IPS
Internet Draft
draft-ietf-ips-iscsi-14.txt
Category: standards-track

Julian Satran
Kalman Meth
IBM

Costa Sapuntzakis
Cisco Systems

Mallikarjun Chadalapaka
Hewlett-Packard Co.

Efri Zeidner
SANGate

iSCSI

Status of this Memo

Abstract

The Small Computer Systems Interface (SCSI) is a popular family of protocols for communicating with I/O devices, especially storage devices. This document describes a transport protocol for SCSI that works on top of TCP. The iSCSI protocol aims to be fully compliant with the rules laid out in the SCSI Architecture Model - 2 [SAM2] document. The current version of iSCSI is 0.

Acknowledgements

This protocol was developed by a design team that, beside 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), Paul Von Stamwitz (Adaptec, now TrueSAN Networks).

Also, a large group of people contributed to this work through their review, comments and valuable insights. We are grateful to all them. We are especially grateful to those who found the time and patience to take part in our weekly phone conferences and intermediate meetings in Almaden and Haifa, so helping to shape this document: Prasenjit Sarkar, Meir Toledano, John Dowdy, Steve Legg, Alain Azagury (IBM), Dave Nagle (CMU), David Black (EMC), John Matze (Veritas - now with Okapi Software), Steve DeGroote, Mark Schrandt (NuSpeed), Gabi Hecht (Gadzoox), Robert Snively and Brian Forbes (Brocade), Nelson

Nachum (StorAge), Uri Elzur (Broadcom). Many more helped clean 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 Konig (?), Michael Krueger (Windriver), Martins Krikis (Intel), Doug Otis (Sanlight), Robert Griswold and Bill Moody (Crossroads), Yaron Klein (Sanrad). The recovery chapter was enhanced with help from 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, thanks to 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, Micky Rodeh, Jai Menon, Clod Barrera and Andy Bechtolsheim for helping this work happen.

This document has to be considered together with the "Naming & Discovery"[NDT], "Boot"[BOOT] and "Securing iSCSI, iFCP and FCIP"[SEC-IPS] documents.

The "Naming & Discovery" document is authored by:

Mark Bakke (Cisco), Jim Hafner, John Hufferd, Kaladhar Voruganti (IBM), Marjorie Krueger (Hewlett-Packard).
.

The "Boot" document is authored by:

Prasenjit Sarkar (IBM), Duncan Missimer (HP) and Costa Sapuntzakis (Cisco).

The "Securing iSCSI, iFCP and FCIP" document is authored by:

Bernard Aboba(Microsoft), Joshua Tseng (Nishan), Jesse Walker (Intel), Venkat Rangan (Rhapsody Networks), Franco Travostino (Nortel Networks).

We are grateful to all them for their good work and for helping us correlate this document with the ones they produced.

Change Log

The following changes were made from draft-ietf-ips-iSCSI-13 to draft-ietf-ips-iSCSI-14:

- Text cleanup
- Clarification on COLD RESET - required by SAM
- fixed in 9.5 recommendation on empty data (was inconsistent with R2T)
- 9.4.6.2 text reffers only to firstburstsize changed error code to "incorrect amount of data"
- changed size to length everywhere
- Reinstated I bit in text request (typo)
- StatSN is retransmitted R2T should be the new value
- Fixed DefaultTime2Wait and changed selection function format in Section 11

The following changes were made from draft-ietf-ips-iSCSI-12 to draft-ietf-ips-iSCSI-13:

- Text cleanup
- Limited decimal encoding to 64 bit integers
- Logout Request reason code moved to byte 1
- Renamed MaxRecvPDULength to MaxRecvDataSegmentLength
- Large Numbers allowed only if explicitely stated
- CHAP is the mandatory to implement in-band authentication and SRP is optional
- A negotiation answer is permitted only if all key=value pairs are complete. A flag indicates completion.
- Clearing effects appendix simplified - SCSI effects are now part of [SPC3]
- Made explicit a rule a bout checking when committing a negotiation
- Added code 4 for Asynch Message - request negotiation

The following changes were made from draft-ietf-ips-iSCSI-11 to draft-ietf-ips-iSCSI-12:

- Clarify the use of A bit and DataACK at the end of data
- Clarified checking to be done for abort task and removed Referenced task tag from task management response
- Range separator is tilde.

- Fixed the paragraph numbering in the appendices.
- Clarified the expected target behavior in a lost F-bit scenario when responding to Abort Task Set/Clear Task Set.
- Added the TargetPortalGroupTag key as a Login/operational key, and its usage semantics were added to Section 4.3 Login Phase.
- Clarified the language in Section 6.1.2 Allegiance Reassignment and Section 6.2 Usage Of Reject PDU in Recovery.
- Clarified the states corresponding to full-feature phase operation in connection and session state diagrams in Chapter 5.
- Delivering all negotiated unsolicited data are mandatory
- Delivering all the data for an R2T is mandatory
- Added a timeout guidance section to Chapter 8
- Added normative naming text (previously in NDT)
- Clarified no duplicate parameter for login
- Added a minimum required to support to text length (16k/64k)
- Changed the name of TSID to TSIH to better reflect its meaning
- Security - IPsec transport mode is MAY and authentication MUST be used when encryption is used
- Added to logout a section clarifying the actions to be taken on task termination by the target
- Removed CRN
- Changed default time2wait & retain to better express typical ratio
- Changes SCSI port element separator to comma
- Async Event data format same as for SCSI response

The following changes were made from draft-ietf-ips-iSCSI-10 to draft-ietf-ips-iSCSI-11:

- ACA is SHOULD
- New format for ISID that allows factory presets
- New wording in section 9.5.4 that makes it clear that initiator must discard discontiguous data PDUs during reassignment.
- Removed Parameter1 field definition for "drop the session" Async Message.
- In state transitions chapter, added Logout timeout to the event set causing T17, and removed the "session close" event from the event set for T6. Changed "status class" to StatusClass.
- Clarified that for ErrorRecoveryLevel < 2, a restart Login PDU terminates all the tasks.
- Clarified the various subcases of interpretation for Time2Retain and Time2Wait in the Logout Response section.
- Added a new section in the recovery chapter on connection timeout management.

- The LogoutLoginMinTime and LogoutLoginMaxTime keys are respectively renamed to DefaultTime2Wait and DefaultTime2Retain, because they are used only on non-Logout events and also to better align with the notion of Time2Wait and Time2Retain that the draft already defines.
- Added the new Appendix on clearing effects.
- Retired the X-bit in Login PDU to make the bit position reserved. Moved the content under X-bit description to a new section 4.3.4 that describes "connection reinstatement".
- Added text to section 6.1.2 that clarifies the expectations on targets during allegiance reassignment.
- Minor changes in error recovery algorithms to change NextC- mdSN to CmdSN in the Session data structure.
- Added a new section 4.3.5 defining the term "session rein- statement".
- Added a new transition N11 to target session state diagram, to address the session reinstatement event. Enhancing the event set for N3(T) and N6(I & T) for the same event. Add- ing the same event to the event sets for target transitions T8, T13, T15, T16, T17, T18, and M2 (I & T).
- Addressed the case of active TTTs when ABORT TASK SET/CLEAR TASK SET is in progress in section 9.5 and section 9.6.
- Added a new Section 9.6.2 Task Management actions on task sets that describes the exact timeline of events on a task set task management function.
- Clarified the usage of ITT for DataACK type of SNACK.
- Added error code for inexistent session to login response
- Changed the FIM SHOULD to should(!)
- Added a TTT field for Data-In when A bit is 1 and to the cor- responding SNACK. To make it consistent changed slightly the layout of Data-IN, SCSI Response and SNACK.
- Clarified the use of LUN with all PDUs holding TTT
- Removed the ? value from negotiations
- Unified text negotiations (login, ffp and formats) in one chapter
- Clarified AHSLength and DataLength for all PDUs
- Clarified use of Reject
- Replaced Protocol Error with Negotiation Failure in negotia- tions
- Removed FFP command before login from Reject Causes
- Added Invalid Request During Login to Login Errors
- Added tape text
- Clarified Security Text
- Aligned marker negotiations with the overall negotiations and added numeric range to the negotiation forms
- Changed target network architecture example in Overview
- Clarified T bit use in Login Reject
- Version back to 00

The following changes were made from draft-ietf-ips-iSCSI-09 to draft-ietf-ips-iSCSI-10:

- Clarifying MaxOutstandingR2T
- Widening the scope of Reject reason code 0x09 to mean "Invalid PDU field".
- Changes in the "iSCSI connection termination" section to make the terminology usage consistent with the rest of the draft.
- Adding transition T18 in standard connection state diagram, and its description.
- Other minor wording changes in the state transitions chapter to address "session close" case and others.
- Adding a new state Q5(IN_CONTINUE) to the target session state diagram to resolve transitions N8 and N9 off Q2.
- Removed the AHS drop bit feature.
- Removed the qualifier field in Task Management Response PDU, and added a new response "Function authorization failed".
- Clarified the fate of regular SCSI reservations on a session timeout, compared to a transient session failure.
- Added wording in R2T section to address the case of receiving a smaller write data sequence than was asked for in an R2T.
- Changes and fixes in recovery algorithms to be consistent with the rest of the draft.
- Changed the "Invalid SNACK" Reject reason code to "Invalid data ACK" because the invalid SNACK is already covered under "Protocol error". Also treating DataSN and R2TSN equivalently in this case.
- Change in the SNACK section to require a Reject "Protocol error" on an invalid SNACK.
- Time2Retain 0 in Logout Response indicates connection/session can’t recover
- Coordinate DataSequenceInOrder with Error recovery level and MaxOutstandingR2T, also stating that only the last read/write sequence is recoverable under digest error recovery if DataSequenceInOrder=Yes
- Alias designation format appendix is again out(!) - T10 has decided it will go in SPC
- Task Management synchronization moved to the target (task management response given after task management action and confirmed delivery of all previous responses)
- Removed the don’t care value in numerical negotiations
- Changed Marker negotiation to allow it to be closed in one round
- Marker position is not dependent of the length of the login phase
- Statement made that reserved bits do not have to be checked at the beginning of Chapter 9
- InitialR2T, BidiInitialR2T and ImmediateData changed to LO
- I bit (equivalent) in responses made 0

- Added a "double response" version for the ? key value to
- ? value can be used only outside Login
- added :, [and] as allowed in key values
- allow 0 in LogoutLoginMax and Min
- after task reassign no SNACK mandated, the function must be performed by target with information made available by reassign
- removed the third party command section - SCSI now handles everything needed (including iSCSI aliasing)

The following changes were made from draft-ietf-ips-iSCSI-08 to draft-ietf-ips-iSCSI-09:

- Added Task management response "task management function not supported"
- Negotiation (numeric) responder driven
- Added vendor specific data to reject
- Allow logout in discovery sessions
- Variable DataPDULength - renamed MaxRecvPDULength
- Key=value pairs can span PDU boundaries
- Uniform treatment of text exchange resets
- Reintroduced DataACK as a special form of SNACK
- Extended ISID in the Login Request
- Removed 0 as a "no limit value" (residue from mode pages)
- Reintroduced LogoutLoginMinTime
- Digests moved to Operational Keys
- Removed X bit in all commands and replaced it in Login and added a cleaning rule to CmdSN numbering
- Several simplifications in state transition section - standard connection and session state diagrams are separately described for initiators and targets
- Several minor technical and language changes in the error recovery section
- Added Irrelevant to negotiations
- Clarification to logout behavior
- Clarification to command ordering
- On SCSI timeout task abort instead of session failure
- Changed version to 0x03 - ALL VERSION NUMBERS are temporary up to "Rafting" (take them with a grain of salt)

The following changes were made from draft-ietf-ips-iSCSI-07 to draft-ietf-ips-iSCSI-08:

- Clarified the use of initiator task tag with regard to the SCSI tag in Section 9.2.1.7 Initiator Task Tag
- Added a clarification to Section 2.2.2.1 Command Numbering and Acknowledging - response to a command should not precede acknowledgment.

- Added clarification to Section 9.7 SCSI Data-out & SCSI Data-in - good status in Data-In must be supported by initiators
- Clarified InitiatorName is required at login in Section 4.3.1 Login Phase Start
- Another clarification for SecurityContextComplete in Section 4.3.2 iSCSI Security Negotiation
- Added "command not supported in this session type" to reject reasons
- Discovery session implies MaxConnections = 1
- Second appearance of TargetAddress deleted
- Padding forbidden for non-end-of-sequence data PDUs
- Removed Boot and Copenhagener Session types
- Changed explanation of ExpDataSN
- Removed/corrected response 05 in Section 9.4.3 Response
- Brought Section 2.2.6 iSCSI Names in line with NDT draft
- Fixed the syntax in accordance with [RFC2372] and [RFC2373]
- Removed forgotten references to the default iSCSI target
- Counters back to Reject Response
- Clarification - SendTargets admissible only in full feature phase
- Changed name of DataOrder and DataDeliveryOrder to DataSequenceOrder and DataPDUInOrder and clarified appendix text
- Padding bytes SHOULD be sent as 0 (instead of MUST be 0)
- UA attention behavior for various resets deleted - replaced with reference to SAM2
- Removed AccessID
- OpParmReset generalized
- Clarified the definition of full-feature phase in Section 2.2.4 iSCSI Full Feature Phase
- Added new Reject reason codes, tabular listing and a pointer to Section 9.14.4 Implicit termination of tasks
- Added additional Reject usage semantics on CmdSN and DataSN to Section 9.14.4 Implicit termination of tasks
- Added a new Logout Response code for failure
- Renamed BUSY as RECOVERY_START, removed RECOVERY_DONE, and merged T11 and T14 transitions into T11-(1,2) in Section 5 State Transitions.
- Corrected initiator handling of format errors
- Clarified usage of command replay
- Removed the delivery in same order as presented from Text Response
- Clarified RefCmdSN function fro abort task
- Corrected length field for AHS of type Extended CDB
- Removed LUN from text management response
- Clarified F bit for Bidirectional commands
- Removed the Async iSCSI event "target reset"
- Removed wording in Section 9.6 Task Management Function Response linking SCSI mode pages to Async Messages
- Changed the ASC/ASCQ values to better mean "not enough unsolicited data"

- Names examples include date
- Removed references to S bit in Section 9.4 SCSI Response
- Fixed NOP to simplify and avoid it consuming CmdSN
- Fixed CRC and examples
- Added the T, CSG & NSG fields to Login Command & Response, rewrote Chapter 3, changed all examples in Appendix C. - Login Phase Examples - to fit the above changes
- Key=value confined to one response
- Add command restart/replay to task management
- Removed cryptographic digests
- Removed "proxy required" status code
- Re-named and fixed descriptions of status codes
- Re-formatted login examples for clarity
- SCSI/iSCSI parameters - fixed Section 3 SCSI Mode Parameters for iSCSI, out DataPDULength, DataSequenceOrder
- Changed all sense keys to aborted command in the table in Section 9.4.2 Status
- Rearranged requests to have all SCSI related grouped etc.
- Fixed Task Management Function Request ABORT TASK and removed the part about it in Chapter 8.
- Reintroduced aliases (the data format) in an appendix. The aliasing mechanism once part of iSCSI is part of [SPC]
- Login negotiations - using only login request response (instead of former login and text)
- F bit in login changed name to T bit
- Stated defaults for mode parameters in chapter 3
- Updated Chapter 7 to reflect the current consensus on security
- Changed all sense keys to aborted command in the table in 2.4.2
- Minor language clarifications in sections 1.2.3, 1.2.5, 1.2.6, 1.2.8.
- Added a new Reject reason code "Task in progress" and clarified language in the same section.
- Added more description to the session state transitions in Chapter 5.
- Several changes in Chapter 6 corresponding to the new task management function "reassign". Other language changes in Chapter 6 for better description. Format errors are mandated to cause session failures.
- Renamed the erstwhile error recovery levels as error recovery classes, and renamed "within-session" recovery to "connection recovery" to better reflect the mechanics.
- Added Section 6.13 Error Recovery Hierarchy to define the error recovery hierarchy.
- Modifications to error recovery algorithms in Appendix F.
- Added a new Reject reason code "Invalid SNACK", added DataSN to Reject PDU.
- Changed Section 9.17 Reject to use the "Invalid SNACK" reason code.

- Removed a Logout reason code in Section 9.14 Logout Request to be consistent with Section 9.9 Asynchronous Message.
- Collapsed the two event fields in Async Event and added vendor specific event
- Immediate data can be negotiated anytime (consistency)
- Removed replay as a protocol notion and all references to it
- SNACK RunLength 0 means all
- Cleaning the bookmark mechanism for text
- New T10 approved ASC/ASQ codes
- Added a incipient definitions section - thanks to Eddy Quicksall
- Change OpParmReset from Yes/No to default/current
- Added Base64 to encode large strings
- The 255 limit for key values is now "unless specified otherwise"
- Cleaned SNACK format
- Removed ExpR2TSN from SCSI command response it is too late
- MaxBurstSize/FirstBurstSize back as key=value
- Removed LogoutLoginMinTime (value provided in exchange)
- Clear language on component function in generating ISID/TSID
- Negotiation breaking is done through abort/reject
- Removed all iSCSI mode pages

The following changes were made from draft-ietf-ips-iSCSI-06 to draft-ietf-ips-iSCSI-07:

- Clarified the "fate" of immediate commands and resources mandated (1.2.2.1) and introduced a reject-code for rejected immediate commands
- Clarify CmdSN handling and checking order for ITT and CmdSN 1.2.2.1
- Added a statement to the effect that a receiver must be able to accept 0 length Data Segments to 2.7.6. Added also a statement to 2.2.1 that a zero-length data segment implies a zero-length digest
- SCSI MODE SELECT will not really set the parameters (will not cause an error either). The parameters will be set exclusively with text mode and can be retrieved with either text or Mode-SENSE. This enables us to disable their change after the Login negotiation. Also added to the negotiation (1.2.4) the value "?" with special meaning of enquiry
- Changed "task" to "command" wherever relevant
- EMDP usage in line with other SCSI protocols. EMDP governs how a target may request data and deliver. Similar to FCP a separate (protocol) parameter governs data PDU ordering within Sequence (DataPDUInOrder). Cleaned wording of DataOrder. Fixed final bit to define sequences in input stream.
- Added a "persistent state" part (1.2.8)

- Some Task Management commands may require authorization or may not be implemented. If not authorized they will return as if executed with a qualifier indicating "not authorized" or "not implemented" (clear LU and the resets)
- Task management commands and responses are "generalized" to all iSCSI tagged commands (they are named now Task Management command and response). Their behavior with respect to their CmdSN is clarified and mandated
- The logic to update ExpCmdSN etc. moved to 1.2.2.1
- Explicitly specified that a target can "initiate" negotiating a parameter (offering)(1.2.4)
- Returned the "direction" bit and a set of codes similar to version 05
- Introduced a "special" session type (CopyManagerSession) to be used between a Copy Manager and all of its target; it may help define authentication and limit the type f commands to be executed in such a session
- Added 8.4 - How to Abort Safely a Command that Was Not Received
- Fixed the Logout Text
- AHSLength is now the first field in the AHS
- Fixed wording in 2.35 indicating AHS is mandatory for Bidirectional commands
- All key=value responses have to be explicit (none, not-understood etc.); no more selection by hiatus
- Targets can also offer key=value pairs (i.e., initiate negotiation) stated explicitly in 2.9.3
- Logout has a CmdSN field
- The Status SNACK can be discarded if the target has no such recovery
- Some parameters have been removed and replaced by "reasonable" defaults (read arbitrary defaults!); many others can't be changed anymore while the session is in full-feature phase
- NOP-Out specifies how LUN is generated when used (copied from NOP-In)
- Initial Marker-Less Interval is not a parameter anymore
- A response with F=1 during negotiation may not contain key=value pairs that may require additional answers from the initiator
- Clarified the meaning of the F bit on Write commands with regard to immediate and unsolicited data; F bit 0 means that unsolicited data will follow while F bit 1 means that this is the last of them (if any)
- You can have both immediate and unsolicited Data-Out PDUs
- DataPDULength and FirstBurstSize of 0 are allowed and mean unlimited length
- Task management command behavior relative to their own CmdSN is now stated in no uncertain terms (they are mandated to execute as if issued at CmdSN and, in case of aborts and

clear/reset no additional response/status is expected for those commands after the task management command response
- DataSN field in R2T renamed as R2TSN (better reflects semantics) and SNACK explicitly says that it requests Data or R2T.
- A session can have only one outstanding text request (not sequence)
- Text for Login Response 0301 changed (removed the maintenance mention)
- Clarified when ExpDataSN is reserved in SCSI Response
- Clarified the text and parameter (timers) for iSCSI event
- Padding bytes should be 0 (2.1)
- TotalAHSLength in 2.1.1.1 includes padding
- DataSegmentLength in 2.1.1.2 excludes padding
- Clarified bits in AHS type
- Limit for key/value string lengths (63, 255) in 2.8.3
- Added an example of SCSI event to Asynchronous Message
- Changed "Who" to "Who can send" in appendix
- Clarified meaning of parameters on 2.18.1 - Asynchronous Message - iSCSI Event
- Clarified the required initiator behavior at logout (not sending other commands) and how one expects the TCP close to be performed in 2.14
- Added a Login Response code indicating that a session can't include a given connection (0208)
- Clarified transition to full feature phase (per session and per connection and the role of the leading connection) in 1.2.5
- Corrected "one outstanding text request per connection" instead of "per session"
- For the Login Response TSID must be valid only if Login is accepted and the F bit is 1
- Added examples illustrating DataSN and R2TSN (from Eddy Quicksall)
- Added more text to the task management command 2.5
- Removed EnableACA and its dependents (in task management) and stated the requirement for a Unit Attention conform to SAM2
- iSCSI Target Name if used on a connection other than the first must be the same as on the first (4.1)
- Fixed the examples in the Login appendix to correspond to the new keys
- Fixed SCSI Response Flags and made them consistent with the Data-In PDU
- All specified keys except X-* MUST be accepted (2.8.3)
- Hexadecimal notation is 0xab123cd (not 0x'ab123cd')
- Clarified CmdSN usage in immediate commands and the meaning of "execution engine" in 1.2.2.1
- Reject response that prevent the creation of a SCSI task or result in a SCSI task being terminated must be followed by a SCSI Response with a Check Condition status 2.19.1

- Additional Runs (AddRuns) dropped from the SNACK request (too complex). With it disappeared also the implicit acknowledgement of sequences "between runs"
- PDUs delivered because of SNACK will be exact replicas of the original PDUs (including all flags) 2.16
- Added CommandReplaySupport key to negotiate support for full command replay (a command can be replayed after the status has been issued but has not been acknowledged) and a reject cause of unsupported command reply
- Added CommandFailoverSupport key to negotiate support for command allegiance change (command retry on another connection)
- Status SNACK for an acknowledged status is a protocol error (cause for reject)
- Reject cause "Command In Progress" when requesting replay before status is issued and while command is running
- Premature SNACKs are silently discarded (2.16)
- Status SNACK has to supported only if within command or within connection recovery is supported. If within session recovery is supported SNACK can be discarded and followed by an Async. Message requesting logout
- StatSN added to Logout Response
- Added "CID not found" to Logout Response reason codes
- Async Message - iSCSI event 2 (request logout) has to be sent on the connection to be dropped. Wording fixed.
- Naming changes - iqn (stands for iSCSI qualified name) introduced as a replacement to fqn. Iqn prefixes also reversed names
- text in 8.3 revised (task management implementation mechanism)
- Fixed bit 7 byte 1 in Task Management response to 1 (consistency)
- Clarified in 1.2.2 behavior when "command window" is 0 (MaxCmdSN = ExpCmdSN -1)
- Added state transitions part (new part 6)
- Refreshed recovery chapter (new part 7)
- Added an appendix with detailed recovery mechanisms (Appendix E)
- Added session types a brief explanation in part 1
- Added DiscoverySession key and SendTargets appendix
- SCSI response made to fit having both a Status and a Response field. Needed for target errors that result in a check condition and ACA. In line with SAM2 that requires both fields (former versions where modeled on FCP).
- The security appendix list SRP as mandatory to implement
- Clarified initial CmdSN and the role of TSID as a serializer
- Long Text Responses - additional fields added to the text request and text response
- Added a SCSI to iSCSI concept mapping section 1.5

- Clarified SNACK wording to indicate that in general command. Request, iSCSI command and iSCSI command have the same meaning. Also status, response or numbered response.
- Changed InitStatSN and clarified how it increases
- Added requirement for a 0x00 delimiter after each key=value
- Added binary negotiations (Yes|No) explicitly to 1.2.4
- All keys and values in the spec are case sensitive (stated in the text request)
- Changed the "operational parameters sent before the security. MAY be discarded" into MUST be discarded
- Changed the login reject 0201 to read - Security Negotiation Failed
- Added to 2.3.1 a paragraph about mandatory consistencies
- Stated clearly that F bit pairing is "local" (per/pair) and not per negotiation
- Clarified dependent parameter status
- Added CRC Example
- Added OpParmReset=Yes
- SecurityContextComplete is mandatory if any option offered
- Added a warning about the implications of not sending all unsolicited data to part 8
- Added a recommendation to send unsolicited data at FirstBurstSize and a response (error) for targets not supporting less
- Many more minor editorial changes, clarifications, typos etc.
- Responses in same position in SCSI response, logout, task etc.

Status of this Memo 2
Abstract 2
Acknowledgements 2
Change Log 4
1. Definitions and Acronyms 24
1.1 Definitions 24
1.2 Acronyms 28
1.3 Conventions used in this document 30
1.3.1 Word Rule 30
1.3.2 Half-Word Rule 31
1.3.3 Byte Rule 31
2. Overview 32
2.1 SCSI Concepts 32
2.2 iSCSI Concepts and Functional Overview 33
2.2.1 Layers and Sessions 33
2.2.2 Ordering and iSCSI Numbering 34
2.2.2.1 Command Numbering and Acknowledging 35
2.2.2.2 Response/Status Numbering and Acknowledging 38
2.2.2.3 Data Sequencing 39
2.2.3 iSCSI Login 39
2.2.4 iSCSI Full Feature Phase 40
2.2.5 iSCSI Connection Termination 43
2.2.6 iSCSI Names 43
2.2.6.1 iSCSI Name Requirements 44
2.2.6.2 iSCSI Name Encoding 46
2.2.6.3 iSCSI Name Structure 46
2.2.6.3.1 Type "iqn." (iSCSI Qualified Name) 47
2.2.6.3.2 Type "eui." (IEEE EUI-64 format) 48
2.2.7 Persistent State 49
2.2.8 Message Synchronization and Steering 49
2.2.8.1 Rationale 49
2.2.8.2 Synchronization (sync) and Steering Functional Model 50
2.2.8.3 Sync and Steering and Other Encapsulation Layers . 52
2.2.8.4 Sync/Steering and iSCSI PDU Length 53
2.3 iSCSI Session Types 54
2.4 SCSI to iSCSI Concepts Mapping Model 54
2.4.1 iSCSI Architecture Model 55
2.4.2 SCSI Architecture Model 57
2.4.3 Consequences of the Model 59
2.4.3.1 I_T Nexus State 60
2.4.3.2 SCSI Mode Pages 60
2.5 Request/Response Summary 61
2.5.1 Request/Response types carrying SCSI payload 61

2.5.1.1 SCSI-Command 61
2.5.1.2 SCSI-Response 62
2.5.1.3 Task Management Function Request 62
2.5.1.4 Task Management Function Response 63
2.5.1.5 SCSI Data-out and SCSI Data-in 63
2.5.1.6 Ready To Transfer (R2T) 64
2.5.2 Requests/Responses carrying SCSI and iSCSI Payload 64
2.5.2.1 Asynchronous Message 64
2.5.3 Requests/Responses carrying iSCSI Only Payload 65
2.5.3.1 Text Request and Text Response 65
2.5.3.2 Login Request and Login Response 65
2.5.3.3 Logout Request and Response 66
2.5.3.4 SNACK Request 66
2.5.3.5 Reject 67
2.5.3.6 NOP-Out Request and NOP-In Response 67
3. SCSI Mode Parameters for iSCSI 68
4. Login and Full Feature Phase Negotiation 69
4.1 Text Format 69
4.2 Text Mode Negotiation 72
4.2.1 List negotiations 74
4.2.2 Simple-value negotiations 75
4.3 Login Phase 76
4.3.1 Login Phase Start 78
4.3.2 iSCSI Security Negotiation 80
4.3.3 Operational Parameter Negotiation During the Login Phase .81
4.3.4 Connection reinstatement 82
4.3.5 Session reinstatement, closure and timeout 83
4.3.5.1 Loss of Nexus notification 83
4.3.6 Session continuation and failure 84
4.4 Operational Parameter Negotiation Outside the Login Phase . . .84
5. State Transitions 86
5.1 Standard Connection State Diagrams 86
5.1.1 Standard Connection State Diagram for an Initiator 86
5.1.2 Standard Connection State Diagram for a Target 88
5.1.3 State Descriptions for Initiators and Targets 90
5.1.4 State Transition Descriptions for Initiators and Targets .91
5.2 Connection Cleanup State Diagram for Initiators and Targets . .95
5.2.1 State Descriptions for Initiators and Targets 96
5.2.2 State Transition Descriptions for Initiators and Targets .97
5.3 Session State Diagrams 98
5.3.1 Session State Diagram for a Target 99
5.3.2 State Descriptions for Initiators and Targets 101
5.3.3 State Transition Descriptions for Initiators and Targets 101

6. iSCSI Error Handling and Recovery 103
6.1 Retry and Reassign in Recovery 103
6.1.1 Usage of Retry 103
6.1.2 Allegiance Reassignment 104
6.2 Usage Of Reject PDU in Recovery 105
6.3 Connection timeout management 105
6.3.1 Timeouts on transport exception events 106
6.3.2 Timeouts on planned decommissioning 106
6.4 Format Errors 106
6.5 Digest Errors 107
6.6 Sequence Errors 108
6.7 SCSI Timeouts 109
6.8 Negotiation Failures 109
6.9 Protocol Errors 110
6.10 Connection Failures 110
6.11 Session Errors 111
6.12 Recovery Classes 112
6.12.1 Recovery Within-command 112
6.12.2 Recovery Within-connection 113
6.12.3 Connection Recovery 114
6.12.4 Session Recovery 114
6.13 Error Recovery Hierarchy 115
7. Security Considerations 118
7.1 iSCSI Security Mechanisms 118
7.2 In-band Initiator-Target Authentication 119
7.2.1 CHAP Considerations 120
7.2.2 SRP Considerations 120
7.3 IPsec 121
7.3.1 Data Integrity and Authentication 121
7.3.2 Confidentiality 121
7.3.3 Policy, Security Associations and Key Management 122
8. Notes to Implementers 124
8.1 Multiple Network Adapters 124
8.1.1 Conservative Reuse of ISIDs 124
8.1.2 iSCSI Name, ISID and TPGT Use 125
8.2 Autosense and Auto Contingent Allegiance (ACA) 127
8.3 iSCSI timeouts 127
8.4 Command Retry and Cleaning Old Command Instances 127
8.5 Synch and Steering Layer and Performance 127
8.6 Considerations for State-dependent devices 128
8.6.1 Determining the proper ErrorRecoveryLevel 128
9. iSCSI PDU Formats 130
9.1 iSCSI PDU Length and Padding 130

9.2 PDU Template, Header, and Opcodes 130
9.2.1 Basic Header Segment (BHS) 131
9.2.1.1 I 132
9.2.1.2 Opcode 132
9.2.1.3 Opcode-specific Fields 133
9.2.1.4 TotalAHSLength 133
9.2.1.5 DataSegmentLength 133
9.2.1.6 LUN 134
9.2.1.7 Initiator Task Tag 134
9.2.2 Additional Header Segment (AHS) 134
9.2.2.1 AHSType 134
9.2.2.2 AHSLength 135
9.2.2.3 Extended CDB AHS 135
9.2.2.4 Bidirectional Expected Read-Data Length AHS 135
9.2.3 Header Digest and Data Digest 136
9.2.4 Data Segment 136
9.3 SCSI Command 137
9.3.1 Flags and Task Attributes (byte 1) 137
9.3.2 CmdSN - Command Sequence Number 138
9.3.3 ExpStatSN 138
9.3.4 Expected Data Transfer Length 138
9.3.5 CDB - SCSI Command Descriptor Block 139
9.3.6 Data Segment - Command Data 139
9.4 SCSI Response 140
9.4.1 Flags (byte 1) 140
9.4.2 Status 141
9.4.3 Response 142
9.4.4 Residual Count 142
9.4.5 Bidirectional Read Residual Count 143
9.4.6 Data Segment - Sense and Response Data Segment 143
9.4.6.1 SenseLength 143
9.4.6.2 Sense Data 144
9.4.7 ExpDataSN 144
9.4.8 StatSN - Status Sequence Number 145
9.4.9 ExpCmdSN - Next Expected CmdSN from this Initiator . . . 145
9.4.10 MaxCmdSN - Maximum CmdSN from this Initiator 145
9.5 Task Management Function Request 146
9.5.1 Function 146
9.5.2 LUN 149
9.5.3 Referenced Task Tag 149
9.5.4 RefCmdSN 149
9.5.5 ExpDataSN 149
9.6 Task Management Function Response 151

9.6.1 Response 151
9.6.2 Task Management actions on task sets 153
9.7 SCSI Data-out & SCSI Data-in 154
9.7.1 F (Final) Bit 156
9.7.2 A (Acknowledge) bit 156
9.7.3 Target Transfer Tag 157
9.7.4 StatSN 157
9.7.5 DataSN 157
9.7.6 Buffer Offset 158
9.7.7 DataSegmentLength 158
9.7.8 Flags (byte 1) 158
9.8 Ready To Transfer (R2T) 160
9.8.1 R2TSN 161
9.8.2 StatSN 161
9.8.3 Desired Data Transfer Length and Buffer Offset 162
9.8.4 Target Transfer Tag 162
9.9 Asynchronous Message 163
9.9.1 AsyncEvent 164
9.9.2 AsyncVCode 165
9.9.3 Sense Data and iSCSI Event Data 165
9.9.3.1 SenseLength 166
9.10 Text Request 167
9.10.1 F (Final) Bit 168
9.10.2 C (Continue) Bit 168
9.10.3 Initiator Task Tag 168
9.10.4 Target Transfer Tag 168
9.10.5 Text 169
9.11 Text Response 171
9.11.1 F (Final) Bit 171
9.11.2 C (Continue) Bit 172
9.11.3 Initiator Task Tag 172
9.11.4 Target Transfer Tag 172
9.11.5 StatSN 173
9.11.6 Text Response Data 173
9.12 Login Request 174
9.12.1 T (Transit) Bit 175
9.12.2 C (Continue) Bit 175
9.12.3 CSG and NSG 175
9.12.4 Version-max 175
9.12.5 Version-min 175
9.12.6 ISID 176
9.12.7 TSIH 177
9.12.8 Connection ID - CID 177

9.12.9 CmdSN 178
9.12.10 ExpStatSN 178
9.12.11 Login Parameters 178
9.13 Login Response 180
9.13.1 Version-max 180
9.13.2 Version-active 181
9.13.3 TSIH 181
9.13.4 StatSN 181
9.13.5 Status-Class and Status-Detail 181
9.13.6 T (Transit) bit 184
9.13.7 C (Continue) Bit 184
9.13.8 Login Parameters 185
9.14 Logout Request 186
9.14.1 Reason Code 188
9.14.2 CID 189
9.14.3 ExpStatSN 189
9.14.4 Implicit termination of tasks 189
9.15 Logout Response 190
9.15.1 Response 190
9.15.2 Time2Wait 191
9.15.3 Time2Retain 191
9.16 SNACK Request 193
9.16.1 Type 194
9.16.2 BegRun 195
9.16.3 RunLength 195
9.17 Reject 197
9.17.1 Reason 198
9.17.2 DataSN 199
9.17.3 StatSN, ExpCmdSN and MaxCmdSN 199
9.17.4 Complete Header of Bad PDU 199
9.18 NOP-Out 200
9.18.1 Initiator Task Tag 201
9.18.2 Target Transfer Tag 201
9.18.3 Ping Data 201
9.19 NOP-In 202
9.19.1 Target Transfer Tag 203
9.19.2 StatSN 203
9.19.3 LUN 203
10. iSCSI Security Keys and Authentication Methods 204
10.1 AuthMethod 204
10.2 Kerberos 205
10.3 Simple Public-Key Mechanism (SPKM) 206
10.4 Secure Remote Password (SRP) 207

10.5 Challenge Handshake Authentication Protocol (CHAP) 208
11. Login/Text Operational Keys 210
11.1 HeaderDigest and DataDigest 210
11.2 MaxConnections 212
11.3 SendTargets 212
11.4 TargetName 212
11.5 InitiatorName 213
11.6 TargetAlias 213
11.7 InitiatorAlias 214
11.8 TargetAddress 214
11.9 TargetPortalGroupTag 215
11.10 InitialR2T 215
11.11 BidiInitialR2T 216
11.12 ImmediateData 217
11.13 MaxRecvDataSegmentLength 218
11.14 MaxBurstLength 218
11.15 FirstBurstLength 219
11.16 DefaultTime2Wait 219
11.17 DefaultTime2Retain 220
11.18 MaxOutstandingR2T 220
11.19 DataPDUInOrder 221
11.20 DataSequenceInOrder 221
11.21 ErrorRecoveryLevel 222
11.22 SessionType 222
11.23 The Vendor Specific Key Format 223
12. IANA Considerations 224
References and Bibliography 225
Authors' Addresses 227
Appendix A. Sync and Steering with Fixed Interval Markers 229
A.1 Markers At Fixed Intervals 229
A.2 Initial Marker-less Interval 230
A.3 Negotiation 230
OFMarker, IFMarker 230
OFMarkInt, IFMarkInt 231
Appendix B. Examples 233
B.2 Write Operation Example 234
B.3 R2TSN/DataSN use Examples 234
B.4 CRC Examples 238
Appendix C. Login Phase Examples 240
Appendix D. SendTargets Operation 249
Appendix E. Algorithmic Presentation of Error Recovery Classes . . 254
E.2 Within-command Error Recovery Algorithms 255
Procedure Descriptions 255

Initiator Algorithms 256
Target Algorithms 258
E.3 Within-connection Recovery Algorithms 260
Procedure Descriptions 260
Initiator Algorithms 261
Target Algorithms 264
E.4 Connection Recovery Algorithms 264
Procedure Descriptions 264
Initiator Algorithms 265
Target Algorithms 267
Appendix F. Clearing effects of various events on targets 269
F.1 Clearing effects on iSCSI objects 269
F.2 Clearing effects on SCSI objects 274
Full Copyright Statement 276

1. Definitions and Acronyms

1.1 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 delivery subsystem

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

- iSCSI Initiator Node: The "initiator".

- 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".

- 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 addresses, 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.

- 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 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.

- 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 con-

nections 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 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 simple unsigned-integer between 1 and 65535 identifies the 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.

- 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 other SCSI entities. 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 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 and its use is mandatory in the iSCSI protocol.

- SCSI Layer: This builds/receives SCSI CDBs (Command Descriptor Blocks) and relays/receives them with the remaining command execute 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 "target image".

- 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 end-point at connection establish-

ment. The TargetPortalGroupTag key may also be returned by the target as a confirmation during session establishment.

- 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 or transport. 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 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.

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

- TSIH (Target Session Identifying Handle): The TSIH is a target assigned tag for a session with a specific named initiator. The target generates it during session establishment and 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.

1.2 Acronyms

Acronym	Definition
3DES	Triple Data Encryption Standard
ACA	Auto Contingent Allegiance
AEN	Asynchronous Event Notification
AES	Advanced Encryption Standard
AH	Additional Header
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
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
ESP	Encapsulating Security Payload
EUI	Extended Unique Identifier
FFP	Full Feature Phase
FFPO	Full Feature Phase Only
Gbps	GigaBits per Second
HBA	Host Bus Adapter
HMAC	Hashed Message Authentication
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	Initiator Task 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
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

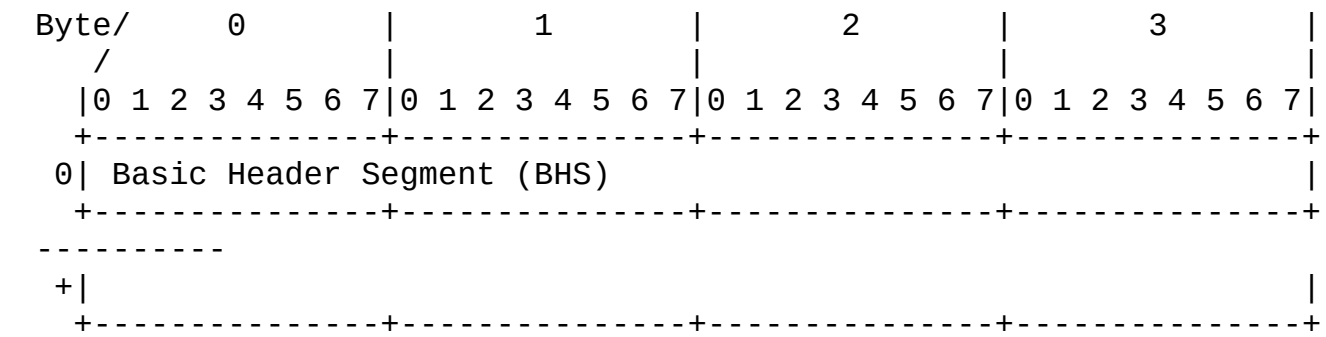
UTF Universal Transformation Format
WG Working Group

1.3 Conventions used in this document

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 RFC2119.

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



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

1.3.1 Word Rule

A word holds 4 consecutive bytes and whenever having a numeric content the word is considered an unsigned number in base 2 positional representation with the lowest numbered byte (e.g., byte 0) bit 0 representing 2**31, bit 1 representing 2**30 and 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.

1.3.2 Half-Word Rule

A half-word holds 2 consecutive bytes and whenever having a numeric content the half-word is considered an unsigned number in base 2 positional representation with the lowest numbered byte (e.g., byte 0) bit 0 representing 2**16, bit 1 representing 2**15 and through Lowest numbered byte + 1 (e.g., byte 1) bit 7 representing 2**0.

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

1.3.3 Byte Rule

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

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

2. Overview

2.1 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 service from a logical unit. 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 queue of tasks is managed by the target. 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. A response terminates a SCSI command.

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

2.2 iSCSI Concepts and Functional Overview

The iSCSI protocol is a mapping of the SCSI remote procedure invocation model (see [SAM]) 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 2.4.1 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.

2.2.1 Layers and Sessions

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

- The SCSI layer builds/receives SCSI CDBs (Command Descriptor Blocks) and relays/receives them with the remaining command execute parameters (cf. SAM2) to/from ->.

- The iSCSI layer that builds/receives iSCSI PDUs and relays/ receives them to/from one or more TCP connections that 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 2.4.2 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. Connections within a session are 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 may have to support two connections during recovery.

2.2.2 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.

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.

2.2.2.1 Command Numbering and Acknowledging

iSCSI supports ordered command delivery within a session. All commands (initiator-to-target PDUs) are numbered.

Many SCSI activities are related to a task (SAM2). The task is identified by the Initiator Task Tag for the life of the task.

Commands in transit from the initiator to the target are numbered by iSCSI; the 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 also carry CmdSN. CmdSN does not advance for commands marked for immediate delivery.

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. For this reason the task management command MUST carry the current CmdSN 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 register (see later in this section).

The means by which one may request immediate delivery for a command 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 to execution is not acknowledged through the numbering scheme. Immediate commands can be rejected by the iSCSI target due to 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.

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 handed over by the iSCSI target layer for execution as soon as detected. iSCSI may avoid delivering some commands for execution 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). Delivery for execution means delivery to the SCSI execution engine or an iSCSI-SCSI protocol specific execution engine (e.g., for text requests).

On any given 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.

The initiator and target are assumed to have the following three registers that are unique session wide and that define the numbering mechanism:

- 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 next.
- ExpCmdSN - the next expected command by the target. The target acknowledges all commands up to, but not including, this number. The initiator has to mark the acknowledged commands as such as soon as a PDU with the corresponding ExpCmdSN is received. 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.

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. Note that the CmdSN carried by immediate commands may lie outside the ExpCmdSN to MaxCmdSN range (e.g., if the initiator has previously sent a non-immediate command carrying the CmdSN equal to MaxCmdSN - i.e., 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 - what commands get 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 being that they travel 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 6.1.1 Usage of Retry). At the target, it is assumed that CmdSN is relevant only while 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) is assumed to precede any other action at the target, and is followed by ordering and delivery if no execution state is found or delivery if an execution state is found.

If an initiator issues a command retry for a command with CmdSN R on a connection when the session CmdSN register is Q, it MUST NOT advance the CmdSN past R + 2**31 -1 unless the connection is no longer operational (has returned to the FREE state - see Section 5.1 Standard Connection State Diagrams), or the connection has been reinstated (see Section 4.3.4 Connection reinstatement), or a non-immediate command with CmdSN equal or greater than Q was issued on the same connection and the reception of the command is acknowledged by the target (see Section 8.4 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 itself.

2.2.2.2 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 to enable data and status recovery after a connection failure. A target can discard all the state information maintained for recovery after the status delivery is acknowledged through ExpStatSN.

A large absolute difference between StatSN and ExpStatSN may indicate a failed connection. Initiators undertake recovery actions if the difference is greater than an implementation defined constant that SHOULD NOT exceed 2**31-1.

Initiators and Targets MUST support the response-numbering scheme.

2.2.2.3 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. The DataSN/R2TSN field enables the initiator to detect missing data or R2T PDUs.

For any given 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.

2.2.3 iSCSI Login

The purpose of the iSCSI login is to enable a TCP connection for iSCSI use, authenticate the parties, negotiate the session's parameters and mark the connection as belonging to an iSCSI session.

A session is used to identify all the connections with a given initiator that belong to the same I_T nexus to a target. (See Section 2.4.2 SCSI Architecture Model for more details on how a session relates to an I_T nexus).

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 MAY wish to authenticate each other and set a security association protocol for the session. This can occur in many different ways and is subject to negotiation.

In order to protect the TCP connection, an IPsec security association MAY be established before the Login request. Using IPsec security for iSCSI is specified in Chapter 7 and in [SEC-IPS].

The iSCSI Login Phase is carried through Login requests and responses. Once suitable authentication has occurred and operational parameters have been set, the initiator may start to send SCSI commands. How the target chooses to authorize an initiator is beyond the scope of this document. A more detailed description of the Login Phase can be found in Chapter 4.

The login PDU includes the ISID part of the session ID (SSID). The target portal group servicing the login is implied by the selection of the connection end-point. 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) (InitiatorName is described 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 2.4.1 iSCSI Architecture Model) and internally in an implementation dependent way. As ISID is used to identify a persistent state, it is subject to reuse restrictions (see Section 2.4.3 Consequences of the Model).

Before the Full Feature Phase is established, only Login Request and Login Response PDUs are allowed. Any other PDU, when received at initiator or target, is a protocol error and MUST result in the connection being terminated. Login requests and responses MUST be used exclusively during Login. On any connection the login phase MUST immediately succeed TCP connection establishment and a single Login Phase is allowed before tearing down a connection.

2.2.4 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) 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 wrapping them in iSCSI PDUs that go over the established iSCSI session.

For an iSCSI request issued over a TCP connection, the corresponding response and/or requested 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 6.1 Retry and Reassign in Recovery.

For SCSI commands that require data and/or a parameter transfer, the (optional) data and the status for the command MUST be sent over the same TCP connection to which the SCSI command is currently allegiant, illustrating the above rule.

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 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.

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 sepa-

rate iSCSI data PDUs. 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.

Targets operate in either solicited (R2T) data mode or unsolicited (non R2T) data mode. The maximum amount of unsolicited data that can be sent with a command is negotiated at login. A target MAY separately enable immediate data without enabling the more general (separate data PDUs) form of unsolicited data.

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 are meant to reduce the protocol overhead (both bandwidth and execution time).

An iSCSI initiator MAY choose to send no unsolicited data, only immediate data or FirstBurstLength bytes of unsolicited data with a command. If any non-immediate unsolicited data are sent, the total unsolicited data MUST be either the negotiated amount or all the data if the total amount is less than the negotiated amount for unsolicited data.

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 actions on receiving an R2T request that specifies data all or part outside the command bounds is unspecified.

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 data, whether immediate or as separate PDUs, than the iSCSI limit for first burst.

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); targets perform residual count calculation. Incoming data for initiators is always

implicitly solicited. SCSI data packets are matched to their corresponding SCSI commands by using Tags specified in the protocol.

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. The above mechanisms are designed to accomplish efficient data delivery and a large degree of control over the data flow.

iSCSI initiators and targets MUST also enforce some ordering rules. Unsolicited data MUST be sent on every connection in the same order in which commands were sent. A target that receives data out of order MAY terminate the session.

2.2.5 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 be initiated by either party only 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 (section 5) is required as a prelude to recovery. By doing connection cleanup before starting recovery, the initiator and target can avoid receiving stale PDUs after recovery.

2.2.6 iSCSI Names

Both targets and initiators require names for the purpose of identification, and so that iSCSI storage resources can 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 operation domain of the end user. However, because the operation domain of an IP network is potentially worldwide, the iSCSI name formats are architected to be world wide unique. To assist naming authorities in the construction of world wide unique names, iSCSI provides two name formats for different types of naming authorities.

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

Some SCSI commands require that protocol-specific identifiers be communicated within SCSI CDBs. See Section 2.4.2 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 [NDT].

2.2.6.1 iSCSI Name Requirements

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 255 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; the iSCSI Initiator Name is still required, but the iSCSI Target Name may be ignored.

iSCSI names must adhere to the following requirements:

a) iSCSI names must be globally unique. No two initiators or targets should have the same name.
b) iSCSI names must be permanent. An iSCSI initiator or target 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 must not rely on a central name broker; the naming authority must be distributed.
e) iSCSI names must support integration with existing unique naming schemes.
f) iSCSI names must rely on existing naming authorities. iSCSI does not have to create its own naming authority.

The encoding of an iSCSI name also has some requirements:

a) iSCSI names must have a single encoding method when transmitted over various protocols.
b) iSCSI names must be relatively simple to compare. The algorithm for comparing two iSCSI names for equivalence must not rely on any external server.
c) iSCSI names must be composed of displayable characters only. iSCSI names should be kept as simple as possible. They must provide for the use of international character sets, and must not be case sensitive. Whitespace characters are not allowed.
d) iSCSI names must be transport-friendly. They must 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 just paths to the same SCSI layer. In most operating systems, the named entity is the operating system image.

A target name should similarly not be tied to hardware interfaces which 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 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, independent of address or location, and that it be persistent and globally unique. Names must be

extensible, and scale with the use of naming authorities. The encoding of the name should be readable by a human, as well as be machine-readable. See [RFC1737] for further requirements.

2.2.6.2 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 contains only the following characters:
    - ASCII dash ('-'=U+002d)
    - ASCII dot ('.'=U+002e)
    - ASCII colon (':'=U+003a)
    - Any character allowed by the output of the iSCSI stringprep template (described in [STPREP-iSCSI])
- when encoded in UTF-8, it is no larger than 255 bytes

The stringprep process is described in [STPREP]; iSCSI's use of the stringprep process is described in [STPREP-iSCSI]. 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.

Note that in most cases, the Stringprep process does not need to be implemented if the names are generated using only lower-case (any character set) alpha-numeric characters.

Once iSCSI names encoded in UTF-8 are "normalized" (there is one and only one representation for each possible name), they may be safely compared byte-for-byte.

2.2.6.3 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 world wide unique names (EUI-64 format).

The type designator strings that may currently be used are:

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

As these two naming authority designators will suffice in nearly every case for both software and hardware-based entities, the creation of additional type designators is prohibited. One of these two type strings MUST be used when constructing an iSCSI name; any type string not listed here is not allowed, as they cannot be guaranteed to be unique.

2.2.6.3.1 Type "iqn." (iSCSI Qualified Name)

This iSCSI name type can be used by any organization which 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 DNS domain name. This domain name does not have to be active, and does not have to resolve to an address; it just needs to be reserved to prevent others from generating iSCSI names using the same domain name.

Because a domain name can expire, be acquired by another entity, and might 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. A date code is provided as part of the "iqn." format for this reason.

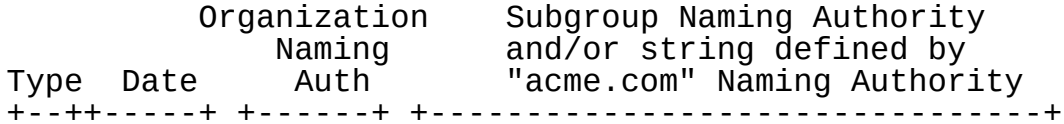
The iSCSI qualified name string consists of:

- 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 date on which the domain name was acquired by this naming authority. 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.

- Another ".".
- The reversed domain name of the naming authority (person or organization) creating this iSCSI name.
- Another ".".
- Any 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, software keys, or anything else that makes sense to uniquely identify the initiator or target. Everything after the reversed domain name, followed by another dot ".", can be assigned as desired by the owner of the domain name. It is the responsibility of the entity that is the naming authority to ensure that the iSCSI names it assigns are world wide unique. For example, "ACME Storage Arrays, Inc.", might own the domain name "acme.com".

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



iqn.2001-04.com.acme.diskarrays-sn-a8675309
iqn.2001-04.com.acme.storage:tape.sys1.xyz
iqn.2001-04.com.acme.storage.tape:sys1.xyz

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

The IEEE Registration Authority provides a service for assigning globally unique identifiers [EUI]. The EUI-64 format is in use as a global identifier in other network protocols such as Fibre Channel. See http://standards.ieee.org/regauth/oui/index.shtml - for more information on registering for EUI identifiers.

The format is "eui." followed by an EUI-64 identifier (16 ASCII-encoded hexidecimal 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 world wide unique names for its products.

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

2.2.7 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 (for iSCSI, the InitiatorName plus the ISID portion of the session identifier). (See Section 2.4.2 SCSI Architecture Model and Section 2.4.3 Consequences of the Model.)

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

All iSCSI session and connection parameters are re-initialized on 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 9.5 Task Management Function Request.)

2.2.8 Message Synchronization and Steering

2.2.8.1 Rationale

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.

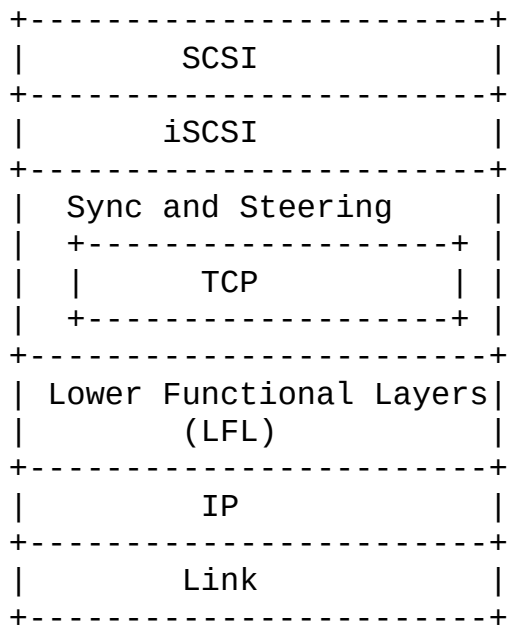
In situations where IP packets are delivered in order from the network, iSCSI message framing is not an issue and messages are processed one after the other. In the presence of IP packet reordering, (i.e., frames being dropped) legacy TCP implementations store the "out of order" TCP segments in temporary 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). Because 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. One of these schemes is detailed in Appendix A. - Sync and Steering with Fixed Interval Markers -. 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.

2.2.8.2 Synchronization (sync) and Steering Functional Model

We assume that iSCSI is implemented according to the following layering scheme:



In this model, LFL can be IPsec (a mechanism changing the IP stream and invisible to TCP). We assume that Sync and Steering operates just underneath iSCSI. An implementation may choose to place Sync and Steering somewhere else in the stack if it can translate the information kept by iSCSI in terms valid for the chosen layer.

According to our layering model, iSCSI considers the information it delivers to the Sync and Steering layer (headers and payloads) as a contiguous stream of bytes mapped to the positive integers from 0 to infinity. In practice, though, iSCSI is not expected to handle infinitely long streams; stream addressing will wrap around at 2**32-1.

This model assumes that the iSCSI layer will deliver complete PDUs to underlying layers in single (atomic) operations. The underlying layer does not need to examine the stream content to discover the PDU boundaries. If a specific implementation performs PDU delivery to the Sync and Steering layer through multiple operations, it MUST bracket an operation set used to deliver a single PDU in a manner that the Sync and Steering Layer can understand.

The Sync and Steering Layer (which is OPTIONAL) MUST retain the PDU end address within the stream for every delivered iSCSI PDU.
To enable the Sync and Steering operation to perform Steering, additional information, including identifying tags and buffer offsets,

MUST also be retained for every sent PDU. The Sync and Steering Layer is required to add enough information to every sent data item (IP packet, TCP packet or some other superstructure) to enable the receiver to steer it to a memory location independent of any other piece.

If the transmission stream is built dynamically, this information is used to insert Sync and Steering information in the transmission stream (at first transmission or at re-transmission) either through a globally accessible table or a call-back mechanism. If the transmission stream is built statically, the Sync and Steering information is inserted in the transmission stream when data are first presented to sync and steering.

The retained information can be released whenever the transmitted data are acknowledged by the receiver. (in the case of dynamically built streams, by deletion from the global table or by an additional callback).

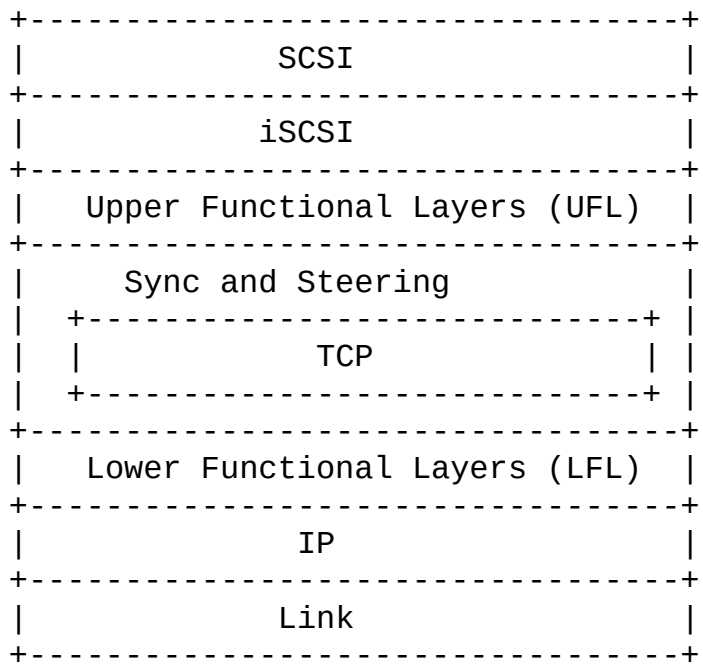
On the outgoing path, the Sync and Steering layer MUST map the outgoing stream addresses from iSCSI stream addresses to TCP stream sequence numbers.

On the incoming path, the Sync and Steering layer extracts the Sync and Steering information from the TCP stream. It then helps steer (place) the data stream to its final location and/or recover iSCSI PDU boundaries when some TCP packets are lost or received out of order. The data stream seen by the receiving iSCSI layer is identical to the data stream that left the sending iSCSI layer. The Sync and Steering information is kept until the PDUs to which it refers are completely processed by the iSCSI layer.

On the incoming path, the Sync and Steering layer does not change the way TCP notifies iSCSI about in-order data arrival. All data placements, in-order or out-of-order, performed by the Sync and Steering layer are hidden from iSCSI while conventional, in order, data arrival notifications generated by TCP are passed through to iSCSI.

2.2.8.3 Sync and Steering and Other Encapsulation Layers

We recognize that in many environments the following is a more appropriate layering model:



In this model, UFL can be TLS (see[RFC2246]) or some other transport conversion mechanism (a mechanism that changes the TCP stream, but that is transparent to iSCSI).

To be effective and act on reception of TCP packets out of order, Sync and Steering has to be underneath UFL, and Sync and Steering data must be left out of any UFL transformation (encryption, compression, padding etc.). However, Sync and Steering MUST take into account the additional data inserted in the stream by UFL. Sync and Steering MAY also restrict the type of transformations UFL may perform on the stream.

This makes implementation of Sync and Steering in the presence of otherwise opaque UFLs less attractive.

2.2.8.4 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.

2.3 iSCSI Session Types

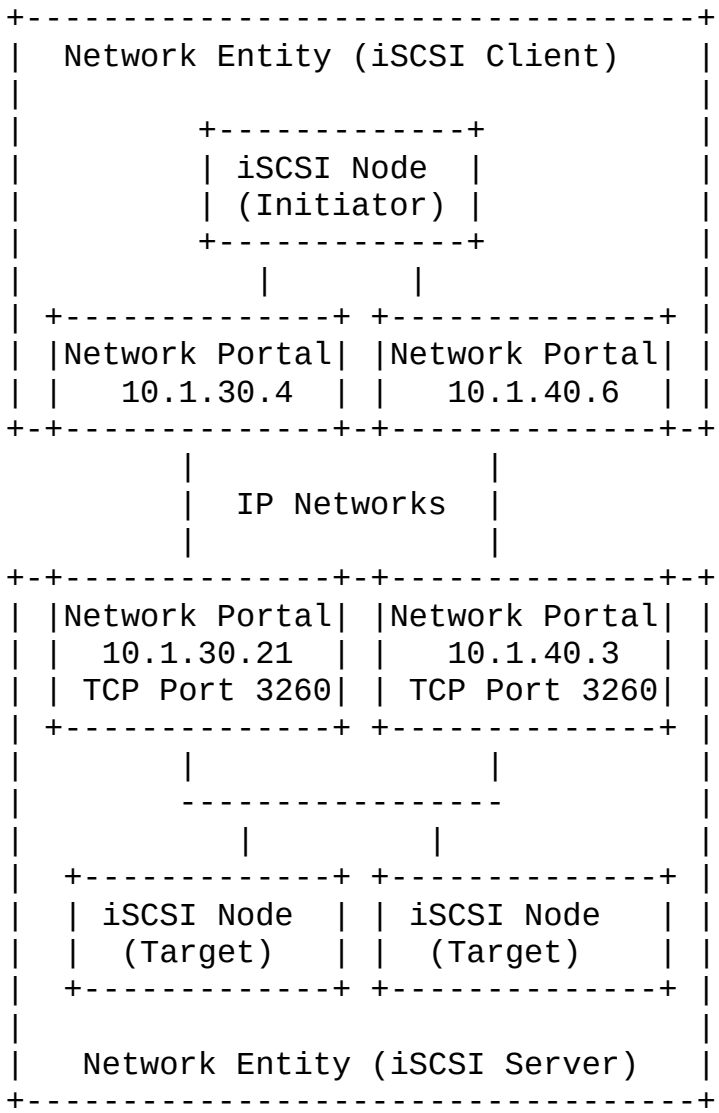
iSCSI defines two types of sessions:

a) Normal operational session - an unrestricted session.
b) Discovery-session - a session opened only for target discovery; the target MAY accept only text requests with the SendTargets key and a logout request with reason "close the session".

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

2.4 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. See Section 2.4.1 iSCSI Architecture Model for detailed descriptions of the components of these diagrams.



2.4.1 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 2.2.6 iSCSI Names and Chapter 11). 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 could 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.

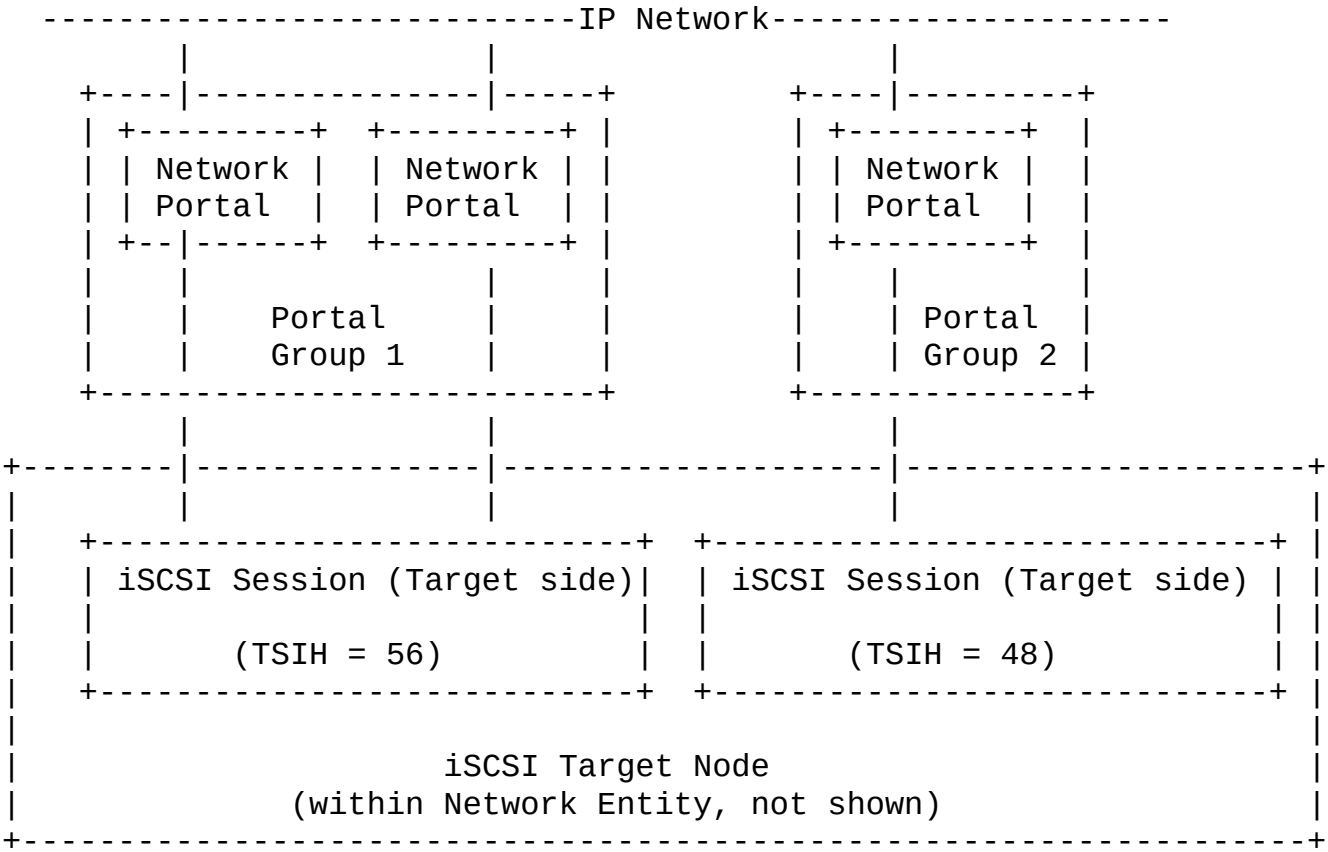
d) Network Portal - a component of a Network Entity that has a TCP/IP network address and that may be used by an iSCSI Node within that Network Entity for the connection(s) within one of its iSCSI sessions. In an initiator, it is identified by its IP address. In a target, it is identified by its IP address 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 1 and 65535 (see Section 11.3 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). See Section Section 8.1.1 Conservative Reuse of ISIDs for references to the Initiator Portal Groups.

f) Portals within a Portal Group are expected to have similar hardware characteristics, as SCSI port specific mode pages may affect all portals within a portal group. (See Section 2.4.3.2 SCSI Mode Pages).

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.



2.4.2 SCSI Architecture Model

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

This relationship implies implementation requirements in order to conform to the SAM2 model and other SCSI operational functions. These requirements are detailed in Section 2.4.3 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 other SCSI entities. 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 a given iSCSI Node. Access to the SCSI Device can only be achieved in an iSCSI normal operational session (see Section 2.3 iSCSI Session Types). The SCSI Device Name is defined to be the iSCSI Name of the node and its use is mandatory 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 2.3 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 is mandatory 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), followed by
- a comma separator (1 byte), followed by
- zero to 3 null pad bytes so that the complete format is a multiple of four bytes long, followed by

- the 6byte value of the ISID (for SCSI initiator port) or the 2byte value of the portal group tag (for SCSI target port) in network byte order (BigEndian).
SCSI port names have a maximum length of 264 bytes for initiator ports, 260 bytes for target ports, and must be a multiple of four bytes long. The ASCII character 'i' or 't' is the label that identifies this port as either a SCSI Initiator Port or a SCSI Target Port. This ASCII character also provides the interpretation and length of the remaining six bytes (initiator) or two bytes (target).

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. That is, the I_T nexus identifier is 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).

2.4.3 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. That is, no more than one nexus relationship (parallel nexus) is allowed. Therefore, between a given iSCSI initiator node and an iSCSI target node, at any given time, only one session can exist 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 be only one session with a given value for ISID that identifies the SCSI initiator port. See Section 9.12.6 ISID.

The structure of the ISID that contains a naming authority component (see Section 9.12.6 ISID and [NDT]) provides a mechanism to facilitate compliance with the ISID rule (See also Section 8.1.1 Conservative Reuse of ISIDs).

The iSCSI Initiator Node is expected to 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 8.1.1 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 initiator or target refer to the given session and a TSIH is required.

2.4.3.1 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 target (or more correctly stated, the device server 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 re-establish 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.

2.4.3.2 SCSI Mode Pages

If the SCSI logical unit device server does not maintain initiator-specific mode pages, and an initiator makes changes to port-specific

mode pages, the changes may affect all other initiators logged in to that iSCSI Target through the same Target Portal Group.

Changes via mode pages to the behavior of a portal group via one iSCSI Target Node should not affect the behavior of this portal group with respect to other iSCSI Target Nodes, even if the underlying implementation of a portal group serves multiple iSCSI Target Nodes in the same Network Entity.

2.5 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 be followed by a CRC (digest).

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

2.5.1 Request/Response types carrying SCSI payload

2.5.1.1 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, Command Sequence Number, Expected Data Transfer Length for one or both transfer directions (the latter for bidirectional commands), and Task Tag. The I_T_L nexus is derived by the initiator and target from the LUN field in the request and the I_T nexus 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) and the expected status number on the connection it is issued (ExpStatSN).

Part or all 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.

2.5.1.2 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.

It contains the residual counts from the operation if any, and 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 is executed but resulted in a exception (e.g., a SCSI CHECK CONDITION), the PDU data segment contains the associated sense data.

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) - ExpDataSN, the Status Sequence Number on this connection - StatSN and the next Expected Command Sequence Number at the target - ExpCmdSN, the Maximum CmdSN acceptable at the target from this initiator.

2.5.1.3 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 and is carried by the LUN (and implied by the session identification).

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.

The task management function request execution is completely performed at the target, (i.e., any coordination between responses to the tasks affected and the task management function request response is done by the target).

2.5.1.4 Task Management Function Response

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

After the task management response indicating task management function completion, the initiator will not receive any additional responses from the affected tasks.

2.5.1.5 SCSI Data-out and SCSI Data-in

The 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 the 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 exec 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 is a complete payload sent in response to an R2T "prompt".

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).

StatSN is also carried by the Data-In PDUs.

To enable a SCSI command to be executed 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).

2.5.1.6 Ready To Transfer (R2T)

R2T is the mechanism by which the SCSI target "prompts" the initiator for output data. R2T passes 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 the initiator.

To help the SCSI target to associate resulting Data-out with an R2T, the R2T carries the Target Transfer Tag 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.

R2T also carries information required for proper operation of the iSCSI protocol, such as an R2TSN (to enable an initiator to detect a missing R2T), StatSN, ExpCmdSN and MaxCmdSN.

2.5.2 Requests/Responses carrying SCSI and iSCSI Payload

2.5.2.1 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.

2.5.3 Requests/Responses carrying iSCSI Only Payload

2.5.3.1 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 key=value, Text Requests/Responses carry text information with a simple 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/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 reset the target (start fresh) it 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.

2.5.3.2 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 (it is called the leading login) and some header fields are determined by the leading login (e.g., the version number, the maximum number of connections, the session identification etc.).

The command counting initial value is also set by the leading login.

Status counting 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 (we call this a connection restart) by using the same Connection ID (CID) as an existing connection, in the login request header, as well as the same session identifying elements of the session to which the old connection was associated.

2.5.3.3 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). The Logout Response indicates also how long the target will keep on holding resources for recovery (e.g., command execution that continues on a new connection) in Time2Retain and how long the initiator must wait before proceeding with recovery in Time2Wait.

2.5.3.4 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.

2.5.3.5 Reject

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

2.5.3.6 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 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.

3. SCSI Mode Parameters for iSCSI

There are no iSCSI specific mode pages.

4. Login and Full Feature Phase Negotiation

4.1 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 they appear in this document. They are case sensitive.

The following character symbols are used in this document for text items:

(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
nul (0x00) - nul 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/Login Request or Text/Login Response to 1. Data segments in a series of PDUs having the C bit set to 1 and ending with a PDU having the C bit set to 0 or including a single PDU having 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 with the meaning of key-name.

A value is whatever follows the first = in the key=value pair up to the end of the key=value pair.

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

key-name: a string of one or more characters consisting of letters, digits, dot, minus, plus, commercial at, and underscore, A key-name MUST begin with a capital letter an must not exceed 63 characters.

text-value: a string of 0 or more characters consisting of letters, digits, dot, minus, plus, commercial at, underscore, slash, left bracket, right bracket and colon.

iSCSI-name-value: a string of one or more characters consisting of minus, dot, colon and any character allowed by the output of the iSCSI string-prep template as specified in [STPREP-iSCSI] (see also Section 2.2.6.2 iSCSI Name Encoding).

iSCSI-local-name-value: an UTF-8 string; no nul 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: hexadecimal constant encoded as a string starting with "0x" or "0X" followed by 1 or more digits or the letters a, b, c, d, e, f, A, B, C, D, E and 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 and the string following 0X (or 0x) represents a base16 number starting with the most significant base16 digit, followed by all other digits in decreasing significance order and ending with the least-significant base16 digit. When used to encode binary strings hexadecimal constants have an implicit byte-length that includes 4 bits for every hexadecimal digit of the constant, including leading zeroes (i.e., 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 - the digit 0 or a string of 1 or more digits starting with a non-zero digit. This encoding is not used for numerical values equal or greater than 2**64. Decimal-constants are used to encode numerical values or binary strings. When used to encode binary strings decimal constants have an implicit byte-length

that is the minimum number of bytes needed to represent the base2 encoding of the decimal number.

base64-constant: base64 constant encoded as a string starting 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 a A) is discouraged and the string following 0B (or 0b) represents a base64 number starting with the most significant base64 digit, followed by all other digits in decreasing significance order 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 6 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)). N.B. correctly encoded base64 strings cannnot have n values of 1, 5 ... k*4+1.

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

large-numerical-value: an unsigned integer 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 that the value to the left.

regular-binary-value: a binary string less 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 implicit byte-length of the encoded string.

large-binary-value: a binary string encoded as a hex-constant or base64-constant. The length of the string is either specified by the key definition or is 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 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 16384 bytes of key=value data in a negotiation sequence except when indicating support for very long authentication items by offering or selecting authentication methods such as public key certificates in which case they MUST support receiving at least 64 kilobytes of key=value data.

4.2 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/Login request and indicates when it is ready for completion (by setting to 1 and keeping to 1 the F bit in a Text Request or the T bit in the Login Request). As negotiation text may span PDU boundaries a Text/Login Request or Text/Login Response PDU having the C bit set to 1 MUST NOT have the F/T bit set to 1.

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

A target or initiator SHOULD NOT use a Text/Login Response or Text/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:

```
Originator-> <key>=<valuex>
Responder-> <key>=<valuey>|NotUnderstood|Irrelevant|Reject
```

The originator or declarer can either be the initiator or the target and the responder can either be the target or initiator, respectively. Targets are not limited to respond to key=value pairs as offered by the initiator. The target may offer key=value pairs of its own.

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

The value offered or declared can be a numerical-value, a numerical-range defined by lower and upper value - both integers separated by tilde, a binary value, a text-value, a iSCSI-name-value, an iSCSI-local-name-value, a boolean-value (Yes or No), or a list of comma separated text-values. A range or a large-numerical-value MAY ONLY be offered if it is explicitly allowed for a key. An iSCSI-name-value and an iSCSI-local-name-value can be used only where explicitly allowed. A selected value can be an 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 responder may answer with the constant "Irrelevant" for all types of negotiation. However the negotiation is not considered as failed if the response is "Irrelevant".

Any key not understood by the responder may be ignored by the responder without affecting the basic function. However, the Text Response 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.

Reject or Irrelevant are legitimate negotiation options where allowed but their excessive use is discouraged. A negotiation is considered complete when the responder has sent the key value pair even if the

value is "Reject", "Irrelevant", or "NotUnderstood. Sending the key again would be a re-negotiation

Some basic key=value pairs are described in Chapter 11. All keys in Chapter 11, except for the X- extension format, MUST be supported by iSCSI initiators and targets and MUST NOT be answered with NotUnderstood.

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

X-com.acme.bar.foo.do_something=3

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

Negotiations MUST be handled as atomic operations - i.e., all negotiated values get 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 to be Yes). Whenever required integrity rules are specified with the keys. Checking for compliance with the integrity rule MUST NOT be performed before all the negotiation parameters are available (the existent and newly negotiated). An iSCSI target MUST perform integrity checking before committing new values for parameters. An initiator MAY perform integrity checking.

4.2.1 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 a valid selection only if it is explicitly offered.

If a responder does not understand any particular value in a list it MUST ignore it. If a responder does support, understand or is allowed to use none of the offered options with a specific originator, it may use the constant "Reject" or terminate the negotiation. The selection of a value not offered is considered a negotiation failure and is handled as a protocol error.

4.2.2 Simple-value negotiations

For simple-value negotiations, the responding party MUST respond with the same key. The value it selects, based on the selection rule specific to the key, becomes the negotiation result. For a numerical range the value selected must be an integer within the offered range or "Reject" (if the range is unacceptable). An offer of a value not admissible (e.g., not within the specified bounds) MAY be answered with the constant "Reject" or the responder MAY select an admissible value. The selection, by the responder, of a value not admissible under the selection rules is considered a negotiation failure and is handled accordingly. The selection rules are key-specific.

For boolean negotiations (keys taking the values Yes or No), the responding party MUST respond 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 with which to respond are expressed as Boolean functions of the value received and the value that the responding party would have selected if given a choice.

Specifically, the two cases in which responses 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 responder becomes the outcome of the negotiation.

4.3 Login Phase

The Login Phase establishes an iSCSI session between an initiator and a target. It sets the iSCSI protocol parameters, security parameters, and authenticates the initiator and target to each other.

The Login Phase is implemented via login request and responses only. 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)

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 and MUST include at least one of them, but the included stage MAY be empty except for the mandatory names.

The login requests and responses contain a field that indicates the negotiation stage (SecurityNegotiation or LoginOperationalNegotiation). If both stages are used, the SecurityNegotiation MUST precede the LoginOperationalNegotiation.

Some operational parameters can be negotiated outside login through text request/response.

Security MUST be completely negotiated within the Login Phase. How to use underlying IPsec security is specified in Chapter 7 and in [SEC-IPS].

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 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 10 and the LoginOperationalNegotiation keys appear in Chapter 11. Only a limited set of keys (marked as Any-Stage in Chapter 11) 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.

Stage transition is performed through a command exchange (request/response) that carries the T bit and the same current stage code. During this exchange, the next stage is selected by the target and MUST NOT exceed the value stated by the initiator. The initiator can request a transition whenever it is ready, but a target can respond with a transition only after one is offered 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 T bit set to 0.

When a transition is requested by the initiator and acknowledged by the target both 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 possible only 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 come only as a response to a Login request with the T bit set to 1, and both the request and response MUST have FullFeaturePhase in 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). If detected by the target this MUST result in a Login reject (initiator error). The initiator MUST drop the connection

4.3.1 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.
- -The 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 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
existing	valid existing	new	add a new connection to the session
existing	valid existing	existing	do connection reinstatement
existing	invalid	any	fail the login ("session does not exist")

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 tar-

get 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 without SecurityNegotiation and will terminate the connection.

If the initiator is willing to negotiate security, but is unwilling to make the initial parameter offer and may accept a connection without security, it issues the Login with the T bit set to 1, the CSG set to SecurityNegotiation, and NSG set to LoginOperationalNegotiation. If the target is also ready to forego security, the Login response is empty and has T bit set to 1, the CSG set to SecurityNegotiation, and NSG set to LoginOperationalNegotiation.

An initiator that can operate without security and 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 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 NSG set to FullFeaturePhase in the next stage.

The first Login Response PDU during the Login Phase from the iSCSI target SHOULD return the TargetPortalGroupTag key that contains the tag value of the iSCSI portal group servicing the Login Request PDU. If the iSCSI target implementation supports altering the portal group configuration (including adding, deleting, and swapping of portals in a portal group), it MUST return the TargetPortalGroupTag key carrying the tag value of the servicing portal group. If the reconfiguration of iSCSI portal groups is a concern in a given environment, the iSCSI initiator MUST use this key to ascertain that it had indeed initiated the Login Phase with the intended target portal group.

4.3.2 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 proprietary 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 also Section 4.2 Text Mode Negotiation). The parameters are encoded in UTF8 as key=value. For security parameters, see Chapter 10.

- When the initiator considers that it 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 NSG to the next stage in the Login response where it finishes sending its security keys. The next 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 (offer options).

4.3.3 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 offer 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 having the T bit set to 1 with a Login response having 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 having the T bit set to 1.

Some session specific parameters can be specified only during the Login Phase 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 Full-FeaturePhase. New or replacement connections can be added to a session only after the session is operational.

For operational parameters, see Chapter 11.

4.3.4 Connection reinstatement

Connection reinstatement is the process of initiator logging in with a ISID-TSIH-CID combination that is possibly active from the target's perspective - thus implicitly logging out the connection state machine 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 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 should support opening a second connection even when they do not support multiple connections in Full Feature Phase.

If the operational ErrorRecoveryLevel is 2, connection reinstatement enables future task reassignment. If the operational ErrorRecovery-Level 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 are internally terminated.

The initiator connection state MUST be CLEANUP_WAIT (section 5.1) for attempting a connection reinstatement.

In practical terms, beside the implicit logout connection, reinstatement is equivalent to a new connection login.

4.3.5 Session reinstatement, closure and timeout

Session reinstatement is the process of initiator logging in with an ISID that is possibly active from the target's perspective - thus implicitly logging out the session state machine corresponding to the ISID and reinstating a new iSCSI session in its place (with the same ISID). Thus, the TSIH in the Login PDU MUST be zero to signal session reinstatement. All the tasks that were active on the old session are internally terminated on a session reinstatement.

The initiator session state MUST be FAILED (Section 5.3 Session State Diagrams) for attempting a session reinstatement.

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

- a successful "session close" logout
- a successful "connection close" logout for the last Full Feature Phase connection when no associated connection states are waiting for cleanup (Section 5.2 Connection Cleanup State Diagram for Initiators and Targets) and no associated task states are waiting for reassignment.

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

4.3.5.1 Loss of Nexus notification

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

a) A successful completion of session reinstatement
b) A session closure event
c) A session timeout event

Certain SCSI object clearing actions may result upon this notification in the SCSI end nodes, as documented in Appendix F. - Clearing effects of various events on targets -.

4.3.6 Session continuation and failure

Session continuation is the process by which the state of a pre-existing session is continued to be in use by either connection reinstatement (Section 4.3.4 Connection reinstatement), or by adding a connection with a new CID. Either of these actions associates the new transport connection with the pre-existing session state.

Session failure is an event where the last Full Feature Phase connection reaches the CLEANUP_WAIT (Section 5.2 Connection Cleanup State Diagram for Initiators and Targets) state, or completes a successful recovery logout thus causing all active tasks (that are formerly allegiant to the connection) to start waiting for task reassignment.

4.4 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 and uses 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 with a text response with the F bit set to 0 to a text request with the F bit set to 1, 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 not an error but 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 reset except for responses to specific keys that explicitly allow repeated key declarations (e.g. TargetAdress). If detected by the target this MUST result in a Reject with a reason of "protocol error". The initiator MUST reset the negotiation as outlined above.

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

5. 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.

An iSCSI connection is a transport connection used for carrying out iSCSI activity. 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.

5.1 Standard Connection State Diagrams

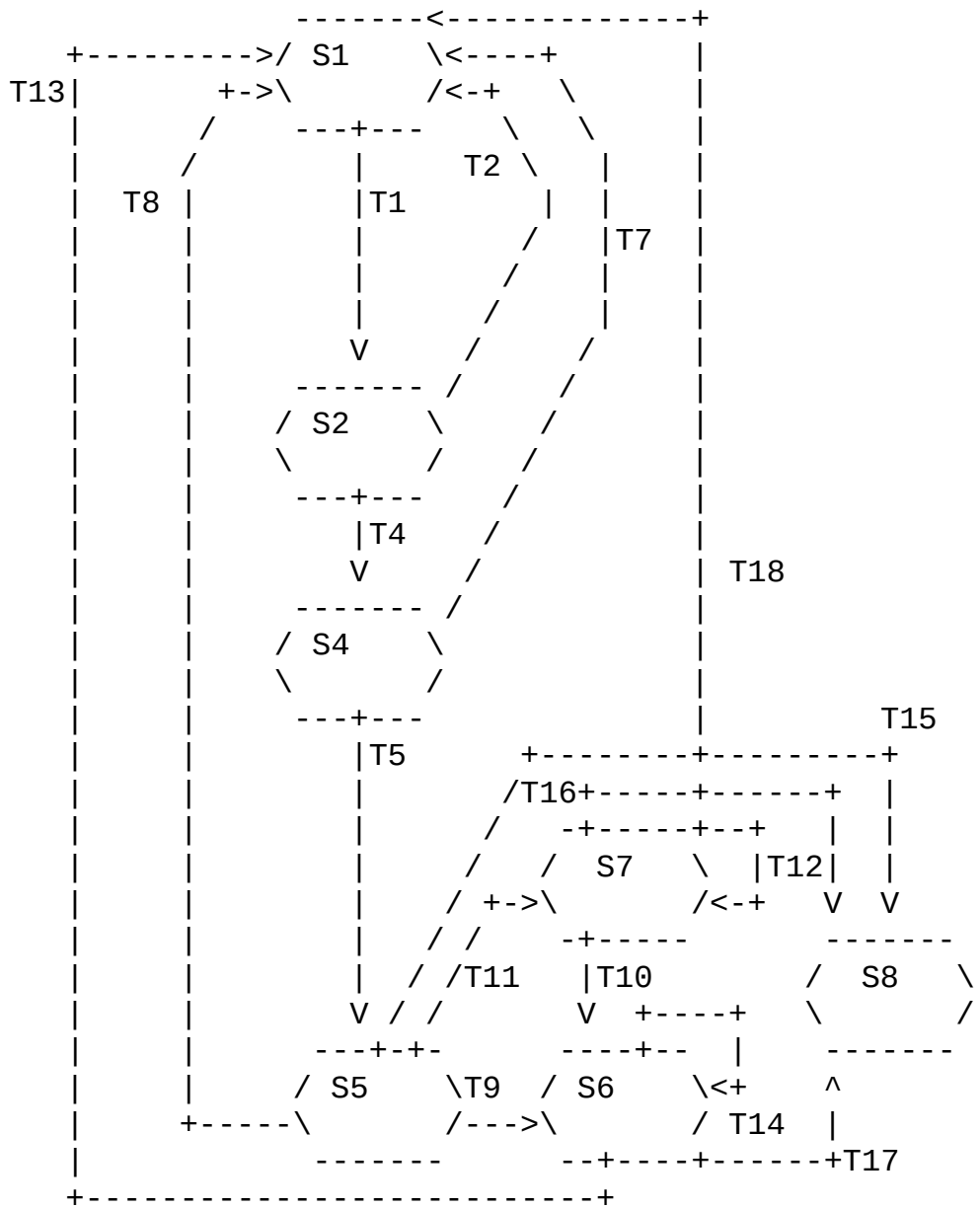
5.1.1 Standard Connection State Diagram for an Initiator

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:



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	-	-	-	-	-	-	-

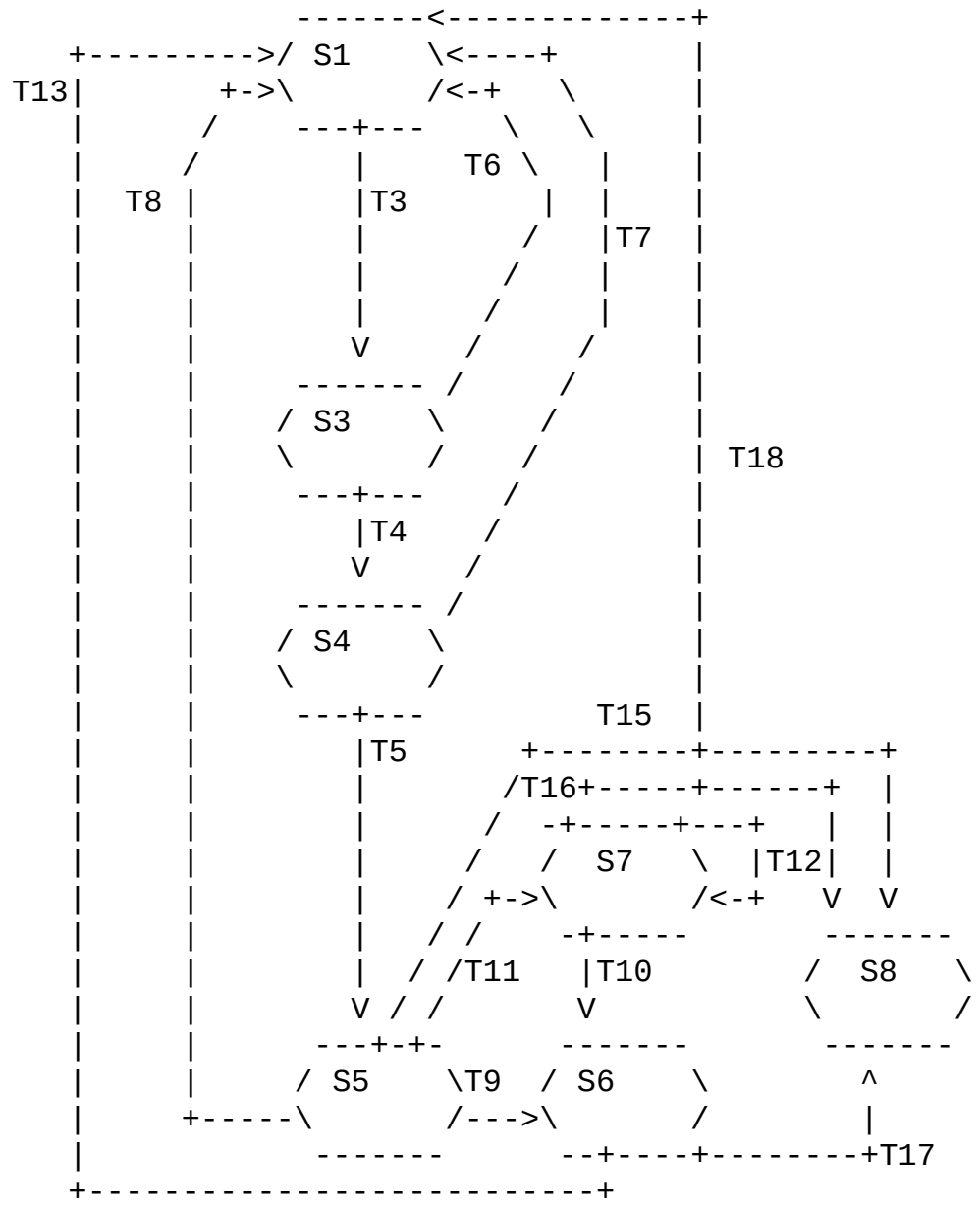
5.1.2 Standard Connection State Diagram for a Target

Symbolic names for States:

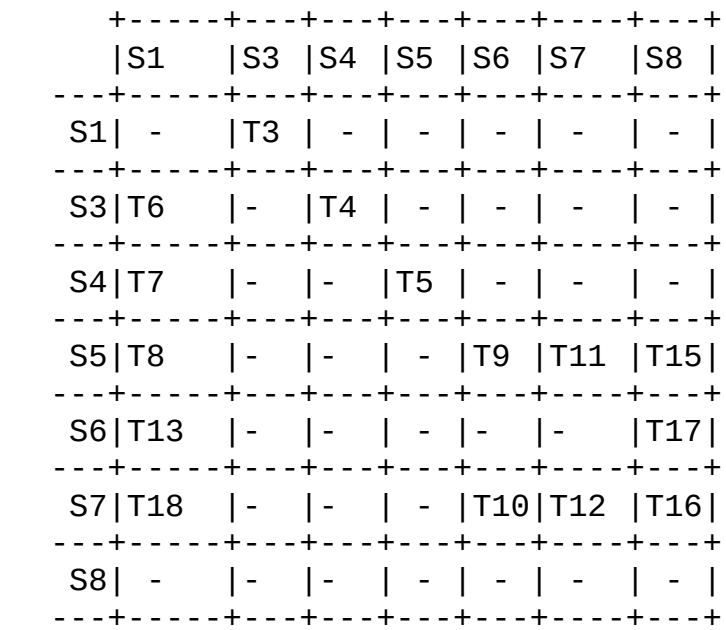
- S1: FREE
- S3: XPT_UP
- S4: IN_LOGIN
- S5: LOGGED_IN
- S6: IN_LOGOUT
- S7: LOGOUT_REQUESTED
- S8: CLEANUP_WAIT

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

The state diagram is as follows:



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



5.1.3 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.

-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.

5.1.4 State Transition Descriptions for Initiators and Targets

-T1:

-initiator: Transport connect request was made (ex: TCP SYN sent).

-target: Illegal

-T2:

-initiator: Transport connection request timed out, or 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, or transport disconnect indication was received, or transport reset was received, or an internal event indicating a transport time-out was received. In all these cases, the connection is to be closed.

-T7:
-initiator - one of the following evens 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 indicating a transport timeout was received
- 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, or 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.

5.2 Connection Cleanup State Diagram for Initiators and Targets

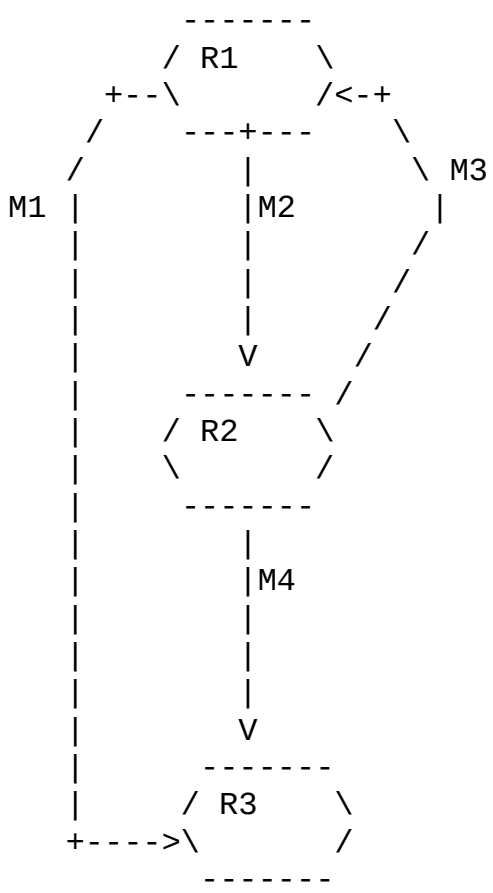
Symbolic names for states:

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 additionally 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 4.3.4) 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 applicable only 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.



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	-	-	-

5.2.1 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.

5.2.2 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, or 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.

5.3 Session State Diagrams

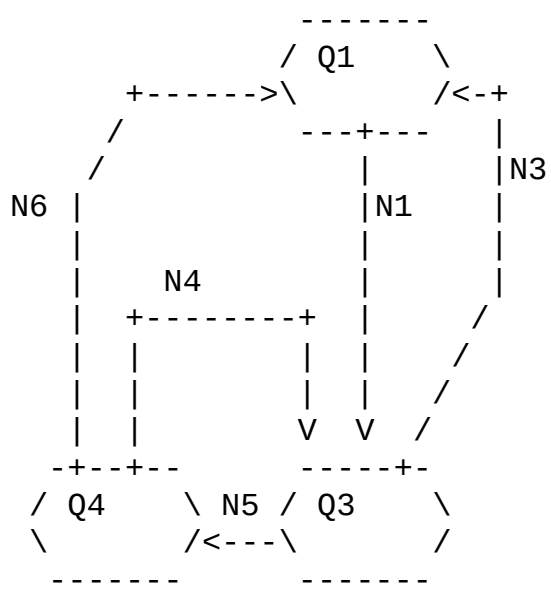
Session State Diagram for an Initiator

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:



State transition table:

	Q1	Q3	Q4
Q1	-	N1	-
Q3	N3	-	N5
Q4	N6	N4	-

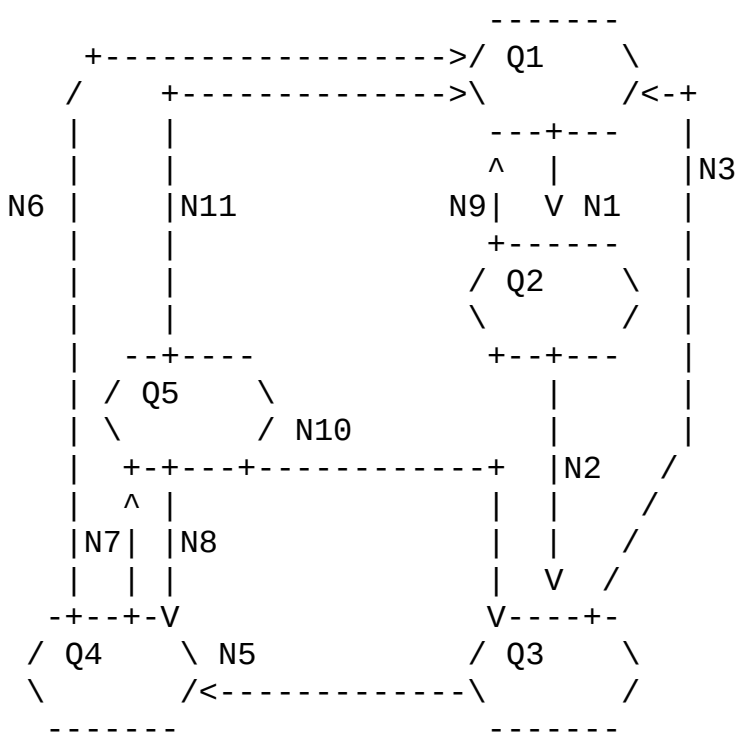
5.3.1 Session State Diagram for a Target

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:



State transition table:

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

5.3.2 State Descriptions for Initiators and Targets

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

5.3.3 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 4.3.6 Session continuation and failure).
  -target: Graceful closing of the session via session closure (Section 4.3.6 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 4.3.6 Session continuation and failure) occurred.
-target: Session failure (Section 4.3.6 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.

6. iSCSI Error Handling and Recovery

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 are 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 re-read 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.

Many of the recovery details in an iSCSI implementation are a local matter, beyond the scope of protocol standardization. However, some external aspects of the processing must be standardized to ensure interoperability. This section describes a general model for recovery in support of interoperability. See Appendix E. - Algorithmic Presentation of Error Recovery Classes - for further detail. 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 Retry and Reassign in Recovery

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

6.1.1 Usage of Retry

By resending the same iSCSI command PDU ("retry") in the absence of a command acknowledgement or 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. In particular, all PDU retransmission (for data, or status) requests for a currently allegiant command in progress must be conveyed to the target using only the SNACK mechanism already described in Section 2.5.3.4 SNACK Request. This, however, does not constitute a requirement on initiators to use SNACK.

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), targets MUST silently discard the duplicate requests if the CmdSN window had not advanced by then. Targets MUST support the retry functionality described above.

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.

6.1.2 Allegiance Reassignment

By issuing a "task reassign" task management request (Section 9.5.1 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 established for the command, that associates it to the connection on which the task management request is being issued.

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 Exp-DataSN 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. However, targets MAY choose to send/receive the entire data on a reassignment of connection allegiance, and it is not considered an error. 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. This might possibly involve retransmission of data/R2T/status PDUs as necessary.

It is optional for targets to support the allegiance reassignment. This capability is negotiated via the ErrorRecoveryLevel text key at the login time. When a target does not support allegiance reassignment, it MUST respond with a task management response code of "Task

failover not supported". If allegiance reassignment is supported by the target, but the task is still allegiant to a different connection, the target MUST respond with a task management response code of "Task still allegiant".

6.2 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 initiators can eventually expect a response even on Reject's, if the Reject is not for the command itself. The non-command Reject's 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 carried one) MUST NOT be considered received by the target (i.e., a command sequence gap must be assumed for the CmdSN), even though the CmdSN can be reliably ascertained in this case.

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.

6.3 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 on 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 target to cater to a possible recovery attempt. No recovery attempt should be made before Time2Wait and task reassignment has to be done within the Time2Retain.

6.3.1 Timeouts on transport exception events

A transport connection shutdown or a transport reset without any preceding iSCSI protocol interactions informing 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 11.16 DefaultTime2Wait) and DefaultTime2Retain (Section 11.17 DefaultTime2Retain) text keys for the session.

6.3.2 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 9.15) 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 9.9.1) specify the timeout values to be used in each of these cases.

These timeout values are applicable only 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.

6.4 Format Errors

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

a) illegal contents of the PDU header (except the Opcode) - for ex., out-of-range values for certain fields
b) inconsistent contents - for ex., value of one field conflicts with that of another.

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 6.12.4 Session Recovery).

6.5 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.

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. Since the digest error indicate 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. [OR]

b) Terminate the task with a response PDU with a CHECK CONDITION Status and an iSCSI Condition of "protocol service CRC error" (Section 9.4.6.2 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 (similar to 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.

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 its turn, 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 had not already been sent for the command, the target MUST terminate the command with an CHECK CONDITION Status and an iSCSI Condition of "SNACK rejected" (Section 9.4.6.2 Sense Data).
ii) if the status was already sent, no further action is necessary for the target. Initiator in this case MUST internally signal the completion with CHECK CONDITION Status and an iSCSI Condition of "protocol service CRC error" (Section 9.4.6.2 Sense Data) disregarding any received status PDU, but must wait for the status to be received before doing so.
b) [OR] 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. [OR]
b) Logout the connection for recovery and continue the tasks on a different connection instance as described in Section 6.1 Retry and Reassign in Recovery. [OR]
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).

6.6 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 hit 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 6.5 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. Target MUST address these implied digest errors as described in Section 6.5 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 6.5 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 interested 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.

6.7 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 6.1 Retry and Reassign in Recovery.

On a ULP timeout for a command (that carried a CmdSN of n), the iSCSI initiator MUST abort the command by either using the Abort Task task management function request, or a "close the connection" Logout if it intends to continue the session. In using an explicit Abort, 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:

- The original command was dropped due to digest error.
- The connection on which the original command was sent was successfully logged out (on 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.

6.8 Negotiation Failures

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

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

The following two rules are to 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 be undone).

6.9 Protocol Errors

The authors recognize that mapping framed messages over a "stream" connection, such as TCP, make 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 re-establishment help handle this type of mapping errors.

All violations of iSCSI PDU exchange sequences specified in this draft 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.

6.10 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. It is assumed that targets and/or initiators recognize a failing connection by either transport level means (TCP), a gap in the command, a response stream that is not filled for a long time, or by a failing iSCSI NOP (ping). The latter

MAY be used periodically by highly reliable implementations. Initiators and targets MAY also use the keep-alive option on the TCP connection to enable early link failure detection on otherwise idle links.

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

- Attempt connection recovery within the session (Section 6.12.3 Connection Recovery).
- Logout the connection with the reason code "closes the connection" (Section 9.14.4 Implicit termination of tasks), reissue missing commands, and implicitly terminate all active commands. This option requires support for the within-connection recovery class (Section 6.12.2 Recovery Within-connection).
- Perform session recovery (Section 6.12.4 Session Recovery).

Either side may choose to escalate to session recovery, and the other side MUST give it precedence. On a connection failure, a target MUST terminate and/or discard all the active immediate commands regardless of which of the above options is used (i.e., immediate commands are not recoverable across connection failures).

6.11 Session Errors

If all the connections of a session fail and cannot be re-established in a short time, or if initiators detect protocol errors repeatedly, an initiator may choose to terminate a session and establish a new session.
The initiator takes the following actions:

- It resets or closes all the transport connections.
- It terminates all outstanding requests with an appropriate response before initiating a new session.

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).
- Aborts all Tasks in the task set for the corresponding initiator.

6.12 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.

6.12.1 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 6.5 Digest Errors, using the option of a recovery R2T.

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 6.6 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 6.6 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 6.5 Digest Errors, using the option of a SNACK.

b) Sequence reception timeout (no status) or response reception timeout - dealt with as specified in Section 6.6 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 6.6 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.

6.12.2 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 6.1 Retry and Reassign in Recovery.

- 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 6.5 Digest Errors using the option of a SNACK. In the second case, sequence error handling is done as specified in Section 6.6 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 detects 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.

6.12.3 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 Logout the failed connection, or Login with an implied Logout, and reassign connection allegiance for all commands still in progress associated with the failed connection on another connection (that MAY be a newly established connection) using the "Task reassign" task management function (see Section 9.5.1 Function). Note that for an initiator a command is in progress as long as it has not received a response or a Data-In PDU including status.

  N.B. The logout function is mandatory, while a new connection establishment is mandatory only 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.

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.

6.12.4 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 therefore place the burden of 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).

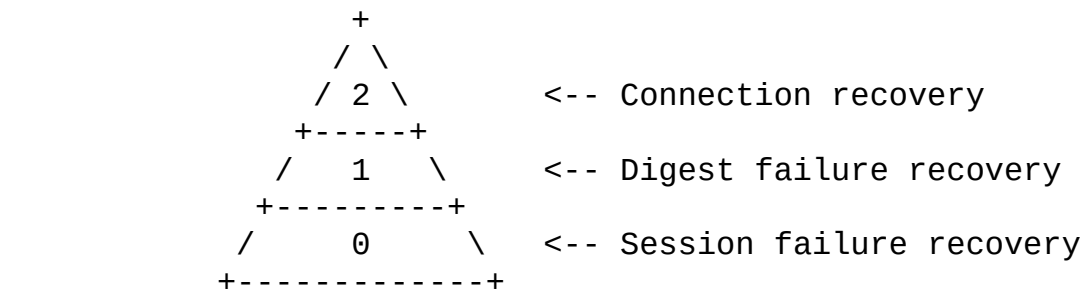
For possible clearing effects of session recovery on SCSI and iSCSI objects, refer to Appendix F. - Clearing effects of various events on targets -.

6.13 Error Recovery Hierarchy

The error recovery classes and features described are organized into a hierarchy for ease in understanding and to limit the myriad of implementation possibilities, with hopes that this significantly contributes to highly interoperable implementations. 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.

The following diagram represents the error recovery hierarchy.



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 6.12.4 Session Recovery)
1	Digest failure recovery (See Note below.)
2	Connection recovery class (Section 6.12.3 Connection Recovery)

Note: Digest failure recovery is comprises two recovery classes: Within-Connection recovery class (Section 6.12.2 Recovery Within-connection) and Within-Command recovery class (Section 6.12.1 Recovery Within-command).

Supporting error recovery level "0" is mandatory, 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

7. 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).

Although technically possible, iSCSI SHOULD NOT be configured without security. iSCSI without security should be confined, in extreme cases, to closed environments without any security risk.

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

7.1 iSCSI Security Mechanisms

The entities involved in iSCSI security are the initiator, target, and the IP communication end points. iSCSI scenarios where 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 [SEC-IPS].

7.2 In-band Initiator-Target Authentication

During login the target authenticates the initiator and the initiator optionally authenticates 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 mechanism alone (without underlying IPsec) should only be used when there is no risk of eavesdropping, message insertion, deletion, modification, and replaying.

Section 10 iSCSI Security Keys and Authentication Methods defines several authentication methods and the exact steps that must be followed in each of them, including the 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 value, e.g., the CHAP response, for which "Authentication Failure" status MUST be specified). The importance of this rule can be illustrated in CHAP with target authentication (Section 10.5 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 since the target must answer the missing CHAP_R key with a Login reject with the "Missing Parameter" status.

7.2.1 CHAP Considerations

Compliant iSCSI implementation MUST implement the CHAP authentication method [RFC1994] (according to Section 10.5 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 bits 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 7.3.2 Confidentiality) to protect the connection. Moreover, in this case IKE authentication with group pre-shared keys SHOULD NOT be used unless it is not essential to protect group members against off-line dictionary attacks by other members.

A compliant implementation SHOULD NOT continue with the login step in which it should send a CHAP response (CHAP_R - Section 10.5 Challenge Handshake Authentication Protocol (CHAP)) unless it can verify that either the CHAP secret is at least 96 bits, or that IPsec encryption is being used to protect the connection.

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.

7.2.2 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.

Upon receiving N and g from the Target, the Initiator MUST verify that they match a well-known group that satisfies the above requirements and abort the connection if they do not match. Well- known SRP groups are provided in [SEC-IPS].

7.3 IPsec

The IPsec mechanism is used by iSCSI 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 key management.

Detailed considerations and recommendations for using IPsec for iSCSI are provided in [SEC-IPS].

7.3.1 Data Integrity and Authentication

Data authentication and integrity is provided by a 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 [AESCBC].

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, in the future it may be desirable for an iSCSI implementation that operates at speeds of 1 Gbps or faster to implement the IPsec sequence number extension [SEQ-EXT].

7.3.2 Confidentiality

Confidentiality is provided by encrypting the data in every packet. When confidentiality is used it MUST be accompanied by data integ-

rity 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 [AESCTR] (NOTE: This is still subject to the IPsec WG's standardization plans).

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

7.3.3 Policy, Security Associations and Key Management

A compliant iSCSI implementation MUST meet the key management requirements of the IPsec protocol suite. Authentication, security association negotiation, and 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 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 pre-shared keys in many cases offer good security, situations where dynamically assigned addresses are used force the use of a group pre-shared key, which creates vulnerability to a man-in-the-middle attack.

- 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, ID_DER_ASN1_GN formats SHOULD NOT be used. The ID_KEY_ID Identity Payload MUST NOT be used.

Manual 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. However, as iSCSI has an in-band discovery mechanism (discovery session and SendTargets), the use or non-use of IPsec in any operational session is assumed to be identical to that of the discovery session.

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.

8. 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 RDMA 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 2.4.3 Consequences of the Model.

8.1 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 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.

8.1.1 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 presumption of "static and physical" does not apply. In any case, the model clauses (particularly, Section 2.4.2 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 2.4.3 Consequences of the Model) only prohibits its 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.

8.1.2 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.

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 Section 2.4.3 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 9.12.6 ISID and [NDT]) facilitates implementation of the ISID coordination by allowing each component vendor to independently (of other vendor's components) coordinate allocation and use and reuse 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 2.4.3 Consequences of the Model) at the initiator.

A vendor of iSCSI hardware (e.g., NICs or HBAs) intended for use in the initiators must allow, in addition to a mechanism for configuring the iSCSI Node Name, for 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.

8.2 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 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.

8.3 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 guidance the implementer may use an average Nop-Out/Nop-In turnaround delay multiplied by a "safety factor" (2-3) as a good estimate for the basic delay of the iSCSI stack for a given connection.

8.4 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 2.2.2.1 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.

8.5 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.

8.6 Considerations for State-dependent devices

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. As an 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 where 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 where 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 no swap was 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. For those commands a retry at SCSI level is not feasible or very difficult and error recovery at iSCSI level is advisable.

8.6.1 Determining the proper ErrorRecoveryLevel

The implementation and usage 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" frequency. 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 ErrorRecoveryLevel=2 when the connection failure is also of high likelihood during a backup/retrieval.

9. 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.

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.

9.1 iSCSI PDU Length and Padding

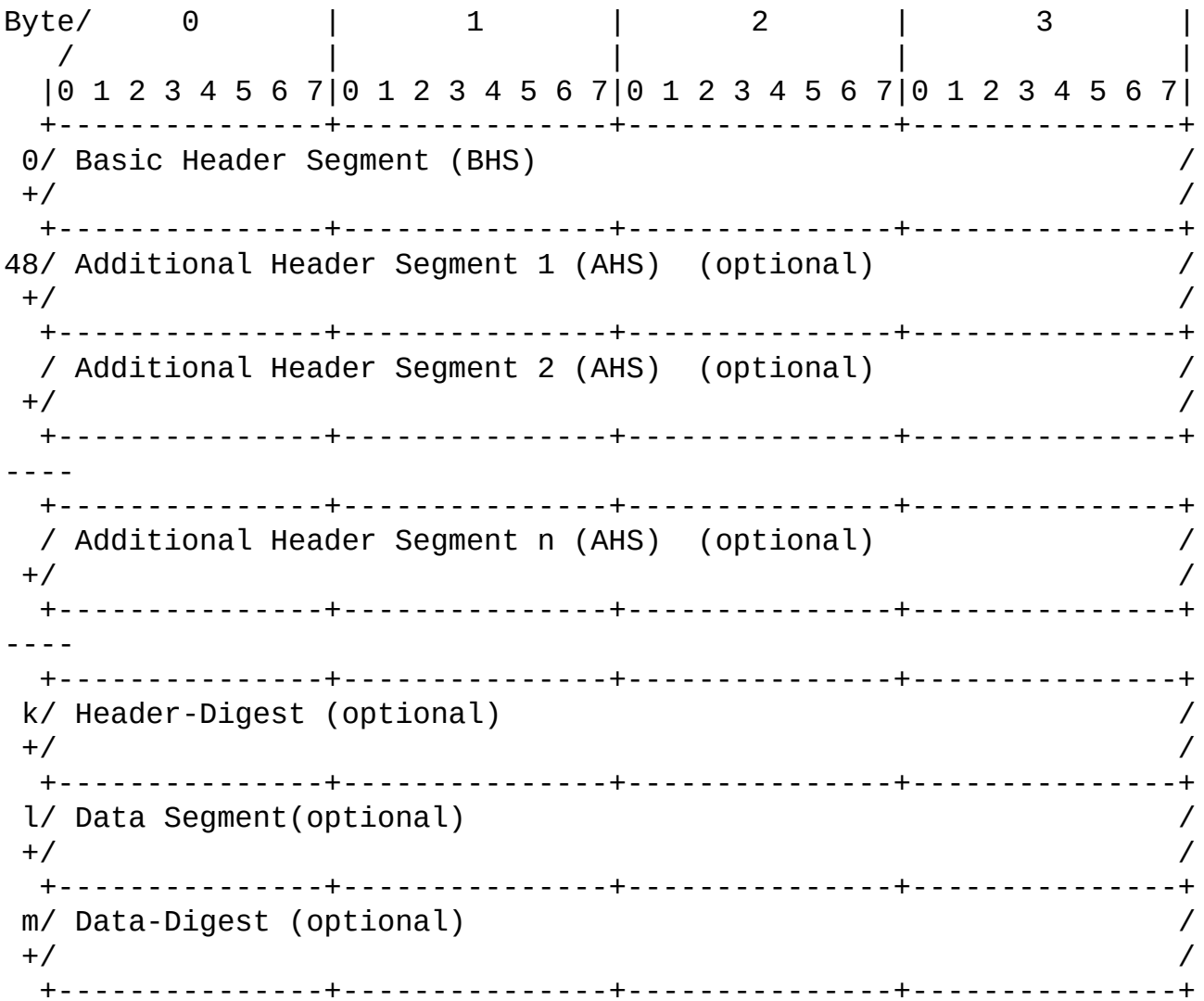
iSCSI PDUs are padded to the closest integer number of four byte words. The padding bytes SHOULD be 0.

9.2 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.

The overall structure of an iSCSI PDU is as follows:



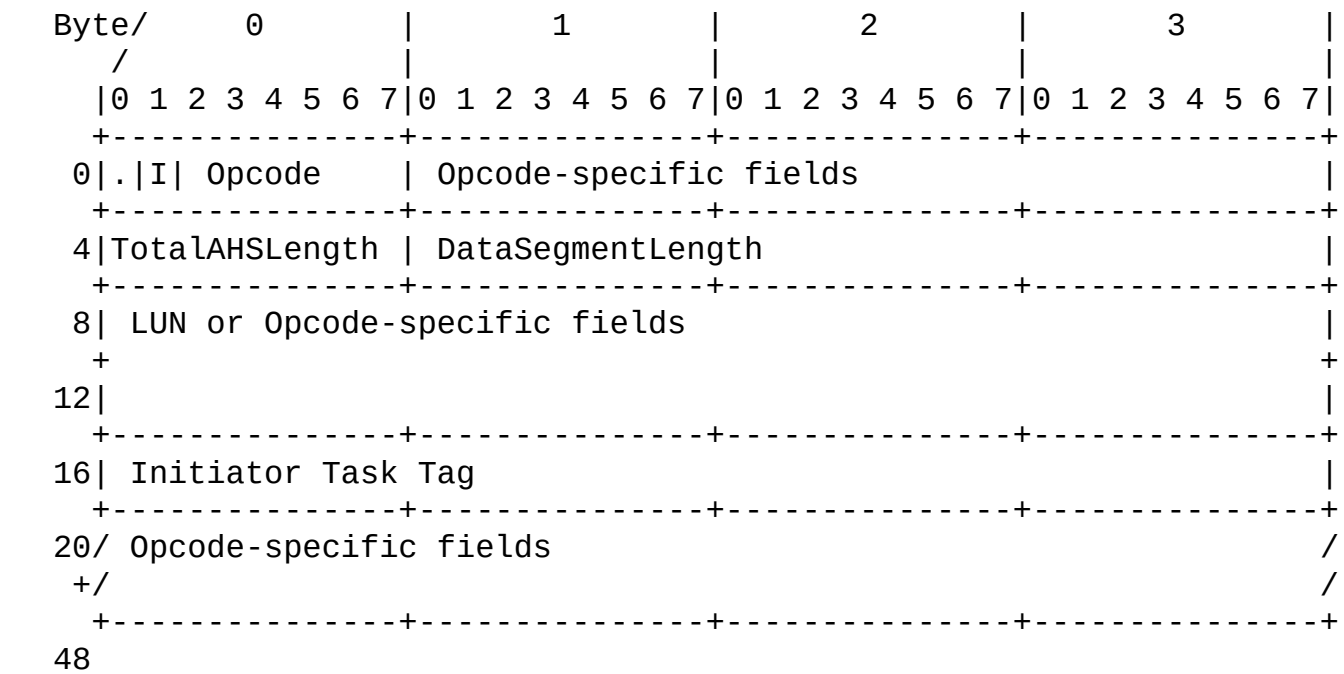
All PDU segments and digests are padded to the closest integer number of four byte words - i.e., all PDU segments and the digests start at a four byte word boundary and the padding ranges from 0 to 3 bytes. The padding bytes SHOULD be sent as 0.

iSCSI response PDUs do not have AH Segments.

9.2.1 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:



9.2.1.1 I

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

9.2.1.2 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 initiators (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.

9.2.1.3 Opcode-specific Fields

These fields have different meanings for different opcode types.

9.2.1.4 TotalAHSLength

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

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

9.2.1.5 DataSegmentLength

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

9.2.1.6 LUN

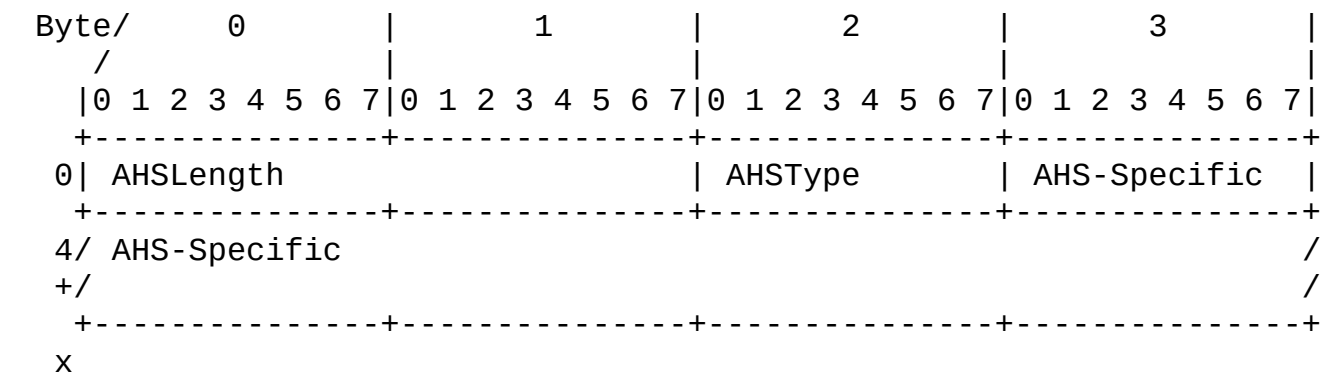
Some opcodes operate on a specific Logical Unit. The Logical Unit Number (LUN) field identifies which Logical Unit. If the opcode does not relate to a Logical Unit, this field is either ignored or may be used in an opcode specific way. The LUN field is 64-bits and should be formatted in accordance with [SAM2] i.e., LUN[0] from [SAM2] is BHS byte 8 and so on up to LUN[7] from [SAM2] that is BHS byte 15.

9.2.1.7 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 time span during which an iSCSI initiator task tag must be unique extends over the time span during which a SCSI task tag must be unique. However, the iSCSI Initiator Task Tag has to exist and be unique even for untagged SCSI commands.

9.2.2 Additional Header Segment (AHS)

The general format of an AHS is:



9.2.2.1 AHSType

The AHSType field is coded as follows:

bit 0-1 - Reserved

bit 2-7 - AHS code

0 - Reserved
1 - Extended CDB

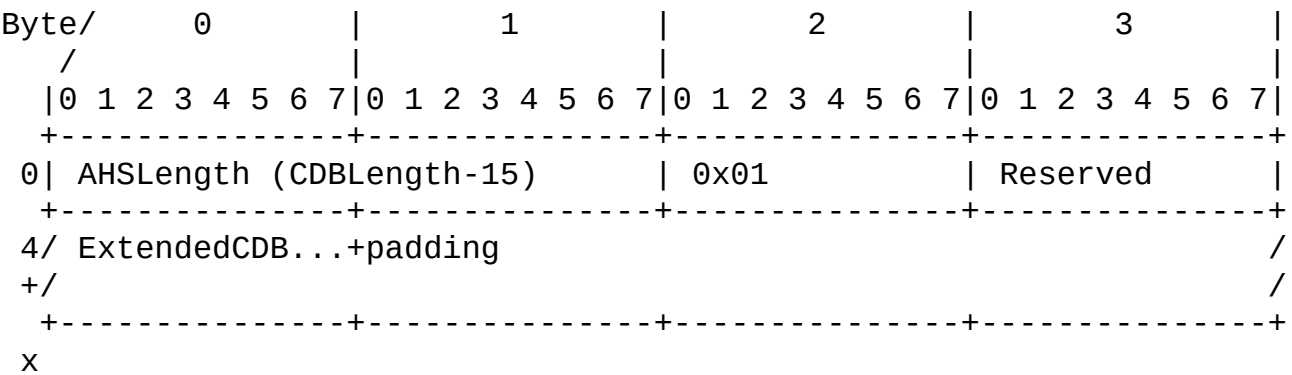
2 - Expected Bidirectional Read Data Length
3 - 59 Reserved
60- 63 Non-iSCSI extensions

9.2.2.2 AHSLength

This field contains the effective length in bytes of the AHS excluding AHSType and AHSLength (not including padding). The AHS is padded to the smallest integer number of 4 byte words (i.e., from 0 up to 3 padding bytes).

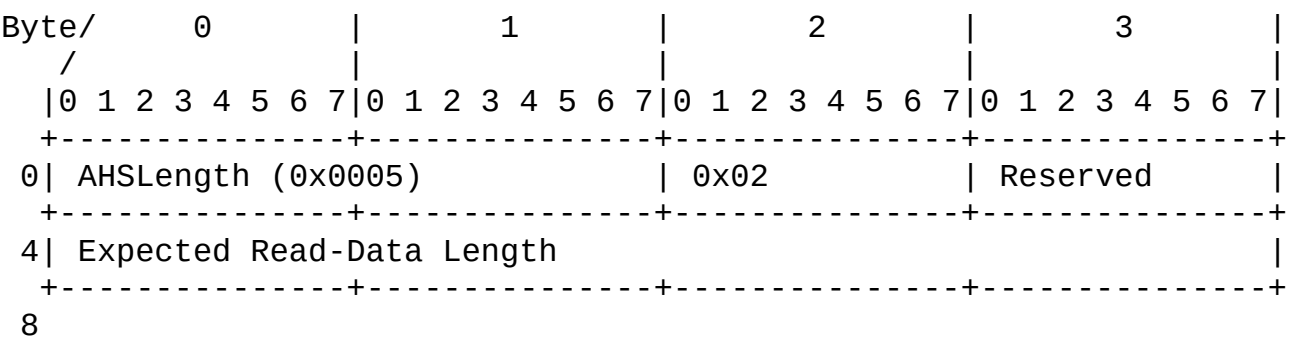
9.2.2.3 Extended CDB AHS

The format of the Extended CDB AHS is:



9.2.2.4 Bidirectional Expected Read-Data Length AHS

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



9.2.3 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 both the proper data as well as the padding bytes.

The digest types are negotiated during the Login Phase.

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

Digests are not included in data or header length fields.

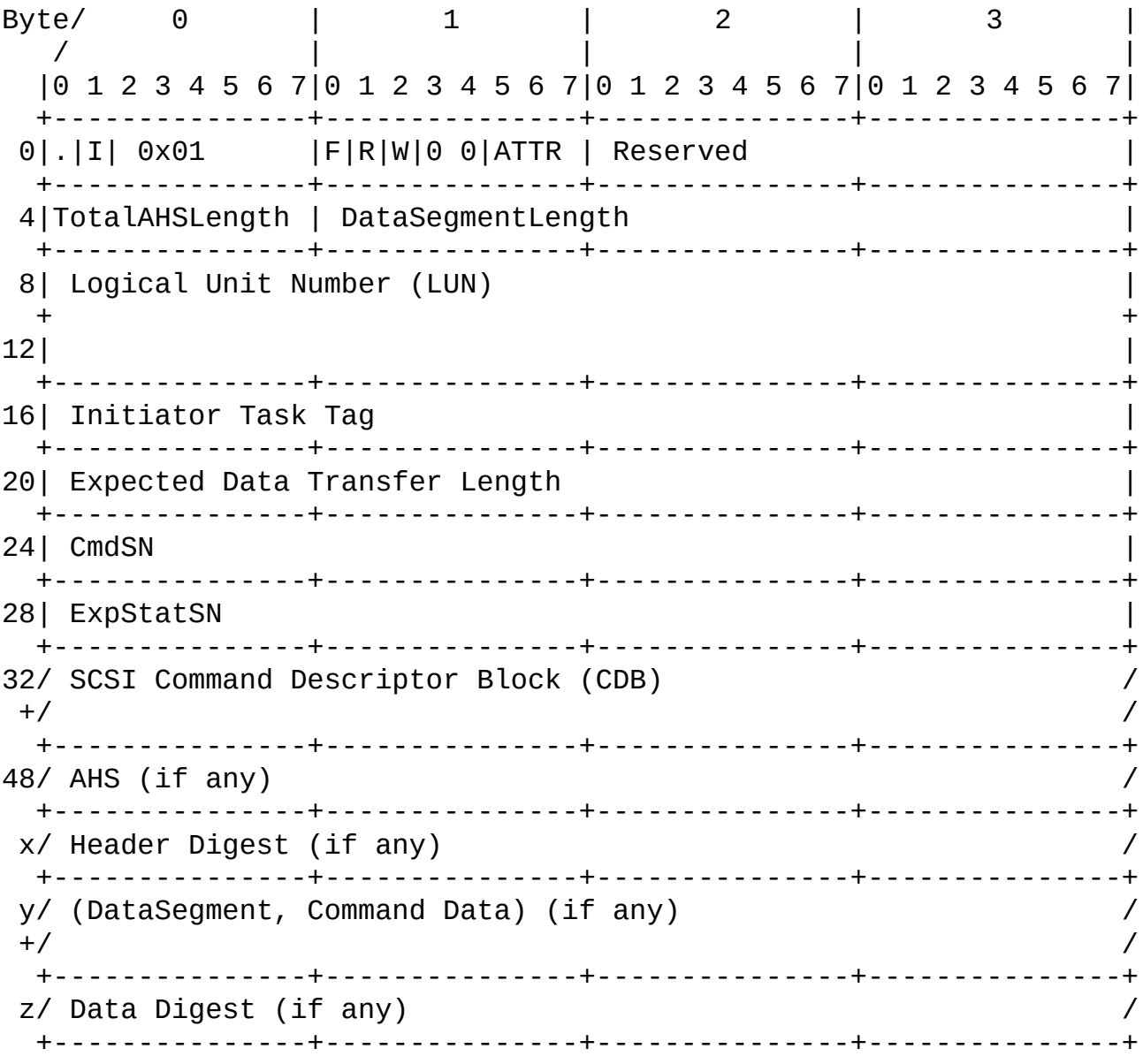
A zero-length Data Segment also implies a zero-length data-digest.

9.2.4 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.

9.3 SCSI Command

The format of the SCSI Command PDU is:



9.3.1 Flags and Task Attributes (byte 1)

The flags for a SCSI Command are:

bit 0 (F) set to 1 when no unsolicited SCSI Data-Out PDUs follow this PDU. For a write, if Expected Data Transfer

Length is larger than the DataSegmentLength the target may solicit additional data through R2T.

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

bit 2 (W) 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.
The R and W MAY both be 1 when the corresponding Expected Data Transfer Lengths are 0, but they CANNOT both be 0 when the corresponding Expected Data Transfer Length and Bidirectional Read Expected Data Transfer Length are not 0.

9.3.2 CmdSN - Command Sequence Number

Enables ordered delivery across multiple connections in a single session.

9.3.3 ExpStatSN

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

9.3.4 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 num-

ber 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 length of unsolicited data OR ONLY the immediate data.

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.

9.3.5 CDB - SCSI Command Descriptor Block

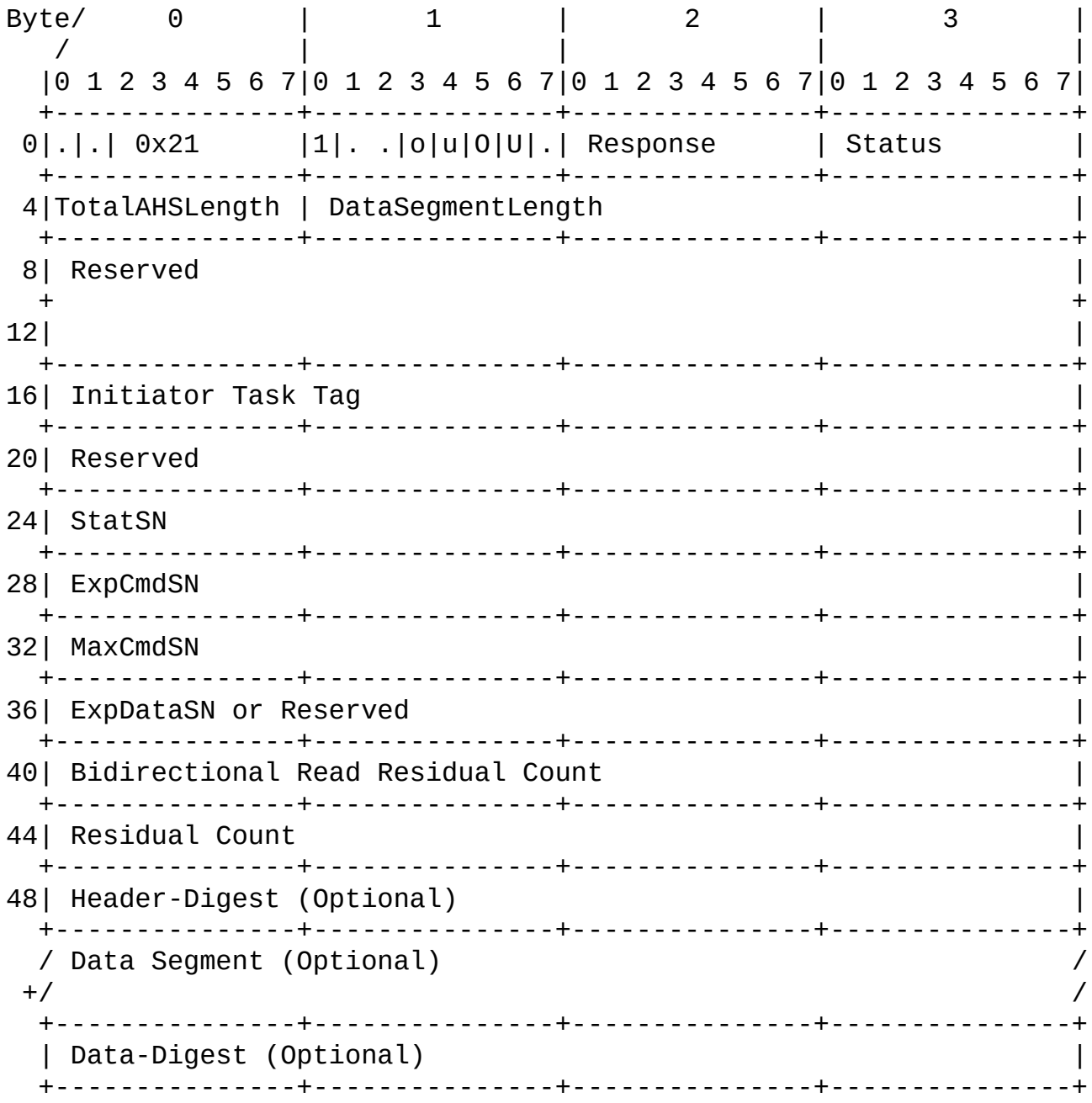
There are 16 bytes in the CDB field to accommodate the commonly used CDBs. Whenever the CDB is larger than 16 bytes, an Extended CDB AHS MUST be used to contain the CDB spillover.

9.3.6 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 (for example, from a WRITE operation) can be placed in the data segment (both cases are referred to as immediate data). These data are governed by the general rules for solicited vs. unsolicited data.

9.4 SCSI Response

The format of the SCSI Response PDU is:



9.4.1 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.
For a response other than "Command Completed at Target" bits 3-6 MUST be 0.

9.4.2 Status

The Status field is used to report the SCSI status of the command (as specified in [SAM2]) and is valid only 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.

9.4.3 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 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 SCSI Response PDU does not arrive before the session is terminated, the SCSI service response is SERVICE DELIVERY OR TARGET FAILURE.

As non-zero response field indicates a failure to execute the command in which case the Status and Sense fields are undefined.

9.4.4 Residual Count

The Residual Count field is only valid in the case where either the U bit or the O bit is set. If neither bit is set, the Residual Count field SHOULD be zero. 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.

9.4.5 Bidirectional Read Residual Count

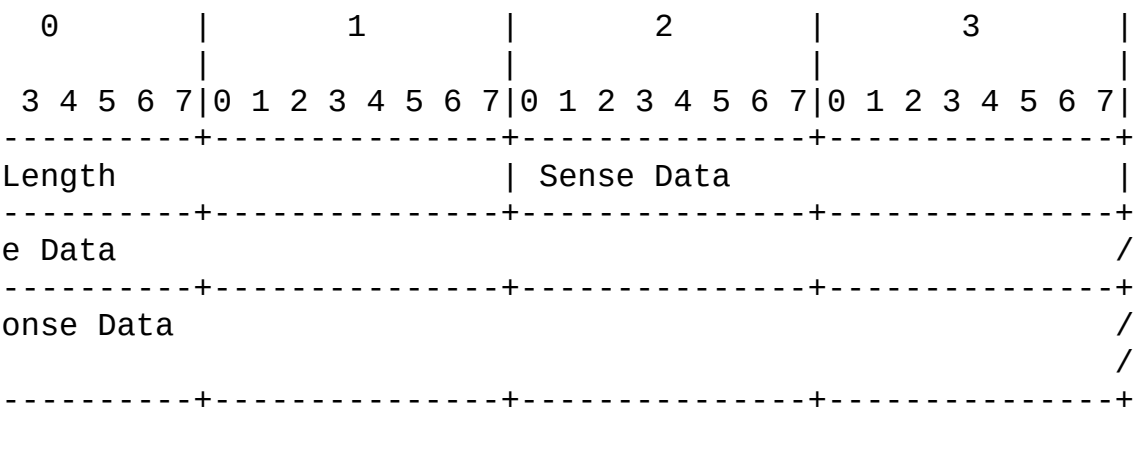
The Bidirectional Read Residual Count field is only valid in the case where either the u bit or the o bit is set. If neither bit is set, the Bidirectional Read Residual Count field SHOULD be zero. If the o bit is set, the Bidirectional Read Residual Count indicates the number of bytes that were not transferred 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.

9.4.6 Data Segment - Sense and Response Data Segment

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

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:



9.4.6.1 SenseLength

Length of Sense Data.

9.4.6.2 Sense Data

The Sense Data contain detailed information about a check condition and [SPC] 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.

9.4.7 ExpDataSN

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

This field is reserved if the response code is not Command Completed at Target or the command is a write command.

9.4.8 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.

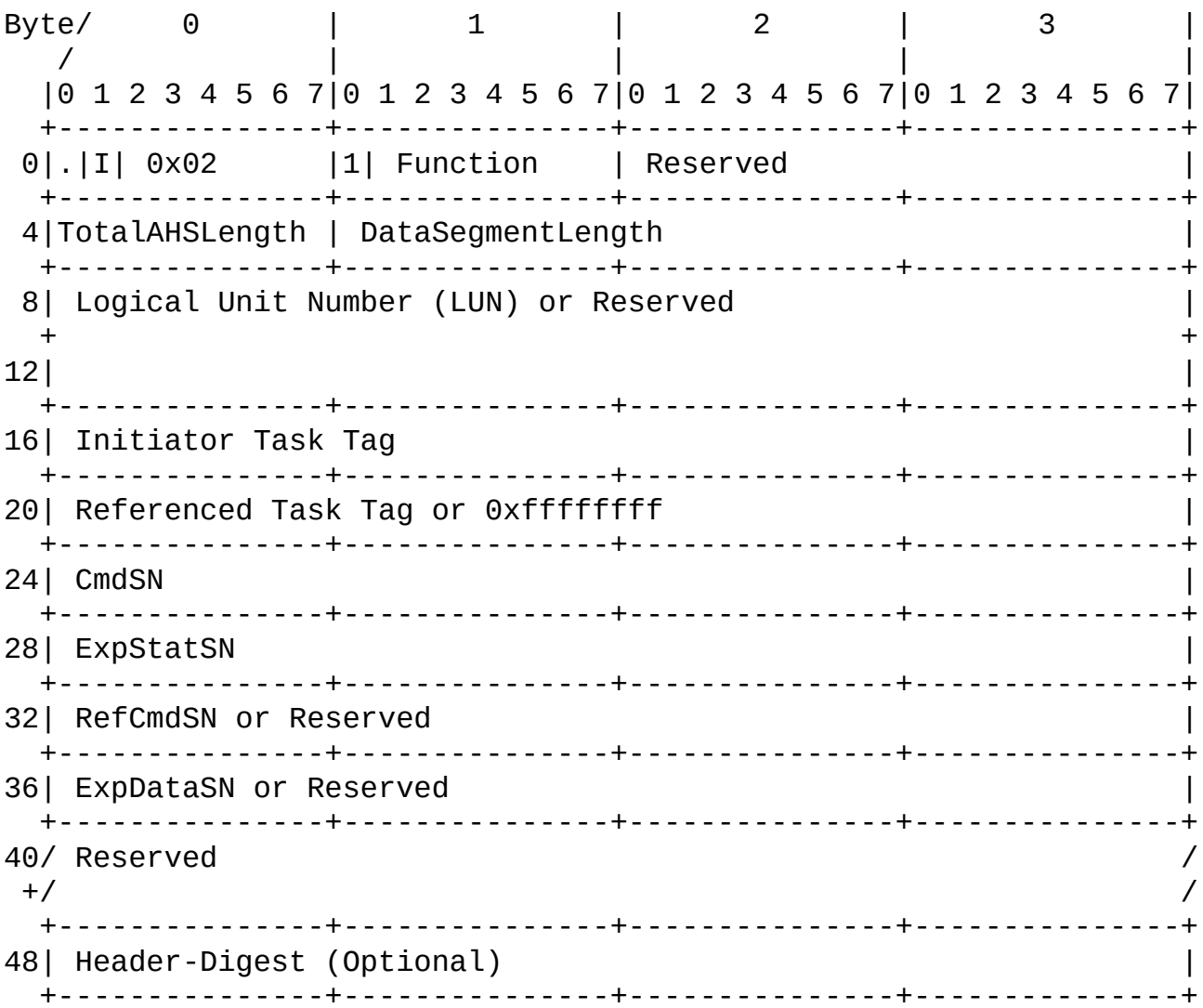
9.4.9 ExpCmdSN - Next Expected CmdSN from this Initiator

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

9.4.10 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 register 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.

9.5 Task Management Function Request



9.5.1 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 for the Logical Unit.

5 - LOGICAL UNIT RESET

6 - TARGET WARM RESET

7 - TARGET COLD RESET

8 - TASK REASSIGN - reassigns connection allegiance for the task identified by the Initiator 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 9.6 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 having a CmdSN lower than the task management 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 not higher than the task management command CmdSN), 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.

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

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 (section 6.12.1) if it is supporting ErrorRecoveryLevel >= 1, or alternatively may drop the connection to complete the requested task set function.

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).

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 function MAY also be subject to SCSI access controls (see [SPC3]) on the requesting initiator. When authorization fails at the target, the appropriate response as described in Section 9.6 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. 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.

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 Ser-

vice 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 be received by the target ONLY after the connection on which the command was previously executing has been successfully logged-out. For additional usage semantics see Section 6.1 Retry and Reassign in Recovery.

TASK REASSIGN MUST be issued as an immediate command.

9.5.2 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.

9.5.3 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.

9.5.4 RefCmdSN

For the ABORT TASK function, initiators MUST always set this to the CmdSN of the task identified by the Referenced Task Tag field. Targets must use this field as described in section 9.6.1 when the task identified by the Referenced Task Tag field is not with the target.

Otherwise this field is reserved.

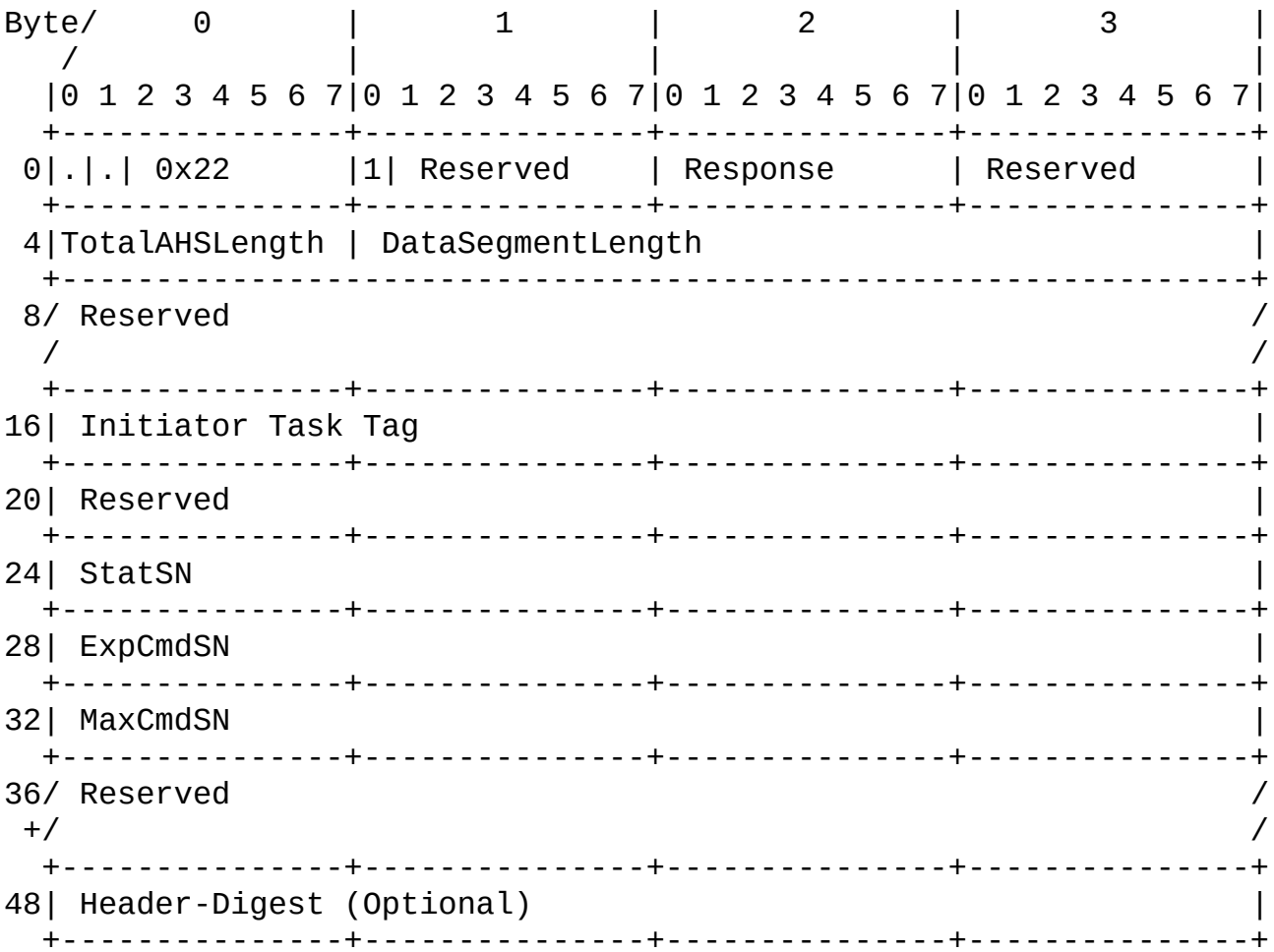
9.5.5 ExpDataSN

If the function is TASK REASSIGN, which establishes a new connection allegiance for a previously issued Read or Bidirectional command, this field will contain the next consecutive input DataSN number expected by the initiator (no gaps) for the referenced command in a previous execution. The initiator MUST discard any discontiguous data PDUs from the previous execution and the target MUST retransmit all data previously transmitted in Data-in PDUs (if any) starting with ExpDataSN. 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.

Otherwise, this field is reserved.

9.6 Task Management Function Response



For the functions ABORT TASK, ABORT TASK SET, CLEAR ACA, CLEAR TASK SET, LOGICAL UNIT RESET, TARGET COLD RESET and TARGET WARM RESET, the target performs the requested Task Management function and sends a Task Management Response back to the initiator.

9.6.1 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 failover 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 6.1.2 Allegiance Reassignment.

For the TARGET COLD RESET and TARGET WARM RESET functions, the target cancels all pending operations. 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 value 0 maps 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 be issued by the target only after all 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 Section 9.6.2 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, 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 (between MaxCmdSN and ExpCmdSN), targets must consider the CmdSN received and return the "Function complete" response.

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, targets must return the "Task does not exist" response.

9.6.2 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:

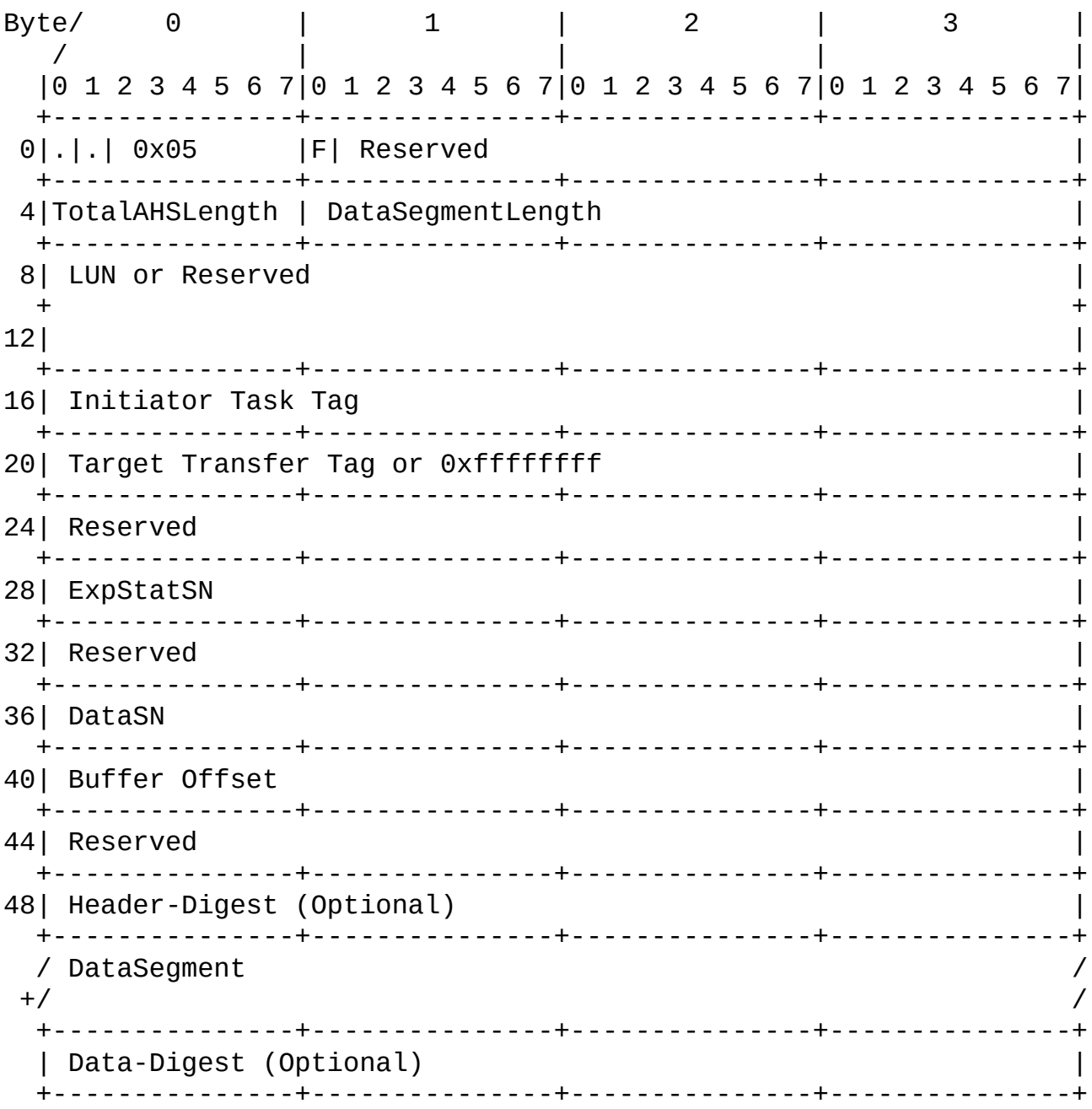
a) issues ABORT TASK SET/CLEAR TASK SET request.
b) continues to respond to each target transfer tag received for the affected task set.
c) receives any responses for the tasks in the affected task set (may process them as usual because they are guaranteed to be valid).
d) receives the task set management response, thus concluding all the tasks in the affected task set.

The target:

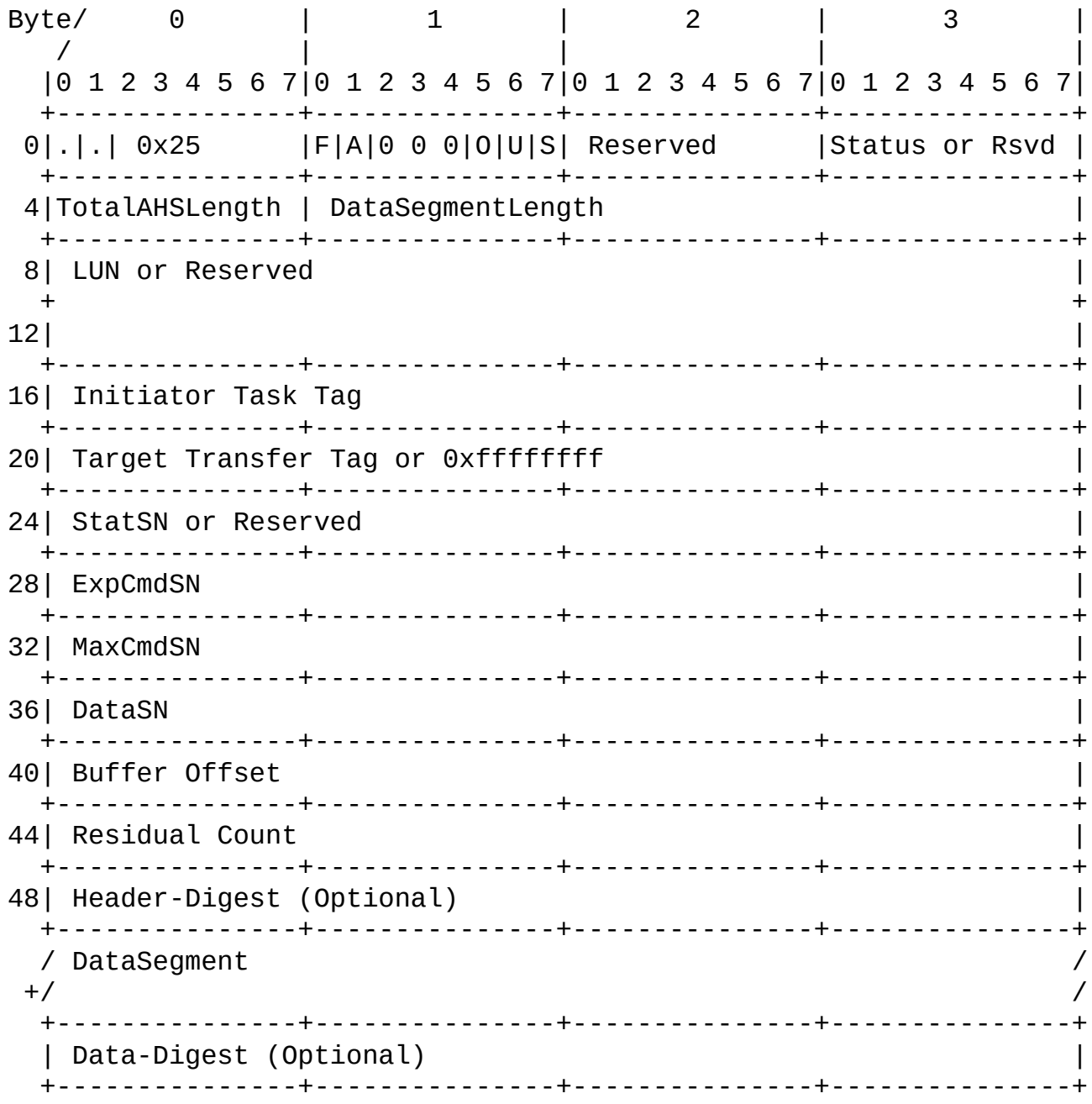
a) receives the ABORT TASK SET/CLEAR TASK SET request.
b) waits for all target transfer tags to be responded and also for all affected tasks in the task set to be received.
c) propagates the command up to and receives the response from the target SCSI layer.
d) takes note of last-sent StatSN on each of the connections in the session, and waits for acknowledgement of each StatSN (may solicit for acknowledgement by way of a NOP-In).
e) sends the task set management response.

9.7 SCSI Data-out & SCSI Data-in

The SCSI Data-out PDU for WRITE operations has the following format:



The SCSI Data-in PDU for READ operations has the following format:



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.

9.7.1 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 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 as well as the end of the output sequences.

9.7.2 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 set the A bit to 1 only 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 MUST NOT do so more frequently than this.

On receiving a Data-In PDU with the A bit set to 1, 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 this latter case (acknowledgement through ExpStatSN) sending a SNACK of type DataACK

in response to the A bit is not mandatory 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 until that Data-In PDU, it MUST 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.

9.7.3 Target Transfer Tag

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 MUST be provided by the target if the A bit is set to 1. The Target Transfer Tag and LUN are copied by the initiator in 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.

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.

9.7.4 StatSN

This field MUST ONLY be set if the S bit is set to 1.

9.7.5 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 2.2.2.3 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).

9.7.6 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.

9.7.7 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.

9.7.8 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. In this case, 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). For Bidirectional commands, the status MUST be sent in a SCSI Response PDU.

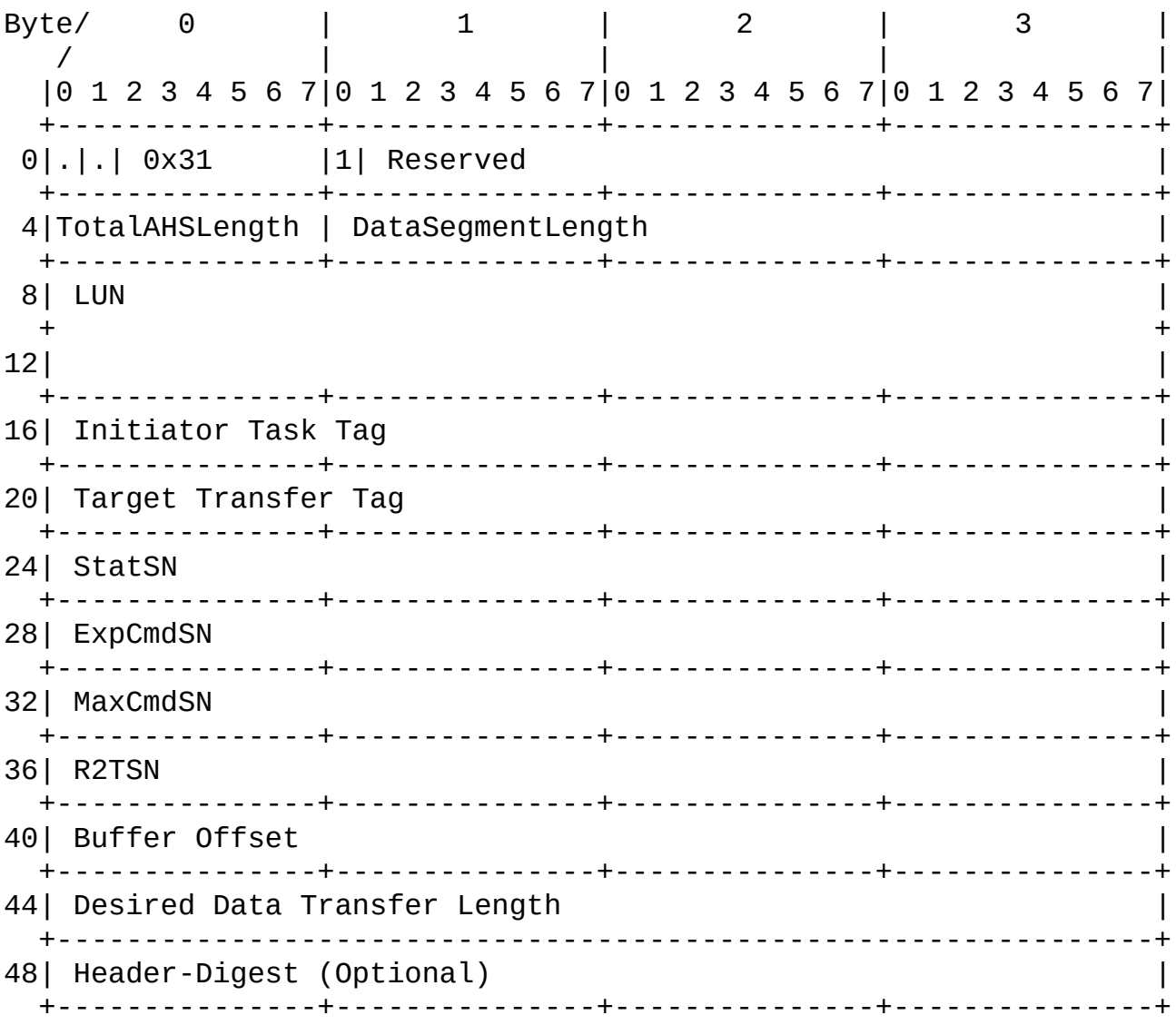
bit 2-3 - Reserved

bit 5-6 - used the same as in a SCSI Response. Those bits are valid only when S is set to 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 have meaningful content only if the S bit is set to 1 and their values are defined in Section 9.4 SCSI Response.

9.8 Ready To Transfer (R2T)



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 (up to a negotiated number). It, therefore, can have a number of pending data transfers. Within a connection, outstanding R2Ts MUST be fulfilled by the initiator in the order in which they were received.

DataSequenceInOrder governs the buffer offset ordering in consecutive R2Ts. If DataSequenceInOrder is Yes, then consecutive R2Ts SHOULD refer to continuous non-overlapping ranges.

9.8.1 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 2.2.2.3 Data Sequencing).

9.8.2 StatSN

The StatSN field will contain the next StatSN. The StatSN for this connection is not advanced.

9.8.3 Desired Data Transfer Length and Buffer Offset

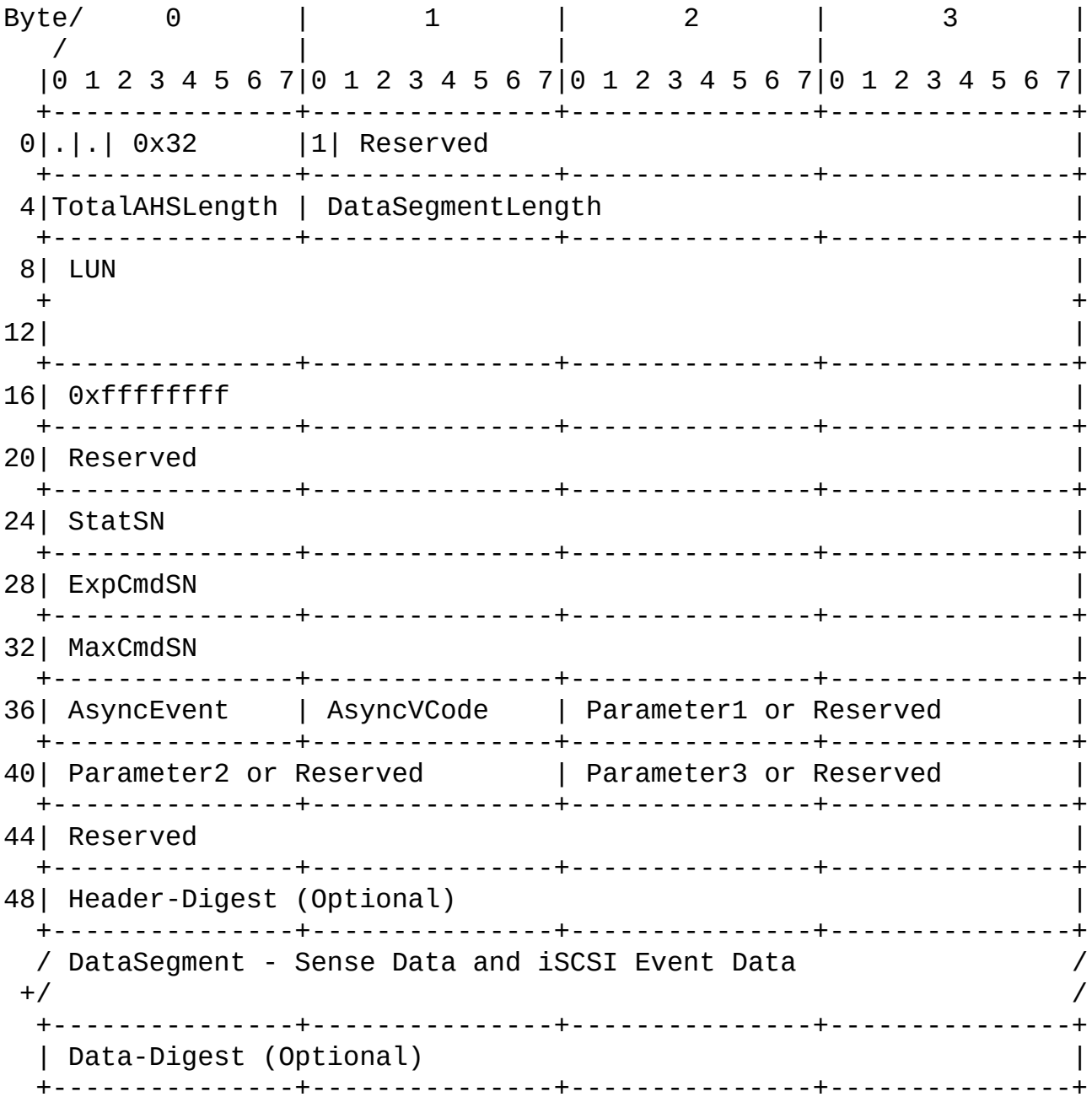
The target specifies how many bytes it wants the initiator to send because of this R2T PDU. The target may request the data from the initiator in several chunks, not necessarily in the original order of the data. The target, therefore, also specifies a Buffer Offset that indicates the point at which 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.

9.8.4 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 used by the target only. There is no protocol rule about the Target Transfer Tag except that the value 0xffffffff is reserved and MUST never be sent by a target in an R2T.

9.9 Asynchronous Message

An Asynchronous Message may be sent from the target to the initiator without corresponding to a particular command. The target specifies the reason for the event and sense data.



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.

9.9.1 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 [SPC]). Its use may be enabled by parameters in the SCSI Control mode page (see [SPC]).

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" (if not the only connection) OR "Close the session" to close all the connections (if using multiple connections). Once this message is received, the initiator SHOULD NOT issue new iSCSI commands. 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 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; no other

responses should be expected from the target for the outstanding commands on this connection in this case.
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.

9.9.2 AsyncVCode

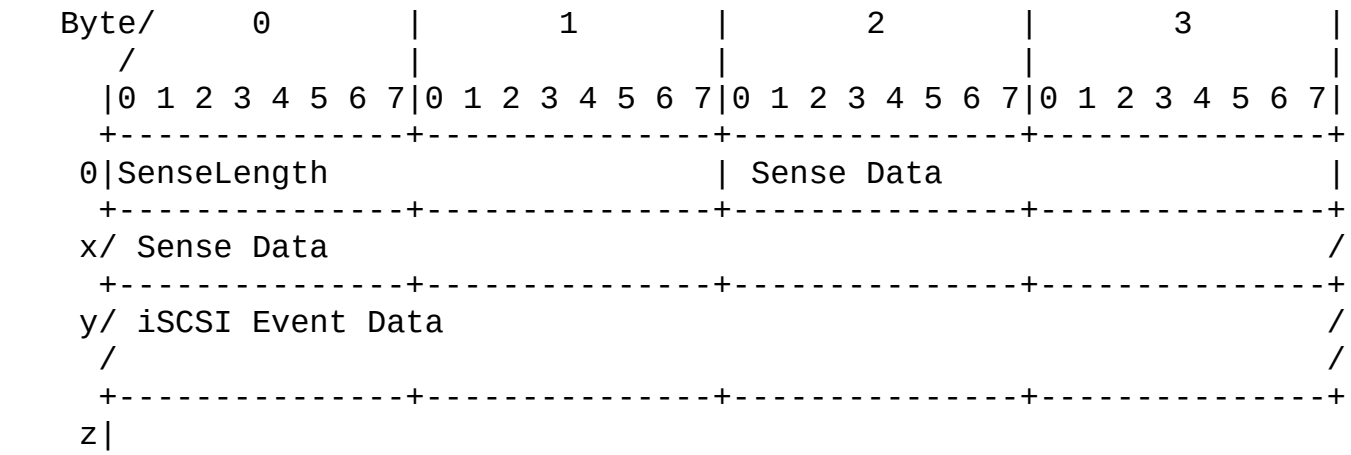
AsyncVCode is a vendor specific detail code that is valid only if the AsyncEvent field indicates a vendor specific event. Otherwise, it is reserved.

9.9.3 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:

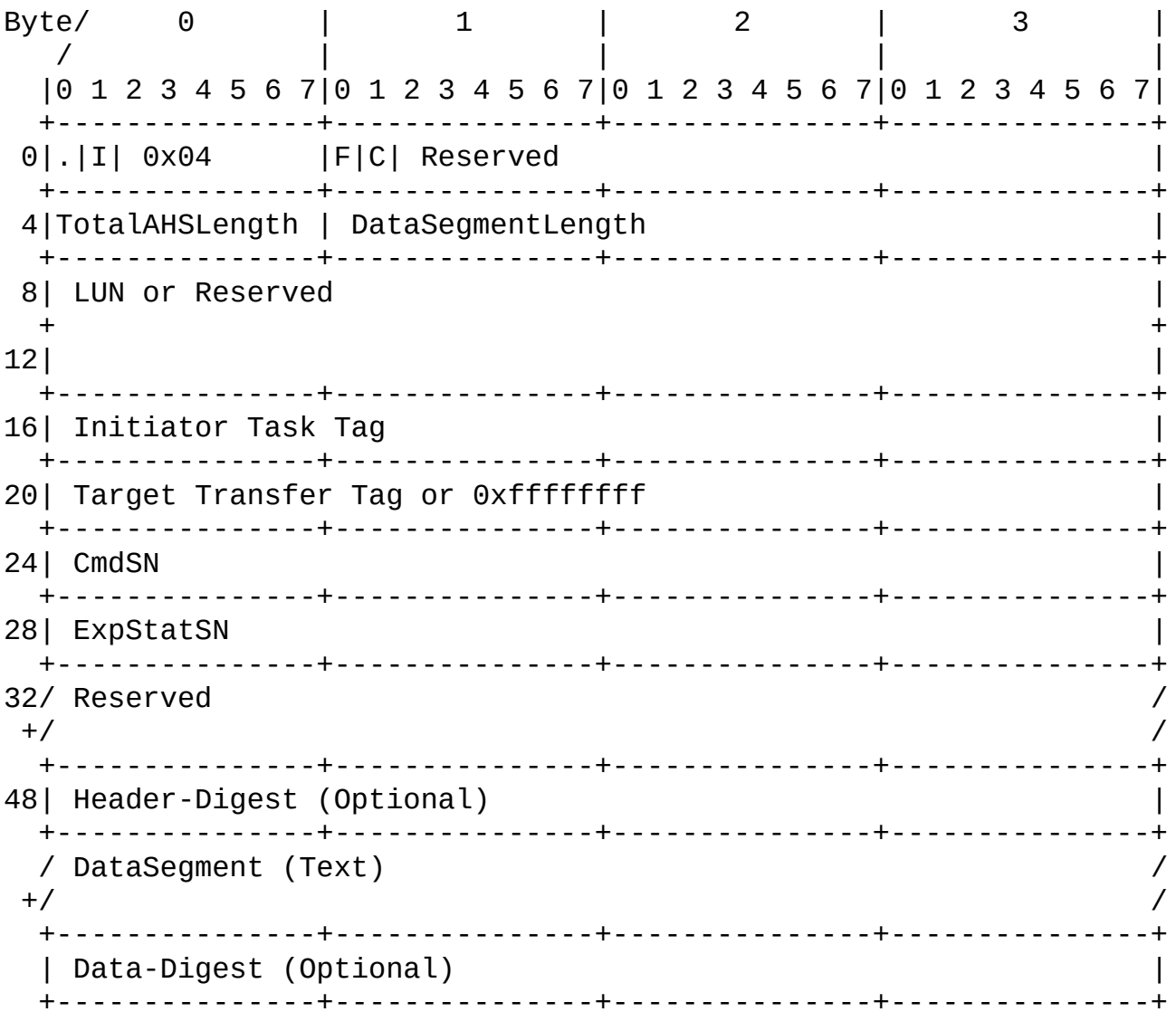


9.9.3.1 SenseLength

Length of Sense Data.

9.10 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.



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.

9.10.1 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.

9.10.2 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.

9.10.3 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.

9.10.4 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 the one sent by the target in the Text Response.

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)
```

9.10.5 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 4.2 Text Mode Negotiation.

Chapter 10 and Chapter 11 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 (i.e., 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.

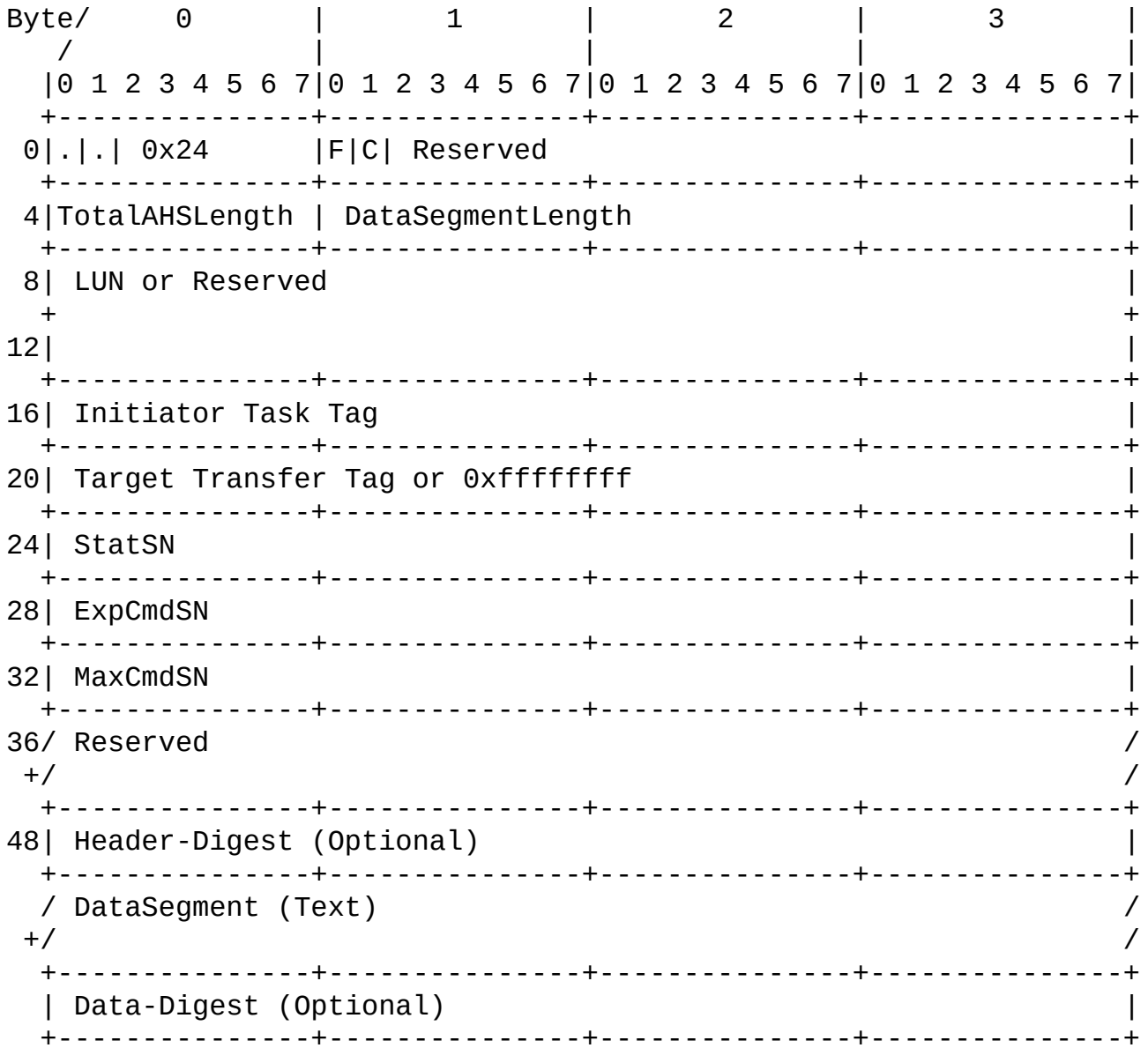
Chapter 4 details the rules for the Text Requests and Responses.

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.

9.11 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.



9.11.1 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.

9.11.2 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.

9.11.3 Initiator Task Tag

The Initiator Task Tag matches the tag used in the initial Text Request.

9.11.4 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.

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.

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.

9.11.5 StatSN

The target StatSN register is advanced by each Text Response.

9.11.6 Text Response Data

The data lengths of a text request 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 9.10.5 Text).

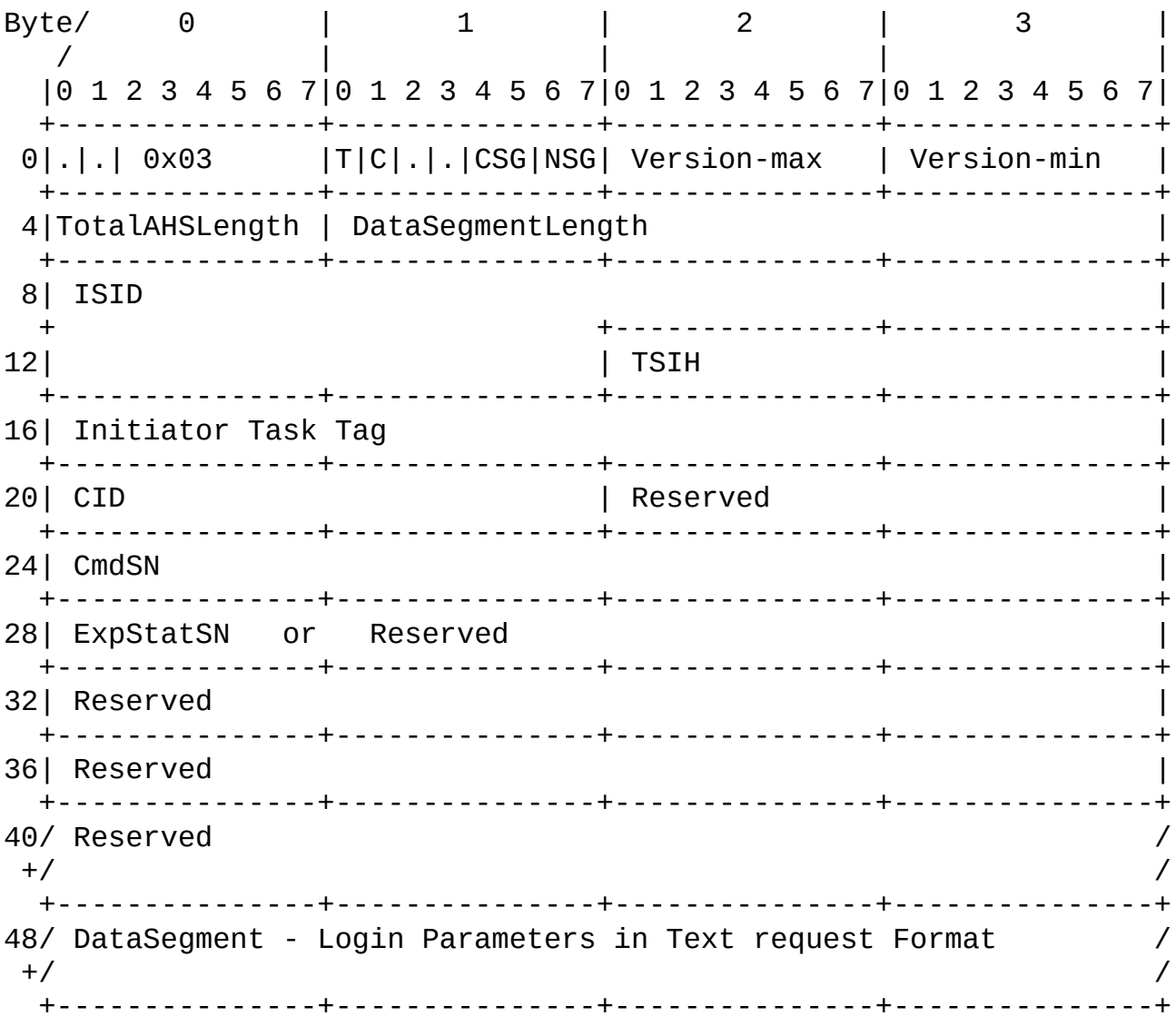
Although the initiator is the requesting party and controls the request-response initiation and termination, the target can offer key=value pairs of its own as part of a sequence and not only in response to the initiator.

9.12 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 4) consists of a sequence of Login requests and responses that carry the same Initiator Task Tag.

Login requests are always considered as immediate.



9.12.1 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 4).

9.12.2 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.

9.12.3 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 they want to move to (see Chapter 4). The next stage value is valid only when the T bit is 1; otherwise, it is reserved.

The stage codes are:

- 0 - SecurityNegotiation
- 1 - LoginOperationalNegotiation
- 3 - FullFeaturePhase

9.12.4 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.

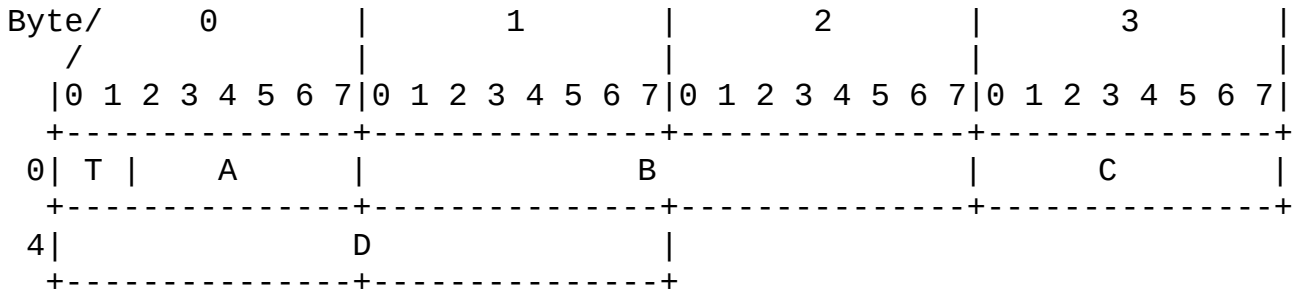
9.12.5 Version-min

Minimum Version supported. The version number of the current draft is 0x00.

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.

9.12.6 ISID

This is an initiator-defined component of the session identifier and is structured as follows (see [NDT] and Section 8.1.1 Conservative Reuse of ISIDs for details):



The T field identifies the format and usage of A, B, C & D as indicated bellow:

T

00b	OUI-format A&B are a 22 bit OUI (the I/G & U/L 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 24bit unsigned integer value in network byte order (byte big-endian). See [NDT] 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 2.4.3 Consequences of the Model and Section 8.1.1 Conservative Reuse of ISIDs).

The T field of 11 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 8.1.1 Conservative Reuse of ISIDs and Section 8.1.2 iSCSI Name, ISID and TPGT Use) and the resultant ISID MUST also be persistent over power cycles, reboot, card swap etc..

9.12.7 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 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 4.3.1 Login Phase Start.

9.12.8 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 4.3.4 Connection reinstatement).

9.12.9 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:

- A leading login phase - 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.

- A non-leading login phase - if the current CmdSN at the time the first login on the connection is issued is 500 - the login request carries CmdSN=500 the second login request carries a CmdSN not lower than 500 (higher if non-immediate requests where issued in the session between the first and the second request in the new login phase) etc..

If the login request is a leading login request the target MUST use the value presented in CmdSN as the target value for ExpCmdSN.

9.12.10 ExpStatSN

This is ExpStatSN for the old connection.

This field is valid only if the Login request restarts a connection (see Section 4.3.4 Connection reinstatement).

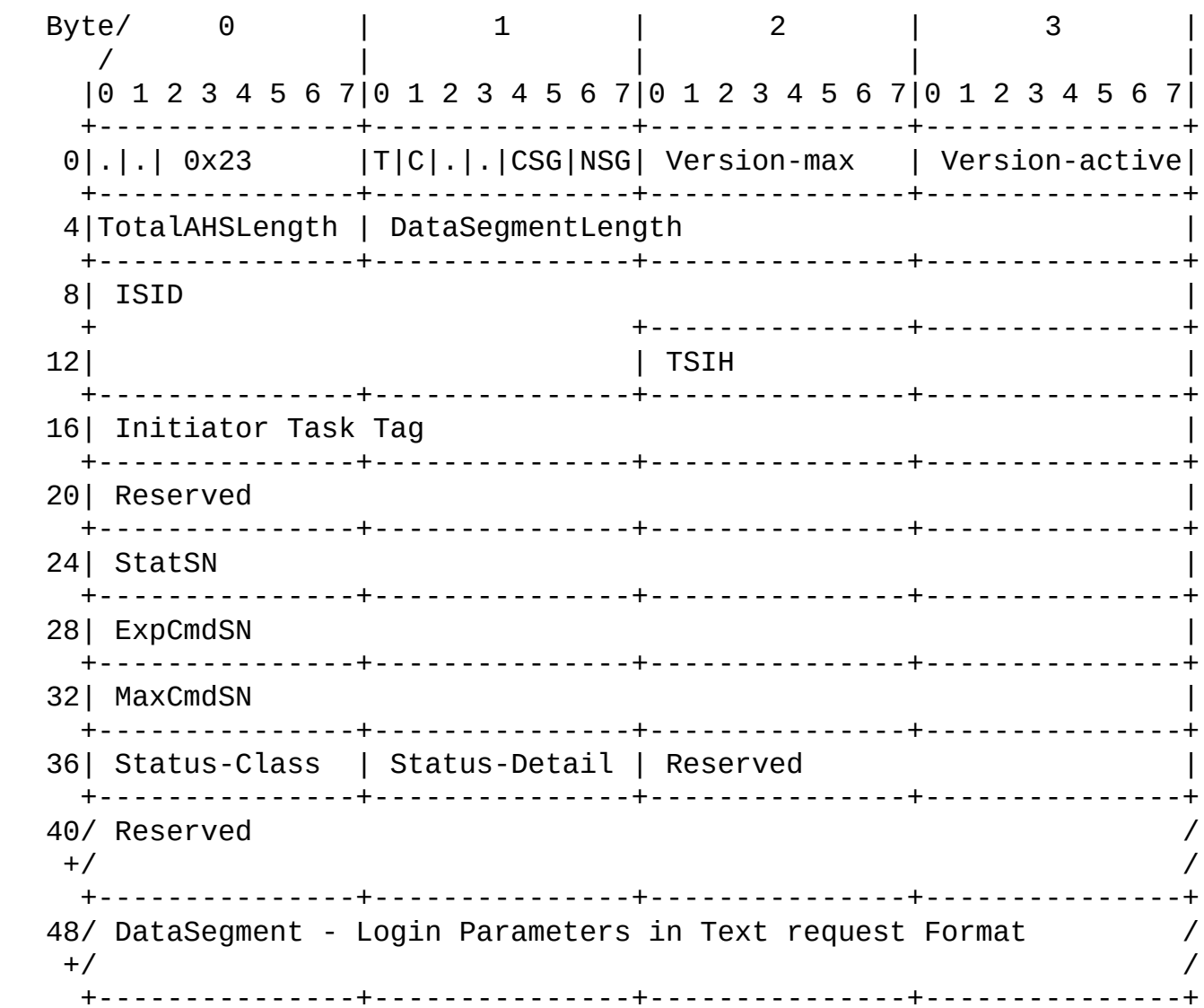
9.12.11 Login Parameters

The initiator MAY 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 9.10.5 Text for text requests/ responses also hold for login requests/responses. Keys and their explanations are listed in Chapter 10 (security negotiation keys) and Chapter 11 (operational parameter negotiation keys). All keys in Chapter 11, except for the X- extension format, MUST be supported by iSCSI initiators and targets. Keys in Chapter 10 only need to be supported when the function to which they refer is mandatory to implement.

9.13 Login Response

The Login Response indicates the progress and/or end of the Login Phase.



9.13.1 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.

9.13.2 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.

9.13.3 TSIH

The TSIH is the target assigned session identifying handle and its internal format and content are not defined by this protocol except for the value 0 that is reserved. For a new session, the target MUST generate a non-zero TSIH and return it in the Login Final-Response (see Section 4.3 Login Phase). In all other cases, this field should be set to the TSIH provided by the initiator in the Login Request.

9.13.4 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 valid only if the Status-Class is 0.

9.13.5 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, as well as better information for logging.

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 valid only 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.

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.
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 an 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 and the NSG is FullFeaturePhase in both the request and the 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.

9.13.6 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 4). 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.

9.13.7 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.

9.13.8 Login Parameters

The initiator MAY 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 9.11.5 StatSN for text requests/ responses also hold for login requests/responses. Keys and their explanations are listed in Chapter 10 (security negotiation keys) and Chapter 11 (operational parameter negotiation keys). All keys in Chapter 11, except for the X- extension format, MUST be supported by iSCSI initiators and targets. Keys in Chapter 10, only need to be supported when the function to which they refer is mandatory to implement.

9.14 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 PDU, an initiator MUST NOT send any new iSCSI requests on the closing connection. If the Logout 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 abort all pending commands, whether acknowledged 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 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.

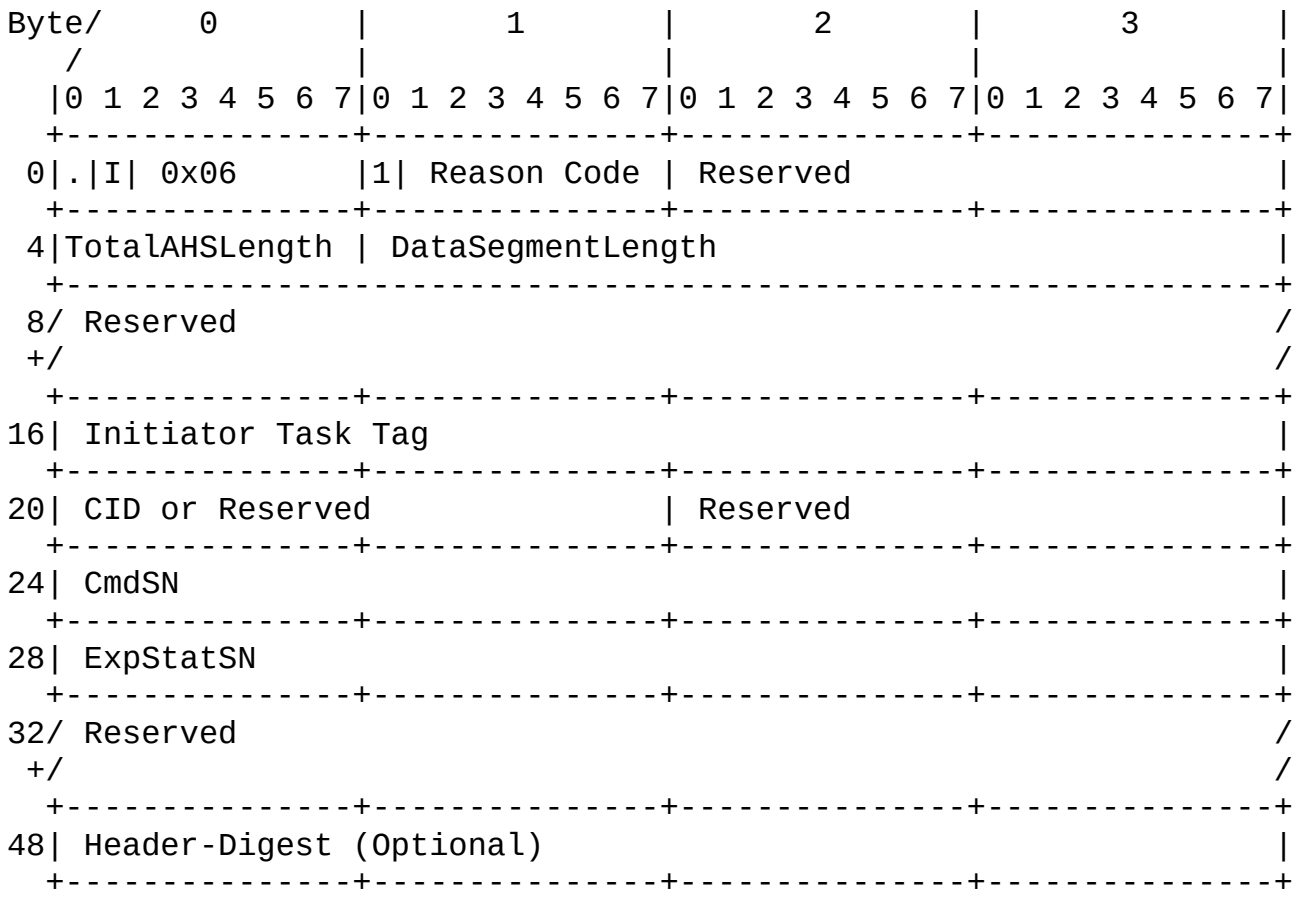
If an initiator intends to start recovery for a failing connection, it MUST use either the Logout request to "clean-up" the target end of a failing connection and enable recovery to start, or use the Login request with a non-zero TSIH and the same CID on a new connection for the same effect (see Section 9.14.2 CID). In sessions with a single

connection, the connection can be closed then a new connection reopened and a restart login can be used for recovery.

A successful completion of a logout request with the reason code of "close the connection" or "remove the connection for recovery" results in the discarding of all tasks waiting in the command reordering queue that are allegiant to the connection being logged out. Those holes 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 completely applicable also for an implicit Logout effected by way of a connection reinstatement or session reinstatement. The Logout reason codes for implicit Logout are specified as 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



9.14.1 Reason Code

Reason Code indicates the reason for Logout as follows:

0 - closes the session. All commands associated with the session (if any) are terminated.

1 - closes the connection. All commands associated with connection (if any) are terminated.

2 - removes 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.

9.14.2 CID

This is the connection ID of the connection to be closed (including closing the TCP stream). This field is valid only if the reason code is not "close the session".

9.14.3 ExpStatSN

This is the last ExpStatSN value for the connection to be closed.

9.14.4 Implicit termination of tasks

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

a) When a connection is implicitly or explicitly logged out with the Reason code of "Closes 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 5.2.2 State Transition Descriptions for Initiators and Targets) and there are active tasks allegiant to that connection.

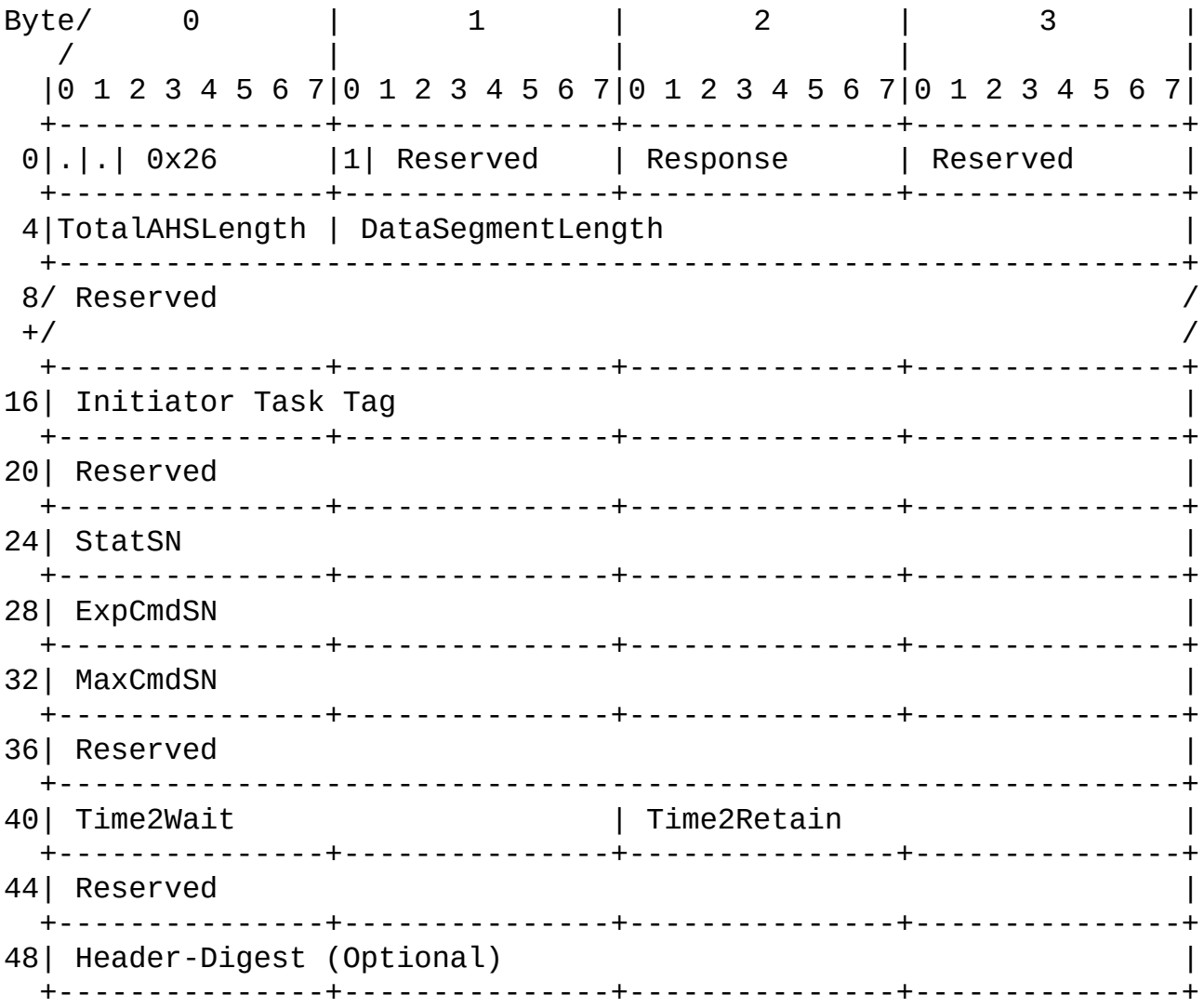
c) When a successful recovery Logout is performed while there are active tasks allegiant to that connection, and those tasks eventually time out after the Time2Wait and Time2Retain periods without allegiance reassignment.

If the tasks terminated in any of the above cases are SCSI tasks, they MUST be internally terminated with CHECK CONDITION status with a sense key of unit attention and ASC/ASCQ values of 0x6E/0x00 (COMMAND TO LOGICAL UNIT FAILED). Note that this status is meaningful only for appropriately handling the internal SCSI state aspects such as queued commands because this status is never communicated back as a terminating status to the initiator.

9.15 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).



9.15.1 Response

Logout response:

0 - connection or session closed successfully.

1 - CID not found.

2 - connection recovery 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.

9.15.2 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.

9.15.3 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.

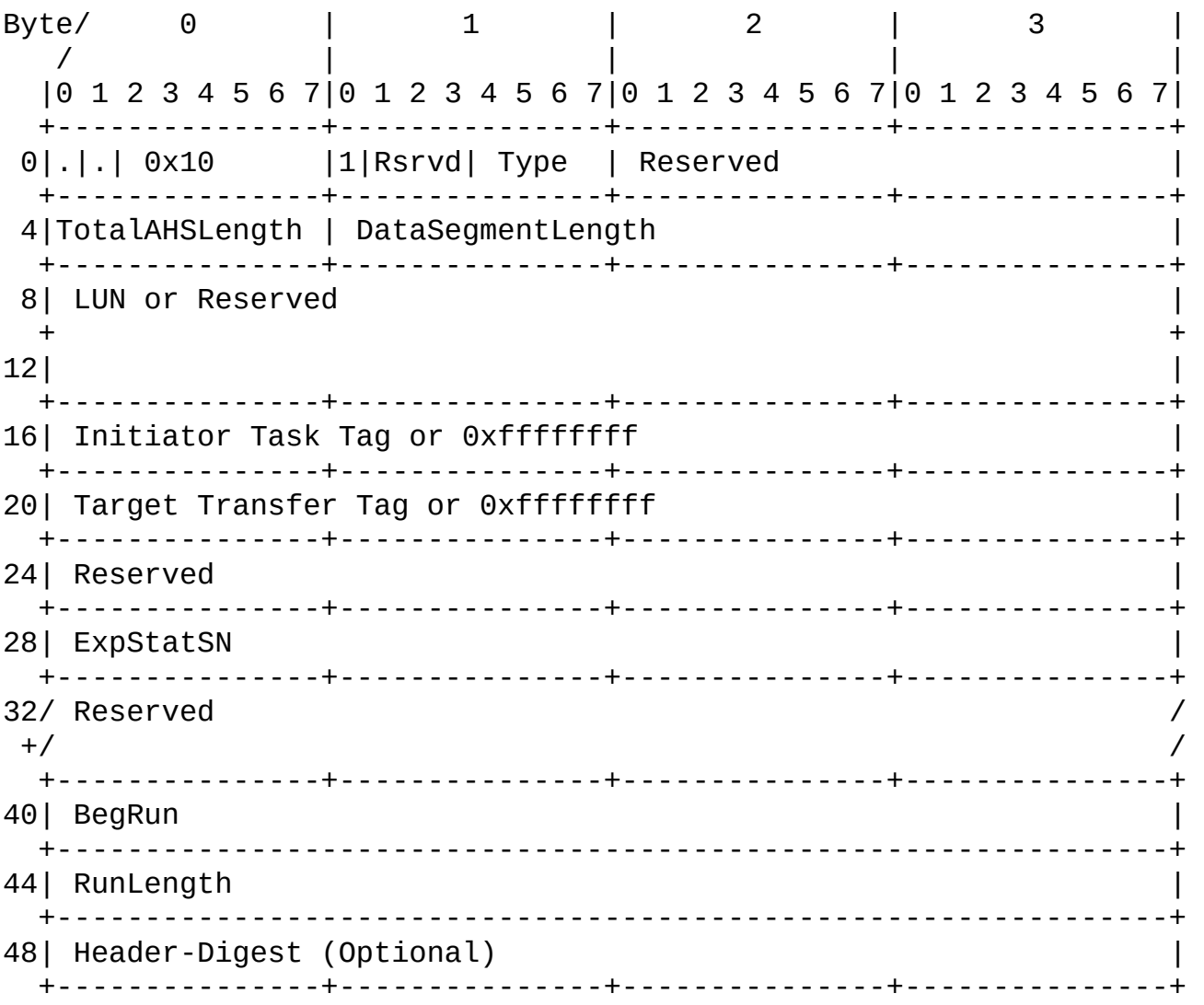
This field is invalid if the Logout response code is 1.

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 had already discarded the connection (and possibly the session) state along with the task states. No reassignment or Logout is required in this case.

9.16 SNACK Request



Support for SNACK is mandatory only if the supported ErrorRecovery-Level of the implementation is greater than zero.

The SNACK request is used to request the retransmission of numbered-responses, data, or R2T PDUs from the target. The SNACK request indicates the missed numbered-response or data "run" to the target, where the run starts with the first missed StatSN, DataSN, or R2TSN and indicates also the number of missed Status, Data, or R2T PDUs (0 has the special meaning of "all after the initial").

The numbered-response(s) or R2T(s), requested by a SNACK, MUST be delivered as exact replicas of the ones the initiator missed except for the fields ExpCmdSN, MaxCmdSN and ExpDataSN which MUST carry the current values. R2T(s)requested by SNACK MUST carry also the current value of StatSN.

The numbered Data-In PDUs, requested by a SNACK with a RunLength different from 0, MUST be delivered as exact replicas of the ones the initiator missed except the fields ExpCmdSN and MaxCmdSN which MUST carry the current values.

If the initiator MaxRecvDataSegmenTLength changed Data-In PDUs requested with RunLength 0 (meaning all PDUs after this number) may be different from the ones originally sent, in order to reflect changes in MaxRecvDataSegmentLength. Their DataSN starts with the requested number and is increased by 1 for each resent Data-In PDU. If DataSN numbers change and a SCSI-Reponse PDU was sent reflecting the DataSN before retransmission it MUST be resent to reflect the new numbers.

Any SNACK that requests a numbered-response, Data, or R2T that was not sent by the target MUST be rejected with a reason code of "Protocol error".

9.16.1 Type

This field encodes the SNACK function as follows:

0-Data/R2T SNACK - requesting retransmission of a Data-In or R2T PDU.

1-Status SNACK - requesting retransmission of a numbered response.

2-DataACK - positively acknowledges Data-In PDUs.

All other values are reserved.

Data/R2T SNACK for a command MUST precede status acknowledgement for the given command.

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.

For DataACK, the Target Transfer Tag has to 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.

An iSCSI target that does not support recovery within connection MAY reject the status SNACK with a Reject PDU. If the target supports recovery within connection, it MAY reject the SNACK after which it MUST issue an Asynchronous Message PDU with an iSCSI event that indicates "Request Logout".

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.

9.16.2 BegRun

The first missed DataSN, R2TSN, or StatSN or the next expected DataSN for a DataACK type SNACK request.

9.16.3 RunLength

The number of sequential missed DataSN, R2TSN or StatSN.

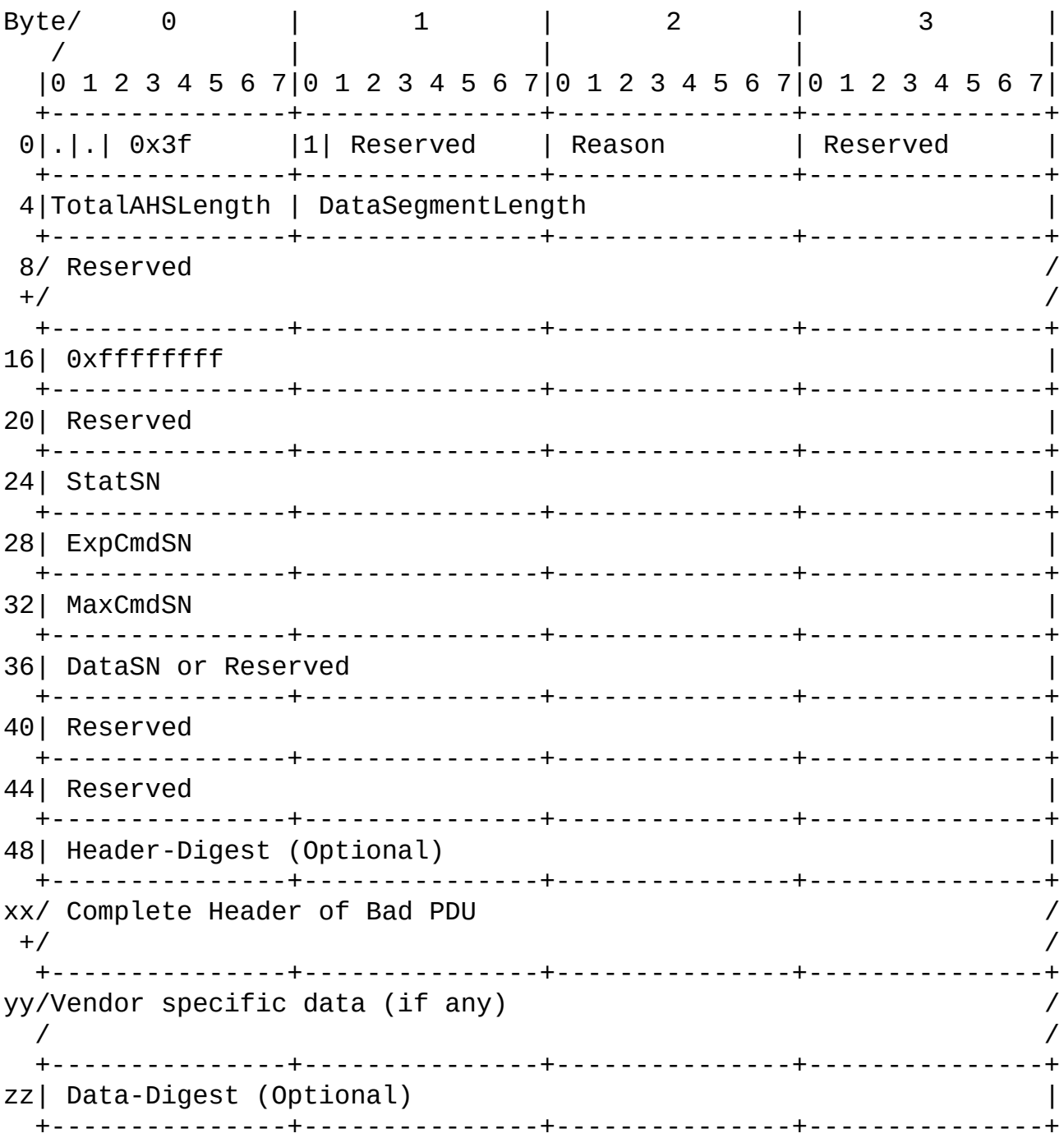
RunLength of "0" signals that all Data-In, R2T or Response PDUs carrying the numbers equal to or greater than BegRun have to be resent.

The RunLength MUST also be 0 for a DataACK SNACK.

The first data SNACK issued after initiator's MaxRecvDataSegmentLength decreased, for a command issued on the same connection before the change in MaxRecvDataSegmentLength, MUST use RunLength "0" to request retransmission of any number of PDUs (including one). The number of retransmitted PDUs in this case may or may not be the same

as the original transmission, depending on whether loss was before or after the MaxRecvDataSegmentLength was changed at the target.

9.17 Reject



Reject is used to indicate an iSCSI error condition (protocol, unsupported option etc.).

9.17.1 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 done only 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, an invalid sequence number in a SNACK.

All other values for Reason are reserved.

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 9.4.3 Response. In those 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 (the command PDU did not contain all the data and the target has not received a Data-out PDU with the Final bit 1 for the unsolicted 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 6.2 Usage Of Reject PDU in Recovery.

9.17.2 DataSN

This field is valid only if the Reason code is "Protocol error" and the SNACK was a Data/R2T SNACK. The DataSN/R2TSN is the last valid sequence number that the target sent for the task.

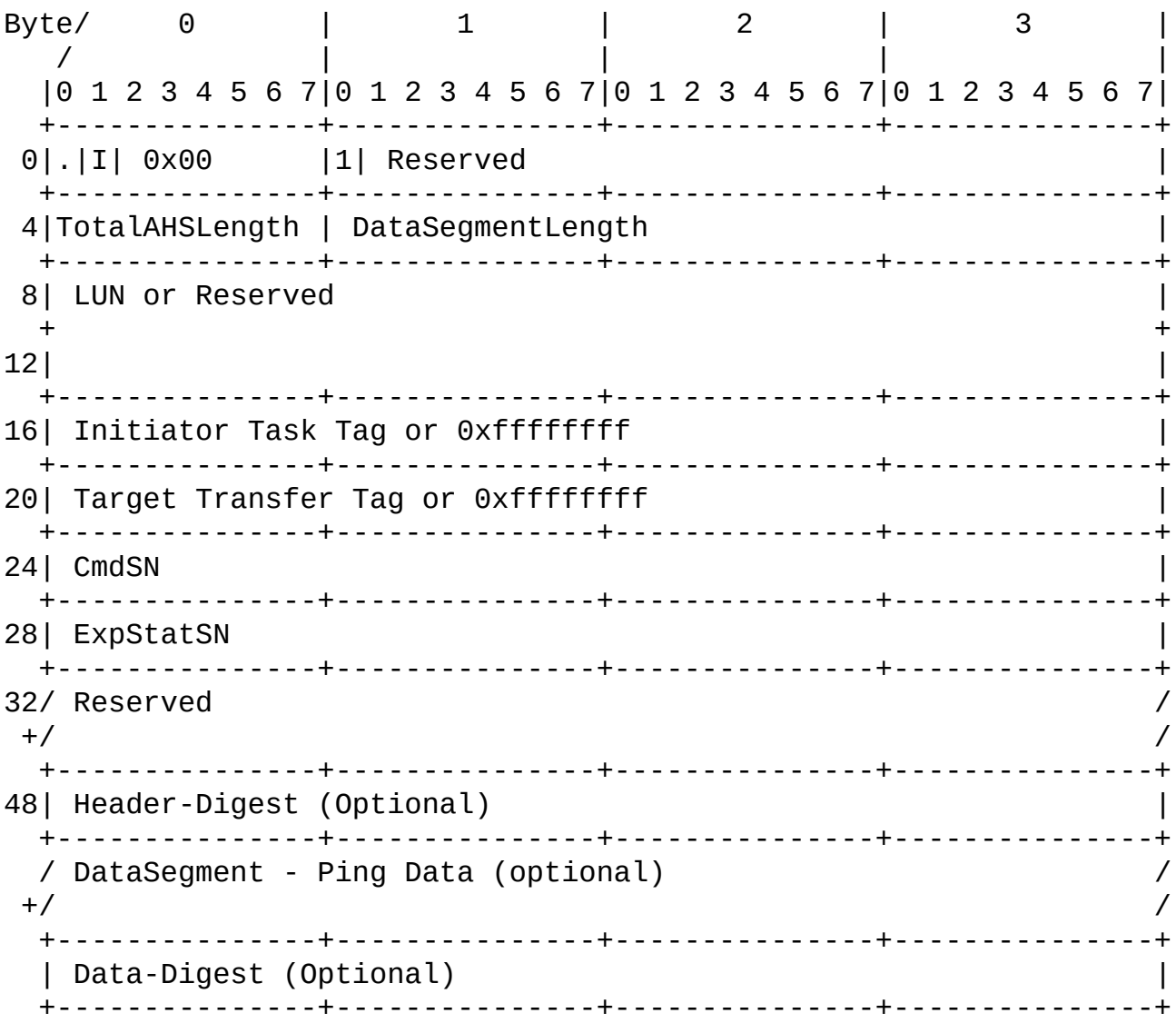
9.17.3 StatSN, ExpCmdSN and MaxCmdSN

Those fields carry their usual values and are not related to the rejected command

9.17.4 Complete Header of Bad PDU

The target returns the header (not including digest) of the PDU in error as the data of the response.

9.18 NOP-Out



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.

When used as a ping request, the Initiator Task Tag MUST be set to a valid value (not the reserved 0xffffffff).

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.

When a target receives the NOP-Out with a valid Initiator Task Tag, it MUST respond with a Nop-In Response (see NOP-In).

9.18.1 Initiator Task Tag

An initiator assigned identifier for the operation.

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.

If the Initiator Task Tag contains 0xffffffff, the CmdSN field contains the next CmdSN. However, CmdSN is not advanced and the I bit must be set to 1.

9.18.2 Target Transfer Tag

A target assigned identifier for the operation.

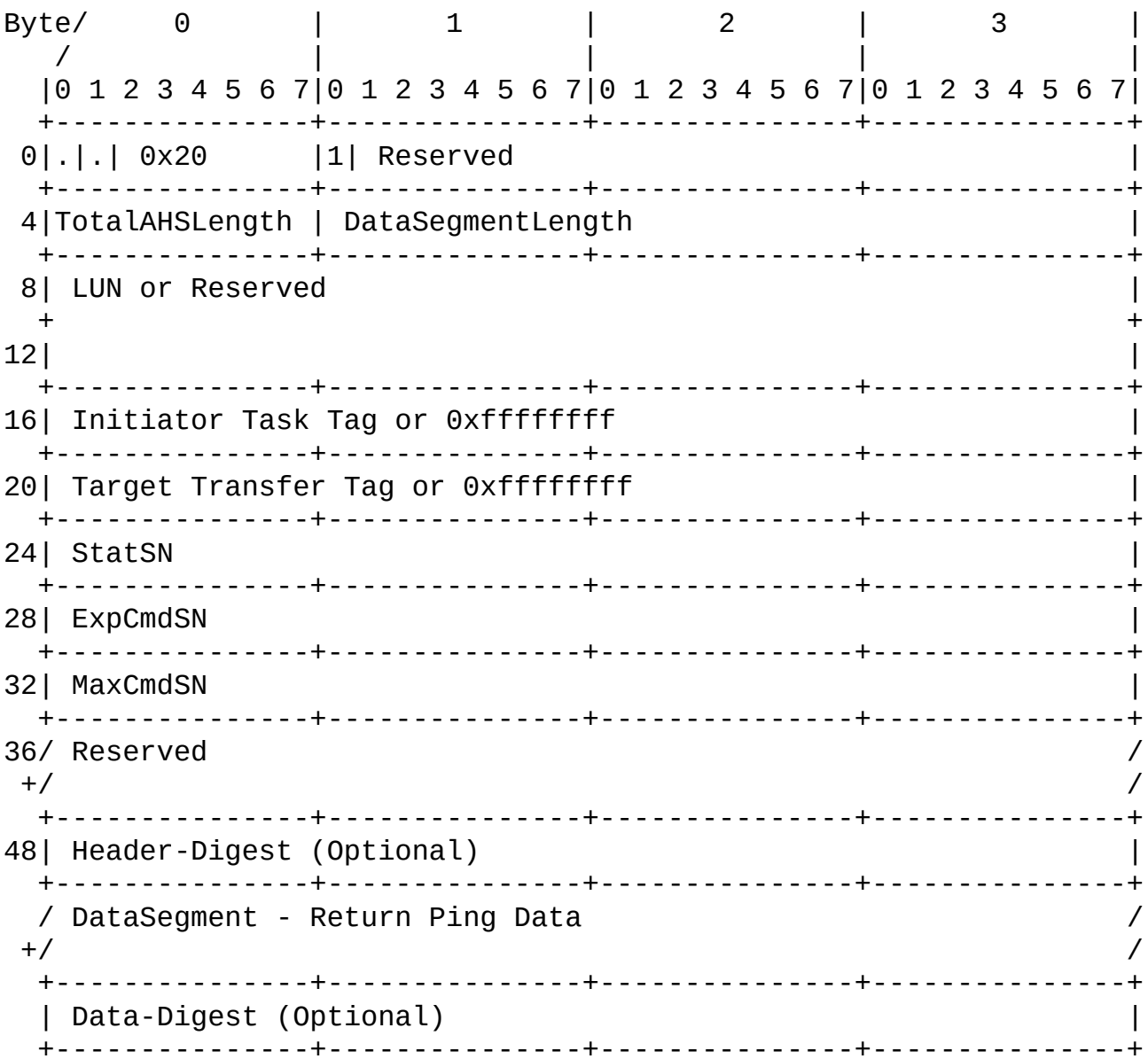
The NOP-Out MUST have the Target Transfer Tag set only if it is issued in response to a NOP-In with a valid Target Transfer Tag. In this case, it copies the Target Transfer Tag from the NOP-In PDU.

When the Target Transfer Tag is set, the LUN field MUST also be copied from the NOP-In.

9.18.3 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 Data Segment Length and indicates the absence of ping data.

9.19 NOP-In



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.

When a target sends a NOP-In with the Initiator Task Tag set to 0xffffffff) it MUST NOT send any data in the data segment (DataSegmentLength MUST be 0).

9.19.1 Target Transfer Tag

A target assigned identifier for the operation.

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.

9.19.2 StatSN

The StatSN field will contain always the next StatSN. However, when the Initiator Task Tag is set to 0xffffffff StatSN for the connection is not advanced.

9.19.3 LUN

A LUN MUST be set to a correct value when the Target Transfer Tag is valid (not the reserved value 0xffffffff).

10. iSCSI Security Keys and Authentication Methods

Only the following keys can be used during the SecurityNegotiation stage of the Login Phase:

SessionType
InitiatorName
TargetName
InitiatorAlias
TargetAlias
TargetPortalGroupTag
AuthMethod and all keys listed under AuthMethod along with all of their associated keys.

SessionType, InitiatorName, TargetName, InitiatorAlias, TargetAlias and TargetPortalGroupTag are described in Chapter 11 as they can be used also in the OperationalNegotiation stage.

All security keys have connection-wide applicability.

10.1 AuthMethod

Use: During Login - Security Negotiation
Senders: Initiator and Target
Scope: connection

AuthMethod = <list-of-options>

The main item of security negotiation is the authentication method (AuthMethod).

The authentication methods that can be used (appear in the list-of-options) 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 [RFC1944]
None	No authentication

The AuthMethod selection is followed by an "authentication exchange" specific to the authentication method selected.

The authentication exchange authenticates the initiator to the target, and optionally, the target to the initiator. Authentication is not mandatory to use but must be supported by the target and initiator.

The initiator and target MUST implement CHAP.

10.2 Kerberos

For KRB5 (Kerberos V5) [RFC1510], the initiator MUST use:

KRB_AP_REQ=<KRB_AP_REQ>

where KRB_AP_REQ is the client message as defined in [RFC1510].

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 large-binary-values and their binary length (not the length of the character string that represents them in encoded form) MUST not exceed 65536 bytes.

10.3 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>

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.

All the SPKM-* tokens are large-binary-values and their binary length (not the length of the character string that represents them in encoded form) MUST not exceed 65536 bytes.

10.4 Secure Remote Password (SRP)

For SRP [RFC2945], the initiator MUST use:

SRP_U=<user> TargetAuth=Yes /* or TargetAuth=No */

The target MUST answer with a Login reject with the "Authorization Failure" status or reply with:

SRP_N=<N> SRP_g=<g> SRP_s=<s>

The initiator MUST either close the connection or continue with:

SRP_A=<A>

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, N, g, s, A, B, M, and H(A | M | K) are defined in [RFC2945] (using the SHA1 hash function, i.e., SRP-SHA1), U is a text string, N,g,s,A,B,M, and H(A | M | K) are binary-values. The length of N,g,s,A,B,M in binary form (not the length of the character string that represents them in encoded form) MUST not exceed 1024 bytes. Further restrictions on allowed N,g values are specified in Section 7.2 In-band Initiator-Target Authentication.

10.5 Challenge Handshake Authentication Protocol (CHAP)

For CHAP [RFC1994], the initiator MUST use:

CHAP_A=<A1,A2...>

Where A1,A2... are proposed algorithms, in order of preference.

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.

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 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 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 SHOULD always offer it as one of the proposed algorithms.

11. Login/Text Operational 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 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 be used only 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 appear also 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.

11.1 HeaderDigest and DataDigest

Use: IO
Senders: Initiator and Target
Scope: CO

HeaderDigest = <list-of-options>
DataDigest = <list-of-options>

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.

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 and 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 to through 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.

- Computing the CRC over any segment (data or header) extended to include the CRC built using the generator 0x11edc6f41 will get always 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

Proprietary algorithms MAY also be negotiated for digests. Whenever a proprietary algorithm is negotiated, "None" or "CRC32C" should be listed as an option in order to guarantee interoperability.

11.2 MaxConnections

Use: LO
Senders: Initiator and Target
Scope: SW

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.

11.3 SendTargets

Use: FFPO
Senders: Initiator
Scope: SW

For a complete description, see Appendix D. - SendTargets Operation.

11.4 TargetName

Use: IO by initiator ALL by target, 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).

11.5 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
InitiatorName=iSCSI

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 Initiator key enables the initiator to identify itself to the remote endpoint.

11.6 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 MUST be communicated to the initiator during a Login Response PDU. This string is not used as an identifier, but can be displayed by the initiator's user interface in a list of targets to which it is connected.

11.7 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 may 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, but can be displayed by the target's user interface in a list of initiators to which it is connected.

This key SHOULD be sent by an initiator within the Login Phase, if available.

11.8 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 (3260).

If the TargetAddress is returned as the result of a redirect status in a login response, the comma and portal group tag are omitted.

If the TargetAddress is returned within a SendTargets response, the portal group tag is required.

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.acme.com,23
```

Use of the portal-group-tag is described in Appendix D. - SendTargets Operation.

11.9 TargetPortalGroupTag

Use: IO by target, Declarative, Any-Stage
Senders: Target
Scope: SW

TargetPortalGroupTag=<numerical-value-from-1-to-65535>

Examples:
TargetPortalGroupTag=1

Target portal group tag is a 16-bit numerical-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.

For the complete usage expectations of this key see Section 4.3 Login Phase.

11.10 InitialR2T

Use: LO

Senders: Initiator and Target
Scope: SW

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, 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 DataTransfer 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).

11.11 BidiInitialR2T

Use: LO
Senders: Initiator and Target
Scope: SW

BidiInitialR2T=<boolean-value>

Examples:

I->BidiInitialR2T=No
T->BidiInitialR2T=No

Default is Yes.
Result function is OR.

The BidiInitialR2T key is used to turn off the default use of BiDiR2T, thus allowing an initiator to send data to a target without the target having sent an R2T to the initiator for the output data

(write part) of a Bidirectional command (having both the R and the W bits set). The default action is that R2T is required, unless both the initiator and the target send this key-pair attribute specifying BidiInitialR2T=No. Only the first outgoing data burst (immediate data and/or separate PDUs) can be sent unsolicited by an R2T.

11.12 ImmediateData

Use: LO
Senders: Initiator and Target
Scope: SW

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

11.13 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 not exceeding MaxRecvDataSegmentLength of the receiver.

A target receiver is additionally limited by MaxBurstLength for solicited data and FirstBurstLength for unsolicited data and an initiator MUST NOT send solicited PDUs exceeding MaxBurstLength nor unsolicited PDUs exceeding FirstBurstLength (or FirstBurstLength-Immediate Data Length if immediate data where sent).

11.14 MaxBurstLength

Use: LO
Senders: Initiator and Target
Scope: SW

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 of Data-In or Data-Out PDUs ending with a Data-In or Data-Out PDU with the F bit set to one.

11.15 FirstBurstLength

Use: LO
Senders: Initiator and Target
Scope: SW

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.

11.16 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 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.

11.17 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.

11.18 MaxOutstandingR2T

Use: LO
Senders: Initiator and Target
Scope: SW

MaxOutstandingR2T=<numerical-value-from-1-to-65535>

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 6.12.1) is encountered for that data sequence.

11.19 DataPDUInOrder

Use: LO
Senders: Initiator and Target
Scope: SW

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.

11.20 DataSequenceInOrder

Use: LO
Senders: Initiator and Target
Scope: SW

DataSequenceInOrder=<boolean-value>

Default is Yes.
Result function is OR.

A Data Sequence is a sequence of Data-In or Data-Out PDUs ending 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.

11.21 ErrorRecoveryLevel

Use: LO
Senders: Initiator and Target
Scope: SW

ErrorRecoveryLevel=<numerical-value-0-to-2>

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 6.13 Error Recovery Hierarchy describes the mapping between the classes and the levels.

11.22 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.

11.23 The Vendor Specific Key Format

Use: ALL
Senders: Initiator and Target
Scope: specific key dependent

X-reversed.vendor.dns_name.do_something=

Keys with this format are used for vendor-specific purposes. These keys always start with X-.

To identify the vendor, we suggest you use the reversed DNS-name as a prefix to the key-proper.

12. IANA Considerations

The temporary (user) well-known port number for iSCSI connections assigned by IANA is 3260.

References and Bibliography

Normative References

[AESCBC] Frankel, S., Kelly, S., Glenn, R., "The AES Cipher Algorithm and Its Use with IPsec", Internet draft (work in progress), draft-ietf-ipsec-ciph-aes-cbc-03.txt, November 2001.
[AESCTR] Walker, J., Moskowitz, R., "The AES128 CTR Mode of Operation and Its Use with IPsec", Internet draft (work in progress), draft-moskowitz-aes128-ctr-00.txt, September 2001.
[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/index.shtml
[RFC790] J. Postel, ASSIGNED NUMBERS, September 1981.
[RFC791] INTERNET PROTOCOL, DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION, September 1981.
[RFC793] TRANSMISSION CONTROL PROTOCOL, DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION, September 1981.
[RFC1035] P. Mockapetris, DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION, November 1987.
[RFC1122] Requirements for Internet Hosts-Communication Layer RFC1122, R. Braden (editor).
[RFC1510] J. Kohl, C. Neuman, "The Kerberos Network Authentication Service (V5)", September 1993.
[RFC1737] K. Sollins, L. Masinter "Functional Requirements for Uniform Resource Names".
[RFC1766] H. Alvestrand, "Tags for the Identification of Languages", March 1995.
[RFC1964] J. Linn, "The Kerberos Version 5 GSS-API Mechanism", June 1996.
[RFC1982] Elz, R., Bush, R., "Serial Number Arithmetic", RFC 1982, August 1996.
[RFC1994] "W. Simpson, PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.
[RFC2025] C. Adams, "The Simple Public-Key GSS-API Mechanism (SPKM)", October 1996.
[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", RFC 2026, October 1996.
[RFC2044] Yergeau, F., "UTF-8, a Transformation Format of Unicode and ISO 10646", October 1996.
[RFC2045] N. Borenstein, N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", November 1996.
[RFC2119] Bradner, S. "Key Words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234] D. Crocker, P. Overell Augmented BNF for Syntax Specifications: ABNF.
[RFC2246] T. Dierks, C. Allen, " The TLS Protocol Version 1.0.
[RFC2373] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.
[RFC2396] T. Berners-Lee, R. Fielding, L. Masinter "Uniform Resource Identifiers".
[RFC2434] T. Narten, and H. Avestrand, "Guidelines for Writing an IANA Considerations Section in RFCs.", RFC2434, October 1998.
[RFC2401] S. Kent, R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.
[RFC2404] C. Madson, R. Glenn, "The Use of HMAC-SHA-1-96 within ESP and AH", RFC 2404, November 1998.
[RFC2406] S. Kent, R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.
[RFC2407] D. Piper, "The Internet IP Security Domain of Interpretation of ISAKMP", RFC 2407, November 1998.
[RFC2409] D. Harkins, D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.
[RFC2451] R. Pereira, R. Adams " The ESP CBC-Mode Cipher Algorithms".
[RFC2732] R. Hinden, B. Carpenter, L. Masinter, "Format for Literal IPv6 Addresses in URL's", RFC 2732, December 1999.
[RFC2945], Wu, T., "The SRP Authentication and Key Exchange System", September 2000.
[SAM] ANSI X3.270-1998, SCSI-3 Architecture Model (SAM).
[SAM2] T10/1157D, SCSI Architecture Model - 2 (SAM-2).
[SBC] NCITS.306-1998, SCSI-3 Block Commands (SBC).
[SEQ-EXT] Kent, S., "IP Encapsulating Security Payload (ESP)", Internet draft (work in progress), draft-ietf-ipsec-esp-v3-01.txt, November 2002.
[SEC-IPS] B. Aboba & team "Securing Block Storage Protocols over IP", Internet draft (work in progress), draft-ietf-ips-security-09.txt, February 2002.
[SPC]T10/1416-D, SCSI-3 Primary Commands.
[SPC3]T10/1416-D, SCSI Primary Commands-3.
[STPREP] P. Hoffman, M. Blanchet, "Preparation of Internationalized Strings", draft-hoffman-stringprep-00.txt, September, 2001.
[STPREP-iSCSI] M. Bakke, "String Profile for iSCSI Names", draft-ietf-ips-iscsi-string-prep-00.txt, November 2001.
[UNICODE] Unicode Standard Annex #15, "Unicode Normalization Forms", http://www.unicode.org/unicode/reports/15

Informative References:

[BOOT] P. Sarkar & team draft-ietf-ips-iscsi-boot-03.txt (work in progress)
[COBS] S. Cheshire and M. Baker, Consistent Overhead Byte Stuffing, IEEE Transactions on Networking, April 1999.
[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.
[CRC] ISO 3309, High-Level Data Link Control (CRC 32).
[NDT] M. Bakke & team, draft-ietf-ips-iscsi-name-disc-05.txt (work in progress)
[Schneier] B. Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 2nd edition, John Wiley & Sons, New York, NY, 1996.

Authors' Addresses

Julian Satran
IBM, Haifa Research Lab
Haifa University Campus - Mount Carmel
Haifa 31905, Israel
Phone +972.4.829.6264
E-mail: Julian_Satran@il.ibm.com

Kalman Meth
Haifa University Campus - Mount Carmel
MATAM - Advanced Technology Center
Haifa 31905, Israel
Phone +972.4.829.6341
E-mail: meth@il.ibm.com

Costa Sapuntzakis
Cisco Systems, Inc.
170 W. Tasman Drive
San Jose, CA 95134, USA
Phone: +1.408.525.5497
E-mail: csapuntz@cisco.com

Efri Zeidner
SANgate Systems, Inc.
41 Hameyasdim Street
P.O.B. 1486
Even-Yehuda, Israel 40500
Phone: +972.9.891.9555
E-mail: efri@sangate.com

Mallikarjun Chadalapaka
Hewlett-Packard Company
8000 Foothills Blvd.

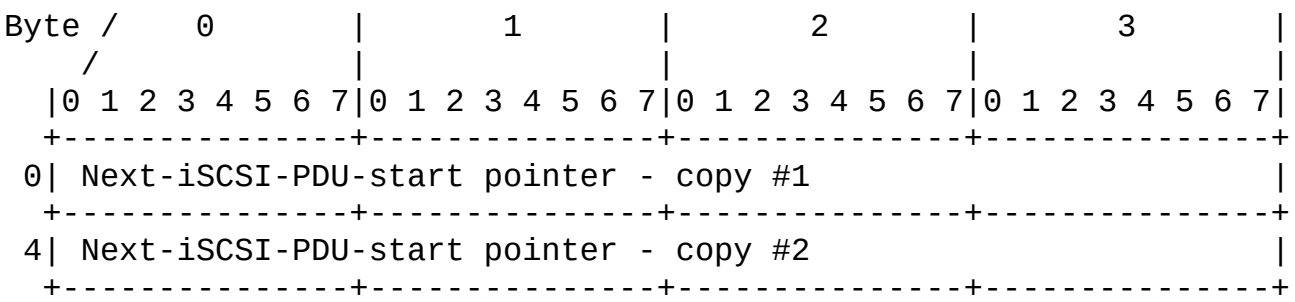
Roseville, CA 95747-5668, USA
Phone: +1.916.785.5621
E-mail: cbm@rose.hp.com

Comments may be sent to Julian Satran

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:



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.

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 inserted value is 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 not pro-

viding markers to a receiver wanting markers may suffer an appreciable performance degradation.

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 started only 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.

As an 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.

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

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.

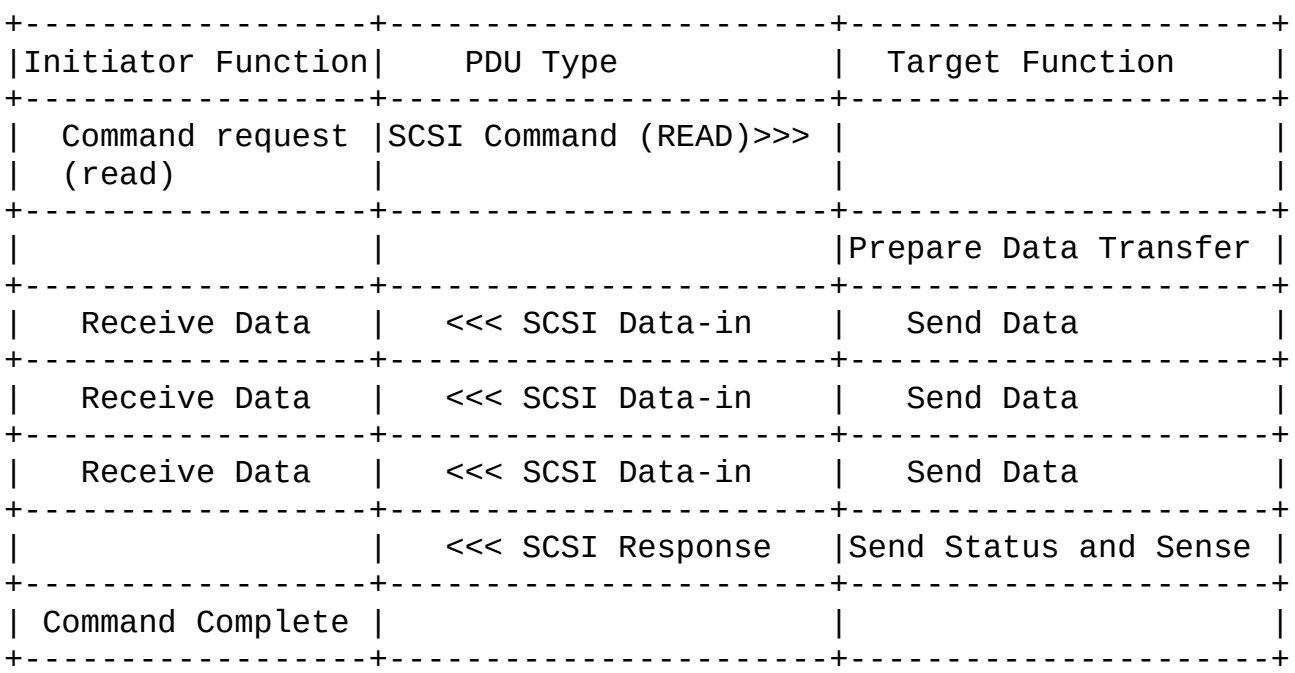
For the offering the initiator or target indicates the minimum to maximum interval (in 4-byte words) it wants the markers for one or both directions. In case it only wants a specific value, only a single value has to be specified. The responder selects a value within the minimum and maximum offered or 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.

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).

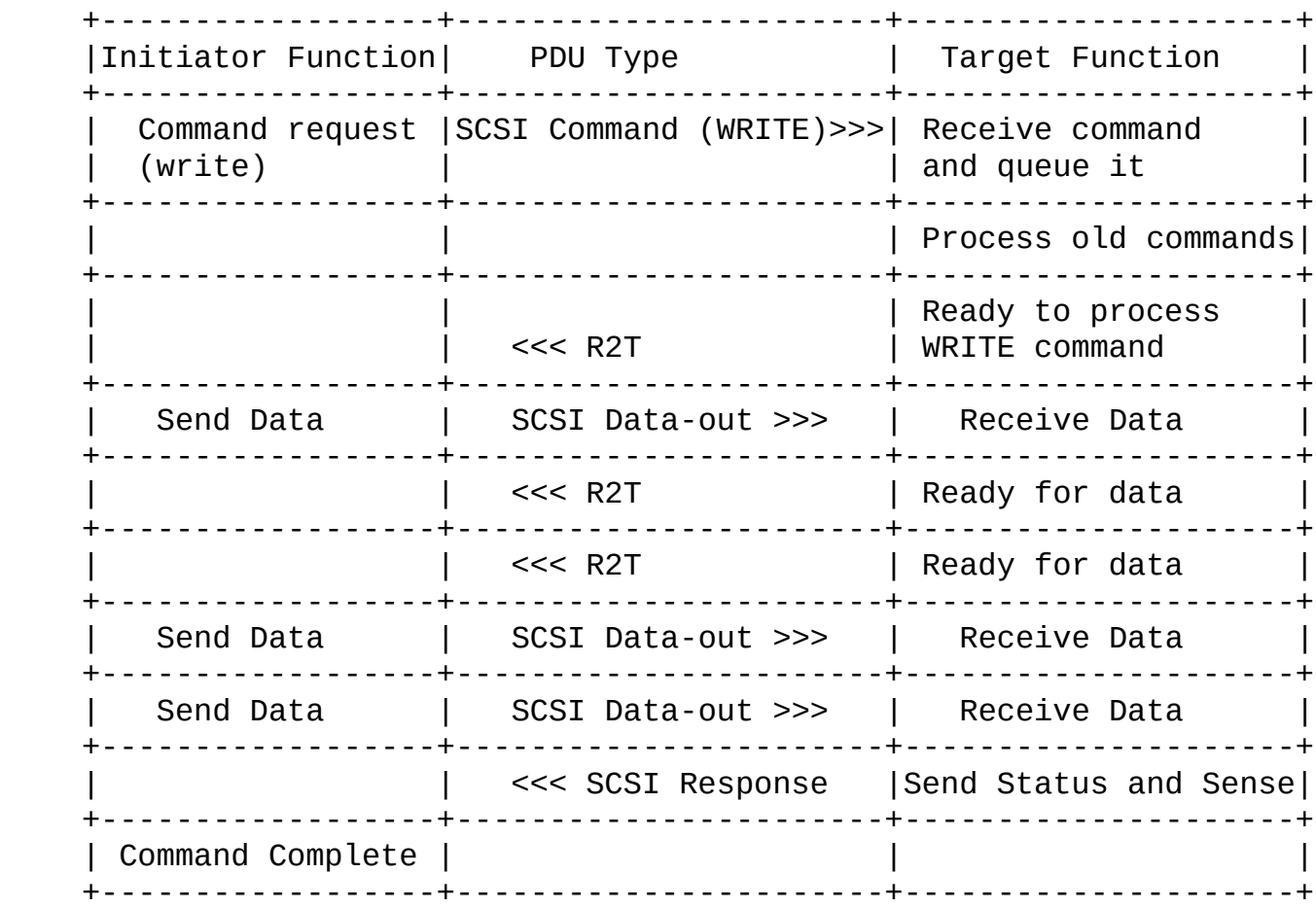
The default is 2048.

Appendix B. Examples

B.1 Read Operation Example

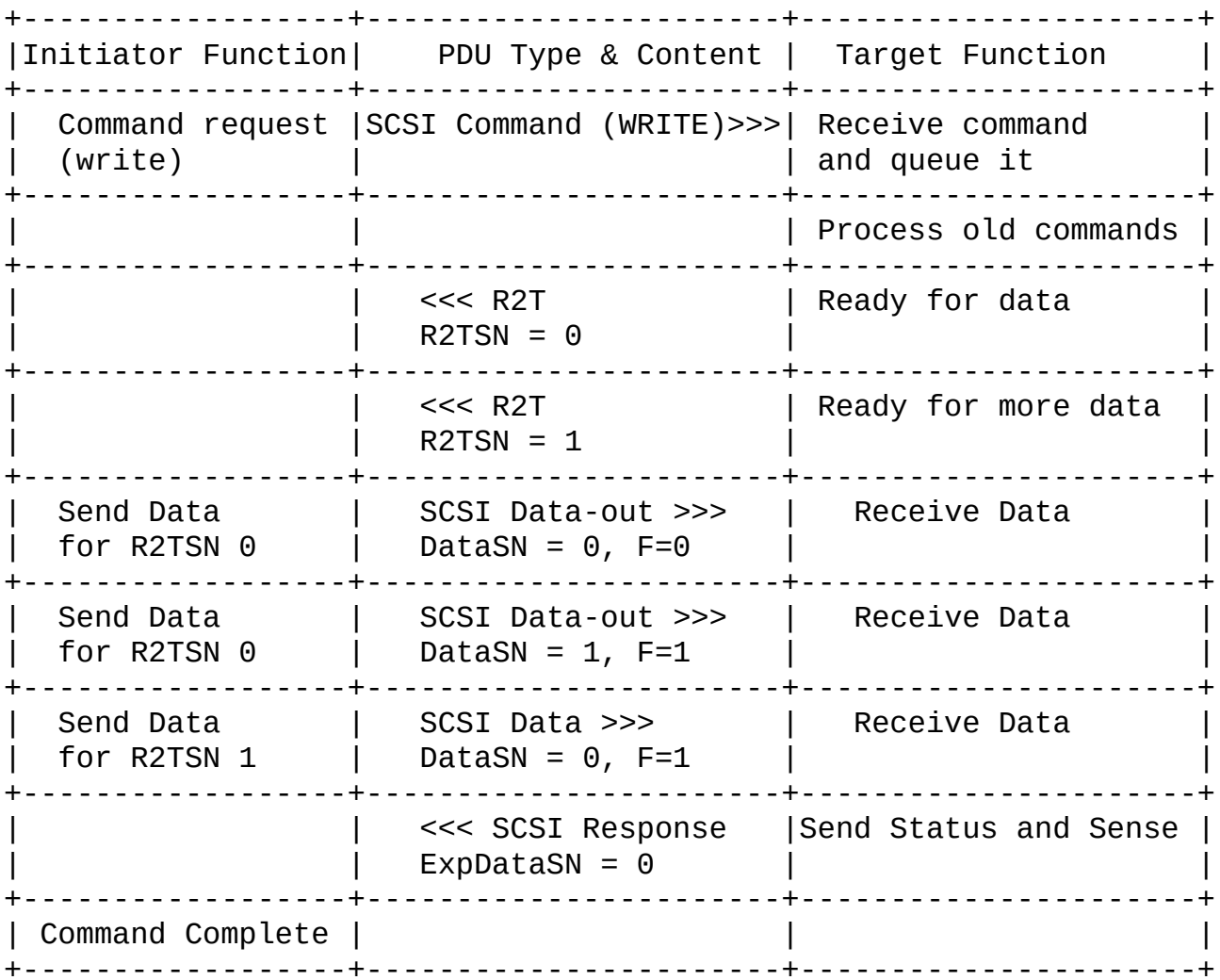


B.2 Write Operation Example

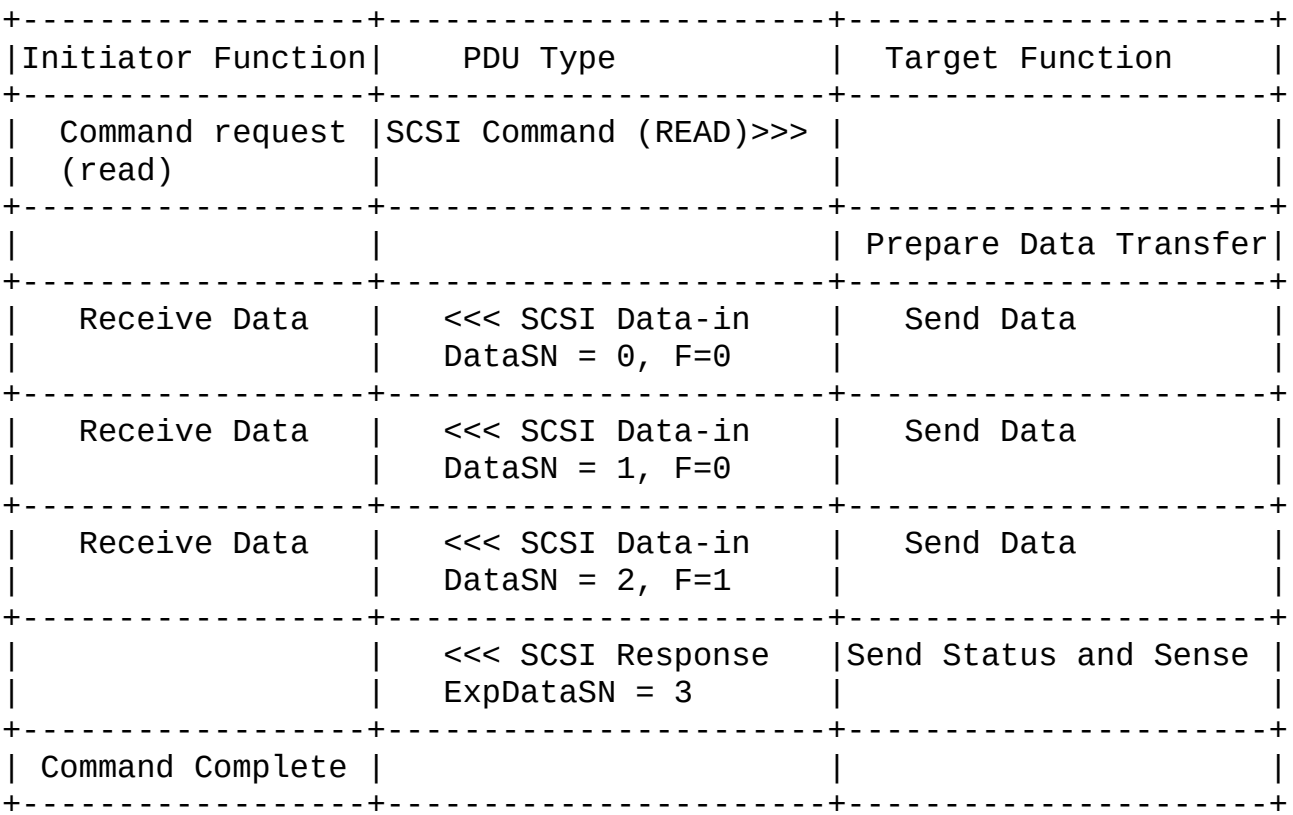


B.3 R2TSN/DataSN use Examples

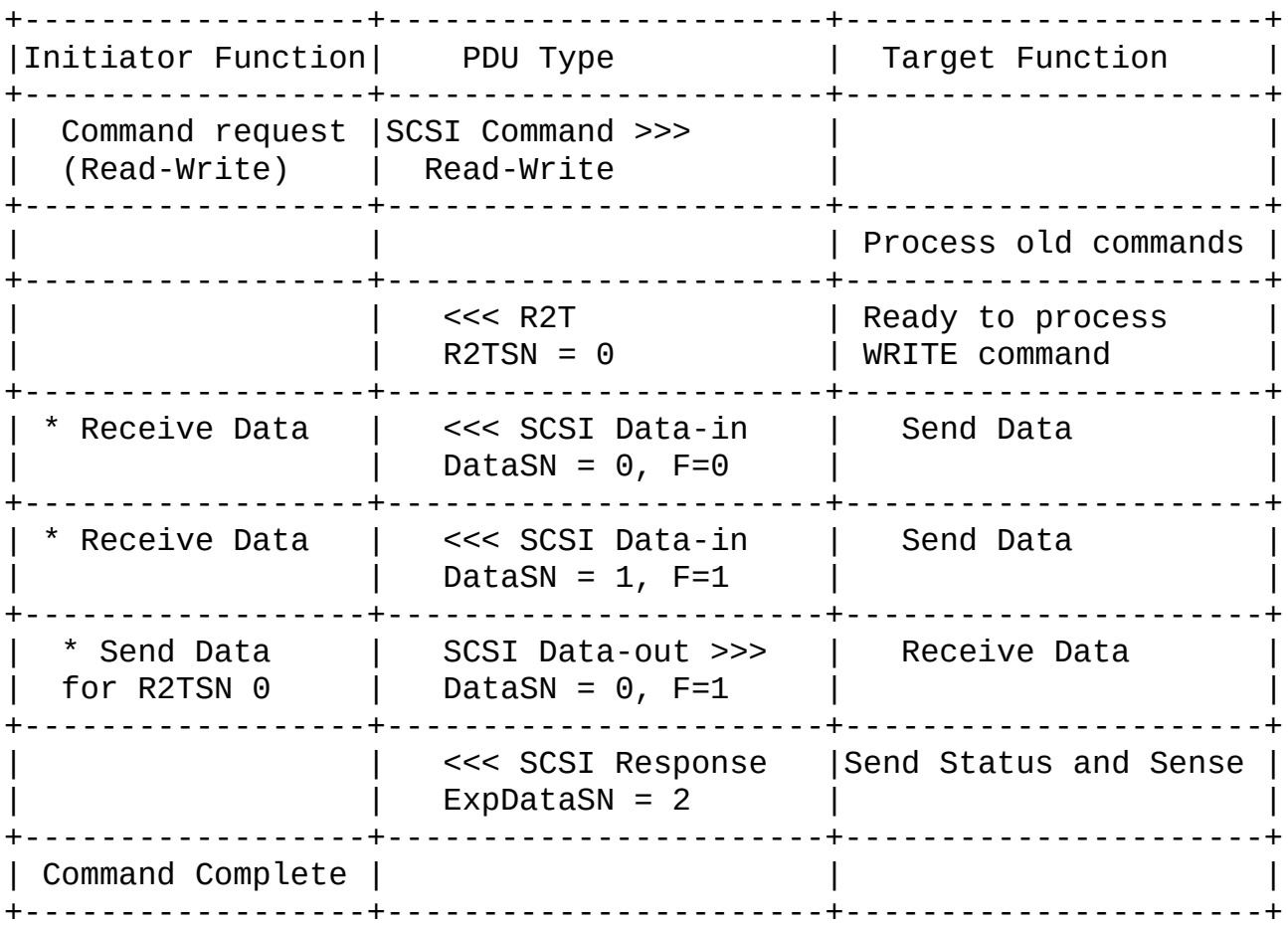
Output (write) data DataSN/R2TSN Example



Input (read) data DataSN Example

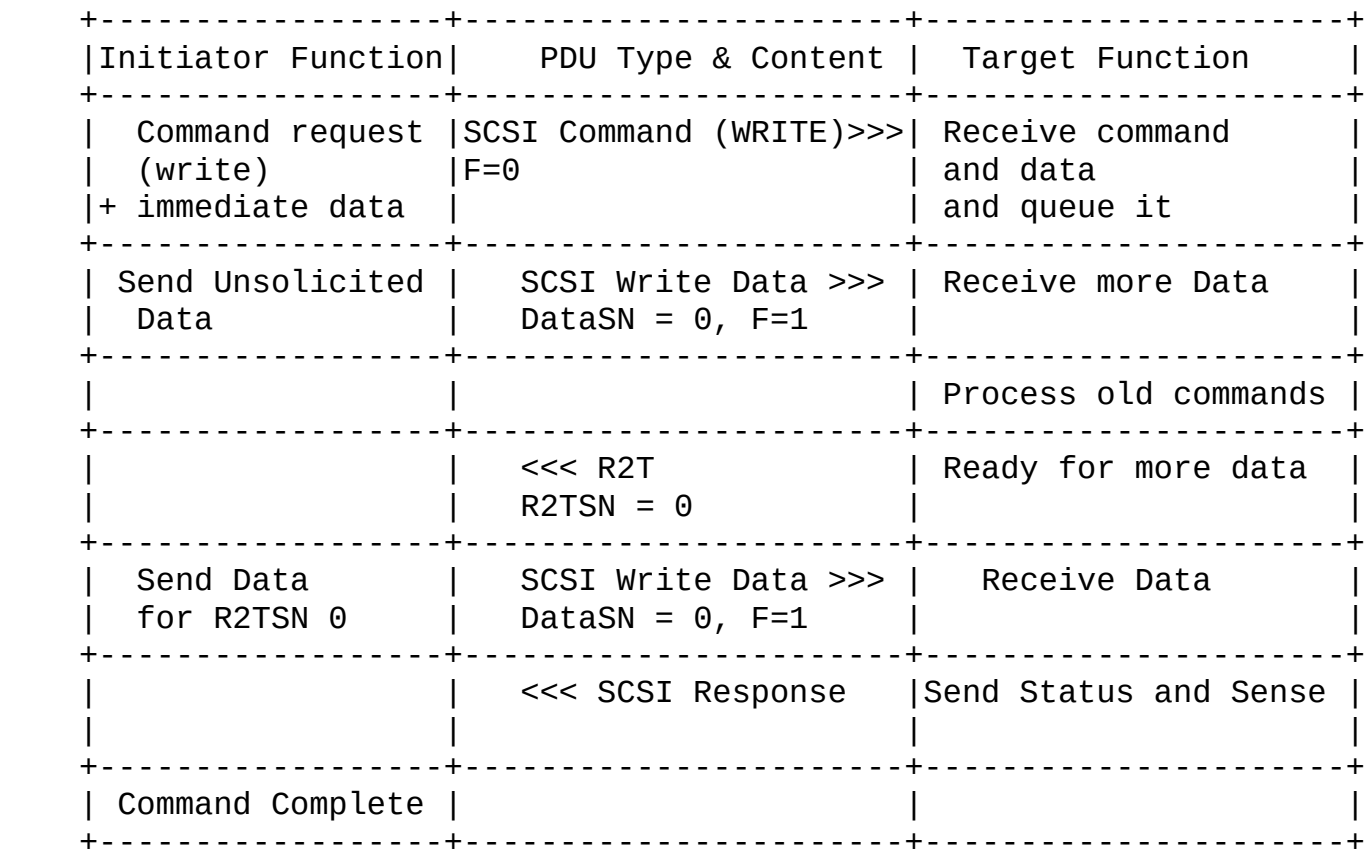


Bidirectional DataSN Example



*) Send data and Receive Data may be transferred simultaneously as in an atomic Read-Old-Write-New or sequential as in an atomic Read-Update-Write (in the alter case the R2T may follow the received data).

Unsolicited and immediate output (write) data with DataSN Example



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

   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
```

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.acme.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=0
T-> Login (CSG,NSG=1,0 T=0) FirstBurstLength=8192 MaxBurst-
  Length=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.acme.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.acme.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 success-
      ful, the initiator terminates the connection (without
      responding to the Login SPKM_REP_TI message).

    If the initiator's authentication by the target is not success-
      ful, 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.acme.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"
```

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.acme.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_N=<N>
    SRP_g=<g>
    SRP_s=<s>

I-> Login (CSG,NSG=0,0 T=0)
    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, only the initiator is authenticated by the target via SRP:

```
I-> Login (CSG,NSG=0,1 T=1)
    InitiatorName=iqn.1999-07.com.os.hostid.77
    TargetName=iqn.1999-07.com.acme.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_N=<N>
    SRP_g=<g>
    SRP_s=<s>

I-> Login (CSG,NSG=0,0 T=0)
    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.acme.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.acme.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.acme.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.acme.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. A target MUST return all path information (IP address-port pairs and portal group tags) for the targets for which the requesting initiator is authorized.

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 need not 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.

<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 been designated.

<nothing>

The session should respond only 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 11.4 TargetName.

A discovery session 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 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.

Each TargetAddress belongs to a portal group, identified by its numeric portal group tag (as in Section 11.9 TargetPortalGroupTag). The iSCSI target name, together with this tag, constitutes the SCSI port identifier; the tag need be unique only within a given target name's 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.acme.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.acme.diskarray.sn.8675309
TargetAddress=10.1.0.45:3000,1
TargetAddress=10.1.1.45:3000,2
TargetName=iqn.1993-11.com.acme.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 also 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.

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 illustrates further the use of the portal group tags:

```
TargetName=iqn.1993-11.com.acme.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 communicated via the MaxConnections text key upon login to the target.

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 ignore all other aspects/cases. Examples of this approach are:

- Handling for only certain Opcode types is shown.
- Only certain reason codes (for example, 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;
        struct TransferContext
                    TransferContextList[MaxOutStandingR2T];
        int InitiatorTaskTag;
        int CmdSN;
};

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-Abort(task control block);
SCSI-Task-Completion(task control block);
Build-And-Send-a-Data-Burst(transport connection, R2T PDU,
                                        task control block);
```

```
Build-And-Send-R2T(transport connection, description of data,
                                                task control block);
Build-And-Send-Status(transport connection, task control block);
Transfer-Context-Timeout-Handler(transfer context);

Implementation-specific tunables -
InitiatorDataSNACKEnabled, TargetDataSNACKSupported,
TargetRecoveryR2TEnabled.
```

Notes:

- Some procedures used in this section, including: Recover-Status-if-Possible, Handle-Status-SNACK-request, Evaluate-a-StatSN are 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 (InitiatorDataSNACKEnabled) {
         if (# of missing PDUs is trackable) {
               Note the missing DataSNs in TCB.
               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) {
     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.
      Build-And-Send-A-Data-Burst(Connection, CurrentPDU, 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;
            }
            send-status-SNACK = Evaluate-a-StatSN(Connection,
                                        CurrentPDU.StatSN);
         }
         if (send-status-SNACK is TRUE)
            Recover-Status-if-Possible(Connection, CurrentPDU);
      } else { /* REST UNRELATED TO WITHIN-COMMAND-RECOVERY, NOT SHOWN
   */
      }
      if ((TCB.SoFarInOrder is TRUE) and
                        (TCB.StatusXferd is 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 (TargetRecoveryR2TEnabled is TRUE) {
                Increment TCB.ActiveR2Ts.
                Build-And-Send-R2T(Connection, MissingDataRange, 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 (this is data retransmission request) {
            if (TargetDataSNACKSupported) {
                 if (the request is satisfiable) {
                       Build-And-Send-A-Data-Burst(CurrentPDU, TCB);
                 } else {
                       snack-failure = TRUE;
                 }
            } else {
                 snack-failure = TRUE;
            }
            if (snack-failure is TRUE) {
                 Build-And-Send-Reject(Connection, CurrentPDU,
                                                    Data-SNACK-Reject);
                 if (TCB.StatusXferd is not TRUE) {
```

```
                    TCB.Reason = "SNACK Rejected";
                    Build-And-Send-Status(Connection, TCB);
                }
            }
        } else {
            Handle-Status-SNACK-request(Connection, CurrentPDU);
        }
    } else { /* REST UNRELATED TO WITHIN-COMMAND-RECOVERY, NOT SHOWN */
    }
}

Transfer-Context-Timeout-Handler(TContext)
{
    Retrieve TCB and Connection from TContext.
    Decrement TCB.ActiveR2Ts.
    if (TargetRecoveryR2TEnabled is TRUE and
                task is not already considered failed) {
        Note the missing data PDUs in MissingDataRange[].
        Build-And-Send-R2T(Connection, MissingDataRange, 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, current StatSN);
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:
InitiatorCommandRetryEnabled, InitiatorStatusExpectNopEnabled, InitiatorProactiveSNACKEnabled, InitiatorStatusSNACKEnabled, TargetStatusSNACKSupported.

Notes:

- The initiator algorithms only deal with unsolicited Nop-In PDUs for generating status SNACKs. 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.

E.3.2 Initiator Algorithms

```
Recover-Status-if-Possible(Connection, CurrentPDU)
{
    if ((Connection.state = LOGGED_IN) and
             connection is not already considered failed) {
       if (InitiatorStatusSNACKEnabled) {
          if (# of missing PDUs is trackable) {
            Note the missing StatSNs in Connection;
            Build-And-Send-SSnack(Connection);
          } else {
            Connection.PerformConnectionCleanup = TRUE;
          }
       } else {
          Connection.PerformConnectionCleanup = TRUE;
       }
       if (Connection.PerformConnectionCleanup is TRUE) {
          Start-Timer(Connection-Cleanup-Handler, Connection, 0);
       }
```

```
        }
    }

    Retransmit-Command-if-Possible(Connection, CmdSN)
    {
        if (InitiatorCommandRetryEnabled) {
           Retrieve the InitiatorTaskTag, and thus TCB for the CmdSN.
           Build-And-Send-Command(Connection, TCB);
        }
    }

    Evaluate-a-StatSN(Connection, StatSN)
    {
        send-status-SNACK = FALSE;
        if (Connection.SoFarInOrder is 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 {
                    discard, return;
                }
           }
           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.StatSN);
         if (send-status-SNACK is 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 (InitiatorStatusExpectNopEnabled) {
        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 (TargetStatusSNACKSupported) {
       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.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 = TaskManagement) {
         if (CurrentPDU.function = "TaskReassign") {
               if (Session.ErrorRecoveryLevel < 2) {
                  Build-And-Send-TaskMgmt-Response(Connection,
                       CurrentPDU, "Task failover 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 merely 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 noted otherwise, the clearing action is applicable only 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.

	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 ex., 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, so is 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 transport connection shutdown, transport connection reset, or transport connection timeout abruptly terminating 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 than 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 4.3.5 Session reinstatement, closure and timeout.

11.This clearing effect is however "Y" only if it is a connection reinstatement and the operational ErrorRecoveryLevel is less than 2.

12.Session continuation is as defined in Section 4.3.6 Session continuation and failure.

13.This clearing effect is valid only 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 valid only for a "close the session" logout in a multi-connection session. In all other cases, the effect is "NA".

15.Applicable only if this leading connection login is a session reinstatement. If that is not the case, this is "NA".

16.This operation affects all logged-in initiators.

18.Session failure is as defined in Section 4.3.6 Session continuation and failure.

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.

Note that because iSCSI defines target cold reset as 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 (thus between the same SCSI ports with the same nexus identifier) establishing the same I_T nexus again, 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.

Full Copyright Statement