IPS
Internet Draft
draft-ietf-ips-iSCSI-10.txt
Category: standards-track

Julian Satran
Daniel Smith
Kalman Meth
Ofer Biran
Jim Hafner
IBM

Costa Sapuntzakis
Mark Bakke
Cisco Systems

Matt Wakeley
Agilent Technologies

Luciano Dalle Ore
Quantum

Paul Von Stamwitz
Adaptec

Randy Haagens
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 memo describes a transport protocol for SCSI that operates on top of TCP. The iSCSI protocol aims to be fully compliant with the requirements laid out in the SCSI Architecture Model - 2 [SAM2] document.

Acknowledgements

In addition to the authors, a large group of people contributed to this work through their review, comments and valuable insights. We are grateful to all of them. We are especially grateful to those who found the time and patience to participate in our weekly phone conferences and intermediate meetings in Almaden and Haifa, thus helping to shape this document: John Hufferd, Prasenjit Sarkar, Meir Toledano, John Dowdy, Steve Legg, Alain Azagury (IBM), Dave Nagle (CMU), David Black (EMC), John Matze (Veritas - now with Stonefly Networks), Steve DeGroote, Mark Shrandt (NuSpeed), Gabi Hecht (Gadzoox), Robert Snively (Brocade), Nelson Nachum (StorAge), Uri Elzur (Broadcom). Many more helped clean up 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 Systems), Michael Krause, Pierre Labat, Santosh Rao, Matthew Burbridge (HP), Stephen Bailey (Sandburst), Robert Elliott (Compaq),

Steve Senum, Ayman Ghanem (CISCO), Barry Reinhold (Trebia Networks), Bob Russell (UNH), Bill Lynn (Adaptec), Doug Otis (Sanlight), Robert Griswold and Bill Moody (Crossroads). The recovery chapter was enhanced with help from Stephen Bailey (Sandburst), Somesh Gupta (HP), Venkat Rangan (Rhapsody Networks), Vince Cavanna, Pat Thaler (Agilent), Eddy Quicksall (iVivity, Inc.) - Eddy also contributed with some examples. 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.

At the time of the writing, this document has to be considered in conjunction 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), Joe Czap, Jim Hafner, John Hufferd, Kaladhar Voruganti (IBM), Howard Hall (Pirus), Jack Harwood (EMC), Yaron Klein (SANRAD), Lawrence Lamers (San Valley Systems), Todd Sperry (Adaptec) and Joshua Tseng (Nishan).

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, William Dixon (Microsoft), David Black (EMC), Joseph Tardo, Uri Elzur (Broadcom), Mark Bakke, Steve Senum (Cisco Systems), Howard Herbert, Jesse Walker (Intel), Julian Satran, Ofer Biran and Charles Kunzinger (IBM).

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

Conventions used in this document

In examples, "I->" and "T->" indicate 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.

Change Log

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" since 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 SPC3
- 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 10
- InitialR2T, BidiInitialR2T and ImmediateData changed to LO
- I bit (equivalent) in responses made 0
- Added a "double response" version for the ? key value to Section 2.2.4 Text Mode Negotiation
- ? 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 10.2.1.7 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 10.7 SCSI Data-out & SCSI Data-in - good status in Data-In must be supported by initiators
- Clarified InitiatorName is required at login in Section 4.1 Login Phase Start
- Another clarification for SecurityContextComplete in Section 4.2 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 10.4.3 Response
- Brought Section 2.2.7 Naming and Addressing 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 2.2.5 iSCSI Full Feature Phase
- Added new Reject reason codes, tabular listing and a pointer to Section 10.14.3 Reason Code
- Added additional Reject usage semantics on CmdSN and DataSN to Section 10.14.3 Reason Code
- 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 6 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 10.6 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 10.4 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 D. - 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 10.4.2 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 9.
- Reintroduced aliases (the data format) in an appendix. The aliasing mechanism once part of iSCSI is part of [SPC3]
- 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 8 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 6.
- Several changes in Chapter 7 corresponding to the new task management function "reassign". Other language changes in Chapter 7 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 7.12 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 10.17 Reject to use the "Invalid SNACK" reason code.
- Removed a Logout reason code in Section 10.14 Logout Request to be consistent with Section 10.9 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 Bi-directional 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 FirstBurst-Size 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.

Table of Contents

1. Definitions

- Alias: 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.

- 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: 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 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 nodes also have addresses. An iSCSI address specifies a single path to an iSCSI node.

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

- 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 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 connec-

tions 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 integer value 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 is defined by a session ID that is composed of an initiator part (ISID) and a target part (TSID).

- 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 is basically 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.

- TSID (Target Session ID): The TSID is the target assigned tag for a session with a specific named initiator that, together with the ISID uniquely identifies a session with that initiator.
It is given to the target during additional connections for the same session to identify the associated session.

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 regard to the initiator. Outbound or outgoing transfers are transfers from initiator to target, while inbound or incoming transfers are from target to 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. In addition, a target sees one "initiator image" across all connections within a session. Initiator that identifying elements, such as the Initiator Task Tag, can be used to identify the same entity 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 NOPOut/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 request PDUs carry this number. The iSCSI initiator allocates CmdSNs with a 32-bit unsigned counter (modulo 2**32). Comparisons and arithmetic on CmdSN SHOULD 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 task management before the tasks on which they are supposed to act. However, their CmdSN is a marker of their position in the stream of commands. The task management command MUST carry the CmdSN that is given to the next non-immediate command. 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.

Beyond the scope of this document is 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.

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 a lack of resources. An iSCSI target MUST be able to handle at least one immediate task management command and one immediate non-task-management iSCSI request 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 SHOULD

use Serial Number Arithmetic as defined in [RFC1982] where SERIAL_BITS = 32.

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

This sequence is required since updates may arrive out of order being that they travel on different TCP connections.

The target MUST NOT transmit a MaxCmdSN that is less than the last ExpCmdSN. For non-immediate commands, the CmdSN field can take any value from ExpCmdSN to MaxCmdSN. The target MUST silently ignore any non-immediate command outside of this range or non-immediate duplicates within the range.

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. 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.

When the current value of the CmdSN register is Q, an initiator MUST not advance CmdSN past R + 2**31 - 1 after reissuing a command with CmdSN R on a connection while this connection is operational, unless a new non-immediate command with CmdSN equal or greater than Q was issued on the given connection and its reception acknowledged by the target (see Section 9.3 Command Retry and Cleaning Old Command

Instances). The non-immediate command MUST be sent in order after the retried command.
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 32bit 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 write command, a target must have issued less than 2**32 R2Ts. Any input or output data sequence MUST contain less than 2**32 numbered 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 8 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 a session ID that is composed of an initiator part ISID and a target part TSID. For a new session, the TSID is null. As part of the response, the target generates a TSID.

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 part). 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.

2.2.4 Text Mode Negotiation

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

The initiator starts the negotiation 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).

The general format of text negotiation is:

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

The originator can either be the initiator or the target and the responder can either be the target or initiator, respectively. Target requests 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 stateless (i.e., the result MUST be based only on newly exchanged values). Not offering a key for negotiation is not equivalent to offering the current (or default) value.

The value can be a number, a single literal constant a Boolean value (yes or no), or a list of comma separated, literal constant values.

In literal list negotiation, the originator sends a list of options (literal constants which may include "None") for each key in its order of preference.

The responding party answers with 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 support or is not allowed to use all of the offered options with a specific originator, it may use the constant "Reject".

For numerical and single literal negotiations, the responding party MUST respond with the required key. The value it selects, based on the selection rule specific to the key, becomes the negotiation result. The selection of a value not admissible under the selection rules is considered a protocol error and is handled accordingly.

For Boolean negotiations (keys taking the values yes or no), the responding party MUST respond with the required 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 select in the absence of knowledge of the received value.

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.

If a specific key is not relevant for the current negotiation, the responder may answer with the constant "Irrelevant" for all types of negotiation.

Any other 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 value "?" with any key has the meaning of enquiry and should be answered with the current value or "NotUnderstood". The value "?" MUST be used ONLY in Full Feature Phase. Whenever the responder has 2 two values for a key - one for the offering-to-responding-party direction and a second one for the responding-to-offering-party direction it will answer with the two values separated by a comma starting with the requesting-to-offering-party direction.

The constants "None", "Reject", "Irrelevant", and "NotUnderstood" are reserved and must only be used as described here.

Some basic key=value pairs are described in Chapter 12. All keys in Chapter 12, except for the X- extension format, MUST be supported by iSCSI initiators and targets.

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

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

2.2.5 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 by default. 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 7.1 Retry and Reassign in Recovery.

For SCSI commands that require data and/or a parameter transfer, the (optional) data and the status for a 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. 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 is sent in response to R2T PDUs. Unsolicited data can be sent as part of an iSCSI command PDU ("immediate data") or in separate iSCSI data PDUs. An initiator may send unsolicited data as immediate up to the negotiated maximum PDU size or in a separate PDU sequence (up to the mode page limit). All subsequent data MUST be solicited. The maximum size 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. In unsolicited mode, an initial R2T that allows a transfer up to the FirstBurstSize is implied. A target MAY separately enable immediate data without enabling the more general (separate data PDUs) form of unsolicited data.

An initiator SHOULD honor an R2T data request for a valid outstanding command (i.e., carrying a valid Initiator Task Tag) provided the command is supposed to deliver outgoing data and the R2T specifies data within the command bounds.

It is considered an error for an initiator to send unsolicited data PDUs to a target that operates in R2T mode (only solicited data is allowed). It is also an error for an initiator to send more data, whether immediate or as separate PDUs, than the SCSI limit for first burst. At login, an initiator MAY request to send data blocks and a first burst of any size; in this case, the target MUST indicate the size of the first burst and of the immediate and data blocks that it is ready to accept.

A target SHOULD NOT silently discard data and 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 these 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.

2.2.6 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 6) 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. In this case, the initiator sends a Logout request on one of the operational connections of a session that indicates which iSCSI connection should be logged out.

2.2.7 Naming and Addressing

All iSCSI initiators and targets are named. Each target or initiator is known by an iSCSI Name. The iSCSI Name is independent of the location of the initiator and target; two formats are provided that allow the use of existing naming authorities to generate names.

One of these formats allows the use of a registered domain name as a naming authority; it is important not to confuse this with an address. The iSCSI Name is a UTF-8 text string and is defined in [NDT].

iSCSI Names are used to provide:

- An initiator identifier for configurations that provide multiple initiators behind a single IP address.
- A target identifier for configurations that present multiple targets behind a single IP address and port.
- A method to recognize multiple paths to the same device on different IP addresses and ports.
- An identifier for source and destination targets for use in third party commands.
- An identifier for initiators and targets to enable them to recognize each other regardless of IP address and port mapping on intermediary firewalls.

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. 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. The key "SessionType=Discovery" is sent by the initiator at login to indicate a discovery session.

The default name "iSCSI" is reserved and is not used as an individual initiator or target name.

iSCSI Names do not require special handling within the iSCSI layer; they are opaque and case-sensitive for purposes of comparison.

Examples of iSCSI Names:

```
iqn.1998-03.com.disk-vendor.diskarrays.sn.45678
iqn.2000-01.com.gateways.yourtargets.24
iqn.1987-06.com.os-vendor.plan9.cdrom.12345
iqn.2001-03.com.service-provider.users.customer235.host90
eui.02004567A425678D
```

iSCSI nodes also have addresses. An iSCSI address specifies a single path to an iSCSI node and has the following format:

```
<domain-name>[:<port>]
```

Where <domain-name> is one of:

- IPv4 address, in dotted decimal notation. Assumed if the name contains exactly four numbers, separated by dots (.), where each number is in the range 0..255.
- IPv6 address, in colon-separated hexadecimal notation, as specified in [RFC2373] and enclosed in "[" and "]" characters, as specified in [RFC2732].
- Fully Qualified Domain Name (host name). Assumed if the <domain-name> is neither an IPv4 nor an IPv6 address.

For iSCSI targets, the <port> in the address is optional; if specified, it is the TCP port on which the target is listening for connections. If the <port> is not specified, the default port 3260, assigned by IANA, will be assumed. For iSCSI initiators, the <port> is omitted.

Examples of addresses:

```
10.40.1.2
[FEDC:BA98:7654:3210:FEDC:BA98:7654:3210]
[1080:0:0:0:8:800:200C:417A]
[3ffe:2a00:100:7031::1]
[1080::8:800:200C:417A]
[::192.9.5.5]
mydisks.example.com
```

To assist in providing a more human-readable user interface for devices that contain iSCSI targets and initiators, a target or initiator may also provide an alias. This alias is a simple UTF-8 string, is not globally unique, and is never interpreted or used to identify an initiator or device within the iSCSI protocol. Its use is described in [NDT].

Third party commands require that protocol-specific addresses be communicated within SCSI CDBs. The iSCSI protocol-specific address consists of an iSCSI name, or an iSCSI name + TCP address.

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

2.2.8 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 10.5 Task Management Function Request.)

2.2.9 Message Synchronization and Steering

2.2.9.1 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.

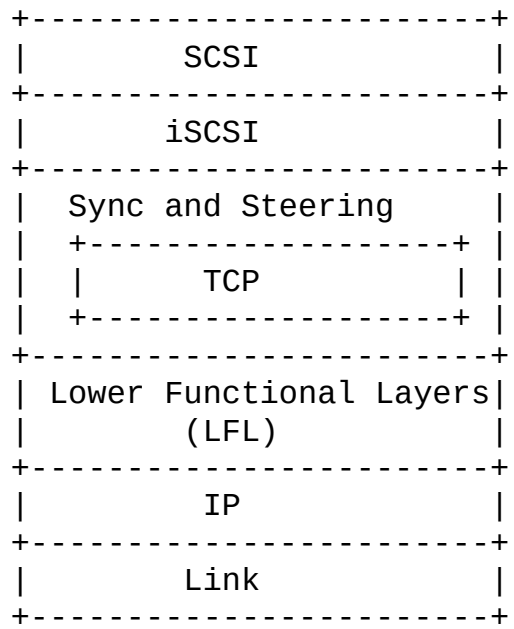
In situations where IP packets are delivered in order from the network, iSCSI message framing is not an issue and messages are processed one after the other. In the presence of IP packet reordering, (i.e., frames being dropped) legacy TCP implementations store the "out of order" TCP segments in temporary 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). Since these segments cannot be steered to the correct location, they must be saved in temporary buffers that must then be copied to the SCSI buffers.

Different schemes can be used to recover synchronization. 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.

2.2.9.2 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 is acknowledged by the receiver. (in the case of dynamically built streams, by deletion from the global table or by an additional callback).

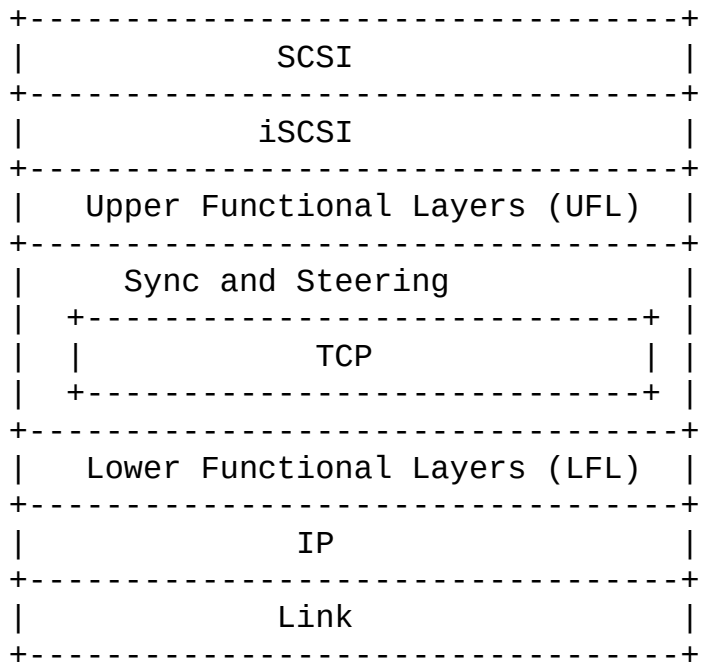
On the outgoing path, the Sync and Steering layer MUST map the outgoing stream addresses from iSCSI stream addresses to TCP stream sequence numbers.

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

2.2.9.3 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.

2.2.9.4 Sync/Steering and iSCSI PDU Size

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) since the

iSCSI header that indicates to which SCSI buffers the data is to be steered was lost. To minimize the amount of buffering, it is recommended that the iSCSI PDU size 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 size it will accept.

2.3 iSCSI Session Types

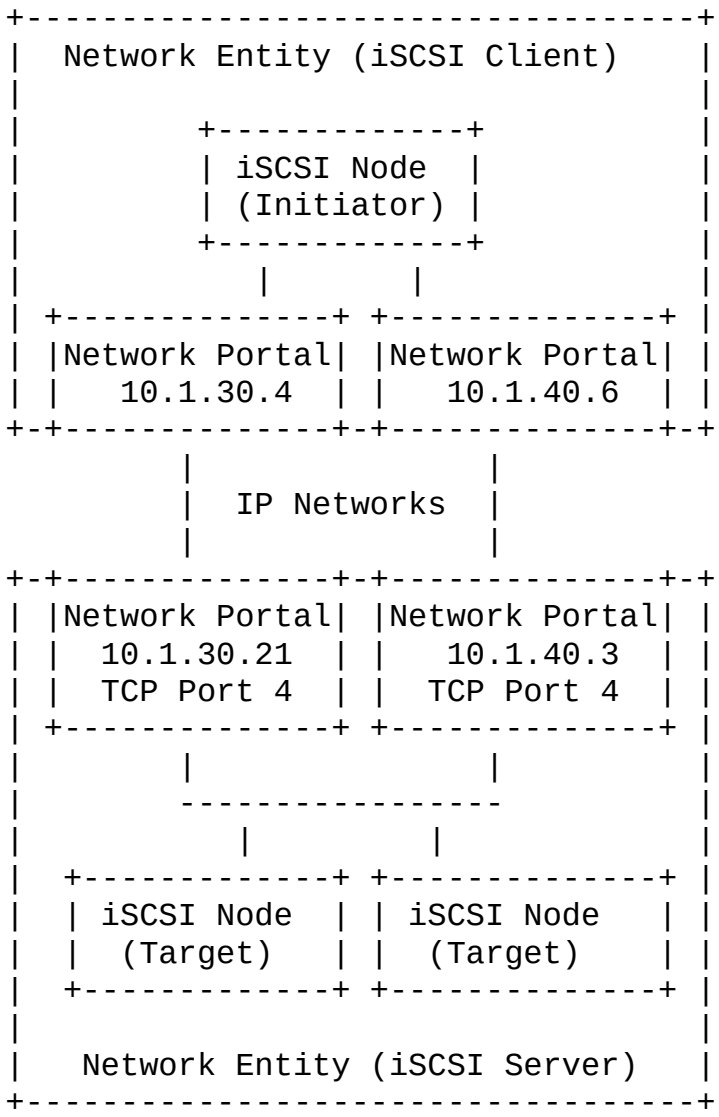
iSCSI defines two types of sessions:

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 2.2.7 Naming and Addressing and Chapter 12). The separation of the iSCSI Name from the addresses used by and for the iSCSI node allows multiple iSCSI nodes to use the same addresses, and the same iSCSI node to use multiple addresses.

c) An alias string 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.

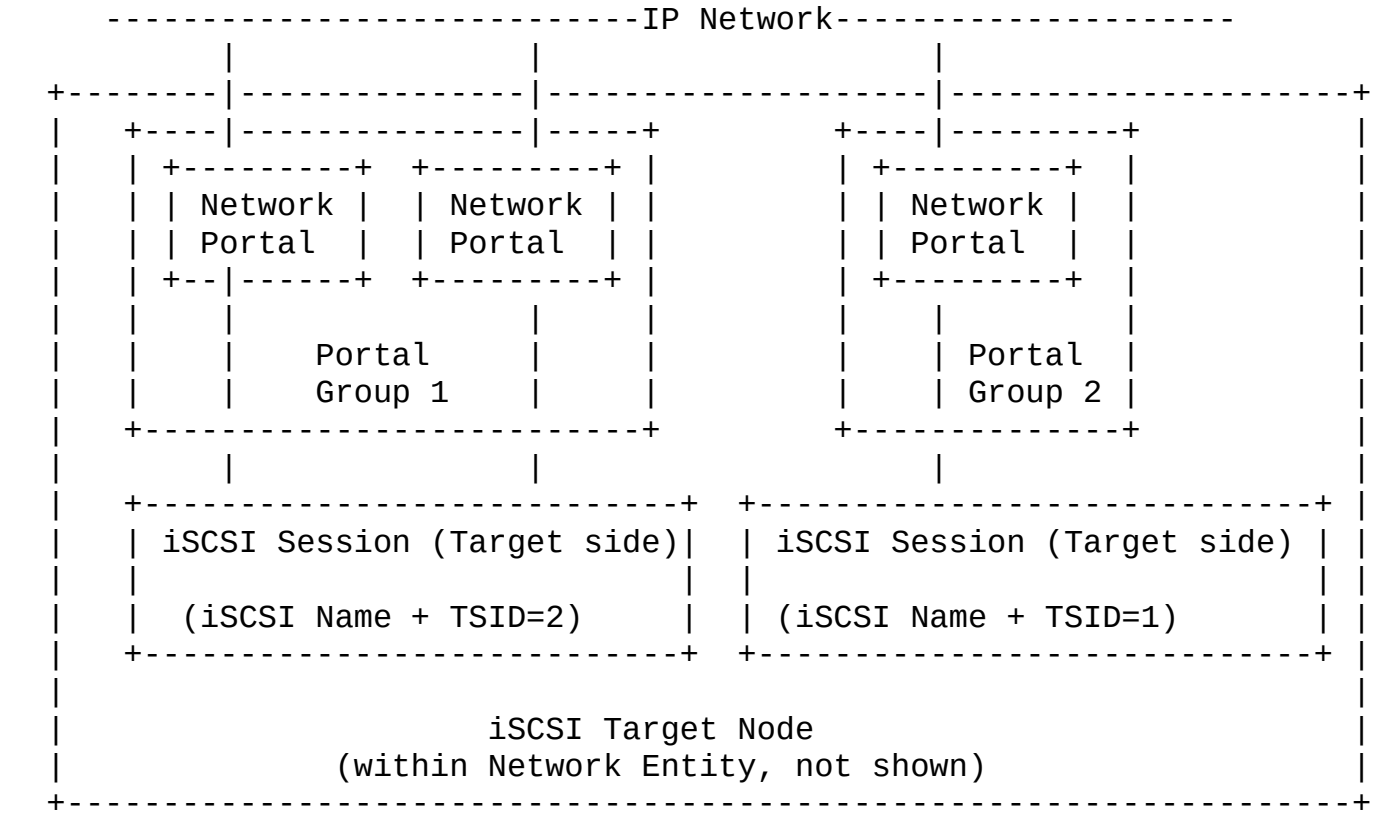
d) Network Portal - a component of a Network Entity that has a TCP/IP network address and that may be used by an iSCSI Node within that Network Entity for the connection(s) within one of its iSCSI sessions. In an initiator, it is identified by its IP address. In a target, it is identified by its IP address 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 integer value between 1 and 65535 (see Section 12.3 SendTargets). All Network Portals with the same portal group tag in the context of a given iSCSI Node are in the same Portal Group.

Both iSCSI Initiators and iSCSI Targets have portal groups, though only the iSCSI Target Portal Groups are used directly in the iSCSI protocol (e.g., in SendTargets). See Section Section 9.1.1 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.

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 the 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 should be formatted as:
- The iSCSI Name in UTF-8 format, followed by
- a null terminator (1byte), followed by
- the ASCII character 'i' (for SCSI Initiator Port) or the ASCII character 't' (for SCSI Target Port), followed by
- a null terminator (1byte), 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 size 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. The following are the two assumptions that are the basis of these requirements:

a) Between a given iSCSI Initiator and iSCSI Target, at any given time, only one session can exist with a given session identifier (SSID).

b) Between a given SCSI initiator port and SCSI target port, only one I_T nexus (session) can exist. That is, no more than one nexus relationship (parallel nexus) is allowed.

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 10.12.6 ISID.

The structure of the ISID that contains a naming authority component (see Section 10.12.6 ISID and [NDT]) provides a mechanism to facilitate compliance with the ISID rule (See also Section 9.1.1 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 9.1.1 Conservative Reuse of ISIDs). Allowing this would be analogous to a single SCSI Initiator Port having relationships (nexus) with multiple SCSI target ports on the same SCSI target device or SCSI target ports on other SCSI target devices. It is also possible to have multiple sessions with different ISIDs to the same Target Portal Group. Each such session would be considered to be with a different initiator even when the sessions originate from the same initiator device. The same ISID may be used by a different iSCSI initiator because it is the iSCSI Name together with the ISID that identifies the SCSI Initiator Port.

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

TSID RULE: The iSCSI Target SHOULD NOT select a TSID for a given login request if the resulting SSID is already in use by an existing session between the target and the requesting iSCSI Initiator. See Section 9.1.1 Conservative Reuse of ISIDs.

2.4.3.1 I_T Nexus State

Certain nexus relationships contain an explicit state (e.g., initiator-specific mode pages or reservation state) 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 7. 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 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 output parameters such as task attributes, Command Reference Number, Expected Data Transfer Length for one or both transfer directions (the later 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 input parameters 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 PDU that a target has sent (to enable the initiator to check that all arrived) - ExpDataSN, the Status Sequence Number on this connection - StatSN and the next Expected Command Sequence Number at target - ExpCmdSN, the Maximum CmdSN acceptable at 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 (Referenced Task Tag - if an abort task failed).

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 SCS 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 indicated 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 of 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.

2.5.1.7 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.

2.5.2 Requests/Responses carrying iSCSI Only Payload

2.5.2.1 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.

2.5.2.2 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 later 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.

Login Requests are always immediate.

A login request may indicate an implied logout (cleanup) of the connection to be logged in (we call this a connection restart) through the X flag in the first login request header.

2.5.2.3 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.

2.5.2.4 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 indicate in a type field in the PDU header). The run is composed of an initial item (StatSN, DataSN, R2TSN) and the number of additional missed Status, Data, or R2T PDUs (0 means only the initial). 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.

2.5.2.5 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.

2.5.2.6 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 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 MaxRecvPDULength is used during Login.

The login phase sequence of commands 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 key=value pair TargetName.

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

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 (using underlying IPsec security is specified in Chapter 8) 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 12 and the LoginOperationalNegotiation keys appear in Chapter 12.
Only a limited set of keys (marked as Declarative in Chapter 12) may be used in any of the two stages.

Neither the initiator nor the target should attempt to negotiate a parameter more than once during any login stage. Attempting to do so will result in the termination of the login and connection.

Any given Login request or response belongs to a specific stage; this determines the negotiation keys allowed with the command 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.

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 Command can come only as a response to a Login command with the T bit set to 1, and both the command and response MUST have FullFeaturePhase in the NSG field.

4.1 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.
- Session and connection Ids.
- The negotiation stage that the initiator is ready to enter.

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 of the response MUST be set to 1 and the CSG and NSG are reserved.
- Login Response with Login Accept as a final response (T bit set to 1 and the NSG in both command and response are set to FullFeaturePhase). The response includes the protocol version supported by the target and the session ID, and may include iSCSI operational or security parameters (that depend on the current stage).

- Login Response with Login Accept as a partial response (NSG not set to FullFeaturePhase in both request and response) that indicates the start of a negotiation sequence. The response includes the protocol version supported by the target and either security or iSCSI parameters (when no security mechanism is chosen) supported by the target.

If the initiator decides to forego the SecurityNegotiation stage, it issues the Login with the CSG set to LoginOperationalNegotiation and the target may reply with a Login Response that indicates that it is unwilling to accept the connection 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.

4.2 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 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 2.2.4 Text Mode Negotiation). The parameters are encoded in UTF8 as key=value. For security parameters, see Chapter 11.

-The initiator must be aware of the imminent completion of the SecurityNegotiation stage and MUST set the T bit to 1 and the NSG to what it would like the next stage to be. The target will answer with a Login response with the T bit set to 1 and the NSG to what it would like the next stage to be. 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 Session ID and the protocol version.

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 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 with the T bit set to 1 with a Login response with 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 with the T bit set to 1.

Whenever parameter action or acceptance are dependent on other parameters, the dependent parameters MUST be sent after the parameters on which they depend. If they are sent within the same command, a response for a parameter might imply responses for others.

Some session specific parameters can be specified only during the login phase begun by a login command that contains a null TSID - 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 12.

5. 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 indicator 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 to a text request with the F bit set to 1 the target responds with a text response with the F bit set to 0, the initiator should keep sending the text request (even empty) with the F bit set to 1, while it still wants to finish the negotiation, until it receives the text response with the F bit set to 1. Responding to a text request with the F bit set to 1 with an empty (no key=value pairs) response with the F bit set to 0 is 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 being answered with an F bit setting of 0 may have the next request issued with the F bit set to 0.

Whenever parameter action or acceptance is dependent on other parameters, the dependent parameters MUST be sent after the parameters on which they depend; if sent within the same command, a response for a parameter might imply responses for others.

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.

Neither the initiator nor the target should attempt to negotiate a parameter more than once during any negotiation sequence without an intervening reset. 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.

6. State Transitions

iSCSI connections and iSCSI sessions go through several well-defined states from connection creation and session creation 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.

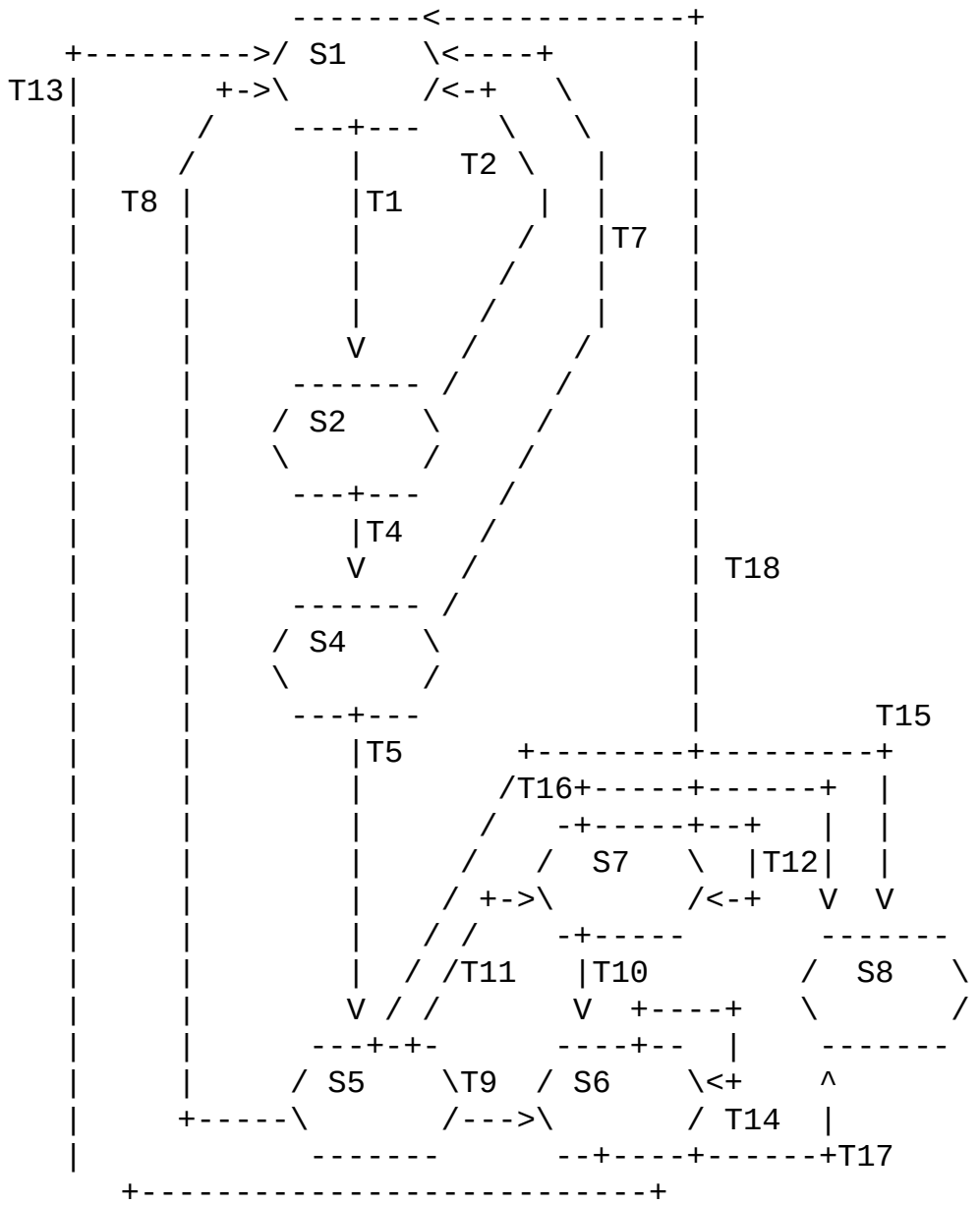
6.1 Standard Connection State Diagrams

6.1.1 Standard Connection State Diagram for an Initiator

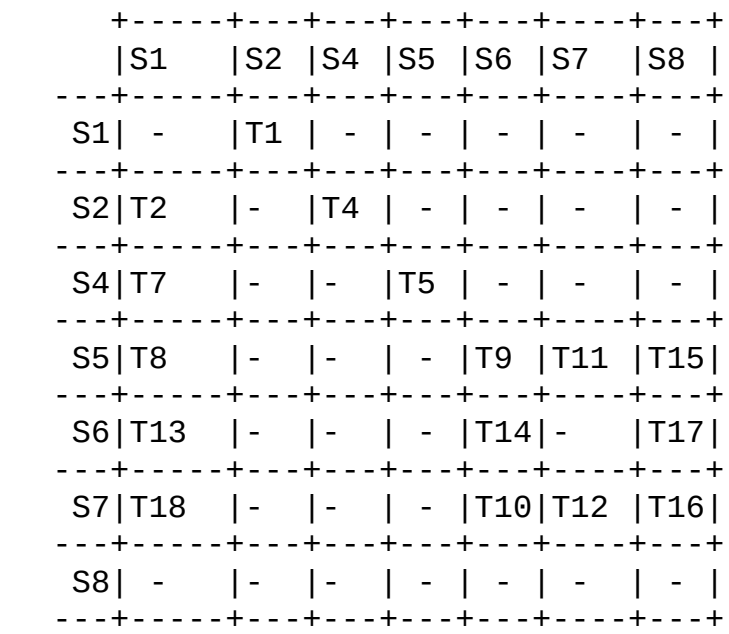
Symbolic names for States:

S1: FREE
S2: XPT_WAIT
S4: IN_LOGIN
S5: LOGGED_IN (full-feature phase)
S6: IN_LOGOUT
S7: LOGOUT_REQUESTED
S8: CLEANUP_WAIT

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.



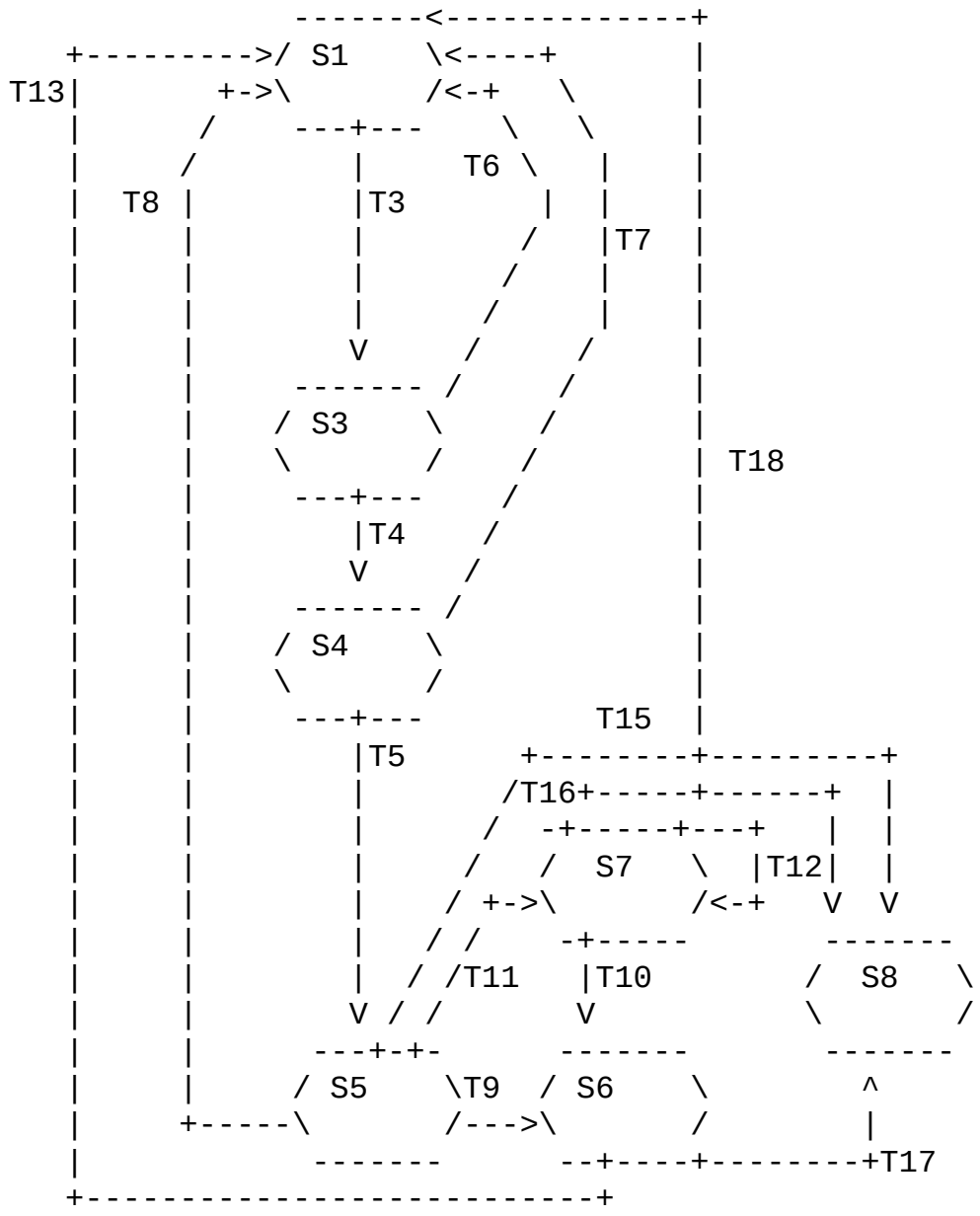
	S1	S2	S4	S5	S6	S7	S8
S1	-	T1	-	-	-	-	-
S2	T2	-	T4	-	-	-	-
S4	T7	-	-	T5	-	-	-
S5	T8	-	-	-	T9	T11	T15
S6	T13	-	-	-	T14	-	T17
S7	T18	-	-	-	T10	T12	T16
S8	-	-	-	-	-	-	-

6.1.2 Standard Connection State Diagram for a Target

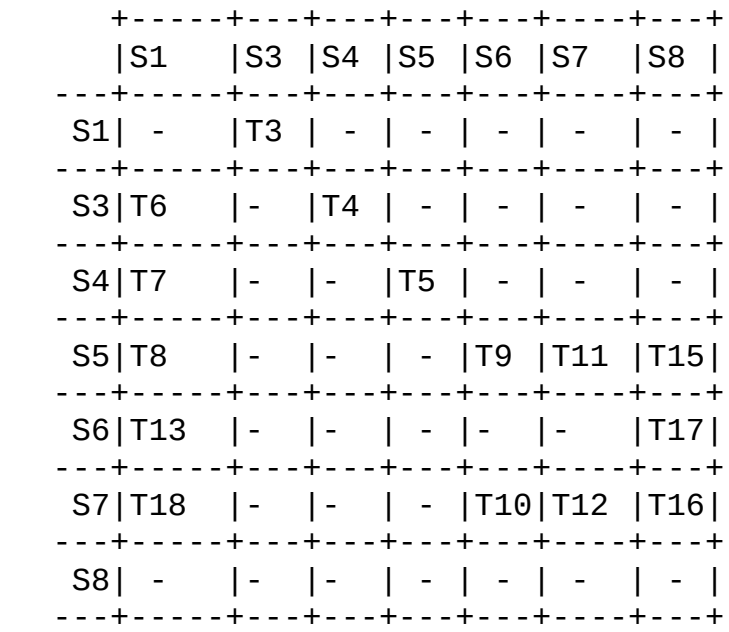
Symbolic names for States:

- S1: FREE
- S3: XPT_UP
- S4: IN_LOGIN
- S5: LOGGED_IN (full-feature phase)
- S6: IN_LOGOUT
- S7: LOGOUT_REQUESTED
- S8: CLEANUP_WAIT

The state diagram is as follows:



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



6.1.3 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.

6.1.4 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 command 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 command was received.

-T5:

-initiator: The final iSCSI Login response with a status class of zero was received.

-target: The final iSCSI Login command 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, or an internal event of sending a Logout response (success) on another connection for a close the session Logout command was received. In all these cases, the connection is to be closed.

-T7:

-initiator: The final iSCSI Login response was received with a non-zero status class, or Login timed out, or transport disconnect indication was received, or transport reset was received, or an internal event indicating a transport time-out was received, or an internal event of receiving a Logout response (success) on another connection for a close the session Logout command was received. In all these cases, the transport connection is closed.
-target: The final iSCSI Login command to conclude the Login phase was received, prompting the target to send the final iSCSI Login response with a non-zero status class, or Login timed out, or transport disconnect indication was received, or transport reset was received, or an internal event indicating a transport timeout was received, or an internal event of sending a Logout response (success) on another connection for a close the session Logout command was received. In all these cases, the connection is to be closed.

-T8:

-initiator: An internal event of receiving a Logout response (success) on a another connection for a close the session Logout command 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 command was 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 command 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 command 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 command was received. In both 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.

-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.
-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 command was received.

-target: An internal event of sending a Logout response (success) on another connection for a "close the session" Logout command was received.

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

6.2 Connection Cleanup State Diagram for Initiators and Targets

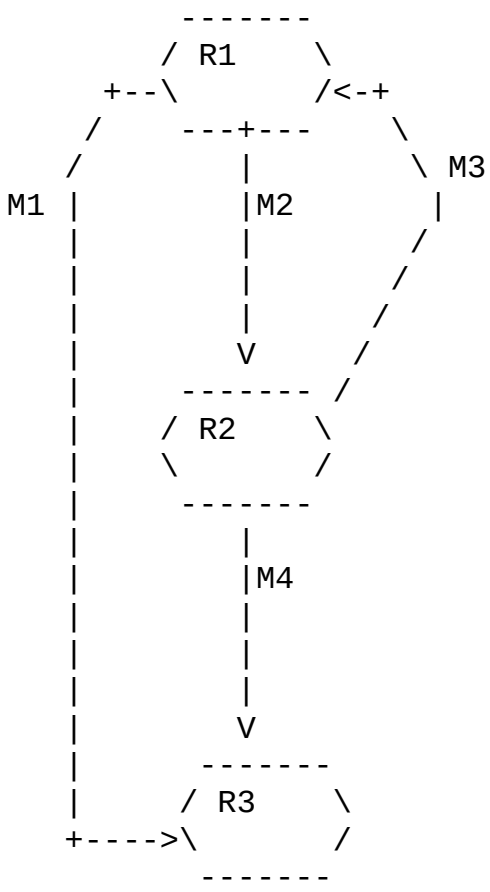
Symbolic names for states:

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 (see Section

10.12.2 X - Restart Connection) for that CID. In either case, the protocol exchanges on CSM-E or CSM-I to 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.

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	-	-	-

6.2.1 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.

6.2.2 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 command.
-M2: An implicit/explicit logout process was initiated by the initiator.
  -In CSM-I usage:

-initiator: An internal event requesting the CID reinstatement was received, thus prompting a connection reinstatement Login to be sent transitioning to state IN_LOGIN.
-target: A connection 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 command 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 command 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 command 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 command was received.

6.3 Session State Diagram

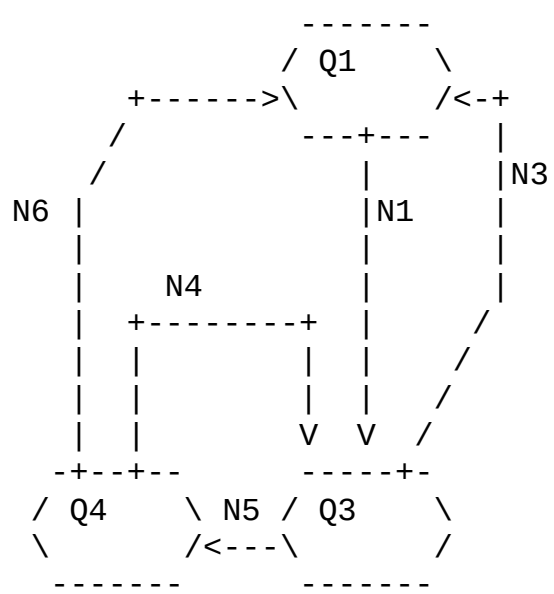
Session continuation is the process by which the state of a pre-existing session continues to be in use by either connection reinstatement (see Section 10.12.2 X - Restart Connection), or by adding a connection with a new CID. Either of these actions associates the new transport connection with the pre-existing session state.

6.3.1 Session State Diagram for an Initiator

Symbolic Names for States:

Q1: FREE
Q3: LOGGED_IN
Q4: FAILED

The state diagram is as follows:



State transition table:

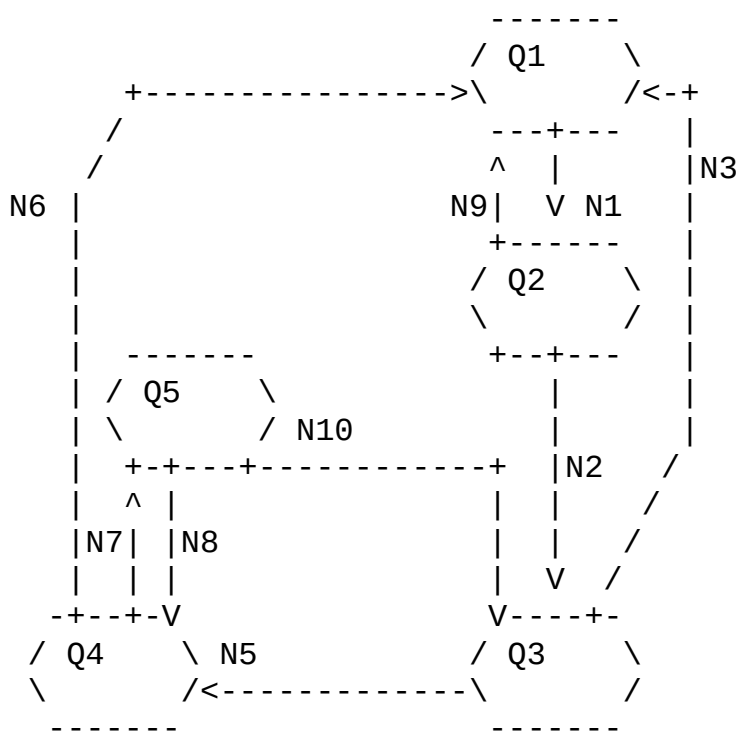
```
     +----+----+----+
     |Q1  |Q3  |Q4  |
-----+----+----+----+
 Q1  | -  |N1  | -  |
-----+----+----+----+
 Q3  |N3  | -  |N5  |
-----+----+----+----+
 Q4  |N6  |N4  | -  |
-----+----+----+----+
```

6.3.2 Session State Diagram for a Target

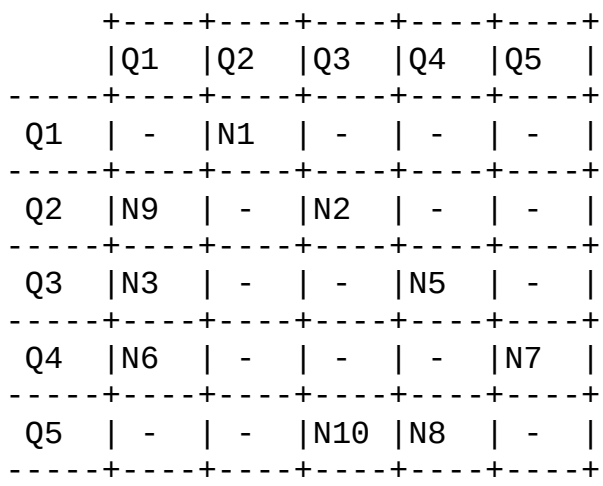
Symbolic Names for States:

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

The state diagram is as follows:



State transition table:



6.3.3 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.

6.3.4 State Transition Descriptions for Initiators and Targets

-N1:

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

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

-N3:
-initiator: Graceful closing of the session was completed either via a "close the session" Logout, or last successful "close the connection" Logout.
-target: Graceful closing of the session was completed either via a "close the session" Logout, or last successful "close the connection" Logout.

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

-N5:
-initiator: Session failure occurred (all connections reported CLEANUP_WAIT).
-target: Session failure occurred (all connections reported CLEANUP_WAIT).

-N6:
-initiator: Session state timeout occurred, or an implicit session logout (by reuse of ISID with TSID=0) 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 an implicit session logout (by reuse of ISID with TSID=0) 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.

7. 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 is available (in host memory) for retransmission while the command is outstanding. It is also assumed that at 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 F. - 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.

7.1 Retry and Reassign in Recovery

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

7.1.1 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. 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.

7.1.2 Allegiance Reassignment

By issuing a "task reassign" task management command (Section 10.5.1 Function), the initiator signals its intent to continue an already active command (but with no current connection allegiance) as part of connection recovery. This means that a new connection allegiance is established for the command, that associates it to the connection on which the task management command is being issued.

In reassigning connection allegiance for a command, the targets SHOULD continue the command from its current state, for example taking advantage of ExpDataSN in the command PDU for read commands (which must be set to zero if there was no data transfer). However, targets MAY choose to send/receive the entire data on a reassignment of connection allegiance, and it is not considered an error.

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".

7.2 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 since the command itself is being discarded.

The above rule means that the initiators can eventually expect a response even on Rejects, if the Reject is not for the command itself. The non-command Rejects only have diagnostic value in logging the errors, and they can be used for retransmission decisions by the initiators.

The CmdSN of the rejected 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). This is true even when the CmdSN can be reliably ascertained, as in the case of a data digest error on immediate data. However, when the DataSN of a rejected data PDU can be ascertained, a target MUST advance ExpDataSN for the current 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.

7.3 Format Errors

Explicit violations of the PDU layout rules stated in this document are format errors. Violations, when detected, usually 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 7.11.4 Session Recovery).

7.4 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 receives any iSCSI PDU with a header digest error, it MUST silently discard the PDU.

When a target receives any iSCSI PDU with a payload digest error, it MUST answer with a Reject iSCSI 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 the reason "protocol service CRC error" (Section 10.4.3 Response). 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 header digest error, it MUST discard the PDU.

LS: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 "Data-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 iSCSI response reason(Section 10.4.3 Response) of "SNACK rejected".

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 the "SNACK rejected" reason (Section 10.4.3 Response) disregarding any received status PDU, but must wait for the status to be received before doing so. [OR]

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

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

a) Request PDU retransmission with a status SNACK. [OR]
b) Logout the connection for recovery and continue the tasks on a different connection instance as described in Section 7.1 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).

7.5 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 7.4 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 7.4 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 7.4 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.

7.6 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 7.1 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: "Task does not exist". This response concludes the task on both ends.

7.7 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 aborted.
- The text request was answered with a reject.

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).

7.8 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 be only be addressed by fixing the implementations; iSCSI defines Reject and response codes to enable this.

7.9 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 7.11.3 Connection Recovery).
- Logout the connection with the reason code "closes the connection" (Section 10.14.3 Reason Code), re-issue missing commands, and implicitly terminate all active commands. This option requires support for the within-connection recovery class (Section 7.11.2 Recovery Within-connection).
- Perform session recovery (Section 7.11.4 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).

7.10 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.

7.11 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.

7.11.1 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 7.4 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 7.5 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 7.5 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 7.4 Digest Errors, using the option of a SNACK.
b) Sequence reception timeout (no status) - dealt with as specified in Section 7.5 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 7.5 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.

7.11.2 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 7.1 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 7.4 Digest Errors using the option of a SNACK. In the second case, sequence error handling is done as specified in Section 7.5 Sequence Errors, using the option of a SNACK.

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

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

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

7.11.3 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 associated with the failed connection on another connection (that MAY be a newly established connection) using the "Task reassign" task management function (see Section 10.5.1 Function).

- 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.

7.11.4 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).

Reserve-Release managed SCSI reservations ("Regular" reservations) that are secured during a given iSCSI session persist until they are cleared using regular SCSI means or (in the absence of such,) until the session object is cleared - i.e. when the FREE state is reached. Only the session continuation (section 6.3) therefore preserves the Regular reservations.

Persistent SCSI reservations are not affected by iSCSI session failures, and only the regular SCSI means can be used to handle these reservations when the session is reconstructed (necessarily between the same SCSI ports and so with the same nexus identifier).

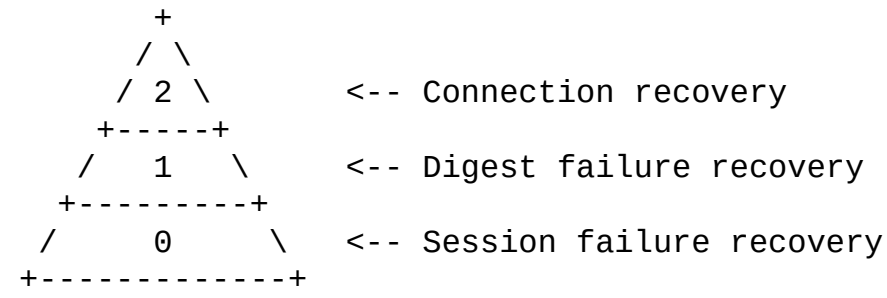
7.12 Error Recovery Hierarchy

The error recovery classes and features described are organized into a hierarchy for ease in understanding and to limit the myriad of imple-

mentation 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 requirement.
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 7.11.4 Session Recovery)
1	Digest failure recovery (See Note below.)
2	Connection recovery class (Section 7.11.3 Connection Recovery)

Note: Digest failure recovery is comprised of two recovery classes: Within-Connection recovery class (Section 7.11.2 Recovery Within-con-

nection) and Within-Command recovery class (Section 7.11.1 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

8. Security Considerations

Historically, native storage systems have not had to consider security because their environments offered minimal security risks. That is, these environments consisted of storage devices either directly attached to hosts or connected via a subnet 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.

8.1 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].

8.2 In-band Initiator-Target Authentication

With this mechanism, 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, since 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.

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

The CHAP authentication method (see Chapter 13) is vulnerable to an off-line dictionary attack. In environments where this attack is a concern, CHAP SHOULD NOT be used without additional protection. Underlying IPsec encryption provides protection against this attack.

The strength of the SRP authentication method (specified in Chapter 13) 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 satisfy the above requirements (and otherwise, abort the connection). This verification MAY start by trying to match N and g with a well-known group that satisfies the above requirements.
Well-known SRP groups are provided in [SEC-IPS].

Compliant iSCSI initiators and targets MUST at least implement the SRP authentication method [RFC2945] (see Chapter 11).

8.3 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].

8.3.1 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 in tunnel mode [RFC2406] with the following iSCSI specific requirements:

- HMAC-SHA1 MUST be implemented [RFC2404].
- AES CBC MAC with XCBC extensions SHOULD be implemented [AES], [XCBC] (NOTE: Still subject to the IETF-IPsec WG's standardization plans).

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, an iSCSI implementation that operates at speeds of 1 Gbps or less MAY implement the IPsec sequence number extension [SEQ-EXT].
Implementation operation at speeds of 10 Gbps or faster SHOULD implement the sequence number extension.

8.3.2 Confidentiality

Confidentiality is provided by encrypting the data in every packet. Confidentiality SHOULD always be used together with data integrity

and authentication to provide comprehensive protection against eavesdropping, message insertion, deletion, modification, and replaying.

An iSCSI compliant initiator or target MUST provide confidentiality by implementing IPsec [RFC2401] with ESP in tunnel mode [RFC2406] 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.

8.3.3 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.

- Both IKE Main Mode and Aggressive Mode MUST be supported. 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, and MUST

carry individual addresses and MUST NOT use the IP Subnet or IP Address Range formats.

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

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

9. Notes to Implementers

This section notes some of the performance and reliability considerations of the iSCSI protocol. This protocol was designed to allow efficient silicon and software implementations. The iSCSI 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.

9.1 Multiple Network Adapters

The iSCSI protocol allows multiple connections, not all of which need to go over the same network adapter. If multiple network connections are to be utilized with hardware support, the iSCSI protocol command-data-status allegiance to one TCP connection ensures that there is no need to replicate information across network adapters or otherwise require them to cooperate.

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

9.1.1 Conservative Reuse of ISIDs

Historically, the SCSI model (and implementations and applications based on that model) has assumed that SCSI ports are static, physical entities. Recent extensions to the SCSI model have taken advantage of persistent worldwide unique names for these ports. In iSCSI however, the SCSI initiator ports are the endpoints of dynamically created sessions, so the 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 reuse to the same target portal group. It does not "preclude" reuse to other target portal groups.
The principle of conservative reuse "encourages" reuse to other target portal groups. When a SCSI target device sees the same (InitiatorName, ISID) pair in different sessions to different target portal groups, it can identify the underlying SCSI Initiator Port on each session as the same SCSI port. In effect, it can recognize multiple paths from the same source.

9.1.2 iSCSI Name and ISID/TSID 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 and TSID for targets) to enforce both the ISID and TSID 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 to provide mechanisms for configuration of the iSCSI Node Name and for configuration and/or coordination of TSIDs across the portal groups instantiated by multiple instances of these components within a target. One mechanism is to allow for static or dynamic partitioning of the TSID namespace among the portal groups. Such a partitioning allows each portal group to act independently of other portal groups when assigning TSIDs, and facilitates enforcement of the TSID RULE (Section 2.4.3 Consequences of the Model).

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

Recognizing that such initiator APIs are not available today, other implementations of the role of this entity are possible.
For example, a human may instantiate the (common) Node name as part of the installation process of each iSCSI component involved in session creation and login. This may be done either by pointing the component to a vendor-specific location for this datum or to a system-wide location. The structure of the ISID namespace (see Section 10.12.6 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 SW, such as the OS vendor) or via private APIs driven by the vendor's own SW.

9.2 Autosense and Auto Contingent Allegiance (ACA)

Autosense refers to the automatic return of sense data to the initiator in case a command did not complete successfully. iSCSI mandates support for 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 mandates support for ACA.

9.3 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 since the retry was issued. This requirement can be fulfilled by an implementation in several ways.

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

9.4 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.

9.5 Unsolicited Data and Performance

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).

However, negotiating an amount of unsolicited data for writes and sending less than the negotiated amount when the total data amount to be sent by a command is larger than the negotiated amount may negatively impact performance and may not be supported by all the targets.

10. iSCSI PDU Formats

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

10.1 iSCSI PDU Length and Padding

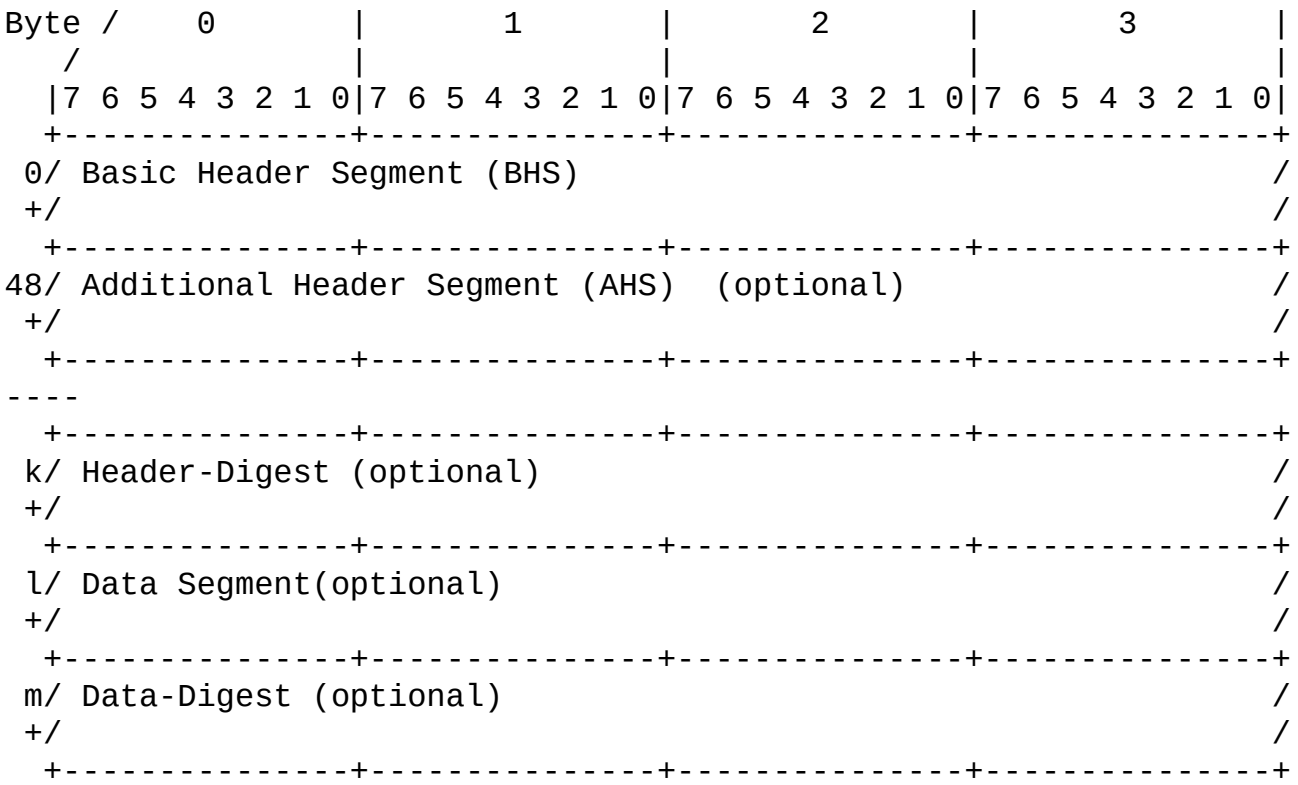
iSCSI PDUs are padded to the closest integer number of four byte words. The padding bytes SHOULD be 0.

10.2 PDU Template, Header, and Opcodes

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

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

The overall structure of a PDU is as follows:

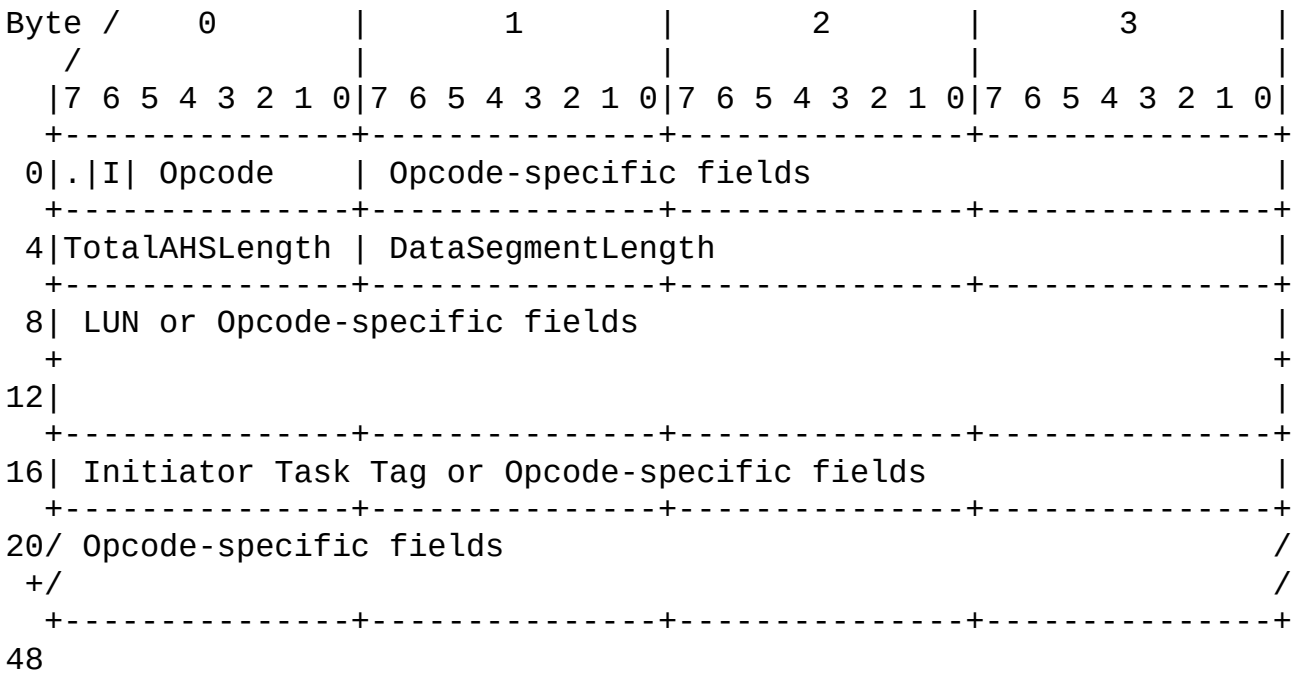


All PDU segments and digests are padded to an integer number of four byte words. The padding bytes SHOULD be sent as 0.

10.2.1 Basic Header Segment (BHS)

The BHS is 48 bytes long. The Opcode, TotalAHSLength, 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:



10.2.1.1 I

For request PDUs, the I bit set to 1 is an immediate delivery marker. This bit is always 1 for response PDUs (PDUs from target to initiator).

10.2.1.2 Opcode

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

The Opcodes are divided into two categories: initiator opcodes and target opcodes. Initiator opcodes are in PDUs sent by the 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 Command

0x04 Text request
0x05 SCSI Data-out (for WRITE operations)
0x06 Logout Command
0x10 SNACK Request
0x1c-0x1e Vendor specific codes

Target opcodes are:

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

All other opcodes are reserved.

10.2.1.3 Opcode-specific Fields

These fields have different meanings for different opcode types.

10.2.1.4 TotalAHSLength

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

10.2.1.5 DataSegmentLength

This is the data segment payload length in bytes (excluding padding).

10.2.1.6 LUN

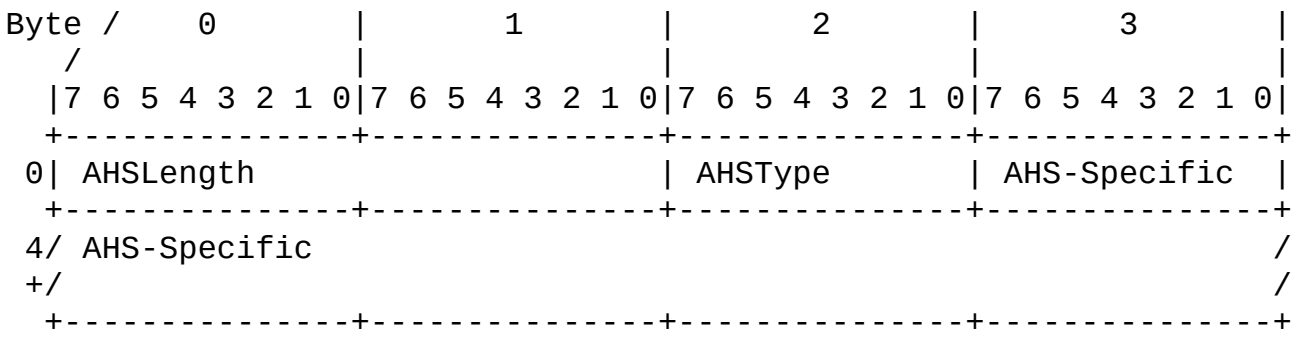
Some opcodes operate on a specific Logical Unit. The Logical Unit Number (LUN) field identifies which Logical Unit. If the opcode does not relate to a Logical Unit, this field is either ignored or may be used in an opcode specific way. The LUN field is 64-bits and should be formatted in accordance with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

10.2.1.7 Initiator Task Tag

The initiator assigns a Task Tag to each iSCSI task it issues. While a task exists, this tag MUST uniquely identify it 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.

10.2.2 Additional Header Segment (AHS)

The general format of an AHS is:



10.2.2.1 AHSType

The AHSType field is coded as follows:

bit 7-6 - Reserved

bit 5-0 - AHS code

0 - Reserved
1 - Extended CDB
2 - Expected Bidirectional Read Data Length
3 - 59 Reserved
60- 63 Non-iSCSI extensions

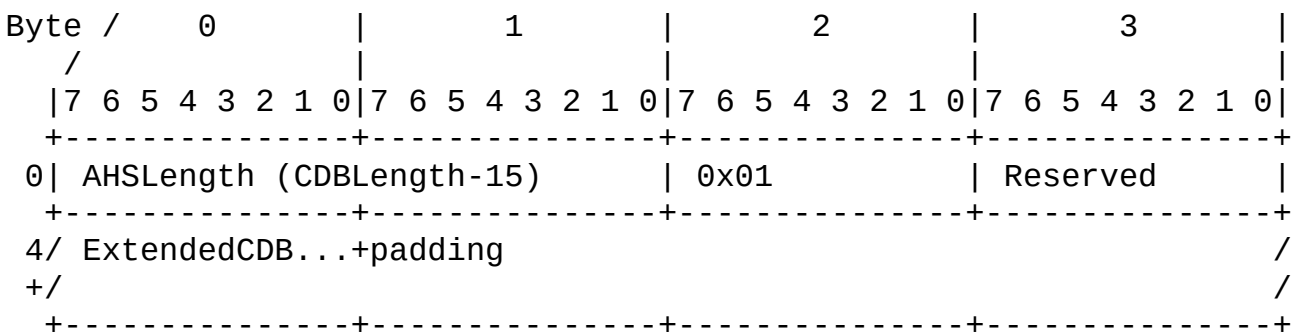
10.2.2.2 AHSLength

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).

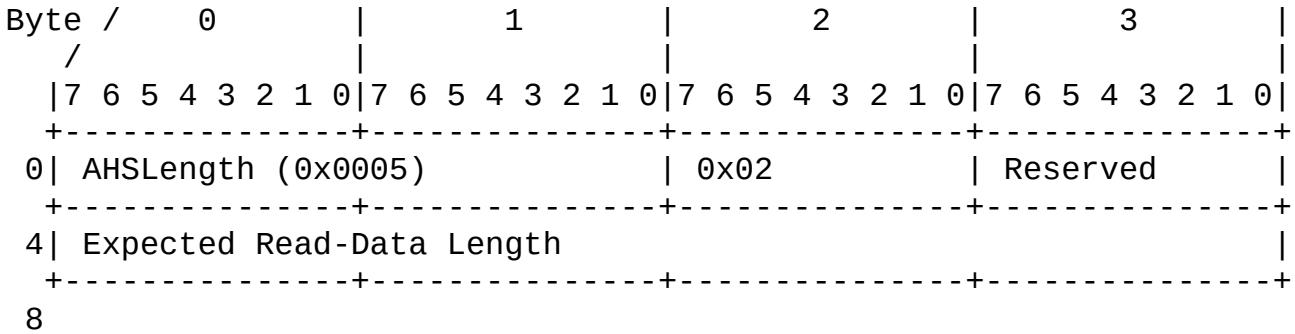
10.2.2.3 Extended CDB AHS

The format of the Extended CDB AHS is:



10.2.2.4 Bidirectional Expected Read-Data Length AHS

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



10.2.3 Header Digest and Data Digest

Optional header and data digests protect the integrity of the header and data, respectively. The digests, if present, are located, respectively, after the header and PDU-specific data and include 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.

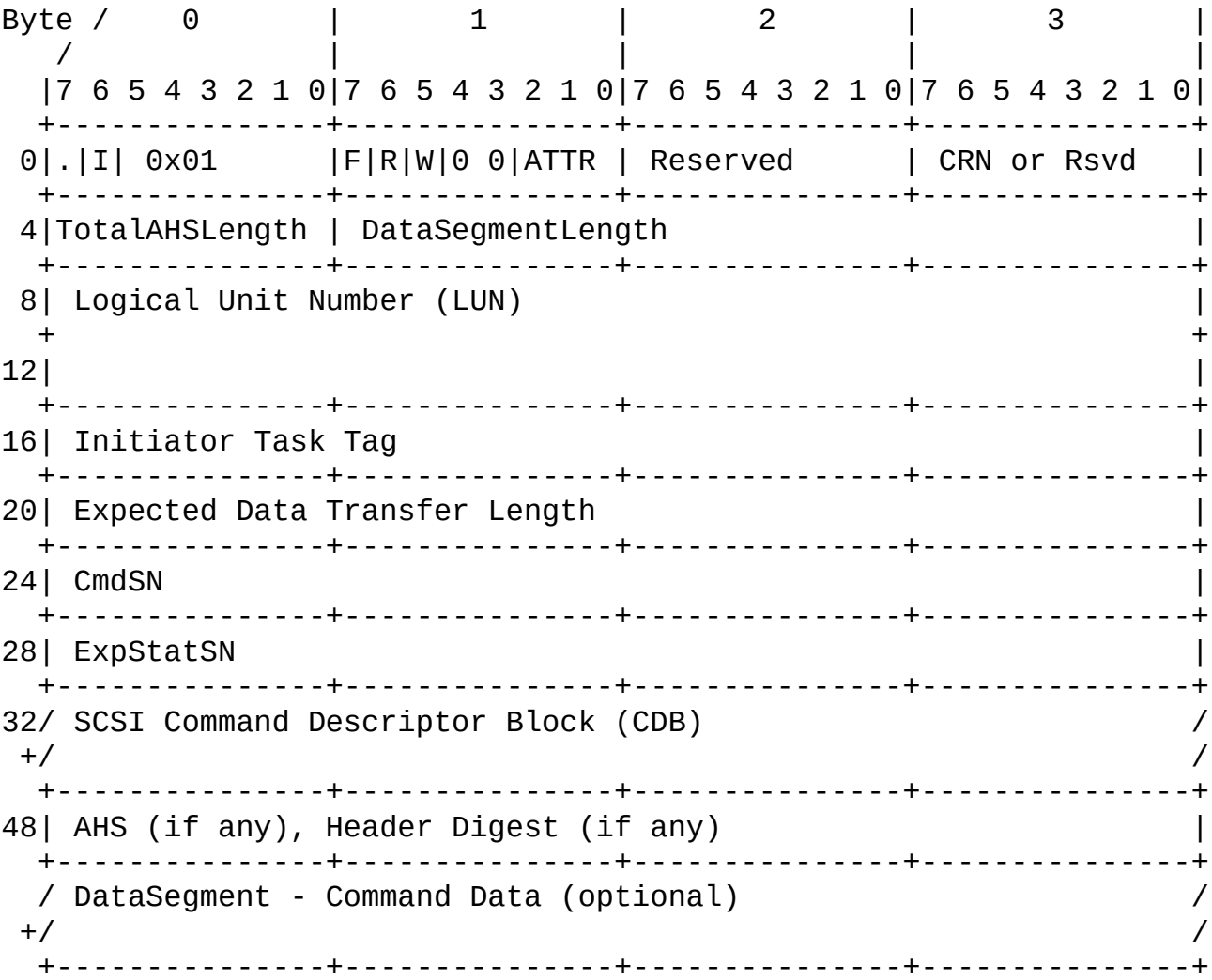
A zero-length Data Segment also implies a zero-length data-digest.

10.2.4 Data Segment

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

10.3 SCSI Command

The format of the SCSI Command PDU is:



10.3.1 Flags and Task Attributes (byte 1)

The flags for a SCSI Command are:

bit 7 (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 6 (R) set to 1 when input data is expected.

bit 5 (W) set to 1 when output data is expected.

bit 4-3 Reserved

bit 2-0 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 Lengths are not 0.

10.3.2 CRN

SCSI command reference number - if present in the SCSI execute command arguments (according to [SAM2]).

10.3.3 CmdSN - Command Sequence Number

Enables ordered delivery across multiple connections in a single session.

10.3.4 ExpStatSN

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

10.3.5 Expected Data Transfer Length

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

expects the target to transfer to the initiator. It corresponds to the SAM2 byte count.

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

If the Expected Data Transfer Length for a write and the length of the immediate data part that follows the command (if any) are the same, than 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 FirstBurstSize (the negotiated maximum amount of unsolicited data the target will accept), the initiator SHOULD send the maximum size of unsolicited data. The target MAY terminate a command in error for which the Expected Data Transfer Length is higher than the FirstBurstSize and for which the initiator sent less than the FirstBurstSize unsolicited 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.

10.3.6 CDB - SCSI Command Descriptor Block

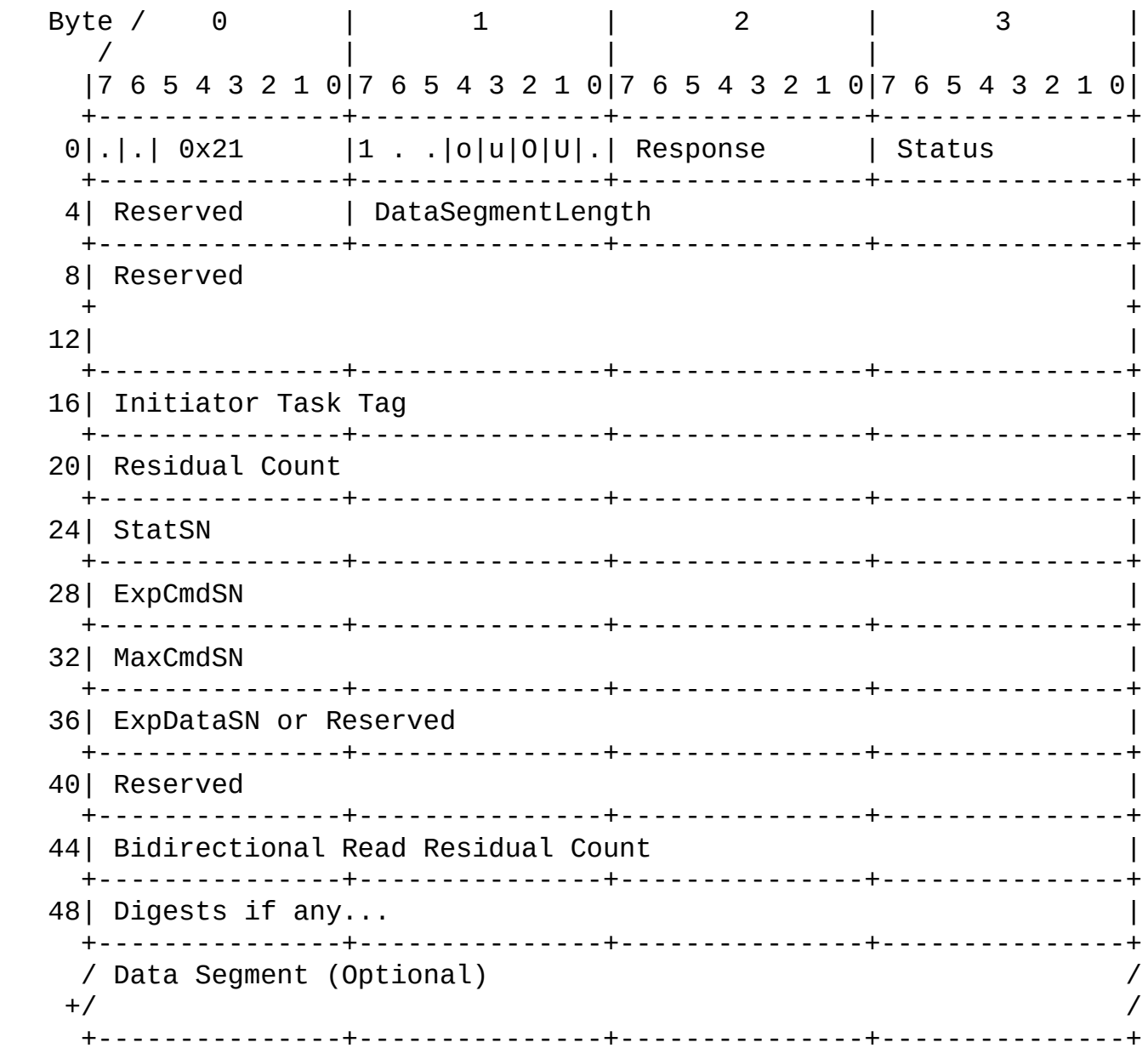
There are 16 bytes in the CDB field to accommodate the commonly used CDBs. Whenever the CDB is larger than 16 bytes, an Extended CDB AHS MUST be used to contain the CDB spillover.

10.3.7 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 same PDU (both cases are referred to as immediate data). These data are governed by the general rules for solicited vs. unsolicited data.

10.4 SCSI Response

The format of the SCSI Response PDU is:



10.4.1 Flags (byte 1)

bit 6-5 Reserved

bit 4 - (o) set for Bidirectional Read Residual Overflow. In this case, the b 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 3 - (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 2 - (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 1 - (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 expected to be transferred. For a bidirectional operation, the Residual Count contains the residual for the write operation.

bit 0 - (0) Reserved

Bits O and U and bits o and u are mutually exclusive.
For a response other than "Command Completed at Target" bit 4-1 MUST be 0.

10.4.2 Status

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

10.4.3 Response

This field contains the iSCSI service response.

iSCSI service response codes defined in this specification are:

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

The Response is used to report a Service Response. The exact mapping of the iSCSI response codes to SAM service response symbols is outside the scope of this document.

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:

Reason	Sense Key	Additional Sense Code & Qualifier
Unexpected unsolicited data	Aborted Command-0B	ASC = 0x0c ASCQ = 0x0c Write Error
Not enough unsolicited 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 "Not enough unsolicited data" condition only if it does not support output (write) operations in which the total data length is higher than FirstBurstSize, but the initiator sent less than FirstBurstSize amount of unsolicited data, and out-of-order R2Ts cannot be used.

10.4.4 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.

10.4.5 Bidirectional Read Residual Count

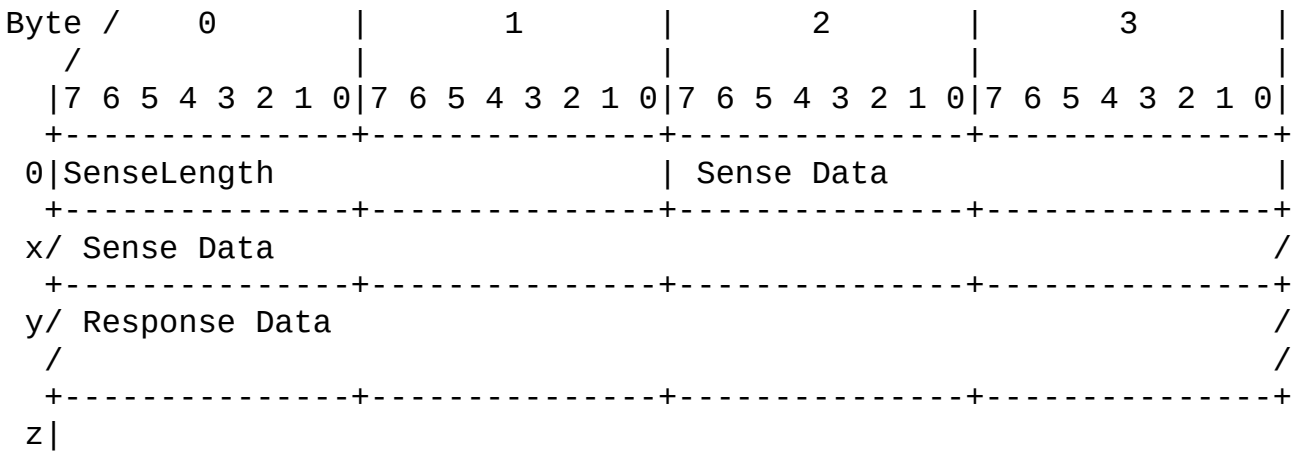
The Bidirectional Read Residual Count field is only valid in the case where either the u bit or the o bit is set. If neither bit is set, the Bidirectional Read Residual Count field SHOULD be zero. If the o bit is set, the Bidirectional Read Residual Count indicates the number of bytes that were not transferred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

10.4.6 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:



10.4.6.1 SenseLength

Length of Sense Data.

10.4.7 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.

10.4.8 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.

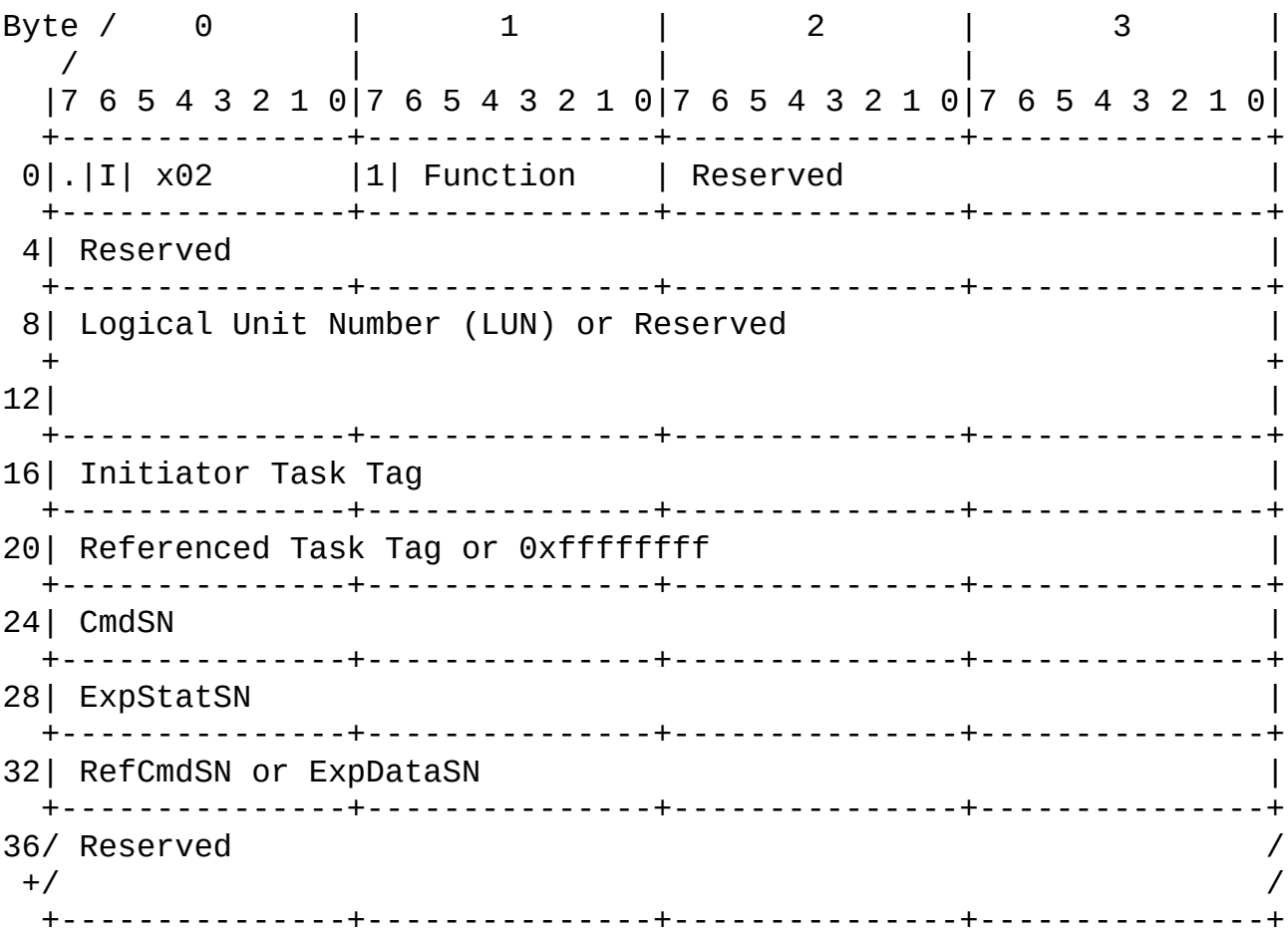
10.4.9 ExpCmdSN - Next Expected CmdSN from this Initiator

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.

10.4.10 MaxCmdSN - Maximum CmdSN Acceptable 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.

10.5 Task Management Function Request



10.5.1 Function

The Task Management functions provide an initiator with a way to explicitly control the execution of one or more Tasks (SCSI and iSCSI tasks). The Task Management functions 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 10.6 Task Management Function Response. All these functions apply to the referenced tasks regardless of whether they are proper SCSI tasks or tagged iSCSI operations. Task management requests must act on all the commands 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 requests - 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.

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 being 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 TARGET RESET function (WARM and COLD) implementation is OPTIONAL and when implemented, should act as described below. Target Reset MAY also be subject to SCSI access controls for the requesting initiator. When authorization fails at the target, the appropriate response as described in Section 10.6 Task Management Function Response must be returned by the target.

For 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.

In addition, for the TARGET COLD RESET, the target MUST then terminate all of its TCP connections to all initiators (all sessions are terminated).

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 7.1 Retry and Reassign in Recovery.

TASK REASSIGN MUST be issued as an immediate command.

10.5.2 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.

10.5.3 Referenced Task Tag

The Initiator Task Tag of the task to be aborted for the TASK ABORT function or reassigned for the TASK REASSIGN function.
For all the other functions this field is reserved.

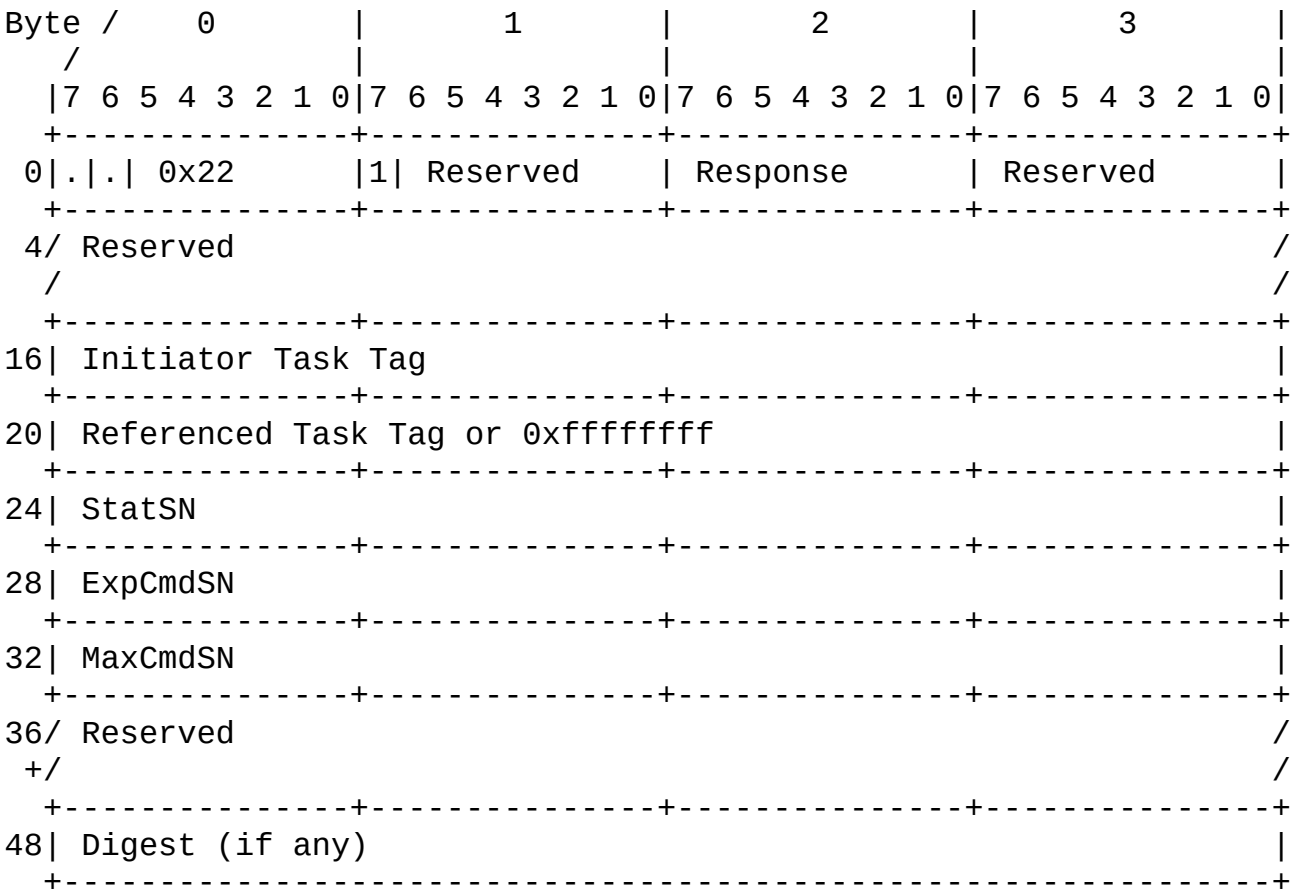
10.5.4 RefCmdSN or ExpDataSN

For ABORT TASK, this is the task CmdSN of the task to be aborted. If RefCmdSN does not match the CmdSN of the command to be aborted at the target, the abort action MUST NOT be performed and the response MUST be 'function rejected'.

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 target MUST retransmit all data previously transmitted in DataIN 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 MaxRecvPDULength in the reassignment.

Otherwise, this field is reserved.

10.6 Task Management Function Response



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

10.6.1 Response

The target provides a Response, which may take on the following values:

a) 0 - Function Complete
b) 1 - Task does not exist
c) 2 - LUN does not exist.
d) 3 - Task still allegiant.
e) 4 - Task 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 7.1.2 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, if needed, is outside the scope of this document.

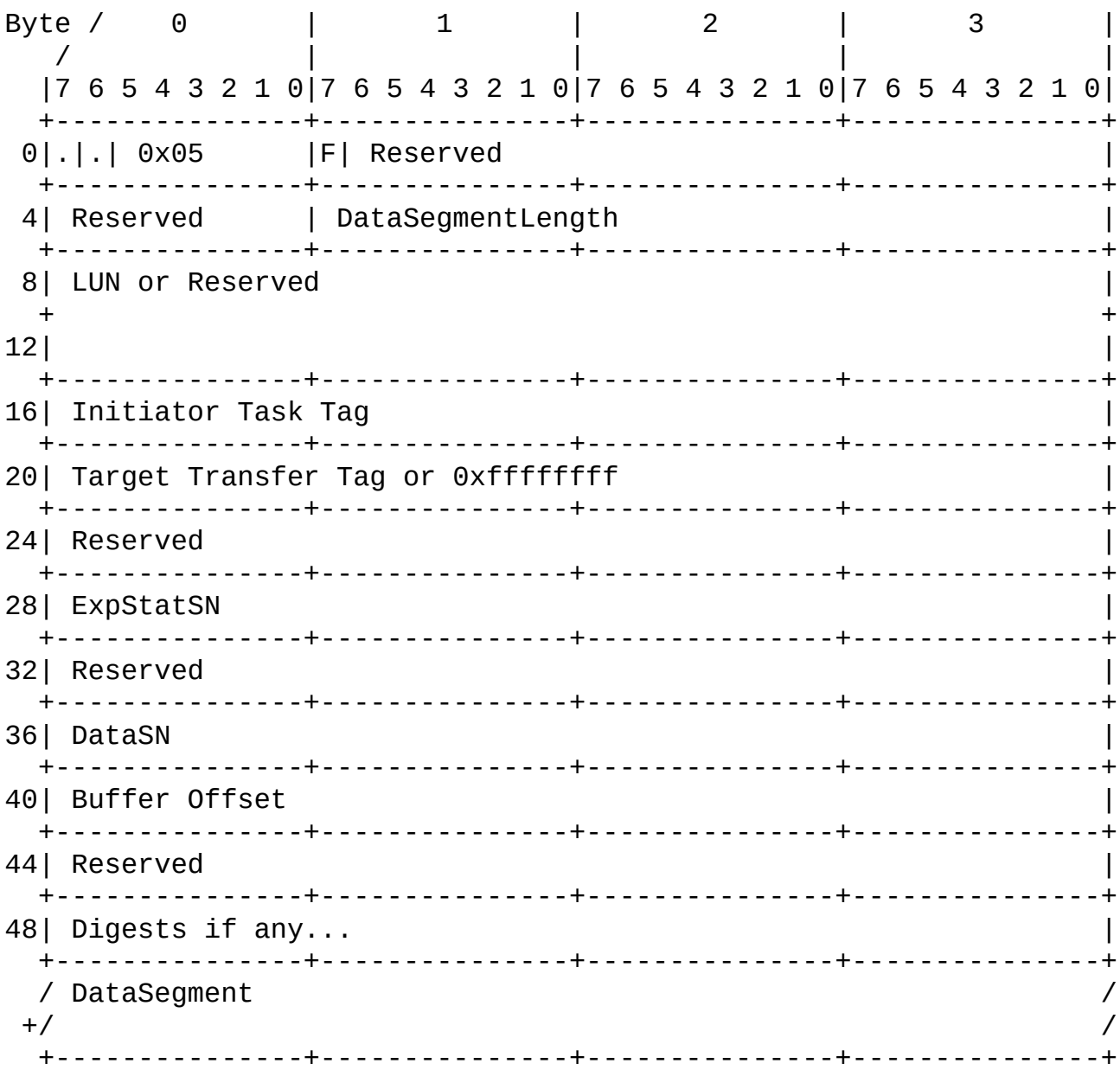
The response to ABORT TASK SET and CLEAR TASK SET MUST be issued by the target only after all the commands affected have been received by the target, the corresponding task management functions have been executed by the SCSI target and the delivery of all previous responses has been confirmed (acknowledged through ExpStatSN) by the initiator on all connections of this session.

10.6.2 Referenced Task Tag

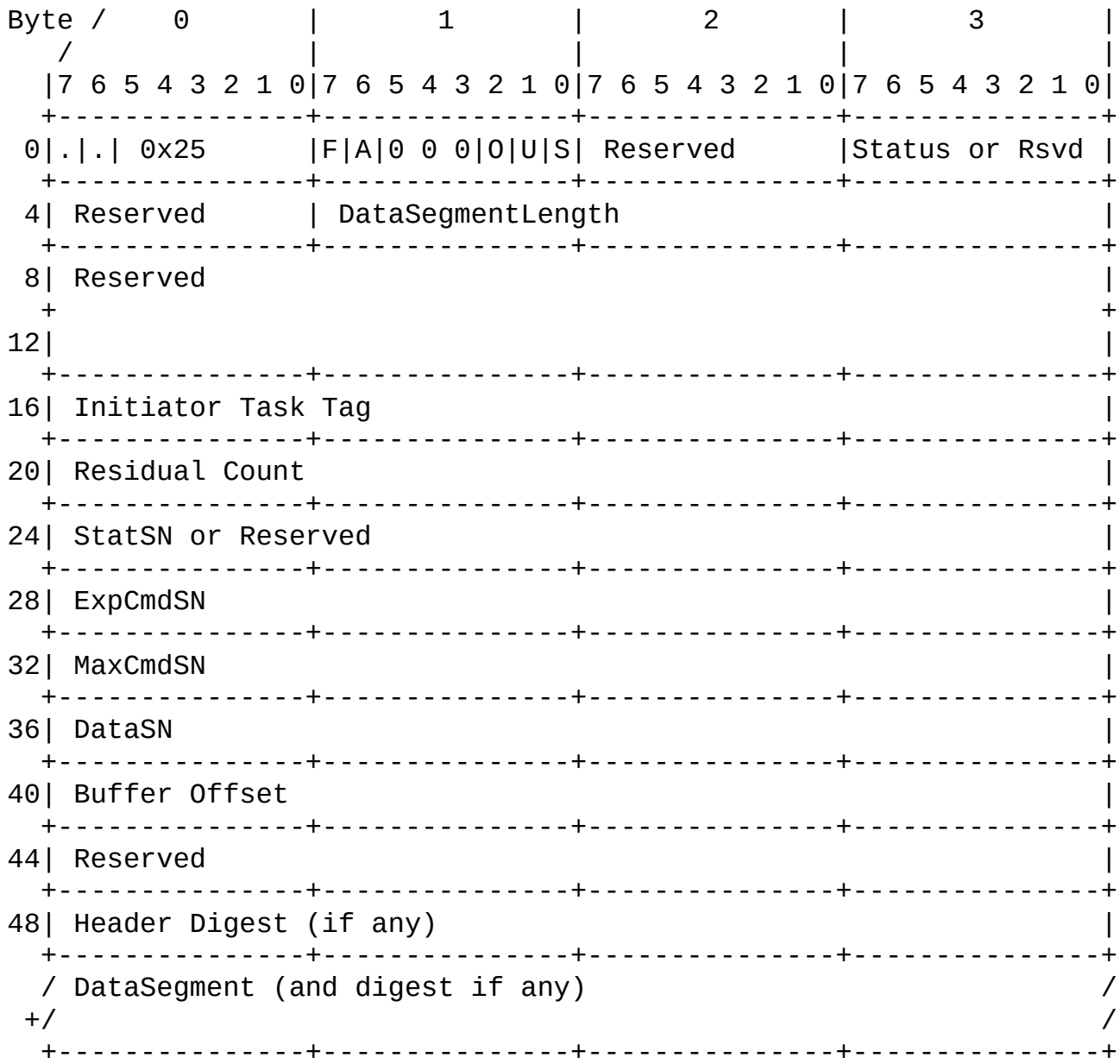
If the Request was ABORT TASK and the Response is "task not found", the Referenced Task Tag contains the Initiator Task Tag of the task that was to be aborted. In other cases, it MUST be set to 0xffffffff.

10.7 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.

10.7.1 F (Final) Bit

For outgoing data, this bit is 1 for the last PDU of unsolicited data or the last PDU of a sequence that answers an R2T.

For incoming data, this bit is 1 for the last input (read) data PDU of a sequence. Input can be split into several sequences, each having its own F bit. Splitting the data stream into sequences does not affect DataSN counting on Data-In PDUs. It MAY be used as a "change direction" indication for Bidirectional operations that need such a change.

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.

10.7.2 A (Acknowledge) bit

For sessions with ErrorRecoveryLevel 1 or higher, the target sets this bit to 1 to indicate that it requests a positive acknowledgement from the initiator for the data received. The target should use the A bit moderately; it MAY set the A bit to 1 only once every MaxBurstSize bytes and MUST NOT do so more frequently than this.

On receiving a Data-In PDU with the A bit set to 1, the initiator MUST issue a SNACK of type DataACK. If the initiator has detected holes in the input sequence, it MUST postpone issuing the SNACK of type DataACK until the holes are filled.

10.7.3 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.

The Target Transfer Tag values are not specified by this protocol except that the value 0xffffffff is reserved and means that the Target Transfer Tag is not supplied. If the Target Transfer Tag is provided, then the LUN field MUST hold a valid value and be consistent with whatever was specified with the command; otherwise, the LUN field is reserved.

10.7.4 StatSN

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

10.7.5 DataSN

For input (read) data PDUs, the DataSN is the data PDU number (starting with 0) within the data transfer for the command identified by the Initiator Task Tag.

For output (write) data PDUs, the DataSN is the data PDU number (starting with 0) 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).

Any input or output data sequence MUST contain less than 2**32 numbered PDUs.

10.7.6 Buffer Offset

The Buffer Offset field contains the offset of this PDU payload data within the complete data transfer. The sum of the buffer offset and length should not exceed the expected transfer length for the command.

The order of data PDUs within a sequence is determined by DataPDUInOrder. When set to yes, it means that PDUs have to be in increasing Buffer Offset order and overlays are forbidden.

The ordering between sequences is determined by DataSequenceInOrder. When set to yes, it means that sequences have to be in increasing Buffer Offset order and overlays are forbidden.

10.7.7 DataSegmentLength

This is the data payload length of a SCSI Data-In or SCSI Data-Out PDU. The sending of 0 length data segments should be avoided, but initiators and targets MUST be able to properly receive 0 length data segments.

The Data Segments of Data-in and Data-out PDUs SHOULD be filled to the integer number of 4 byte words (real payload) unless the F bit is set to 1.

10.7.8 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.

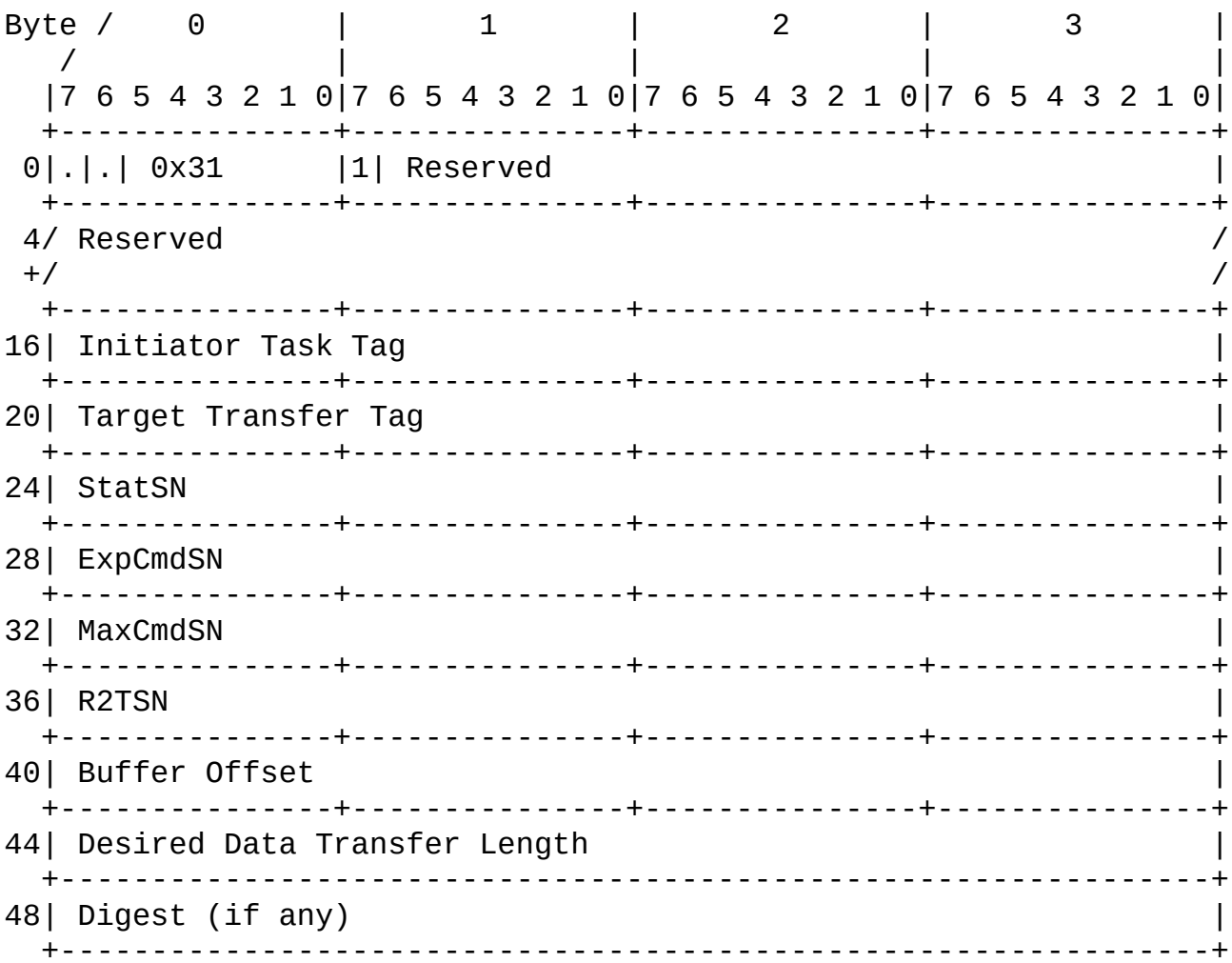
bit 3-5 - not used (should be set to 0).

bit 1-2 - used the same as in a SCSI Response.

bit 0 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 they values are as define in Section 10.4 SCSI Response.

10.8 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 agreed by sending the InitialR2T=no key-pair to each other, which occurs either during Login or through the Text request/Response mechanism.

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. The data length of the Data PDU MUST not exceed 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 SHOULD 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.

10.8.1 R2TSN

R2TSN is the R2T PDU number (starting with 0) within the command identified by the Initiator Task Tag.

The number of R2Ts in a command MUST be less than 0xffffffff.

10.8.2 StatSN

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

10.8.3 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

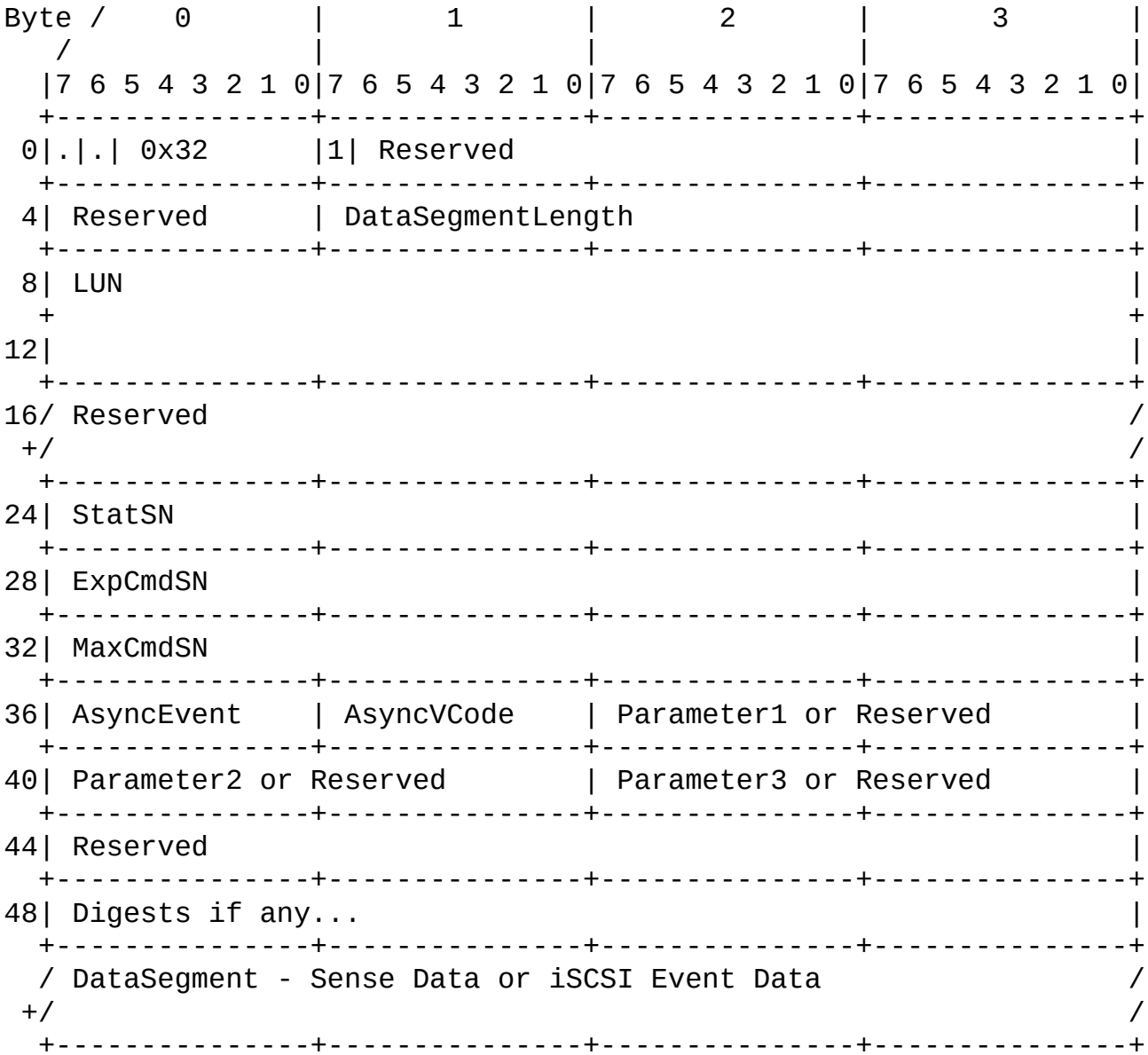
to the beginning of the total data transfer. The Desired Data Transfer Length SHOULD not be 0 and MUST not exceed MaxBurstSize.

10.8.4 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 is copied in the outgoing data PDUs and is used by the target only. There is no protocol rule about the Target Transfer Tag, but it is assumed that it is used to tag the response data to the target (alone or in combination with the LUN).

10.9 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.

10.9.1 AsyncEvent

The codes used for iSCSI Asynchronous Messages (Events) are:

0 - a SCSI Asynchronous Event is reported in the sense data. Sense Data that accompanies the report, in the data segment, identifies the condition. The sending of a SCSI Event (Asynchronous Event Notification in SCSI terminology) is controlled by a SCSI Control Mode Page bit.

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" (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.
The Parameter3 field (Time2Retain) indicates the maximum time to reconnect and/or restart commands after the initial wait (Time2Wait).
If the initiator does not attempt to reconnect and/or restart 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 (Time2Retain).
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.
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.
The Parameter3 field (Time2Retain) indicates the maximum time to reconnect and/or restart commands after the initial wait (Time2Wait).
If the initiator does not attempt to reconnect and/or restart 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.

255 - vendor specific iSCSI Event. The AsyncVCode details the vendor code, and data MAY accompany the report.

All other event codes are reserved.

10.9.2 AsyncVCode

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

10.9.3 Sense Data or iSCSI Event Data

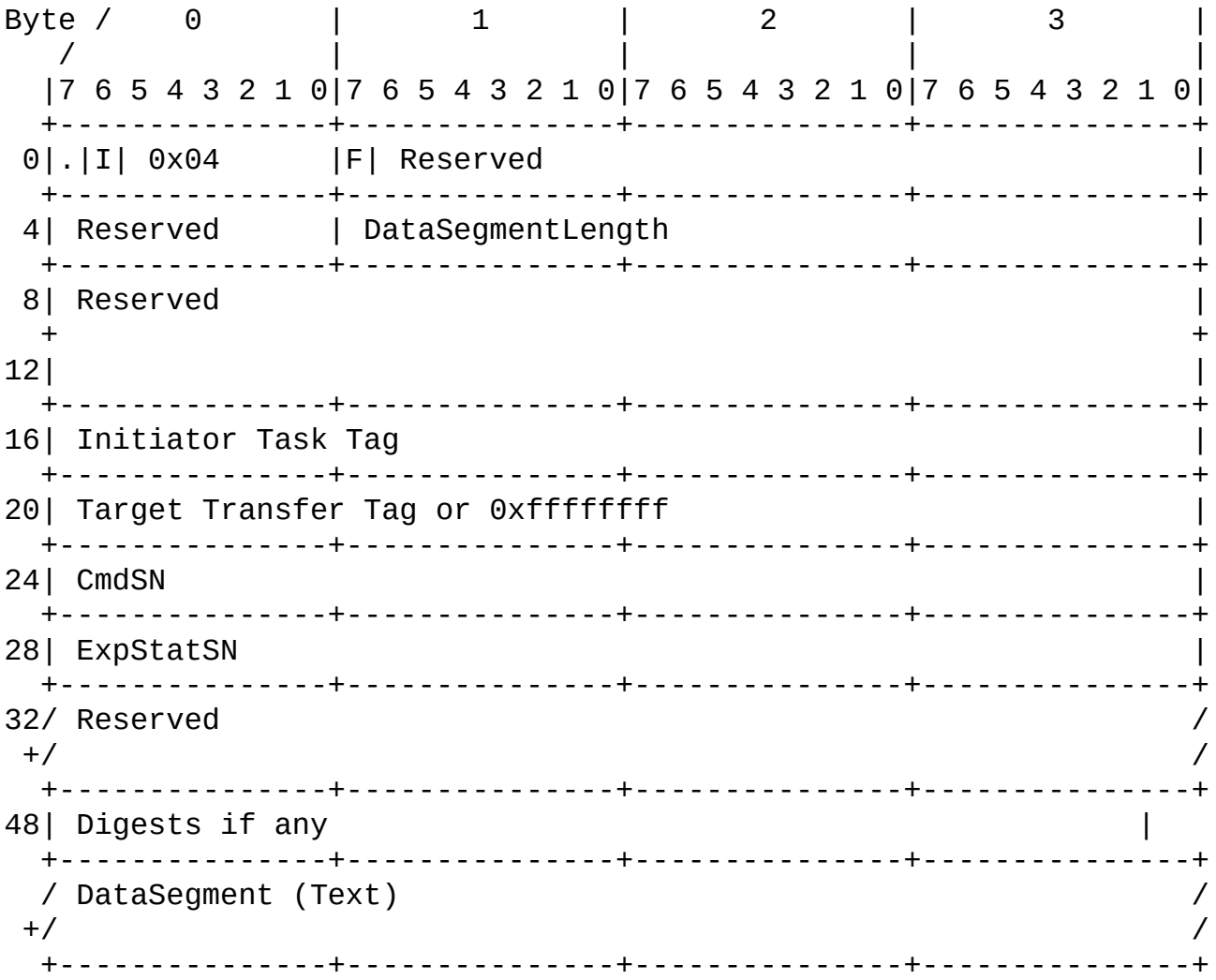
For a SCSI Event, this data accompanies the report in the data segment and identifies the condition.

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.

10.10 Text Request

The Text Request is provided to allow for the exchange of information and for future extensions. It permits the initiator to inform a target of its capabilities or to request some special operations.

An initiator MUST have only one outstanding Text Request on a connection at any given time.



10.10.1 F (Final) Bit

When set to 1, indicates that this is the last or only text request in a sequence of commands; otherwise, it indicates that more commands will follow.

10.10.2 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).

10.10.3 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 should reset any internal state associated with the Initiator Task Tag.

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.

A target MAY reset its internal state if an exchange is stalled by the initiator for a long time or if it is running out of resources.

Long text responses are handled as in the following example:

```
I->T Text SendTargets=all (F=1,TTT=0xffffffff)
T->I Text <part 1> (F=0,TTT=0x12345678)
I->T Text <empty> (F=1, TTT=0x12345678)
T->I Text <part 2> (F=0, TTT=0x12345678)
I->T Text <empty> (F=1, TTT=0x12345678)
...
T->I Text <part n> (F=1, TTT=0xffffffff)
```

10.10.4 Text

The initiator sends the target a set of key=value or key=list 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. Text keys and values MUST ONLY contain letters (a-z, A-Z), digits (0-9), space (0x20), point (.), minus (-), plus (+), and underscore (_). The key and value are separated by a '=' (0x3d) delimiter. Every key=value pair (including the last or only pair) MUST be followed by one null (0x00) delimiter. A list is a set of values separated by comma (0x2c). Text values may also contain colon (:) and brackets ([and]).

Character strings are represented as plain text. Binary items can be encoded using their decimal representation (with or without leading zeros) or hexadecimal representation (e.g., 8190 is 0x1ffe). Upper and lower case letters may be used interchangeably in hexadecimal notation (i.e., 0x1aBc, 0x1AbC, 0X1aBc, and 0x1ABC are equivalent). Binary items can also be encoded using the more compact Base64 encoding as specified by [RFC2045] preceded by the 0b. Key names MUST NOT exceed 63 bytes.

If not otherwise specified, the maximum length of an individual value (not its encoded representation) is 255 bytes not including the delimiter (comma or null).

The data lengths of a text request or response MUST NOT exceed MaxRecvPDULength (a per connection negotiated parameter).

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).

The target responds by sending its response back to the initiator. The response text format is similar to the request text format.

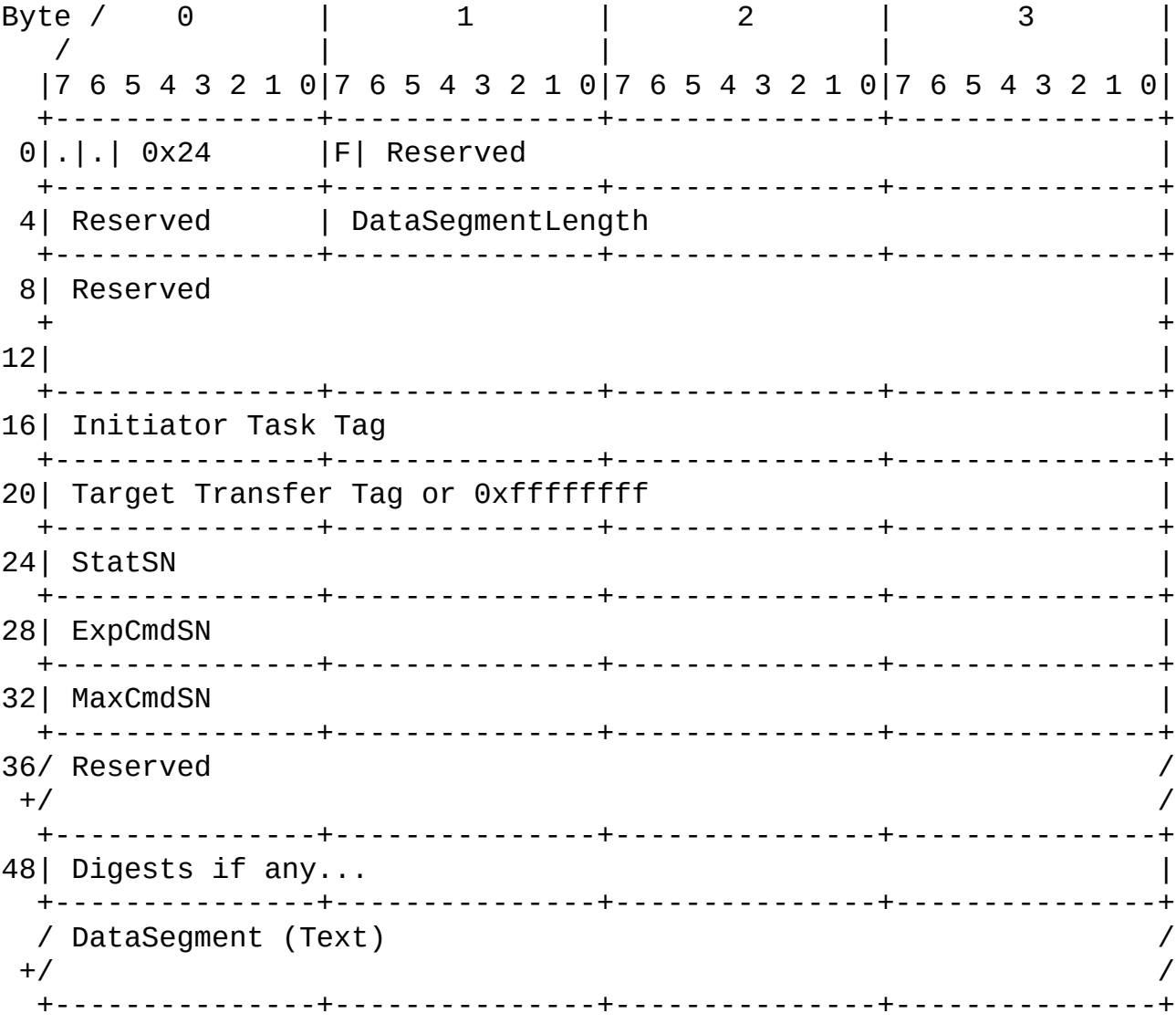
As text for text requests and responses can span several PDUs (e.g., if the PDU length does not allow the whole text to be contained in a single PDU), the text response MAY refer to key=value pairs presented in an earlier text request and the text in the request may refer to earlier responses.

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.

10.11 Text Response

The Text Response PDU contains the target's responses to the initiator's Text request. The format of the Text field matches that of the Text request.



10.11.1 F (Final) Bit

When set to 1, in response to a text request with the Final bit set to 1, the F bit indicates that the target has finished the whole operation. Otherwise, if set to 0 in response to a text request with the Final Bit set to 1, it indicates that the target has more work to do (invites a follow-on text request). A text response with the F bit

set to 1 in response to a text request with the F bit set to 0 is a protocol error.

A text response with the F bit set to 1 MUST NOT contain key=value pairs that may require additional answers from the initiator.

A text response with the F bit set to 1 MUST have a Target Transfer Tag field set to the reserved value of 0xffffffff.

A text response with the F bit set to 0 MUST have a Target Transfer Tag field set to a value other than the reserved 0xffffffff.

10.11.2 Initiator Task Tag

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

10.11.3 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.

The initiator MUST copy this Target Transfer Tag in its next request to indicate that it wants the rest of the data.

If the target receives a Text Request with the Target Transfer Tag set to the reserved value of 0xffffffff, it discards 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, state expressed through the Target Transfer Tag if the initiator fails to continue the exchange for some time. The target may reject subsequent Text requests with the Target Transfer Tag set to the "stale" value.

10.11.4 Text Response Data

The Text Response Data Segment contains responses in the same key=value format as the Text request and with the same length and cod-

ing constraints. Chapter 11 and Chapter 12 list some basic Text key=value pairs, some of which can be used in Login Request/Response and some in Text Request/Response.

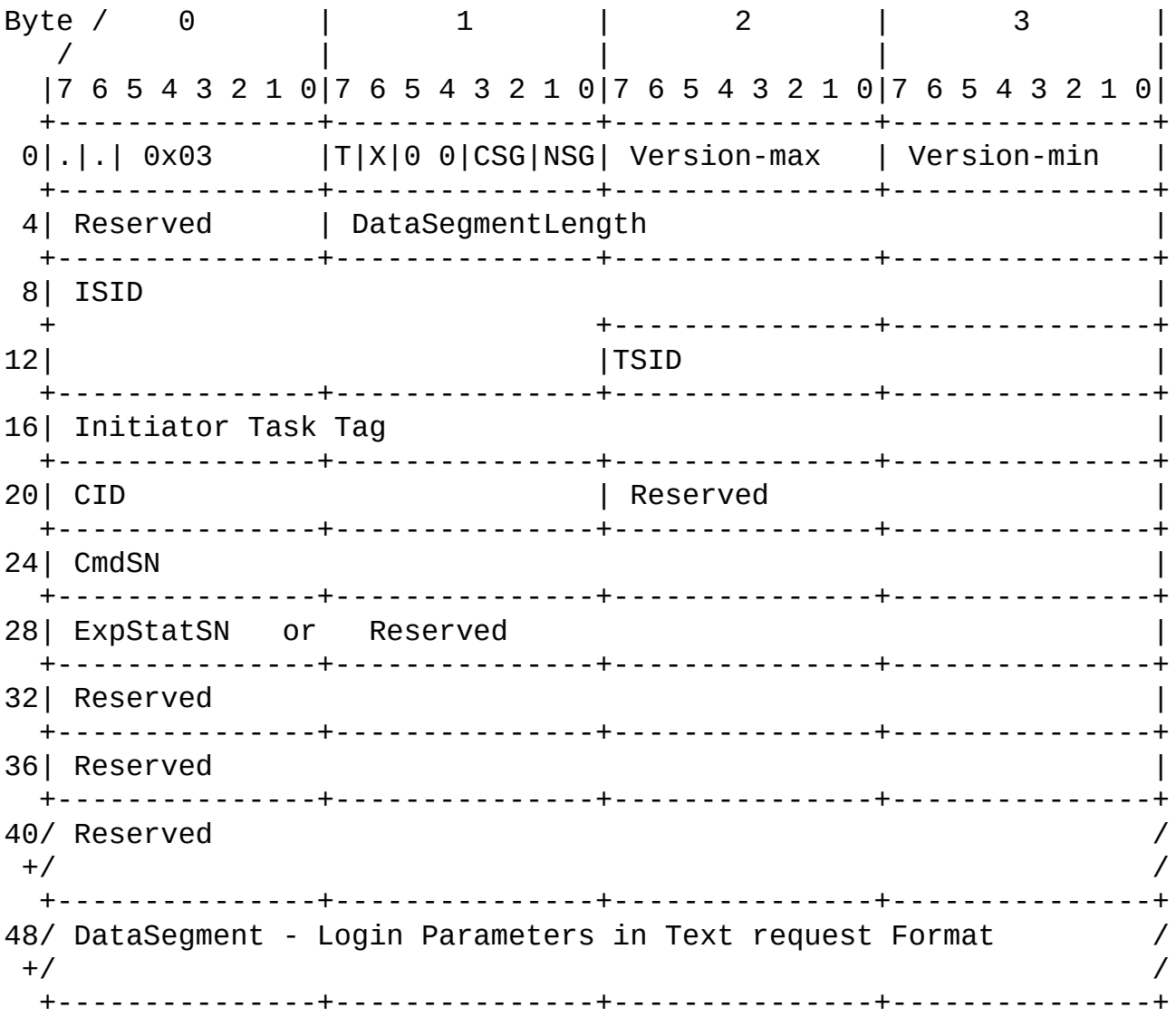
As text for text requests and responses can span several PDUs (e.g., if the PDU length does not allow the whole text to be contained in a single PDU) the text response MAY refer to key=value pairs presented in an earlier text request.

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

10.12 Login Request

After establishing a TCP connection between an initiator and a target, the initiator MUST start a Login phase to gain further access to the target's resources.

The Login Phase (see Chapter 4) consists of a sequence of Login requests and responses that carry the same Initiator Task Tag.



10.12.1 T (Transit) Bit

If set to 1, indicates that the initiator is ready to transit to the next stage.

If the T bit is set to 1 and NSG is FullFeaturePhase, then this also indicates that the initiator is ready for the Final Login Response (see Chapter 4).

If the response class is 0 the target MAY answer with a Login response with the T bit set to 1 ONLY if the T bit is set to 1 in the request.

If the response class is not 0 the T bit value MUST be ignored by the initiator.

10.12.2 X - Restart Connection

If this bit is set to 1, then this command is an attempt to reinstate a failed connection or a failed session.

The TSID MUST be non-zero if the X bit is 1. CID does not change and this command performs first 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 TSID is 0 then the X bit MUST be 0.

The X bit MAY be set to 1 ONLY on the first request of the Login phase.

If the operational ErrorRecoveryLevel is 2, connection reinstatement is a complete connection recovery, which enables future task reassignment. If the operational ErrorRecoveryLevel is less than 2, connection reinstatement refers to the replacement of the old CID without enabling task reassignment.

10.12.3 CSG and NSG

Through these fields, Current Stage (CSG) and Next Stage (NSG), the Login negotiation commands 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

10.12.4 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.

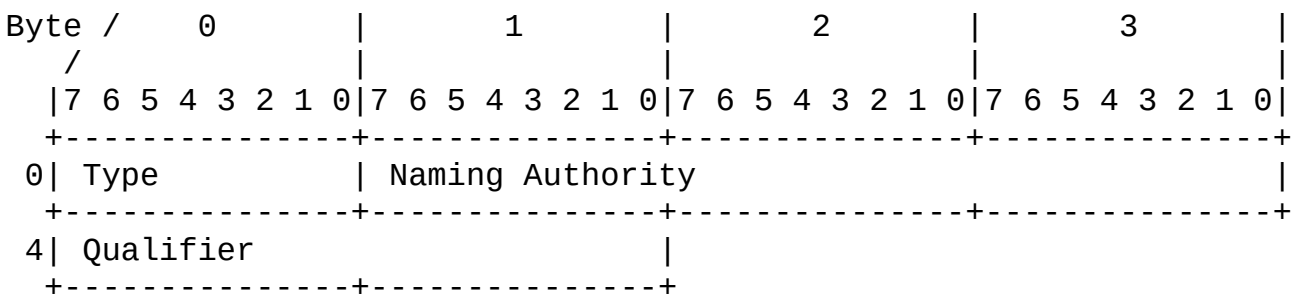
10.12.5 Version-min

Minimum Version supported. The version number of the current draft is 0x3.

All Login requests within the Login phase MUST carry the same Version-min. The target MUST use the value presented with the first login request.

10.12.6 ISID

This is an initiator-defined component of the session identifier (SSID). The ISID is structured as follows. See [NDT] and Section 9.1.1 Conservative Reuse of ISIDs for details.



The Type field identifies the format of the Naming Authority field and takes on three defined values with all other possible values reserved as indicated bellow:

Type	naming authority format
0x00	IEEE OUI
0x01	IANA Enterprise Number (EN)
0x02	"Random"
0x03-0xFF	Reserved

The Naming Authority field identifies the vendor or organization whose component (SW or HW) generates this ISID. A vendor or organization with one or more OUIs, and/or one or more Enterprise Numbers, MUST use at least one of these numbers and select the appropriate value for the Type field when its components generate ISIDs. An OUI or EN MUST be set in the Naming Authority field in network byte order (BigEndian).

If the Type field is 02h, the Naming Authority field SHOULD be set to a random or pseudo-random 24bit unsigned integer value in network byte order (BigEndian). See [NDT] for how this affects the principle of "conservative reuse".

The Qualifier field is a 16 bit unsigned integer value that provides a range of possible values for the ISID within the Type and Naming Authority 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 9.1.1 Conservative Reuse of ISIDs).

10.12.7 TSID

The TSID is the target assigned component of the session identifier (SSID). Together with the ISID, provided by the initiator, TSID uniquely identifies the session from that specific target with that initiator.

On a Login request, a TSID value of 0 indicates a request to open a new session.

A non-zero TSID indicates a request to add a connection to an existing session.

10.12.8 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.

10.12.9 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 (e.g., if the leading login carries the CmdSN 123 all other Login requests carry the CmdSN 123 and the first non-immediate command also carries the CmdSN 123).

The target MUST use the value presented with the first login request.

10.12.10 ExpStatSN

This is ExpStatSN for the old connection.

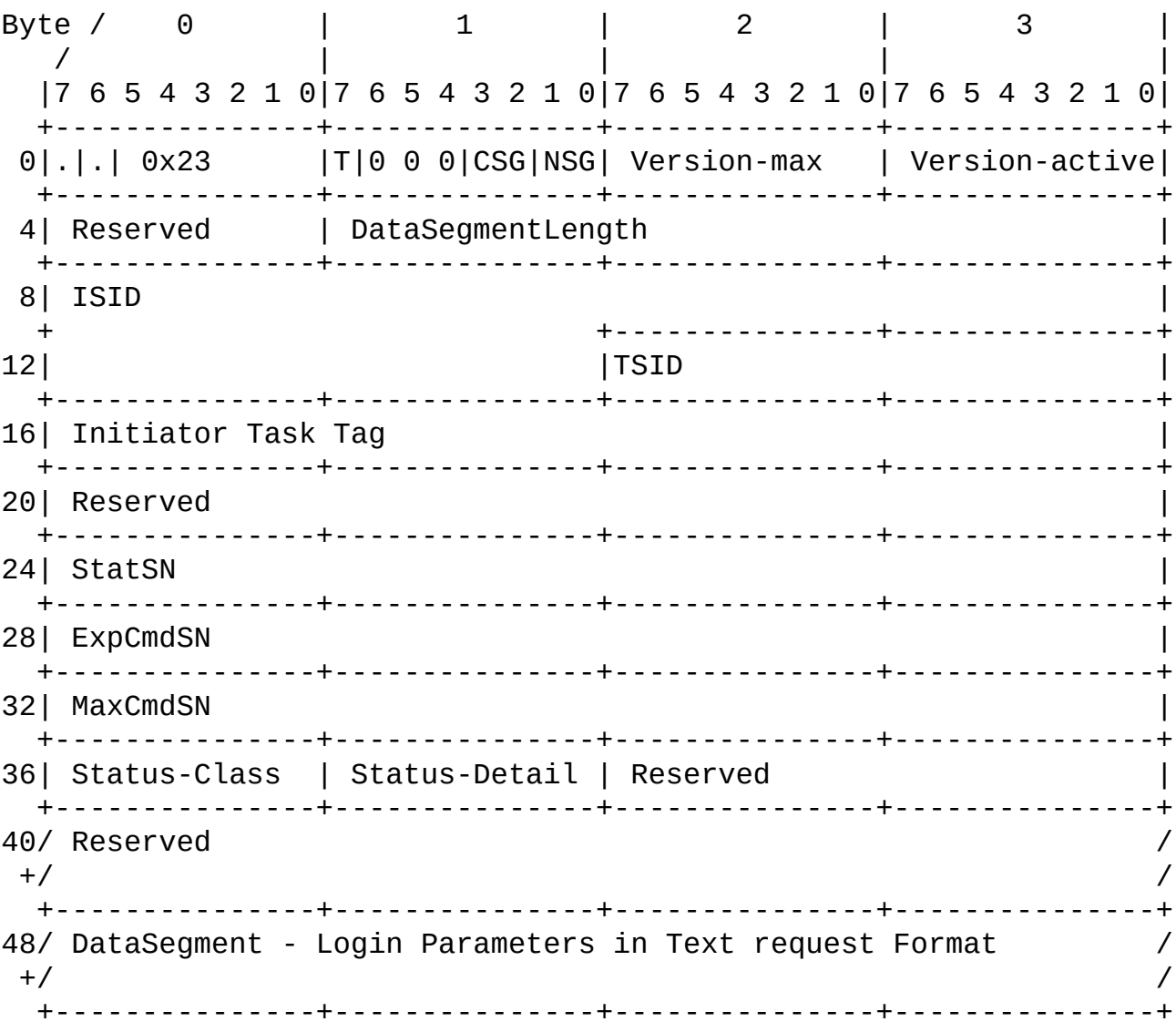
This field is valid only if the Login request restarts a connection (i.e., X bit is 1 and TSID is not zero).

10.12.11 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 10.10.4 Text for text requests/ responses also hold for login requests/responses. Keys and their explanations are listed in Chapter 11 (security negotiation keys) and Chapter 12 (operational parameter negotiation keys). All keys in Chapter 12, except for the X- extension format, MUST be supported by iSCSI initiators and targets. Keys in Chapter 12, only need to be supported when the function to which they refer is mandatory to implement.

10.13 Login Response

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



10.13.1 Version-max

This is the highest version number supported by the target.

All Login responses within the Login phase MUST carry the same Version-max.

The initiator MUST use the value presented as a response to the first login request.

10.13.2 Version-active

Indicates the highest version supported by the target and initiator. If the target does not support a version within the range specified by the initiator, the target rejects the login and this field indicates the lowest version supported by the target.

All Login responses within the Login phase MUST carry the same Version-active.

The initiator MUST use the value presented as a response to the first login request.

10.13.3 TSID

The TSID is the target assigned component of the session identifier (SSID). TSID and the ISID provided by the initiator uniquely identify the session with that initiator. TSID MUST be valid only in the final response.

10.13.4 StatSN

For the first Login Response (the response to the first Login Request), this is the starting status Sequence Number for the connection. The next response of any kind, including the next login response, if any, in the same login phase, will carry this number + 1. This field is valid only if the Status Class is 0.

10.13.5 Status-Class and Status-Detail

The Status returned in a Login Response indicates the execution status of the login phase. The status includes:

Status-Class
Status-Detail

0 Status-Class indicates success.

A non-zero Status-Class indicates an exception. In this case, Status-Class is sufficient for a simple initiator to use when handling errors, without having to look at the Status-Detail. The Status-

Detail allows finer-grained error recovery for more sophisticated initiators, as well as better information for error 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 client 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	No more connections are accepted on this SID.
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.
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.

10.13.6 T (Transit) bit

The T bit is set to 1 as an indicator of the end of the stage. If the T bit is set to 1 and NSG is FullFeaturePhase, then this is also the Final Login Response (see Chapter 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.

If the status class is not 0 the T bit value MUST be set to 1 by the target and ignored by the initiator.

10.14 Logout Request

The Logout request is used to perform a controlled closing of a connection.

An initiator MAY use a logout command 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 commands on the closing connection. If the Logout is intended to close the session, new iSCSI commands MUST NOT be sent on any of the connections participating in the session.

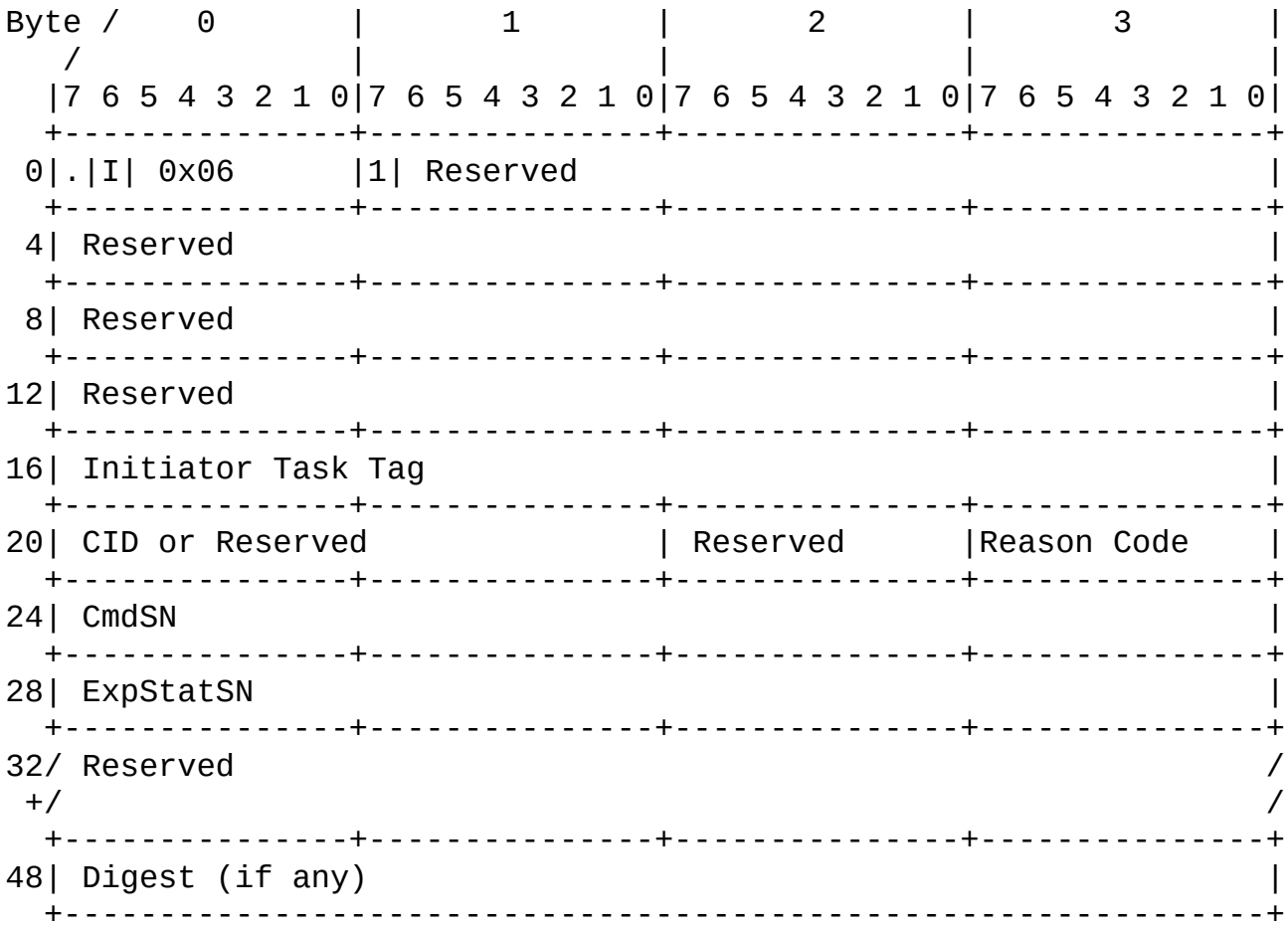
When receiving a Logout request with the reason code of "close the connection" or "close the session", the target MUST abort all pending commands, whether acknowledged or not, on that connection or session respectively. When receiving a Logout request with the reason code "remove connection for recovery", 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 aborted 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 aborted 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 command to "clean-up" the target end of a failing connection and enable recovery to start, or use the restart option of the Login command for the same effect. In sessions with a single connection, this may imply the opening of a second connection with the sole purpose of cleaning-up the first. In this case, the restart option of the Login should be used.

Sending a logout request with the reason code of "close the connection" or "remove the connection for recovery" may result in the discarding of some unacknowledged commands. Those holes in command sequence numbers will have to be handled by appropriate recovery (see Chapter 7) unless the session is also closed.



10.14.1 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".

10.14.2 ExpStatSN

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

10.14.3 Reason Code

Indicates the reason for Logout:

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

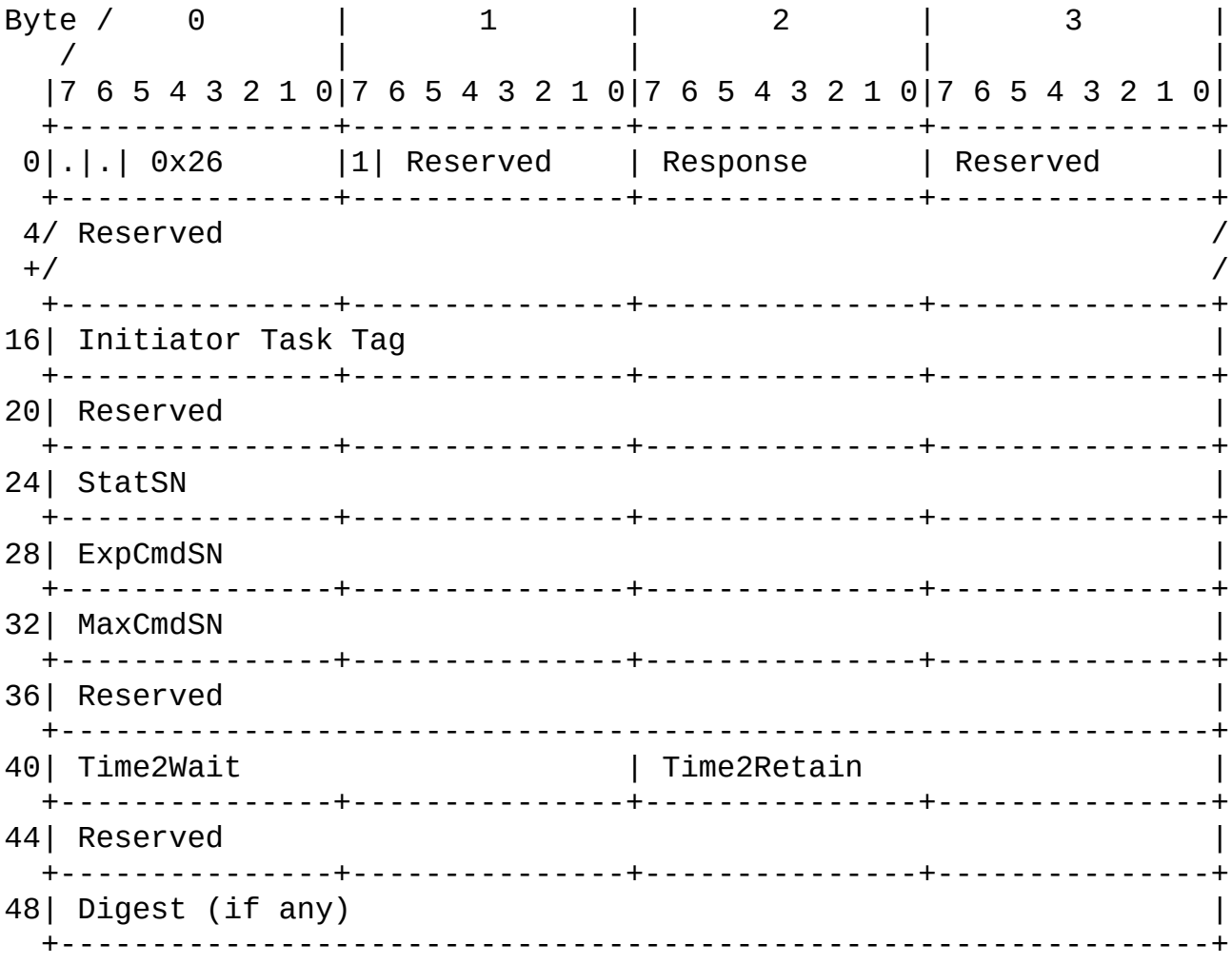
1 - closes the connection. All commands associated with connection (if any) are aborted.

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.

10.15 Logout Response

The logout response is used by the target to indicate if the cleanup operation for the connection(s) has completed.

After Logout, the TCP connection referred by the CID MUST be closed at both ends (or all connections must be closed if the logout reason was session close).



10.15.1 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.

10.15.2 Time2Wait

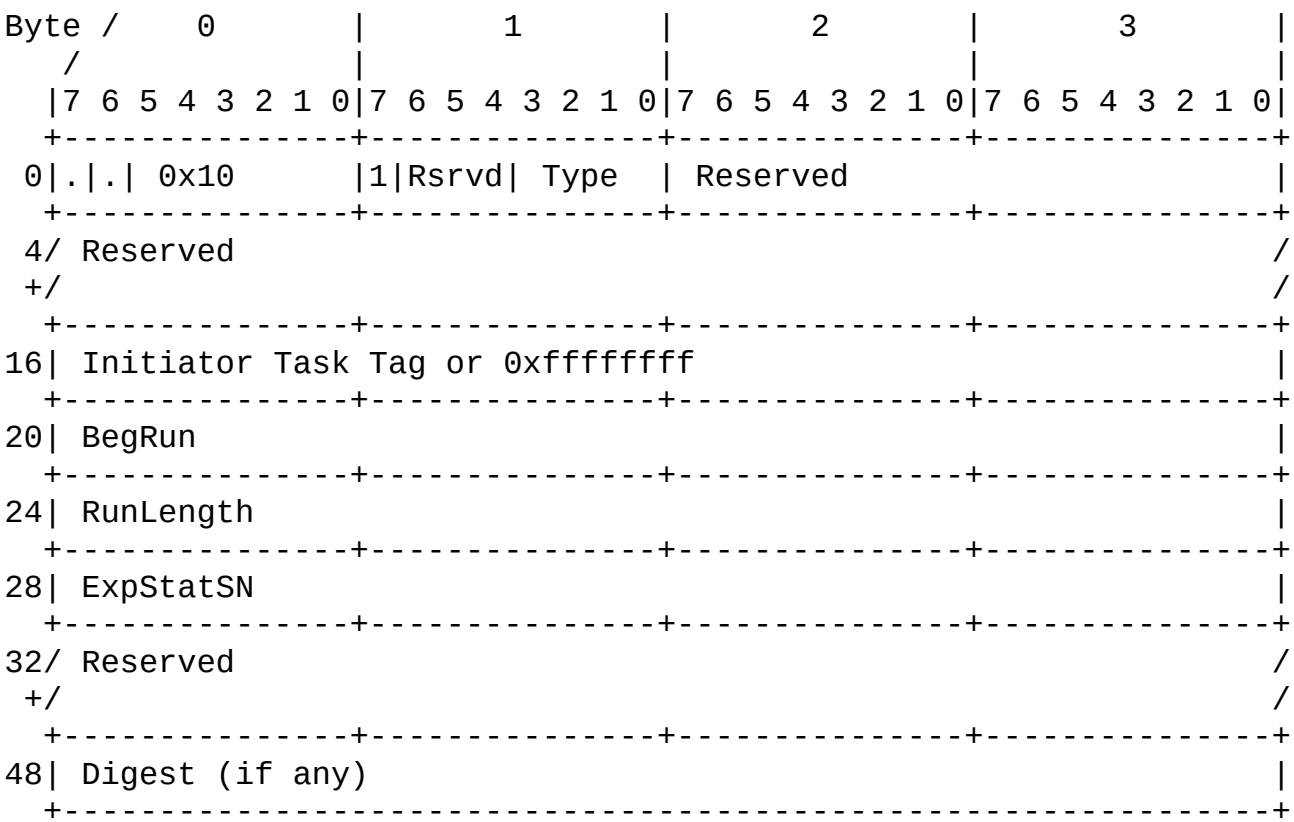
The minimum amount of time, in seconds, to wait before Login to add or reinstate a new connection to this session on this target. If Time2Wait is 0 a new Login may be attempted immediately.

10.15.3 Time2Retain

If ErrorRecoveryLevel is less than 2, the maximum time, in seconds, that the target waits for a connection reinstatement Login, after the initial wait (Time2Wait), after which the connection state is discarded. If it is the last connection of a session, the whole session state is discarded after Time2Retain. If ErrorRecoveryLevel is 2, this is the maximum 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 Time2Retain is 0 the connection (or session state) is discarded by the target.

10.16 SNACK Request



Support for SNACK is optional.

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 and MUST include all its flags. However, the fields ExpCmdSN, MaxCmdSN and ExpDataSN MUST carry the current values.

The numbered Data-In PDUs, requested by a SNACK with a RunLength different from 0, have to be delivered as exact replicas of the ones the initiator missed and MUST include all its flags. However, the fields ExpCmdSN and MaxCmdSN MUST carry the current values. 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 MaxRecvPDULength.

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".

10.16.1 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 a Data/R2T SNACK, the Initiator Task Tag MUST be set to the Initiator Task Tag of the referenced Command. Otherwise, it is reserved.

An iSCSI target that does not support recovery within connection MAY discard the status SNACK. If the target supports recovery within connection, it MAY discard 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 MaxBurstSize limit is reached).

The DataACK is used to free resources at the target and not to request or imply data retransmission.

10.16.2 BegRun

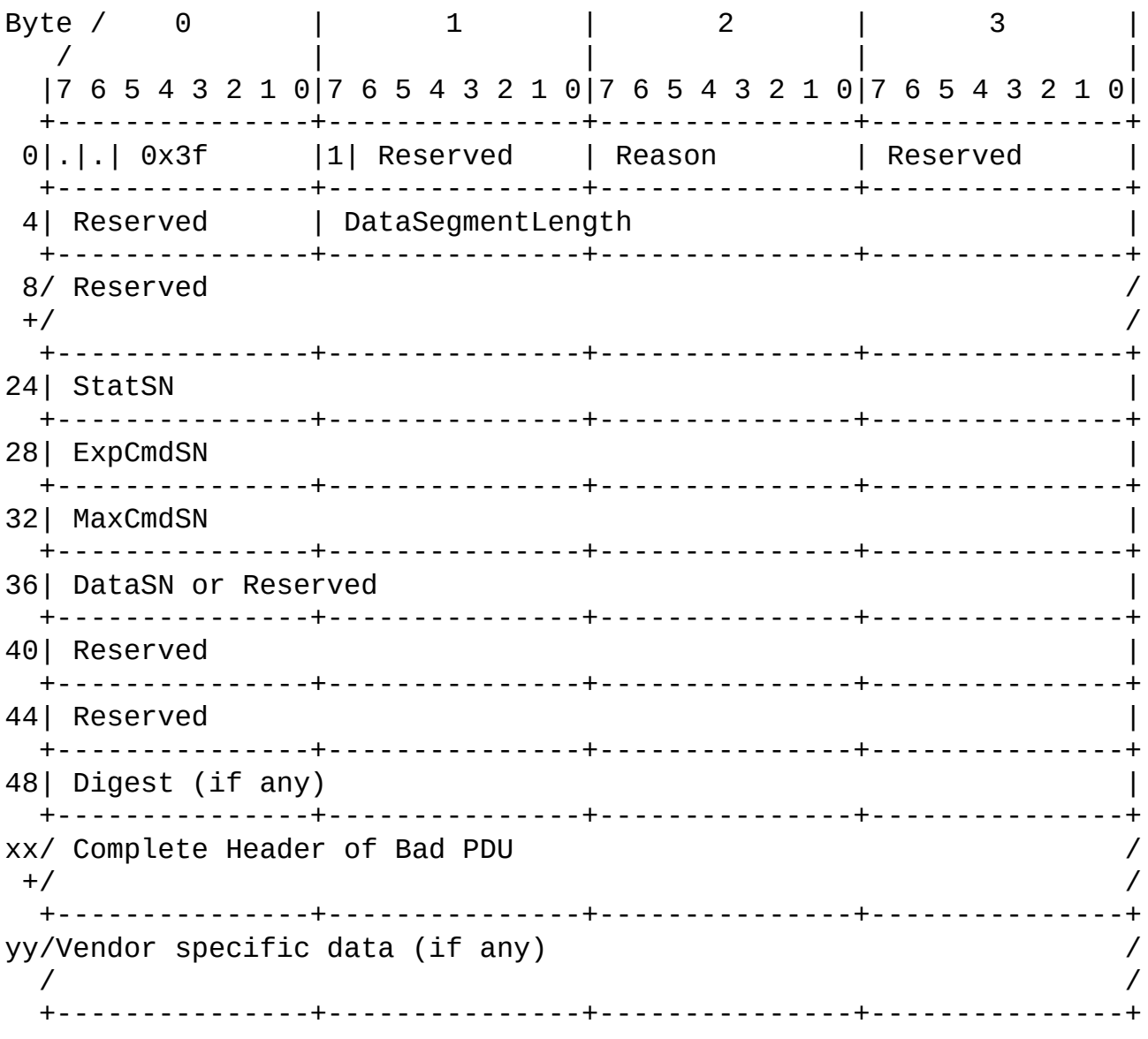
The first missed DataSN, R2TSN, or StatSN or the next expected DataSN for a DataACK type SNACK request.

10.16.3 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 to BegRun have to be resent.

The first data SNACK, issued after initiator’s MaxRecvPDULength decreased, for a command issued on the same connection before the change in MaxRecvPDULength, 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 MaxRecvPDULength was changed at the target.

10.17 Reject



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

10.17.1 Reason

The reject Reason is coded as follows:

Code (hex)	Explanation	Can the original PDU be re-sent?
0x01	Full Feature Phase Command before login	no
0x02	Data (payload) Digest Error	yes (Note 1)
0x03	Data-SNACK Reject	yes
0x04	Protocol Error (e.g., SNACK request for a status that was already acknowledged	no
0x05	Command not supported in this session type	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 or a data transfer. Some examples are invalid TTT/ITT, buffer offset, LUN qualifying a TTT.

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 10.4.3 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), the target MUST wait until it receives the Data-out PDU with the F bit set to 1 before sending the Response PDU.

For additional usage semantics of Reject PDU, see Section 7.2 Usage Of Reject PDU in Recovery.

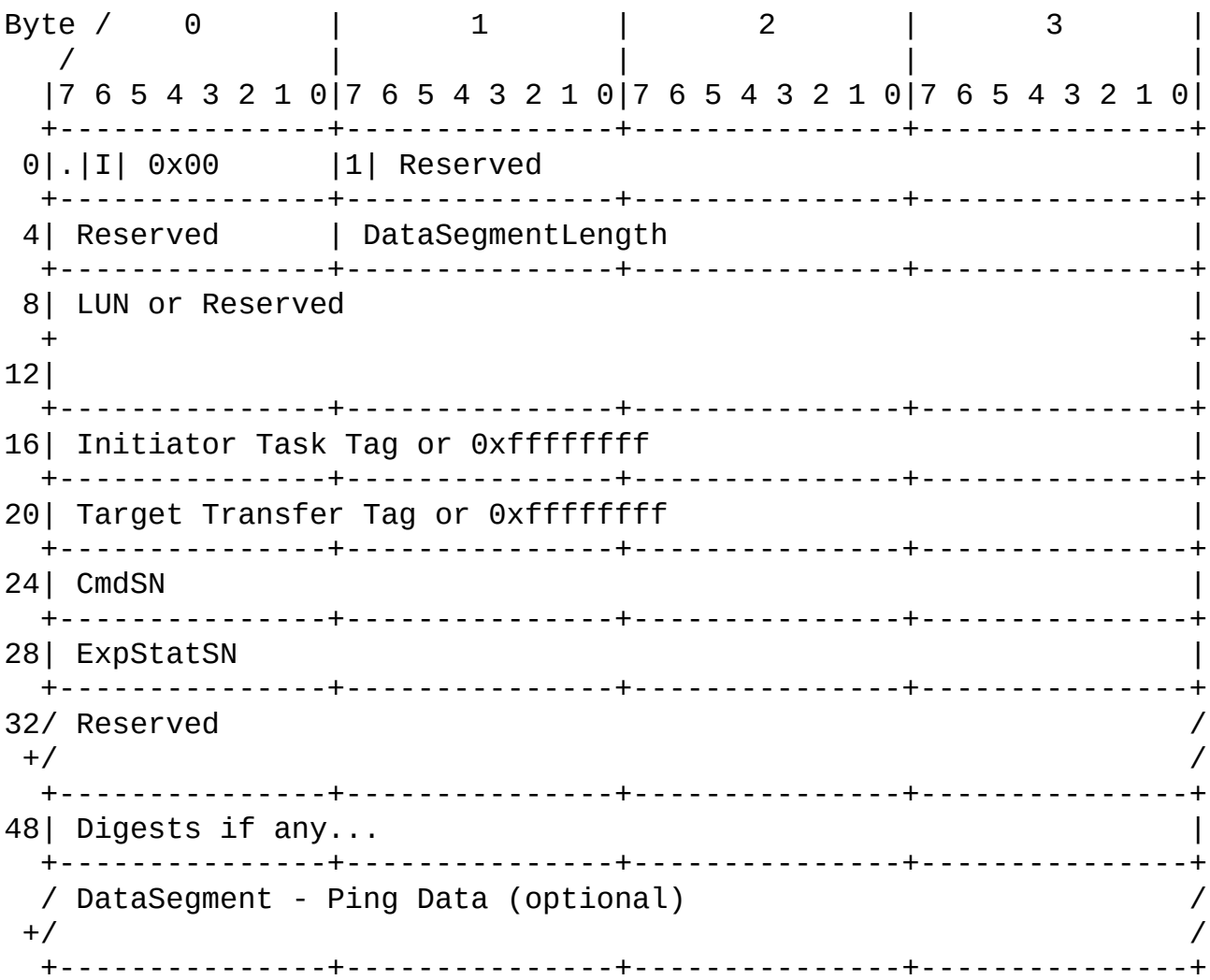
10.17.2 DataSN

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.

10.17.3 Complete Header of Bad PDU

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

10.18 NOP-Out



A NOP-Out may be used by an initiator as a "ping command" 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 command, 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).

10.18.1 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.

10.18.2 Target Transfer Tag

A target assigned identifier for the operation.

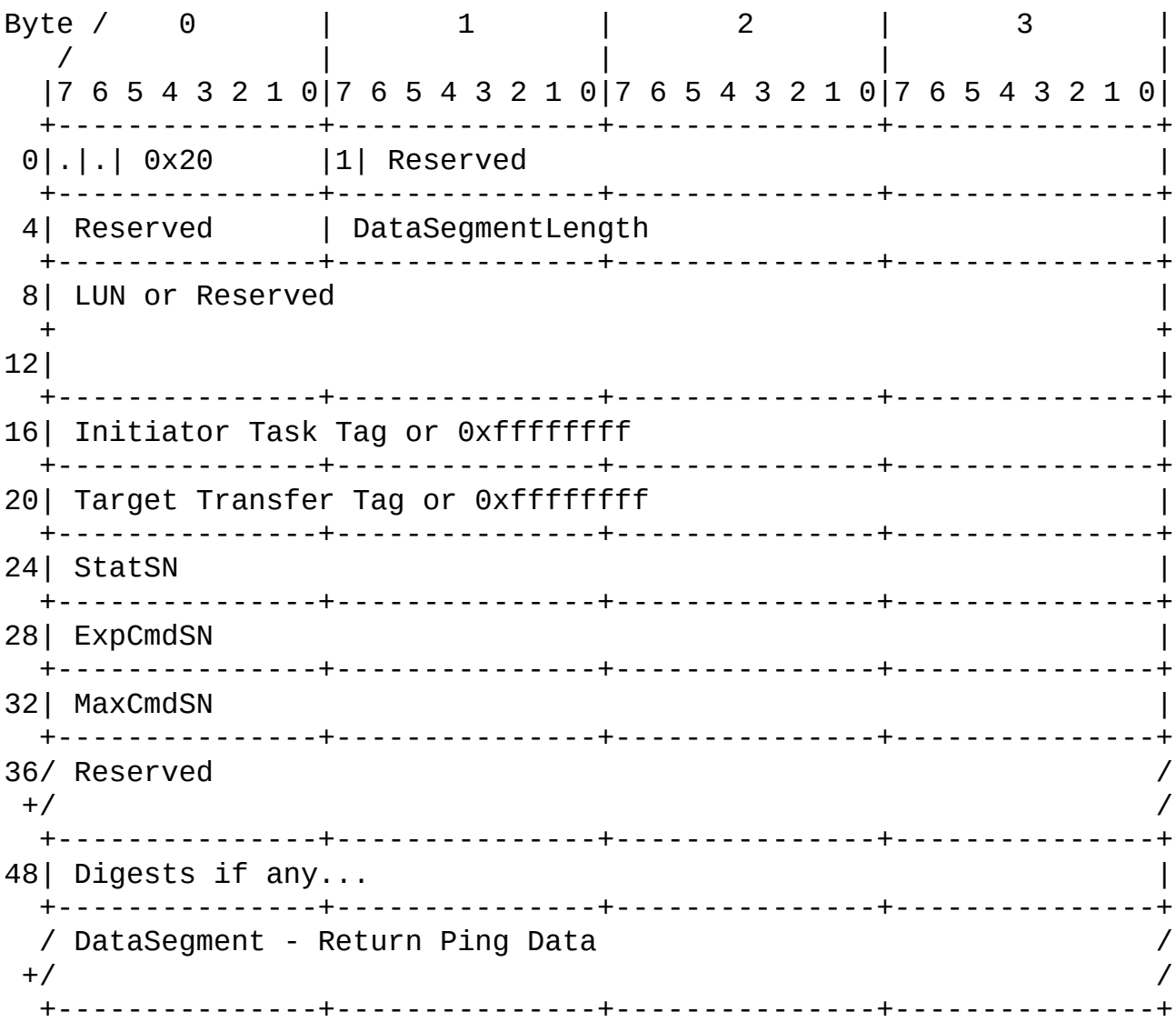
The NOP-Out MUST have the Target Transfer Tag set only if it is issued in response to a NOP-In with a valid Target Transfer Tag. In this case, it copies the Target Transfer Tag from the NOP-In PDU.

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

10.18.3 Ping Data

Ping data is reflected in the NOP-In Response. The length of the reflected data is limited to MaxRecvPDULength. The length of ping data is indicated by the Data Segment Length. 0 is a valid value for the Data Segment Length and indicates the absence of ping data.

10.19 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 Command. It MUST also duplicate up to the first MaxRecvPDULength bytes of

the initiator provided Ping Data. For such a response, the Target Transfer Tag MUST be 0xffffffff.

When a target send a NOP-In as a "ping" (the Initiator Task Tag is 0xffffffff) it MUST NOT send any data in the data segment (DataSegmentLength MUST be 0).

10.19.1 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.

Whenever the NOP-In is sent as a "ping" to an initiator (not as a response to a NOP-Out), the StatSN field will contain the next StatSN. However, StatSN for this connection is not advanced.

10.19.2 LUN

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

11. iSCSI Security Keys and Values

The following keys can only be used during the SecurityNegotiatian stage of the Login Phase.

All security keys have connection-wide applicability.

11.1 AuthMethod

Senders: Initiator and Target

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
SPKM1	Simple Public-Key GSS-API Mechanism
SPKM2	Simple Public-Key GSS-API Mechanism
SRP	Secure Remote Password
CHAP	Challenge Handshake Authentication Protocol
none	No authentication

KRB5 is defined in [RFC1510].
SPKM1 and SPKM2 are defined in [RFC2025].
SRP is defined in [RFC2945] and CHAP is defined in [RFC1994].

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 SRP.

11.2 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 answer 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 items and their binary length (not the length of the character string that represents them in encoded form) MUST not exceed 65536 bytes.

11.3 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(by[RFC2025], 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 items and their binary length (not the length of the character string that represents them in encoded form) MUST not exceed 65536 bytes.

11.4 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 items, and their binary length (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 8.2 In-band Initiator-Target Authentication.

11.5 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 items 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.

12. Login/Text Operational Keys

The ISID and TSID collectively form the SSID (session id). A TSID of zero indicates a leading connection. 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 "declarative" may appear also in the SecurityNegotiation stage while all other keys described in this chapter are operational keys.

Key scope is indicated as session-wide (SW) or connection-only (CO).

12.1 HeaderDigest and DataDigest

Use: IO
Senders: Initiator and Target
Scope: CO

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

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, for example 0x3b stands for 0011 1011. The polynomial 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 should be calculated as follows:

- Data are assumed to be in the numbering order that appears in the draft and starts with byte 0 bit 0 to 7 continues with byte 1 bit 0 etc. (Big Endian on bytes / Little Endian on bits).
- The first 32 bits of the message are complemented.
- The n PDU bits are considered coefficients of a polynomial M(x) of order n-1, with bit 0 of byte 0 being x^(n-1).
- 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.
- After the last bit of the original segment, the CRC bits are transmitted with x^31 first followed by x^30 etc. (when examples are provided, the value to be specified in the examples follows the same rules of representation as the rest of this document).
- A receiver of a "good" segment (data or header) including the CRC built using the generator 0x11edc6f41 will get the value 0x1c2d19ed as its CRC (this is a polynomial value and not a word as outlined in this draft).

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.

12.2 MaxConnections

Use: LO
Senders: Initiator and Target
Scope: SW

MaxConnections=<number-from-1-to-65535>

Default is 1.

Initiator and target negotiate the maximum number of connections requested/acceptable. The lower of the two numbers is selected.

12.3 SendTargets

Use: FFPO
Senders: Initiator
Scope: SW

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

12.4 TargetName

Use: IO by initiator ALL by target, Declarative
Senders: Initiator and Target
Scope: SW

TargetName=<iSCSI-Name>

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).

12.5 InitiatorName

Use: IO, Declarative
Senders: Initiator
Scope: SW

InitiatorName=<iSCSI-Name>

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.

12.6 TargetAlias

Use: ALL, Declarative
Senders: Target
Scope: SW

TargetAlias=<UTF-8 string>

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.

12.7 InitiatorAlias

Use: ALL, Declarative
Senders: Initiator
Scope: SW

InitiatorAlias=<UTF-8 string>

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.

12.8 TargetAddress

Use: ALL, Declarative
Senders: Target
Scope: SW

TargetAddress=domainname[:port][,portal-group-tag]

If the TCP port is not specified, it is assumed to be the IANA-assigned default port for iSCSI.

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
```

The TargetAddress key is further described in Appendix E. - SendTargets Operation -.

12.9 InitialR2T

Use: LO
Senders: Initiator and Target
Scope: SW

InitialR2T=<yes|no>

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=0 and Desired Data Transfer Length=min (FirstBurstSize, Expected Data Transfer Length). The default action is that R2T is required, unless both the initiator and the target send this key-pair attribute specifying InitialR2T=no. Only the first outgoing data burst (immediate data and/or separate PDUs) can be sent unsolicited (i.e., not requiring an explicit R2T).

12.10 BidiInitialR2T

Use: LO
Senders: Initiator and Target
Scope: SW

BidiInitialR2T=<yes|no>

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. Once BidiInitialR2T has been set to 'no', it cannot be set back to 'yes'. Only the first outgoing data burst (immediate data and/or separate PDUs) can be sent unsolicited by an R2T.

12.11 ImmediateData

Use: LO
Senders: Initiator and Target
Scope: SW

ImmediateData=<yes|no>

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 them 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	Result (up to FirstBurstSize)
no	no	Unsolicited data in data PDUs only.
no	yes	Immediate & separate unsolicited data.
yes	no	Unsolicited data disallowed.
yes	yes	Immediate unsolicited data only.

12.12 MaxRecvPDULength

Use: ALL
Senders: Initiator and Target
Scope: CO

MaxRecvPDULength=<number-512-to-(2**24-1)>

Default is 8191 bytes.

This is a connection specific parameter.
The initiator or target declares the maximum data segment length in bytes they can receive in an iSCSI PDU.

For a target the value limiting the size of the receive PDUs is the lower of the declared MaxRecvPDULength and the negotiated MaxBurst-Size for solicited data or FirstBurstSize for unsolicited data.

12.13 MaxBurstSize

Use: LO
Senders: Initiator and Target
Scope: SW

MaxBurstSize=<number-512-to-(2**24-1)>

Default is 262144 (256 Kbytes).

The initiator and target negotiate maximum SCSI data payload in bytes in an 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.

The minimum of the two numbers is selected.

12.14 FirstBurstSize

Use: LO
Senders: Initiator and Target
Scope: SW

FirstBurstSize=<number-512-to-(2**24-1)>

Default is 65536 (64 Kbytes).

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.

The minimum of the two numbers is selected.

FirstBurstSize MUST NOT exceed MaximumBurstSize.

12.15 LogoutLoginMaxTime

Use: LO
Senders: Initiator and Target
Scope: SW

LogoutLoginMaxTime=<number-0-to-3600>

Default is 3.

The initiator and target negotiate the maximum time, in seconds after an initial wait (Time2Wait), before which the connection reinstatement is still possible after a 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.

The lesser of the two values is selected and will be used anywhere a explicit value is not otherwise provided (Time2Retain).

A value of 0 indicates that connection state is immediately discarded by the target.

12.16 LogoutLoginMinTime

Use: LO
Senders: Initiator and Target
Scope: SW

LogoutLoginMinTime=<number-0-to-3600>

Default is 3.

The initiator and target negotiate the minimum time, in seconds, to wait before attempting connection reinstatement after a connection termination or a connection reset.

The maximum of the two values is selected and will be used anywhere an explicit value is not otherwise provided(Time2Wait).

A value of 0 indicates that connection reinstatement can be attempted immediately.

12.17 MaxOutstandingR2T

Use: LO
Senders: Initiator and Target
Scope: SW

MaxOutstandingR2T=<number-from-1-to-65535>

Default is 1.

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 7.11.1) is encountered for that data sequence.

12.18 DataPDUInOrder

Use: LO
Senders: Initiator and Target
Scope: SW

DataPDUInOrder=<yes|no>

Default is yes.
Result function is OR.

No is used by iSCSI to indicate that the data PDUs within sequences can be in any order. Yes is used to indicate that data PDUs within sequences have to be at continuously increasing addresses and overlays are forbidden.

12.19 DataSequenceInOrder

Use: LO
Senders: Initiator and Target
Scope: SW

DataSequenceInOrder=<yes|no>

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 ErrorRecoveryLevel is not 0 and 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. MaxOustandingR2T MUST be set to 1 in this case.

12.20 ErrorRecoveryLevel

Use: LO
Senders: Initiator and Target
Scope: SW

ErrorRecoveryLevel=<0 to 2>

Default is 0.

The initiator and target negotiate the recovery level supported. The minimum of the two values is selected.

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

12.21 SessionType

Use: LO, Declarative
Senders: Initiator
Scope: SW

SessionType= <discovery|normal>

Default is Normal.

The Initiator indicates the type of session it wants to create. The target can either accept it or reject it.

A discovery session indicates to the Target that the only purpose of this Session is discovery. The only requests a target accepts in this type of session are a text request with a SendTargets key and a logout request with reason "close the session".

The discovery session implies MaxConnections = 1 and overrides both the default and an explicit setting.

12.22 The 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.

13. IANA Considerations

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

References and Bibliography

[AC] A Detailed Proposal for Access Control, Jim Hafner, T10/99-245

[AES] J. Daemen, V. Rijman, "AES Proposal: Rijndael" NIST AES proposal, http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf, September 1999.

[XCBC] J. Black, P. Rogaway "Comments to NIST Concerning AES Modes of Operations: A Suggestion for Handling Arbitrary-Length Messages with the CBC MAC", http://csrc.nist.gov/encryption/modes/proposed-modes/xcbc-mac/xcbc-mac-spec.pdf, NIST proposed modes of operations, August 2001.

[AESCTR] J. Etienne, "The counter-mode and its use with ESP", Internet draft (work in progress), draft-etienne-ipsec-esp-ctr-mode-00.txt, May 2001.

[BOOT] P. Sarkar & team draft-ietf-ips-iscsi-boot-01.txt

[CAM] ANSI X3.232-199X, Common Access Method-3.

[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.

[COBS] S. Cheshire and M. Baker, Consistent Overhead Byte Stuffing, IEEE Transactions on Networking, April 1999.

[CRC] ISO 3309, High-Level Data Link Control (CRC 32).

[NDT] M. Bakke & team, draft-ietf-ips-iscsi-name-disc-03.txt

[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.

[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.
[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).
[Schneier] B. Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 2nd edition, John Wiley & Sons, New York, NY, 1996.
[SEC-IPS] B. Aboba & team "Securing iSCSI, iFCP and FCIP" - draft-ietf-ips-security-03.txt.
[SEQ-EXT] Steve Kent, IPsec sequence number extension proposal, IETF 50.
[SPC] NCITS.351:200, SCSI-3 Primary Commands (SPC).
[SPC3]T10/1416-D, SCSI-3 Primary Commands (SPC).
[XCBC] J. Black, P. Rogaway "Comments to NIST Concerning AES Modes of Operations: A Suggestion for Handling Arbitrary-Length Messages with the CBC MAC", http://csrc.nist.gov/encryption/modes/proposed-

modes/xcbc-mac/xcbc-mac-spec.pdf, NIST proposed modes of operations, August 2001.

Authors' Addresses

Julian Satran
IBM, Haifa Research Lab
MATAM - Advanced Technology Center
Haifa 31905, Israel
Phone +972.4.829.6264
E-mail: Julian_Satran@vnet.ibm.com

Kalman Meth
IBM, Haifa Research Lab
MATAM - Advanced Technology Center
Haifa 31905, Israel
Phone +972.4.829.6341
E-mail: meth@il.ibm.com

Ofer Biran
IBM, Haifa Research Lab
MATAM - Advanced Technology Center
Haifa 31905, Israel
Phone +972.4.829.6253
E-mail: biran@il.ibm.com

Daniel F. Smith
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099, USA
Phone: +1.408.927.2072
E-mail: dfsmith@almaden.ibm.com

Jim Hafner
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
Phone: +1.408.927.1892
E-mail: hafner@almaden.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

Mark Bakke
Cisco Systems, Inc.
6450 Wedgwood Road
Maple Grove, MN
USA 55311
Phone: +1.763.398.1000
E-Mail: mbakke@cisco.com

Randy Haagens
Hewlett-Packard Company
8000 Foothills Blvd.
Roseville, CA 95747-5668, USA
Phone: +1.916.785.4578
E-mail: Randy_Haagens@hp.com

Matt Wakeley (current address)
Sierra Logic, Inc.
Phone: +1.916.772.1234 ext 116
E-mail: matt_wakeley@sierralogic.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

Paul von Stamwitz (current address)
TrueSAN Networks, Inc.
Phone: +1.408.869.4219
E-mail: pvonstamwitz@truesan.com

Luciano Dalle Ore
Quantum Corp.
Phone: +1.408.232.6524
E-mail: ldalleore@snapserver.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

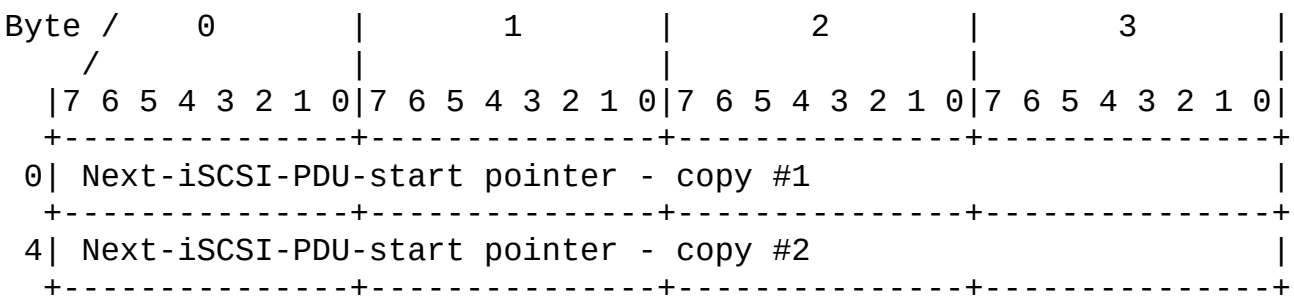
Yaron Klein
SANRAD
24 Raul Valenberg St.
Tel-Aviv, 69719 Israel
Phone: +972.3.765.9998
E-mail: klein@sanrad.com

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 schemes 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. In certain environments, a sender not willing to supply markers to a receiver willing to accept markers MAY suffer from a considerable performance degradation.

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 alto-

gether. If a receiver indicates that it desires a marker, the sender SHOULD agree (during negotiation) and provide the marker at the desired interval.

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 last 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.

As an example if the marker interval is 512 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.

A.3.1 FMarker

Use: IO
Senders: Initiator and Target

FMarker=<send|receive|send-receive|no>

Default is no.

This is a connection specific parameter.

Examples:

```
I->FMarker=send-receive
T->FMarker=send-receive
```

Results in the Marker being used in both directions while

```
I->FMarker=send-receive
T->FMarker=receive
```

Results in Marker being used from the initiator to the target, but not from the target to initiator.

A.3.2 RFMarkInt, SFMarkInt - offering

Use: IO
Senders: Initiator and Target

RFMarkInt=<number-from-1-to-65535>[,<number-from-1-to-65535>]
SFMarkInt=<number-from-1-to-65535>[,<number-from-1-to-65535>]

This is a connection specific parameter.

The receiver or sender indicates the minimum to maximum interval (in 4-byte words) it wants the markers. In case the receiver or sender only wants a specific value, only a single value has to be specified. The responding sender or receiver selects a value within the minimum and maximum offered or the only value offered or indicates through the FMarker key=value its inability to set and/or receive markers. 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). Whenever FMarker and RFMarkInt are both sent, they MUST appear on the same Login Request/Response.

The default is 2048.

A.3.3 SFMarkInt, RFMarkInt - responding

Use: IO
Senders: Initiator and Target

SFMarkInt=<number-from-1-to-65535>
RFMarkInt=<number-from-1-to-65535>

This is a connection specific parameter.

Indicates at what interval (in 4-byte words) the sender agrees to send or the receiver wants to receive the markers. The number MUST be within the range required offering party. 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).

Default is 2048.

Appendix B. Sync and Steering with Constant Overhead Word Stuffing (COWS)

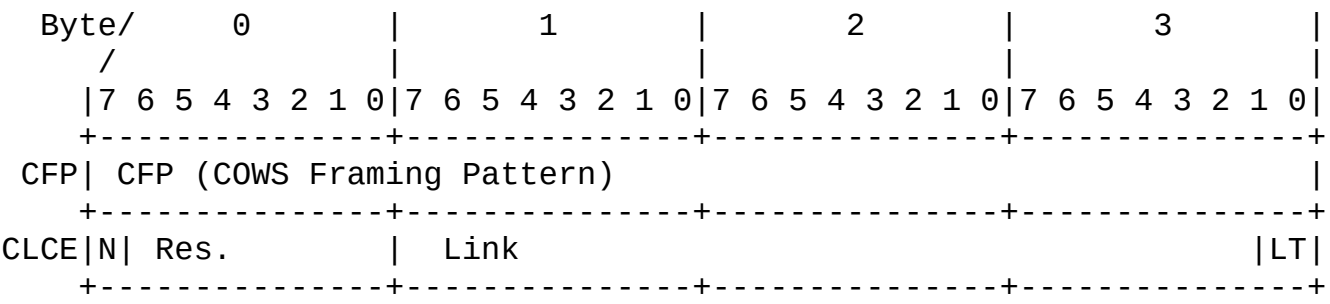
This appendix presents a simple scheme for synchronization (PDU boundary retrieval). The basic mechanism described is inspired by [COBS]. However, unlike the mechanism outlined in [COBS], here the PDU is extended by 2 or 3 4 byte words regardless of the PDU length. With COWS:

- The iSCSI PDU is "prefixed" by an 8 byte COWS header (CH), including a 4 byte word aligned COWS framing pattern (CFP) and a 4 byte COWS link chain element (CLCE).
- Any appearance of the pattern within the PDU is replaced either by a forward or (for long payloads) a backward link to the next appearance of the pattern, or to the end (for forward links) or an end of chain in indicator formatted as a CLCE.
- The iSCSI PDU may be followed by a trailer that consists of a single CLCE.

All of these elements form a COWS Extended PDU (CEPDU).

COWS uses a framing pattern defined by the sender. A special version of COWS that does not require pattern replacement, but requires the sender to guarantee that a CH will always appear at the beginning of a TCP segment, may be used by specialized software stacks and/or hardware adapters and its use may be negotiated (PDU Alignment Option - PAO).

The CH format is:



CFP is a pattern selected by the sender and communicated to the receiver during the login phase. A CFP has to be specified by the sender for each direction.

Except for the PAO, every occurrence of CFP within the payload is replaced by a CLCE.

The CLCE is composed of:

- N - a bit is selected to be the complement of the corresponding bit in the Framing pattern.
- Link - number of non-link 4 byte words to the next/previous link.
- LT - link type coded as follows:
  - 00 - Forward Last Link - no more links follow. The Link field is the number of words to the end of PDU.
  - 01 - Forward More Links - the link field is the number of words to the next link.
  - 10 - Backward Last Link - the link field is 0.
  - 11 - Backward More Links - the link field is the number of words from the preceding field.

The LT field in the CH MUST be 00 or 01, and all CLCE within the iSCSI header (if any) MUST have an LT field of 00 or 01 (the iSCSI header is encoded ONLY with forward links).

The sender can use the backward linking mechanism to avoid storing very long data payloads before sending them and MUST be processed by the receiver.

If using backward linking, the sender MUST include a tailing CLCE in the ECPDU. The tailing CLCE is the first CLCE in the "back-tracking chain" and MUST be linked to by the last CLCE in the "forward-tracking chain".

COWS ECPDU can follow one of the following outlines:

a) Only Forward Pointing

CH (mandatory)
.
.
Forward Pointing CLCE (optional)
.
.
.

```
   Forward Pointing CLCE (optional)
   .
   .
   Last Payload Word
```

b) Forward-and-Backward Pointing

```
   CH (mandatory)
   .
   .
   Forward Pointing CLCE (optional - may point to trailer if
   last forward)
   .
   .
   .
   Backward Pointing CLCE (optional - may have link of 0)
   .
   .
   Last Payload Word
   Backward Pointing CLCE (mandatory - may have a link of 0)
```

c) PDU alignment option

```
   CH (mandatory)
   .
   .
   .
   Last Payload Word (also end of TCP segment)
```

For cases a) and b), the payload on the wire is guaranteed not to contain a CFP or a word aligned pattern anywhere but in CH. For case c), the CFP is supposed the appear only aligned to TCP segment boundaries and be implement with specialized software stacks and hardware. For this case the Link value, LT and the Reserved bits may is used as a further validity checks (TBD???).

if all cases of the iSCSI PDUs are not constrained to a one or a limited number of TCP segments.

B.4 Negotiation

B.5 Sent PDU processing

pseudo-language description...

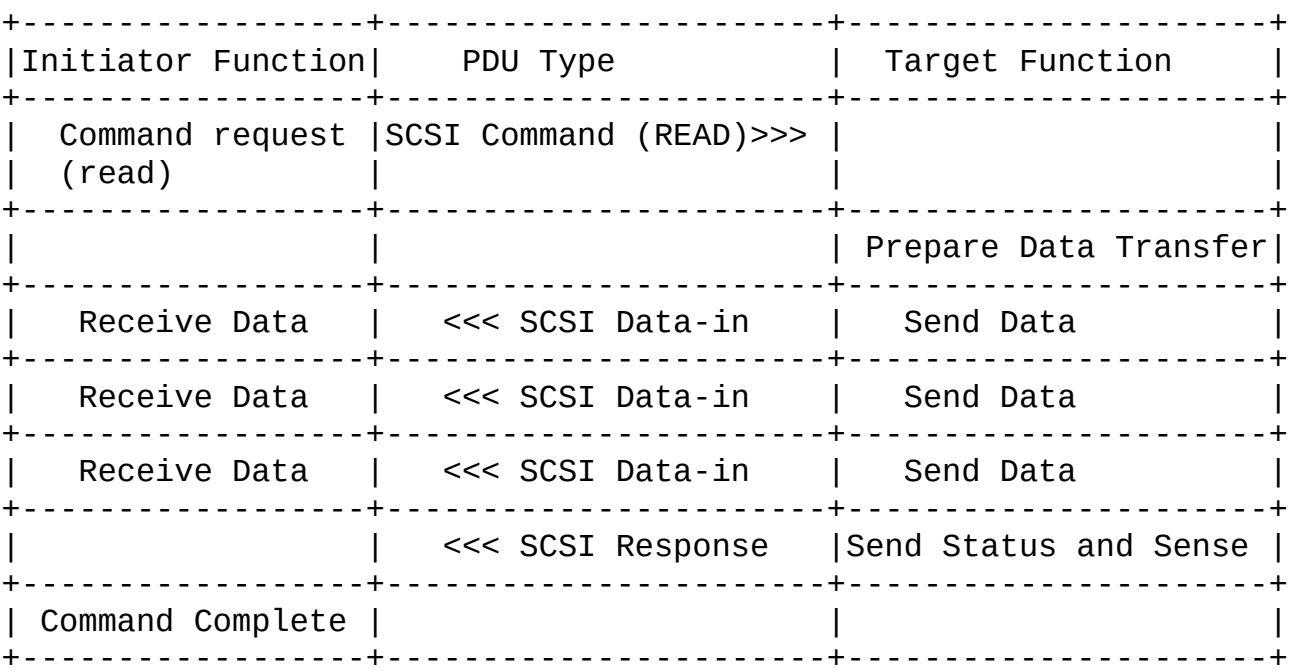
B.6 Received PDU processing

pseudo-language description

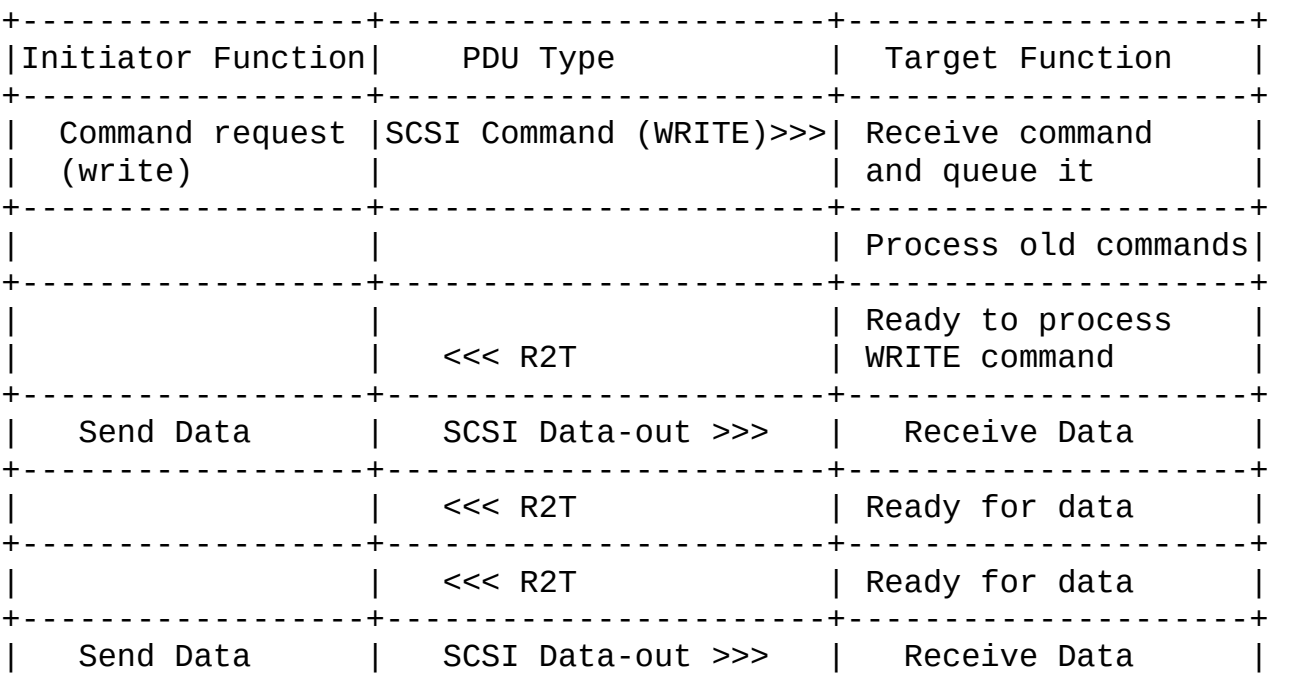
B.7 Search for framing processing

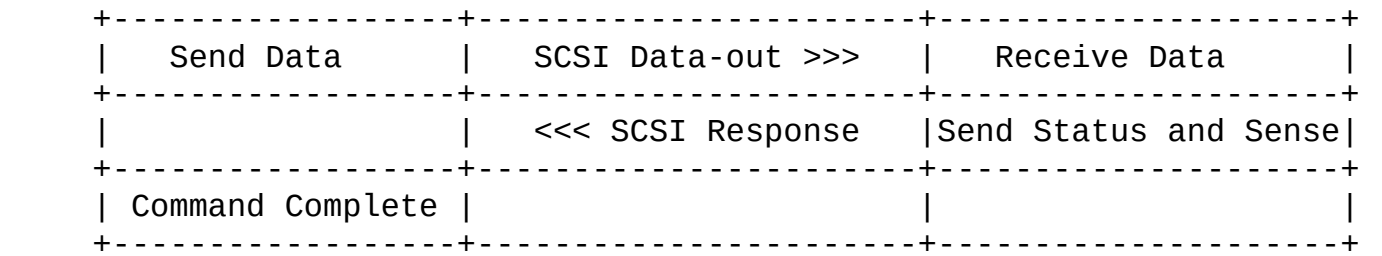
Appendix C. Examples

C.8 Read Operation Example



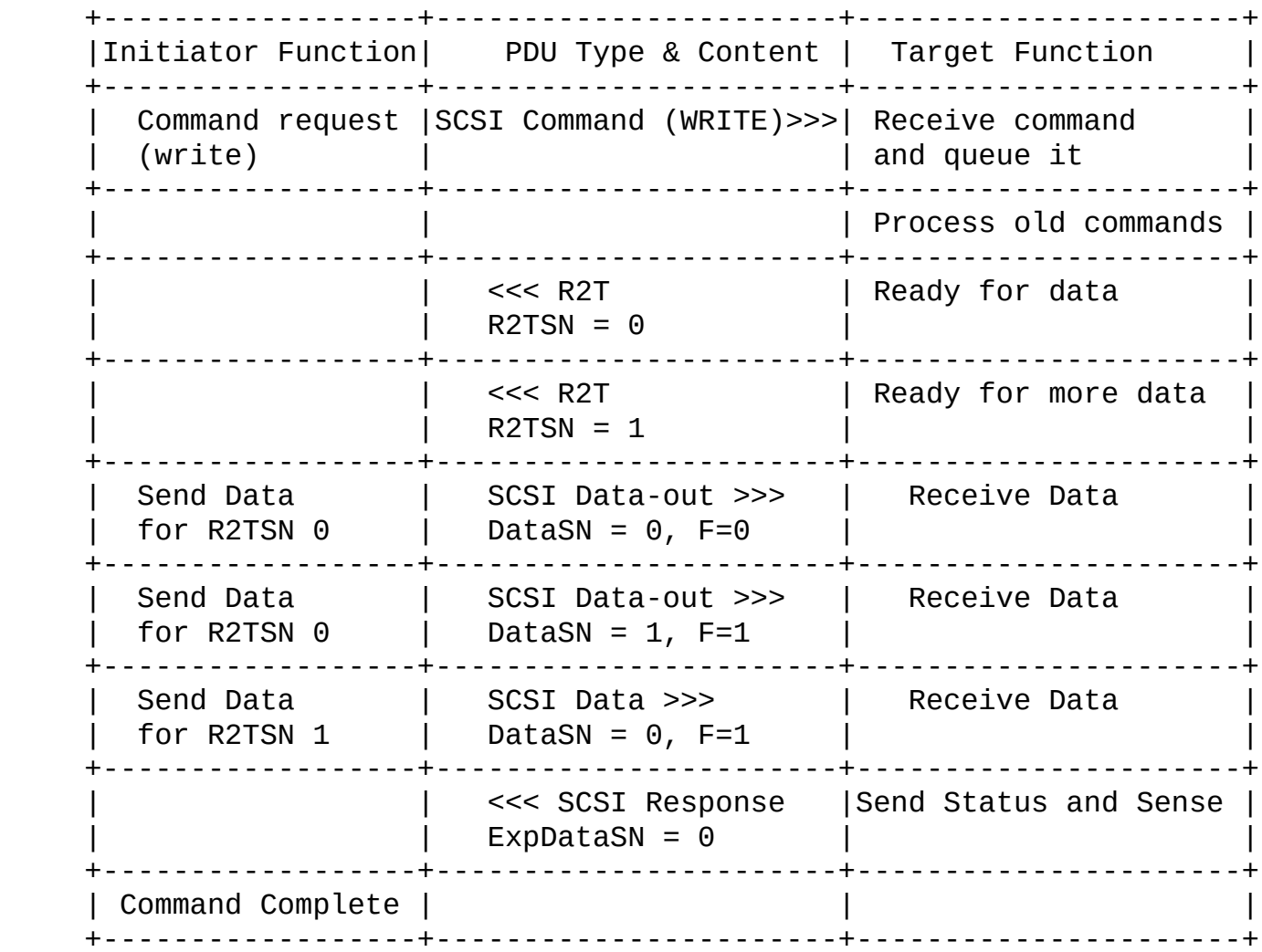
C.9 Write Operation Example



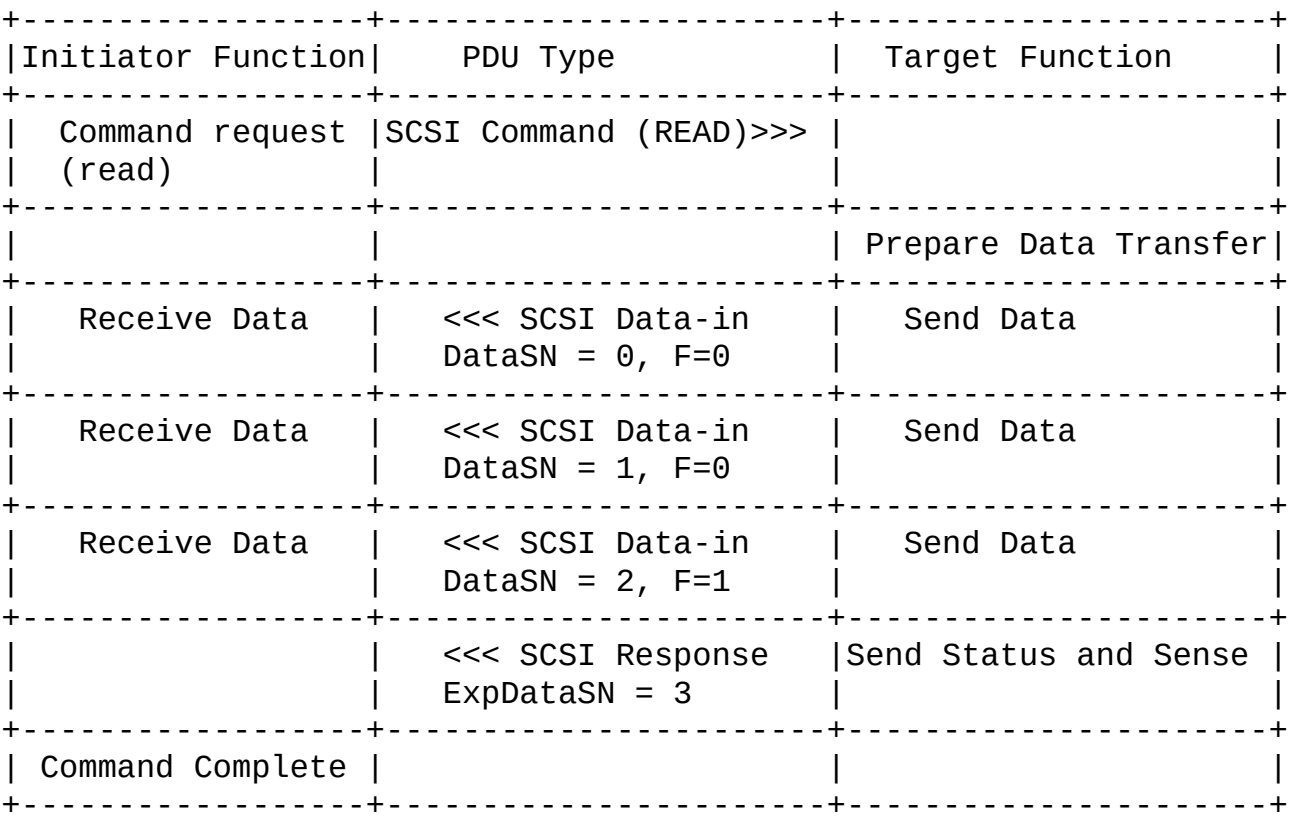


C.10 R2TSN/DataSN use Examples

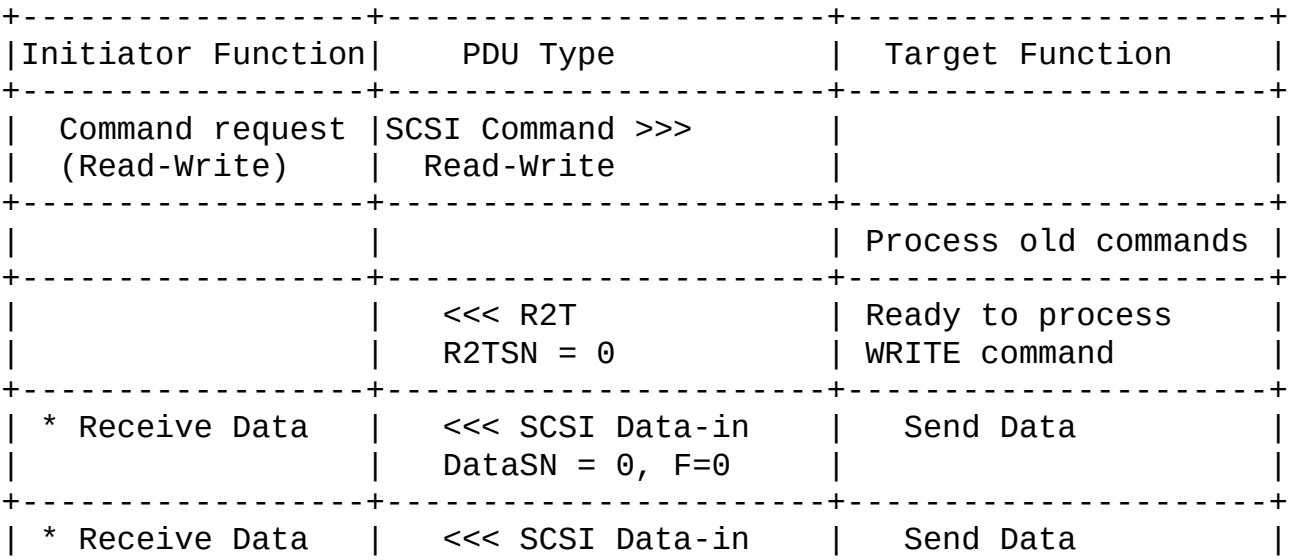
Output (write) data DataSN/R2TSN Example

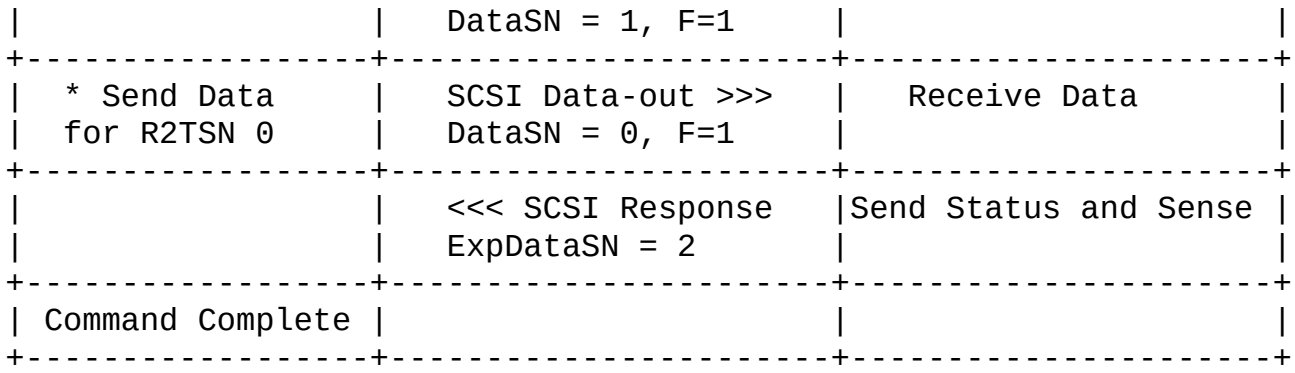


Input (read) data DataSN Example



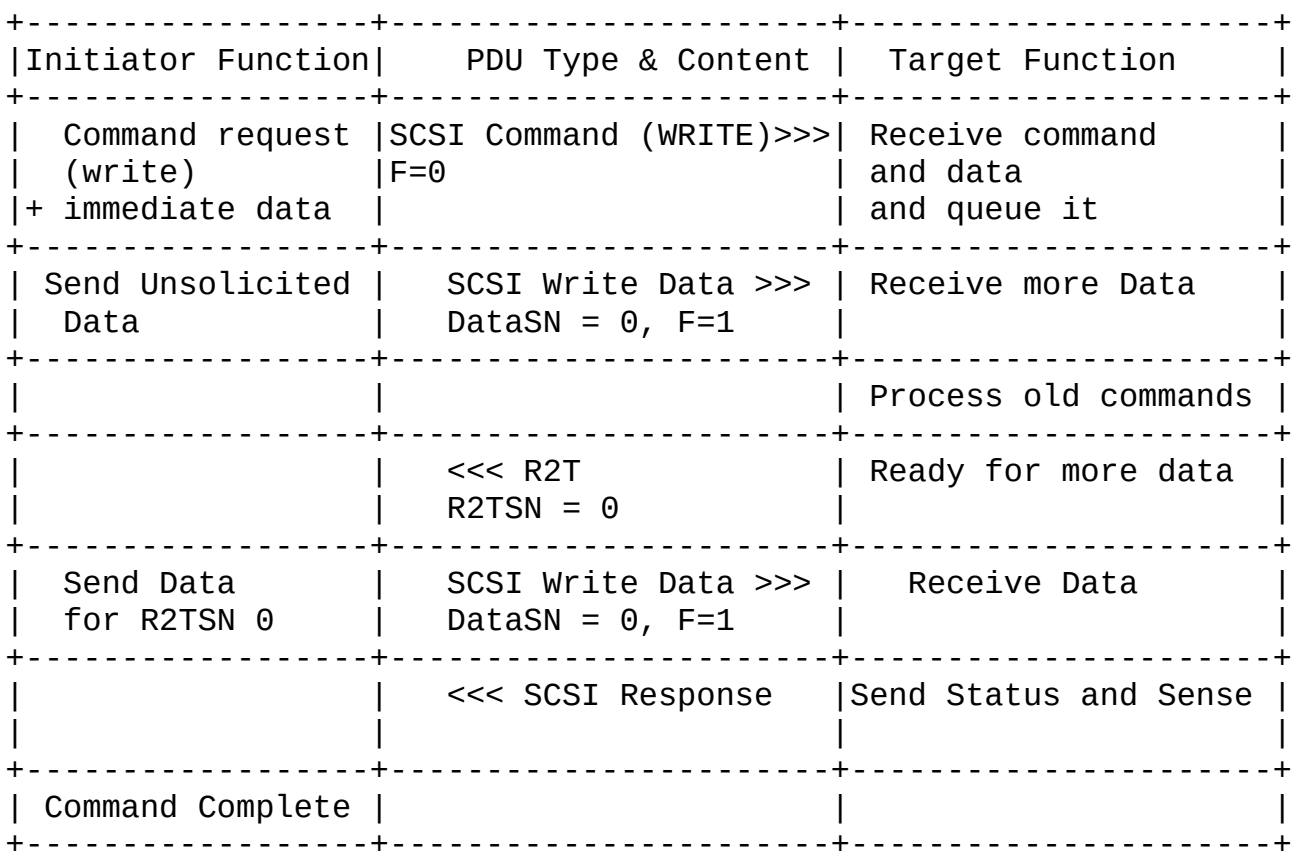
Bidirectional DataSN Example





*) 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



C.11 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 D. 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) FirstBurstSize=0
T-> Login (CSG,NSG=1,0 T=0) FirstBurstSize=8192 MaxBurst-
  Size=8192
I-> Login (CSG,NSG=1,0 T=0) MaxBurstSize=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 respond-
  ing 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 E. SendTargets Operation

To reduce the amount of configuration required on an initiator, iSCSI provides the SendTargets text request. This initiator sends this command to request 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 IP addresses, and MUST support the SendTargets command on the discovery session. A target MUST support the SendTargets command on operational sessions; these will only return address information about the target to which the session is connected, and do not return information about other targets.

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 MAY 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 MAY NOT return targets other than the one to which the session is logged in.

The response to this command is a text response that contains a list of zero or more targets and, optionally, their addresses. Each target is returned as a target record. A target record begins with the TargetName text key, followed by a list of TargetAddress text keys, and bounded by the end of the text response or the next TargetName key, which begins a new record. No text keys other than TargetName and TargetAddress are permitted within a SendTargets response.

For the format of the TargetName, see Section 12.4 TargetName.

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 MAY 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.

A SendTargets response MUST NOT contain iSCSI default target names.

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 and tcp port are as specified in the Section 2.2.7 Naming and Addressing.

Each TargetAddress belongs to a portal group, identified by its numeric, decimal portal group tag. 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 session to a default target open, and MAY send subsequently 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 could have been returned instead of an IP address, and that an IPv6 addresses (5 to 16 dotted-decimal numbers) could have also been returned.

The next text response shows a target that supports spanning sessions across multiple addresses, which indicates 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 F. 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.

F.12 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 ExpectedStatSN;
        int MissingStatSNList[MaxMissingSPDU];
        Boolean PerformConnectionCleanup;
};

struct Session {
        int NumConnections;
        int NextCmdSN;
        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);
```

F.13 Within-command Error Recovery Algorithms

F.13.1 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.

F.13.2 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);
      }
   }
```

F.13.3 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 size and
                      data received is less than FirstBurstSize)
                   or {current PDU is solicited and the size 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);
         }
     }
   }
```

F.14 Within-connection Recovery Algorithms

F.14.1 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 size);

   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 since 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.

F.14.1.1 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 our NextCmdSN) {
                        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);
              }
          }
     }
```

F.14.1.2 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,
                            LogoutLoginMinTime, LogoutLoginMaxTime);
    }
}
```

F.14.2 Connection Recovery Algorithms

F.14.2.1 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.

F.14.2.2 Initiator Algorithms

```
Receive-a-In-PDU(Connection, CurrentPDU)
{
    check-basic-validity(CurrentPDU);
    if (Header-Digest-Bad) discard, return;
    Retrieve TCB from CurrentPDU.InitiatorTaskTag.
    if (CurrentPDU.type = Async) {
        if (CurrentPDU.AsyncEvent = ConnectionDropped) {
           Retrieve the AffectedConnection for CurrentPDU.Parameter1.
          AffectedConnection.State = CLEANUP_WAIT;
        } else if (CurrentPDU.AsyncEvent = LogoutRequest)) {
           Retrieve the AffectedConnection for CurrentPDU.Parameter1.
          AffectedConnection.State = LOGOUT_REQUESTED;
          AffectedConnection.PerformConnectionCleanup = TRUE;
          Start-Timer(Connection-Cleanup-Handler,
                      AffectedConnection, CurrentPDU.Parameter2);
        } else if (CurrentPDU.AsyncEvent = SessionDropped)) {
          for (each Connection) {
              Connection.state = CLEANUP_WAIT;
          }
          Session.state = FAILED;
          Start-Timer(Session-Continuation-Handler,
                      Session, CurrentPDU.Parameter2);
        }

    } 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.
    Start-Timer(Connection-Resource-Timeout-Handler,
              Connection, LogoutLoginMaxTime);
    if (Connection can still exchange iSCSI PDUs) {
        NewConnection = Connection;
    } else {
        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;
            Start-Timer(Connection-Cleanup-Handler, Connection,
                                              LogooutLoginMinTime);

        } else {
            Connection.State = FREE;
        }
    }
```

F.14.2.3 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, Sucess);
               } 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,
                                (LogoutLoginMinTime+LogoutLoginMinTime));
            if (this Session has full-feature phase connections left) {
                DifferentConnection =
                   Pick-A-Logged-In-Connection(Session);
                Build-And-Send-Async(DifferentConnection,
                      DroppedConnection, LogoutLoginMinTime,
                        LogoutLoginMaxTime);
            }
        } else {
            Connection.State = FREE;
        }
    }
```

Full Copyright Statement