptor does not understand any particular value in a list, it MUST ignore it. If an acceptor does not support, does not understand, or is not allowed to use any of the proposed options with a specific originator, it may use the constant "Reject" or terminate the negotiation. The selection of a value not proposed MUST be handled as a protocol error.

5.2.2. Simple-value Negotiations

For simple-value negotiations, the accepting party MUST answer with the same key. The value it selects becomes the negotiation result.

Proposing a value not admissible (e.g., not within the specified bounds) MAY be answered with the constant "Reject" or the acceptor MAY select an admissible value.

Docker EX1014
Page 56 of 257

The selection by the acceptor, of a value not admissible under the selection rules is considered a protocol error. The selection rules are key-specific.

For a numerical range, the value selected must be an integer within the proposed range or "Reject" (if the range is unacceptable).

In Boolean negotiations (i.e., those that result in keys taking the values Yes or No), the accepting party MUST answer with the same key and the result of the negotiation when the received value does not determine that result by itself. The last value transmitted becomes the negotiation result. The rules for selecting the value to answer with are expressed as Boolean functions of the value received, and the value that the accepting party would have selected if given a choice.

Specifically, the two cases in which answers are OPTIONAL are:

- The Boolean function is "AND" and the value "No" is received. The outcome of the negotiation is "No".
- The Boolean function is "OR" and the value "Yes" is received. The outcome of the negotiation is "Yes".

Responses are REQUIRED in all other cases, and the value chosen and sent by the acceptor becomes the outcome of the negotiation.

5.3. Login Phase

The Login Phase establishes an iSCSI connection between an initiator and a target; it also creates a new session or associates the connection to an existing session. The Login Phase sets the iSCSI protocol parameters, security parameters, and authenticates the initiator and target to each other.

The Login Phase is only implemented via Login Request and Responses. The whole Login Phase is considered as a single task and has a single Initiator Task Tag (similar to the linked SCSI commands).

The default MaxRecvDataSegmentLength is used during Login.

The Login Phase sequence of requests and responses proceeds as follows:

- Login initial request
- Login partial response (optional)
- More Login Requests and Responses (optional)
- Login Final-Response (mandatory)

Docker EX1014
Page 57 of 257

The initial Login Request of any connection MUST include the InitiatorName key=value pair. The initial Login Request of the first connection of a session MAY also include the SessionType key=value pair. For any connection within a session whose type is not "Discovery", the first Login Request MUST also include the TargetName key=value pair.

The Login Final-response accepts or rejects the Login Request.

The Login Phase MAY include a SecurityNegotiation stage and a LoginOperationalNegotiation stage or both, but MUST include at least one of them. The included stage MAY be empty except for the mandatory names.

The Login Requests and Responses contain a field (CSG) that indicates the current negotiation stage (SecurityNegotiation or LoginOperationalNegotiation). If both stages are used, the SecurityNegotiation MUST precede the LoginOperationalNegotiation.

Some operational parameters can be negotiated outside the login through Text Requests and Responses.

Security MUST be completely negotiated within the Login Phase. The use of underlying IPsec security is specified in Chapter 8 and in [RFC3723]. iSCSI support for security within the protocol only consists of authentication in the Login Phase.

In some environments, a target or an initiator is not interested in authenticating its counterpart. It is possible to bypass authentication through the Login Request and Response.

The initiator and target MAY want to negotiate iSCSI authentication parameters. Once this negotiation is completed, the channel is considered secure.

Most of the negotiation keys are only allowed in a specific stage. The SecurityNegotiation keys appear in Chapter 11 and the LoginOperationalNegotiation keys appear in Chapter 12. Only a limited set of keys (marked as Any-Stage in Chapter 12) may be used in any of the two stages.

Any given Login Request or Response belongs to a specific stage; this determines the negotiation keys allowed with the request or response. It is considered to be a protocol error to send a key that is not allowed in the current stage.

Docker EX1014
Page 58 of 257

Stage transition is performed through a command exchange (request/response) that carries the T bit and the same CSG code. During this exchange, the next stage is selected by the target through the "next stage" code (NSG). The selected NSG MUST NOT exceed the value stated by the initiator. The initiator can request a transition whenever it is ready, but a target can only respond with a transition after one is proposed by the initiator.

In a negotiation sequence, the T bit settings in one pair of Login Request-Responses have no bearing on the T bit settings of the next pair. An initiator that has a T bit set to 1 in one pair and is answered with a T bit setting of 0, may issue the next request with the T bit set to 0.

When a transition is requested by the initiator and acknowledged by the target, both the initiator and target switch to the selected stage.

Targets MUST NOT submit parameters that require an additional initiator Login Request in a Login Response with the T bit set to 1.

Stage transitions during login (including entering and exit) are only possible as outlined in the following table:

From To ->	Security	Operational	FullFeature
(start)	yes	yes	no
Security	no	yes	yes
Operational	no	no	yes

The Login Final-Response that accepts a Login Request can only come as a response to a Login Request with the T bit set to 1, and both the request and response MUST indicate FullFeaturePhase as the next phase via the NSG field.

Neither the initiator nor the target should attempt to declare or negotiate a parameter more than once during login except for responses to specific keys that explicitly allow repeated key declarations (e.g., TargetAddress). An attempt to renegotiate/redeclare parameters not specifically allowed MUST be detected by the initiator and target. If such an attempt is detected

Docker EX1014
Page 59 of 257

by the target, the target MUST respond with Login reject (initiator error); if detected by the initiator, the initiator MUST drop the connection.

5.3.1. Login Phase Start

The Login Phase starts with a Login Request from the initiator to the target. The initial Login Request includes:

- Protocol version supported by the initiator.
- iSCSI Initiator Name and iSCSI Target Name
- ISID, TSIH, and connection Ids
- Negotiation stage that the initiator is ready to enter.

A login may create a new session or it may add a connection to an existing session. Between a given iSCSI Initiator Node (selected only by an InitiatorName) and a given iSCSI target defined by an iSCSI TargetName and a Target Portal Group Tag, the login results are defined by the following table:

ISID	TSIH	CID	Target action
new	non-zero	any	fail the login ("session does not exist")
new	zero	any	instantiate a new session
existing	zero	any	do session reinstatement (see section 5.3.5)
existing	non-zero existing	new	add a new connection to the session
existing	non-zero existing	existing	do connection reinstatement (see section 5.3.4)
existing	non-zero new	any	fail the login ("session does not exist")

Determination of "existing" or "new" are made by the target.

Optionally, the Login Request may include:

- Security parameters
OR
- iSCSI operational parameters
AND/OR
- The next negotiation stage that the initiator is ready to enter.

The target can answer the login in the following ways:

- Login Response with Login reject. This is an immediate rejection from the target that causes the connection to terminate and the session to terminate if this is the first (or only) connection of a new session. The T bit and the CSG and NSG fields are reserved.
- Login Response with Login Accept as a final response (T bit set to 1 and the NSG in both request and response are set to FullFeaturePhase). The response includes the protocol version supported by the target and the session ID, and may include iSCSI operational or security parameters (that depend on the current stage).
- Login Response with Login Accept as a partial response (NSG not set to FullFeaturePhase in both request and response) that indicates the start of a negotiation sequence. The response includes the protocol version supported by the target and either security or iSCSI parameters (when no security mechanism is chosen) supported by the target.

If the initiator decides to forego the SecurityNegotiation stage, it issues the Login with the CSG set to LoginOperationalNegotiation and the target may reply with a Login Response that indicates that it is unwilling to accept the connection (see Section 10.13 Login Response) without SecurityNegotiation and will terminate the connection with a response of Authentication failure (see Section 10.13.5 Status-Class and Status-Detail).

If the initiator is willing to negotiate iSCSI security, but is unwilling to make the initial parameter proposal and may accept a connection without iSCSI security, it issues the Login with the T bit set to 1, the CSG set to SecurityNegotiation, and the NSG set to LoginOperationalNegotiation. If the target is also ready to skip security, the Login Response only contains the TargetPortalGroupTag key (see Section 12.9 TargetPortalGroupTag), the T bit set to 1, the CSG set to SecurityNegotiation, and the NSG set to LoginOperationalNegotiation.

Docker EX1014
Page 61 of 257

An initiator that chooses to operate without iSCSI security, with all the operational parameters taking the default values, issues the Login with the T bit set to 1, the CSG set to LoginOperationalNegotiation, and the NSG set to FullFeaturePhase. If the target is also ready to forego security and can finish its LoginOperationalNegotiation, the Login Response has T bit set to 1, the CSG set to LoginOperationalNegotiation, and the NSG set to FullFeaturePhase in the next stage.

During the Login Phase the iSCSI target MUST return the TargetPortalGroupTag key with the first Login Response PDU with which it is allowed to do so (i.e., the first Login Response issued after the first Login Request with the C bit set to 0) for all session types when TargetName is given and the response is not a redirection. The TargetPortalGroupTag key value indicates the iSCSI portal group servicing the Login Request PDU. If the reconfiguration of iSCSI portal groups is a concern in a given environment, the iSCSI initiator should use this key to ascertain that it had indeed initiated the Login Phase with the intended target portal group.

5.3.2. iSCSI Security Negotiation

The security exchange sets the security mechanism and authenticates the initiator user and the target to each other. The exchange proceeds according to the authentication method chosen in the negotiation phase and is conducted using the Login Requests' and responses' key=value parameters.

An initiator directed negotiation proceeds as follows:

- The initiator sends a Login Request with an ordered list of the options it supports (authentication algorithm). The options are listed in the initiator's order of preference. The initiator MAY also send private or public extension options.

- The target MUST reply with the first option in the list it supports and is allowed to use for the specific initiator unless it does not support any, in which case it MUST answer with "Reject" (see Section 5.2 Text Mode Negotiation). The parameters are encoded in UTF8 as key=value. For security parameters, see Chapter 11.

- When the initiator considers that it is ready to conclude the SecurityNegotiation stage, it sets the T bit to 1 and the NSG to what it would like the next stage to be. The target will then set the T bit to 1 and set the NSG to the next stage in the Login Response when it finishes sending its security keys. The next

Docker EX1014
Page 62 of 257

stage selected will be the one the target selected. If the next stage is FullFeaturePhase, the target MUST respond with a Login Response with the TSIH value.

If the security negotiation fails at the target, then the target MUST send the appropriate Login Response PDU. If the security negotiation fails at the initiator, the initiator SHOULD close the connection.

It should be noted that the negotiation might also be directed by the target if the initiator does support security, but is not ready to direct the negotiation (propose options).

5.3.3. Operational Parameter Negotiation During the Login Phase

Operational parameter negotiation during the login MAY be done:

- Starting with the first Login Request if the initiator does not propose any security/integrity option.

- Starting immediately after the security negotiation if the initiator and target perform such a negotiation.

Operational parameter negotiation MAY involve several Login Request-Response exchanges started and terminated by the initiator. The initiator MUST indicate its intent to terminate the negotiation by setting the T bit to 1; the target sets the T bit to 1 on the last response.

If the target responds to a Login Request that has the T bit set to 1 with a Login Response that has the T bit set to 0, the initiator should keep sending the Login Request (even empty) with the T bit set to 1, while it still wants to switch stage, until it receives the Login Response that has the T bit set to 1 or it receives a key that requires it to set the T bit to 0.

Some session specific parameters can only be specified during the Login Phase of the first connection of a session (i.e., begun by a Login Request that contains a zero-valued TSIH) - the leading Login Phase (e.g., the maximum number of connections that can be used for this session).

A session is operational once it has at least one connection in FullFeaturePhase. New or replacement connections can only be added to a session after the session is operational.

For operational parameters, see Chapter 12.

Docker EX1014
Page 63 of 257

5.3.4. Connection Reinstatement

Connection reinstatement is the process of an initiator logging in with an ISID-TSIH-CID combination that is possibly active from the target's perspective, which causes the implicit logging out of the connection corresponding to the CID, and reinstating a new Full Feature Phase iSCSI connection in its place (with the same CID). Thus, the TSIH in the Login PDU MUST be non-zero and the CID does not change during a connection reinstatement. The Login Request performs the logout function of the old connection if an explicit logout was not performed earlier. In sessions with a single connection, this may imply the opening of a second connection with the sole purpose of cleaning up the first. Targets MUST support opening a second connection even when they do not support multiple connections in Full Feature Phase if ErrorRecoveryLevel is 2 and SHOULD support opening a second connection if ErrorRecoveryLevel is less than 2.

If the operational ErrorRecoveryLevel is 2, connection reinstatement enables future task reassignment. If the operational ErrorRecoveryLevel is less than 2, connection reinstatement is the replacement of the old CID without enabling task reassignment. In this case, all the tasks that were active on the old CID must be immediately terminated without further notice to the initiator.

The initiator connection state MUST be CLEANUP_WAIT (section 7.1.3) when the initiator attempts a connection reinstatement.

In practical terms, in addition to the implicit logout of the old connection, reinstatement is equivalent to a new connection login.

5.3.5. Session Reinstatement, Closure, and Timeout

Session reinstatement is the process of the initiator logging in with an ISID that is possibly active from the target's perspective. Thus implicitly logging out the session that corresponds to the ISID and reinstating a new iSCSI session in its place (with the same ISID). Therefore, the TSIH in the Login PDU MUST be zero to signal session reinstatement. Session reinstatement causes all the tasks that were active on the old session to be immediately terminated by the target without further notice to the initiator.

The initiator session state MUST be FAILED (Section 7.3 Session State Diagrams) when the initiator attempts a session reinstatement.

Docker EX1014
Page 64 of 257

Session closure is an event defined to be one of the following:

- A successful "session close" logout.
- A successful "connection close" logout for the last Full Feature Phase connection when no other connection in the session is waiting for cleanup (Section 7.2 Connection Cleanup State Diagram for Initiators and Targets) and no tasks in the session are waiting for reassignment.

Session timeout is an event defined to occur when the last connection state timeout expires and no tasks are waiting for reassignment. This takes the session to the FREE state (N6 transition in the session state diagram).

5.3.5.1. Loss of Nexus Notification

The iSCSI layer provides the SCSI layer with the "I_T nexus loss" notification when any one of the following events happens:

a) Successful completion of session reinstatement.
b) Session closure event.
c) Session timeout event.

Certain SCSI object clearing actions may result due to the notification in the SCSI end nodes, as documented in Appendix F. - Clearing Effects of Various Events on Targets -.

5.3.6. Session Continuation and Failure

Session continuation is the process by which the state of a preexisting session continues to be used by connection reinstatement (Section 5.3.4 Connection Reinstatement), or by adding a connection with a new CID. Either of these actions associates the new transport connection with the session state.

Session failure is an event where the last Full Feature Phase connection reaches the CLEANUP_WAIT state (Section 7.2 Connection Cleanup State Diagram for Initiators and Targets), or completes a successful recovery logout, thus causing all active tasks (that are formerly allegiant to the connection) to start waiting for task reassignment.

Docker EX1014
Page 65 of 257

5.4. Operational Parameter Negotiation Outside the Login Phase

Some operational parameters MAY be negotiated outside (after) the Login Phase.

Parameter negotiation in Full Feature Phase is done through Text requests and responses. Operational parameter negotiation MAY involve several Text request-response exchanges, which the initiator always starts and terminates using the same Initiator Task Tag. The initiator MUST indicate its intent to terminate the negotiation by setting the F bit to 1; the target sets the F bit to 1 on the last response.

If the target responds to a Text request with the F bit set to 1 and with a Text response with the F bit set to 0, the initiator should keep sending the Text request (even empty) with the F bit set to 1, while it still wants to finish the negotiation, until it receives the Text response with the F bit set to 1. Responding to a Text request with the F bit set to 1 with an empty (no key=value pairs) response with the F bit set to 0 is discouraged.

Targets MUST NOT submit parameters that require an additional initiator Text request in a Text response with the F bit set to 1.

In a negotiation sequence, the F bit settings in one pair of Text request-responses have no bearing on the F bit settings of the next pair. An initiator that has the F bit set to 1 in a request and is being answered with an F bit setting of 0 may issue the next request with the F bit set to 0.

Whenever the target responds with the F bit set to 0, it MUST set the Target Transfer Tag to a value other than the default 0xffffffff.

An initiator MAY reset an operational parameter negotiation by issuing a Text request with the Target Transfer Tag set to the value 0xffffffff after receiving a response with the Target Transfer Tag set to a value other than 0xffffffff. A target may reset an operational parameter negotiation by answering a Text request with a Reject PDU.

Neither the initiator nor the target should attempt to declare or negotiate a parameter more than once during any negotiation sequence without an intervening operational parameter negotiation reset, except for responses to specific keys that explicitly allow repeated key declarations (e.g., TargetAddress). If detected by the target, this MUST result in a Reject PDU with a reason of "protocol error". The initiator MUST reset the negotiation as outlined above.

Docker EX1014
Page 66 of 257

Parameters negotiated by a text exchange negotiation sequence only become effective after the negotiation sequence is completed.

6. iSCSI Error Handling and Recovery

6.1. Overview

6.1.1. Background

The following two considerations prompted the design of much of the error recovery functionality in iSCSI:

i) An iSCSI PDU may fail the digest check and be dropped, despite being received by the TCP layer. The iSCSI layer must optionally be allowed to recover such dropped PDUs.
ii) A TCP connection may fail at any time during the data transfer. All the active tasks must optionally be allowed to continue on a different TCP connection within the same session.

Implementations have considerable flexibility in deciding what degree of error recovery to support, when to use it and by which mechanisms to achieve the required behavior. Only the externally visible actions of the error recovery mechanisms must be standardized to ensure interoperability.

This chapter describes a general model for recovery in support of interoperability. See Appendix E. - Algorithmic Presentation of Error Recovery Classes - for further detail on how the described model may be implemented. Compliant implementations do not have to match the implementation details of this model as presented, but the external behavior of such implementations must correspond to the externally observable characteristics of the presented model.

6.1.2. Goals

The major design goals of the iSCSI error recovery scheme are as follows:

a) Allow iSCSI implementations to meet different requirements by defining a collection of error recovery mechanisms that implementations may choose from.
b) Ensure interoperability between any two implementations supporting different sets of error recovery capabilities.
c) Define the error recovery mechanisms to ensure command ordering even in the face of errors, for initiators that demand ordering.

Docker EX1014
Page 67 of 257

d) Do not make additions in the fast path, but allow moderate complexity in the error recovery path.
e) Prevent both the initiator and target from attempting to recover the same set of PDUs at the same time. For example, there must be a clear "error recovery functionality distribution" between the initiator and target.

6.1.3. Protocol Features and State Expectations

The initiator mechanisms defined in connection with error recovery are:

a) NOP-OUT to probe sequence numbers of the target (section 10.18)
b) Command retry (section 6.2.1)
c) Recovery R2T support (section 6.7)
d) Requesting retransmission of status/data/R2T using the SNACK facility (section 10.16)
e) Acknowledging the receipt of the data (section 10.16)
f) Reassigning the connection allegiance of a task to a different TCP connection (section 6.2.2)
g) Terminating the entire iSCSI session to start afresh (section 6.1.4.4)

The target mechanisms defined in connection with error recovery are:

a) NOP-IN to probe sequence numbers of the initiator (section 10.19)
b) Requesting retransmission of data using the recovery R2T feature (section 6.7)
c) SNACK support (section 10.16) d) Requesting that parts of read data be acknowledged (section 10.7.2)
e) Allegiance reassignment support (section 6.2.2)
f) Terminating the entire iSCSI session to force the initiator to start over (section 6.1.4.4)

For any outstanding SCSI command, it is assumed that iSCSI, in conjunction with SCSI at the initiator, is able to keep enough information to be able to rebuild the command PDU, and that outgoing data is available (in host memory) for retransmission while the command is outstanding. It is also assumed that at the target, incoming data (read data) MAY be kept for recovery or it can be reread from a device server.

It is further assumed that a target will keep the "status & sense" for a command it has executed if it supports status retransmission. A target that agrees to support data retransmission is expected to be prepared to retransmit the outgoing data (i.e., Data-In) on request

Docker EX1014
Page 68 of 257

until either the status for the completed command is acknowledged, or the data in question has been separately acknowledged.

6.1.4. Recovery Classes

iSCSI enables the following classes of recovery (in the order of increasing scope of affected iSCSI tasks):

- Within a command (i.e., without requiring command restart).
- Within a connection (i.e., without requiring the connection to be rebuilt, but perhaps requiring command restart).
- Connection recovery (i.e., perhaps requiring connections to be rebuilt and commands to be reissued).
- Session recovery.

The recovery scenarios detailed in the rest of this section are representative rather than exclusive. In every case, they detail the lowest class recovery that MAY be attempted. The implementer is left to decide under which circumstances to escalate to the next recovery class and/or what recovery classes to implement. Both the iSCSI target and initiator MAY escalate the error handling to an error recovery class, which impacts a larger number of iSCSI tasks in any of the cases identified in the following discussion.

In all classes, the implementer has the choice of deferring errors to the SCSI initiator (with an appropriate response code), in which case the task, if any, has to be removed from the target and all the side effects, such as ACA, must be considered.

Use of within-connection and within-command recovery classes MUST NOT be attempted before the connection is in Full Feature Phase.

In the detailed description of the recovery classes, the mandating terms (MUST, SHOULD, MAY, etc.) indicate normative actions to be executed if the recovery class is supported and used.

6.1.4.1. Recovery Within-command

At the target, the following cases lend themselves to within-command recovery:

- Lost data PDU - realized through one of the following:

  a) Data digest error - dealt with as specified in Section 6.7 Digest Errors, using the option of a recovery R2T.

Docker EX1014
Page 69 of 257

b) Sequence reception timeout (no data or partial-data-and-no-F-bit) - considered an implicit sequence error and dealt with as specified in Section 6.8 Sequence Errors, using the option of a recovery R2T.
c) Header digest error, which manifests as a sequence reception timeout or a sequence error - dealt with as specified in Section 6.8 Sequence Errors, using the option of a recovery R2T.

At the initiator, the following cases lend themselves to within-command recovery:

Lost data PDU or lost R2T - realized through one of the following:

a) Data digest error - dealt with as specified in Section 6.7 Digest Errors, using the option of a SNACK.
b) Sequence reception timeout (no status) or response reception timeout - dealt with as specified in Section 6.8 Sequence Errors, using the option of a SNACK.
c) Header digest error, which manifests as a sequence reception timeout or a sequence error - dealt with as specified in Section 6.8 Sequence Errors, using the option of a SNACK.

To avoid a race with the target, which may already have a recovery R2T or a termination response on its way, an initiator SHOULD NOT originate a SNACK for an R2T based on its internal timeouts (if any). Recovery in this case is better left to the target.

The timeout values used by the initiator and target are outside the scope of this document. Sequence reception timeout is generally a large enough value to allow the data sequence transfer to be complete.

6.1.4.2. Recovery Within-connection

At the initiator, the following cases lend themselves to within-connection recovery:

- Requests not acknowledged for a long time. Requests are acknowledged explicitly through ExpCmdSN or implicitly by receiving data and/or status. The initiator MAY retry non-acknowledged commands as specified in Section 6.2 Retry and Reassign in Recovery.

Docker EX1014
Page 70 of 257

- Lost iSCSI numbered Response. It is recognized by either identifying a data digest error on a Response PDU or a Data-In PDU carrying the status, or by receiving a Response PDU with a higher StatSN than expected. In the first case, digest error handling is done as specified in Section 6.7 Digest Errors using the option of a SNACK. In the second case, sequence error handling is done as specified in Section 6.8 Sequence Errors, using the option of a SNACK.

At the target, the following cases lend themselves to within-connection recovery:

- Status/Response not acknowledged for a long time. The target MAY issue a NOP-IN (with a valid Target Transfer Tag or otherwise) that carries the next status sequence number it is going to use in the StatSN field. This helps the initiator detect any missing StatSN(s) and issue a SNACK for the status.

The timeout values used by the initiator and the target are outside the scope of this document.

6.1.4.3. Connection Recovery

At an iSCSI initiator, the following cases lend themselves to connection recovery:

- TCP connection failure: The initiator MUST close the connection. It then MUST either implicitly or explicitly logout the failed connection with the reason code "remove the connection for recovery" and reassign connection allegiance for all commands still in progress associated with the failed connection on one or more connections (some or all of which MAY be newly established connections) using the "Task reassign" task management function (see Section 10.5.1 Function). For an initiator, a command is in progress as long as it has not received a response or a Data-In PDU including status.

  Note: The logout function is mandatory. However, a new connection establishment is only mandatory if the failed connection was the last or only connection in the session.

- Receiving an Asynchronous Message that indicates one or all connections in a session has been dropped. The initiator MUST handle it as a TCP connection failure for the connection(s) referred to in the Message.

Docker EX1014
Page 71 of 257

At an iSCSI target, the following cases lend themselves to connection recovery:

- TCP connection failure. The target MUST close the connection and, if more than one connection is available, the target SHOULD send an Asynchronous Message that indicates it has dropped the connection. Then, the target will wait for the initiator to continue recovery.

6.1.4.4. Session Recovery

Session recovery should be performed when all other recovery attempts have failed. Very simple initiators and targets MAY perform session recovery on all iSCSI errors and rely on recovery on the SCSI layer and above.

Session recovery implies the closing of all TCP connections, internally aborting all executing and queued tasks for the given initiator at the target, terminating all outstanding SCSI commands with an appropriate SCSI service response at the initiator, and restarting a session on a new set of connection(s) (TCP connection establishment and login on all new connections).

For possible clearing effects of session recovery on SCSI and iSCSI objects, refer to Appendix F. - Clearing Effects of Various Events on Targets -.

6.1.5. Error Recovery Hierarchy

The error recovery classes described so far are organized into a hierarchy for ease in understanding and to limit the implementation complexity. With few and well defined recovery levels interoperability is easier to achieve. The attributes of this hierarchy are as follows:

a) Each level is a superset of the capabilities of the previous level. For example, Level 1 support implies supporting all capabilities of Level 0 and more.
b) As a corollary, supporting a higher error recovery level means increased sophistication and possibly an increase in resource requirements.
c) Supporting error recovery level "n" is advertised and negotiated by each iSCSI entity by exchanging the text key "ErrorRecoveryLevel=n". The lower of the two exchanged values is the operational ErrorRecoveryLevel for the session.

Docker EX1014
Page 72 of 257

The following diagram represents the error recovery hierarchy.

```
                      +
                     /
                    / 2 \       <-- Connection recovery
                   +-----+
                  /   1   \     <-- Digest failure recovery
                 +---------+
                /     0     \   <-- Session failure recovery
               +-------------+
```

The following table lists the error recovery capabilities expected from the implementations that support each error recovery level.

ErrorRecoveryLevel	Associated Error recovery capabilities
0	Session recovery class (Section 6.1.4.4 Session Recovery)
1	Digest failure recovery (See Note below.) plus the capabilities of ER Level 0
2	Connection recovery class (Section 6.1.4.3 Connection Recovery) plus the capabilities of ER Level 1

Note: Digest failure recovery is comprised of two recovery classes: Within-Connection recovery class (Section 6.1.4.2 Recovery Within-connection) and Within-Command recovery class (Section 6.1.4.1 Recovery Within-command).

When a defined value of ErrorRecoveryLevel is proposed by an originator in a text negotiation, the originator MUST support the functionality defined for the proposed value and additionally, the functionality corresponding to any defined value numerically less than the proposed. When a defined value of ErrorRecoveryLevel is returned by a responder in a text negotiation, the responder MUST support the functionality corresponding to the ErrorRecoveryLevel it is accepting.

When either party attempts to use error recovery functionality beyond what is negotiated, the recovery attempts MAY fail unless an a priori agreement outside the scope of this document exists between the two parties to provide such support.

Docker EX1014
Page 73 of 257

Implementations MUST support error recovery level "0", while the rest are OPTIONAL to implement. In implementation terms, the above striation means that the following incremental sophistication with each level is required.

Level transition	Incremental requirement
0->1	PDU retransmissions on the same connection
1->2	Retransmission across connections and allegiance reassignment

6.2. Retry and Reassign in Recovery

This section summarizes two important and somewhat related iSCSI protocol features used in error recovery.

6.2.1. Usage of Retry

By resending the same iSCSI command PDU ("retry") in the absence of a command acknowledgement (by way of an ExpCmdSN update) or a response, an initiator attempts to "plug" (what it thinks are) the discontinuities in CmdSN ordering on the target end. Discarded command PDUs, due to digest errors, may have created these discontinuities.

Retry MUST NOT be used for reasons other than plugging command sequence gaps, and in particular, cannot be used for requesting PDU retransmissions from a target. Any such PDU retransmission requests for a currently allegiant command in progress may be made using the SNACK mechanism described in section 10.16, although the usage of SNACK is OPTIONAL.

If initiators, as part of plugging command sequence gaps as described above, inadvertently issue retries for allegiant commands already in progress (i.e., targets did not see the discontinuities in CmdSN ordering), the duplicate commands are silently ignored by targets as specified in section 3.2.2.1.

When an iSCSI command is retried, the command PDU MUST carry the original Initiator Task Tag and the original operational attributes (e.g., flags, function names, LUN, CDB etc.) as well as the original CmdSN. The command being retried MUST be sent on the same connection as the original command unless the original connection was already successfully logged out.

6.2.2. Allegiance Reassignment

By issuing a "task reassign" task management request (Section 10.5.1 Function), the initiator signals its intent to continue an already active command (but with no current connection allegiance) as part of connection recovery. This means that a new connection allegiance is requested for the command, which seeks to associate it to the connection on which the task management request is being issued. Before the allegiance reassignment is attempted for a task, an implicit or explicit Logout with the reason code "remove the connection for recovery" (see section 10.14) MUST be successfully completed for the previous connection to which the task was allegiant.

In reassigning connection allegiance for a command, the targets SHOULD continue the command from its current state. For example, when reassigning read commands, the target SHOULD take advantage of the ExpDataSN field provided by the Task Management function request (which must be set to zero if there was no data transfer) and bring the read command to completion by sending the remaining data and sending (or resending) the status. ExpDataSN acknowledges all data sent up to, but not including, the Data-In PDU and or R2T with DataSN (or R2TSN) equal to ExpDataSN. However, targets may choose to send/receive all unacknowledged data or all of the data on a reassignment of connection allegiance if unable to recover or maintain an accurate state. Initiators MUST not subsequently request data retransmission through Data SNACK for PDUs numbered less than ExpDataSN (i.e., prior to the acknowledged sequence number). For all types of commands, a reassignment request implies that the task is still considered in progress by the initiator and the target must conclude the task appropriately if the target returns the "Function Complete" response to the reassignment request. This might possibly involve retransmission of data/R2T/status PDUs as necessary, but MUST involve the (re)transmission of the status PDU.

It is OPTIONAL for targets to support the allegiance reassignment. This capability is negotiated via the ErrorRecoveryLevel text key during the login time. When a target does not support allegiance reassignment, it MUST respond with a Task Management response code of "Allegiance reassignment not supported". If allegiance reassignment is supported by the target, but the task is still allegiant to a different connection, or a successful recovery Logout of the previously allegiant connection was not performed, the target MUST respond with a Task Management response code of "Task still allegiant".

Docker EX1014
Page 75 of 257

If allegiance reassignment is supported by the target, the Task Management response to the reassignment request MUST be issued before the reassignment becomes effective.

If a SCSI Command that involves data input is reassigned, any SNACK Tag it holds for a final response from the original connection is deleted and the default value of 0 MUST be used instead.

6.3. Usage Of Reject PDU in Recovery

Targets MUST NOT implicitly terminate an active task by sending a Reject PDU for any PDU exchanged during the life of the task. If the target decides to terminate the task, a Response PDU (SCSI, Text, Task, etc.) must be returned by the target to conclude the task. If the task had never been active before the Reject (i.e., the Reject is on the command PDU), targets should not send any further responses because the command itself is being discarded.

The above rule means that the initiator can eventually expect a response on receiving Rejects, if the received Reject is for a PDU other than the command PDU itself. The non-command Rejects only have diagnostic value in logging the errors, and they can be used for retransmission decisions by the initiators.

The CmdSN of the rejected command PDU (if it is a non-immediate command) MUST NOT be considered received by the target (i.e., a command sequence gap must be assumed for the CmdSN), even though the CmdSN of the rejected command PDU may be reliably ascertained. Upon receiving the Reject, the initiator MUST plug the CmdSN gap in order to continue to use the session. The gap may be plugged either by transmitting a command PDU with the same CmdSN, or by aborting the task (see section 6.9 on how an abort may plug a CmdSN gap).

When a data PDU is rejected and its DataSN can be ascertained, a target MUST advance ExpDataSN for the current data burst if a recovery R2T is being generated. The target MAY advance its ExpDataSN if it does not attempt to recover the lost data PDU.

6.4. Connection Timeout Management

iSCSI defines two session-global timeout values (in seconds) - Time2Wait and Time2Retain - that are applicable when an iSCSI Full Feature Phase connection is taken out of service either intentionally or by an exception. Time2Wait is the initial "respite time" before attempting an explicit/implicit Logout for the CID in question or task reassignment for the affected tasks (if any). Time2Retain is the maximum time after the initial respite interval that the task and/or connection state(s) is/are guaranteed to be maintained on the

Docker EX1014
Page 76 of 257

target to cater to a possible recovery attempt. Recovery attempts for the connection and/or task(s) SHOULD NOT be made before Time2Wait seconds, but MUST be completed within Time2Retain seconds after that initial Time2Wait waiting period.

6.4.1. Timeouts on Transport Exception Events

A transport connection shutdown or a transport reset without any preceding iSCSI protocol interactions informing the end-points of the fact causes a Full Feature Phase iSCSI connection to be abruptly terminated. The timeout values to be used in this case are the negotiated values of defaultTime2Wait (Section 12.15 DefaultTime2Wait) and DefaultTime2Retain (Section 12.16 DefaultTime2Retain) text keys for the session.

6.4.2. Timeouts on Planned Decommissioning

Any planned decommissioning of a Full Feature Phase iSCSI connection is preceded by either a Logout Response PDU, or an Async Message PDU. The Time2Wait and Time2Retain field values (section 10.15) in a Logout Response PDU, and the Parameter2 and Parameter3 fields of an Async Message (AsyncEvent types "drop the connection" or "drop all the connections"; section 10.9.1) specify the timeout values to be used in each of these cases.

These timeout values are only applicable for the affected connection, and the tasks active on that connection. These timeout values have no bearing on initiator timers (if any) that are already running on connections or tasks associated with that session.

6.5. Implicit Termination of Tasks

A target implicitly terminates the active tasks due to iSCSI protocol dynamics in the following cases:

a) When a connection is implicitly or explicitly logged out with the reason code of "Close the connection" and there are active tasks allegiant to that connection.

b) When a connection fails and the connection state eventually times out (state transition M1 in Section 7.2.2 State Transition Descriptions for Initiators and Targets) and there are active tasks allegiant to that connection.

c) When a successful Logout with the reason code of "remove the connection for recovery" is performed while there are active tasks allegiant to that connection, and those tasks eventually

Docker EX1014
Page 77 of 257

time out after the Time2Wait and Time2Retain periods without allegiance reassignment.

d) When a connection is implicitly or explicitly logged out with the reason code of "Close the session" and there are active tasks in that session.

If the tasks terminated in the above cases a), b, c) and d)are SCSI tasks, they must be internally terminated as if with CHECK CONDITION status. This status is only meaningful for appropriately handling the internal SCSI state and SCSI side effects with respect to ordering because this status is never communicated back as a terminating status to the initiator. However additional actions may have to be taken at SCSI level depending on the SCSI context as defined by the SCSI standards (e.g., queued commands and ACA, in cases a), b), and c), after the tasks are terminated, the target MUST report a Unit Attention condition on the next command processed on any connection for each affected I_T_L nexus with the status of CHECK CONDITION, and the ASC/ASCQ value of 47h/7Fh - "SOME COMMANDS CLEARED BY ISCSI PROTOCOL EVENT" , etc. - see [SAM2] and [SPC3]).

6.6. Format Errors

The following two explicit violations of PDU layout rules are format errors:

a) Illegal contents of any PDU header field except the Opcode (legal values are specified in Section 10 iSCSI PDU Formats).
b) Inconsistent field contents (consistent field contents are specified in Section 10 iSCSI PDU Formats).

Format errors indicate a major implementation flaw in one of the parties.

When a target or an initiator receives an iSCSI PDU with a format error, it MUST immediately terminate all transport connections in the session either with a connection close or with a connection reset and escalate the format error to session recovery (see Section 6.1.4.4 Session Recovery).

6.7. Digest Errors

The discussion of the legal choices in handling digest errors below excludes session recovery as an explicit option, but either party detecting a digest error may choose to escalate the error to session recovery.

Docker EX1014
Page 78 of 257

When a target or an initiator receives any iSCSI PDU, with a header digest error, it MUST either discard the header and all data up to the beginning of a later PDU or close the connection. Because the digest error indicates that the length field of the header may have been corrupted, the location of the beginning of a later PDU needs to be reliably ascertained by other means such as the operation of a sync and steering layer.

When a target receives any iSCSI PDU with a payload digest error, it MUST answer with a Reject PDU with a reason code of Data-Digest-Error and discard the PDU.

- If the discarded PDU is a solicited or unsolicited iSCSI data PDU (for immediate data in a command PDU, non-data PDU rule below applies), the target MUST do one of the following:
  a) Request retransmission with a recovery R2T.
  b) Terminate the task with a response PDU with a CHECK CONDITION Status and an iSCSI Condition of "protocol service CRC error" (Section 10.4.7.2 Sense Data). If the target chooses to implement this option, it MUST wait to receive all the data (signaled by a Data PDU with the final bit set for all outstanding R2Ts) before sending the response PDU. A task management command (such as an abort task) from the initiator during this wait may also conclude the task.
- No further action is necessary for targets if the discarded PDU is a non-data PDU. In case of immediate data being present on a discarded command, the immediate data is implicitly recovered when the task is retried (see section 6.2.1), followed by the entire data transfer for the task.

When an initiator receives any iSCSI PDU with a payload digest error, it MUST discard the PDU.

- If the discarded PDU is an iSCSI data PDU, the initiator MUST do one of the following:

  a) Request the desired data PDU through SNACK. In response to the SNACK, the target MUST either resend the data PDU or reject the SNACK with a Reject PDU with a reason code of "SNACK reject" in which case:
     i) If the status has not already been sent for the command, the target MUST terminate the command with a CHECK CONDITION Status and an iSCSI Condition of "SNACK rejected" (Section 10.4.7.2 Sense Data).
     ii) If the status was already sent, no further action is necessary for the target. The initiator in this case MUST wait for the status to be received and then discard it, so as to internally signal the completion with CHECK CONDITION

Docker EX1014
Page 79 of 257

Status and an iSCSI Condition of "protocol service CRC error" (Section 10.4.7.2 Sense Data).

b) Abort the task and terminate the command with an error.

- If the discarded PDU is a response PDU, the initiator MUST do one of the following:

  a) Request PDU retransmission with a status SNACK.
  b) Logout the connection for recovery and continue the tasks on a different connection instance as described in Section 6.2 Retry and Reassign in Recovery.
  c) Logout to close the connection (abort all the commands associated with the connection).

- No further action is necessary for initiators if the discarded PDU is an unsolicited PDU (e.g., Async, Reject). Task timeouts as in the initiator waiting for a command completion, or process timeouts, as in the target waiting for a Logout, will ensure that the correct operational behavior will result in these cases despite the discarded PDU.

6.8. Sequence Errors

When an initiator receives an iSCSI R2T/data PDU with an out of order R2TSN/DataSN or a SCSI response PDU with an ExpDataSN that implies missing data PDU(s), it means that the initiator must have detected a header or payload digest error on one or more earlier R2T/data PDUs. The initiator MUST address these implied digest errors as described in Section 6.7 Digest Errors. When a target receives a data PDU with an out of order DataSN, it means that the target must have hit a header or payload digest error on at least one of the earlier data PDUs. The target MUST address these implied digest errors as described in Section 6.7 Digest Errors.

When an initiator receives an iSCSI status PDU with an out of order StatSN that implies missing responses, it MUST address the one or more missing status PDUs as described in Section 6.7 Digest Errors. As a side effect of receiving the missing responses, the initiator may discover missing data PDUs. If the initiator wants to recover the missing data for a command, it MUST NOT acknowledge the received responses that start from the StatSN of the relevant command, until it has completed receiving all the data PDUs of the command.

When an initiator receives duplicate R2TSNs (due to proactive retransmission of R2Ts by the target) or duplicate DataSNs (due to proactive SNACKs by the initiator), it MUST discard the duplicates.

6.9. SCSI Timeouts

An iSCSI initiator MAY attempt to plug a command sequence gap on the target end (in the absence of an acknowledgement of the command by way of ExpCmdSN) before the ULP timeout by retrying the unacknowledged command, as described in Section 6.2 Retry and Reassign in Recovery.

On a ULP timeout for a command (that carried a CmdSN of n), if the iSCSI initiator intends to continue the session, it MUST abort the command by either using an appropriate Task Management function request for the specific command, or a "close the connection" Logout. When using an ABORT TASK, if the ExpCmdSN is still less than (n+1), the target may see the abort request while missing the original command itself due to one of the following reasons:

- Original command was dropped due to digest error.
- Connection on which the original command was sent was successfully logged out. Upon logout, the unacknowledged commands issued on the connection being logged out are discarded.

If the abort request is received and the original command is missing, targets MUST consider the original command with that RefCmdSN to be received and issue a Task Management response with the response code: "Function Complete". This response concludes the task on both ends. If the abort request is received and the target can determine (based on the Referenced Task Tag) that the command was received and executed and also that the response was sent prior to the abort, then the target MUST respond with the response code of "Task Does Not Exist".

6.10. Negotiation Failures

Text request and response sequences, when used to set/negotiate operational parameters, constitute the negotiation/parameter setting. A negotiation failure is considered to be one or more of the following:

- None of the choices, or the stated value, is acceptable to one of the sides in the negotiation.
- The text request timed out and possibly terminated.
- The text request was answered with a Reject PDU.

The following two rules should be used to address negotiation failures:

- During Login, any failure in negotiation MUST be considered a login process failure and the Login Phase must be terminated, and with it, the connection. If the target detects the failure, it must terminate the login with the appropriate Login Response code.

- A failure in negotiation, while in the Full Feature Phase, will terminate the entire negotiation sequence that may consist of a series of text requests that use the same Initiator Task Tag. The operational parameters of the session or the connection MUST continue to be the values agreed upon during an earlier successful negotiation (i.e., any partial results of this unsuccessful negotiation MUST NOT take effect and MUST be discarded).

6.11. Protocol Errors

Mapping framed messages over a "stream" connection, such as TCP, makes the proposed mechanisms vulnerable to simple software framing errors. On the other hand, the introduction of framing mechanisms to limit the effects of these errors may be onerous on performance for simple implementations. Command Sequence Numbers and the above mechanisms for connection drop and reestablishment help handle this type of mapping errors.

All violations of iSCSI PDU exchange sequences specified in this document are also protocol errors. This category of errors can only be addressed by fixing the implementations; iSCSI defines Reject and response codes to enable this.

6.12. Connection Failures

iSCSI can keep a session in operation if it is able to keep/establish at least one TCP connection between the initiator and the target in a timely fashion. Targets and/or initiators may recognize a failing connection by either transport level means (TCP), a gap in the command sequence number, a response stream that is not filled for a long time, or by a failing iSCSI NOP (acting as a ping). The latter MAY be used periodically to increase the speed and likelihood of detecting connection failures. Initiators and targets MAY also use the keep-alive option on the TCP connection to enable early link failure detection on otherwise idle links.

Docker EX1014
Page 82 of 257

On connection failure, the initiator and target MUST do one of the following:

- Attempt connection recovery within the session (Section 6.1.4.3 Connection Recovery).

- Logout the connection with the reason code "closes the connection" (Section 10.14.5 Implicit termination of tasks), re-issue missing commands, and implicitly terminate all active commands. This option requires support for the within-connection recovery class (Section 6.1.4.2 Recovery Within-connection).

- Perform session recovery (Section 6.1.4.4 Session Recovery).

Either side may choose to escalate to session recovery (via the initiator dropping all the connections, or via an Async Message that announces the similar intent from a target), and the other side MUST give it precedence. On a connection failure, a target MUST terminate and/or discard all of the active immediate commands regardless of which of the above options is used (i.e., immediate commands are not recoverable across connection failures).

6.13. Session Errors

If all of the connections of a session fail and cannot be reestablished in a short time, or if initiators detect protocol errors repeatedly, an initiator may choose to terminate a session and establish a new session.

In this case, the initiator takes the following actions:

- Resets or closes all the transport connections.
- Terminates all outstanding requests with an appropriate response before initiating a new session. If the same I_T nexus is intended to be reestablished, the initiator MUST employ session reinstatement (see section 5.3.5).

When the session timeout (the connection state timeout for the last failed connection) happens on the target, it takes the following actions:

- Resets or closes the TCP connections (closes the session).
- Terminates all active tasks that were allegiant to the connection(s) that constituted the session.

A target MUST also be prepared to handle a session reinstatement request from the initiator, that may be addressing session errors.

Docker EX1014
Page 83 of 257

7. State Transitions

iSCSI connections and iSCSI sessions go through several well-defined states from the time they are created to the time they are cleared.

The connection state transitions are described in two separate but dependent state diagrams for ease in understanding. The first diagram, "standard connection state diagram", describes the connection state transitions when the iSCSI connection is not waiting for, or undergoing, a cleanup by way of an explicit or implicit Logout. The second diagram, "connection cleanup state diagram", describes the connection state transitions while performing the iSCSI connection cleanup.

The "session state diagram" describes the state transitions an iSCSI session would go through during its lifetime, and it depends on the states of possibly multiple iSCSI connections that participate in the session.

States and state transitions are described in the text, tables and diagrams. The diagrams are used for illustration. The text and the tables are the governing specification.

7.1. Standard Connection State Diagrams

7.1.1. State Descriptions for Initiators and Targets

State descriptions for the standard connection state diagram are as follows:

- S1: FREE
  - initiator: State on instantiation, or after successful connection closure.
  - target: State on instantiation, or after successful connection closure.
- S2: XPT_WAIT
  - initiator: Waiting for a response to its transport connection establishment request.
  - target: Illegal
- S3: XPT_UP
  - initiator: Illegal
  - target: Waiting for the Login process to commence.
- S4: IN_LOGIN
  - initiator: Waiting for the Login process to conclude, possibly involving several PDU exchanges.
  - target: Waiting for the Login process to conclude, possibly involving several PDU exchanges.

Docker EX1014
Page 84 of 257

-S5: LOGGED_IN
-initiator: In Full Feature Phase, waiting for all internal, iSCSI, and transport events.
-target: In Full Feature Phase, waiting for all internal, iSCSI, and transport events.
-S6: IN_LOGOUT
-initiator: Waiting for a Logout response.
-target: Waiting for an internal event signaling completion of logout processing.
-S7: LOGOUT_REQUESTED
-initiator: Waiting for an internal event signaling readiness to proceed with Logout.
-target: Waiting for the Logout process to start after having requested a Logout via an Async Message.
-S8: CLEANUP_WAIT
-initiator: Waiting for the context and/or resources to initiate the cleanup processing for this CSM.
-target: Waiting for the cleanup process to start for this CSM.

7.1.2. State Transition Descriptions for Initiators and Targets

-T1:
-initiator: Transport connect request was made (e.g., TCP SYN sent).
-target: Illegal
-T2:
-initiator: Transport connection request timed out, a transport reset was received, or an internal event of receiving a Logout response (success) on another connection for a "close the session" Logout request was received.
-target:Illegal
-T3:
-initiator: Illegal
-target: Received a valid transport connection request that establishes the transport connection.
-T4:
-initiator: Transport connection established, thus prompting the initiator to start the iSCSI Login.
-target: Initial iSCSI Login Request was received.
-T5:
-initiator: The final iSCSI Login Response with a Status-Class of zero was received.
-target: The final iSCSI Login Request to conclude the Login Phase was received, thus prompting the target to send the final iSCSI Login Response with a Status-Class of zero.

-T6:
-initiator: Illegal
-target: Timed out waiting for an iSCSI Login, transport disconnect indication was received, transport reset was received, or an internal event indicating a transport timeout was received. In all these cases, the connection is to be closed.

-T7:
-initiator - one of the following events caused the transition:
- The final iSCSI Login Response was received with a non-zero Status-Class.
- Login timed out.
- A transport disconnect indication was received.
- A transport reset was received.
- An internal event was received indicating a transport timeout.
- An internal event of receiving a Logout response (success) on another connection for a "close the session" Logout request was received.

In all these cases, the transport connection is closed.

-target - one of the following events caused the transition:
- The final iSCSI Login Request to conclude the Login Phase was received, prompting the target to send the final iSCSI Login Response with a non-zero Status-Class.
- Login timed out.
- Transport disconnect indication was received.
- Transport reset was received.
- An internal event indicating a transport timeout was received.
- On another connection a "close the session" Logout request was received.

In all these cases, the connection is to be closed.

-T8:
-initiator: An internal event of receiving a Logout response (success) on another connection for a "close the session" Logout request was received, thus closing this connection requiring no further cleanup.
-target: An internal event of sending a Logout response (success) on another connection for a "close the session" Logout request was received, or an internal event of a successful connection/session reinstatement is received, thus prompting the target to close this connection cleanly.

```
-T9, T10:
     -initiator: An internal event that indicates the readiness to
         start the Logout process was received, thus prompting an
         iSCSI Logout to be sent by the initiator.
     -target: An iSCSI Logout request was received.
-T11, T12:
     -initiator: Async PDU with AsyncEvent "Request Logout" was
         received.
     -target: An internal event that requires the decommissioning of
         the connection is received, thus causing an Async PDU with
         an AsyncEvent "Request Logout" to be sent.
-T13:
     -initiator: An iSCSI Logout response (success) was received, or
         an internal event of receiving a Logout response (success)
         on another connection for a "close the session" Logout
         request was received.
     -target: An internal event was received that indicates
         successful processing of the Logout, which prompts an iSCSI
         Logout response (success) to be sent; an internal event of
         sending a Logout response (success) on another connection
         for a "close the session" Logout request was received; or an
         internal event of a successful connection/session
         reinstatement is received.  In all these cases, the
         transport connection is closed.

-T14:
     -initiator: Async PDU with AsyncEvent "Request Logout" was
         received again.
     -target: Illegal
-T15, T16:
     -initiator: One or more of the following events caused this
         transition:
         -Internal event that indicates a transport connection
            timeout was received thus prompting transport RESET or
            transport connection closure.
         -A transport RESET.
         -A transport disconnect indication.
         -Async PDU with AsyncEvent "Drop connection" (for this CID).
         -Async PDU with AsyncEvent "Drop all connections".
     -target: One or more of the following events caused this
         transition:
         -Internal event that indicates a transport connection
            timeout was received, thus prompting transport RESET or
            transport connection closure.
         -An internal event of a failed connection/session
            reinstatement is received.
         -A transport RESET.
         -A transport disconnect indication.
```

```
            -Internal emergency cleanup event was received which prompts
               an Async PDU with AsyncEvent "Drop connection" (for this
               CID), or event "Drop all connections".
   -T17:
        -initiator: One or more of the following events caused this
            transition:
            -Logout response, (failure i.e., a non-zero status) was
               received, or Logout timed out.
            -Any of the events specified for T15 and T16.
        -target:  One or more of the following events caused this
            transition:
            -Internal event that indicates a failure of the Logout
               processing was received, which prompts a Logout response
               (failure, i.e., a non-zero status) to be sent.
            -Any of the events specified for T15 and T16.
   -T18:
        -initiator: An internal event of receiving a Logout response
            (success) on another connection for a "close the session"
            Logout request was received.
        -target: An internal event of sending a Logout response
            (success) on another connection for a "close the session"
            Logout request was received, or an internal event of a
            successful connection/session reinstatement is received.  In
            both these cases, the connection is closed.
```

The CLEANUP_WAIT state (S8) implies that there are possible iSCSI tasks that have not reached conclusion and are still considered busy.

7.1.3. Standard Connection State Diagram for an Initiator

Symbolic names for States:

```
S1: FREE
S2: XPT_WAIT
S4: IN_LOGIN
S5: LOGGED_IN
S6: IN_LOGOUT
S7: LOGOUT_REQUESTED
S8: CLEANUP_WAIT
```

States S5, S6, and S7 constitute the Full Feature Phase operation of the connection.

The state diagram is as follows:

```
                    -------<-------------+
      +--------->/ S1    \<----+       |
   T13|       +->\       /<-+   \      |
      |      /    ---+---    \   \     |
      |     /        |     T2 \   |    |
      |  T8 |        |T1       |  |    |
      |     |        |        /   |T7  |
      |     |        |       /    |    |
      |     |        |      /     |    |
      |     |        V     /     /     |
      |     |     ------- /     /      |
      |     |    / S2    \     /       |
      |     |    \       /    /        |
      |     |     ---+---    /         |
      |     |        |T4    /           |
      |     |        V     /            | T18
      |     |     ------- /             |
      |     |    / S4    \              |
      |     |    \       /              |
      |     |     ---+---               |         T15
      |     |        |T5      +--------+---------+
      |     |        |       /T16+-----+------+  |
      |     |        |      /   -+-----+--+   |  |
      |     |        |     /   /  S7   \  |T12|  |
      |     |        |    / +->\       /<-+   V  V
      |     |        |   / /    -+-----       -------
      |     |        |  / /T11   |T10        /  S8   \
      |     |        V / /       V  +----+   \       /
      |     |      ---+-+-      ----+--  |    -------
      |     |     / S5    \T9  / S6    \<+    ^
      |     +-----\       /--->\       / T14  |
      |            -------      --+----+------+T17
      +----------------------------+
```

The following state transition table represents the above diagram. Each row represents the starting state for a given transition, which after taking a transition marked in a table cell would end in the state represented by the column of the cell. For example, from state S1, the connection takes the T1 transition to arrive at state S2. The fields marked "-" correspond to undefined transitions.

	S1	S2	S4	S5	S6	S7	S8
S1	-	T1	-	-	-	-	-
S2	T2	-	T4	-	-	-	-
S4	T7	-	-	T5	-	-	-
S5	T8	-	-	-	T9	T11	T15
S6	T13	-	-	-	T14	-	T17
S7	T18	-	-	-	T10	T12	T16
S8	-	-	-	-	-	-	-

7.1.4. Standard Connection State Diagram for a Target

Symbolic names for States:

S1: FREE
S3: XPT_UP
S4: IN_LOGIN
S5: LOGGED_IN
S6: IN_LOGOUT
S7: LOGOUT_REQUESTED
S8: CLEANUP_WAIT

States S5, S6, and S7 constitute the Full Feature Phase operation of the connection.

Docker EX1014
Page 90 of 257

The state diagram is as follows:

```
                      -------<-------------+
           +--------->/ S1    \<----+       |
        T13|       +->\       /<-+   \      |
           |      /    ---+---    \   \     |
           |     /        |     T6 \   |    |
           |  T8 |        |T3       |  |    |
           |     |        |        /   |T7  |
           |     |        |       /    |    |
           |     |        |      /     |    |
           |     |        V     /     /     |
           |     |     ------- /     /      |
           |     |    / S3    \     /       |
           |     |    \       /    /        | T18
           |     |     ---+---    /         |
           |     |        |T4    /          |
           |     |        V     /           |
           |     |     ------- /            |
           |     |    / S4    \             |
           |     |    \       /             |
           |     |     ---+---         T15  |
           |     |        |T5      +--------+---------+
           |     |        |       /T16+-----+------+  |
           |     |        |      /  -+-----+---+   |  |
           |     |        |     /   /  S7   \  |T12|  |
           |     |        |    / +->\       /<-+   V  V
           |     |        |   / /    -+-----       -------
           |     |        |  / /T11   |T10        /  S8   \
           |     |        V / /       V           \       /
           |     |      ---+-+-      -------       -------
           |     |     / S5    \T9  / S6    \        ^
           |     +-----\       /--->\       /        |
           |            -------      --+----+--------+T17
           +---------------------------+
```

The following state transition table represents the above diagram, and follows the conventions described for the initiator diagram.

	S1	S3	S4	S5	S6	S7	S8
S1	-	T3	-	-	-	-	-
S3	T6	-	T4	-	-	-	-
S4	T7	-	-	T5	-	-	-
S5	T8	-	-	-	T9	T11	T15
S6	T13	-	-	-	-	-	T17
S7	T18	-	-	-	T10	T12	T16
S8	-	-	-	-	-	-	-

7.2. Connection Cleanup State Diagram for Initiators and Targets

Symbolic names for states:

R1: CLEANUP_WAIT (same as S8)
R2: IN_CLEANUP
R3: FREE (same as S1)

Whenever a connection state machine (e.g., CSM-C) enters the CLEANUP_WAIT state (S8), it must go through the state transitions described in the connection cleanup state diagram either a) using a separate full-feature phase connection (let's call it CSM-E) in the LOGGED_IN state in the same session, or b) using a new transport connection (let's call it CSM-I) in the FREE state that is to be added to the same session. In the CSM-E case, an explicit logout for the CID that corresponds to CSM-C (either as a connection or session logout) needs to be performed to complete the cleanup. In the CSM-I case, an implicit logout for the CID that corresponds to CSM-C needs to be performed by way of connection reinstatement (section 5.3.4) for that CID. In either case, the protocol exchanges on CSM-E or CSM-I determine the state transitions for CSM-C. Therefore, this cleanup state diagram is only applicable to the instance of the connection in cleanup (i.e., CSM-C). In the case of an implicit logout for example, CSM-C reaches FREE (R3) at the time CSM-I reaches LOGGED_IN. In the case of an explicit logout, CSM-C reaches FREE (R3) when CSM-E receives a successful logout response while continuing to be in the LOGGED_IN state.

An initiator must initiate an explicit or implicit connection logout for a connection in the CLEANUP_WAIT state, if the initiator intends to continue using the associated iSCSI session.

The following state diagram applies to both initiators and targets.

```
                        -------
                       / R1    \
                  +--\        /<-+
                 /    ---+---
                /         |        \ M3
            M1 |         |M2       |
               |         |        /
               |         |       /
               |         |      /
               |         V     /
               |      ------- /
               |     / R2    \
               |     \       /
               |      -------
               |         |
               |         |M4
               |         |
               |         |
               |         |
               |         V
               |       -------
               |      / R3    \
               +---->\        /
                       -------
```

The following state transition table represents the above diagram, and follows the same conventions as in earlier sections.

	R1	R2	R3
R1	-	M2	M1
R2	M3	-	M4
R3	-	-	-

7.2.1. State Descriptions for Initiators and Targets

```
-R1: CLEANUP_WAIT (Same as S8)
     -initiator: Waiting for the internal event to initiate the
         cleanup processing for CSM-C.
     -target: Waiting for the cleanup process to start for CSM-C.
-R2: IN_CLEANUP
     -initiator: Waiting for the connection cleanup process to
         conclude for CSM-C.
     -target: Waiting for the connection cleanup process to conclude
         for CSM-C.
-R3: FREE (Same as S1)
     -initiator: End state for CSM-C.
     -target: End state for CSM-C.
```

7.2.2. State Transition Descriptions for Initiators and Targets

```
-M1: One or more of the following events was received:
     -initiator:
         -An internal event that indicates connection state timeout.
         -An internal event of receiving a successful Logout response
            on a different connection for a "close the session"
            Logout.
     -target:
         -An internal event that indicates connection state timeout.
         -An internal event of sending a Logout response (success) on
            a different connection for a "close the session" Logout
            request.

-M2: An implicit/explicit logout process was initiated by the
     initiator.
     -In CSM-I usage:
         -initiator: An internal event requesting the connection (or
            session) reinstatement was received, thus prompting a
            connection (or session) reinstatement Login to be sent
            transitioning CSM-I to state IN_LOGIN.
         -target: A connection/session reinstatement Login was
            received while in state XPT_UP.
     -In CSM-E usage:
         -initiator: An internal event that indicates that an
            explicit logout was sent for this CID in state LOGGED_IN.
         -target: An explicit logout was received for this CID in
            state LOGGED_IN.
```

-M3: Logout failure detected
- -In CSM-I usage:
  - -initiator: CSM-I failed to reach LOGGED_IN and arrived into FREE instead.
  - -target: CSM-I failed to reach LOGGED_IN and arrived into FREE instead.
- -In CSM-E usage:
  - -initiator: CSM-E either moved out of LOGGED_IN, or Logout timed out and/or aborted, or Logout response (failure) was received.
  - -target: CSM-E either moved out of LOGGED_IN, Logout timed out and/or aborted, or an internal event that indicates a failed Logout processing was received. A Logout response (failure) was sent in the last case.

-M4: Successful implicit/explicit logout was performed.

- In CSM-I usage:
  - -initiator: CSM-I reached state LOGGED_IN, or an internal event of receiving a Logout response (success) on another connection for a "close the session" Logout request was received.
  - -target: CSM-I reached state LOGGED_IN, or an internal event of sending a Logout response (success) on a different connection for a "close the session" Logout request was received.
- In CSM-E usage:
  - -initiator: CSM-E stayed in LOGGED_IN and received a Logout response (success), or an internal event of receiving a Logout response (success) on another connection for a "close the session" Logout request was received.
  - -target: CSM-E stayed in LOGGED_IN and an internal event indicating a successful Logout processing was received, or an internal event of sending a Logout response (success) on a different connection for a "close the session" Logout request was received.

7.3. Session State Diagrams

7.3.1. Session State Diagram for an Initiator

Symbolic Names for States:

Q1: FREE
Q3: LOGGED_IN
Q4: FAILED

State Q3 represents the Full Feature Phase operation of the session.

The state diagram is as follows:

```
                         -------
                        / Q1    \
                 +------>\       /<-+
                /         ---+---   |
               /             |      |N3
           N6 |              |N1    |
              |              |      |
              |    N4        |      |
              |  +--------+  |     /
              |  |        |  |    /
              |  |        |  |   /
              |  |        V  V  /
            -+--+--      -----+-
           / Q4    \ N5 / Q3    \
           \       /<---\       /
            -------      -------
```

The state transition table is as follows:

```
     +----+----+----+
     |Q1  |Q3  |Q4  |
-----+----+----+----+
 Q1  | -  |N1  | -  |
-----+----+----+----+
 Q3  |N3  | -  |N5  |
-----+----+----+----+
 Q4  |N6  |N4  | -  |
-----+----+----+----+
```

7.3.2. Session State Diagram for a Target

Symbolic Names for States:

```
  Q1: FREE
  Q2: ACTIVE
  Q3: LOGGED_IN
  Q4: FAILED
  Q5: IN_CONTINUE
```

State Q3 represents the Full Feature Phase operation of the session.

The state diagram is as follows:

```
                                  -------
             +------------------>/ Q1    \
            /    +--------------->\       /<-+
            |    |                 ---+---   |
            |    |                 ^  |      |N3
         N6 |    |N11           N9|  V N1   |
            |    |                 +------   |
            |    |                / Q2    \  |
            |    |                \       /  |
            |  --+----             +--+---   |
            | / Q5    \               |      |
            | \       / N10           |      |
            |  +-+---+------------+  |N2   /
            |  ^ |                 |  |    /
            |N7| |N8               |  |   /
            |  | |                 |  V  /
           -+--+-V                 V----+-
          / Q4    \ N5            / Q3    \
          \       /<-------------\       /
           -------                 -------
```

The state transition table is as follows:

```
     +----+----+----+----+----+
     |Q1  |Q2  |Q3  |Q4  |Q5  |
-----+----+----+----+----+----+
 Q1  | -  |N1  | -  | -  | -  |
-----+----+----+----+----+----+
 Q2  |N9  | -  |N2  | -  | -  |
-----+----+----+----+----+----+
 Q3  |N3  | -  | -  |N5  | -  |
-----+----+----+----+----+----+
 Q4  |N6  | -  | -  | -  |N7  |
-----+----+----+----+----+----+
 Q5  |N11 | -  |N10 |N8  | -  |
-----+----+----+----+----+----+
```

7.3.3. State Descriptions for Initiators and Targets

-Q1: FREE
    -initiator: State on instantiation or after cleanup.
    -target: State on instantiation or after cleanup.

Docker EX1014
Page 97 of 257

-Q2: ACTIVE
-initiator: Illegal.
-target: The first iSCSI connection in the session transitioned to IN_LOGIN, waiting for it to complete the login process.

-Q3: LOGGED_IN
-initiator: Waiting for all session events.
-target: Waiting for all session events.

-Q4: FAILED
-initiator: Waiting for session recovery or session continuation.
-target: Waiting for session recovery or session continuation.

-Q5: IN_CONTINUE
-initiator: Illegal.
-target: Waiting for session continuation attempt to reach a conclusion.

7.3.4. State Transition Descriptions for Initiators and Targets

-N1:
-initiator: At least one transport connection reached the LOGGED_IN state.
-target: The first iSCSI connection in the session had reached the IN_LOGIN state.

-N2:
-initiator: Illegal.
-target: At least one iSCSI connection reached the LOGGED_IN state.

-N3:
-initiator: Graceful closing of the session via session closure (Section 5.3.6 Session Continuation and Failure).
-target: Graceful closing of the session via session closure (Section 5.3.6 Session Continuation and Failure) or a successful session reinstatement cleanly closed the session.

-N4:
-initiator: A session continuation attempt succeeded.
-target: Illegal.

-N5:
-initiator: Session failure (Section 5.3.6 Session Continuation and Failure) occurred.
-target: Session failure (Section 5.3.6 Session Continuation and Failure) occurred.

-N6:
-initiator: Session state timeout occurred, or a session reinstatement cleared this session instance. This results in the freeing of all associated resources and the session state is discarded.
-target: Session state timeout occurred, or a session reinstatement cleared this session instance. This results in the freeing of all associated resources and the session state is discarded.

-N7:
-initiator: Illegal.
-target: A session continuation attempt is initiated.

-N8:
-initiator: Illegal.
-target: The last session continuation attempt failed.

-N9:
-initiator: Illegal.
-target: Login attempt on the leading connection failed.

-N10:
-initiator: Illegal.
-target: A session continuation attempt succeeded.

-N11:
-initiator: Illegal.
-target: A successful session reinstatement cleanly closed the session.

8. Security Considerations

Historically, native storage systems have not had to consider security because their environments offered minimal security risks. That is, these environments consisted of storage devices either directly attached to hosts or connected via a Storage Area Network (SAN) distinctly separate from the communications network. The use of storage protocols, such as SCSI, over IP-networks requires that security concerns be addressed. iSCSI implementations MUST provide means of protection against active attacks (e.g., pretending to be another identity, message insertion, deletion, modification, and replaying) and passive attacks (e.g., eavesdropping, gaining advantage by analyzing the data sent over the line).

Docker EX1014
Page 99 of 257

Although technically possible, iSCSI SHOULD NOT be configured without security. iSCSI configured without security should be confined, in extreme cases, to closed environments without any security risk. [RFC3723] specifies the mechanisms that must be used in order to mitigate risks fully described in that document.

The following section describes the security mechanisms provided by an iSCSI implementation.

8.1. iSCSI Security Mechanisms

The entities involved in iSCSI security are the initiator, target, and the IP communication end points. iSCSI scenarios in which multiple initiators or targets share a single communication end point are expected. To accommodate such scenarios, iSCSI uses two separate security mechanisms: In-band authentication between the initiator and the target at the iSCSI connection level (carried out by exchange of iSCSI Login PDUs), and packet protection (integrity, authentication, and confidentiality) by IPsec at the IP level. The two security mechanisms complement each other. The in-band authentication provides end-to-end trust (at login time) between the iSCSI initiator and the target while IPsec provides a secure channel between the IP communication end points.

Further details on typical iSCSI scenarios and the relation between the initiators, targets, and the communication end points can be found in [RFC3723].

8.2. In-band Initiator-Target Authentication

During login, the target MAY authenticate the initiator and the initiator MAY authenticate the target. The authentication is performed on every new iSCSI connection by an exchange of iSCSI Login PDUs using a negotiated authentication method.

The authentication method cannot assume an underlying IPsec protection, because IPsec is optional to use. An attacker should gain as little advantage as possible by inspecting the authentication phase PDUs. Therefore, a method using clear text (or equivalent) passwords is not acceptable; on the other hand, identity protection is not strictly required.

The authentication mechanism protects against an unauthorized login to storage resources by using a false identity (spoofing). Once the authentication phase is completed, if the underlying IPsec is not used, all PDUs are sent and received in clear. The authentication

Docker EX1014
Page 100 of 257

mechanism alone (without underlying IPsec) should only be used when there is no risk of eavesdropping, message insertion, deletion, modification, and replaying.

Section 11 iSCSI Security Text Keys and Authentication Methods defines several authentication methods and the exact steps that must be followed in each of them, including the iSCSI-text-keys and their allowed values in each step. Whenever an iSCSI initiator gets a response whose keys, or their values, are not according to the step definition, it MUST abort the connection. Whenever an iSCSI target gets a response whose keys, or their values, are not according to the step definition, it MUST answer with a Login reject with the "Initiator Error" or "Missing Parameter" status. These statuses are not intended for cryptographically incorrect values such as the CHAP response, for which "Authentication Failure" status MUST be specified. The importance of this rule can be illustrated in CHAP with target authentication (see Section 11.1.4 Challenge Handshake Authentication Protocol (CHAP)) where the initiator would have been able to conduct a reflection attack by omitting his response key (CHAP_R) using the same CHAP challenge as the target and reflecting the target’s response back to the target. In CHAP, this is prevented because the target must answer the missing CHAP_R key with a Login reject with the "Missing Parameter" status.

For some of the authentication methods, a key specifies the identity of the iSCSI initiator or target for authentication purposes. The value associated with that key MAY be different from the iSCSI name and SHOULD be configurable. (CHAP_N, see Section 11.1.4 Challenge Handshake Authentication Protocol (CHAP) and SRP_U, see Section 11.1.3 Secure Remote Password (SRP)).

8.2.1. CHAP Considerations

Compliant iSCSI initiators and targets MUST implement the CHAP authentication method [RFC1994] (according to Section 11.1.4 Challenge Handshake Authentication Protocol (CHAP) including the target authentication option).

When CHAP is performed over a non-encrypted channel, it is vulnerable to an off-line dictionary attack. Implementations MUST support use of up to 128 bit random CHAP secrets, including the means to generate such secrets and to accept them from an external generation source. Implementations MUST NOT provide secret generation (or expansion) means other than random generation.

An administrative entity of an environment in which CHAP is used with a secret that has less than 96 random bits MUST enforce IPsec encryption (according to the implementation requirements in Section

Docker EX1014
Page 101 of 257

8.3.2 Confidentiality) to protect the connection. Moreover, in this case IKE authentication with group pre-shared cryptographic keys SHOULD NOT be used unless it is not essential to protect group members against off-line dictionary attacks by other members.

CHAP secrets MUST be an integral number of bytes (octets). A compliant implementation SHOULD NOT continue with the login step in which it should send a CHAP response (CHAP_R, Section 11.1.4 Challenge Handshake Authentication Protocol (CHAP)) unless it can verify that the CHAP secret is at least 96 bits, or that IPsec encryption is being used to protect the connection.

Any CHAP secret used for initiator authentication MUST NOT be configured for authentication of any target, and any CHAP secret used for target authentication MUST NOT be configured for authentication of any initiator. If the CHAP response received by one end of an iSCSI connection is the same as the CHAP response that the receiving endpoint would have generated for the same CHAP challenge, the response MUST be treated as an authentication failure and cause the connection to close (this ensures that the same CHAP secret is not used for authentication in both directions). Also, if an iSCSI implementation can function as both initiator and target, different CHAP secrets and identities MUST be configured for these two roles. The following is an example of the attacks prevented by the above requirements:

Rogue wants to impersonate Storage to Alice, and knows that a single secret is used for both directions of Storage-Alice authentication.

Rogue convinces Alice to open two connections to Rogue, and Rogue identifies itself as Storage on both connections.

Rogue issues a CHAP challenge on connection 1, waits for Alice to respond, and then reflects Alice’s challenge as the initial challenge to Alice on connection 2.

If Alice doesn’t check for the reflection across connections, Alice’s response on connection 2 enables Rogue to impersonate Storage on connection 1, even though Rogue does not know the Alice-Storage CHAP secret.

Originators MUST NOT reuse the CHAP challenge sent by the Responder for the other direction of a bidirectional authentication. Responders MUST check for this condition and close the iSCSI TCP connection if it occurs.

Docker EX1014
Page 102 of 257

The same CHAP secret SHOULD NOT be configured for authentication of multiple initiators or multiple targets, as this enables any of them to impersonate any other one of them, and compromising one of them enables the attacker to impersonate any of them. It is recommended that iSCSI implementations check for use of identical CHAP secrets by different peers when this check is feasible, and take appropriate measures to warn users and/or administrators when this is detected.

When an iSCSI initiator or target authenticates itself to counterparts in multiple administrative domains, it SHOULD use a different CHAP secret for each administrative domain to avoid propagating security compromises across domains.

Within a single administrative domain:
- A single CHAP secret MAY be used for authentication of an initiator to multiple targets.
- A single CHAP secret MAY be used for an authentication of a target to multiple initiators when the initiators use an external server (e.g., RADIUS) to verify the target's CHAP responses and do not know the target's CHAP secret.

If an external response verification server (e.g., RADIUS) is not used, employing a single CHAP secret for authentication of a target to multiple initiators requires that all such initiators know that target secret. Any of these initiators can impersonate the target to any other such initiator, and compromise of such an initiator enables an attacker to impersonate the target to all such initiators. Targets SHOULD use separate CHAP secrets for authentication to each initiator when such risks are of concern; in this situation it may be useful to configure a separate logical iSCSI target with its own iSCSI Node Name for each initiator or group of initiators among which such separation is desired.

8.2.2. SRP Considerations

The strength of the SRP authentication method (specified in [RFC2945]) is dependent on the characteristics of the group being used (i.e., the prime modulus N and generator g). As described in [RFC2945], N is required to be a Sophie-German prime (of the form N = 2q + 1, where q is also prime) and the generator g is a primitive root of GF(n). In iSCSI authentication, the prime modulus N MUST be at least 768 bits.

The list of allowed SRP groups is provided in [RFC3723].

Docker EX1014
Page 103 of 257

8.3. IPsec

iSCSI uses the IPsec mechanism for packet protection (cryptographic integrity, authentication, and confidentiality) at the IP level between the iSCSI communicating end points. The following sections describe the IPsec protocols that must be implemented for data integrity and authentication, confidentiality, and cryptographic key management.

An iSCSI initiator or target may provide the required IPsec support fully integrated or in conjunction with an IPsec front-end device. In the latter case, the compliance requirements with regard to IPsec support apply to the "combined device". Only the "combined device" is to be considered an iSCSI device.

Detailed considerations and recommendations for using IPsec for iSCSI are provided in [RFC3723].

8.3.1. Data Integrity and Authentication

Data authentication and integrity is provided by a cryptographic keyed Message Authentication Code in every sent packet. This code protects against message insertion, deletion, and modification. Protection against message replay is realized by using a sequence counter.

An iSCSI compliant initiator or target MUST provide data integrity and authentication by implementing IPsec [RFC2401] with ESP [RFC2406] in tunnel mode and MAY provide data integrity and authentication by implementing IPsec with ESP in transport mode. The IPsec implementation MUST fulfill the following iSCSI specific requirements:

- HMAC-SHA1 MUST be implemented [RFC2404].
- AES CBC MAC with XCBC extensions SHOULD be implemented [RFC3566].

The ESP anti-replay service MUST also be implemented.

At the high speeds iSCSI is expected to operate, a single IPsec SA could rapidly cycle through the 32-bit IPsec sequence number space. In view of this, it may be desirable in the future for an iSCSI implementation that operates at speeds of 1 Gbps or greater to implement the IPsec sequence number extension [SEQ-EXT].

Docker EX1014
Page 104 of 257

8.3.2. Confidentiality

Confidentiality is provided by encrypting the data in every packet. When confidentiality is used it MUST be accompanied by data integrity and authentication to provide comprehensive protection against eavesdropping, message insertion, deletion, modification, and replaying.

An iSCSI compliant initiator or target MUST provide confidentiality by implementing IPsec [RFC2401] with ESP [RFC2406] in tunnel mode and MAY provide confidentiality by implementing IPsec with ESP in transport mode, with the following iSCSI specific requirements:

- 3DES in CBC mode MUST be implemented [RFC2451].
- AES in Counter mode SHOULD be implemented [RFC3686].

DES in CBC mode SHOULD NOT be used due to its inherent weakness. The NULL encryption algorithm MUST also be implemented.

8.3.3. Policy, Security Associations, and Cryptographic Key Management

A compliant iSCSI implementation MUST meet the cryptographic key management requirements of the IPsec protocol suite. Authentication, security association negotiation, and cryptographic key management MUST be provided by implementing IKE [RFC2409] using the IPsec DOI [RFC2407] with the following iSCSI specific requirements:

- Peer authentication using a pre-shared cryptographic key MUST be supported. Certificate-based peer authentication using digital signatures MAY be supported. Peer authentication using the public key encryption methods outlined in IKE sections 5.2 and 5.3[7] SHOULD NOT be used.

- When digital signatures are used to achieve authentication, an IKE negotiator SHOULD use IKE Certificate Request Payload(s) to specify the certificate authority. IKE negotiators SHOULD check the pertinent Certificate Revocation List (CRL) before accepting a PKI certificate for use in IKE authentication procedures.

- Conformant iSCSI implementations MUST support IKE Main Mode and SHOULD support Aggressive Mode. IKE main mode with pre-shared key authentication method SHOULD NOT be used when either the initiator or the target uses dynamically assigned IP addresses. While in many cases pre-shared keys offer good security, situations in which dynamically assigned addresses are used force the use of a group pre-shared key, which creates vulnerability to a man-in-the-middle attack.

Docker EX1014
Page 105 of 257

- In the IKE Phase 2 Quick Mode, exchanges for creating the Phase 2 SA, the Identity Payload, fields MUST be present. ID_IPV4_ADDR, ID_IPV6_ADDR (if the protocol stack supports IPv6) and ID_FQDN Identity payloads MUST be supported; ID_USER_FQDN SHOULD be supported. The IP Subnet, IP Address Range, ID_DER_ASN1_DN, and ID_DER_ASN1_GN formats SHOULD NOT be used. The ID_KEY_ID Identity Payload MUST NOT be used.

Manual cryptographic keying MUST NOT be used because it does not provide the necessary re-keying support.

When IPsec is used, the receipt of an IKE Phase 2 delete message SHOULD NOT be interpreted as a reason for tearing down the iSCSI TCP connection. If additional traffic is sent on it, a new IKE Phase 2 SA will be created to protect it.

The method used by the initiator to determine whether the target should be connected using IPsec is regarded as an issue of IPsec policy administration, and thus not defined in the iSCSI standard.

If an iSCSI target is discovered via a SendTargets request in a discovery session not using IPsec, the initiator should assume that it does not need IPsec to establish a session to that target. If an iSCSI target is discovered using a discovery session that does use IPsec, the initiator SHOULD use IPsec when establishing a session to that target.

9. Notes to Implementers

This section notes some of the performance and reliability considerations of the iSCSI protocol. This protocol was designed to allow efficient silicon and software implementations. The iSCSI task tag mechanism was designed to enable Direct Data Placement (DDP - a DMA form) at the iSCSI level or lower.

The guiding assumption made throughout the design of this protocol is that targets are resource constrained relative to initiators.

Implementers are also advised to consider the implementation consequences of the iSCSI to SCSI mapping model as outlined in Section 3.4.3 Consequences of the Model.

9.1. Multiple Network Adapters

The iSCSI protocol allows multiple connections, not all of which need to go over the same network adapter. If multiple network connections are to be utilized with hardware support, the iSCSI protocol

Docker EX1014
Page 106 of 257

command-data-status allegiance to one TCP connection ensures that there is no need to replicate information across network adapters or otherwise require them to cooperate.

However, some task management commands may require some loose form of cooperation or replication at least on the target.

9.1.1. Conservative Reuse of ISIDs

Historically, the SCSI model (and implementations and applications based on that model) has assumed that SCSI ports are static, physical entities. Recent extensions to the SCSI model have taken advantage of persistent worldwide unique names for these ports. In iSCSI however, the SCSI initiator ports are the endpoints of dynamically created sessions, so the presumptions of "static and physical" do not apply. In any case, the model clauses (particularly, Section 3.4.2 SCSI Architecture Model) provide for persistent, reusable names for the iSCSI-type SCSI initiator ports even though there does not need to be any physical entity bound to these names.

To both minimize the disruption of legacy applications and to better facilitate the SCSI features that rely on persistent names for SCSI ports, iSCSI implementations SHOULD attempt to provide a stable presentation of SCSI Initiator Ports (both to the upper OS-layers and to the targets to which they connect). This can be achieved in an initiator implementation by conservatively reusing ISIDs. In other words, the same ISID should be used in the Login process to multiple target portal groups (of the same iSCSI Target or different iSCSI Targets). The ISID RULE (Section 3.4.3 Consequences of the Model) only prohibits reuse to the same target portal group. It does not "preclude" reuse to other target portal groups. The principle of conservative reuse "encourages" reuse to other target portal groups. When a SCSI target device sees the same (InitiatorName, ISID) pair in different sessions to different target portal groups, it can identify the underlying SCSI Initiator Port on each session as the same SCSI port. In effect, it can recognize multiple paths from the same source.

9.1.2. iSCSI Name, ISID, and TPGT Use

The designers of the iSCSI protocol envisioned there being one iSCSI Initiator Node Name per operating system image on a machine. This enables SAN resource configuration and authentication schemes based on a system’s identity. It supports the notion that it should be possible to assign access to storage resources based on "initiator device" identity.

Docker EX1014
Page 107 of 257

When there are multiple hardware or software components coordinated as a single iSCSI Node, there must be some (logical) entity that represents the iSCSI Node that makes the iSCSI Node Name available to all components involved in session creation and login. Similarly, this entity that represents the iSCSI Node must be able to coordinate session identifier resources (ISID for initiators) to enforce both the ISID and TSIH RULES (see Section 3.4.3 Consequences of the Model).

For targets, because of the closed environment, implementation of this entity should be straightforward. However, vendors of iSCSI hardware (e.g., NICs or HBAs) intended for targets, SHOULD provide mechanisms for configuration of the iSCSI Node Name across the portal groups instantiated by multiple instances of these components within a target.

However, complex targets making use of multiple Target Portal Group Tags may reconfigure them to achieve various quality goals. The initiators have two mechanisms at their disposal to discover and/or check reconfiguring targets - the discovery session type and a key returned by the target during login to confirm the TPGT. An initiator should attempt to "rediscover" the target configuration anytime a session is terminated unexpectedly.

For initiators, in the long term, it is expected that operating system vendors will take on the role of this entity and provide standard APIs that can inform components of their iSCSI Node Name and can configure and/or coordinate ISID allocation, use, and reuse.

Recognizing that such initiator APIs are not available today, other implementations of the role of this entity are possible. For example, a human may instantiate the (common) Node name as part of the installation process of each iSCSI component involved in session creation and login. This may be done either by pointing the component to a vendor-specific location for this datum or to a system-wide location. The structure of the ISID namespace (see Section 10.12.5 ISID and [RFC3721]) facilitates implementation of the ISID coordination by allowing each component vendor to independently (of other vendor’s components) coordinate allocation, use, and reuse of its own partition of the ISID namespace in a vendor-specific manner. Partitioning of the ISID namespace within initiator portal groups managed by that vendor allows each such initiator portal group to act independently of all other portal groups when selecting an ISID for a login; this facilitates enforcement of the ISID RULE (see Section 3.4.3 Consequences of the Model) at the initiator.

Docker EX1014
Page 108 of 257

A vendor of iSCSI hardware (e.g., NICs or HBAs) intended for use in initiators MUST implement a mechanism for configuring the iSCSI Node Name. Vendors, and administrators must ensure that iSCSI Node Names are unique worldwide. It is therefore important that when one chooses to reuse the iSCSI Node Name of a disabled unit, not to re-assign that name to the original unit unless its worldwide uniqueness can be ascertained again.

In addition, a vendor of iSCSI hardware must implement a mechanism to configure and/or coordinate ISIDs for all sessions managed by multiple instances of that hardware within a given iSCSI Node. Such configuration might be either permanently pre-assigned at the factory (in a necessarily globally unique way), statically assigned (e.g., partitioned across all the NICs at initialization in a locally unique way), or dynamically assigned (e.g., on-line allocator, also in a locally unique way). In the latter two cases, the configuration may be via public APIs (perhaps driven by an independent vendor’s software, such as the OS vendor) or via private APIs driven by the vendor’s own software.

9.2. Autosense and Auto Contingent Allegiance (ACA)

Autosense refers to the automatic return of sense data to the initiator in case a command did not complete successfully. iSCSI initiators and targets MUST support and use autosense.

ACA helps preserve ordered command execution in the presence of errors. As iSCSI can have many commands in-flight between initiator and target, iSCSI initiators and targets SHOULD support ACA.

9.3. iSCSI Timeouts

iSCSI recovery actions are often dependent on iSCSI time-outs being recognized and acted upon before SCSI time-outs. Determining the right time-outs to use for various iSCSI actions (command acknowledgements expected, status acknowledgements, etc.) is very much dependent on infrastructure (hardware, links, TCP/IP stack, iSCSI driver). As a guide, the implementer may use an average Nop-Out/Nop-In turnaround delay multiplied by a "safety factor" (e.g., 4) as a good estimate for the basic delay of the iSCSI stack for a given connection. The safety factor should account for the network load variability. For connection teardown the implementer may want to consider also the TCP common practice for the given infrastructure.

Docker EX1014
Page 109 of 257

Text negotiations MAY also be subject to either time-limits or limits in the number of exchanges. Those SHOULD be generous enough to avoid affecting interoperability (e.g., allowing each key to be negotiated on a separate exchange).

The relation between iSCSI timeouts and SCSI timeouts should also be considered. SCSI timeouts should be longer than iSCSI timeouts plus the time required for iSCSI recovery whenever iSCSI recovery is planned. Alternatively, an implementer may choose to interlock iSCSI timeouts and recovery with SCSI timeouts so that SCSI recovery will become active only where iSCSI is not planned to, or failed to, recover.

The implementer may also want to consider the interaction between various iSCSI exception events - such as a digest failure - and subsequent timeouts. When iSCSI error recovery is active, a digest failure is likely to result in discovering a missing command or data PDU. In these cases, an implementer may want to lower the timeout values to enable faster initiation for recovery procedures.

9.4. Command Retry and Cleaning Old Command Instances

To avoid having old, retried command instances appear in a valid command window after a command sequence number wrap around, the protocol requires (see Section 3.2.2.1 Command Numbering and Acknowledging) that on every connection on which a retry has been issued, a non-immediate command be issued and acknowledged within a 2**31-1 commands interval from the CmdSN of the retried command. This requirement can be fulfilled by an implementation in several ways.

The simplest technique to use is to send a (non-retry) non-immediate SCSI command (or a NOP if no SCSI command is available for a while) after every command retry on the connection on which the retry was attempted. As errors are deemed rare events, this technique is probably the most effective, as it does not involve additional checks at the initiator when issuing commands.

9.5. Synch and Steering Layer and Performance

While a synch and steering layer is optional, an initiator/target that does not have it working against a target/initiator that demands synch and steering may experience performance degradation caused by packet reordering and loss. Providing a synch and steering mechanism is recommended for all high-speed implementations.

Docker EX1014
Page 110 of 257

9.6. Considerations for State-dependent Devices and Long-lasting SCSI Operations

Sequential access devices operate on the principle that the position of the device is based on the last command processed. As such, command processing order and knowledge of whether or not the previous command was processed is of the utmost importance to maintain data integrity. For example, inadvertent retries of SCSI commands when it is not known if the previous SCSI command was processed is a potential data integrity risk.

For a sequential access device, consider the scenario in which a SCSI SPACE command to backspace one filemark is issued and then re-issued due to no status received for the command. If the first SPACE command was actually processed, the re-issued SPACE command, if processed, will cause the position to change. Thus, a subsequent write operation will write data to the wrong position and any previous data at that position will be overwritten.

For a medium changer device, consider the scenario in which an EXCHANGE MEDIUM command (the SOURCE ADDRESS and DESTINATION ADDRESS are the same thus performing a swap) is issued and then re-issued due to no status received for the command. If the first EXCHANGE MEDIUM command was actually processed, the re-issued EXCHANGE MEDIUM command, if processed, will perform the swap again. The net effect is that a swap was not performed thus leaving a data integrity exposure.

All commands that change the state of the device (as in SPACE commands for sequential access devices, and EXCHANGE MEDIUM for medium changer device), MUST be issued as non-immediate commands for deterministic and in order delivery to iSCSI targets.

For many of those state changing commands, the execution model also assumes that the command is executed exactly once. Devices implementing READ POSITION and LOCATE provide a means for SCSI level command recovery and new tape-class devices should support those commands. In their absence a retry at SCSI level is difficult and error recovery at iSCSI level is advisable.

Devices operating on long latency delivery subsystems and performing long lasting SCSI operations may need mechanisms that enable connection replacement while commands are running (e.g., during an extended copy operation).

Docker EX1014
Page 111 of 257

9.6.1. Determining the Proper ErrorRecoveryLevel

The implementation and use of a specific ErrorRecoveryLevel should be determined based on the deployment scenarios of a given iSCSI implementation. Generally, the following factors must be considered before deciding on the proper level of recovery:

a) Application resilience to I/O failures.
b) Required level of availability in the face of transport connection failures.
c) Probability of transport layer "checksum escape". This in turn decides the iSCSI digest failure frequency, and thus the criticality of iSCSI-level error recovery. The details of estimating this probability are outside the scope of this document.

A consideration of the above factors for SCSI tape devices as an example suggests that implementations SHOULD use ErrorRecoveryLevel=1 when transport connection failure is not a concern and SCSI level recovery is unavailable, and ErrorRecoveryLevel=2 when the connection failure is also of high likelihood during a backup/retrieval.

For extended copy operations, implementations SHOULD use ErrorRecoveryLevel=2 whenever there is a relatively high likelihood of connection failure.

10. iSCSI PDU Formats

All multi-byte integers that are specified in formats defined in this document are to be represented in network byte order (i.e., big endian). Any field that appears in this document assumes that the most significant byte is the lowest numbered byte and the most significant bit (within byte or field) is the lowest numbered bit unless specified otherwise.

Any compliant sender MUST set all bits not defined and all reserved fields to zero unless specified otherwise. Any compliant receiver MUST ignore any bit not defined and all reserved fields unless specified otherwise. Receipt of reserved code values in defined fields MUST be reported as a protocol error.

Reserved fields are marked by the word "reserved", some abbreviation of "reserved", or by "." for individual bits when no other form of marking is technically feasible.

Docker EX1014
Page 112 of 257

10.1. iSCSI PDU Length and Padding

iSCSI PDUs are padded to the closest integer number of four byte words. The padding bytes SHOULD be sent as 0.

10.2. PDU Template, Header, and Opcodes

All iSCSI PDUs have one or more header segments and, optionally, a data segment. After the entire header segment group a header-digest MAY follow. The data segment MAY also be followed by a data-digest.

The Basic Header Segment (BHS) is the first segment in all of the iSCSI PDUs. The BHS is a fixed-length 48-byte header segment. It MAY be followed by Additional Header Segments (AHS), a Header-Digest, a Data Segment, and/or a Data-Digest.

Docker EX1014
Page 113 of 257

The overall structure of an iSCSI PDU is as follows:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0/ Basic Header Segment (BHS)                                     /
 +/                                                                /
  +---------------+---------------+---------------+---------------+
48/ Additional Header Segment 1 (AHS)  (optional)                  /
 +/                                                                /
  +---------------+---------------+---------------+---------------+
  / Additional Header Segment 2 (AHS)  (optional)                  /
 +/                                                                /
  +---------------+---------------+---------------+---------------+
----
  +---------------+---------------+---------------+---------------+
  / Additional Header Segment n (AHS)  (optional)                  /
 +/                                                                /
  +---------------+---------------+---------------+---------------+
----
  +---------------+---------------+---------------+---------------+
 k/ Header-Digest (optional)                                       /
 +/                                                                /
  +---------------+---------------+---------------+---------------+
 l/ Data Segment(optional)                                         /
 +/                                                                /
  +---------------+---------------+---------------+---------------+
 m/ Data-Digest (optional)                                         /
 +/                                                                /
  +---------------+---------------+---------------+---------------+
```

All PDU segments and digests are padded to the closest integer number of four byte words. For example, all PDU segments and digests start at a four byte word boundary and the padding ranges from 0 to 3 bytes. The padding bytes SHOULD be sent as 0.

iSCSI response PDUs do not have AH Segments.

10.2.1. Basic Header Segment (BHS)

The BHS is 48 bytes long. The Opcode and DataSegmentLength fields appear in all iSCSI PDUs. In addition, when used, the Initiator Task Tag and Logical Unit Number always appear in the same location in the header.

The format of the BHS is:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|I| Opcode    |F|  Opcode-specific fields                     |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN or Opcode-specific fields                                 |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20/ Opcode-specific fields                                        /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
48
```

10.2.1.1 I

For request PDUs, the I bit set to 1 is an immediate delivery marker.

10.2.1.2. Opcode

The Opcode indicates the type of iSCSI PDU the header encapsulates.

The Opcodes are divided into two categories: initiator opcodes and target opcodes. Initiator opcodes are in PDUs sent by the initiator (request PDUs). Target opcodes are in PDUs sent by the target (response PDUs).

Initiators MUST NOT use target opcodes and targets MUST NOT use initiator opcodes.

Initiator opcodes defined in this specification are:

```
0x00 NOP-Out
0x01 SCSI Command (encapsulates a SCSI Command Descriptor Block)
0x02 SCSI Task Management function request
0x03 Login Request
0x04 Text Request
0x05 SCSI Data-Out (for WRITE operations)
0x06 Logout Request
0x10 SNACK Request
0x1c-0x1e Vendor specific codes
```

Target opcodes are:

0x20 NOP-In
0x21 SCSI Response - contains SCSI status and possibly sense information or other response information.
0x22 SCSI Task Management function response
0x23 Login Response
0x24 Text Response
0x25 SCSI Data-In - for READ operations.
0x26 Logout Response
0x31 Ready To Transfer (R2T) - sent by target when it is ready to receive data.
0x32 Asynchronous Message - sent by target to indicate certain special conditions.
0x3c-0x3e Vendor specific codes
0x3f Reject

All other opcodes are reserved.

10.2.1.3. Final (F) bit

When set to 1 it indicates the final (or only) PDU of a sequence.

10.2.1.4. Opcode-specific Fields

These fields have different meanings for different opcode types.

10.2.1.5. TotalAHSLength

Total length of all AHS header segments in units of four byte words including padding, if any.

The TotalAHSLength is only used in PDUs that have an AHS and MUST be 0 in all other PDUs.

10.2.1.6. DataSegmentLength

This is the data segment payload length in bytes (excluding padding). The DataSegmentLength MUST be 0 whenever the PDU has no data segment.

10.2.1.7. LUN

Some opcodes operate on a specific Logical Unit. The Logical Unit Number (LUN) field identifies which Logical Unit. If the opcode does not relate to a Logical Unit, this field is either ignored or may be used in an opcode specific way. The LUN field is 64-bits and should

be formatted in accordance with [SAM2]. For example, LUN[0] from [SAM2] is BHS byte 8 and so on up to LUN[7] from [SAM2], which is BHS byte 15.

10.2.1.8. Initiator Task Tag

The initiator assigns a Task Tag to each iSCSI task it issues. While a task exists, this tag MUST uniquely identify the task session-wide. SCSI may also use the initiator task tag as part of the SCSI task identifier when the timespan during which an iSCSI initiator task tag must be unique extends over the timespan during which a SCSI task tag must be unique. However, the iSCSI Initiator Task Tag must exist and be unique even for untagged SCSI commands.

10.2.2. Additional Header Segment (AHS)

The general format of an AHS is:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0| AHSLength                     | AHSType       | AHS-Specific  |
  +---------------+---------------+---------------+---------------+
 4/ AHS-Specific                                                 /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
 x
```

10.2.2.1. AHSType

The AHSType field is coded as follows:

bit 0-1 - Reserved

bit 2-7 - AHS code

0 - Reserved
1 - Extended CDB
2 - Expected Bidirectional Read Data Length
3 - 63 Reserved

10.2.2.2. AHSLength

This field contains the effective length in bytes of the AHS excluding AHSType and AHSLength and padding, if any. The AHS is padded to the smallest integer number of 4 byte words (i.e., from 0 up to 3 padding bytes).

10.2.2.3. Extended CDB AHS

The format of the Extended CDB AHS is:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0| AHSLength (CDBLength-15)      | 0x01          | Reserved      |
  +---------------+---------------+---------------+---------------+
 4/ ExtendedCDB...+padding                                       /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
 x
```

This type of AHS MUST NOT be used if the CDBLength is less than 17. The length includes the reserved byte 3.

10.2.2.4. Bidirectional Expected Read-Data Length AHS

The format of the Bidirectional Read Expected Data Transfer Length AHS is:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0| AHSLength (0x0005)            | 0x02          | Reserved      |
  +---------------+---------------+---------------+---------------+
 4| Expected Read-Data Length                                     |
  +---------------+---------------+---------------+---------------+
 8
```

10.2.3. Header Digest and Data Digest

Optional header and data digests protect the integrity of the header and data, respectively. The digests, if present, are located, respectively, after the header and PDU-specific data, and cover respectively the header and the PDU data, each including the padding bytes, if any.

The existence and type of digests are negotiated during the Login Phase.

The separation of the header and data digests is useful in iSCSI routing applications, in which only the header changes when a message is forwarded. In this case, only the header digest should be recalculated.

Digests are not included in data or header length fields.

A zero-length Data Segment also implies a zero-length data-digest.

10.2.4. Data Segment

The (optional) Data Segment contains PDU associated data. Its payload effective length is provided in the BHS field - DataSegmentLength. The Data Segment is also padded to an integer number of 4 byte words.

10.3. SCSI Command

The format of the SCSI Command PDU is:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|I| 0x01      |F|R|W|. .|ATTR | Reserved                      |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| Logical Unit Number (LUN)                                     |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Expected Data Transfer Length                                 |
  +---------------+---------------+---------------+---------------+
24| CmdSN                                                         |
  +---------------+---------------+---------------+---------------+
28| ExpStatSN                                                     |
  +---------------+---------------+---------------+---------------+
32/ SCSI Command Descriptor Block (CDB)                           /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
48/ AHS (Optional)                                                /
  +---------------+---------------+---------------+---------------+
 x/ Header Digest (Optional)                                      /
  +---------------+---------------+---------------+---------------+
 y/ (DataSegment, Command Data) (Optional)                        /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
 z/ Data Digest (Optional)                                        /
  +---------------+---------------+---------------+---------------+
```

10.3.1. Flags and Task Attributes (byte 1)

The flags for a SCSI Command are:

bit 0 (F) is set to 1 when no unsolicited SCSI Data-Out PDUs follow this PDU. When F=1 for a write and if Expected Data Transfer Length is larger than the DataSegmentLength, the target may solicit additional data through R2T.

bit 1 (R) is set to 1 when the command is expected to input data.

bit 2 (W) is set to 1 when the command is expected to output data.

bit 3-4 Reserved.

bit 5-7 contains Task Attributes.

Task Attributes (ATTR) have one of the following integer values (see [SAM2] for details):

0 - Untagged
1 - Simple
2 - Ordered
3 - Head of Queue
4 - ACA
5-7 - Reserved

Setting both the W and the F bit to 0 is an error. Either or both of R and W MAY be 1 when either the Expected Data Transfer Length and/or Bidirectional Read Expected Data Transfer Length are 0, but they MUST NOT both be 0 when the Expected Data Transfer Length and/or Bidirectional Read Expected Data Transfer Length are not 0 (i.e., when some data transfer is expected the transfer direction is indicated by the R and/or W bit).

10.3.2. CmdSN - Command Sequence Number

Enables ordered delivery across multiple connections in a single session.

10.3.3. ExpStatSN

Command responses up to ExpStatSN-1 (mod 2**32) have been received (acknowledges status) on the connection.

10.3.4. Expected Data Transfer Length

For unidirectional operations, the Expected Data Transfer Length field contains the number of bytes of data involved in this SCSI operation. For a unidirectional write operation (W flag set to 1 and R flag set to 0), the initiator uses this field to specify the number of bytes of data it expects to transfer for this operation. For a unidirectional read operation (W flag set to 0 and R flag set to 1), the initiator uses this field to specify the number of bytes of data it expects the target to transfer to the initiator. It corresponds to the SAM2 byte count.

For bidirectional operations (both R and W flags are set to 1), this field contains the number of data bytes involved in the write transfer. For bidirectional operations, an additional header segment MUST be present in the header sequence that indicates the Bidirectional Read Expected Data Transfer Length. The Expected Data Transfer Length field and the Bidirectional Read Expected Data Transfer Length field correspond to the SAM2 byte count

If the Expected Data Transfer Length for a write and the length of the immediate data part that follows the command (if any) are the same, then no more data PDUs are expected to follow. In this case, the F bit MUST be set to 1.

If the Expected Data Transfer Length is higher than the FirstBurstLength (the negotiated maximum amount of unsolicited data the target will accept), the initiator MUST send the maximum amount of unsolicited data OR ONLY the immediate data, if any.

Upon completion of a data transfer, the target informs the initiator (through residual counts) of how many bytes were actually processed (sent and/or received) by the target.

10.3.5. CDB - SCSI Command Descriptor Block

There are 16 bytes in the CDB field to accommodate the commonly used CDBs. Whenever the CDB is larger than 16 bytes, an Extended CDB AHS MUST be used to contain the CDB spillover.

10.3.6. Data Segment - Command Data

Some SCSI commands require additional parameter data to accompany the SCSI command. This data may be placed beyond the boundary of the iSCSI header in a data segment. Alternatively, user data (e.g., from a WRITE operation) can be placed in the data segment (both cases are

Docker EX1014
Page 121 of 257

referred to as immediate data). These data are governed by the rules for solicited vs. unsolicited data outlined in Section 3.2.4.2 Data Transfer Overview.

10.4. SCSI Response

The format of the SCSI Response PDU is:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x21      |1|. .|o|u|O|U|.| Response      | Status        |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| Reserved                                                      |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| SNACK Tag or Reserved                                         |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| ExpDataSN or Reserved                                         |
  +---------------+---------------+---------------+---------------+
40| Bidirectional Read Residual Count or Reserved                 |
  +---------------+---------------+---------------+---------------+
44| Residual Count or Reserved                                    |
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
  / Data Segment (Optional)                                       /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
  | Data-Digest (Optional)                                        |
  +---------------+---------------+---------------+---------------+
```

10.4.1. Flags (byte 1)

bit 1-2 Reserved.

bit 3 - (o) set for Bidirectional Read Residual Overflow. In this case, the Bidirectional Read Residual Count indicates the number of bytes that were not transferred to the initiator because the initiator's Expected Bidirectional Read Data Transfer Length was not sufficient.

bit 4 - (u) set for Bidirectional Read Residual Underflow. In this case, the Bidirectional Read Residual Count indicates the number of bytes that were not transferred to the initiator out of the number of bytes expected to be transferred.

bit 5 - (O) set for Residual Overflow. In this case, the Residual Count indicates the number of bytes that were not transferred because the initiator's Expected Data Transfer Length was not sufficient. For a bidirectional operation, the Residual Count contains the residual for the write operation.

bit 6 - (U) set for Residual Underflow. In this case, the Residual Count indicates the number of bytes that were not transferred out of the number of bytes that were expected to be transferred. For a bidirectional operation, the Residual Count contains the residual for the write operation.

bit 7 - (0) Reserved.

Bits O and U and bits o and u are mutually exclusive (i.e., having both o and u or O and U set to 1 is a protocol error). For a response other than "Command Completed at Target", bits 3-6 MUST be 0.

10.4.2. Status

The Status field is used to report the SCSI status of the command (as specified in [SAM2]) and is only valid if the Response Code is Command Completed at target.

Some of the status codes defined in [SAM2] are:

```
0x00 GOOD
0x02 CHECK CONDITION
0x08 BUSY
0x18 RESERVATION CONFLICT
0x28 TASK SET FULL
0x30 ACA ACTIVE
0x40 TASK ABORTED
```

See [SAM2] for the complete list and definitions.

If a SCSI device error is detected while data from the initiator is still expected (the command PDU did not contain all the data and the target has not received a Data PDU with the final bit Set), the target MUST wait until it receives a Data PDU with the F bit set in the last expected sequence before sending the Response PDU.

10.4.3. Response

This field contains the iSCSI service response.

iSCSI service response codes defined in this specification are:

```
0x00 - Command Completed at Target
0x01 - Target Failure
0x80-0xff - Vendor specific
```

All other response codes are reserved.

The Response is used to report a Service Response. The mapping of the response code into a SCSI service response code value, if needed, is outside the scope of this document. However, in symbolic terms response value 0x00 maps to the SCSI service response (see [SAM2] and [SPC3]) of TASK COMPLETE or LINKED COMMAND COMPLETE. All other Response values map to the SCSI service response of SERVICE DELIVERY OR TARGET FAILURE.

If a PDU that includes SCSI status (Response PDU or Data-In PDU including status) does not arrive before the session is terminated, the SCSI service response is SERVICE DELIVERY OR TARGET FAILURE.

A non-zero Response field indicates a failure to execute the command in which case the Status and Flag fields are undefined.

Docker EX1014
Page 124 of 257

10.4.4. SNACK Tag

This field contains a copy of the SNACK Tag of the last SNACK Tag accepted by the target on the same connection and for the command for which the response is issued. Otherwise it is reserved and should be set to 0.

After issuing a R-Data SNACK the initiator must discard any SCSI status unless contained in an SCSI Response PDU carrying the same SNACK Tag as the last issued R-Data SNACK for the SCSI command on the current connection.

For a detailed discussion on R-Data SNACK see Section 10.16 SNACK Request.

10.4.5. Residual Count

The Residual Count field MUST be valid in the case where either the U bit or the O bit is set. If neither bit is set, the Residual Count field is reserved. Targets may set the residual count and initiators may use it when the response code is "completed at target" (even if the status returned is not GOOD). If the O bit is set, the Residual Count indicates the number of bytes that were not transferred because the initiator’s Expected Data Transfer Length was not sufficient. If the U bit is set, the Residual Count indicates the number of bytes that were not transferred out of the number of bytes expected to be transferred.

10.4.6. Bidirectional Read Residual Count

The Bidirectional Read Residual Count field MUST be valid in the case where either the u bit or the o bit is set. If neither bit is set, the Bidirectional Read Residual Count field is reserved. Targets may set the Bidirectional Read Residual Count and initiators may use it when the response code is "completed at target". If the o bit is set, the Bidirectional Read Residual Count indicates the number of bytes that were not transferred to the initiator because the initiator’s Expected Bidirectional Read Transfer Length was not sufficient. If the u bit is set, the Bidirectional Read Residual Count indicates the number of bytes that were not transferred to the initiator out of the number of bytes expected to be transferred.

10.4.7. Data Segment - Sense and Response Data Segment

iSCSI targets MUST support and enable autosense. If Status is CHECK CONDITION (0x02), then the Data Segment MUST contain sense data for the failed command.

Docker EX1014
Page 125 of 257

For some iSCSI responses, the response data segment MAY contain some response related information, (e.g., for a target failure, it may contain a vendor specific detailed description of the failure).

If the DataSegmentLength is not 0, the format of the Data Segment is as follows:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|SenseLength                    | Sense Data                    |
  +---------------+---------------+---------------+---------------+
 x/ Sense Data                                                   /
  +---------------+---------------+---------------+---------------+
 y/ Response Data                                                /
  /                                                              /
  +---------------+---------------+---------------+---------------+
 z|
```

10.4.7.1. SenseLength

Length of Sense Data.

10.4.7.2. Sense Data

The Sense Data contains detailed information about a check condition and [SPC3] specifies the format and content of the Sense Data.

Certain iSCSI conditions result in the command being terminated at the target (response Command Completed at Target) with a SCSI Check Condition Status as outlined in the next table:

iSCSI Condition	Sense Key	Additional Sense Code & Qualifier
Unexpected unsolicited data	Aborted Command-0B	ASC = 0x0c ASCQ = 0x0c Write Error
Incorrect amount of data	Aborted Command-0B	ASC = 0x0c ASCQ = 0x0d Write Error
Protocol Service CRC error	Aborted Command-0B	ASC = 0x47 ASCQ = 0x05 CRC Error Detected
SNACK rejected	Aborted Command-0B	ASC = 0x11 ASCQ = 0x13 Read Error

The target reports the "Incorrect amount of data" condition if during data output the total data length to output is greater than FirstBurstLength and the initiator sent unsolicited non-immediate data but the total amount of unsolicited data is different than FirstBurstLength. The target reports the same error when the amount of data sent as a reply to an R2T does not match the amount requested.

10.4.8. ExpDataSN

The number of R2T and Data-In (read) PDUs the target has sent for the command.

This field MUST be 0 if the response code is not Command Completed at Target or the target sent no Data-In PDUs for the command.

10.4.9. StatSN - Status Sequence Number

StatSN is a Sequence Number that the target iSCSI layer generates per connection and that in turn, enables the initiator to acknowledge status reception. StatSN is incremented by 1 for every response/status sent on a connection except for responses sent as a result of a retry or SNACK. In the case of responses sent due to a retransmission request, the StatSN MUST be the same as the first time the PDU was sent unless the connection has since been restarted.

10.4.10. ExpCmdSN - Next Expected CmdSN from this Initiator

ExpCmdSN is a Sequence Number that the target iSCSI returns to the initiator to acknowledge command reception. It is used to update a local variable with the same name. An ExpCmdSN equal to MaxCmdSN+1 indicates that the target cannot accept new commands.

10.4.11. MaxCmdSN - Maximum CmdSN from this Initiator

MaxCmdSN is a Sequence Number that the target iSCSI returns to the initiator to indicate the maximum CmdSN the initiator can send. It is used to update a local variable with the same name. If MaxCmdSN is equal to ExpCmdSN-1, this indicates to the initiator that the target cannot receive any additional commands. When MaxCmdSN changes at the target while the target has no pending PDUs to convey this information to the initiator, it MUST generate a NOP-IN to carry the new MaxCmdSN.

Docker EX1014
Page 128 of 257

10.5. Task Management Function Request

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|I| 0x02      |1| Function    | Reserved                    |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                            |
  +---------------+---------------+---------------+---------------+
 8| Logical Unit Number (LUN) or Reserved                         |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Referenced Task Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| CmdSN                                                         |
  +---------------+---------------+---------------+---------------+
28| ExpStatSN                                                     |
  +---------------+---------------+---------------+---------------+
32| RefCmdSN or Reserved                                          |
  +---------------+---------------+---------------+---------------+
36| ExpDataSN or Reserved                                         |
  +---------------+---------------+---------------+---------------+
40/ Reserved                                                     /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
```

10.5.1. Function

The Task Management functions provide an initiator with a way to explicitly control the execution of one or more Tasks (SCSI and iSCSI tasks). The Task Management function codes are listed below. For a more detailed description of SCSI task management, see [SAM2].

1 - ABORT TASK - aborts the task identified by the Referenced Task Tag field.

2 - ABORT TASK SET - aborts all Tasks issued via this session on the logical unit.

3 - CLEAR ACA - clears the Auto Contingent Allegiance condition.

4 - CLEAR TASK SET - aborts all Tasks in the appropriate task set as defined by the TST field in the Control mode page (see [SPC3]).

5 - LOGICAL UNIT RESET

6 - TARGET WARM RESET

7 - TARGET COLD RESET

8 - TASK REASSIGN - reassigns connection allegiance for the task identified by the Referenced Task Tag field to this connection, thus resuming the iSCSI exchanges for the task.

For all these functions, the Task Management function response MUST be returned as detailed in Section 10.6 Task Management Function Response. All these functions apply to the referenced tasks regardless of whether they are proper SCSI tasks or tagged iSCSI operations. Task management requests must act on all the commands from the same session having a CmdSN lower than the task management CmdSN. LOGICAL UNIT RESET, TARGET WARM RESET and TARGET COLD RESET may affect commands from other sessions or commands from the same session with CmdSN equal or exceeding CmdSN.

If the task management request is marked for immediate delivery, it must be considered immediately for execution, but the operations involved (all or part of them) may be postponed to allow the target to receive all relevant tasks. According to [SAM2], for all the tasks covered by the Task Management response (i.e., with CmdSN lower than the task management command CmdSN) but except the Task Management response to a TASK REASSIGN, additional responses MUST NOT be delivered to the SCSI layer after the Task Management response. The iSCSI initiator MAY deliver to the SCSI layer all responses received before the Task Management response (i.e., it is a matter of implementation if the SCSI responses, received before the Task Management response but after the task management request was issued, are delivered to the SCSI layer by the iSCSI layer in the initiator). The iSCSI target MUST ensure that no responses for the tasks covered by a task management function are delivered to the iSCSI initiator after the Task Management response except for a task covered by a TASK REASSIGN.

For ABORT TASK SET and CLEAR TASK SET, the issuing initiator MUST continue to respond to all valid target transfer tags (received via R2T, Text Response, NOP-In, or SCSI Data-In PDUs) related to the affected task set, even after issuing the task management request. The issuing initiator SHOULD however terminate (i.e., by setting the F-bit to 1) these response sequences as quickly as possible. The target on its part MUST wait for responses on all affected target

Docker EX1014
Page 130 of 257

transfer tags before acting on either of these two task management requests. In case all or part of the response sequence is not received (due to digest errors) for a valid TTT, the target MAY treat it as a case of within-command error recovery class (see Section 6.1.4.1 Recovery Within-command) if it is supporting ErrorRecoveryLevel >= 1, or alternatively may drop the connection to complete the requested task set function.

If an ABORT TASK is issued for a task created by an immediate command then RefCmdSN MUST be that of the Task Management request itself (i.e., CmdSN and RefCmdSN are equal); otherwise RefCmdSN MUST be set to the CmdSN of the task to be aborted (lower than CmdSN).

If the connection is still active (it is not undergoing an implicit or explicit logout), ABORT TASK MUST be issued on the same connection to which the task to be aborted is allegiant at the time the Task Management Request is issued. If the connection is implicitly or explicitly logged out (i.e., no other request will be issued on the failing connection and no other response will be received on the failing connection), then an ABORT TASK function request may be issued on another connection. This Task Management request will then establish a new allegiance for the command to be aborted as well as abort it (i.e., the task to be aborted will not have to be retried or reassigned, and its status, if issued but not acknowledged, will be reissued followed by the Task Management response).

At the target an ABORT TASK function MUST NOT be executed on a Task Management request; such a request MUST result in Task Management response of "Function rejected".

For the LOGICAL UNIT RESET function, the target MUST behave as dictated by the Logical Unit Reset function in [SAM2].

The implementation of the TARGET WARM RESET function and the TARGET COLD RESET function is OPTIONAL and when implemented, should act as described below. The TARGET WARM RESET is also subject to SCSI access controls on the requesting initiator as defined in [SPC3]. When authorization fails at the target, the appropriate response as described in Section 10.6 Task Management Function Response MUST be returned by the target. The TARGET COLD RESET function is not subject to SCSI access controls, but its execution privileges may be managed by iSCSI mechanisms such as login authentication.

When executing the TARGET WARM RESET and TARGET COLD RESET functions, the target cancels all pending operations on all Logical Units known by the issuing initiator. Both functions are equivalent to the Target Reset function specified by [SAM2]. They can affect many other initiators logged in with the servicing SCSI target port.

Docker EX1014
Page 131 of 257

The target MUST treat the TARGET COLD RESET function additionally as a power on event, thus terminating all of its TCP connections to all initiators (all sessions are terminated). For this reason, the Service Response (defined by [SAM2]) for this SCSI task management function may not be reliably delivered to the issuing initiator port.

For the TASK REASSIGN function, the target should reassign the connection allegiance to this new connection (and thus resume iSCSI exchanges for the task). TASK REASSIGN MUST ONLY be received by the target after the connection on which the command was previously executing has been successfully logged-out. The Task Management response MUST be issued before the reassignment becomes effective. For additional usage semantics see Section 6.2 Retry and Reassign in Recovery.

At the target a TASK REASSIGN function request MUST NOT be executed to reassign the connection allegiance of a Task Management function request, an active text negotiation task, or a Logout task; such a request MUST result in Task Management response of "Function rejected".

TASK REASSIGN MUST be issued as an immediate command.

10.5.2. TotalAHSLength and DataSegmentLength

For this PDU TotalAHSLength and DataSegmentLength MUST be 0.

10.5.3. LUN

This field is required for functions that address a specific LU (ABORT TASK, CLEAR TASK SET, ABORT TASK SET, CLEAR ACA, LOGICAL UNIT RESET) and is reserved in all others.

10.5.4. Referenced Task Tag

The Initiator Task Tag of the task to be aborted for the ABORT TASK function or reassigned for the TASK REASSIGN function. For all the other functions this field MUST be set to the reserved value 0xffffffff.

10.5.5. RefCmdSN

If an ABORT TASK is issued for a task created by an immediate command then RefCmdSN MUST be that of the Task Management request itself (i.e., CmdSN and RefCmdSN are equal).

Docker EX1014
Page 132 of 257

For an ABORT TASK of a task created by non-immediate command RefCmdSN MUST be set to the CmdSN of the task identified by the Referenced Task Tag field. Targets must use this field as described in section 10.6.1 when the task identified by the Referenced Task Tag field is not with the target.

Otherwise, this field is reserved.

10.5.6. ExpDataSN

For recovery purposes, the iSCSI target and initiator maintain a data acknowledgement reference number - the first input DataSN number unacknowledged by the initiator. When issuing a new command, this number is set to 0. If the function is TASK REASSIGN, which establishes a new connection allegiance for a previously issued Read or Bidirectional command, ExpDataSN will contain an updated data acknowledgement reference number or the value 0; the latter indicating that the data acknowledgement reference number is unchanged. The initiator MUST discard any data PDUs from the previous execution that it did not acknowledge and the target MUST transmit all Data-In PDUs (if any) starting with the data acknowledgement reference number. The number of retransmitted PDUs may or may not be the same as the original transmission depending on if there was a change in MaxRecvDataSegmentLength in the reassignment. The target MAY also send no more Data-In PDUs if all data has been acknowledged.

The value of ExpDataSN MUST be 0 or higher than the DataSN of the last acknowledged Data-In PDU, but not larger than DataSN+1 of the last Data-In PDU sent by the target. Any other value MUST be ignored by the target.

For other functions this field is reserved.

10.6. Task Management Function Response

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x22      |1| Reserved    | Response      | Reserved      |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------------------------------------------------------+
 8/ Reserved                                                     /
  /                                                              /
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36/ Reserved                                                     /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
```

For the functions ABORT TASK, ABORT TASK SET, CLEAR ACA, CLEAR TASK SET, LOGICAL UNIT RESET, TARGET COLD RESET, TARGET WARM RESET and TASK REASSIGN, the target performs the requested Task Management function and sends a Task Management response back to the initiator. For TASK REASSIGN, the new connection allegiance MUST ONLY become effective at the target after the target issues the Task Management Response.

10.6.1. Response

The target provides a Response, which may take on the following values:

a) 0 - Function complete.
b) 1 - Task does not exist.
c) 2 - LUN does not exist.
d) 3 - Task still allegiant.
e) 4 - Task allegiance reassignment not supported.

f) 5 - Task management function not supported.
g) 6 - Function authorization failed.
h) 255 - Function rejected.

All other values are reserved.

For a discussion on usage of response codes 3 and 4, see Section 6.2.2 Allegiance Reassignment.

For the TARGET COLD RESET and TARGET WARM RESET functions, the target cancels all pending operations across all Logical Units known to the issuing initiator. For the TARGET COLD RESET function, the target MUST then close all of its TCP connections to all initiators (terminates all sessions).

The mapping of the response code into a SCSI service response code value, if needed, is outside the scope of this document. However, in symbolic terms Response values 0 and 1 map to the SCSI service response of FUNCTION COMPLETE. All other Response values map to the SCSI service response of FUNCTION REJECTED. If a Task Management function response PDU does not arrive before the session is terminated, the SCSI service response is SERVICE DELIVERY OR TARGET FAILURE.

The response to ABORT TASK SET and CLEAR TASK SET MUST only be issued by the target after all of the commands affected have been received by the target, the corresponding task management functions have been executed by the SCSI target, and the delivery of all responses delivered until the task management function completion have been confirmed (acknowledged through ExpStatSN) by the initiator on all connections of this session. For the exact timeline of events, refer to Section 10.6.2 Task Management Actions on Task Sets.

For the ABORT TASK function,

a) If the Referenced Task Tag identifies a valid task leading to a successful termination, then targets must return the "Function complete" response.
b) If the Referenced Task Tag does not identify an existing task, but if the CmdSN indicated by the RefCmdSN field in the Task Management function request is within the valid CmdSN window and less than the CmdSN of the Task Management function request itself, then targets must consider the CmdSN received and return the "Function complete" response.

Docker EX1014
Page 135 of 257

c) If the Referenced Task Tag does not identify an existing task and if the CmdSN indicated by the RefCmdSN field in the Task Management function request is outside the valid CmdSN window, then targets must return the "Task does not exist" response.

10.6.2. Task Management Actions on Task Sets

The execution of ABORT TASK SET and CLEAR TASK SET Task Management function requests consists of the following sequence of events in the specified order on each of the entities.

The initiator:

a) Issues ABORT TASK SET/CLEAR TASK SET request.
b) Continues to respond to each target transfer tag received for the affected task set.
c) Receives any responses for the tasks in the affected task set (may process them as usual because they are guaranteed to be valid).
d) Receives the task set management response, thus concluding all the tasks in the affected task set.

The target:

a) Receives the ABORT TASK SET/CLEAR TASK SET request.
b) Waits for all target transfer tags to be responded to and for all affected tasks in the task set to be received.
c) Propagates the command to and receives the response from the target SCSI layer.
d) Takes note of last-sent StatSN on each of the connections in the iSCSI sessions (one or more) sharing the affected task set, and waits for acknowledgement of each StatSN (may solicit for acknowledgement by way of a NOP-In). If some tasks originate from non-iSCSI I_T_L nexi then the means by which the target insures that all affected tasks have returned their status to the initiator are defined by the specific protocol.

e) Sends the task set management response to the issuing initiator.

10.6.3. TotalAHSLength and DataSegmentLength

For this PDU TotalAHSLength and DataSegmentLength MUST be 0.

10.7. SCSI Data-Out & SCSI Data-In

The SCSI Data-Out PDU for WRITE operations has the following format:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x05      |F| Reserved                                    |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN or Reserved                                               |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
28| ExpStatSN                                                     |
  +---------------+---------------+---------------+---------------+
32| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
36| DataSN                                                        |
  +---------------+---------------+---------------+---------------+
40| Buffer Offset                                                 |
  +---------------+---------------+---------------+---------------+
44| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
  / DataSegment                                                   /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
  | Data-Digest (Optional)                                        |
  +---------------+---------------+---------------+---------------+
```

The SCSI Data-In PDU for READ operations has the following format:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x25      |F|A|0 0 0|O|U|S| Reserved      |Status or Rsvd |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                            |
  +---------------+---------------+---------------+---------------+
 8| LUN or Reserved                                               |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| StatSN or Reserved                                            |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| DataSN                                                        |
  +---------------+---------------+---------------+---------------+
40| Buffer Offset                                                 |
  +---------------+---------------+---------------+---------------+
44| Residual Count                                                |
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
  / DataSegment                                                   /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
  | Data-Digest (Optional)                                        |
  +---------------+---------------+---------------+---------------+
```

Status can accompany the last Data-In PDU if the command did not end with an exception (i.e., the status is "good status" - GOOD, CONDITION MET or INTERMEDIATE CONDITION MET). The presence of status (and of a residual count) is signaled though the S flag bit. Although targets MAY choose to send even non-exception status in separate responses, initiators MUST support non-exception status in Data-In PDUs.

10.7.1. F (Final) Bit

For outgoing data, this bit is 1 for the last PDU of unsolicited data or the last PDU of a sequence that answers an R2T.

For incoming data, this bit is 1 for the last input (read) data PDU of a sequence. Input can be split into several sequences, each having its own F bit. Splitting the data stream into sequences does not affect DataSN counting on Data-In PDUs. It MAY be used as a "change direction" indication for Bidirectional operations that need such a change.

DataSegmentLength MUST not exceed MaxRecvDataSegmentLength for the direction it is sent and the total of all the DataSegmentLength of all PDUs in a sequence MUST not exceed MaxBurstLength (or FirstBurstLength for unsolicited data). However the number of individual PDUs in a sequence (or in total) may be higher than the MaxBurstLength (or FirstBurstLength) to MaxRecvDataSegmentLength ratio (as PDUs may be limited in length by the sender capabilities). Using DataSegmentLength of 0 may increase beyond what is reasonable for the number of PDUs and should therefore be avoided.

For Bidirectional operations, the F bit is 1 for both the end of the input sequences and the end of the output sequences.

10.7.2. A (Acknowledge) Bit

For sessions with ErrorRecoveryLevel 1 or higher, the target sets this bit to 1 to indicate that it requests a positive acknowledgement from the initiator for the data received. The target should use the A bit moderately; it MAY only set the A bit to 1 once every MaxBurstLength bytes, or on the last Data-In PDU that concludes the entire requested read data transfer for the task from the target’s perspective, and it MUST NOT do so more frequently. The target MUST NOT set to 1 the A bit for sessions with ErrorRecoveryLevel=0. The initiator MUST ignore the A bit set to 1 for sessions with ErrorRecoveryLevel=0.

On receiving a Data-In PDU with the A bit set to 1 on a session with ErrorRecoveryLevel greater than 0, if there are no holes in the read data until that Data-In PDU, the initiator MUST issue a SNACK of type DataACK except when it is able to acknowledge the status for the task immediately via ExpStatSN on other outbound PDUs if the status for the task is also received. In the latter case (acknowledgement through ExpStatSN), sending a SNACK of type DataACK in response to the A bit is OPTIONAL, but if it is done, it must not be sent after the status acknowledgement through ExpStatSN. If the initiator has detected holes in the read data prior to that Data-In PDU, it MUST

Docker EX1014
Page 139 of 257

postpone issuing the SNACK of type DataACK until the holes are filled. An initiator also MUST NOT acknowledge the status for the task before those holes are filled. A status acknowledgement for a task that generated the Data-In PDUs is considered by the target as an implicit acknowledgement of the Data-In PDUs if such an acknowledgement was requested by the target.

10.7.3. Flags (byte 1)

The last SCSI Data packet sent from a target to an initiator for a SCSI command that completed successfully (with a status of GOOD, CONDITION MET, INTERMEDIATE or INTERMEDIATE CONDITION MET) may also optionally contain the Status for the data transfer. As Sense Data cannot be sent together with the Command Status, if the command is completed with an error, then the response and sense data MUST be sent in a SCSI Response PDU (i.e., MUST NOT be sent in a SCSI Data packet). If Status is sent with the data, then a SCSI Response PDU MUST NOT be sent as this would violate SCSI rules (a single status). For Bidirectional commands, the status MUST be sent in a SCSI Response PDU.

bit 2-4 - Reserved.

bit 5-6 - used the same as in a SCSI Response. These bits are only valid when S is set to 1. For details see Section 10.4.1 Flags (byte 1).

bit 7 S (status)- set to indicate that the Command Status field contains status. If this bit is set to 1, the F bit MUST also be set to 1.

The fields StatSN, Status, and Residual Count only have meaningful content if the S bit is set to 1 and their values are defined in Section 10.4 SCSI Response.

10.7.4. Target Transfer Tag and LUN

On outgoing data, the Target Transfer Tag is provided to the target if the transfer is honoring an R2T. In this case, the Target Transfer Tag field is a replica of the Target Transfer Tag provided with the R2T.

On incoming data, the Target Transfer Tag and LUN MUST be provided by the target if the A bit is set to 1; otherwise they are reserved. The Target Transfer Tag and LUN are copied by the initiator into the SNACK of type DataACK that it issues as a result of receiving a SCSI Data-In PDU with the A bit set to 1.

Docker EX1014
Page 140 of 257

The Target Transfer Tag values are not specified by this protocol except that the value 0xffffffff is reserved and means that the Target Transfer Tag is not supplied. If the Target Transfer Tag is provided, then the LUN field MUST hold a valid value and be consistent with whatever was specified with the command; otherwise, the LUN field is reserved.

10.7.5. DataSN

For input (read) or bidirectional Data-In PDUs, the DataSN is the input PDU number within the data transfer for the command identified by the Initiator Task Tag.

R2T and Data-In PDUs, in the context of bidirectional commands, share the numbering sequence (see Section 3.2.2.3 Data Sequencing).

For output (write) data PDUs, the DataSN is the Data-Out PDU number within the current output sequence. The current output sequence is either identified by the Initiator Task Tag (for unsolicited data) or is a data sequence generated for one R2T (for data solicited through R2T).

10.7.6. Buffer Offset

The Buffer Offset field contains the offset of this PDU payload data within the complete data transfer. The sum of the buffer offset and length should not exceed the expected transfer length for the command.

The order of data PDUs within a sequence is determined by DataPDUInOrder. When set to Yes, it means that PDUs have to be in increasing Buffer Offset order and overlays are forbidden.

The ordering between sequences is determined by DataSequenceInOrder. When set to Yes, it means that sequences have to be in increasing Buffer Offset order and overlays are forbidden.

10.7.7. DataSegmentLength

This is the data payload length of a SCSI Data-In or SCSI Data-Out PDU. The sending of 0 length data segments should be avoided, but initiators and targets MUST be able to properly receive 0 length data segments.

The Data Segments of Data-In and Data-Out PDUs SHOULD be filled to the integer number of 4 byte words (real payload) unless the F bit is set to 1.

Docker EX1014
Page 141 of 257

10.8. Ready To Transfer (R2T)

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x31      |1| Reserved                                    |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN                                                           |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag                                           |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| R2TSN                                                         |
  +---------------+---------------+---------------+---------------+
40| Buffer Offset                                                 |
  +---------------+---------------+---------------+---------------+
44| Desired Data Transfer Length                                  |
  +---------------------------------------------------------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
```

When an initiator has submitted a SCSI Command with data that passes from the initiator to the target (WRITE), the target may specify which blocks of data it is ready to receive. The target may request that the data blocks be delivered in whichever order is convenient for the target at that particular instant. This information is passed from the target to the initiator in the Ready To Transfer (R2T) PDU.

In order to allow write operations without an explicit initial R2T, the initiator and target MUST have negotiated the key InitialR2T to No during Login.

An R2T MAY be answered with one or more SCSI Data-Out PDUs with a matching Target Transfer Tag. If an R2T is answered with a single Data-Out PDU, the Buffer Offset in the Data PDU MUST be the same as

the one specified by the R2T, and the data length of the Data PDU MUST be the same as the Desired Data Transfer Length specified in the R2T. If the R2T is answered with a sequence of Data PDUs, the Buffer Offset and Length MUST be within the range of those specified by R2T, and the last PDU MUST have the F bit set to 1. If the last PDU (marked with the F bit) is received before the Desired Data Transfer Length is transferred, a target MAY choose to Reject that

PDU with "Protocol error" reason code. DataPDUInOrder governs the Data-Out PDU ordering. If DataPDUInOrder is set to Yes, the Buffer Offsets and Lengths for consecutive PDUs MUST form a continuous non-overlapping range and the PDUs MUST be sent in increasing offset order.

The target may send several R2T PDUs. It, therefore, can have a number of pending data transfers. The number of outstanding R2T PDUs are limited by the value of the negotiated key MaxOutstandingR2T. Within a connection, outstanding R2Ts MUST be fulfilled by the initiator in the order in which they were received.

R2T PDUs MAY also be used to recover Data Out PDUs. Such an R2T (Recovery-R2T) is generated by a target upon detecting the loss of one or more Data-Out PDUs due to:

- Digest error
- Sequence error
- Sequence reception timeout

A Recovery-R2T carries the next unused R2TSN, but requests part of or the entire data burst that an earlier R2T (with a lower R2TSN) had already requested.

DataSequenceInOrder governs the buffer offset ordering in consecutive R2Ts. If DataSequenceInOrder is Yes, then consecutive R2Ts MUST refer to continuous non-overlapping ranges except for Recovery-R2Ts.

10.8.1. TotalAHSLength and DataSegmentLength

For this PDU TotalAHSLength and DataSegmentLength MUST be 0.

10.8.2. R2TSN

R2TSN is the R2T PDU input PDU number within the command identified by the Initiator Task Tag.

For bidirectional commands R2T and Data-In PDUs share the input PDU numbering sequence (see Section 3.2.2.3 Data Sequencing).

Docker EX1014
Page 143 of 257

10.8.3. StatSN

The StatSN field will contain the next StatSN. The StatSN for this connection is not advanced after this PDU is sent.

10.8.4. Desired Data Transfer Length and Buffer Offset

The target specifies how many bytes it wants the initiator to send because of this R2T PDU. The target may request the data from the initiator in several chunks, not necessarily in the original order of the data. The target, therefore, also specifies a Buffer Offset that indicates the point at which the data transfer should begin, relative to the beginning of the total data transfer. The Desired Data Transfer Length MUST NOT be 0 and MUST not exceed MaxBurstLength.

10.8.5. Target Transfer Tag

The target assigns its own tag to each R2T request that it sends to the initiator. This tag can be used by the target to easily identify the data it receives. The Target Transfer Tag and LUN are copied in the outgoing data PDUs and are only used by the target. There is no protocol rule about the Target Transfer Tag except that the value 0xffffffff is reserved and MUST NOT be sent by a target in an R2T.

10.9. Asynchronous Message

An Asynchronous Message may be sent from the target to the initiator without correspondence to a particular command. The target specifies the reason for the event and sense data.

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x32      |1| Reserved                                    |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN or Reserved                                               |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| 0xffffffff                                                    |
  +---------------+---------------+---------------+---------------+
20| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| AsyncEvent    | AsyncVCode    | Parameter1 or Reserved        |
  +---------------+---------------+---------------+---------------+
40| Parameter2 or Reserved        | Parameter3 or Reserved        |
  +---------------+---------------+---------------+---------------+
44| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
  / DataSegment - Sense Data and iSCSI Event Data                 /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
  | Data-Digest (Optional)                                        |
  +---------------+---------------+---------------+---------------+
```

Some Asynchronous Messages are strictly related to iSCSI while others are related to SCSI [SAM2].

StatSN counts this PDU as an acknowledgeable event (StatSN is advanced), which allows for initiator and target state synchronization.

10.9.1. AsyncEvent

The codes used for iSCSI Asynchronous Messages (events) are:

0 - a SCSI Asynchronous Event is reported in the sense data. Sense Data that accompanies the report, in the data segment, identifies the condition. The sending of a SCSI Event (Asynchronous Event Reporting in SCSI terminology) is dependent on the target support for SCSI asynchronous event reporting (see [SAM2]) as indicated in the standard INQUIRY data (see [SPC3]). Its use may be enabled by parameters in the SCSI Control mode page (see [SPC3]).

1 - target requests Logout. This Async Message MUST be sent on the same connection as the one requesting to be logged out. The initiator MUST honor this request by issuing a Logout as early as possible, but no later than Parameter3 seconds. Initiator MUST send a Logout with a reason code of "Close the connection" OR "Close the session" to close all the connections. Once this message is received, the initiator SHOULD NOT issue new iSCSI commands on the connection to be logged out. The target MAY reject any new I/O requests that it receives after this Message with the reason code "Waiting for Logout". If the initiator does not Logout in Parameter3 seconds, the target should send an Async PDU with iSCSI event code "Dropped the connection" if possible, or simply terminate the transport connection. Parameter1 and Parameter2 are reserved.

2 - target indicates it will drop the connection. The Parameter1 field indicates the CID of the connection that is going to be dropped.

The Parameter2 field (Time2Wait) indicates, in seconds, the minimum time to wait before attempting to reconnect or reassign.

The Parameter3 field (Time2Retain) indicates the maximum time allowed to reassign commands after the initial wait (in Parameter2).

If the initiator does not attempt to reconnect and/or reassign the outstanding commands within the time specified by Parameter3, or if Parameter3 is 0, the target will terminate all outstanding commands on this connection. In this case, no other responses should be expected from the target for the outstanding commands on this connection.

Docker EX1014
Page 146 of 257

A value of 0 for Parameter2 indicates that reconnect can be attempted immediately.

3 - target indicates it will drop all the connections of this session.

Parameter1 field is reserved.

The Parameter2 field (Time2Wait) indicates, in seconds, the minimum time to wait before attempting to reconnect. The Parameter3 field (Time2Retain) indicates the maximum time allowed to reassign commands after the initial wait (in Parameter2).

If the initiator does not attempt to reconnect and/or reassign the outstanding commands within the time specified by Parameter3, or if Parameter3 is 0, the session is terminated.

In this case, the target will terminate all outstanding commands in this session; no other responses should be expected from the target for the outstanding commands in this session. A value of 0 for Parameter2 indicates that reconnect can be attempted immediately.

4 - target requests parameter negotiation on this connection. The initiator MUST honor this request by issuing a Text Request (that can be empty) on the same connection as early as possible, but no later than Parameter3 seconds, unless a Text Request is already pending on the connection, or by issuing a Logout Request. If the initiator does not issue a Text Request the target may reissue the Asynchronous Message requesting parameter negotiation.

255 - vendor specific iSCSI Event. The AsyncVCode details the vendor code, and data MAY accompany the report.

All other event codes are reserved.

10.9.2. AsyncVCode

AsyncVCode is a vendor specific detail code that is only valid if the AsyncEvent field indicates a vendor specific event. Otherwise, it is reserved.

10.9.3. LUN

The LUN field MUST be valid if AsyncEvent is 0. Otherwise, this field is reserved.

10.9.4. Sense Data and iSCSI Event Data

For a SCSI event, this data accompanies the report in the data segment and identifies the condition.

For an iSCSI event, additional vendor-unique data MAY accompany the Async event. Initiators MAY ignore the data when not understood while processing the rest of the PDU.

If the DataSegmentLength is not 0, the format of the DataSegment is as follows:

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|SenseLength                    | Sense Data                    |
  +---------------+---------------+---------------+---------------+
 x/ Sense Data                                                   /
  +---------------+---------------+---------------+---------------+
 y/ iSCSI Event Data                                             /
  /                                                              /
  +---------------+---------------+---------------+---------------+
 z|
```

10.9.4.1. SenseLength

This is the length of Sense Data. When the Sense Data field is empty (e.g., the event is not a SCSI event) SenseLength is 0.

10.10. Text Request

The Text Request is provided to allow for the exchange of information and for future extensions. It permits the initiator to inform a target of its capabilities or to request some special operations.

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|I| 0x04      |F|C| Reserved                                  |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN or Reserved                                               |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| CmdSN                                                         |
  +---------------+---------------+---------------+---------------+
28| ExpStatSN                                                     |
  +---------------+---------------+---------------+---------------+
32/ Reserved                                                      /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
  / DataSegment (Text)                                            /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
  | Data-Digest (Optional)                                        |
  +---------------+---------------+---------------+---------------+
```

An initiator MUST have at most one outstanding Text Request on a connection at any given time.

On a connection failure, an initiator must either explicitly abort any active allegiant text negotiation task or must cause such a task to be implicitly terminated by the target.

10.10.1. F (Final) Bit

When set to 1, indicates that this is the last or only text request in a sequence of Text Requests; otherwise, it indicates that more Text Requests will follow.

10.10.2. C (Continue) Bit

When set to 1, indicates that the text (set of key=value pairs) in this Text Request is not complete (it will be continued on subsequent Text Requests); otherwise, it indicates that this Text Request ends a set of key=value pairs. A Text Request with the C bit set to 1 MUST have the F bit set to 0.

10.10.3. Initiator Task Tag

The initiator assigned identifier for this Text Request. If the command is sent as part of a sequence of text requests and responses, the Initiator Task Tag MUST be the same for all the requests within the sequence (similar to linked SCSI commands). The I bit for all requests in a sequence also MUST be the same.

10.10.4. Target Transfer Tag

When the Target Transfer Tag is set to the reserved value 0xffffffff, it tells the target that this is a new request and the target resets any internal state associated with the Initiator Task Tag (resets the current negotiation state).

The target sets the Target Transfer Tag in a text response to a value other than the reserved value 0xffffffff whenever it indicates that it has more data to send or more operations to perform that are associated with the specified Initiator Task Tag. It MUST do so whenever it sets the F bit to 0 in the response. By copying the Target Transfer Tag from the response to the next Text Request, the initiator tells the target to continue the operation for the specific Initiator Task Tag. The initiator MUST ignore the Target Transfer Tag in the Text Response when the F bit is set to 1.

This mechanism allows the initiator and target to transfer a large amount of textual data over a sequence of text-command/text-response exchanges, or to perform extended negotiation sequences.

If the Target Transfer Tag is not 0xffffffff, the LUN field MUST be sent by the target in the Text Response.

Docker EX1014
Page 150 of 257

A target MAY reset its internal negotiation state if an exchange is stalled by the initiator for a long time or if it is running out of resources.

Long text responses are handled as in the following example:

```
I->T Text SendTargets=All (F=1,TTT=0xffffffff)
T->I Text <part 1> (F=0,TTT=0x12345678)
I->T Text <empty> (F=1, TTT=0x12345678)
T->I Text <part 2> (F=0, TTT=0x12345678)
I->T Text <empty> (F=1, TTT=0x12345678)
...
T->I Text <part n> (F=1, TTT=0xffffffff)
```

10.10.5. Text

The data lengths of a text request MUST NOT exceed the iSCSI target MaxRecvDataSegmentLength (a per connection and per direction negotiated parameter). The text format is specified in Section 5.2 Text Mode Negotiation.

Chapter 11 and Chapter 12 list some basic Text key=value pairs, some of which can be used in Login Request/Response and some in Text Request/Response.

A key=value pair can span Text request or response boundaries. A key=value pair can start in one PDU and continue on the next. In other words the end of a PDU does not necessarily signal the end of a key=value pair.

The target responds by sending its response back to the initiator. The response text format is similar to the request text format. The text response MAY refer to key=value pairs presented in an earlier text request and the text in the request may refer to earlier responses.

Chapter 5 details the rules for the Text Requests and Responses.

Text operations are usually meant for parameter setting/ negotiations, but can also be used to perform some long lasting operations.

Text operations that take a long time should be placed in their own Text request.

Docker EX1014
Page 151 of 257

10.11. Text Response

The Text Response PDU contains the target's responses to the initiator's Text request. The format of the Text field matches that of the Text request.

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x24      |F|C| Reserved                                  |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN or Reserved                                               |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36/ Reserved                                                     /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
  / DataSegment (Text)                                           /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
  | Data-Digest (Optional)                                        |
  +---------------+---------------+---------------+---------------+
```

10.11.1. F (Final) Bit

When set to 1, in response to a Text Request with the Final bit set to 1, the F bit indicates that the target has finished the whole operation. Otherwise, if set to 0 in response to a Text Request with the Final Bit set to 1, it indicates that the target has more work to do (invites a follow-on text request). A Text Response with the F bit set to 1 in response to a Text Request with the F bit set to 0 is a protocol error.

A Text Response with the F bit set to 1 MUST NOT contain key=value pairs that may require additional answers from the initiator.

A Text Response with the F bit set to 1 MUST have a Target Transfer Tag field set to the reserved value of 0xffffffff.

A Text Response with the F bit set to 0 MUST have a Target Transfer Tag field set to a value other than the reserved 0xffffffff.

10.11.2. C (Continue) Bit

When set to 1, indicates that the text (set of key=value pairs) in this Text Response is not complete (it will be continued on subsequent Text Responses); otherwise, it indicates that this Text Response ends a set of key=value pairs. A Text Response with the C bit set to 1 MUST have the F bit set to 0.

10.11.3. Initiator Task Tag

The Initiator Task Tag matches the tag used in the initial Text Request.

10.11.4. Target Transfer Tag

When a target has more work to do (e.g., cannot transfer all the remaining text data in a single Text Response or has to continue the negotiation) and has enough resources to proceed, it MUST set the Target Transfer Tag to a value other than the reserved value of 0xffffffff. Otherwise, the Target Transfer Tag MUST be set to 0xffffffff.

When the Target Transfer Tag is not 0xffffffff, the LUN field may be significant.

The initiator MUST copy the Target Transfer Tag and LUN in its next request to indicate that it wants the rest of the data.

When the target receives a Text Request with the Target Transfer Tag set to the reserved value of 0xffffffff, it resets its internal information (resets state) associated with the given Initiator Task Tag (restarts the negotiation).

When a target cannot finish the operation in a single Text Response, and does not have enough resources to continue, it rejects the Text Request with the appropriate Reject code.

Docker EX1014
Page 153 of 257

A target may reset its internal state associated with an Initiator Task Tag (the current negotiation state), state expressed through the Target Transfer Tag if the initiator fails to continue the exchange for some time. The target may reject subsequent Text Requests with the Target Transfer Tag set to the "stale" value.

10.11.5. StatSN

The target StatSN variable is advanced by each Text Response sent.

10.11.6. Text Response Data

The data lengths of a text response MUST NOT exceed the iSCSI initiator MaxRecvDataSegmentLength (a per connection and per direction negotiated parameter).

The text in the Text Response Data is governed by the same rules as the text in the Text Request Data (see Section 10.10.5 Text).

Although the initiator is the requesting party and controls the request-response initiation and termination, the target can offer key=value pairs of its own as part of a sequence and not only in response to the initiator.

10.12. Login Request

After establishing a TCP connection between an initiator and a target, the initiator MUST start a Login Phase to gain further access to the target’s resources.

The Login Phase (see Chapter 5) consists of a sequence of Login Requests and Responses that carry the same Initiator Task Tag.

Login Requests are always considered as immediate.

Docker EX1014
Page 154 of 257

```
   Byte/     0       |       1       |       2       |       3       |
      /              |               |               |               |
     |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
     +---------------+---------------+---------------+---------------+
    0|.|1| 0x03      |T|C|.|.|CSG|NSG| Version-max   | Version-min   |
     +---------------+---------------+---------------+---------------+
    4|TotalAHSLength | DataSegmentLength                             |
     +---------------+---------------+---------------+---------------+
    8| ISID                                                          |
     +                               +---------------+---------------+
   12|                               | TSIH                          |
     +---------------+---------------+---------------+---------------+
   16| Initiator Task Tag                                            |
     +---------------+---------------+---------------+---------------+
   20| CID                           | Reserved                      |
     +---------------+---------------+---------------+---------------+
   24| CmdSN                                                         |
     +---------------+---------------+---------------+---------------+
   28| ExpStatSN   or   Reserved                                     |
     +---------------+---------------+---------------+---------------+
   32| Reserved                                                      |
     +---------------+---------------+---------------+---------------+
   36| Reserved                                                      |
     +---------------+---------------+---------------+---------------+
   40/ Reserved                                                      /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
   48/ DataSegment - Login Parameters in Text request Format         /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
```

10.12.1. T (Transit) Bit

If set to 1, indicates that the initiator is ready to transit to the next stage.

If the T bit is set to 1 and NSG is FullFeaturePhase, then this also indicates that the initiator is ready for the Final Login Response (see Chapter 5).

10.12.2. C (Continue) Bit

When set to 1, indicates that the text (set of key=value pairs) in this Login Request is not complete (it will be continued on subsequent Login Requests); otherwise, it indicates that this Login Request ends a set of key=value pairs. A Login Request with the C bit set to 1 MUST have the T bit set to 0.

10.12.3. CSG and NSG

Through these fields, Current Stage (CSG) and Next Stage (NSG), the Login negotiation requests and responses are associated with a specific stage in the session (SecurityNegotiation, LoginOperationalNegotiation, FullFeaturePhase) and may indicate the next stage to which they want to move (see Chapter 5). The next stage value is only valid when the T bit is 1; otherwise, it is reserved.

The stage codes are:

- 0 - SecurityNegotiation
- 1 - LoginOperationalNegotiation
- 3 - FullFeaturePhase

All other codes are reserved.

10.12.4. Version

The version number of the current draft is 0x00. As such, all devices MUST carry version 0x00 for both Version-min and Version-max.

10.12.4.1. Version-max

Maximum Version number supported.

All Login Requests within the Login Phase MUST carry the same Version-max.

The target MUST use the value presented with the first Login Request.

10.12.4.2. Version-min

All Login Requests within the Login Phase MUST carry the same Version-min. The target MUST use the value presented with the first Login Request.

10.12.5. ISID

This is an initiator-defined component of the session identifier and is structured as follows (see [RFC3721] and Section 9.1.1 Conservative Reuse of ISIDs for details):

```
 Byte/     0       |       1       |       2       |       3       |
    /              |               |               |               |
   |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
   +---------------+---------------+---------------+---------------+
  8| T |    A      |               B               |      C        |
   +---------------+---------------+---------------+---------------+
 12|               D               |
   +---------------+---------------+
```

The T field identifies the format and usage of A, B, C, and D as indicated below:

```
  T

  00b     OUI-Format
          A&B are a 22 bit OUI
          (the I/G & U/L bits are omitted)
          C&D 24 bit qualifier
  01b     EN - Format (IANA Enterprise Number)
          A - Reserved
          B&C EN (IANA Enterprise Number)
          D - Qualifier
  10b     "Random"
          A - Reserved
          B&C Random
          D - Qualifier
  11b     A,B,C&D Reserved
```

For the T field values 00b and 01b, a combination of A and B (for 00b) or B and C (for 01b) identifies the vendor or organization whose component (software or hardware) generates this ISID. A vendor or organization with one or more OUIs, or one or more Enterprise Numbers, MUST use at least one of these numbers and select the appropriate value for the T field when its components generate ISIDs. An OUI or EN MUST be set in the corresponding fields in network byte order (byte big-endian).

If the T field is 10b, B and C are set to a random 24-bit unsigned integer value in network byte order (byte big-endian). See [RFC3721] for how this affects the principle of "conservative reuse".

The Qualifier field is a 16 or 24-bit unsigned integer value that provides a range of possible values for the ISID within the selected namespace. It may be set to any value within the constraints specified in the iSCSI protocol (see Section 3.4.3 Consequences of the Model and Section 9.1.1 Conservative Reuse of ISIDs).

The T field value of 11b is reserved.

If the ISID is derived from something assigned to a hardware adapter or interface by a vendor, as a preset default value, it MUST be configurable to a value assigned according to the SCSI port behavior desired by the system in which it is installed (see Section 9.1.1 Conservative Reuse of ISIDs and Section 9.1.2 iSCSI Name, ISID, and TPGT Use). The resultant ISID MUST also be persistent over power cycles, reboot, card swap, etc.

10.12.6. TSIH

TSIH must be set in the first Login Request. The reserved value 0 MUST be used on the first connection for a new session. Otherwise, the TSIH sent by the target at the conclusion of the successful login of the first connection for this session MUST be used. The TSIH identifies to the target the associated existing session for this new connection.

All Login Requests within a Login Phase MUST carry the same TSIH.

The target MUST check the value presented with the first Login Request and act as specified in Section 5.3.1 Login Phase Start.

10.12.7. Connection ID - CID

A unique ID for this connection within the session.

All Login Requests within the Login Phase MUST carry the same CID.

The target MUST use the value presented with the first Login Request.

A Login Request with a non-zero TSIH and a CID equal to that of an existing connection implies a logout of the connection followed by a Login (see Section 5.3.4 Connection Reinstatement). For the details of the implicit Logout Request, see Section 10.14 Logout Request.

10.12.8. CmdSN

CmdSN is either the initial command sequence number of a session (for the first Login Request of a session - the "leading" login), or the command sequence number in the command stream if the login is for a new connection in an existing session.

Examples:

- Login on a leading connection - if the leading login carries the CmdSN 123, all other Login Requests in the same Login Phase carry the CmdSN 123 and the first non-immediate command in FullFeaturePhase also carries the CmdSN 123.

- Login on other than a leading connection - if the current CmdSN at the time the first login on the connection is issued is 500, then that PDU carries CmdSN=500. Subsequent Login Requests that are needed to complete this Login Phase may carry a CmdSN higher than 500 if non-immediate requests that were issued on other connections in the same session advance CmdSN.

If the Login Request is a leading Login Request, the target MUST use the value presented in CmdSN as the target value for ExpCmdSN.

10.12.9. ExpStatSN

For the first Login Request on a connection this is ExpStatSN for the old connection and this field is only valid if the Login Request restarts a connection (see Section 5.3.4 Connection Reinstatement).

For subsequent Login Requests it is used to acknowledge the Login Responses with their increasing StatSN values.

10.12.10. Login Parameters

The initiator MUST provide some basic parameters in order to enable the target to determine if the initiator may use the target’s resources and the initial text parameters for the security exchange.

All the rules specified in Section 10.10.5 Text for text requests also hold for Login Requests. Keys and their explanations are listed in Chapter 11 (security negotiation keys) and Chapter 12 (operational parameter negotiation keys). All keys in Chapter 12, except for the X extension formats, MUST be supported by iSCSI initiators and targets. Keys in Chapter 11 only need to be supported when the function to which they refer is mandatory to implement.

Docker EX1014
Page 159 of 257

10.13. Login Response

The Login Response indicates the progress and/or end of the Login Phase.

```
Byte/     0       |       1       |       2       |       3       |
   /                |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x23      |T|C|.|.|CSG|NSG| Version-max   | Version-active|
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| ISID                                                          |
  +                               +---------------+---------------+
12|                               | TSIH                          |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| Status-Class  | Status-Detail | Reserved                      |
  +---------------+---------------+---------------+---------------+
40/ Reserved                                                     /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
48/ DataSegment - Login Parameters in Text request Format        /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
```

10.13.1. Version-max

This is the highest version number supported by the target.

All Login Responses within the Login Phase MUST carry the same Version-max.

The initiator MUST use the value presented as a response to the first Login Request.

10.13.2. Version-active

Indicates the highest version supported by the target and initiator. If the target does not support a version within the range specified by the initiator, the target rejects the login and this field indicates the lowest version supported by the target.

All Login Responses within the Login Phase MUST carry the same Version-active.

The initiator MUST use the value presented as a response to the first Login Request.

10.13.3. TSIH

The TSIH is the target assigned session identifying handle. Its internal format and content are not defined by this protocol except for the value 0 that is reserved. With the exception of the Login Final-Response in a new session, this field should be set to the TSIH provided by the initiator in the Login Request. For a new session, the target MUST generate a non-zero TSIH and ONLY return it in the Login Final-Response (see Section 5.3 Login Phase).

10.13.4. StatSN

For the first Login Response (the response to the first Login Request), this is the starting status Sequence Number for the connection. The next response of any kind, including the next Login Response, if any, in the same Login Phase, will carry this number + 1. This field is only valid if the Status-Class is 0.

10.13.5. Status-Class and Status-Detail

The Status returned in a Login Response indicates the execution status of the Login Phase. The status includes:

```
  Status-Class
  Status-Detail
```

0 Status-Class indicates success.

A non-zero Status-Class indicates an exception. In this case, Status-Class is sufficient for a simple initiator to use when handling exceptions, without having to look at the Status-Detail. The Status-Detail allows finer-grained exception handling for more sophisticated initiators and for better information for logging.

Docker EX1014
Page 161 of 257

The status classes are as follows:

0 - Success - indicates that the iSCSI target successfully received, understood, and accepted the request. The numbering fields (StatSN, ExpCmdSN, MaxCmdSN) are only valid if Status-Class is 0.

1 - Redirection - indicates that the initiator must take further action to complete the request. This is usually due to the target moving to a different address. All of the redirection status class responses MUST return one or more text key parameters of the type "TargetAddress", which indicates the target's new address. A redirection response MAY be issued by a target prior or after completing a security negotiation if a security negotiation is required. A redirection SHOULD be accepted by an initiator even without having the target complete a security negotiation if any security negotiation is required, and MUST be accepted by the initiator after the completion of the security negotiation if any security negotiation is required.

2 - Initiator Error (not a format error) - indicates that the initiator most likely caused the error. This MAY be due to a request for a resource for which the initiator does not have permission. The request should not be tried again.

3 - Target Error - indicates that the target sees no errors in the initiator's Login Request, but is currently incapable of fulfilling the request. The initiator may re-try the same Login Request later.

The table below shows all of the currently allocated status codes. The codes are in hexadecimal; the first byte is the status class and the second byte is the status detail.

Status	Code (hex)	Description
Success	0000	Login is proceeding OK (*1).
Target moved temporarily	0101	The requested iSCSI Target Name (ITN) has temporarily moved to the address provided.
Target moved permanently	0102	The requested ITN has permanently moved to the address provided.

Initiator error	0200	Miscellaneous iSCSI initiator errors.
Authentication failure	0201	The initiator could not be successfully authenticated or target authentication is not supported.
Authorization failure	0202	The initiator is not allowed access to the given target.
Not found	0203	The requested ITN does not exist at this address.
Target removed	0204	The requested ITN has been removed and no forwarding address is provided.
Unsupported version	0205	The requested iSCSI version range is not supported by the target.
Too many connections	0206	Too many connections on this SSID.
Missing parameter	0207	Missing parameters (e.g., iSCSI Initiator and/or Target Name).
Can’t include in session	0208	Target does not support session spanning to this connection (address).
Session type not supported	0209	Target does not support this type of of session or not from this Initiator.
Session does not exist	020a	Attempt to add a connection to a non-existent session.
Invalid during login	020b	Invalid Request type during Login.
Target error	0300	Target hardware or software error.
Service unavailable	0301	The iSCSI service or target is not currently operational.
Out of resources	0302	The target has insufficient session, connection, or other resources.

(*1) If the response T bit is 1 in both the request and the matching response, and the NSG is FullFeaturePhase in both the request and the matching response, the Login Phase is finished and the initiator may proceed to issue SCSI commands.

If the Status Class is not 0, the initiator and target MUST close the TCP connection.

If the target wishes to reject the Login Request for more than one reason, it should return the primary reason for the rejection.

10.13.6. T (Transit) bit

The T bit is set to 1 as an indicator of the end of the stage. If the T bit is set to 1 and NSG is FullFeaturePhase, then this is also the Final Login Response (see Chapter 5). A T bit of 0 indicates a "partial" response, which means "more negotiation needed".

A Login Response with a T bit set to 1 MUST NOT contain key=value pairs that may require additional answers from the initiator within the same stage.

If the status class is 0, the T bit MUST NOT be set to 1 if the T bit in the request was set to 0.

10.13.7. C (Continue) Bit

When set to 1, indicates that the text (set of key=value pairs) in this Login Response is not complete (it will be continued on subsequent Login Responses); otherwise, it indicates that this Login Response ends a set of key=value pairs. A Login Response with the C bit set to 1 MUST have the T bit set to 0.

10.13.8. Login Parameters

The target MUST provide some basic parameters in order to enable the initiator to determine if it is connected to the correct port and the initial text parameters for the security exchange.

All the rules specified in Section 10.11.6 Text Response Data for text responses also hold for Login Responses. Keys and their explanations are listed in Chapter 11 (security negotiation keys) and Chapter 12 (operational parameter negotiation keys). All keys in Chapter 12, except for the X extension formats, MUST be supported by iSCSI initiators and targets. Keys in Chapter 11, only need to be supported when the function to which they refer is mandatory to implement.

10.14. Logout Request

The Logout Request is used to perform a controlled closing of a connection.

An initiator MAY use a Logout Request to remove a connection from a session or to close an entire session.

After sending the Logout Request PDU, an initiator MUST NOT send any new iSCSI requests on the closing connection. If the Logout Request is intended to close the session, new iSCSI requests MUST NOT be sent on any of the connections participating in the session.

When receiving a Logout Request with the reason code of "close the connection" or "close the session", the target MUST terminate all pending commands, whether acknowledged via ExpCmdSN or not, on that connection or session respectively.

When receiving a Logout Request with the reason code "remove connection for recovery", the target MUST discard all requests not yet acknowledged via ExpCmdSN that were issued on the specified connection, and suspend all data/status/R2T transfers on behalf of pending commands on the specified connection.

The target then issues the Logout Response and half-closes the TCP connection (sends FIN). After receiving the Logout Response and attempting to receive the FIN (if still possible), the initiator MUST completely close the logging-out connection. For the terminated commands, no additional responses should be expected.

A Logout for a CID may be performed on a different transport connection when the TCP connection for the CID has already been terminated. In such a case, only a logical "closing" of the iSCSI connection for the CID is implied with a Logout.

All commands that were not terminated or not completed (with status) and acknowledged when the connection is closed completely can be reassigned to a new connection if the target supports connection recovery.

If an initiator intends to start recovery for a failing connection, it MUST use the Logout Request to "clean-up" the target end of a failing connection and enable recovery to start, or the Login Request with a non-zero TSIH and the same CID on a new connection for the same effect (see Section 10.14.3 CID). In sessions with a single connection, the connection can be closed and then a new connection reopened. A connection reinstatement login can be used for recovery (see Section 5.3.4 Connection Reinstatement).

Docker EX1014
Page 165 of 257

A successful completion of a Logout Request with the reason code of "close the connection" or "remove the connection for recovery" results at the target in the discarding of unacknowledged commands received on the connection being logged out. These are commands that have arrived on the connection being logged out, but have not been delivered to SCSI because one or more commands with a smaller CmdSN has not been received by iSCSI. See Section 3.2.2.1 Command Numbering and Acknowledging. The resulting holes the in command sequence numbers will have to be handled by appropriate recovery (see Chapter 6) unless the session is also closed.

The entire logout discussion in this section is also applicable for an implicit Logout realized via a connection reinstatement or session reinstatement. When a Login Request performs an implicit Logout, the implicit Logout is performed as if having the reason codes specified below:

Reason code	Type of implicit Logout
0	session reinstatement
1	connection reinstatement when the operational ErrorRecoveryLevel < 2
2	connection reinstatement when the operational ErrorRecoveryLevel = 2

```
   Byte/     0       |       1       |       2       |       3       |
      /              |               |               |               |
     |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
     +---------------+---------------+---------------+---------------+
    0|.|I| 0x06      |1| Reason Code | Reserved                    |
     +---------------+---------------+---------------+---------------+
    4|TotalAHSLength | DataSegmentLength                             |
     +---------------------------------------------------------------+
    8/ Reserved                                                     /
    +/                                                              /
     +---------------+---------------+---------------+---------------+
   16| Initiator Task Tag                                            |
     +---------------+---------------+---------------+---------------+
   20| CID or Reserved               | Reserved                      |
     +---------------+---------------+---------------+---------------+
   24| CmdSN                                                         |
     +---------------+---------------+---------------+---------------+
   28| ExpStatSN                                                     |
     +---------------+---------------+---------------+---------------+
   32/ Reserved                                                     /
    +/                                                              /
     +---------------+---------------+---------------+---------------+
   48| Header-Digest (Optional)                                      |
     +---------------+---------------+---------------+---------------+
```

10.14.1. Reason Code

Reason Code indicates the reason for Logout as follows:

0 - close the session. All commands associated with the session (if any) are terminated.

1 - close the connection. All commands associated with connection (if any) are terminated.

2 - remove the connection for recovery. Connection is closed and all commands associated with it, if any, are to be prepared for a new allegiance.

All other values are reserved.

10.14.2. TotalAHSLength and DataSegmentLength

For this PDU TotalAHSLength and DataSegmentLength MUST be 0.

10.14.3. CID

This is the connection ID of the connection to be closed (including closing the TCP stream). This field is only valid if the reason code is not "close the session".

10.14.4. ExpStatSN

This is the last ExpStatSN value for the connection to be closed.

10.14.5. Implicit termination of tasks

A target implicitly terminates the active tasks due to the iSCSI protocol in the following cases:

a) When a connection is implicitly or explicitly logged out with the reason code of "Close the connection" and there are active tasks allegiant to that connection.

b) When a connection fails and eventually the connection state times out (state transition M1 in Section 7.2.2 State Transition Descriptions for Initiators and Targets) and there are active tasks allegiant to that connection.

c) When a successful recovery Logout is performed while there are active tasks allegiant to that connection, and those tasks eventually time out after the Time2Wait and Time2Retain periods without allegiance reassignment.

d) When a connection is implicitly or explicitly logged out with the reason code of "Close the session" and there are active tasks in that session.

If the tasks terminated in any of the above cases are SCSI tasks, they must be internally terminated as if with CHECK CONDITION status. This status is only meaningful for appropriately handling the internal SCSI state and SCSI side effects with respect to ordering because this status is never communicated back as a terminating status to the initiator. However additional actions may have to be taken at SCSI level depending on the SCSI context as defined by the SCSI standards (e.g., queued commands and ACA, in cases a), b), and c), after the tasks are terminated, the target MUST report a Unit Attention condition on the next command processed on any connection for each affected I_T_L nexus with the status of CHECK CONDITION, and

Docker EX1014
Page 168 of 257

the ASC/ASCQ value of 47h/7Fh - "SOME COMMANDS CLEARED BY ISCSI PROTOCOL EVENT" - etc. - see [SAM2] and [SPC3]).

10.15. Logout Response

The Logout Response is used by the target to indicate if the cleanup operation for the connection(s) has completed.

After Logout, the TCP connection referred by the CID MUST be closed at both ends (or all connections must be closed if the logout reason was session close).

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x26      |1| Reserved    | Response      | Reserved      |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------------------------------------------------------+
 8/ Reserved                                                     /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| Reserved                                                      |
  +---------------------------------------------------------------+
40| Time2Wait                     | Time2Retain                   |
  +---------------+---------------+---------------+---------------+
44| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
```

10.15.1. Response

Logout Response:

0 - connection or session closed successfully.

1 - CID not found.

2 - connection recovery is not supported. If Logout reason code was recovery and target does not support it as indicated by the ErrorRecoveryLevel.

3 - cleanup failed for various reasons.

10.15.2. TotalAHSLength and DataSegmentLength

For this PDU TotalAHSLength and DataSegmentLength MUST be 0.

10.15.3. Time2Wait

If the Logout Response code is 0 and if the operational ErrorRecoveryLevel is 2, this is the minimum amount of time, in seconds, to wait before attempting task reassignment. If the Logout Response code is 0 and if the operational ErrorRecoveryLevel is less than 2, this field is to be ignored.

This field is invalid if the Logout Response code is 1.

If the Logout response code is 2 or 3, this field specifies the minimum time to wait before attempting a new implicit or explicit logout.

If Time2Wait is 0, the reassignment or a new Logout may be attempted immediately.

10.15.4. Time2Retain

If the Logout response code is 0 and if the operational ErrorRecoveryLevel is 2, this is the maximum amount of time, in seconds, after the initial wait (Time2Wait), the target waits for the allegiance reassignment for any active task after which the task state is discarded. If the Logout response code is 0 and if the operational ErrorRecoveryLevel is less than 2, this field is to be ignored.

This field is invalid if the Logout response code is 1.

Docker EX1014
Page 170 of 257

If the Logout response code is 2 or 3, this field specifies the maximum amount of time, in seconds, after the initial wait (Time2Wait), the target waits for a new implicit or explicit logout.

If it is the last connection of a session, the whole session state is discarded after Time2Retain.

If Time2Retain is 0, the target has already discarded the connection (and possibly the session) state along with the task states. No reassignment or Logout is required in this case.

10.16. SNACK Request

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x10      |1|.|.|.| Type  | Reserved                      |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN or Reserved                                               |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag or 0xffffffff                              |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or SNACK Tag or 0xffffffff                |
  +---------------+---------------+---------------+---------------+
24| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
28| ExpStatSN                                                     |
  +---------------+---------------+---------------+---------------+
32/ Reserved                                                     /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
40| BegRun                                                        |
  +---------------------------------------------------------------+
44| RunLength                                                     |
  +---------------------------------------------------------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
```

If the implementation supports ErrorRecoveryLevel greater than zero, it MUST support all SNACK types.

The SNACK is used by the initiator to request the retransmission of numbered-responses, data, or R2T PDUs from the target. The SNACK request indicates the numbered-responses or data "runs" whose retransmission is requested by the target, where the run starts with the first StatSN, DataSN, or R2TSN whose retransmission is requested and indicates the number of Status, Data, or R2T PDUs requested including the first. 0 has special meaning when used as a starting number and length:

- When used in RunLength, it means all PDUs starting with the initial.
- When used in both BegRun and RunLength, it means all unacknowledged PDUs.

The numbered-response(s) or R2T(s), requested by a SNACK, MUST be delivered as exact replicas of the ones that the target transmitted originally except for the fields ExpCmdSN, MaxCmdSN, and ExpDataSN, which MUST carry the current values. R2T(s)requested by SNACK MUST also carry the current value of StatSN.

The numbered Data-In PDUs, requested by a Data SNACK MUST be delivered as exact replicas of the ones that the target transmitted originally except for the fields ExpCmdSN and MaxCmdSN, which MUST carry the current values and except for resegmentation (see Section 10.16.3 Resegmentation).

Any SNACK that requests a numbered-response, Data, or R2T that was not sent by the target or was already acknowledged by the initiator, MUST be rejected with a reason code of "Protocol error".

10.16.1. Type

This field encodes the SNACK function as follows:

0-Data/R2T SNACK - requesting retransmission of one or more Data-In or R2T PDUs.

1-Status SNACK - requesting retransmission of one or more numbered responses.

2-DataACK - positively acknowledges Data-In PDUs.

3-R-Data SNACK - requesting retransmission of Data-In PDUs with possible resegmentation and status tagging.

Docker EX1014
Page 172 of 257

All other values are reserved.

Data/R2T SNACK, Status SNACK, or R-Data SNACK for a command MUST precede status acknowledgement for the given command.

10.16.2. Data Acknowledgement

If an initiator operates at ErrorRecoveryLevel 1 or higher, it MUST issue a SNACK of type DataACK after receiving a Data-In PDU with the A bit set to 1. However, if the initiator has detected holes in the input sequence, it MUST postpone issuing the SNACK of type DataACK until the holes are filled. An initiator MAY ignore the A bit if it deems that the bit is being set aggressively by the target (i.e., before the MaxBurstLength limit is reached).

The DataACK is used to free resources at the target and not to request or imply data retransmission.

An initiator MUST NOT request retransmission for any data it had already acknowledged.

10.16.3. Resegmentation

If the initiator MaxRecvDataSegmentLength changed between the original transmission and the time the initiator requests retransmission, the initiator MUST issue a R-Data SNACK (see Section 10.16.1 Type). With R-Data SNACK, the initiator indicates that it discards all the unacknowledged data and expects the target to resend it. It also expects resegmentation. In this case, the retransmitted Data-In PDUs MAY be different from the ones originally sent in order to reflect changes in MaxRecvDataSegmentLength. Their DataSN starts with the BegRun of the last DataACK received by the target if any was received; otherwise it starts with 0 and is increased by 1 for each resent Data-In PDU.

A target that has received a R-Data SNACK MUST return a SCSI Response that contains a copy of the SNACK Tag field from the R-Data SNACK in the SCSI Response SNACK Tag field as its last or only Response. For example, if it has already sent a response containing another value in the SNACK Tag field or had the status included in the last Data-In PDU, it must send a new SCSI Response PDU. If a target sends more than one SCSI Response PDU due to this rule, all SCSI responses must carry the same StatSN (see Section 10.4.4 SNACK Tag). If an initiator attempts to recover a lost SCSI Response (with a Status SNACK, see Section 10.16.1 Type) when more than one response has been sent, the target will send the SCSI Response with the latest content known to the target, including the last SNACK Tag for the command.

For considerations in allegiance reassignment of a task to a connection with a different MaxRecvDataSegmentLength, refer to Section 6.2.2 Allegiance Reassignment.

10.16.4. Initiator Task Tag

For Status SNACK and DataACK, the Initiator Task Tag MUST be set to the reserved value 0xffffffff. In all other cases, the Initiator Task Tag field MUST be set to the Initiator Task Tag of the referenced command.

10.16.5. Target Transfer Tag or SNACK Tag

For an R-Data SNACK, this field MUST contain a value that is different from 0 or 0xffffffff and is unique for the task (identified by the Initiator Task Tag). This value MUST be copied by the iSCSI target in the last or only SCSI Response PDU it issues for the command.

For DataACK, the Target Transfer Tag MUST contain a copy of the Target Transfer Tag and LUN provided with the SCSI Data-In PDU with the A bit set to 1.

In all other cases, the Target Transfer Tag field MUST be set to the reserved value of 0xffffffff.

10.16.6. BegRun

The DataSN, R2TSN, or StatSN of the first PDU whose retransmission is requested (Data/R2T and Status SNACK), or the next expected DataSN (DataACK SNACK).

BegRun 0 when used in conjunction with RunLength 0 means resend all unacknowledged Data-In, R2T or Response PDUs.

BegRun MUST be 0 for a R-Data SNACK.

10.16.7. RunLength

The number of PDUs whose retransmission is requested.

RunLength 0 signals that all Data-In, R2T, or Response PDUs carrying the numbers equal to or greater than BegRun have to be resent.

The RunLength MUST also be 0 for a DataACK SNACK in addition to R-Data SNACK.

Docker EX1014
Page 174 of 257

10.17. Reject

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x3f      |1| Reserved    | Reason        | Reserved      |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8/ Reserved                                                     /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
16| 0xffffffff                                                    |
  +---------------+---------------+---------------+---------------+
20| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| DataSN/R2TSN or Reserved                                      |
  +---------------+---------------+---------------+---------------+
40| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
44| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
xx/ Complete Header of Bad PDU                                    /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
yy/Vendor specific data (if any)                                  /
  /                                                               /
  +---------------+---------------+---------------+---------------+
zz| Data-Digest (Optional)                                        |
  +---------------+---------------+---------------+---------------+
```

Reject is used to indicate an iSCSI error condition (protocol, unsupported option, etc.).

10.17.1. Reason

The reject Reason is coded as follows:

Code (hex)	Explanation	Can the original PDU be re-sent?
0x01	Reserved	no
0x02	Data (payload) Digest Error	yes (Note 1)
0x03	SNACK Reject	yes
0x04	Protocol Error (e.g., SNACK request for a status that was already acknowledged)	no
0x05	Command not supported	no
0x06	Immediate Command Reject - too many immediate commands	yes
0x07	Task in progress	no
0x08	Invalid Data ACK	no
0x09	Invalid PDU field	no (Note 2)
0x0a	Long Operation Reject - Can’t generate Target Transfer Tag - out of resources	yes
0x0b	Negotiation Reset	no
0x0c	Waiting for Logout	no

Note 1: For iSCSI, Data-Out PDU retransmission is only done if the target requests retransmission with a recovery R2T. However, if this is the data digest error on immediate data, the initiator may choose to retransmit the whole PDU including the immediate data.

Note 2: A target should use this reason code for all invalid values of PDU fields that are meant to describe a task, a response, or a data transfer. Some examples are invalid TTT/ITT, buffer offset, LUN qualifying a TTT, and an invalid sequence number in a SNACK.

All other values for Reason are reserved.

Docker EX1014
Page 176 of 257

In all the cases in which a pre-instantiated SCSI task is terminated because of the reject, the target MUST issue a proper SCSI command response with CHECK CONDITION as described in Section 10.4.3 Response. In these cases in which a status for the SCSI task was already sent before the reject, no additional status is required. If the error is detected while data from the initiator is still expected (i.e., the command PDU did not contain all the data and the target has not received a Data-Out PDU with the Final bit set to 1 for the unsolicited data, if any, and all outstanding R2Ts, if any), the target MUST wait until it receives the last expected Data-Out PDUs with the F bit set to 1 before sending the Response PDU.

For additional usage semantics of Reject PDU, see Section 6.3 Usage Of Reject PDU in Recovery.

10.17.2. DataSN/R2TSN

This field is only valid if the rejected PDU is a Data/R2T SNACK and the Reject reason code is "Protocol error" (see Section 10.16 SNACK Request). The DataSN/R2TSN is the next Data/R2T sequence number that the target would send for the task, if any.

10.17.3. StatSN, ExpCmdSN and MaxCmdSN

These fields carry their usual values and are not related to the rejected command. StatSN is advanced after a Reject.

10.17.4. Complete Header of Bad PDU

The target returns the header (not including digest) of the PDU in error as the data of the response.

Docker EX1014
Page 177 of 257

10.18. NOP-Out

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|I| 0x00      |1| Reserved                                    |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN or Reserved                                               |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag or 0xffffffff                              |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| CmdSN                                                         |
  +---------------+---------------+---------------+---------------+
28| ExpStatSN                                                     |
  +---------------+---------------+---------------+---------------+
32/ Reserved                                                     /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
  / DataSegment - Ping Data (optional)                            /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
  | Data-Digest (Optional)                                        |
  +---------------+---------------+---------------+---------------+
```

A NOP-Out may be used by an initiator as a "ping request" to verify that a connection/session is still active and all its components are operational. The NOP-In response is the "ping echo".

A NOP-Out is also sent by an initiator in response to a NOP-In.

A NOP-Out may also be used to confirm a changed ExpStatSN if another PDU will not be available for a long time.

Upon receipt of a NOP-In with the Target Transfer Tag set to a valid value (not the reserved 0xffffffff), the initiator MUST respond with a NOP-Out. In this case, the NOP-Out Target Transfer Tag MUST contain a copy of the NOP-In Target Transfer Tag.

10.18.1. Initiator Task Tag

The NOP-Out MUST have the Initiator Task Tag set to a valid value only if a response in the form of NOP-In is requested (i.e., the NOP-Out is used as a ping request). Otherwise, the Initiator Task Tag MUST be set to 0xffffffff.

When a target receives the NOP-Out with a valid Initiator Task Tag, it MUST respond with a Nop-In Response (see Section 10.19 NOP-In).

If the Initiator Task Tag contains 0xffffffff, the I bit MUST be set to 1 and the CmdSN is not advanced after this PDU is sent.

10.18.2. Target Transfer Tag

A target assigned identifier for the operation.

The NOP-Out MUST only have the Target Transfer Tag set if it is issued in response to a NOP-In with a valid Target Transfer Tag. In this case, it copies the Target Transfer Tag from the NOP-In PDU. Otherwise, the Target Transfer Tag MUST be set to 0xffffffff.

When the Target Transfer Tag is set to a value other than 0xffffffff, the LUN field MUST also be copied from the NOP-In.

10.18.3. Ping Data

Ping data are reflected in the NOP-In Response. The length of the reflected data are limited to MaxRecvDataSegmentLength. The length of ping data are indicated by the DataSegmentLength. 0 is a valid value for the DataSegmentLength and indicates the absence of ping data.

Docker EX1014
Page 179 of 257

10.19. NOP-In

```
Byte/     0       |       1       |       2       |       3       |
   /              |               |               |               |
  |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
  +---------------+---------------+---------------+---------------+
 0|.|.| 0x20      |1| Reserved                                    |
  +---------------+---------------+---------------+---------------+
 4|TotalAHSLength | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN or Reserved                                               |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag or 0xffffffff                              |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36/ Reserved                                                     /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
48| Header-Digest (Optional)                                      |
  +---------------+---------------+---------------+---------------+
  / DataSegment - Return Ping Data                                /
 +/                                                              /
  +---------------+---------------+---------------+---------------+
  | Data-Digest (Optional)                                        |
  +---------------+---------------+---------------+---------------+
```

NOP-In is either sent by a target as a response to a NOP-Out, as a "ping" to an initiator, or as a means to carry a changed ExpCmdSN and/or MaxCmdSN if another PDU will not be available for a long time (as determined by the target).

When a target receives the NOP-Out with a valid Initiator Task Tag (not the reserved value 0xffffffff), it MUST respond with a NOP-In with the same Initiator Task Tag that was provided in the NOP-Out request. It MUST also duplicate up to the first MaxRecvDataSegmentLength bytes of the initiator provided Ping Data. For such a response, the Target Transfer Tag MUST be 0xffffffff.

Otherwise, when a target sends a NOP-In that is not a response to a Nop-Out received from the initiator, the Initiator Task Tag MUST be set to 0xffffffff and the Data Segment MUST NOT contain any data (DataSegmentLength MUST be 0).

10.19.1. Target Transfer Tag

If the target is responding to a NOP-Out, this is set to the reserved value 0xffffffff.

If the target is sending a NOP-In as a Ping (intending to receive a corresponding NOP-Out), this field is set to a valid value (not the reserved 0xffffffff).

If the target is initiating a NOP-In without wanting to receive a corresponding NOP-Out, this field MUST hold the reserved value of 0xffffffff.

10.19.2. StatSN

The StatSN field will always contain the next StatSN. However, when the Initiator Task Tag is set to 0xffffffff, StatSN for the connection is not advanced after this PDU is sent.

10.19.3. LUN

A LUN MUST be set to a correct value when the Target Transfer Tag is valid (not the reserved value 0xffffffff).

11. iSCSI Security Text Keys and Authentication Methods

Only the following keys are used during the SecurityNegotiation stage of the Login Phase:

- SessionType
- InitiatorName
- TargetName
- TargetAddress
- InitiatorAlias
- TargetAlias
- TargetPortalGroupTag
- AuthMethod and the keys used by the authentication methods specified under Section 11.1 AuthMethod along with all of their associated keys as well as Vendor Specific Authentication Methods.

Other keys MUST NOT be used.

SessionType, InitiatorName, TargetName, InitiatorAlias, TargetAlias, and TargetPortalGroupTag are described in Chapter 12 as they can be used also in the OperationalNegotiation stage.

All security keys have connection-wide applicability.

11.1. AuthMethod

Use: During Login - Security Negotiation Senders: Initiator and Target Scope: connection

AuthMethod = <list-of-values>

The main item of security negotiation is the authentication method (AuthMethod).

The authentication methods that can be used (appear in the list-of-values) are either those listed in the following table or are vendor-unique methods:

Name	Description
KRB5	Kerberos V5 - defined in [RFC1510]
SPKM1	Simple Public-Key GSS-API Mechanism defined in [RFC2025]
SPKM2	Simple Public-Key GSS-API Mechanism defined in [RFC2025]
SRP	Secure Remote Password defined in [RFC2945]
CHAP	Challenge Handshake Authentication Protocol defined in [RFC1994]
None	No authentication

The AuthMethod selection is followed by an "authentication exchange" specific to the authentication method selected.

The authentication method proposal may be made by either the initiator or the target. However the initiator MUST make the first step specific to the selected authentication method as soon as it is

Docker EX1014
Page 182 of 257

selected. It follows that if the target makes the authentication method proposal the initiator sends the first keys(s) of the exchange together with its authentication method selection.

The authentication exchange authenticates the initiator to the target, and optionally, the target to the initiator. Authentication is OPTIONAL to use but MUST be supported by the target and initiator.

The initiator and target MUST implement CHAP. All other authentication methods are OPTIONAL.

Private or public extension algorithms MAY also be negotiated for authentication methods. Whenever a private or public extension algorithm is part of the default offer (the offer made in absence of explicit administrative action) the implementer MUST ensure that CHAP is listed as an alternative in the default offer and "None" is not part of the default offer.

Extension authentication methods MUST be named using one of the following two formats:

a) Z-reversed.vendor.dns_name.do_something=
b) Z<#><IANA-registered-string>=

Authentication methods named using the Z- format are used as private extensions. Authentication methods named using the Z# format are used as public extensions that must be registered with IANA and MUST be described by an informational RFC.

For all of the public or private extension authentication methods, the method specific keys MUST conform to the format specified in Section 5.1 Text Format for standard-label.

To identify the vendor for private extension authentication methods, we suggest you use the reversed DNS-name as a prefix to the proper digest names.

The part of digest-name following Z- and Z# MUST conform to the format for standard-label specified in Section 5.1 Text Format.

Support for public or private extension authentication methods is OPTIONAL.

The following subsections define the specific exchanges for each of the standardized authentication methods. As mentioned earlier the first step is always done by the initiator.

Docker EX1014
Page 183 of 257

11.1.1. Kerberos

For KRB5 (Kerberos V5) [RFC1510] and [RFC1964], the initiator MUST use:

```
KRB_AP_REQ=<KRB_AP_REQ>
```

where KRB_AP_REQ is the client message as defined in [RFC1510].

The default principal name assumed by an iSCSI initiator or target (prior to any administrative configuration action) MUST be the iSCSI Initiator Name or iSCSI Target Name respectively, prefixed by the string "iscsi/".

If the initiator authentication fails, the target MUST respond with a Login reject with "Authentication Failure" status. Otherwise, if the initiator has selected the mutual authentication option (by setting MUTUAL-REQUIRED in the ap-options field of the KRB_AP_REQ), the target MUST reply with:

```
KRB_AP_REP=<KRB_AP_REP>
```

where KRB_AP_REP is the server’s response message as defined in [RFC1510].

If mutual authentication was selected and target authentication fails, the initiator MUST close the connection.

KRB_AP_REQ and KRB_AP_REP are binary-values and their binary length (not the length of the character string that represents them in encoded form) MUST not exceed 65536 bytes.

11.1.2. Simple Public-Key Mechanism (SPKM)

For SPKM1 and SPKM2 [RFC2025], the initiator MUST use:

```
SPKM_REQ=<SPKM-REQ>
```

where SPKM-REQ is the first initiator token as defined in [RFC2025].

[RFC2025] defines situations where each side may send an error token that may cause the peer to re-generate and resend its last token. This scheme is followed in iSCSI, and the error token syntax is:

```
SPKM_ERROR=<SPKM-ERROR>
```


Docker EX1014
Page 184 of 257

However, SPKM-DEL tokens that are defined by [RFC2025] for fatal errors will not be used by iSCSI. If the target needs to send a SPKM-DEL token, it will, instead, send a Login "login reject" message with the "Authentication Failure" status and terminate the connection. If the initiator needs to send a SPKM-DEL token, it will close the connection.

In the following sections, we assume that no SPKM-ERROR tokens are required.

If the initiator authentication fails, the target MUST return an error. Otherwise, if the AuthMethod is SPKM1 or if the initiator has selected the mutual authentication option (by setting mutual-state bit in the options field of the REQ-TOKEN in the SPKM-REQ), the target MUST reply with:

```
SPKM_REP_TI=<SPKM-REP-TI>
```

where SPKM-REP-TI is the target token as defined in [RFC2025].

If mutual authentication was selected and target authentication fails, the initiator MUST close the connection. Otherwise, if the AuthMethod is SPKM1, the initiator MUST continue with:

```
SPKM_REP_IT=<SPKM-REP-IT>
```

where SPKM-REP-IT is the second initiator token as defined in [RFC2025]. If the initiator authentication fails, the target MUST answer with a Login reject with "Authentication Failure" status.

SPKM requires support for very long authentication items.

All the SPKM-* tokens are binary-values and their binary length (not the length of the character string that represents them in encoded form) MUST not exceed 65536 bytes.

11.1.3. Secure Remote Password (SRP)

For SRP [RFC2945], the initiator MUST use:

```
SRP_U=<U> TargetAuth=Yes   /* or TargetAuth=No */
```

The target MUST answer with a Login reject with the "Authorization Failure" status or reply with:

```
SRP_GROUP=<G1,G2...> SRP_s=<s>
```

Where G1,G2... are proposed groups, in order of preference.

The initiator MUST either close the connection or continue with:

SRP_A=<A> SRP_GROUP=<G>

Where G is one of G1,G2... that were proposed by the target.

The target MUST answer with a Login reject with the "Authentication Failure" status or reply with:

SRP_B=<B>

The initiator MUST close the connection or continue with:

SRP_M=<M>

If the initiator authentication fails, the target MUST answer with a Login reject with "Authentication Failure" status. Otherwise, if the initiator sent TargetAuth=Yes in the first message (requiring target authentication), the target MUST reply with:

SRP_HM=<H(A | M | K)>

If the target authentication fails, the initiator MUST close the connection.

Where U, s, A, B, M, and H(A | M | K) are defined in [RFC2945] (using the SHA1 hash function, such as SRP-SHA1) and G,Gn (Gn stands for G1,G2...) are identifiers of SRP groups specified in [RFC3723]. G, Gn, and U are text strings, s,A,B,M, and H(A | M | K) are binary-values. The length of s,A,B,M and H(A | M | K) in binary form (not the length of the character string that represents them in encoded form) MUST not exceed 1024 bytes.

For the SRP_GROUP, all the groups specified in [RFC3723] up to 1536 bits (i.e., SRP-768, SRP-1024, SRP-1280, SRP-1536) must be supported by initiators and targets. To guarantee interoperability, targets MUST always offer "SRP-1536" as one of the proposed groups.

11.1.4. Challenge Handshake Authentication Protocol (CHAP)

For CHAP [RFC1994], in the first step, the initiator MUST send:

CHAP_A=<A1,A2...>

Where A1,A2... are proposed algorithms, in order of preference.

Docker EX1014
Page 186 of 257

In the second step, the target MUST answer with a Login reject with the "Authentication Failure" status or reply with:

```
CHAP_A=<A> CHAP_I=<I> CHAP_C=<C>
```

Where A is one of A1,A2... that were proposed by the initiator.

In the third step, the initiator MUST continue with:

```
CHAP_N=<N> CHAP_R=<R>
```

or, if it requires target authentication, with:

```
CHAP_N=<N> CHAP_R=<R> CHAP_I=<I> CHAP_C=<C>
```

If the initiator authentication fails, the target MUST answer with a Login reject with "Authentication Failure" status. Otherwise, if the initiator required target authentication, the target MUST either answer with a Login reject with "Authentication Failure" or reply with:

```
CHAP_N=<N> CHAP_R=<R>
```

If target authentication fails, the initiator MUST close the connection.

Where N, (A,A1,A2), I, C, and R are (correspondingly) the Name, Algorithm, Identifier, Challenge, and Response as defined in [RFC1994], N is a text string, A,A1,A2, and I are numbers, and C and R are large-binary-values and their binary length (not the length of the character string that represents them in encoded form) MUST not exceed 1024 bytes.

For the Algorithm, as stated in [RFC1994], one value is required to be implemented:

```
5     (CHAP with MD5)
```

To guarantee interoperability, initiators MUST always offer it as one of the proposed algorithms.

12. Login/Text Operational Text Keys

Some session specific parameters MUST only be carried on the leading connection and cannot be changed after the leading connection login (e.g., MaxConnections, the maximum number of connections). This

Docker EX1014
Page 187 of 257

holds for a single connection session with regard to connection restart. The keys that fall into this category have the use: LO (Leading Only).

Keys that can only be used during login have the use: IO (initialize only), while those that can be used in both the Login Phase and Full Feature Phase have the use: ALL.

Keys that can only be used during Full Feature Phase use FFPO (Full Feature Phase only).

Keys marked as Any-Stage may also appear in the SecurityNegotiation stage while all other keys described in this chapter are operational keys.

Keys that do not require an answer are marked as Declarative.

Key scope is indicated as session-wide (SW) or connection-only (CO).

Result function, wherever mentioned, states the function that can be applied to check the validity of the responder selection. Minimum means that the selected value cannot exceed the offered value. Maximum means that the selected value cannot be lower than the offered value. AND means that the selected value must be a possible result of a Boolean "and" function with an arbitrary Boolean value (e.g., if the offered value is No the selected value must be No). OR means that the selected value must be a possible result of a Boolean "or" function with an arbitrary Boolean value (e.g., if the offered value is Yes the selected value must be Yes).

12.1. HeaderDigest and DataDigest

Use: IO
Senders: Initiator and Target
Scope: CO

HeaderDigest = <list-of-values>
DataDigest = <list-of-values>

Default is None for both HeaderDigest and DataDigest.

Digests enable the checking of end-to-end, non-cryptographic data integrity beyond the integrity checks provided by the link layers and the covering of the whole communication path including all elements that may change the network level PDUs such as routers, switches, and proxies.

Docker EX1014
Page 188 of 257

The following table lists cyclic integrity checksums that can be negotiated for the digests and that MUST be implemented by every iSCSI initiator and target. These digest options only have error detection significance.

Name	Description	Generator
CRC32C	32 bit CRC	0x11edc6f41
None	no digest	

The generator polynomial for this digest is given in hex-notation (e.g., 0x3b stands for 0011 1011 and the polynomial is x**5+X**4+x**3+x+1).

When the Initiator and Target agree on a digest, this digest MUST be used for every PDU in Full Feature Phase.

Padding bytes, when present in a segment covered by a CRC, SHOULD be set to 0 and are included in the CRC.

The CRC MUST be calculated by a method that produces the same results as the following process:

- The PDU bits are considered as the coefficients of a polynomial M(x) of degree n-1; bit 7 of the lowest numbered byte is considered the most significant bit (x^n-1), followed by bit 6 of the lowest numbered byte through bit 0 of the highest numbered byte (x^0).

- The most significant 32 bits are complemented.

- The polynomial is multiplied by x^32 then divided by G(x). The generator polynomial produces a remainder R(x) of degree <= 31.

- The coefficients of R(x) are considered a 32 bit sequence.

- The bit sequence is complemented and the result is the CRC.

- The CRC bits are mapped into the digest word. The x^31 coefficient in bit 7 of the lowest numbered byte of the digest continuing through to the byte up to the x^24 coefficient in bit 0 of the lowest numbered byte, continuing with the x^23 coefficient in bit 7 of next byte through x^0 in bit 0 of the highest numbered byte.

Docker EX1014
Page 189 of 257

- Computing the CRC over any segment (data or header) extended to include the CRC built using the generator 0x11edc6f41 will always get the value 0x1c2d19ed as its final remainder (R(x)). This value is given here in its polynomial form (i.e., not mapped as the digest word).

For a discussion about selection criteria for the CRC, see [RFC3385]. For a detailed analysis of the iSCSI polynomial, see [Castagnoli93].

Private or public extension algorithms MAY also be negotiated for digests. Whenever a private or public digest extension algorithm is part of the default offer (the offer made in absence of explicit administrative action) the implementer MUST ensure that CRC32C is listed as an alternative in the default offer and "None" is not part of the default offer.

Extension digest algorithms MUST be named using one of the following two formats:

a) Y-reversed.vendor.dns_name.do_something=
b) Y<#><IANA-registered-string>=

Digests named using the Y- format are used for private purposes (unregistered). Digests named using the Y# format (public extension) must be registered with IANA and MUST be described by an informational RFC.

For private extension digests, to identify the vendor, we suggest you use the reversed DNS-name as a prefix to the proper digest names.

The part of digest-name following Y- and Y# MUST conform to the format for standard-label specified in Section 5.1 Text Format.

Support for public or private extension digests is OPTIONAL.

12.2. MaxConnections

Use: LO
Senders: Initiator and Target
Scope: SW
Irrelevant when: SessionType=Discovery

MaxConnections=<numerical-value-from-1-to-65535>

Default is 1.
Result function is Minimum.

Initiator and target negotiate the maximum number of connections requested/acceptable.

12.3. SendTargets

Use: FFPO
Senders: Initiator
Scope: SW

For a complete description, see Appendix D. - SendTargets Operation -.

12.4. TargetName

Use: IO by initiator, FFPO by target - only as response to a SendTargets, Declarative, Any-Stage

Senders: Initiator and Target
Scope: SW

TargetName=<iSCSI-name-value>

Examples:

```
TargetName=iqn.1993-11.com.disk-vendor:diskarrays.sn.45678
TargetName=eui.020000023B040506
```

The initiator of the TCP connection MUST provide this key to the remote endpoint in the first login request if the initiator is not establishing a discovery session. The iSCSI Target Name specifies the worldwide unique name of the target.

The TargetName key may also be returned by the "SendTargets" text request (which is its only use when issued by a target).

TargetName MUST not be redeclared within the login phase.

Docker EX1014
Page 191 of 257

12.5. InitiatorName

Use: IO, Declarative, Any-Stage
Senders: Initiator
Scope: SW

InitiatorName=<iSCSI-name-value>

Examples:

```
InitiatorName=iqn.1992-04.com.os-vendor.plan9:cdrom.12345
InitiatorName=iqn.2001-02.com.ssp.users:customer235.host90
```

The initiator of the TCP connection MUST provide this key to the remote endpoint at the first Login of the Login Phase for every connection. The InitiatorName key enables the initiator to identify itself to the remote endpoint.

InitiatorName MUST not be redeclared within the login phase.

12.6. TargetAlias

Use: ALL, Declarative, Any-Stage
Senders: Target
Scope: SW

TargetAlias=<iSCSI-local-name-value>

Examples:

```
TargetAlias=Bob-s Disk
TargetAlias=Database Server 1 Log Disk
TargetAlias=Web Server 3 Disk 20
```

If a target has been configured with a human-readable name or description, this name SHOULD be communicated to the initiator during a Login Response PDU if SessionType=Normal (see Section 12.21 SessionType). This string is not used as an identifier, nor is it meant to be used for authentication or authorization decisions. It can be displayed by the initiator’s user interface in a list of targets to which it is connected.

Docker EX1014
Page 192 of 257

12.7. InitiatorAlias

Use: ALL, Declarative, Any-Stage
Senders: Initiator
Scope: SW

InitiatorAlias=<iSCSI-local-name-value>

Examples:

```
InitiatorAlias=Web Server 4
InitiatorAlias=spyalley.nsa.gov
InitiatorAlias=Exchange Server
```

If an initiator has been configured with a human-readable name or description, it SHOULD be communicated to the target during a Login Request PDU. If not, the host name can be used instead. This string is not used as an identifier, nor is meant to be used for authentication or authorization decisions. It can be displayed by the target's user interface in a list of initiators to which it is connected.

12.8. TargetAddress

Use: ALL, Declarative, Any-Stage
Senders: Target
Scope: SW

TargetAddress=domainname[:port][,portal-group-tag]

The domainname can be specified as either a DNS host name, a dotted-decimal IPv4 address, or a bracketed IPv6 address as specified in [RFC2732].

If the TCP port is not specified, it is assumed to be the IANA-assigned default port for iSCSI (see Section 13 IANA Considerations).

If the TargetAddress is returned as the result of a redirect status in a login response, the comma and portal group tag MUST be omitted.

If the TargetAddress is returned within a SendTargets response, the portal group tag MUST be included.

Examples:

```
TargetAddress=10.0.0.1:5003,1
TargetAddress=[1080:0:0:0:8:800:200C:417A],65
TargetAddress=[1080::8:800:200C:417A]:5003,1
TargetAddress=computingcenter.example.com,23
```

Use of the portal-group-tag is described in Appendix D. - SendTargets Operation -. The formats for the port and portal-group-tag are the same as the one specified in Section 12.9 TargetPortalGroupTag.

12.9. TargetPortalGroupTag

Use: IO by target, Declarative, Any-Stage
Senders: Target
Scope: SW

TargetPortalGroupTag=<16-bit-binary-value>

Examples:
TargetPortalGroupTag=1

The target portal group tag is a 16-bit binary-value that uniquely identifies a portal group within an iSCSI target node. This key carries the value of the tag of the portal group that is servicing the Login request. The iSCSI target returns this key to the initiator in the Login Response PDU to the first Login Request PDU that has the C bit set to 0 when TargetName is given by the initiator.

For the complete usage expectations of this key see Section 5.3 Login Phase.

12.10. InitialR2T

Use: LO
Senders: Initiator and Target
Scope: SW
Irrelevant when: SessionType=Discovery

InitialR2T=<boolean-value>

Examples:

```
I->InitialR2T=No
T->InitialR2T=No
```

Default is Yes.
Result function is OR.

The InitialR2T key is used to turn off the default use of R2T for unidirectional and the output part of bidirectional commands, thus allowing an initiator to start sending data to a target as if it has received an initial R2T with Buffer Offset=Immediate Data Length and Desired Data Transfer Length=(min(FirstBurstLength, Expected Data Transfer Length) - Received Immediate Data Length).

The default action is that R2T is required, unless both the initiator and the target send this key-pair attribute specifying InitialR2T=No. Only the first outgoing data burst (immediate data and/or separate PDUs) can be sent unsolicited (i.e., not requiring an explicit R2T).

12.11. ImmediateData

Use: LO
Senders: Initiator and Target
Scope: SW
Irrelevant when: SessionType=Discovery

ImmediateData=<boolean-value>

Default is Yes.
Result function is AND.

The initiator and target negotiate support for immediate data. To turn immediate data off, the initiator or target must state its desire to do so. ImmediateData can be turned on if both the initiator and target have ImmediateData=Yes.

If ImmediateData is set to Yes and InitialR2T is set to Yes (default), then only immediate data are accepted in the first burst.

If ImmediateData is set to No and InitialR2T is set to Yes, then the initiator MUST NOT send unsolicited data and the target MUST reject unsolicited data with the corresponding response code.

If ImmediateData is set to No and InitialR2T is set to No, then the initiator MUST NOT send unsolicited immediate data, but MAY send one unsolicited burst of Data-Out PDUs.

If ImmediateData is set to Yes and InitialR2T is set to No, then the initiator MAY send unsolicited immediate data and/or one unsolicited burst of Data-Out PDUs.

The following table is a summary of unsolicited data options:

InitialR2T	ImmediateData	Unsolicited Data Out PDUs	Immediate Data
No	No	Yes	No
No	Yes	Yes	Yes
Yes	No	No	No
Yes	Yes	No	Yes

12.12. MaxRecvDataSegmentLength

Use: ALL, Declarative
Senders: Initiator and Target
Scope: CO

MaxRecvDataSegmentLength=<numerical-value-512-to-(2**24-1)>

Default is 8192 bytes.

The initiator or target declares the maximum data segment length in bytes it can receive in an iSCSI PDU.

The transmitter (initiator or target) is required to send PDUs with a data segment that does not exceed MaxRecvDataSegmentLength of the receiver.

A target receiver is additionally limited by MaxBurstLength for solicited data and FirstBurstLength for unsolicited data. An initiator MUST NOT send solicited PDUs exceeding MaxBurstLength nor unsolicited PDUs exceeding FirstBurstLength (or FirstBurstLength-Immediate Data Length if immediate data were sent).

12.13. MaxBurstLength

Use: LO
Senders: Initiator and Target
Scope: SW
Irrelevant when: SessionType=Discovery

MaxBurstLength=<numerical-value-512-to-(2**24-1)>

Default is 262144 (256 Kbytes).
Result function is Minimum.

The initiator and target negotiate maximum SCSI data payload in bytes in a Data-In or a solicited Data-Out iSCSI sequence. A sequence consists of one or more consecutive Data-In or Data-Out PDUs that end with a Data-In or Data-Out PDU with the F bit set to one.

12.14. FirstBurstLength

Use: LO
Senders: Initiator and Target
Scope: SW
Irrelevant when: SessionType=Discovery
Irrelevant when: (InitialR2T=Yes and ImmediateData=No)

FirstBurstLength=<numerical-value-512-to-(2**24-1)>

Default is 65536 (64 Kbytes).
Result function is Minimum.

The initiator and target negotiate the maximum amount in bytes of unsolicited data an iSCSI initiator may send to the target during the execution of a single SCSI command. This covers the immediate data (if any) and the sequence of unsolicited Data-Out PDUs (if any) that follow the command.

FirstBurstLength MUST NOT exceed MaxBurstLength.

12.15. DefaultTime2Wait

Use: LO
Senders: Initiator and Target
Scope: SW

DefaultTime2Wait=<numerical-value-0-to-3600>

Default is 2.
Result function is Maximum.

The initiator and target negotiate the minimum time, in seconds, to wait before attempting an explicit/implicit logout or an active task reassignment after an unexpected connection termination or a connection reset.

A value of 0 indicates that logout or active task reassignment can be attempted immediately.

12.16. DefaultTime2Retain

Use: LO Senders: Initiator and Target Scope: SW

DefaultTime2Retain=<numerical-value-0-to-3600>

Default is 20. Result function is Minimum.

The initiator and target negotiate the maximum time, in seconds after an initial wait (Time2Wait), before which an active task reassignment is still possible after an unexpected connection termination or a connection reset.

This value is also the session state timeout if the connection in question is the last LOGGED_IN connection in the session.

A value of 0 indicates that connection/task state is immediately discarded by the target.

12.17. MaxOutstandingR2T

Use: LO
Senders: Initiator and Target
Scope: SW

MaxOutstandingR2T=<numerical-value-from-1-to-65535>
Irrelevant when: SessionType=Discovery

Default is 1.
Result function is Minimum.

Initiator and target negotiate the maximum number of outstanding R2Ts per task, excluding any implied initial R2T that might be part of that task. An R2T is considered outstanding until the last data PDU (with the F bit set to 1) is transferred, or a sequence reception timeout (Section 6.1.4.1 Recovery Within-command) is encountered for that data sequence.

12.18. DataPDUInOrder

Use: LO
Senders: Initiator and Target
Scope: SW
Irrelevant when: SessionType=Discovery

DataPDUInOrder=<boolean-value>

Default is Yes.
Result function is OR.

No is used by iSCSI to indicate that the data PDUs within sequences can be in any order. Yes is used to indicate that data PDUs within sequences have to be at continuously increasing addresses and overlays are forbidden.

12.19. DataSequenceInOrder

Use: LO
Senders: Initiator and Target
Scope: SW
Irrelevant when: SessionType=Discovery

DataSequenceInOrder=<boolean-value>

Default is Yes.
Result function is OR.

A Data Sequence is a sequence of Data-In or Data-Out PDUs that end with a Data-In or Data-Out PDU with the F bit set to one. A Data-Out sequence is sent either unsolicited or in response to an R2T. Sequences cover an offset-range.

If DataSequenceInOrder is set to No, Data PDU sequences may be transferred in any order.

If DataSequenceInOrder is set to Yes, Data Sequences MUST be transferred using continuously non-decreasing sequence offsets (R2T buffer offset for writes, or the smallest SCSI Data-In buffer offset within a read data sequence).

If DataSequenceInOrder is set to Yes, a target may retry at most the last R2T, and an initiator may at most request retransmission for the last read data sequence. For this reason, if ErrorRecoveryLevel is not 0 and DataSequenceInOrder is set to Yes then MaxOustandingR2T MUST be set to 1.

12.20. ErrorRecoveryLevel

Use: LO
Senders: Initiator and Target
Scope: SW

ErrorRecoveryLevel=<numerical-value-0-to-2>

Docker EX1014
Page 199 of 257

Default is 0.
Result function is Minimum.

The initiator and target negotiate the recovery level supported.

Recovery levels represent a combination of recovery capabilities. Each recovery level includes all the capabilities of the lower recovery levels and adds some new ones to them.

In the description of recovery mechanisms, certain recovery classes are specified. Section 6.1.5 Error Recovery Hierarchy describes the mapping between the classes and the levels.

12.21. SessionType

Use: LO, Declarative, Any-Stage
Senders: Initiator
Scope: SW

SessionType= <Discovery|Normal>

Default is Normal.

The initiator indicates the type of session it wants to create. The target can either accept it or reject it.

A discovery session indicates to the Target that the only purpose of this Session is discovery. The only requests a target accepts in this type of session are a text request with a SendTargets key and a logout request with reason "close the session".

The discovery session implies MaxConnections = 1 and overrides both the default and an explicit setting.

12.22. The Private or Public Extension Key Format

Use: ALL
Senders: Initiator and Target
Scope: specific key dependent

X-reversed.vendor.dns_name.do_something=

or

X<#><IANA-registered-string>=

Docker EX1014
Page 200 of 257

Keys with this format are used for public or private extension purposes. These keys always start with X- if unregistered with IANA (private) or X# if registered with IANA (public).

For unregistered keys, to identify the vendor, we suggest you use the reversed DNS-name as a prefix to the key-proper.

The part of key-name following X- and X# MUST conform to the format for key-name specified in Section 5.1 Text Format.

For IANA registered keys the string following X# must be registered with IANA and the use of the key MUST be described by an informational RFC.

Vendor specific keys MUST ONLY be used in normal sessions.

Support for public or private extension keys is OPTIONAL.

13. IANA Considerations

This section conforms to [RFC2434].

The well-known user TCP port number for iSCSI connections assigned by IANA is 3260 and this is the default iSCSI port. Implementations needing a system TCP port number may use port 860, the port assigned by IANA as the iSCSI system port; however in order to use port 860, it MUST be explicitly specified - implementations MUST NOT default to use of port 860, as 3260 is the only allowed default.

Extension keys, authentication methods, or digest types for which a vendor or group of vendors intend to provide publicly available descriptions MUST be described by an RFC and MUST be registered with IANA.

The IANA has set up the following three registries:

a) iSCSI extended key registry
b) iSCSI authentication methods registry
c) iSCSI digests registry

[RFC3723] also instructs IANA to maintain a registry for the values of the SRP_GROUP key. The format of these values must conform to the one specified for iSCSI extension item-label in Section 13.5.4 Standard iSCSI extension item-label format.

Docker EX1014
Page 201 of 257

For the iSCSI authentication methods registry and the iSCSI digests registry, IANA MUST also assign a 16-bit unsigned integer number (the method number for the authentication method and the digest number for the digest).

The following initial values for the registry for authentication methods are specified by the standards action of this document:

Authentication Method	Number
CHAP	1
SRP	2
KRB5	3
SPKM1	4
SPKM2	5

All other record numbers from 0 to 255 are reserved. IANA will register numbers above 255.

Authentication methods with numbers above 255 MUST be unique within the registry and MUST be used with the prefix Z#.

The following initial values for the registry for digests are specified by the standards action of this document:

Digest	Number
CRC32C	1

All other record numbers from 0 to 255 are reserved. IANA will register numbers above 255.

Digests with numbers above 255 MUST be unique within the registry and MUST be used with the prefix Y#.

The RFC that describes the item to be registered MUST indicate in the IANA Considerations section the string and iSCSI registry to which it should be recorded.

Extension Keys, Authentication Methods, and digests (iSCSI extension items) must conform to a number of requirements as described below.

Docker EX1014
Page 202 of 257

13.1. Naming Requirements

Each iSCSI extension item must have a unique name in its category. This name will be used as a standard-label for the key, access method, or digest and must conform to the syntax specified in Section 13.5.4 Standard iSCSI extension item-label format for iSCSI extension item-labels.

13.2. Mechanism Specification Requirements

For iSCSI extension items all of the protocols and procedures used by a given iSCSI extension item must be described, either in the specification of the iSCSI extension item itself or in some other publicly available specification, in sufficient detail for the iSCSI extension item to be implemented by any competent implementor. Use of secret and/or proprietary methods in iSCSI extension items are expressly prohibited. In addition, the restrictions imposed by [RFC1602] on the standardization of patented algorithms must be respected.

13.3. Publication Requirements

All iSCSI extension items must be described by an RFC. The RFC may be informational rather than Standards-Track, although Standards Track review and approval are encouraged for all iSCSI extension items.

13.4. Security Requirements

Any known security issues that arise from the use of the iSCSI extension item must be completely and fully described. It is not required that the iSCSI extension item be secure or that it be free from risks, but that the known risks be identified. Publication of a new iSCSI extension item does not require an exhaustive security review, and the security considerations section is subject to continuing evaluation.

Additional security considerations should be addressed by publishing revised versions of the iSCSI extension item specification.

For each of these registries, IANA must record the registered string, which MUST conform to the format rules described in Section 13.5.4 Standard iSCSI extension item-label format for iSCSI extension item-labels, and the RFC number that describes it. The key prefix (X#, Y# or Z#) is not part of the recorded string.

Docker EX1014
Page 203 of 257

13.5. Registration Procedure

Registration of a new iSCSI extension item starts with the construction of an Internet Draft to become an RFC.

13.5.1. Present the iSCSI extension item to the Community

Send a proposed access type specification to the IPS WG mailing list, or if the IPS WG is disbanded at the registration time, to a mailing list designated by the IETF Transport Area Director for a review period of a month. The intent of the public posting is to solicit comments and feedback on the iSCSI extension item specification and a review of any security considerations.

13.5.2. iSCSI extension item review and IESG approval

When the one month period has passed, the IPS WG chair or a person nominated by the IETF Transport Area Director (the iSCSI extension item reviewer) forwards the Internet Draft to the IESG for publication as an informational RFC or rejects it. If the specification is a standards track document, the usual IETF procedures for such documents are followed.

Decisions made by the iSCSI extension item reviewer must be published within two weeks after the month-long review period. Decisions made by the iSCSI extension item reviewer can be appealed through the IESG appeal process.

13.5.3. IANA Registration

Provided that the iSCSI extension item has either passed review or has been successfully appealed to the IESG, and the specification is published as an RFC, then IANA will register the iSCSI extension item and make the registration available to the community.

13.5.4. Standard iSCSI extension item-label format

The following character symbols are used iSCSI extension item-labels (the hexadecimal values represent Unicode code points):

(a-z, A-Z) - letters
(0-9) - digits
"." (0x2e) - dot
"-" (0x2d) - minus
"+" (0x2b) - plus
"@" (0x40) - commercial at
"_" (0x5f) - underscore

Docker EX1014
Page 204 of 257

An iSCSI extension item-label is a string of one or more characters that consist of letters, digits, dot, minus, plus, commercial at, or underscore. An iSCSI extension item-label MUST begin with a capital letter and must not exceed 63 characters.

13.6. IANA Procedures for Registering iSCSI extension items

The identity of the iSCSI extension item reviewer is communicated to the IANA by the IESG. Then, the IANA only acts in response to iSCSI extension item definitions that are approved by the iSCSI extension item reviewer and forwarded by the reviewer to the IANA for registration, or in response to a communication from the IESG that an iSCSI extension item definition appeal has overturned the iSCSI extension item reviewer’s ruling.

References

Normative References

[CAM] ANSI X3.232-199X, Common Access Method-3.

[EUI] "Guidelines for 64-bit Global Identifier (EUI-64)", http: //standards.ieee.org/regauth/oui/tutorials/EUI64.html

[OUI] "IEEE OUI and Company_Id Assignments", http://standards.ieee.org/regauth/oui

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1035] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, November 1987.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts-Communication Layer", STD 3, RFC 1122, October 1989.

[RFC1510] Kohl, J. and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[RFC1737] Sollins, K. and L. Masinter "Functional Requirements for Uniform Resource Names"RFC 1737, December 1994.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, August 1996.

[RFC1994] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

[RFC2025] Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC 2025, October 1996.

[RFC2045] Borenstein, N. and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2119] Bradner, S. "Key Words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2279] Yergeau, F., "UTF-8, a Transformation Format of ISO 10646", RFC 2279 October 1996.

[RFC2373] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter "Uniform Resource Identifiers", RFC 2396, August 1998.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC2404] Madson, C. and R. Glenn, "The Use of HMAC-SHA-1-96 within ESP and AH", RFC 2404, November 1998.

[RFC2406] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[RFC2407] Piper, D., "The Internet IP Security Domain of Interpretation of ISAKMP", RFC 2407, November 1998.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC2409, November 1998.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs.", BCP 26, RFC 2434, October 1998.

Docker EX1014
Page 206 of 257

[RFC2451] Pereira, R. and R. Adams " The ESP CBC-Mode Cipher Algorithms", RFC 2451, November 1998.

[RFC2732] Hinden, R., Carpenter, B. and L. Masinter, "Format for Literal IPv6 Addresses in URL’s", RFC 2451, December 1999.

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, September 2000.

[RFC3066] Alvestrand, H., "Tags for the Identification of Languages", STD 47, RFC 3066, January 2001.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[RFC3566] Frankel, S. and H. Herbert, "The AES-XCBC-MAC-96 Algorithm and Its Use With IPsec", RFC 3566, September 2003.

[RFC3686] Housley, R., "Using Advanced Encryption Standard (AES) Counter Mode with IPsec Encapsulating Security Payload (ESP)", RFC 3686, January 2004.

[RFC3722] Bakke, M., "String Profile for Internet Small Computer Systems Interface (iSCSI) Names", RFC 3722, March 2004.

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

[SAM2] T10/1157D, SCSI Architecture Model - 2 (SAM-2).

[SBC] NCITS.306-1998, SCSI-3 Block Commands (SBC).

[SPC3] T10/1416-D, SCSI Primary Commands-3.

[UNICODE] Unicode Standard Annex #15, "Unicode Normalization Forms", http://www.unicode.org/unicode/reports/tr15

Docker EX1014
Page 207 of 257

Informative References

[BOOT] P. Sarkar, et al., "Bootstrapping Clients using the iSCSI Protocol", Work in Progress, July 2003.

[Castagnoli93] G. Castagnoli, S. Braeuer and M. Herrman "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits", IEEE Transact. on Communications, Vol. 41, No. 6, June 1993.

[CORD] Chadalapaka, M. and R. Elliott, "SCSI Command Ordering Considerations with iSCSI", Work in Progress.

[RFC3347] Krueger, M., Haagens, R., Sapuntzakis, C. and M. Bakke, "Small Computer Systems Interface protocol over the Internet (iSCSI) Requirements and Design Considerations", RFC 3347, July 2002.

[RFC3385] Sheinwald, D., Staran, J., Thaler, P. and V. Cavanna, "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC)/Checksum Considerations", RFC 3385, September 2002.

[RFC3721] Bakke M., Hafner, J., Hufferd, J., Voruganti, K. and M. Krueger, "Internet Small Computer Systems Interface (iSCSI) Naming and Discovery, RFC 3721, March 2004.

[SEQ-EXT] Kent, S., "IP Encapsulating Security Payload (ESP)", Work in Progress, July 2002.

Appendix A. Sync and Steering with Fixed Interval Markers

This appendix presents a simple scheme for synchronization (PDU boundary retrieval). It uses markers that include synchronization information placed at fixed intervals in the TCP stream.

A Marker consists of:

```
Byte /    0       |       1       |       2       |       3       |
    /             |               |               |               |
   |0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|0 1 2 3 4 5 6 7|
   +---------------+---------------+---------------+---------------+
  0| Next-iSCSI-PDU-start pointer - copy #1                        |
   +---------------+---------------+---------------+---------------+
  4| Next-iSCSI-PDU-start pointer - copy #2                        |
   +---------------+---------------+---------------+---------------+
```

The Marker scheme uses payload byte stream counting that includes every byte placed by iSCSI in the TCP stream except for the markers themselves. It also excludes any bytes that TCP counts but are not originated by iSCSI.

Markers MUST NOT be included in digest calculation.

The Marker indicates the offset to the next iSCSI PDU header. The Marker is eight bytes in length and contains two 32-bit offset fields that indicate how many bytes to skip in the TCP stream in order to find the next iSCSI PDU header. The marker uses two copies of the pointer so that a marker that spans a TCP packet boundary should leave at least one valid copy in one of the packets.

The structure and semantics of an inserted marker are independent of the marker interval.

The use of markers is negotiable. The initiator and target MAY indicate their readiness to receive and/or send markers during login separately for each connection. The default is No.

A.1. Markers At Fixed Intervals

A marker is inserted at fixed intervals in the TCP byte stream. During login, each end of the iSCSI session specifies the interval at which it is willing to receive the marker, or it disables the marker altogether. If a receiver indicates that it desires a marker, the sender MAY agree (during negotiation) and provide the marker at the desired interval. However, in certain environments, a sender that does not provide markers to a receiver that wants markers may suffer an appreciable performance degradation.

Docker EX1014
Page 209 of 257

The marker interval and the initial marker-less interval are counted in terms of the bytes placed in the TCP stream data by iSCSI.

When reduced to iSCSI terms, markers MUST indicate the offset to a 4-byte word boundary in the stream. The least significant two bits of each marker word are reserved and are considered 0 for offset computation.

Padding iSCSI PDU payloads to 4-byte word boundaries simplifies marker manipulation.

A.2. Initial Marker-less Interval

To enable the connection setup including the Login Phase negotiation, marking (if any) is only started at the first marker interval after the end of the Login Phase. However, in order to enable the marker inclusion and exclusion mechanism to work without knowledge of the length of the Login Phase, the first marker will be placed in the TCP stream as if the Marker-less interval had included markers.

Thus, all markers appear in the stream at locations conforming to the formula: [(MI + 8) * n - 8] where MI = Marker Interval, n = integer number.

For example, if the marker interval is 512 bytes and the login ended at byte 1003 (first iSCSI placed byte is 0), the first marker will be inserted after byte 1031 in the stream.

A.3. Negotiation

The following operational key=value pairs are used to negotiate the fixed interval markers. The direction (output or input) is relative to the initiator.

A.3.1. OFMarker, IFMarker

Use: IO
Senders: Initiator and Target
Scope: CO

OFMarker=<boolean-value>
IFMarker=<boolean-value>

Default is No.

Result function is AND.

Docker EX1014
Page 210 of 257

OFMarker is used to turn on or off the initiator to target markers on the connection. IFMarker is used to turn on or off the target to initiator markers on the connection.

Examples:

```
I->OFMarker=Yes,IFMarker=Yes
T->OFMarker=Yes,IFMarker=Yes
```

Results in the Marker being used in both directions while:

```
I->OFMarker=Yes,IFMarker=Yes
T->OFMarker=Yes,IFMarker=No
```

Results in Marker being used from the initiator to the target, but not from the target to initiator.

A.3.2. OFMarkInt, IFMarkInt

Use: IO
Senders: Initiator and Target
Scope: CO
OFMarkInt is Irrelevant when: OFMarker=No
IFMarkInt is Irrelevant when: IFMarker=No

Offering:

```
OFMarkInt=<numeric-range-from-1-to-65535>
IFMarkInt=<numeric-range-from-1-to-65535>
```

Responding:

```
OFMarkInt=<numeric-value-from-1-to-65535>|Reject
IFMarkInt=<numeric-value-from-1-to-65535>|Reject
```

OFMarkInt is used to set the interval for the initiator to target markers on the connection. IFMarkInt is used to set the interval for the target to initiator markers on the connection.

For the offering, the initiator or target indicates the minimum to maximum interval (in 4-byte words) it wants the markers for one or both directions. In case it only wants a specific value, only a single value has to be specified. The responder selects a value within the minimum and maximum offered or the only value offered or indicates through the xFMarker key=value its inability to set and/or receive markers. When the interval is unacceptable the responder answers with "Reject". Reject is resetting the marker function in the specified direction (Output or Input) to No.

Docker EX1014
Page 211 of 257

The interval is measured from the end of a marker to the beginning of the next marker. For example, a value of 1024 means 1024 words (4096 bytes of iSCSI payload between markers).

The default is 2048.

Appendix B. Examples

B.1. Read Operation Example

```
+------------------+-----------------------+----------------------+
|Initiator Function|    PDU Type           |  Target Function     |
+------------------+-----------------------+----------------------+
|  Command request |SCSI Command (READ)>>> |                      |
|  (read)          |                       |                      |
+------------------+-----------------------+----------------------+
|                  |                       |Prepare Data Transfer |
+------------------+-----------------------+----------------------+
|   Receive Data   |   <<< SCSI Data-In    |   Send Data          |
+------------------+-----------------------+----------------------+
|   Receive Data   |   <<< SCSI Data-In    |   Send Data          |
+------------------+-----------------------+----------------------+
|   Receive Data   |   <<< SCSI Data-In    |   Send Data          |
+------------------+-----------------------+----------------------+
|                  |   <<< SCSI Response   |Send Status and Sense |
+------------------+-----------------------+----------------------+
| Command Complete |                       |                      |
+------------------+-----------------------+----------------------+
```

B.2. Write Operation Example

```
+------------------+-----------------------+---------------------+
|Initiator Function|    PDU Type           |  Target Function    |
+------------------+-----------------------+---------------------+
| Command request  |SCSI Command (WRITE)>>>| Receive command     |
|  (write)         |                       | and queue it        |
+------------------+-----------------------+---------------------+
|                  |                       | Process old commands|
+------------------+-----------------------+---------------------+
|                  |                       | Ready to process    |
|                  |   <<< R2T             | WRITE command       |
+------------------+-----------------------+---------------------+
|   Send Data      |   SCSI Data-Out >>>   |   Receive Data      |
+------------------+-----------------------+---------------------+
|                  |   <<< R2T             | Ready for data      |
+------------------+-----------------------+---------------------+
|                  |   <<< R2T             | Ready for data      |
+------------------+-----------------------+---------------------+
|   Send Data      |   SCSI Data-Out >>>   |   Receive Data      |
+------------------+-----------------------+---------------------+
|   Send Data      |   SCSI Data-Out >>>   |   Receive Data      |
+------------------+-----------------------+---------------------+
|                  |   <<< SCSI Response   |Send Status and Sense|
+------------------+-----------------------+---------------------+
| Command Complete |                       |                     |
+------------------+-----------------------+---------------------+
```

B.3. R2TSN/DataSN Use Examples

Output (write) data DataSN/R2TSN Example

```
+------------------+-----------------------+----------------------+
|Initiator Function|    PDU Type & Content |  Target Function     |
+------------------+-----------------------+----------------------+
|  Command request |SCSI Command (WRITE)>>>| Receive command      |
|  (write)         |                       | and queue it         |
+------------------+-----------------------+----------------------+
|                  |                       | Process old commands |
+------------------+-----------------------+----------------------+
|                  |   <<< R2T             | Ready for data       |
|                  |   R2TSN = 0           |                      |
+------------------+-----------------------+----------------------+
|                  |   <<< R2T             | Ready for more data  |
|                  |   R2TSN = 1           |                      |
+------------------+-----------------------+----------------------+
|  Send Data       |   SCSI Data-Out >>>   |   Receive Data       |
|  for R2TSN 0     |   DataSN = 0, F=0     |                      |
+------------------+-----------------------+----------------------+
|  Send Data       |   SCSI Data-Out >>>   |   Receive Data       |
|  for R2TSN 0     |   DataSN = 1, F=1     |                      |
+------------------+-----------------------+----------------------+
|  Send Data       |   SCSI Data >>>       |   Receive Data       |
|  for R2TSN 1     |   DataSN = 0, F=1     |                      |
+------------------+-----------------------+----------------------+
|                  |   <<< SCSI Response   |Send Status and Sense |
|                  |   ExpDataSN = 0       |                      |
+------------------+-----------------------+----------------------+
| Command Complete |                       |                      |
+------------------+-----------------------+----------------------+
```

Input (read) data DataSN Example

```
+------------------+----------------------+----------------------+
|Initiator Function|    PDU Type          |  Target Function     |
+------------------+----------------------+----------------------+
|  Command request |SCSI Command (READ)>>> |                      |
|  (read)          |                      |                      |
+------------------+----------------------+----------------------+
|                  |                      | Prepare Data Transfer|
+------------------+----------------------+----------------------+
|   Receive Data   |   <<< SCSI Data-In   |   Send Data          |
|                  |   DataSN = 0, F=0    |                      |
+------------------+----------------------+----------------------+
|   Receive Data   |   <<< SCSI Data-In   |   Send Data          |
|                  |   DataSN = 1, F=0    |                      |
+------------------+----------------------+----------------------+
|   Receive Data   |   <<< SCSI Data-In   |   Send Data          |
|                  |   DataSN = 2, F=1    |                      |
+------------------+----------------------+----------------------+
|                  |   <<< SCSI Response  |Send Status and Sense |
|                  |   ExpDataSN = 3      |                      |
+------------------+----------------------+----------------------+
| Command Complete |                      |                      |
+------------------+----------------------+----------------------+
```

Bidirectional DataSN Example

```
+------------------+-----------------------+----------------------+
|Initiator Function|    PDU Type           | Target Function      |
+------------------+-----------------------+----------------------+
| Command request |SCSI Command >>>        |                      |
| (Read-Write)     | Read-Write            |                      |
+------------------+-----------------------+----------------------+
|                  |                       | Process old commands |
+------------------+-----------------------+----------------------+
|                  |   <<< R2T             | Ready to process     |
|                  |   R2TSN = 0           | WRITE command        |
+------------------+-----------------------+----------------------+
| * Receive Data   |   <<< SCSI Data-In    |   Send Data          |
|                  |   DataSN = 1, F=0     |                      |
+------------------+-----------------------+----------------------+
| * Receive Data   |   <<< SCSI Data-In    |   Send Data          |
|                  |   DataSN = 2, F=1     |                      |
+------------------+-----------------------+----------------------+
| * Send Data      |   SCSI Data-Out >>>   |   Receive Data       |
| for R2TSN 0      |   DataSN = 0, F=1     |                      |
+------------------+-----------------------+----------------------+
|                  |   <<< SCSI Response   |Send Status and Sense |
|                  |   ExpDataSN = 3       |                      |
+------------------+-----------------------+----------------------+
| Command Complete |                       |                      |
+------------------+-----------------------+----------------------+
```

*) Send data and Receive Data may be transferred simultaneously as in an atomic Read-Old-Write-New or sequentially as in an atomic Read-Update-Write (in the latter case the R2T may follow the received data).

Unsolicited and immediate output (write) data with DataSN Example

```
+------------------+-----------------------+----------------------+
|Initiator Function|    PDU Type & Content |  Target Function     |
+------------------+-----------------------+----------------------+
|  Command request |SCSI Command (WRITE)>>>| Receive command      |
|  (write)         |F=0                    | and data             |
|+ Immediate data  |                       | and queue it         |
+------------------+-----------------------+----------------------+
| Send Unsolicited |   SCSI Write Data >>> | Receive more Data    |
|  Data            |   DataSN = 0, F=1     |                      |
+------------------+-----------------------+----------------------+
|                  |                       | Process old commands |
+------------------+-----------------------+----------------------+
|                  |   <<< R2T             | Ready for more data  |
|                  |   R2TSN = 0           |                      |
+------------------+-----------------------+----------------------+
|  Send Data       |   SCSI Write Data >>> |   Receive Data       |
|  for R2TSN 0     |   DataSN = 0, F=1     |                      |
+------------------+-----------------------+----------------------+
|                  |   <<< SCSI Response   |Send Status and Sense |
|                  |                       |                      |
+------------------+-----------------------+----------------------+
| Command Complete |                       |                      |
+------------------+-----------------------+----------------------+
```

B.4. CRC Examples

N.B. all Values are Hexadecimal

32 bytes of zeroes:

```
  Byte:        0  1  2  3

     0:       00 00 00 00
   ...
    28:       00 00 00 00

   CRC:       aa 36 91 8a
```

32 bytes of ones:

```
  Byte:        0  1  2  3

     0:       ff ff ff ff
   ...
    28:       ff ff ff ff
```


Docker EX1014
Page 217 of 257

```
      CRC:       43 ab a8 62

   32 bytes of incrementing 00..1f:

     Byte:        0  1  2  3

        0:       00 01 02 03
      ...
       28:       1c 1d 1e 1f

      CRC:       4e 79 dd 46

   32 bytes of decrementing 1f..00:

     Byte:        0  1  2  3

        0:       1f 1e 1d 1c
      ...
       28:       03 02 01 00

      CRC:       5c db 3f 11

   An iSCSI - SCSI Read (10) Command PDU

    Byte:        0  1  2  3

       0:       01 c0 00 00
       4:       00 00 00 00
       8:       00 00 00 00
      12:       00 00 00 00
      16:       14 00 00 00
      20:       00 00 04 00
      24:       00 00 00 14
      28:       00 00 00 18
      32:       28 00 00 00
      36:       00 00 00 00
      40:       02 00 00 00
      44:       00 00 00 00

     CRC:       56 3a 96 d9
```

Appendix C. Login Phase Examples

In the first example, the initiator and target authenticate each other via Kerberos:

```
I-> Login (CSG,NSG=0,1 T=1)
    InitiatorName=iqn.1999-07.com.os:hostid.77
    TargetName=iqn.1999-07.com.example:diskarray.sn.88
    AuthMethod=KRB5,SRP,None

T-> Login (CSG,NSG=0,0 T=0)
    AuthMethod=KRB5

I-> Login (CSG,NSG=0,1 T=1)
    KRB_AP_REQ=<krb_ap_req>

(krb_ap_req contains the Kerberos V5 ticket and authenticator
   with MUTUAL-REQUIRED set in the ap-options field)

If the authentication is successful, the target proceeds with:

T-> Login (CSG,NSG=0,1 T=1)
    KRB_AP_REP=<krb_ap_rep>

(krb_ap_rep is the Kerberos V5 mutual authentication reply)

If the authentication is successful, the initiator may proceed
   with:

I-> Login (CSG,NSG=1,0 T=0) FirstBurstLength=8192
T-> Login (CSG,NSG=1,0 T=0) FirstBurstLength=4096
     MaxBurstLength=8192
I-> Login (CSG,NSG=1,0 T=0) MaxBurstLength=8192
    ... more iSCSI Operational Parameters

T-> Login (CSG,NSG=1,0 T=0)
    ... more iSCSI Operational Parameters

And at the end:

I-> Login (CSG,NSG=1,3 T=1)
    optional iSCSI parameters

T-> Login (CSG,NSG=1,3 T=1) "login accept"
```

If the initiator's authentication by the target is not successful, the target responds with:

```
T-> Login "login reject"
```

instead of the Login KRB_AP_REP message, and terminates the connection.

If the target's authentication by the initiator is not successful, the initiator terminates the connection (without responding to the Login KRB_AP_REP message).

In the next example only the initiator is authenticated by the target via Kerberos:

```
I-> Login (CSG,NSG=0,1 T=1)
   InitiatorName=iqn.1999-07.com.os:hostid.77
   TargetName=iqn.1999-07.com.example:diskarray.sn.88
   AuthMethod=SRP,KRB5,None

T-> Login-PR (CSG,NSG=0,0 T=0)
   AuthMethod=KRB5

I-> Login (CSG,NSG=0,1 T=1)
    KRB_AP_REQ=krb_ap_req
```

(MUTUAL-REQUIRED not set in the ap-options field of krb_ap_req)

If the authentication is successful, the target proceeds with:

```
T-> Login (CSG,NSG=0,1 T=1)

I-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters

T-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters

. . .

T-> Login (CSG,NSG=1,3 T=1)"login accept"
```

In the next example, the initiator and target authenticate each other via SPKM1:

```
I-> Login (CSG,NSG=0,1 T=1)
    InitiatorName=iqn.1999-07.com.os:hostid.77
    TargetName=iqn.1999-07.com.example:diskarray.sn.88
    AuthMethod=SPKM1,KRB5,None

T-> Login (CSG,NSG=0,0 T=0)
    AuthMethod=SPKM1

I-> Login (CSG,NSG=0,0 T=0)
    SPKM_REQ=<spkm-req>

(spkm-req is the SPKM-REQ token with the mutual-state bit in the
  options field of the REQ-TOKEN set)

T-> Login (CSG,NSG=0,0 T=0)
    SPKM_REP_TI=<spkm-rep-ti>

If the authentication is successful, the initiator proceeds:

I-> Login (CSG,NSG=0,1 T=1)
    SPKM_REP_IT=<spkm-rep-it>

If the authentication is successful, the target proceeds with:

T-> Login (CSG,NSG=0,1 T=1)

The initiator may proceed:

I-> Login  (CSG,NSG=1,0 T=0) ... iSCSI parameters
T-> Login  (CSG,NSG=1,0 T=0) ... iSCSI parameters

And at the end:

I-> Login  (CSG,NSG=1,3 T=1)
    optional iSCSI parameters

T-> Login (CSG,NSG=1,3 T=1) "login accept"


If the target’s authentication by the initiator is not
     successful, the initiator terminates the connection (without
     responding to the Login SPKM_REP_TI message).
```

If the initiator's authentication by the target is not successful, the target responds with:

```
T-> Login "login reject"
```

instead of the Login "proceed and change stage" message, and terminates the connection.

In the next example, the initiator and target authenticate each other via SPKM2:

```
I-> Login (CSG,NSG=0,0 T=0)
    InitiatorName=iqn.1999-07.com.os:hostid.77
    TargetName=iqn.1999-07.com.example:diskarray.sn.88
          AuthMethod=SPKM1,SPKM2

T-> Login-PR (CSG,NSG=0,0 T=0)
    AuthMethod=SPKM2

I-> Login (CSG,NSG=0,1 T=1)
    SPKM_REQ=<spkm-req>
```

(spkm-req is the SPKM-REQ token with the mutual-state bit in the options field of the REQ-TOKEN not set)

If the authentication is successful, the target proceeds with:

```
T-> Login (CSG,NSG=0,1 T=1)
```

The initiator may proceed:

```
I-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters

T-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters
```

And at the end:

```
I-> Login  (CSG,NSG=1,3 T=1)
    optional iSCSI parameters

T-> Login (CSG,NSG=1,3 T=1) "login accept"
```


Docker EX1014
Page 222 of 257

In the next example, the initiator and target authenticate each other via SRP:

```
I-> Login (CSG,NSG=0,1 T=1)
    InitiatorName=iqn.1999-07.com.os:hostid.77
    TargetName=iqn.1999-07.com.example:diskarray.sn.88
    AuthMethod=KRB5,SRP,None

T-> Login-PR (CSG,NSG=0,0 T=0)
    AuthMethod=SRP

I-> Login (CSG,NSG=0,0 T=0)
    SRP_U=<user>
    TargetAuth=Yes

T-> Login (CSG,NSG=0,0 T=0)
    SRP_GROUP=SRP-1536,SRP-1024
    SRP_s=<s>

I-> Login (CSG,NSG=0,0 T=0)
    SRP_GROUP=SRP-1536
    SRP_A=<A>

T-> Login (CSG,NSG=0,0 T=0)
    SRP_B=<B>

I-> Login (CSG,NSG=0,1 T=1)
    SRP_M=<M>
```

If the initiator authentication is successful, the target proceeds:

```
T-> Login (CSG,NSG=0,1 T=1)
    SRP_HM=<H(A | M | K)>
```

Where N, g, s, A, B, M, and H(A | M | K) are defined in [RFC2945].

If the target authentication is not successful, the initiator terminates the connection; otherwise, it proceeds.

```
I-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters

T-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters
```

And at the end:

```
I-> Login (CSG,NSG=1,3 T=1)
    optional iSCSI parameters

T-> Login  (CSG,NSG=1,3 T=1) "login accept"
```

If the initiator authentication is not successful, the target responds with:

```
T-> Login "login reject"
```

Instead of the T-> Login SRP_HM=<H(A | M | K)> message and terminates the connection.

In the next example, the initiator and target authenticate each other via SRP:

```
I-> Login (CSG,NSG=0,1 T=1)
    InitiatorName=iqn.1999-07.com.os:hostid.77
    TargetName=iqn.1999-07.com.example:diskarray.sn.88
    AuthMethod=KRB5,SRP,None

T-> Login-PR (CSG,NSG=0,0 T=0)
    AuthMethod=SRP

I-> Login (CSG,NSG=0,0 T=0)
    SRP_U=<user>
    TargetAuth=No

 T-> Login (CSG,NSG=0,0 T=0)
     SRP_GROUP=SRP-1536
     SRP_s=<s>

I-> Login (CSG,NSG=0,0 T=0)
    SRP_GROUP=SRP-1536
    SRP_A=<A>

T-> Login (CSG,NSG=0,0 T=0)
    SRP_B=<B>

I-> Login (CSG,NSG=0,1 T=1)
    SRP_M=<M>
```

If the initiator authentication is successful, the target proceeds:

```
T-> Login (CSG,NSG=0,1 T=1)
```

```
I-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters

T-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters

And at the end:

I-> Login (CSG,NSG=1,3 T=1)
    optional iSCSI parameters

T-> Login (CSG,NSG=1,3 T=1) "login accept"
```

In the next example the initiator and target authenticate each other via CHAP:

```
I-> Login (CSG,NSG=0,0 T=0)
    InitiatorName=iqn.1999-07.com.os:hostid.77
    TargetName=iqn.1999-07.com.example:diskarray.sn.88
    AuthMethod=KRB5,CHAP,None

T-> Login-PR (CSG,NSG=0,0 T=0)
    AuthMethod=CHAP

I-> Login (CSG,NSG=0,0 T=0)
    CHAP_A=<A1,A2>

T-> Login (CSG,NSG=0,0 T=0)
    CHAP_A=<A1>
    CHAP_I=<I>
    CHAP_C=<C>

I-> Login (CSG,NSG=0,1 T=1)
    CHAP_N=<N>
    CHAP_R=<R>
    CHAP_I=<I>
    CHAP_C=<C>
```

If the initiator authentication is successful, the target proceeds:

```
T-> Login (CSG,NSG=0,1 T=1)
    CHAP_N=<N>
    CHAP_R=<R>
```

If the target authentication is not successful, the initiator aborts the connection; otherwise, it proceeds.

```
I-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters
T-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters
```

And at the end:

```
I-> Login (CSG,NSG=1,3 T=1)
    optional iSCSI parameters

T-> Login (CSG,NSG=1,3 T=1) "login accept"
```

If the initiator authentication is not successful, the target responds with:

```
T-> Login "login reject"
```

Instead of the Login CHAP_R=<response> "proceed and change stage" message and terminates the connection.

In the next example, only the initiator is authenticated by the target via CHAP:

```
I-> Login (CSG,NSG=0,1 T=0)
    InitiatorName=iqn.1999-07.com.os:hostid.77
    TargetName=iqn.1999-07.com.example:diskarray.sn.88
    AuthMethod=KRB5,CHAP,None

T-> Login-PR (CSG,NSG=0,0 T=0)
    AuthMethod=CHAP

I-> Login (CSG,NSG=0,0 T=0)
    CHAP_A=<A1,A2>
```

```
T-> Login (CSG,NSG=0,0 T=0)
    CHAP_A=<A1>
    CHAP_I=<I>
    CHAP_C=<C>

I-> Login (CSG,NSG=0,1 T=1)
    CHAP_N=<N>
    CHAP_R=<R>
```

If the initiator authentication is successful, the target proceeds:

```
T-> Login (CSG,NSG=0,1 T=1)

I-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters

T-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters
```

And at the end:

```
I-> Login (CSG,NSG=1,3 T=1)
    optional iSCSI parameters

T-> Login (CSG,NSG=1,3 T=1) "login accept"
```

In the next example, the initiator does not offer any security parameters. It therefore may offer iSCSI parameters on the Login PDU with the T bit set to 1, and the target may respond with a final Login Response PDU immediately:

```
I-> Login (CSG,NSG=1,3 T=1)
    InitiatorName=iqn.1999-07.com.os:hostid.77
    TargetName=iqn.1999-07.com.example:diskarray.sn.88
    ... iSCSI parameters

T-> Login (CSG,NSG=1,3 T=1) "login accept"
    ... ISCSI parameters
```

In the next example, the initiator does offer security parameters on the Login PDU, but the target does not choose any (i.e., chooses the "None" values):

```
I-> Login (CSG,NSG=0,1 T=1)
    InitiatorName=iqn.1999-07.com.os:hostid.77
    TargetName=iqn.1999-07.com.example:diskarray.sn.88
    AuthMethod=KRB5,SRP,None
```

```
T-> Login-PR (CSG,NSG=0,1 T=1)
    AuthMethod=None

I-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters

T-> Login (CSG,NSG=1,0 T=0)
    ... iSCSI parameters

And at the end:

I-> Login (CSG,NSG=1,3 T=1)
    optional iSCSI parameters

T-> Login (CSG,NSG=1,3 T=1) "login accept"
```

Appendix D. SendTargets Operation

To reduce the amount of configuration required on an initiator, iSCSI provides the SendTargets text request. The initiator uses the SendTargets request to get a list of targets to which it may have access, as well as the list of addresses (IP address and TCP port) on which these targets may be accessed.

To make use of SendTargets, an initiator must first establish one of two types of sessions. If the initiator establishes the session using the key "SessionType=Discovery", the session is a discovery session, and a target name does not need to be specified. Otherwise, the session is a normal, operational session. The SendTargets command MUST only be sent during the Full Feature Phase of a normal or discovery session.

A system that contains targets MUST support discovery sessions on each of its iSCSI IP address-port pairs, and MUST support the SendTargets command on the discovery session. In a discovery session, a target MUST return all path information (target name and IP address-port pairs and portal group tags) for the targets on the target network entity which the requesting initiator is authorized to access.

A target MUST support the SendTargets command on operational sessions; these will only return path information about the target to which the session is connected, and do not need to return information about other target names that may be defined in the responding system.

An initiator MAY make use of the SendTargets as it sees fit.

A SendTargets command consists of a single Text request PDU. This PDU contains exactly one text key and value. The text key MUST be SendTargets. The expected response depends upon the value, as well as whether the session is a discovery or operational session.

The value must be one of:

All

The initiator is requesting that information on all relevant targets known to the implementation be returned. This value MUST be supported on a discovery session, and MUST NOT be supported on an operational session.

Docker EX1014
Page 229 of 257

<iSCSI-target-name>

If an iSCSI target name is specified, the session should respond with addresses for only the named target, if possible. This value MUST be supported on discovery sessions. A discovery session MUST be capable of returning addresses for those targets that would have been returned had value=All had been designated.

<nothing>

The session should only respond with addresses for the target to which the session is logged in. This MUST be supported on operational sessions, and MUST NOT return targets other than the one to which the session is logged in.

The response to this command is a text response that contains a list of zero or more targets and, optionally, their addresses. Each target is returned as a target record. A target record begins with the TargetName text key, followed by a list of TargetAddress text keys, and bounded by the end of the text response or the next TargetName key, which begins a new record. No text keys other than TargetName and TargetAddress are permitted within a SendTargets response.

For the format of the TargetName, see Section 12.4 TargetName.

In a discovery session, a target MAY respond to a SendTargets request with its complete list of targets, or with a list of targets that is based on the name of the initiator logged in to the session.

A SendTargets response MUST NOT contain target names if there are no targets for the requesting initiator to access.

Each target record returned includes zero or more TargetAddress fields.

Each target record starts with one text key of the form:

TargetName=<target-name-goes-here>

Followed by zero or more address keys of the form:

TargetAddress=<hostname-or-ipaddress>[:<tcp-port>],
<portal-group-tag>

The hostname-or-ipaddress contains a domain name, IPv4 address, or IPv6 address, as specified for the TargetAddress key.

A hostname-or-ipaddress duplicated in TargetAddress responses for a given node (the port is absent or equal) would probably indicate that multiple address families are in use at once (IPV6 and IPV4).

Each TargetAddress belongs to a portal group, identified by its numeric portal group tag (as in Section 12.9 TargetPortalGroupTag). The iSCSI target name, together with this tag, constitutes the SCSI port identifier; the tag only needs to be unique within a given target's name list of addresses.

Multiple-connection sessions can span iSCSI addresses that belong to the same portal group.

Multiple-connection sessions cannot span iSCSI addresses that belong to different portal groups.

If a SendTargets response reports an iSCSI address for a target, it SHOULD also report all other addresses in its portal group in the same response.

A SendTargets text response can be longer than a single Text Response PDU, and makes use of the long text responses as specified.

After obtaining a list of targets from the discovery target session, an iSCSI initiator may initiate new sessions to log in to the discovered targets for full operation. The initiator MAY keep the discovery session open, and MAY send subsequent SendTargets commands to discover new targets.

Examples:

This example is the SendTargets response from a single target that has no other interface ports.

Initiator sends text request that contains:

```
SendTargets=All
```

Target sends a text response that contains:

```
TargetName=iqn.1993-11.com.example:diskarray.sn.8675309
```

All the target had to return in the simple case was the target name. It is assumed by the initiator that the IP address and TCP port for this target are the same as used on the current connection to the default iSCSI target.

The next example has two internal iSCSI targets, each accessible via two different ports with different IP addresses. The following is the text response:

```
TargetName=iqn.1993-11.com.example:diskarray.sn.8675309
TargetAddress=10.1.0.45:3000,1 TargetAddress=10.1.1.45:3000,2
TargetName=iqn.1993-11.com.example:diskarray.sn.1234567
TargetAddress=10.1.0.45:3000,1 TargetAddress=10.1.1.45:3000,2
```

Both targets share both addresses; the multiple addresses are likely used to provide multi-path support. The initiator may connect to either target name on either address. Each of the addresses has its own portal group tag; they do not support spanning multiple-connection sessions with each other. Keep in mind that the portal group tags for the two named targets are independent of one another; portal group "1" on the first target is not necessarily the same as portal group "1" on the second target.

In the above example, a DNS host name or an IPv6 address could have been returned instead of an IPv4 address.

The next text response shows a target that supports spanning sessions across multiple addresses, and further illustrates the use of the portal group tags:

```
 TargetName=iqn.1993-11.com.example:diskarray.sn.8675309

TargetAddress=10.1.0.45:3000,1 TargetAddress=10.1.1.46:3000,1
TargetAddress=10.1.0.47:3000,2 TargetAddress=10.1.1.48:3000,2
TargetAddress=10.1.1.49:3000,3
```

In this example, any of the target addresses can be used to reach the same target. A single-connection session can be established to any of these TCP addresses. A multiple-connection session could span addresses .45 and .46 or .47 and .48, but cannot span any other combination. A TargetAddress with its own tag (.49) cannot be combined with any other address within the same session.

This SendTargets response does not indicate whether .49 supports multiple connections per session; it is communicated via the MaxConnections text key upon login to the target.

Docker EX1014
Page 232 of 257

Appendix E. Algorithmic Presentation of Error Recovery Classes

This appendix illustrates the error recovery classes using a pseudo-programming-language. The procedure names are chosen to be obvious to most implementers. Each of the recovery classes described has initiator procedures as well as target procedures. These algorithms focus on outlining the mechanics of error recovery classes, and do not exhaustively describe all other aspects/cases. Examples of this approach are:

- Handling for only certain Opcode types is shown.
- Only certain reason codes (e.g., Recovery in Logout command) are outlined.
- Resultant cases, such as recovery of Synchronization on a header digest error are considered out-of-scope in these algorithms. In this particular example, a header digest error may lead to connection recovery if some type of sync and steering layer is not implemented.

These algorithms strive to convey the iSCSI error recovery concepts in the simplest terms, and are not designed to be optimal.

E.1. General Data Structure and Procedure Description

This section defines the procedures and data structures that are commonly used by all the error recovery algorithms. The structures may not be the exhaustive representations of what is required for a typical implementation.

```
Data structure definitions -
struct TransferContext {
        int TargetTransferTag;
        int ExpectedDataSN;
};

struct TCB {              /* task control block */
        Boolean SoFarInOrder;
        int ExpectedDataSN; /* used for both R2Ts, and Data */
        int MissingDataSNList[MaxMissingDPDU];
        Boolean FbitReceived;
        Boolean StatusXferd;
        Boolean CurrentlyAllegiant;
        int ActiveR2Ts;
        int Response;
        char *Reason;
```


Docker EX1014
Page 233 of 257

```
            struct TransferContext
                        TransferContextList[MaxOutStandingR2T];
            int InitiatorTaskTag;
            int CmdSN;

            int SNACK_Tag;

    };

    struct Connection {
            struct Session SessionReference;
            Boolean SoFarInOrder;
            int CID;
            int State;

            int CurrentTimeout;
            int ExpectedStatSN;
            int MissingStatSNList[MaxMissingSPDU];
            Boolean PerformConnectionCleanup;
    };

    struct Session {
            int NumConnections;
            int CmdSN;
            int Maxconnections;
            int ErrorRecoveryLevel;
            struct iSCSIEndpoint OtherEndInfo;
            struct Connection ConnectionList[MaxSupportedConns];
    };

    Procedure descriptions -
    Receive-a-In-PDU(transport connection, inbound PDU);
    check-basic-validity(inbound PDU);
    Start-Timer(timeout handler, argument, timeout value);
    Build-And-Send-Reject(transport connection, bad PDU, reason code);
```

E.2. Within-command Error Recovery Algorithms

E.2.1. Procedure Descriptions

```
    Recover-Data-if-Possible(last required DataSN, task control
    block);
    Build-And-Send-DSnack(task control block);
    Build-And-Send-RDSnack(task control block);
    Build-And-Send-Abort(task control block);
    SCSI-Task-Completion(task control block);
    Build-And-Send-A-Data-Burst(transport connection, data-descriptor,
                                                 task control block);
```

```
Build-And-Send-R2T(transport connection, data-descriptor,
                                          task control block);
Build-And-Send-Status(transport connection, task control block);
Transfer-Context-Timeout-Handler(transfer context);
```

Notes:

- One procedure used in this section: Handle-Status-SNACK-request is defined in Within-connection recovery algorithms.

- The Response processing pseudo-code, shown in the target algorithms, applies to all solicited PDUs that carry StatSN - SCSI Response, Text Response etc.

E.2.2. Initiator Algorithms

```
Recover-Data-if-Possible(LastRequiredDataSN, TCB)
{
  if (operational ErrorRecoveryLevel > 0) {
       if (# of missing PDUs is trackable) {
             Note the missing DataSNs in TCB.
             if (the task spanned a change in
                       MaxRecvDataSegmentLength) {
                  if (TCB.StatusXferd is TRUE)
                     drop the status PDU;
                  Build-And-Send-RDSnack(TCB);
             } else {
                  Build-And-Send-DSnack(TCB);
             }
       } else {
           TCB.Reason = "Protocol service CRC error";
           }
  } else {
        TCB.Reason = "Protocol service CRC error";
  }
  if (TCB.Reason == "Protocol service CRC error") {
        Clear the missing PDU list in the TCB.
        if (TCB.StatusXferd is not TRUE)
           Build-And-Send-Abort(TCB);
  }
}

Receive-a-In-PDU(Connection, CurrentPDU)
{
  check-basic-validity(CurrentPDU);
  if (Header-Digest-Bad) discard, return;
  Retrieve TCB for CurrentPDU.InitiatorTaskTag.
```

```
  if ((CurrentPDU.type == Data)
             or (CurrentPDU.type = R2T)) {
     if (Data-Digest-Bad for Data) {
           send-data-SNACK = TRUE;
       LastRequiredDataSN = CurrentPDU.DataSN;
         } else {
           if (TCB.SoFarInOrder = TRUE) {
               if (current DataSN is expected) {
                    Increment TCB.ExpectedDataSN.
               } else {

                    TCB.SoFarInOrder = FALSE;
                    send-data-SNACK = TRUE;
                  }
           } else {
                  if (current DataSN was considered missing) {
                      remove current DataSN from missing PDU list.
                  } else if (current DataSN is higher than expected)
{
                        send-data-SNACK = TRUE;
                  } else {
                        discard, return;
                  }
                  Adjust TCB.ExpectedDataSN if appropriate.
           }
           LastRequiredDataSN = CurrentPDU.DataSN - 1;
        }
        if (send-data-SNACK is TRUE and
               task is not already considered failed) {
           Recover-Data-if-Possible(LastRequiredDataSN, TCB);
     }
        if (missing data PDU list is empty) {
           TCB.SoFarInOrder = TRUE;
        }
     if (CurrentPDU.type == R2T) {
        Increment ActiveR2Ts for this task.

        Create a data-descriptor for the data burst.
        Build-And-Send-A-Data-Burst(Connection, data-descriptor,

                                                   TCB);
     }
  } else if (CurrentPDU.type == Response) {
     if (Data-Digest-Bad) {
           send-status-SNACK = TRUE;
        } else {
        TCB.StatusXferd = TRUE;
        Store the status information in TCB.
```

```
        if (ExpDataSN does not match) {
             TCB.SoFarInOrder = FALSE;
             Recover-Data-if-Possible(current DataSN, TCB);
        }
           if (missing data PDU list is empty) {
                TCB.SoFarInOrder = TRUE;
           }
     }
  } else { /* REST UNRELATED TO WITHIN-COMMAND-RECOVERY, NOT
              SHOWN */
  }
  if ((TCB.SoFarInOrder == TRUE) and
                       (TCB.StatusXferd == TRUE)) {
     SCSI-Task-Completion(TCB);
  }
}
```

E.2.3. Target Algorithms

```
Receive-a-In-PDU(Connection, CurrentPDU)
{
  check-basic-validity(CurrentPDU);
  if (Header-Digest-Bad) discard, return;
  Retrieve TCB for CurrentPDU.InitiatorTaskTag.
  if (CurrentPDU.type == Data) {
      Retrieve TContext from CurrentPDU.TargetTransferTag;
      if (Data-Digest-Bad) {
            Build-And-Send-Reject(Connection, CurrentPDU,
                                 Payload-Digest-Error);
         Note the missing data PDUs in MissingDataRange[].
            send-recovery-R2T = TRUE;
         } else {
         if (current DataSN is not expected) {
             Note the missing data PDUs in MissingDataRange[].
                send-recovery-R2T = TRUE;
            }
         if (CurrentPDU.Fbit == TRUE) {
             if (current PDU is solicited) {
                    Decrement TCB.ActiveR2Ts.
             }
             if ((current PDU is unsolicited and
                    data received is less than I/O length and
                      data received is less than FirstBurstLength)
                 or (current PDU is solicited and the length of
                      this burst is less than expected)) {
                 send-recovery-R2T = TRUE;
                 Note the missing data in MissingDataRange[].
             }
```

```
            }
         }
         Increment TContext.ExpectedDataSN.
      if (send-recovery-R2T is TRUE  and
                task is not already considered failed) {
         if (operational ErrorRecoveryLevel > 0) {
             Increment TCB.ActiveR2Ts.
             Create a data-descriptor for the data burst
                        from MissingDataRange.
             Build-And-Send-R2T(Connection, data-descriptor, TCB);
         } else {
              if (current PDU is the last unsolicited)
                 TCB.Reason = "Not enough unsolicited data";
              else
                  TCB.Reason = "Protocol service CRC error";
         }
      }
      if (TCB.ActiveR2Ts == 0) {
         Build-And-Send-Status(Connection, TCB);
      }
  } else if (CurrentPDU.type == SNACK) {
      snack-failure = FALSE;
      if (operational ErrorRecoveryLevel > 0) {
         if (CurrentPDU.type == Data/R2T) {
              if (the request is satisfiable) {

                 if (request for Data) {
                    Create a data-descriptor for the data burst
                        from BegRun and RunLength.
                    Build-And-Send-A-Data-Burst(Connection,

                                   data-descriptor, TCB);
                 } else { /* R2T */
                    Create a data-descriptor for the data burst
                        from BegRun and RunLength.
                    Build-And-Send-R2T(Connection, data-descriptor,
                                    TCB);
                  }
              } else {
                    snack-failure = TRUE;
              }
         } else if (CurrentPDU.type == status) {
              Handle-Status-SNACK-request(Connection, CurrentPDU);
         } else if (CurrentPDU.type == DataACK) {
              Consider all data upto CurrentPDU.BegRun as
              acknowledged.
              Free up the retransmission resources for that data.
         } else if (CurrentPDU.type == R-Data SNACK) {
```

```
                Create a data descriptor for a data burst covering
                all unacknowledged data.
             Build-And-Send-A-Data-Burst(Connection,
                                 data-descriptor, TCB);
             TCB.SNACK_Tag = CurrentPDU.SNACK_Tag;
             if (there's no more data to send) {
                Build-And-Send-Status(Connection, TCB);
             }
          }
      } else { /* operational ErrorRecoveryLevel = 0 */
             snack-failure = TRUE;

      }
      if (snack-failure == TRUE) {
          Build-And-Send-Reject(Connection, CurrentPDU,
                                                SNACK-Reject);
          if (TCB.StatusXferd != TRUE) {
             TCB.Reason = "SNACK Rejected";
             Build-And-Send-Status(Connection, TCB);
          }
      }

  } else { /* REST UNRELATED TO WITHIN-COMMAND-RECOVERY, NOT SHOWN */
  }
}

Transfer-Context-Timeout-Handler(TContext)
{
  Retrieve TCB and Connection from TContext.
  Decrement TCB.ActiveR2Ts.
  if (operational ErrorRecoveryLevel > 0 and
                task is not already considered failed) {
      Note the missing data PDUs in MissingDataRange[].
      Create a data-descriptor for the data burst
                         from MissingDataRange[].
      Build-And-Send-R2T(Connection, data-descriptor, TCB);
  } else {
      TCB.Reason = "Protocol service CRC error";
      if (TCB.ActiveR2Ts = 0) {
         Build-And-Send-Status(Connection, TCB);
      }
  }
}
```

E.3. Within-connection Recovery Algorithms

E.3.1. Procedure Descriptions

```
Procedure descriptions:
Recover-Status-if-Possible(transport connection,
                                   currently received PDU);
Evaluate-a-StatSN(transport connection, currently received PDU);
Retransmit-Command-if-Possible(transport connection, CmdSN);
Build-And-Send-SSnack(transport connection);
Build-And-Send-Command(transport connection, task control block);
Command-Acknowledge-Timeout-Handler(task control block);
Status-Expect-Timeout-Handler(transport connection);
Build-And-Send-Nop-Out(transport connection);
Handle-Status-SNACK-request(transport connection, status SNACK
PDU);
Retransmit-Status-Burst(status SNACK, task control block);
Is-Acknowledged(beginning StatSN, run length);

Implementation-specific tunables:
InitiatorProactiveSNACKEnabled
```

Notes:

- The initiator algorithms only deal with unsolicited Nop-In PDUs for generating status SNACKs. A solicited Nop-In PDU has an assigned StatSN, which, when out of order, could trigger the out of order StatSN handling in Within-command algorithms, again leading to Recover-Status-if-Possible.

- The pseudo-code shown may result in the retransmission of unacknowledged commands in more cases than necessary. This will not, however, affect the correctness of the operation because the target is required to discard the duplicate CmdSNs.

- The procedure Build-And-Send-Async is defined in the Connection recovery algorithms.

- The procedure Status-Expect-Timeout-Handler describes how initiators may proactively attempt to retrieve the Status if they so choose. This procedure is assumed to be triggered much before the standard ULP timeout.

Docker EX1014
Page 240 of 257

E.3.2. Initiator Algorithms

```
Recover-Status-if-Possible(Connection, CurrentPDU)
{
  if ((Connection.state == LOGGED_IN) and
                 connection is not already considered failed) {
     if (operational ErrorRecoveryLevel > 0) {
        if (# of missing PDUs is trackable) {
              Note the missing StatSNs in Connection
             that were not already requested with SNACK;
          Build-And-Send-SSnack(Connection);
            } else {
              Connection.PerformConnectionCleanup = TRUE;
        }
     } else {
            Connection.PerformConnectionCleanup = TRUE;
     }
     if (Connection.PerformConnectionCleanup == TRUE) {
        Start-Timer(Connection-Cleanup-Handler, Connection, 0);
         }
  }
}

Retransmit-Command-if-Possible(Connection, CmdSN)
{

  if (operational ErrorRecoveryLevel > 0) {
     Retrieve the InitiatorTaskTag, and thus TCB for the CmdSN.
     Build-And-Send-Command(Connection, TCB);
  }
}

Evaluate-a-StatSN(Connection, CurrentPDU)
{
  send-status-SNACK = FALSE;
  if (Connection.SoFarInOrder == TRUE) {
     if (current StatSN is the expected) {
          Increment Connection.ExpectedStatSN.
     } else {
              Connection.SoFarInOrder = FALSE;
              send-status-SNACK = TRUE;
         }
  } else {
     if (current StatSN was considered missing) {
          remove current StatSN from the missing list.
     } else {
              if (current StatSN is higher than expected){
                  send-status-SNACK = TRUE;
```

```
              } else {
                  send-status-SNACK = FALSE;
              discard the PDU;
          }
     }
     Adjust Connection.ExpectedStatSN if appropriate.
     if (missing StatSN list is empty) {
          Connection.SoFarInOrder = TRUE;
         }
  }
  return send-status-SNACK;
}

Receive-a-In-PDU(Connection, CurrentPDU)
{
  check-basic-validity(CurrentPDU);
  if (Header-Digest-Bad) discard, return;
  Retrieve TCB for CurrentPDU.InitiatorTaskTag.
  if (CurrentPDU.type == Nop-In) {
        if (the PDU is unsolicited) {
              if (current StatSN is not expected) {
                   Recover-Status-if-Possible(Connection,
                                CurrentPDU);
              }
              if (current ExpCmdSN is not Session.CmdSN) {
                  Retransmit-Command-if-Possible(Connection,
                                CurrentPDU.ExpCmdSN);
              }
        }
  } else if (CurrentPDU.type == Reject) {
        if (it is a data digest error on immediate data) {
              Retransmit-Command-if-Possible(Connection,
                                 CurrentPDU.BadPDUHeader.CmdSN);
        }
  } else if (CurrentPDU.type == Response) {
       send-status-SNACK = Evaluate-a-StatSN(Connection,
                                   CurrentPDU);
       if (send-status-SNACK == TRUE)
           Recover-Status-if-Possible(Connection, CurrentPDU);
  } else { /* REST UNRELATED TO WITHIN-CONNECTION-RECOVERY,
            * NOT SHOWN */
  }
}

Command-Acknowledge-Timeout-Handler(TCB)
{
  Retrieve the Connection for TCB.
  Retransmit-Command-if-Possible(Connection, TCB.CmdSN);
```

```
}

Status-Expect-Timeout-Handler(Connection)
{
  if (operational ErrorRecoveryLevel > 0) {
      Build-And-Send-Nop-Out(Connection);
  } else if (InitiatorProactiveSNACKEnabled){
      if ((Connection.state == LOGGED_IN) and
             connection is not already considered failed) {
           Build-And-Send-SSnack(Connection);
      }
  }
}
```

E.3.3. Target Algorithms

```
Handle-Status-SNACK-request(Connection, CurrentPDU)
{
  if (operational ErrorRecoveryLevel > 0) {
     if (request for an acknowledged run) {
         Build-And-Send-Reject(Connection, CurrentPDU,
                                           Protocol-Error);
     } else if (request for an untransmitted run) {
         discard, return;
     } else {
         Retransmit-Status-Burst(CurrentPDU, TCB);
     } else {
        Build-And-Send-Async(Connection, DroppedConnection,
                                 DefaultTime2Wait,
                                 DefaultTime2Retain);
  }
}
```

E.4. Connection Recovery Algorithms

E.4.1. Procedure Descriptions

```
Build-And-Send-Async(transport connection, reason code,
                                    minimum time, maximum time);
Pick-A-Logged-In-Connection(session);
Build-And-Send-Logout(transport connection, logout connection
                  identifier, reason code);
PerformImplicitLogout(transport connection, logout connection
                  identifier, target information);
PerformLogin(transport connection, target information);
CreateNewTransportConnection(target information);
Build-And-Send-Command(transport connection, task control block);
Connection-Cleanup-Handler(transport connection);
```

```
Connection-Resource-Timeout-Handler(transport connection);
Quiesce-And-Prepare-for-New-Allegiance(session, task control
block);
Build-And-Send-Logout-Response(transport connection,
                        CID of connection in recovery, reason
code);
Build-And-Send-TaskMgmt-Response(transport connection,
                      task mgmt command PDU, response code);
Establish-New-Allegiance(task control block, transport
connection);
Schedule-Command-To-Continue(task control block);
```

Notes:

- Transport exception conditions, such as unexpected connection termination, connection reset, and hung connection while the connection is in the full-feature phase, are all assumed to be asynchronously signaled to the iSCSI layer using the Transport_Exception_Handler procedure.

E.4.2. Initiator Algorithms

```
Receive-a-In-PDU(Connection, CurrentPDU) {
  check-basic-validity(CurrentPDU);
  if (Header-Digest-Bad) discard, return;

  Retrieve TCB from CurrentPDU.InitiatorTaskTag.
  if (CurrentPDU.type == Async) {
      if (CurrentPDU.AsyncEvent == ConnectionDropped) {
         Retrieve the AffectedConnection for
CurrentPDU.Parameter1.
         AffectedConnection.CurrentTimeout =
CurrentPDU.Parameter3;
         AffectedConnection.State = CLEANUP_WAIT;
         Start-Timer(Connection-Cleanup-Handler,
                     AffectedConnection,
CurrentPDU.Parameter2);
      } else if (CurrentPDU.AsyncEvent == LogoutRequest)) {
        AffectedConnection = Connection;
        AffectedConnection.State = LOGOUT_REQUESTED;
        AffectedConnection.PerformConnectionCleanup = TRUE;
        AffectedConnection.CurrentTimeout =
CurrentPDU.Parameter3;
        Start-Timer(Connection-Cleanup-Handler,
                    AffectedConnection, 0);
      } else if (CurrentPDU.AsyncEvent == SessionDropped)) {
        for (each Connection) {
            Connection.State = CLEANUP_WAIT;
            Connection.CurrentTimeout = CurrentPDU.Parameter3;
```

```
                    Start-Timer(Connection-Cleanup-Handler,
                              Connection, CurrentPDU.Parameter2);
                  }
                  Session.state = FAILED;
                }

            } else if (CurrentPDU.type == LogoutResponse) {
                Retrieve the CleanupConnection for CurrentPDU.CID.
                if (CurrentPDU.Response = failure) {
                   CleanupConnection.State = CLEANUP_WAIT;
                } else {
                    CleanupConnection.State = FREE;
                }
            } else if (CurrentPDU.type == LoginResponse) {
                 if (this is a response to an implicit Logout) {
                    Retrieve the CleanupConnection.
                    if (successful) {
                        CleanupConnection.State = FREE;
                        Connection.State = LOGGED_IN;
                    } else {
                         CleanupConnection.State = CLEANUP_WAIT;
                         DestroyTransportConnection(Connection);
                    }
                 }
            } else { /* REST UNRELATED TO CONNECTION-RECOVERY,

                      * NOT SHOWN */
            }
            if (CleanupConnection.State == FREE) {
               for (each command that was active on CleanupConnection) {
               /* Establish new connection allegiance */
                    NewConnection = Pick-A-Logged-In-Connection(Session);
                    Build-And-Send-Command(NewConnection, TCB);
                }
            } }

          Connection-Cleanup-Handler(Connection) {
            Retrieve Session from Connection.
            if (Connection can still exchange iSCSI PDUs) {
                NewConnection = Connection;
            } else {
                Start-Timer(Connection-Resource-Timeout-Handler,
                      Connection, Connection.CurrentTimeout);
                if (there are other logged-in connections) {
                     NewConnection = Pick-A-Logged-In-
          Connection(Session);
                } else {
                     NewConnection =
```

```
                    CreateTransportConnection(Session.OtherEndInfo);
                  Initiate an implicit Logout on NewConnection for
                                                  Connection.CID.
                  return;
              }
          }
          Build-And-Send-Logout(NewConnection, Connection.CID,
                                            RecoveryRemove); }

        Transport_Exception_Handler(Connection) {
          Connection.PerformConnectionCleanup = TRUE;
          if (the event is an unexpected transport disconnect) {
              Connection.State = CLEANUP_WAIT;

              Connection.CurrentTimeout = DefaultTime2Retain;
              Start-Timer(Connection-Cleanup-Handler, Connection,
                                              DefaultTime2Wait);

          } else {
              Connection.State = FREE;
          } }
```

E.4.3. Target Algorithms

```
        Receive-a-In-PDU(Connection, CurrentPDU)
        {
          check-basic-validity(CurrentPDU);
          if (Header-Digest-Bad) discard, return;
          else if (Data-Digest-Bad) {
                Build-And-Send-Reject(Connection, CurrentPDU,
                                          Payload-Digest-Error);
                discard, return;
          }
          Retrieve TCB and Session.
          if (CurrentPDU.type == Logout) {
             if (CurrentPDU.ReasonCode = RecoveryRemove) {
                 Retrieve the CleanupConnection from CurrentPDU.CID).
                 for (each command active on CleanupConnection) {
                      Quiesce-And-Prepare-for-New-Allegiance(Session,
                                                 TCB);
                      TCB.CurrentlyAllegiant = FALSE;
                 }
                 Cleanup-Connection-State(CleanupConnection);
                 if ((quiescing successful) and (cleanup successful)) {
                      Build-And-Send-Logout-Response(Connection,
                                       CleanupConnection.CID, Success);
                 } else {
                      Build-And-Send-Logout-Response(Connection,
```

```
                                        CleanupConnection.CID, Failure);
                  }
               }
            } else if ((CurrentPDU.type == Login) and
                           operational ErrorRecoveryLevel == 2) {
                  Retrieve the CleanupConnection from CurrentPDU.CID).
                  for (each command active on CleanupConnection) {
                   Quiesce-And-Prepare-for-New-Allegiance(Session, TCB);
                       TCB.CurrentlyAllegiant = FALSE;
                  }
                  Cleanup-Connection-State(CleanupConnection);
                  if ((quiescing successful) and (cleanup successful)) {
                       Continue with the rest of the Login processing;
                  } else {
                       Build-And-Send-Login-Response(Connection,
                                  CleanupConnection.CID, Target Error);
                  }
               }

            } else if (CurrentPDU.type == TaskManagement) {
                 if (CurrentPDU.function == "TaskReassign") {
                        if (Session.ErrorRecoveryLevel < 2) {
                           Build-And-Send-TaskMgmt-Response(Connection,
                                CurrentPDU, "Allegiance reassignment
                                                     not supported");
                        } else if (task is not found) {
                           Build-And-Send-TaskMgmt-Response(Connection,
                                CurrentPDU, "Task not in task set");
                        } else if (task is currently allegiant) {
                           Build-And-Send-TaskMgmt-Response(Connection,
                                     CurrentPDU, "Task still allegiant");
                        } else {
                           Establish-New-Allegiance(TCB, Connection);
                           TCB.CurrentlyAllegiant = TRUE;
                           Schedule-Command-To-Continue(TCB);
                        }
                 }
            } else { /* REST UNRELATED TO CONNECTION-RECOVERY,
                      * NOT SHOWN */
            }
          }

          Transport_Exception_Handler(Connection)
          {
            Connection.PerformConnectionCleanup = TRUE;
            if (the event is an unexpected transport disconnect) {
                Connection.State = CLEANUP_WAIT;
                Start-Timer(Connection-Resource-Timeout-Handler,
```

```
               Connection,

         (DefaultTime2Wait+DefaultTime2Retain));
                 if (this Session has full-feature phase connections
                      left)
         {
                   DifferentConnection =
                      Pick-A-Logged-In-Connection(Session);
                    Build-And-Send-Async(DifferentConnection,
                          DroppedConnection, DefaultTime2Wait,
                            DefaultTime2Retain);
              }
           } else {
               Connection.State = FREE;
           }
         }
```

Appendix F. Clearing Effects of Various Events on Targets

F.1. Clearing Effects on iSCSI Objects

The following tables describe the target behavior on receiving the events specified in the rows of the table. The second table is an extension of the first table and defines clearing actions for more objects on the same events. The legend is:

Y = Yes (cleared/discarded/reset on the event specified in the row). Unless otherwise noted, the clearing action is only applicable for the issuing initiator port.
N = No (not affected on the event specified in the row, i.e., stays at previous value).
NA = Not Applicable or Not Defined.

Docker EX1014
Page 249 of 257

	IT(1)	IC(2)	CT(5)	ST(6)	PP(7)
connection failure(8)	Y	Y	N	N	Y
connection state timeout (9)	NA	NA	Y	N	NA
session timeout/ closure/reinstatement (10)	Y	Y	Y	Y	Y(14)
session continuation (12)	NA	NA	N(11)	N	NA
successful connection close logout	Y	Y	Y	N	Y(13)
session failure (18)	Y	Y	N	N	Y
successful recovery Logout	Y	Y	N	N	Y(13)
failed Logout	Y	Y	N	N	Y
connection Login (leading)	NA	NA	NA	Y(15)	NA
connection Login (non-leading)	NA	NA	N(11)	N	Y
target cold reset(16)	Y	Y	Y	Y	Y
target warm reset(16)	Y	Y	Y	Y	Y
LU reset(19)	Y	Y	Y	Y	Y
powercycle(16)	Y	Y	Y	Y	Y

1. Incomplete TTTs - Target Transfer Tags on which the target is still expecting PDUs to be received. Examples include TTTs received via R2T, NOP-IN, etc.

2. Immediate Commands - immediate commands, but waiting for execution on a target. For example, Abort Task Set.

5. Connection Tasks - tasks that are active on the iSCSI connection in question.

6. Session Tasks - tasks that are active on the entire iSCSI session. A union of "connection tasks" on all participating connections.

7. Partial PDUs (if any) - PDUs that are partially sent and waiting for transport window credit to complete the transmission.

8. Connection failure is a connection exception condition - one of the transport connections shutdown, transport connections reset, or transport connections timed out, which abruptly terminated the iSCSI full-feature phase connection. A connection failure always takes the connection state machine to the CLEANUP_WAIT state.

9. Connection state timeout happens if a connection spends more time that agreed upon during Login negotiation in the CLEANUP_WAIT state, and this takes the connection to the FREE state (M1 transition in connection cleanup state diagram).

10. These are defined in Section 5.3.5 Session Reinstatement, Closure, and Timeout.

11. This clearing effect is "Y" only if it is a connection reinstatement and the operational ErrorRecoveryLevel is less than 2.

12. Session continuation is defined in Section 5.3.6 Session Continuation and Failure.

13. This clearing effect is only valid if the connection is being logged out on a different connection and when the connection being logged out on the target may have some partial PDUs pending to be sent. In all other cases, the effect is "NA".

14. This clearing effect is only valid for a "close the session" logout in a multi-connection session. In all other cases, the effect is "NA".

15. Only applicable if this leading connection login is a session reinstatement. If this is not the case, it is "NA".

16. This operation affects all logged-in initiators.

18. Session failure is defined in Section 5.3.6 Session Continuation and Failure.

Docker EX1014
Page 251 of 257

19. This operation affects all logged-in initiators and the clearing effects are only applicable to the LU being reset.

	DC(1)	DD(2)	SS(3)	CS(4)	DS(5)
connection failure	N	Y	N	N	N
connection state timeout	Y	NA	Y	N	NA
session timeout/ closure/reinstatement	Y	Y	Y(7)	Y	NA
session continuation	N(11)	NA*12	NA	N	NA*13
successful connection close Logout	Y	Y	Y	N	NA
session failure	N	Y	N	N	N
successful recovery Logout	Y	Y	Y	N	N
failed Logout	N	Y(9)	N	N	N
connection Login (leading	NA	NA	N(8)	N(8)	NA
connection Login (non-leading)	N(11)	NA*12	N(8)	N	NA*13
target cold reset	Y	Y	Y	Y(10)	NA
target warm reset	Y	Y	N	N	NA
LU reset	N	Y	N	N	N
powercycle	Y	Y	Y	Y(10)	NA

1. Discontiguous Commands - commands allegiant to the connection in question and waiting to be reordered in the iSCSI layer. All "Y"s in this column assume that the task causing the event (if indeed the event is the result of a task) is issued as an immediate command, because the discontiguities can be ahead of the task.

2. Discontiguous Data - data PDUs received for the task in question and waiting to be reordered due to prior discontiguities in DataSN.

3. StatSN

4. CmdSN

5. DataSN

7. It clears the StatSN on all the connections.

8. This sequence number is instantiated on this event.

9. A logout failure drives the connection state machine to the CLEANUP_WAIT state, similar to the connection failure event. Hence, it has a similar effect on this and several other protocol aspects.

10. This is cleared by virtue of the fact that all sessions with all initiators are terminated.

11. This clearing effect is "Y" if it is a connection reinstatement.

12. This clearing effect is "Y" only if it is a connection reinstatement and the operational ErrorRecoveryLevel is 2.

13. This clearing effect is "N" only if it is a connection reinstatement and the operational ErrorRecoveryLevel is 2.

F.2. Clearing Effects on SCSI Objects

The only iSCSI protocol action that can effect clearing actions on SCSI objects is the "I_T nexus loss" notification (Section 4.3.5.1 Loss of Nexus notification). [SPC3] describes the clearing effects of this notification on a variety of SCSI attributes. In addition, SCSI standards documents (such as [SAM2] and [SBC]) define additional clearing actions that may take place for several SCSI objects on SCSI events such as LU resets and power-on resets.

Since iSCSI defines a target cold reset as a protocol-equivalent to a target power-cycle, the iSCSI target cold reset must also be considered as the power-on reset event in interpreting the actions defined in the SCSI standards.

When the iSCSI session is reconstructed (between the same SCSI ports with the same nexus identifier) reestablishing the same I_T nexus, all SCSI objects that are defined to not clear on the "I_T nexus loss" notification event, such as persistent reservations, are automatically associated to this new session.

Docker EX1014
Page 253 of 257

Acknowledgements

This protocol was developed by a design team that, in addition to the authors, included Daniel Smith, Ofer Biran, Jim Hafner and John Hufferd (IBM), Mark Bakke (Cisco), Randy Haagens (HP), Matt Wakeley (Agilent, now Sierra Logic), Luciano Dalle Ore (Quantum), and Paul Von Stamwitz (Adaptec, now TrueSAN Networks).

Furthermore, a large group of people contributed to this work through their review, comments, and valuable insights. We are grateful to all of them. We especially thank those people who found the time and patience to take part in our weekly phone conferences and intermediate meetings in Almaden and Haifa, which helped shape this document: Prasenjit Sarkar, Meir Toledano, John Dowdy, Steve Legg, Alain Azagury (IBM), Dave Nagle (CMU), David Black (EMC), John Matze (Veritas - now Okapi Software), Steve DeGroote, Mark Schrandt (Cisco), Gabi Hecht (Gadzoox), Robert Snively and Brian Forbes (Brocade), Nelson Nachum (StorAge), and Uri Elzur (Broadcom). Many others helped edit and improve this document within the IPS working group. We are especially grateful to David Robinson and Raghavendra Rao (Sun), Charles Monia, Joshua Tseng (Nishan), Somesh Gupta (Silverback), Michael Krause, Pierre Labat, Santosh Rao, Matthew Burbridge, Bob Barry, Robert Elliott, Nick Martin (HP), Stephen Bailey (Sandburst), Steve Senum, Ayman Ghanem, Dave Peterson (Cisco), Barry Reinhold (Trebia Networks), Bob Russell (UNH), Eddy Quicksall (iVivity, Inc.), Bill Lynn and Michael Fischer (Adaptec), Vince Cavanna, Pat Thaler (Agilent), Jonathan Stone (Stanford), Luben Tuikov (Splentec), Paul Koning (EqualLogic), Michael Krueger (Windriver), Martins Krikis (Intel), Doug Otis (Sanlight), John Marberg (IBM), Robert Griswold and Bill Moody (Crossroads), Bill Studenmund (Wasabi Systems), Elizabeth Rodriguez (Brocade) and Yaron Klein (Sanrad). The recovery chapter was enhanced with the help of Stephen Bailey (Sandburst), Somesh Gupta (Silverback), and Venkat Rangan (Rhapsody Networks). Eddy Quicksall contributed some examples and began the Definitions section. Michael Fischer and Bob Barry started the Acronyms section. Last, but not least, we thank Ralph Weber for keeping us in line with T10 (SCSI) standardization.

We would like to thank Steve Hetzler for his unwavering support and for coming up with such a good name for the protocol, and Micky Rodeh, Jai Menon, Clod Barrera, and Andy Bechtolsheim for helping make this work happen.

In addition to this document, we recommend you acquaint yourself with the following in order to get a full understanding of the iSCSI specification: "iSCSI Naming & Discovery"[RFC3721], "Bootstrapping Clients using the iSCSI Protocol" [BOOT], "Securing Block Storage Protocols over IP" [RFC3723] documents, "iSCSI Requirements and

Docker EX1014
Page 254 of 257

Design Considerations" [RFC3347] and "SCSI Command Ordering Considerations with iSCSI" [CORD].

The "iSCSI Naming & Discovery" document is authored by:

Mark Bakke (Cisco), Jim Hafner, John Hufferd, Kaladhar Voruganti (IBM), and Marjorie Krueger (HP).

The "Bootstrapping Clients using the iSCSI Protocol" document is authored by:

Prasenjit Sarkar (IBM), Duncan Missimer (HP), and Costa Sapuntzakis (Cisco).

The "Securing Block Storage Protocols over IP" document is authored by:

Bernard Aboba (Microsoft), Joshua Tseng (Nishan), Jesse Walker (Intel), Venkat Rangan (Rhapsody Networks), and Franco Travostino (Nortel Networks).

The "iSCSI Requirements and Design Considerations" document is authored by:

Marjorie Krueger, Randy Haagens (HP), Costa Sapuntzakis, and Mark Bakke (Cisco).

The "SCSI Command Ordering Considerations with iSCSI" document is authored by:

Mallikarjun Chadalapaka, Rob Elliot (HP)

We are grateful to all of them for their good work and for helping us correlate this document with the ones they produced.

Docker EX1014
Page 255 of 257

Authors' Addresses

Julian Satran
IBM Research Laboratory in Haifa
Haifa University Campus - Mount Carmel
Haifa 31905, Israel

Phone +972.4.829.6264
EMail: Julian_Satran@il.ibm.com

Kalman Meth
IBM Research Laboratory in Haifa
Haifa University Campus - Mount Carmel
Haifa 31905, Israel

Phone +972.4.829.6341
EMail: meth@il.ibm.com

Costa Sapuntzakis
Stanford University
353 Serra Mall Dr #407
Stanford, CA 94305

Phone: +1.650.723.2458
EMail: csapuntz@alum.mit.edu

Efri Zeidner
XIV Ltd.
1 Azrieli Center,
Tel-Aviv 67021, Israel

Phone: +972.3.607.4722
EMail: efri@xiv.co.il

Mallikarjun Chadalapaka
Hewlett-Packard Company
8000 Foothills Blvd.
Roseville, CA 95747-5668, USA

Phone: +1.916.785.5621
EMail: cbm@rose.hp.com

Full Copyright Statement

Copyright (C) The Internet Society (2004). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

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.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.