IPS
Internet Draft
Document: draft-ietf-ips-iSCSI-08.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

Yaron Klein
SANRAD

# 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), 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) and 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 (RhapsodyNetworks), 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 "IPSec"[SEC-IPS] documents.

The "Naming & Discovery" 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" is authored by:

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

The "IPSec" 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-07 to draft-ietf-ips-iSCSI-08:

- Clarified the use of initiator task tag with regard to the SCSI tag in 3.2.1.8
- Added a clarification to 2.2.2.1 - response to a command should not precede acknowledgment.
- Added clarification to 3.7 - good status in Data-In must be supported by initiators
- Clarified InitiatorName is required at login in 5.1
- Another clarification for SecurityContextComplete in 5.2
- 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 CopyManager Session types
- Changed explanation of ExpDataSN
- Removed/corrected response 05 in 3.4.3
- Brought 1.2.7 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 1.2.5
- Added new Reject reason codes, tabular listing and a pointer to 3.17.1
- Added additional Reject usage semantics on CmdSN and DataSN to 3.17.1
- 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 7.
- 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 3.6 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 3.4
- 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 5, changed all examples in Appendix A 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 chapter 4, out DataPDULength, DataSequenceOrder
- Changed all sense keys to aborted command in the table in 3.4.2
- 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 F. 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 10 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 section 7.3.
- Several changes in section 8 corresponding to the new task management function "reassign". Other language changes in section 8 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 8.12 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 3.17 to use the "Invalid SNACK" reason code.
- Removed a Logout reason code in 3.14 to be consistent with section 3.9.
- 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 FirstBurstSize and a response (error) for targets not supporting less
- Many more minor editorial changes, clarifications, typos etc.
- Responses in same position in SCSI response, logout, task etc.

Table of Contents

## 1. Definitions

- 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.

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

- 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.

- 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).

- 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".

- 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.

- ISID (Initiator Session ID): An ID generated by the initiator during the leading login for a session. It is used for all additional logins for the same session. Between a given iSCSI Initiator and iSCSI Target Portal Group (SCSI target port), there can be only one session with a given ISID (identifying a SCSI initiator port).

- SSID (Session ID): A session is defined by a session ID that is composed of an initiator part (ISID) and a target part (TSID).

- 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.

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

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

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

- 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 is usable by some iSCSI Nodes contained in that Network Entity to gain access to the IP network.

- 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.

- 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 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.

- 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 Initiator Node: The "initiator".

- iSCSI Target Node: The "target".

- 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.

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

- 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 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 SCSI Initiator Port and SCSI Target Port are different.

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

- 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 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.

- 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 initiator to target, while inbound or incoming transfers are from target to initiator.

# 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 to familiarize readers with the terminology of the SCSI architecture.

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 processed 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) is the data structure used to contain the command parameters that are to be handed by an initiator 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.

In the rest of this document, the terms "initiator" and "target" refer to "iSCSI initiator node" and "iSCSI target node", respectively (see 2.5.1) 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 specified otherwise, status, response or numbered response have the same meaning.

### 2.2.1 Layers and Sessions

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

-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 2.5.2). 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, like LUN, are the same. In addition, across all connections within a session, a target sees one "initiator image". Initiator identifying elements like 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 even targets and initiators supporting 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.

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

#### 2.2.2.1 Command Numbering and Acknowledging

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

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

Commands in transit from the initiator to the target layer 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 as such through an immediate delivery flag. They too carry CmdSN, but 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 includes 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 they are supposed to act upon.  However, their CmdSN is a marker of their position in the stream of commands.  The task management command MUST carry the CmdSN that would be given to the next non-immediate command.  The initiator and target must ensure that the task management commands act as specified by SAM2 - i.e., both commands and responses appear as if delivered in order.

Not covered in this document are the means by which one may request immediate delivery for a command or by which iSCSI will decide by itself to mark a PDU for immediate delivery.

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

Except for 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 command for execution if so required by some prior SCSI or iSCSI action (e.g., clear task set Task Management request received before all the commands it was supposed to act on).  Delivery for execution means delivery to the

SCSI execution engine or an iSCSI-SCSI protocol specific execution engine (e.g., for text requests).

The initiator and target are assumed to have three registers, unique session wide, 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.
- ExpCmdSN - the next expected command by the target. The target acknowledges all commands up to but not including this number and 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 is able to 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.

MaxCmdSN and ExpCmdSN fields are processed as follows:

-if the PDU MaxCmdSN is less than the PDU ExpCmdSN-1 (in Serial Arithmetic Sense and with a difference bounded by 2**31-1), they are both ignored
-if the PDU MaxCmdSN is less than the local MaxCmdSN (in Serial Arithmetic Sense and with a difference bounded by 2**31-1), it is ignored; else it updates the local MaxCmdSN
-if the PDU ExpCmdSN is less than the local ExpCmdSN (in Serial Arithmetic Sense and with a difference bounded by 2**31-1), it is ignored; else it updates the local ExpCmdSN

This sequence is required as updates may arrive out of order because they travel on different TCP connections.

The target MUST NOT transmit a MaxCmdSN that is more than 2**31 - 1 above 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 this range or non-immediate duplicates within the range that have not been flagged with the retry bit (the X bit in the opcode).

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 execution state (can't find the Initiator Task Tag).  Afterwards CmdSN becomes irrelevant.  Testing for execution state 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 execution state is found.

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.

Status numbering starts with the Login response to the first Login request (C=0) of the connection. The Login response includes an initial value for status numbering.

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 difference between StatSN and ExpStatSN may indicate a failed connection. Initiators MUST undertake recovery actions if the difference is greater than 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 - i.e. the first R2T has an R2TSN of 0 and advances by 1 for each subsequent R2T. Unlike command and status, the data PDUs and R2Ts are not acknowledged except as implied by status. The DataSN/R2TSN field is meant to enable the initiator to detect missing data PDUs and simplify this operation at the target.

For any given write command a target must have issued less than 2**32-1 R2Ts. Any input or output data sequence MUST contain less than 2**32-1 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, open a security association protocol, and mark the connection as belonging to an iSCSI session.

A session is used to identify to a target all the connections with a given initiator that belong to the same I_T nexus (See 2.5.2 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 10 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 5.

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 and ISID (InitiatorName is described later in this part). Any persistent state (e.g., persistent reservations) on the target 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 2.5.1) and internally in an implementation dependent way. As ISID is used to identify persistent state, it is subject to reuse restrictions (see 2.5.3).

Before full feature phase is established, only Login 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.

In "list" negotiation, the offering party sends for each key a list of values (which may include "none") in its order of preference.

The responding party answers with the first value from the list it supports and is allowed to use for the specific initiator.

The value "none" MUST always be used to indicate a missing function. However, none is a valid selection only if it is explicitly offered.

If a target is not supporting, or not allowed to use with a specific initiator, any of the offered options, it may use the value "reject". The values "none" and "reject" are reserved and must be used only as described here.  Any key not understood is answered with "NotUnderstood".

The general format of text negotiation is:

```
   Offer-> <key>=<value1>,<value2>,...,<valuen>
   Answer-> <key>=<valuex>|reject|NotUnderstood
```

In "numerical" negotiations, the offering and responding party state a numerical value. The result of the negotiation is key dependent; frequently the lower or the higher of the two values is used.

For numerical negotiations, the responding party MUST respond with the required key.

Binary negotiations (for keys taking the values yes or no) are a restricted form of numerical negotiations and the result is a key dependent Boolean function of the two inputs. The negotiation MAY proceed only up to the point where both parties can unequivocally compute the result; continuing beyond this point is OPTIONAL (e.g., if the function is AND and one of the parties says "no" then this may end the negotiation).

The value "?" with any key has the meaning of enquiry and should be answered with the current value or "NotUnderstood".

The target may offer key=value pairs of its own. Target requests are not limited to matching key=value pairs as offered by the initiator. However, only the initiator can initiate the negotiation start (through the first Text request) and completion (by setting to 1 and keeping to 1 the F bit in a Text request).

### 2.2.5 iSCSI Full Feature Phase

Once the initiator is authorized to do so, the iSCSI session is in 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) either 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 on it or the connection is currently logged out.  In 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 8.1.

As an illustration of the above rule, SCSI commands that require data and/or 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.

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 and 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) as well as 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.  FirstBurstSize is an iSCSI specific mode page value that can be set and enquired according with SCSI commands.

Targets operate in either solicited (R2T) data mode or unsolicited (non R2T) data mode.  In unsolicited mode, an initial R2T allowing 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 operating 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. The agreed upon limits for the first burst as well as the maximum data PDU are recorded in (and are retrievable from) the disconnect-reconnect mode page.

A target SHOULD NOT silently discard data and request retransmission through R2T. Initiators SHOULD NOT do any scoreboarding for data - 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 that are 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 to achieve deadlock-free operation. Unsolicited data MUST be sent on every connection in the same order in which commands were sent. A target receiving data out of order SHOULD terminate the session.

### 2.2.6 iSCSI Connection Termination

Connection termination is assumed to be an exceptional event. Graceful TCP connection shutdowns are done by sending TCP FINs. Graceful connection shutdowns MUST only occur when there are no outstanding tasks that have allegiance to the connection and when the connection is not in full-feature phase. A target SHOULD respond rapidly to a FIN from the initiator by closing its half of the connection after waiting for all outstanding commands that have allegiance to the connection to conclude and send their status. Connection termination with outstanding commands may require recovery actions.

Connection termination is also required as a prelude to recovery. By terminating a connection before starting recovery, the initiator and target can avoid having stale PDUs being received after recovery. In this case, the initiator sends a Logout request on any of the operational connections of a session indicating what connection should be terminated.

Logout can also be issued by an initiator at the explicit request of a target (through an Asynchronous Message PDU) or the connection can be autonomously terminated by the target after announcing it to the initiator (through an Asynchronous Message PDU).

### 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 when generating them. 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 2.4) 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 iSCSI layer; they are opaque and case-sensitive for the 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.

```
<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, a default port, to be 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]
[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 containing 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, SCSI requires, in some cases, persistent identification of the SCSI initiator port name (for iSCSI, the InitiatorName plus the ISID of the session) (See 2.5.2 and 2.5.3).

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

All iSCSI session and connection parameters are reinitialized on session and connection creation.

Commands persist beyond connection termination if the session persists and command recovery within session is supported. However command execution as perceived by iSCSI (i.e., involving iSCSI protocol exchanges for the affected task) is suspended when a connection is dropped until a new allegiance is established by the 'task reassign' task management function (section 3.5)

### 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 that are of varying length. 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; messages are processed one after the other. In the presence of IP packet reordering (e.g., 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 containing 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 an Appendix C.  To make those 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:

```
        +------------------------+
        |        SCSI            |
        +------------------------+
        |       iSCSI            |
        +------------------------+
        |  Sync and Steering     |
        |  +-------------------+ |
        |  |       TCP         | |
        |  +-------------------+ |
        +------------------------+
        | Lower Functional Layers|
        |        (LFL)           |
        +------------------------+
        |         IP             |
        +------------------------+
        |        Link            |
        +------------------------+
```

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. Note that 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 model of layering, 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 doe not need to examine the stream content to discover the PDU boundaries. If a specific implementation does 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 understandable to the Sync and Steering Layer.

The Sync and Steering Layer (which itself 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 to every sent data item (IP packet, TCP packet or some other superstructure) enough information 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.

The retained information can be released whenever the transmitted data is acknowledged by the receiver (in 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. Then it 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 it refers to 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:

```
+-----------------------------------+
|             SCSI                  |
+-----------------------------------+
|            iSCSI                  |
+-----------------------------------+
|   Upper Functional Layers (UFL)   |
+-----------------------------------+
|     Sync and Steering             |
|  +-----------------------------+  |
|  |            TCP              |  |
|  +-----------------------------+  |
+-----------------------------------+
|   Lower Functional Layers (LFL)   |
+-----------------------------------+
|              IP                   |
+-----------------------------------+
|             Link                  |
+-----------------------------------+
```

In this model, UFL can be TLS (see[RFC2246]) or some other transport conversion mechanism (a mechanism changing the TCP stream but 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 has to 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 need to be buffered (in temporary buffers) since the iSCSI header that indicates what SCSI buffers the data is to be steered to 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 size of an iSCSI PDU it will accept.

## 2.3 Third Party Commands

SCSI allows every addressable entity to be either an initiator or a target. In host-to-host communication, each such entity can take on the initiator role. In typical I/O operations between a host and a peripheral subsystem, the host plays the initiator role and the peripheral subsystem plays the target role.

For EXTENDED COPY and other third party SCSI commands, that involve device-to-device communication, such as (EXTENDED) COPY and COMPARE, SCSI defines a copy-manager. The copy-manager takes on the role of initiator in the device-to-device communication. The copy-manager is the "original-target" of the command and acts as initiator for a (variable) number of the devices, which are called sources and destinations. Sources and destinations act as targets.
The copy operation is described in one CDB to the copy-manager, along with a series of descriptor blocks. Each descriptor block addresses source and destination target, LU, and a description of the work to be done in terms of blocks or bytes as required by the device types.
The relevant SCSI standards do not require full support of the (EXTENDED) COPY or COMPARE, nor do they provide a detailed execution model.

## 2.4 iSCSI session types

iSCSI defines two types of sessions:

normal operational session - an unrestricted session
discovery-session - a session opened only for target
discovery; the target MAY accept only text requests with
the SendTargets key

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

## 2.5 SCSI to iSCSI concepts mapping model

The following diagram shows an example of how multiple iSCSI Nodes (targets in this case) can co-exist within the same Network Entity and can share Network Portals (IP addresses and TCP ports). Other more complex configurations are also possible. Detailed descriptions of the components of these diagrams are given in 2.5.1.

```
              +-----------------------------------+
              |  Network Entity (iSCSI Client)    |
              |                                   |
              |         +-------------+           |
              |         | iSCSI Node  |           |
              |         | (Initiator) |           |
              |         +-------------+           |
              |            |       |              |
              | +--------------+ +--------------+ |
              | |Network Portal| |Network Portal| |
              | |   10.1.30.4  | |   10.1.40.6  | |
              +-+--------------+-+--------------+-+
                       |               |
                       |  IP Networks  |
                       |               |
              +-+--------------+-+--------------+-+
              | |Network Portal| |Network Portal| |
              | |  10.1.30.21  | |   10.1.40.3  | |
              | |  TCP Port 4  | |  TCP Port 4  | |
              | +--------------+ +--------------+ |
              |        |               |          |
              |        -----------------          |
              |           |         |             |
              |  +-------------+ +--------------+ |
              |  | iSCSI Node  | | iSCSI Node   | |
              |  |  (Target)   | |  (Target)    | |
              |  +-------------+ +--------------+ |
              |                                   |
              |   Network Entity (iSCSI Server)   |
              +-----------------------------------+
```

### 2.5.1 iSCSI Architectural Model

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

a) 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 (see item (c)), each of which is usable by some iSCSI Nodes (see item (b)) contained in that Network Entity to gain access to the IP network.

b) 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 (see item (c)). An iSCSI Node is identified by its iSCSI Name (see 2.2.7 and Appendix D).  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.

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.

c) 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.

d) 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 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 Groups are identified within an iSCSI Node by a portal group tag, a simple integer value between 1 and 65535 (see SendTargets in Appendix D, item 11). 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 Appendix E). See 9.1.1 for references to the Initiator Portal Groups.

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 2.5.3.2 SCSI Mode Pages)

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

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

### 2.5.2 SCSI Architecture Model

This part describes the relationship between the SCSI Architecture Model [SAM2] constructs of 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 2.5.3.

a) 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 (see 2.4). 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 - 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 SCSI Initiator Port and SCSI Target Port are different.

SCSI Initiator Port: this maps to the endpoint of an iSCSI normal operational session (see 2.4). 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 shall 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 4 bytes long, followed by

- the 2byte value of ISID (for SCSI initiator port) or portal group tag (for SCSI target port)in network byte order (BigEndian).

SCSI port names have a maximum length of 260 bytes and must be a multiple of 4 bytes long. The ASCII character 'i' or 't' is the label that identifies this as either a SCSI Initiator Port or a SCSI Target Port name, and so also provides the interpretation of the value of the final two bytes as either an ISID or a portal group tag.

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

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

### 2.5.3 Consequences of the model

This section describes implementation and behavioral requirements that result from the mapping of SCSI constructs to iSCSI constructs defined above. Two assumptions are at the basis of the requirements stated here.

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

b) Between a given SCSI initiator port and SCSI target port, there can be only one I_T nexus (session); 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), then can be only one session with a given ISID (identifying a SCSI initiator port). See 3.12.8.

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 - this would be analogous to a single SCSI Initiator Port having conversations with multiple ports on the same target or ports on other targets.  It is also possible to have multiple sessions with different ISIDs to the same Target Portal Group.  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, two nexuses with the same identifier should never occur.

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

### 2.5.3.1 I_T nexus state

Certain nexus relationships contain explicit state (e.g., initiator-specific mode pages or reservation state) that may need to be preserved by the target (actually, 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 do session recovery as described in section 8.  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.

### 2.5.3.2 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.

# 3. 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 appearing 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 bits not defined MUST be set to zero. Any reserved fields and values MUST be 0 unless specified otherwise.

## 3.1 iSCSI PDU Length and Padding

iSCSI PDUs are padded to an integer number of 4 byte words. The padding bytes SHOULD be 0.

## 3.2 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 there MAY be a header-digest. The data segment MAY also be followed by a data-digest.

The Basic Header Segment (BHS) is the first segment in all 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:

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

All PDU segments and digests are padded to an integer number of 4 byte words. The padding bytes SHOULD be sent as 0.

### 3.2.1 Basic Header Segment (BHS)

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

The format of the BHS is:

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

### 3.2.1.1 X

For request PDUs (PDUs from initiator to target) the X bit set to 1 is a Retry/Restart indicator. This bit is always 1 for response PDUs (PDUs from target to initiator).

### 3.2.1.2 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).

### 3.2.1.3 Opcode

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

All other opcodes are reserved.

#### 3.2.1.4 Opcode-specific Fields

These fields have different meanings for different opcode types.

#### 3.2.1.5 TotalAHSLength

Total length of all AHS header segments in 4 byte words including padding if any.

#### 3.2.1.6 DataSegmentLength

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

#### 3.2.1.7 LUN

Some opcodes operate on a specific Logical Unit. The Logical Unit Number (LUN) field identifies which Logical Unit. If the opcode does not relate to a Logical Unit, this field either is ignored or may be used in an opcode specific way. The LUN field is 64-bits and it is to be formatted in accordance with [SAM2].

#### 3.2.1.8 Initiator Task Tag

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

The initiator task tag may be used by SCSI too as part of the SCSI task identifier as the time span during which an iSCSI initiator task tag has to be unique extends over the time span during which a SCSI task tag has to be unique.  However, the iSCSI Initiator Task Tag has to exist and be unique even for untagged SCSI commands.

### 3.2.2 Additional Header Segment (AHS)

The general format of an AHS is:

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

#### 3.2.2.1 AHSType

The AHSType field is coded as follows:

```
   bit 7 - Drop Bit - if set to 1 this AHS may be ignored if not
   understood; if set to 0 this AHS must be rejected if not
   understood.
   bit 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
```

#### 3.2.2.2 AHSLength

This field contains the effective length in bytes of the AHS excluding AHSType and AHSLength (not including padding). The AHS is padded to an integer number of 4 byte words.

#### 3.2.2.3 Extended CDB AHS

The format of the Extended CDB AHS is:

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

3.2.2.4 Bidirectional Expected Read-Data Length AHS

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

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

3.2.3 Header Digest and Data Digest

Optional header and data digests protect the integrity of 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 implies also a zero-length data-digest.

### 3.2.4 Data Segment

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

### 3.3 SCSI Command

The format of the SCSI Command PDU is:

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

### 3.3.1 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 contain Task Attributes

The Task Attributes (ATTR) has 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 |

Having both the W and the F bit set to 0 is an error.
The R and W MAY both be 1 while the corresponding Expected Data Transfer Lengths are 0 but they MUST NOT both be 0 when the corresponding Expected Data Transfer Lengths are not 0.

### 3.3.2 CRN

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

### 3.3.3 CmdSN - Command Sequence Number

Enables ordered delivery across multiple connections in a single session.

### 3.3.4 ExpStatSN

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

### 3.3.5 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 (W flag set to 1 and R flag set to 0) operation, the initiator uses this field to specify the number of bytes of data it expects to transfer for this operation. For a unidirectional read (W flag set to 0 and R flag set to 1) operation, 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 indicating 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 immediate data part that follows the command (if any) are the same then no more data PDUs are expected to follow. In this case, the F bit MUST be set to 1.

If the Expected Data Transfer Length is higher than the 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 in error a command for which the Expected Data Transfer Length is higher than the FirstBurstSize and for which the initiator sent less than FirstBurstSize unsolicited data.

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

### 3.3.6 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.

### 3.3.7 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 (as from a WRITE operation) can be placed in the same PDU (both cases referred to as immediate data). Those data are governed by the general rules for solicited vs. unsolicited data.

## 3.4 SCSI Response

The format of the SCSI Response PDU is:

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

### 3.4.1 Flags (byte 1)

bit 7-5 Reserved
bit 4   (o) same as bit 2 but for the read operation of a bidirectional operation
bit 3   (u) same as bit 1 but for the read-part of a bidirectional operation

bit 2 (O) set for Residual Overflow. In this case, the Residual Count indicates how many bytes could not be transferred because the initiator's Expected Data Transfer Length was too small. For a bidirectional operation, contains the residual for the write operation.
bit 1 (U) set for Residual Underflow. In this case, the Residual Count indicates how many bytes were not transferred out of those expected to be transferred. For a bidirectional operation, contains the residual for the write operation.
bit 0 (0) Reserved

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

### 3.4.2 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.

### 3.4.3 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 |

"Not enough unsolicited data" condition is reported by the target 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 can't be used.

### 3.4.4 Residual Count

The Residual Count field is valid only 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 U bit is set, the Residual Count indicates how many bytes were not transferred out of those expected to be transferred.  If the O bit is set, the Residual Count indicates how many bytes could not be transferred because the initiator's Expected Data Transfer Length was too small.

### 3.4.5 Bidirectional Read Residual Count

The Bidirectional Read Residual Count field is valid only 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 u bit is set, the Bidirectional Read Residual Count indicates how many bytes were not transferred to the initiator out of those expected to be transferred.  If the o bit is set, the Bidirectional Read Residual Count indicates how many bytes could not be transferred to the initiator because the initiator's Expected Bidirectional Read Transfer Length was too small.

### 3.4.6 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:

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

#### 3.4.6.1 SenseLength

Length of Sense Data.

### 3.4.7 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.

### 3.4.8 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 case of responses sent because of a retransmission request the StatSN used MUST be the same as the first time the PDU was sent unless the connection was restarted since then.

### 3.4.9 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.

### 3.4.10 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 that indicates to the initiator that the target can't 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.

## 3.5 Task Management Function Request

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

### 3.5.1 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 (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 by this initiator on the Logical Unit.
3    CLEAR ACA - clears the Auto Contingent Allegiance condition.
4    CLEAR TASK SET - Aborts all Tasks (from all initiators) for the Logical Unit.
5    LOGICAL UNIT RESET
6    TARGET WARM RESET

```
7    TARGET COLD RESET
8    TASK REASSIGN - reassign connection allegiance for the
     task identified by the Initiator Task Tag field on this
     connection, thus resuming the iSCSI exchanges for the task
```

For all these functions, if executed, the Task Management Function Response MUST be returned using the Initiator Task Tag to identify the operation for which it is responding. All those functions apply to the referenced tasks regardless if they are proper SCSI tasks or tagged iSCSI operations.  Task management commands must be executed as if all the commands having a CmdSN lower or equal to the task management CmdSN have been received by the target (i.e., have to be executed as if received for ordered delivery even when marked for immediate delivery).  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.

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 still active (it is not undergoing an implicit or explicit logout).  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 both establish a new allegiance for the command to be aborted, and abort it as well (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). For the ABORT TASK function, the target MUST NOT deliver additional responses after sending the task management response. In case both responses were delivered, whether the initiator should deliver task responses before delivering the task management response or not while an ABORT TASK is executing is a matter of implementation.

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 be also subject to SCSI access controls for the requesting initiator. When not implemented or when authorization fails at target, Target Reset functions should end as if the function was executed successfully and the response qualifier will detail what was executed.

For the TARGET WARM RESET and TARGET COLD RESET functions, the target cancels all pending operations and are both equivalent to the Target Reset function specified by [SAM2]. They can both affect many other initiators.

In addition, for the TARGET COLD RESET the target then MUST 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 8.1.

TASK REASSIGN MUST be issued as an immediate command.

### 3.5.2 LUN

This field is required for functions addressing a specific LU (ABORT TASK, CLEAR TASK SET, ABORT TASK SET, CLEAR ACA, LOGICAL UNIT RESET) and is reserved in all others.

### 3.5.3 Referenced Task Tag

Initiator Task Tag of the task to be aborted or reassigned.

### 3.5.4 RefCmdSN or ExpDataSN

For ABORT TASK the task CmdSN to enable task removal. 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 establishing 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.

Otherwise, this field is reserved.

## 3.6 Task Management Function Response

```
   Byte /    0       |       1       |       2       |       3       |
      /              |               |               |               |
     |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
     +---------------+---------------+---------------+---------------+
    0|1|1| 0x22      |1| Reserved    | Response      | Qualifier     |
     +---------------+---------------+---------------+---------------+
    4/ Reserved                                                      /
     /                                                               /
     +---------------+---------------+---------------+---------------+
   16| Initiator Task Tag                                            |
     +---------------+---------------+---------------+---------------+
   20| Referenced Task Tag or 0xffffffff                             |
     +---------------+---------------+---------------+---------------+
   24| StatSN                                                        |
     +---------------+---------------+---------------+---------------+
   28| ExpCmdSN                                                      |
     +---------------+---------------+---------------+---------------+
   32| MaxCmdSN                                                      |
     +---------------+---------------+---------------+---------------+
   36/ Reserved                                                      /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
   48
```

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

### 3.6.1 Response and Qualifier

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

```
           0 - Function Complete
           1 - Task specified in the Referenced Task Tag field was
           not in task set
           2 - LUN does not exist
           3 - Task still allegiant
           4 - Task failover not supported
      255   Function Rejected
```

All other values are reserved.

The Qualifier field provides additional information about the Response.

For a Response of "Function Complete" the valid Qualifiers are:

0 - Function Executed
1 - Not Authorized

For a discussion on usage of response codes 3 and 4, see section 8.1.2.

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.

### 3.6.2 Referenced Task Tag

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

## 3.7 SCSI Data-out & SCSI Data-in

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

```
   Byte /    0       |       1       |       2       |       3       |
      /              |               |               |               |
     |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
     +---------------+---------------+---------------+---------------+
    0|0|0| 0x05      |F| Reserved                                    |
     +---------------+---------------+---------------+---------------+
    4| Reserved      | DataSegmentLength                             |
     +---------------+---------------+---------------+---------------+
    8| LUN or Reserved                                               |
     +                                                               +
   12|                                                               |
     +---------------+---------------+---------------+---------------+
   16| Initiator Task Tag                                            |
     +---------------+---------------+---------------+---------------+
   20| Target Transfer Tag or 0xffffffff                             |
     +---------------+---------------+---------------+---------------+
   24| Reserved                                                      |
     +---------------+---------------+---------------+---------------+
   28| ExpStatSN                                                     |
     +---------------+---------------+---------------+---------------+
   32| Reserved                                                      |
     +---------------+---------------+---------------+---------------+
   36| DataSN                                                        |
     +---------------+---------------+---------------+---------------+
   40| Buffer Offset                                                 |
     +---------------+---------------+---------------+---------------+
   44| Reserved                                                      |
     +---------------+---------------+---------------+---------------+
   48| Digests if any...                                             |
     +---------------+---------------+---------------+---------------+
     / DataSegment                                                   /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
```

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

```
Byte /    0       |       1       |       2       |       3       |
   /              |               |               |               |
  |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
  +---------------+---------------+---------------+---------------+
 0|1|1| 0x25      |F|0 0 0 0|O|U|S| Reserved      |Status or Rsvd |
  +---------------+---------------+---------------+---------------+
 4| Reserved      | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| Reserved                                                      |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Residual Count                                                |
  +---------------+---------------+---------------+---------------+
24| StatSN or Reserved                                            |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| DataSN                                                        |
  +---------------+---------------+---------------+---------------+
40| Buffer Offset                                                 |
  +---------------+---------------+---------------+---------------+
44| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
48| Digests if any...                                             |
  +---------------+---------------+---------------+---------------+
  / DataSegment                                                   /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
```

Status can accompany the last Data-in PDU if the command did not end with an exception. 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.

3.7.1 F (Final) Bit

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

For incoming data, this bit is 1 for the last input (read) data PDU of a sequence. Input can be split in several sequences each one having it's own F bit. Splitting the data stream in 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 and/or end of recoverable sequences by targets with a limited retransmission buffer. A target that implements ErrorRecoveryLevel 1 or higher MUST use the F bit to indicate the end-of-sequence of Data-In PDUs is it is going to discard.

For Bidirectional operations, the F bit is 1 both for the end of the input sequences as well as the end of the output sequences.

### 3.7.2 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.

### 3.7.3 StatSN

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

### 3.7.4 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 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**32-1 numbered PDUs.

### 3.7.5 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)

### 3.7.6 DataSegmentLength

This is the data payload length of a SCSI Data-In or SCSI Data-Out PDU; 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 integer number of 4 byte words (real payload) unless the F bit is set to 1.

### 3.7.7 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 3-6 not used (should be set to 0)
   bit 1-2 as in an 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, Residual Count have meaningful content only if the S bit is set to 1.

## 3.8 Ready To Transfer (R2T)

```
   Byte /    0       |       1       |       2       |       3       |
      /              |               |               |               |
     |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
     +---------------+---------------+---------------+---------------+
    0|1|1| 0x31      |1| Reserved                                      |
     +---------------+---------------+---------------+---------------+
    4/ Reserved                                                      /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
   16| Initiator Task Tag                                            |
     +---------------+---------------+---------------+---------------+
   20| Target Transfer Tag                                           |
     +---------------+---------------+---------------+---------------+
   24| StatSN                                                        |
     +---------------+---------------+---------------+---------------+
   28| ExpCmdSN                                                      |
     +---------------+---------------+---------------+---------------+
   32| MaxCmdSN                                                      |
     +---------------+---------------+---------------+---------------+
   36| R2TSN                                                         |
     +---------------+---------------+---------------+---------------+
   40| Buffer Offset                                                 |
     +---------------+---------------+---------------+---------------+
   44| Desired Data Transfer Length                                  |
     +---------------------------------------------------------------+
   48
```

When an initiator has submitted a SCSI Command with data passing 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 to do so by sending the InitialR2T=no key-pair to each other, which happens 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, the last PDU SHOULD have the F bit set to 1. The Data-Out PDU ordering is governed by DataPDUInOrder. 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) and thus have a number of data transfers pending. Within a connection, outstanding R2Ts MUST be fulfilled by the initiator in the order in which they were received.

Buffer offset ordering in consecutive R2Ts is governed by DataSequenceInOrder. If DataSequenceInOrder is yes then consecutive R2Ts SHOULD refer to continuous non-overlapping ranges. However, even when DataSequenceInOrder is no, a target MAY send out-of-order R2Ts (e.g., for recovery) and an initiator MAY choose to terminate a command when receiving an out-of-order R2T that it can't fulfill, with an appropriate response after aborting the command at the target with the appropriate task management command.

### 3.8.1 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.

### 3.8.2 StatSN

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

### 3.8.3 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.

### 3.8.4 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 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).

### 3.9 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.

```
   Byte /    0       |       1       |       2       |       3       |
      /              |               |               |               |
     |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
     +---------------+---------------+---------------+---------------+
    0|1|1| 0x32      |1| Reserved                                    |
     +---------------+---------------+---------------+---------------+
    4| Reserved      | DataSegmentLength                             |
     +---------------+---------------+---------------+---------------+
    8| LUN                                                           |
     +                                                               +
   12|                                                               |
     +---------------+---------------+---------------+---------------+
   16/ Reserved                                                      /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
   24| StatSN                                                        |
     +---------------+---------------+---------------+---------------+
   28| ExpCmdSN                                                      |
     +---------------+---------------+---------------+---------------+
   32| MaxCmdSN                                                      |
     +---------------+---------------+---------------+---------------+
   36| AsyncEvent    | AsyncVCode    | Parameter1 or Reserved        |
     +---------------+---------------+---------------+---------------+
   40| Parameter2 or Reserved        | Parameter3 or Reserved        |
     +---------------+---------------+---------------+---------------+
   44| Reserved                                                      |
     +---------------+---------------+---------------+---------------+
   48| Digests if any...                                             |
     +---------------+---------------+---------------+---------------+
     / DataSegment - Sense Data or iSCSI Event Data                  /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
```

Some Asynchronous Messages are strictly related to iSCSI while others are related to SCSI [SAM2].

Please note that StatSN counts this PDU as an acknowledgeable event (StatSN is advanced), allowing initiator and target state synchronization.

### 3.9.1 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. 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 being requested to be logged out.  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" to cleanly shutdown the connection.  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 on what CID the connection will dropped.
The Parameter2 field indicates, in seconds, the minimum time to wait before attempting to reconnect.
Parameter3 indicates the maximum time to reconnect and/or restart commands after the initial wait (Parameter2).
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.
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 Parameter2 field indicates, in seconds, the minimum time to wait before attempting to reconnect.

The Parameter3 field indicates the maximum time to reconnect and restart commands after the initial wait (Parameter2).
If the initiator does not attempt to reconnect within the time specified by Parameter 3 or, if Parameter 3 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.

### 3.9.2 AsyncVCode

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

### 3.9.3 Sense Data or iSCSI Event Data

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

For an iSCSI Event additional data that MAY accompany the report

## 3.10 Text Request

The Text Request is provided to allow 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.

```
Byte /    0       |       1       |       2       |       3       |
   /              |               |               |               |
  |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
  +---------------+---------------+---------------+---------------+
 0|X|I| 0x04      |F| Reserved                                    |
  +---------------+---------------+---------------+---------------+
 4| Reserved      | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| Reserved                                                      |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| CmdSN                                                         |
  +---------------+---------------+---------------+---------------+
28| ExpStatSN                                                     |
  +---------------+---------------+---------------+---------------+
32/ Reserved                                                      /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
48| Digests if any...                                             |
  +---------------+---------------+---------------+---------------+
  / DataSegment (Text)                                            /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
```

### 3.10.1 F (Final) Bit

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

### 3.10.2 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).

### 3.10.3 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 bookmarks associated with the Initiator Task Tag.

When the target sets in a text response the Target Transfer Tag to a value other than the reserved value 0xffffffff it indicates that it has more data to send associated with the specified Initiator Task Tag (an internal bookmark). By copying the Target Transfer Tag from the response to the next Text Request, the initiator tells the target to continue from its last bookmark for the specific Initiator Task Tag.

This mechanism allows a target to transfer a large amount of textual data over a sequence of text-command/text-response exchanges.

A target MAY reject an old Bookmark.

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)
```

### 3.10.4 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). The key and value are separated by a '=' (0x3d) delimiter. Every key=value pair (including the last or only pair) MUST be followed by at least one null (0x00) delimiter.  A list is a set of values separated by comma (0x2c).

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).  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 specified otherwise 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 4096 bytes.

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

Some basic key=value pairs are described in Appendix D. All keys in Appendix D, 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
```

Any other key not understood by the target may be ignored by the target without affecting basic function. However the Text Response for a key that was not understood MUST be key=NotUnderstood.

Text operations are usually meant for parameter setting/negotiations but can be used also to perform some long lasting operations.

Text operations that will take a long time should be placed in their own Text request.

### 3.11 Text Response

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

```
Byte /    0       |       1       |       2       |       3       |
   /              |               |               |               |
  |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
  +---------------+---------------+---------------+---------------+
 0|1|1| 0x24      |F| Reserved                                    |
  +---------------+---------------+---------------+---------------+
 4| Reserved      | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| Reserved                                                      |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36/ Reserved                                                      /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
48| Digests if any...                                             |
  +---------------+---------------+---------------+---------------+
  / DataSegment (Text)                                            /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
```

#### 3.11.1 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 a F bit set to 1 MUST NOT contain key=value pairs that may require additional answers from the initiator.

### 3.11.2 Initiator Task Tag

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

### 3.11.3 Target Transfer Tag

When a target can't transfer all the remaining text data in a single Text response, it attempts to set an internal bookmark. If successful, the target sets the Target Transfer Tag to the bookmark value and associates the bookmark with the Initiator Task Tag.

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 Task Tag set to the reserved value of 0xffffffff it resets the internal bookmark associated with the given Initiator Task Tag.

When a target can't transfer all the text data in a single text response and it cannot set an internal bookmark it rejects the Text request with an appropriate Reject code. A target may reset its internal bookmark(s) after some time in order to reclaim resources associated with the bookmark and reject subsequent Text requests with the Target Transfer Tag set to a bookmark value.

When all the text data yet to be sent fits in a single Text response the Target Transfer Tag of the response is set to 0xffffffff and the internal bookmark associated with the Initiator Task Tag is reset.

### 3.11.4 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 coding constraints. Appendix A and Appendix D lists some basic Text requests and their Responses.

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 an identical key=value pair offered by the initiator.

A Key=value pair must be confined to a given text response even in the presence of bookmark - i.e., it must start and end within one Text Response.

### 3.12 Login Request

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

The Login Phase (see chapter 5) consists of a sequence of Login requests and responses that carry the same Initiator Task Tag.

```
Byte /    0       |       1       |       2       |       3       |
    /             |               |               |               |
   |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
   +---------------+---------------+---------------+---------------+
  0|X|I| 0x03      |T|0 0 0|CSG|NSG| Version-max   | Version-min   |
   +---------------+---------------+---------------+---------------+
  4| Reserved      | DataSegmentLength                             |
   +---------------+---------------+---------------+---------------+
  8| CID                           | Reserved                      |
   +---------------+---------------+---------------+---------------+
 12| ISID                          |TSID                           |
   +---------------+---------------+---------------+---------------+
 16| Initiator Task Tag                                            |
   +---------------+---------------+---------------+---------------+
 20| Reserved                                                      |
   +---------------+---------------+---------------+---------------+
 24| CmdSN                                                         |
   +---------------+---------------+---------------+---------------+
 28| ExpStatSN   or   Reserved                                     |
   +---------------+---------------+---------------+---------------+
 32| Reserved                                                      |
   +---------------+---------------+---------------+---------------+
 36| Reserved                                                      |
   +---------------+---------------+---------------+---------------+
 40/ Reserved                                                      /
  +/                                                               /
   +---------------+---------------+---------------+---------------+
 48/ DataSegment - Login Parameters in Text request Format         /
  +/                                                               /
   +---------------+---------------+---------------+---------------+
```

#### 3.12.1 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.

If TSID is not 0 then this is a connection restart. 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 not supporting 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.

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

### 3.12.2 I - Immediate

Login MUST be issued as an immediate request (I=1).

### 3.12.3 T (Transit) Bit

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

If the T bit is set to 1 and NSG is FullFeaturePhase then this is also indicating that the initiator is ready for the Final Login Response (see chapter 5).

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

### 3.12.4 CSG and NSG

Through these fields, called Current Stage (CSG) and Next Stage (NSG), the Login negotiation commands and responses are associated with a specific stage in the session (SecurityNegotiation, LoginOperationalNegotiation, FullPhaseOperationalNegotiations) and may indicate the next stage they want to move to (see chapter 5).

The next stage value is valid only when the T bit is 1 and is reserved otherwise.

The stage codes are:

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

### 3.12.5 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.

### 3.12.6 Version-min

Minimum Version supported.

The version number of the current draft is 0x2.

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.

### 3.12.7 Connection ID - CID

This is 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.

### 3.12.8 ISID

This is an initiator-defined session-identifier. It MUST be the same for all connections within a session. A SCSI initiator port is uniquely identified by the value pair (InitiatorName, ISID).

When a target detects an attempt to open a new session by the same SCSI initiator port (same InitiatorName and ISID) to the same target portal group it MUST close the old session and establish a new session.

All Login requests within the Login phase MUST carry the same ISID.

The target MUST use the value presented with the login request with C=0.

### 3.12.9 TSID

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.

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.

### 3.12.10 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.

### 3.12.11 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).

### 3.12.12 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. The format of the parameters is as specified for the Text request. Keys and their explanations are listed in the Appendix A (security negotiation keys) and Appendix D (operational parameter negotiation keys). All keys in Appendix D, except for the X- extension format, MUST be supported by iSCSI initiators and targets. Keys in Appendix A MUST be supported only when the function they refer to is mandatory to implement.

## 3.13 Login Response

The Login Response indicates the progress and/or end of the login phase. Note that after security is established, the login response is authenticated.

```
Byte /    0       |       1       |       2       |       3       |
   /              |               |               |               |
  |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
  +---------------+---------------+---------------+---------------+
 0|1|1| 0x23      |T|0 0 0|CSG|NSG| Version-max   | Version-active|
  +---------------+---------------+---------------+---------------+
 4| Reserved      | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
12| ISID                          |TSID                           |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| Status-Class  | Status-Detail | Reserved                      |
  +---------------+---------------+---------------+---------------+
40/ Reserved                                                      /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
48| Digests if any...                                             |
  +---------------+---------------+---------------+---------------+
  / DataSegment - Login Parameters in Text request Format         /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
```

### 3.13.1 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 response to the first login request.

### 3.13.2 Version-active

Indicates the version supported (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 response to the first login request.

### 3.13.3 TSID

The TSID is a tag set by the target that together with the ISID identifies a unique session with the initiator.
It MUST be valid only in the final response.

### 3.13.4 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 will carry this number + 1). This field is valid only if the Status Class is 0.

### 3.13.5 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

A 0 Status-Class indicates success.

A non-zero Status-Class indicates 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.

For a non-zero Status-Class the T, CSG & NSG  fields MUST be 0.

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 further action must be taken by the initiator 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 - indicates that the initiator 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 | Description |
|---|---|---|
| | (hex) | |
| Success | 0000 | Login is proceeding OK (*1) |
| Target Moved Temporarily | 0101 | The requested ITN has moved temporarily to the address provided. |
| Target Moved Permanently | 0102 | The requested ITN has moved permanently 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 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 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.

### 3.13.6 T (Transit) bit

T bit is set to 1 as an indicator of end of stage. If the T bit is set to 1 and NSG is FullFeaturePhase then this is also the Final Login Response (see chapter 5). A T bit of 0 indicates a "partial" response, which means "more negotiation needed".

A login response with a T bit set to 1 MUST NOT contain key=value pairs that may require additional answers from the initiator within the same stage.

If the status class is 0, the T bit MUST NOT be set to 1 if the T bit in the request was set to 0.

## 3.14 Logout Command

The Logout command 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 except SNACK and task management commands required for recovery.

After receiving the Logout command the target aborts all pending commands on that connection/session if the logout reason code is "close the connection", " or "close the session" and suspends all data/status/R2T transfers on behalf of pending commands if the reason code is "remove connection for recovery". 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 after that.

Note that a Logout for a CID may be performed on a different transport connection when the TCP connection for the CID had 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.

```
   Byte /    0       |       1       |       2       |       3       |
      /              |               |               |               |
     |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
     +---------------+---------------+---------------+---------------+
    0|0|I| 0x06      |1| Reserved                                    |
     +---------------+---------------+---------------+---------------+
    4| Reserved                                                      |
     +---------------+---------------+---------------+---------------+
    8| CID or Reserved               | Reserved      |Reason Code    |
     +---------------+---------------+---------------+---------------+
   12| Reserved                                                      |
     +---------------+---------------+---------------+---------------+
   16| Initiator Task Tag                                            |
     +---------------+---------------+---------------+---------------+
   20| Reserved                                                      |
     +---------------+---------------+---------------+---------------+
   24| CmdSN                                                         |
     +---------------+---------------+---------------+---------------+
   28| ExpStatSN                                                     |
     +---------------+---------------+---------------+---------------+
   32/ Reserved                                                      /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
   48
```

### 3.14.1 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 session".

### 3.14.2 ExpStatSN

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

### 3.14.3 Reason Code

Indicate the reason for Logout:

0 - closes the session - the session is closed - all commands associated with the session (if any) are aborted
1 - closes the connection - the connection is closed - 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

### 3.15 Logout Response

The logout response is used by the target to indicate that the cleanup operation for the connection has completed.

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

```
Byte /    0          |       1         |       2         |       3         |
   /                 |                 |                 |                 |
  |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
  +---------------+---------------+---------------+---------------+
 0|1|1| 0x26      |1| Reserved    | Response      | Reserved      |
  +---------------+---------------+---------------+---------------+
 4/ Reserved                                                      /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag                                            |
  +---------------+---------------+---------------+---------------+
20| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36| Reserved                                                      |
  +---------------------------------------------------------------+
40| Time2Wait                     | Time2Retain                   |
  +---------------+---------------+---------------+---------------+
44| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
48
```

#### 3.15.1 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

### 3.15.2 Time2Wait

Minimum time in seconds to wait before Login for adding or reinstating a new connection to this session on this target.

### 3.15.3 Time2Retain

Maximum time to wait for a Login that associates non acknowledged status to a new connection. After this time the status is discarded as acknowledged by hiatus.

### 3.16 SNACK Request

```
Byte /    0       |       1       |       2       |       3       |
   /              |               |               |               |
  |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
  +---------------+---------------+---------------+---------------+
 0|0|1| 0x10      |1|Rsrvd| Type  | Reserved                      |
  +---------------+---------------+---------------+---------------+
 4/ Reserved                                                      /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag or 0xffffffff                              |
  +---------------+---------------+---------------+---------------+
20| BegRun                                                        |
  +---------------+---------------+---------------+---------------+
24| RunLength                                                     |
  +---------------+---------------+---------------+---------------+
28| ExpStatSN                                                     |
  +---------------+---------------+---------------+---------------+
32| Reserved                                                      |
  +---------------+---------------+---------------+---------------+
36| ExpDataSN or Reserved                                         |
  +---------------+---------------+---------------+---------------+
32/ Reserved                                                      /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
48
```

Support for SNACK is optional.

SNACK request is used to request retransmission of numbered-responses, data or R2T PDUs from the target. The SNACK request indicates to the target the missed numbered-response or data run, where the run is composed of an initial missed StatSN, DataSN or R2TSN and the number of additional missed Status, Data or R2T PDUs (0 means only the initial).

The numbered-response, Data or R2T PDUs requested by a SNACK have to be delivered as exact replicas of the ones the initiator missed including all its flags.

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

### 3.16.1 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

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.

For a Status SNACK the ExpDataSN field is reserved.

An iSCSI target that does not support recovery within connection MAY discard status SNACK. If the target supports command recovery within session it MAY discard the SNACK after which it MUST issue an Asynchronous Message PDU with an iSCSI event indicating "Request Logout".

### 3.16.2 BegRun

First missed DataSN, R2TSN or StatSN

### 3.16.3 RunLength

RunLength is the number of sequential missed DataSN, R2TSN or StatSN. RunLength 0 signals that all Data-In, R2T or Response PDUs carrying numbers equal or greater to BegRun have to be resent.

## 3.17 Reject

```
   Byte /    0          |       1         |       2         |       3         |
       /                |                 |                 |                 |
      |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
      +---------------+---------------+---------------+---------------+
     0|1|1| 0x3f      |1| Reserved    | Reason        |               |
      +---------------+---------------+---------------+---------------+
     4| Reserved      | DataSegmentLength                             |
      +---------------+---------------+---------------+---------------+
     8/ Reserved                                                      /
     +/                                                               /
      +---------------+---------------+---------------+---------------+
    24| StatSN                                                        |
      +---------------+---------------+---------------+---------------+
    28| ExpCmdSN                                                      |
      +---------------+---------------+---------------+---------------+
    32| MaxCmdSN                                                      |
      +---------------+---------------+---------------+---------------+
    36| DataSN or Reserved                                            |
      +---------------+---------------+---------------+---------------+
    40| Reserved                                                      |
      +---------------+---------------+---------------+---------------+
    44| Reserved                                                      |
      +---------------+---------------+---------------+---------------+
    48| Digests if any...                                             |
      +---------------+---------------+---------------+---------------+
    xx/ Complete Header of Bad PDU                                    /
     +/                                                               /
      +---------------+---------------+---------------+---------------+
    yy
```

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

### 3.17.1 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 SNACK | no |
| 0x09 | Bookmark Reject - No bookmark for this Initiator Task Tag | no |
| 0x0a | Bookmark Reject - Can't generate bookmark - out of resources | yes |
| 0x0b | Negotiation Reset | no |

Note 1: For iSCSI data PDUS, this is done only if target requests retransmission with a recovery R2T. However, if this is the data digest error on immediate data, no signal from the target is necessary for PDU retransmission if desired so by the initiator.

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 3.4.3. 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 Set) the target MUST wait until it receives the Data-out PDU with the F bit set before sending the Response PDU.

For additional usage semantics of Reject PDU, please see section 1.1.

### 3.17.2 DataSN

This field is valid only if the Reason code is "Invalid SNACK" and the SNACK was a data SNACK. The DataSN is the last sequence number that the target sent for the task.

### 3.17.3 Complete Header of Bad PDU

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

## 3.18 NOP-Out

```
   Byte /    0       |       1       |       2       |       3       |
      /              |               |               |               |
     |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
     +---------------+---------------+---------------+---------------+
    0|X|I| 0x00      |1| Reserved                                    |
     +---------------+---------------+---------------+---------------+
    4| Reserved      | DataSegmentLength                             |
     +---------------+---------------+---------------+---------------+
    8| LUN or Reserved                                               |
     +                                                               +
   12|                                                               |
     +---------------+---------------+---------------+---------------+
   16| Initiator Task Tag or 0xffffffff                              |
     +---------------+---------------+---------------+---------------+
   20| Target Transfer Tag or 0xffffffff                             |
     +---------------+---------------+---------------+---------------+
   24| CmdSN                                                         |
     +---------------+---------------+---------------+---------------+
   28| ExpStatSN                                                     |
     +---------------+---------------+---------------+---------------+
   32/ Reserved                                                      /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
   48| Digests if any...                                             |
     +---------------+---------------+---------------+---------------+
     / DataSegment - Ping Data (optional)                            /
    +/                                                               /
     +---------------+---------------+---------------+---------------+
```

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 there is no other PDU to carry it for a long time.

When used as a ping command, the Initiator Task Tag MUST be set to 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 NOP-In Target Task 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).

### 3.18.1 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 as usual the next CmdSN but CmdSN is not advanced and the I bit must be set to 1.

### 3.18.2 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 which case it copies the Target Transfer Tag from the NOP-In PDU.

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

### 3.18.3 Ping Data

Ping data is reflected in the NOP-In Response. Note that the length of the reflected data is limited to 4096 bytes and the initiator should avoid sending more than 4096 bytes. 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.

### 3.19 NOP-In

```
Byte /    0       |       1       |       2       |       3       |
   /              |               |               |               |
  |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
  +---------------+---------------+---------------+---------------+
 0|1|1| 0x20      |1| Reserved                                    |
  +---------------+---------------+---------------+---------------+
 4| Reserved      | DataSegmentLength                             |
  +---------------+---------------+---------------+---------------+
 8| LUN or Reserved                                               |
  +                                                               +
12|                                                               |
  +---------------+---------------+---------------+---------------+
16| Initiator Task Tag or 0xffffffff                              |
  +---------------+---------------+---------------+---------------+
20| Target Transfer Tag or 0xffffffff                             |
  +---------------+---------------+---------------+---------------+
24| StatSN                                                        |
  +---------------+---------------+---------------+---------------+
28| ExpCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
32| MaxCmdSN                                                      |
  +---------------+---------------+---------------+---------------+
36/ Reserved                                                      /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
48| Digests if any...                                             |
  +---------------+---------------+---------------+---------------+
  / DataSegment - Return Ping Data                                /
 +/                                                               /
  +---------------+---------------+---------------+---------------+
```

NOP-In is either sent by a target as a response to a NOP-Out, as a "ping" to an initiator, or a means to carry a changed ExpCmdSN and/or MaxCmdSN if there is no other PDU to carry them for a long time.

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 first 4096 bytes of the initiator provided Ping Data.  For such a response, the Target Transfer Tag MUST be 0xffffffff.

### 3.19.1 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 not issued in response to a NOP-Out the StatSN field will contain as usual the next StatSN but StatSN for this connection is not advanced.

### 3.19.2 LUN

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

## 4. SCSI Mode Parameters for iSCSI

There are no iSCSI specific mode pages.

# 5. Login Phase

In the rest of this chapter, whenever we mention security we mean security and/or data integrity.

The login phase establishes an iSCSI session between initiator and 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 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 MUST include the InitiatorName and SessionType key=value pairs. If the SessionType is not "discovery" then the initial 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.

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 10 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 and data integrity parameters. Once this negotiation is completed, the channel is considered secure.

Most of the negotiation keys are allowed only in a specific stage - the SecurityNegotiation keys appear all in Appendix A while the LoginOperationalNegotiation keys appear in Appendix D.
Only a limited set of keys (marked as Declarative in Appendix D) may be used in any of the 2 stages.

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

Stage transition is performed through a command exchange (request/response) carrying the T bit and the same current stage code. During this exchange, the next stage selected is the lower of the two next stage codes. The initiator can request a transition whenever it is ready but a target can respond with a transition only after it is offered one 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 having T bit set to 1 in one pair and being answered with an T bit setting of 0 may issue the next request with T bit set to 0.

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

| From To -><br>\|<br>V | 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.

## 5.1 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 (currently 0x'02')
-Session and connection Ids
-The negotiation stage that the initiator is ready to enter

Optionally the login request may include:

-Security/Integrity 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 session to terminate. The T bit of the response MUST be set to 1 and the CSG and NSG are ignored
-Login Response with Login Accept as a final response (T bit set to 1 and the NSG in both command a 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 (depending on the current stage).
-Login Response with Login Accept as a partial response (T bit set to 0 or NSG not set to FullFeaturePhase in both request and response) indicating the start of a negotiation sequence.  The response includes the protocol version supported by the target and either security/integrity parameters or iSCSI parameters (when no security/integrity 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 indicating that it is unwilling to accept the connection without SecurityNegotiation and terminate the connection.

If the initiator is willing to negotiate security but it 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 LoginOperationalNegotiation the Login response is empty and has T bit set to 1, the CSG set to LoginOperationalNegotiation and NSG set to FullFeaturePhase in the next stage.

The iSCSI Initiator Name MUST be sent in the login command for the first connection of a session. The iSCSI Target Name MUST also be sent in the login command for the first connection of a session ONLY if the session type is normal (i.e., a target can be accessed without a Target Name only if the session type is a discovery session). If sent on new connections within an existing session the iSCSI Target Name and the iSCSI Initiator Name MUST be the same as the one used for the leading connection.

The iSCSI Names MUST be in text request format.

## 5.2 iSCSI Security and Integrity Negotiation

The security exchange sets the security mechanism and authenticates the user and the target to each other. The exchange proceeds according to the algorithms that were chosen in the negotiation phase and is conducted by the login requests and responses key=value parameters.

The negotiable security mechanisms include the following modes:

-Initiator-target authentication - the host and the target authenticate themselves to each other. A negotiable algorithm such as Kerberos provides this feature.
-PDU integrity - an integrity/authentication digest is attached to each packet. The algorithm is negotiable.

Using IPsec for encryption or authentication may eliminate part of the security negotiation at the iSCSI level but not necessarily all.

If security is established in the login phase then after the security negotiation is complete, each iSCSI PDU MUST include the appropriate digest field if any.

An initiator directed negotiation proceeds as follows:

-The initiator sends a login request with an ordered list of the options it supports for each subject (authentication algorithm, iSCSI parameters and so on). The options are listed in the initiator's order of preference.
-The target MUST reply with the first option in the list it supports and is allowed for the specific initiator.  The parameters are encoded in UTF8 as key=value.  The initiator MAY also send proprietary options. The "none" option, if allowed, MUST be included in the list, which indicates that no algorithm is supported by the target. For a list of security parameters see Appendix A.

-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 "lower" of the two. If the next stage is FullFeaturePhase, the target MUST respond with a Login Response with the Session ID and the protocol version.

All PDUs sent after the security negotiation stage MUST be built using the agreed security.

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 shall drop 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).

## 5.3 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/integrity negotiation if the initiator and target perform such a negotiation

An operational parameter negotiation on a connection MUST NOT start before the security negotiation if a security negotiation exists.

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 must keep sending the Login request (even empty) with the T bit set to 1 until it gets the Login Response with the T bit set to 1.

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

Session specific parameters can be specified only during the login phase begun by a login command containing a null TSID (e.g., the maximum number of connections that can be used for this session). Connection specific parameters, if any, can be specified during the login phase begun by any login command. Thus, a session is operational once it has at least one connection.

For a list of operational parameters, see Appendix D.

# 6. 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 always started and terminated by the initiator. The initiator MUST indicate its intent to terminate the negotiation by setting the F bit to 1; the target sets the F bit to 1 on the last response.

If the target responds to a text request with the F bit set to 1, with a text response with the F bit set to 0, the initiator must keep sending the text request (even empty) with the F bit set to 1 until it gets 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) is not an error but is discouraged.

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 having 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 they depend on; if they are sent within the same command a response for a parameter might imply responses for others.

An initiator MAY reset an operational parameter negotiation by issuing an ABORT TASK Task management request. A target may reset an operational parameter negotiation by answering a Text request with a Reject.

## 7. State transitions

An iSCSI connection and an iSCSI session go through several well-defined states from the time the connection and the session are created to the time they are cleared.

An iSCSI connection is a transport connection that is used for carrying out iSCSI activity. The connection state transitions are described in two separate but dependent state diagrams for ease of understanding. The first of these two is called a "standard connection state diagram" and it describes the connection state transitions when the iSCSI connection is not in connection recovery mode. The second diagram is called a "connection recovery state diagram" which describes the connection state transitions while performing connection recovery.

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 are participating in the session.

### 7.1 Standard connection state diagram

Symbolic names for States:

```
   S1:  FREE
   S2:  XPT_WAIT (illegal for target)
   S3:  XPT_UP
   S4:  LOGIN_SENT (initiator)/LOGIN_RCVD (target)
   S5:  FAILED
   S6:  EXITING
   S7:  LOGGED_IN (full-feature phase)
   S8:  LOGO_SENT (initiator)/LOGO_RCVD(target)
   S9:  LOGGED_OUT
   S10: ASYNC_MSG_SENT (target)/ ASYNC_MSG_RCVD(initiator)
   S11: LOGO_FAILED
   S12: XPT_CLEANUP
   S13: RECOVERY_START
```

Due to the number of states and the transitions involved in the description, the standard connection state diagram is defined using only a state transition table.  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 T4 transition to arrive at state S3).  Transitions that take place because of the same set of events, and which arrive into the same end state (from different starting states), share the same transition number, but are given different suffixes.  The fields marked "-" correspond to undefined transitions.

| | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 | S11 | S12 | S13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S1 | - | T1 | T4 | - | - | - | - | - | - | - | - | - | - |
| S2 | T3 | - | T2 | - | - | - | - | - | - | - | - | - | - |
| S3 | T21-1 | - | - | T5 | - | T9-1 | - | - | - | - | - | - | - |
| S4 | T21-2 | - | T8 | - | T7 | T9-2 | T6 | - | - | - | - | - | - |
| S5 | T21-3 | - | - | - | - | T9-3 | - | - | - | - | - | - | - |
| S6 | T10 | - | - | - | - | - | - | - | - | - | - | - | - |
| S7 | - | - | - | - | - | - | - | T11-1 | - | T13-1 | - | T20-1 | T19-1 |
| S8 | - | - | - | - | - | - | - | T15 | T12 | - | T16 | T20-2 | T19-2 |
| S9 | T21-4 | - | - | - | - | T9-4 | - | - | - | - | - | - | - |
| S10 | - | - | - | - | - | - | - | T11-2 | - | T13-2 | - | T20-3 | T19-3 |
| S11 | - | - | - | - | - | - | - | - | - | - | - | T17 | T19-4 |
| S12 | - | - | - | - | - | - | - | - | - | - | - | - | T18 |
| S13 | - | - | - | - | - | - | - | - | - | - | - | - | - |

State transition descriptions:

T1: Transport connect request was made (ex: TCP SYN sent). (Initiator only)
T2: Transport connection is established. (Initiator only)
T3: Transport connection request had timed out or failed. (Initiator only)
T4: Transport connection is established. (Target only)
T5: iSCSI login was sent by the initiator (or was received for a target).
T6: A login success was received/sent
T7: A login redirection/initiator error/target error was received, or login timed out. (Initiator only)
T8: A login redirection/initiator error/target error was sent. (Target only)
T9-1, T9-2, T9-3, T9-4: Transport disconnect request was sent/indication received (ex: TCP FIN received/sent).
T10: Both sides closed the transport connection.
T11-1, T11-2: Logout was sent by the initiator (or was received for a target).
T12: Logout Response (success) was received by the initiator (or sent by the target)
T13-1, T13-2: Async PDU with iSCSI event 2 received by the initiator (or sent by the target)
T15: Async PDU with iSCSI event 2 received (initiator only)
T16: Logout Response (failure) was received by the initiator (or sent by the target)
T17: Transport disconnect request was sent/indication received (ex: TCP FIN received/sent).
T18: Both sides closed the transport connection.
T19-1, T19-2, T19-3, T19-4: Transport connection deemed non-responsive by either end; or transport RESET received by either; or Async PDU with iSCSI event 3 (for this CID), or event 4 received by the initiator.
T20-1, T20-2, T20-3: Unexpected transport disconnect indication received by either side.
T21-1, T21-2, T21-3, T21-4: Transport connection deemed non-responsive by either end; or transport RESET received by either end.

The RECOVERY_START state (S13) implies that there are possibly iSCSI tasks that have not reached conclusion and are still considered busy.

## 7.2 Connection recovery state diagram

Symbolic names for states:

R1: RECOVERY_START (same as S13)

R2: IN_RECOVERY
R3: FREE (same as S1)

Whenever a connection state machine (say, CSM-R) enters the RECOVERY_START state (S13), it must go through the state transitions additionally described in the connection recovery state diagram. These additional state transitions may be traversed either by using a connection in the LOGGED_IN state with an explicit logout (let us call it CSM-E), or on a new transport connection in the FREE state with an implicit logout (let us call it CSM-I). This recovery state diagram hence is applicable only to the instance of the connection in recovery, i.e. CSM-R. In the case of an implicit logout for example, CSM-R reaches FREE at the time CSM-I reaches LOGGED_IN. In the case of an explicit logout, CSM-R reaches FREE when CSM-E receives a successful logout response while continuing to be in the LOGGED_IN state.

State diagram -

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

State transition table:

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

State transition descriptions:

M1: Connection state timeout happened on either side.

M2: An implicit /explicit logout was sent by the initiator (or received by the target)

- In CSM-I case, a recovery login was sent by the initiator (or received by the target) in state S1. [OR]

- In CSM-E case, a logout was sent by the initiator (or received by the target) in state S7.

M3: Logout failure detected

- CSM-I failed to reach S7, instead arrived into S1. [OR]
- CSM-E either moved out of S7/Logout timed out and/or aborted/Logout Response (failure) received by the initiator (or sent by the target).

M4: Successful implicit/explicit logout was performed.

- CSM-I reached state S7. [OR]
- CSM-E stayed in S7, and received Logout Response (success) by the initiator (or sent by the target).

## 7.3 Session state diagram

If any connection participating in a session is LOGGED_IN (S7), the session state is LOGGED_IN (Q3 below). The first connection entering into S7 and the last connection leaving S7 toggle the session state.

Symbolic Names for States:

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

State diagram:

```
                       -------
                      / Q1    \
                   +->\       /<-+
                  /    ---+---    \
                 /        |        \ N4
             N6 |         |N1       |
                |         |        /
                |         |       /
                |         |      /
                |         V     /
                |      ------- /
                | N7  / Q2    \
                |  +->\       /<-+
                |  |+--+----+-   |
                |  ||       |    |  N3
                |  ||N5     |N2  |
                |  ||       |    |
                |  ||       |    |
                |  ||       |    |
                |  |V       V    |
               -+--+--      -----+-
              / Q4    \ N5 / Q3    \
              \       /<---\       /
               -------      -------
```

State transition table:

| | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|
| Q1 | - | N1 | - | - |
| Q2 | N4 | - | N2 | N5 |
| Q3 | - | N3 | - | N5 |
| Q4 | N6 | N7 | - | - |

State transition descriptions:

N1: At least one transport connection was established for the session.

N2: At least one transport connection reached the LOGGED_IN state.
N3: Last LOGGED_IN connection had ceased to be LOGGED_IN.
N4: Last participating transport connection was dropped.
N5: Session failure (all connections reported RECOVERY_START, or recovery failed)
N6: Session state timeout happened on either side.
N7: Connection recovery attempt preserving session state - either via connection reinstatement or new CID addition.

# 8. 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 (and the corresponding appendix in more detail) describes a general model for recovery, in support of interoperability.  Compliant implementations need not match details of this model as presented, but the external behavior of such implementations must correspond to the externally observable characteristics model.

## 8.1 Retry and Reassign in Recovery

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

### 8.1.1 Usage of Retry

By setting the Retry bit (X-bit) in the iSCSI command PDU, initiator attempts to "plug" (what it thinks are) the discontinuities in CmdSN ordering on the target end.  These discontinuities may have been created because of discarded command PDUs due to digest errors or format errors.  It is illegal to set the X-bit for immediate commands.

Note that the X-bit MUST NOT be used for any 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 does not however constitute a requirement on initiators to use SNACK.

Initiators as part of plugging command sequence gaps described above may inadvertently issue retries for allegiant commands already in progress at times (i.e. targets did not see the discontinuities in

CmdSN ordering). Targets MUST reject such command PDUs with a reason code of "Command in progress". This reason code may be used by initiators to tune their command retransmission logic, identify inadvertent connection allegiance switching attempts, and to get an updated target view of the command. Targets MUST support the retry bit and the associated functionality described above.

When the X-bit is specified, the command PDU MUST carry the original Initiator Task Tag and the original operational attributes (ex. flags, function names, LUN, CDB etc.). In addition, it MUST hold the original CmdSN.

### 8.1.2 Allegiance Reassignment

By issuing a "task reassign" task management command (section 3.5.1), 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 being established for the command, associating 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 (must be set to zero if there had been 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, target MUST respond with a task management response code of "Task still allegiant".

## 8.2 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 seeing Rejects, if the Reject is not for the command itself. The non-command Rejects only have diagnostic value in logging the errors, and they may be used for retransmission decisions as well by the initiators.

It should also be noted that 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. Target MAY advance its ExpDataSN if it does not attempt to recover the lost data PDU.

## 8.3 Format Errors

Explicit violations of the PDU layout rules stated in this document are format errors. This when detected, usually indicates 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 reset all transport connections in the session immediately and escalate the format error to session recovery (section 8.11.4).

## 8.4 Digest Errors

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 an solicited/unsolicited iSCSI data (for immediate data in a command PDU, non-data PDU rule below applies) PDU, target MUST do one of the following:
  a) Request retransmission with an R2T. [OR]
  b) Terminate the task with a command response PDU with the reason "delivery subsystem failure" (section 3.4.3). If the target chooses to implement this, 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 command response PDU.

- 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.

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, initiator MUST do one of the following -
  a) Request the desired data PDU through SNACK. In its turn, the target MUST either reject the SNACK with a Reject PDU with a reason-code of "Data-SNACK Reject" in which case the target MUST terminate the command with an iSCSI command response reason of "SNACK rejected", or resend the data PDU. [OR]
  b) Abort the task and terminate the command with an error.

- If the discarded PDU is a response PDU, 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. [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.

### 8.18.1Sequence 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 implying 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. Initiator MUST address these implied digest errors as described in section 8.4. 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 8.4.

When an initiator receives an iSCSI status PDU with an out-of-order StatSN implying missing responses, it MUST address the one or more missing status PDUs as described in section 8.4. As a side effect of receiving the missing responses, the initiator may discover missing data PDUs. The initiator MUST NOT acknowledge the received responses until it has completed receiving all the data PDUs of a SCSI command.

## 8.6 SCSI Timeouts

An iSCSI initiator SHOULD 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 re-transmitting the unacknowledged command with the retry bit set, as described in section 8.1.

On a ULP timeout for a command that carried a CmdSN of n, if the ExpCmdSN is still less than (n+1) on ULP timeout, the iSCSI initiator MUST assume a session failure and take the appropriate actions as described in section 8.11.41.1.1.

## 8.7 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 both 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.
- A text request is 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 as the login process failure and the connection must be dropped.
- A failure in negotiation while in the full-feature phase MUST terminate the entire negotiation sequence possibly consisting of a series of text requests using 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.

## 8.8 Protocol Errors

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

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

## 8.9 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 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 or 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:

a) Attempt connection recovery within the session (section 8.11.3) [OR]
b) Logout the connection with the reason code "closes the connection" (section 3.14.3) and re-issue missing commands, and implicitly terminate all active commands. Note that this option requires support for the within-connection recovery class (section 8.11.2) [OR]
c) Perform session recovery (section 8.11.4).

Either side may choose to escalate to session recovery, and the other side MUST give precedence to it. 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.

## 8.10 Session Errors

If all the connections of a session fail and cannot be reestablished in a short time or if initiators detect protocol errors repeatedly, an initiator may choose to terminate a session and establish a new session. It terminates all outstanding requests with an appropriate response before initiating a new session. The target takes the following actions:

- Resets the TCP connections (closes the session).
- Aborts all Tasks in the task set for the corresponding initiator.

## 8.11 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 impacting 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 (like ACA) have to be considered.

Usage of within a connection and within a command recovery classes MUST NOT be attempted before the connection is in full feature phase.

### 8.11.1 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) a data digest error - to be dealt with as specified in section 8.4
  b) a sequence reception timeout (no data or partial-data-and-no-F-bit) - to be dealt with as specified in section 8.5

c) a header digest error which manifests as a sequence reception timeout, or a sequence error - to be dealt with as specified in section 8.5

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) a data digest error - to be dealt with as specified in section 8.4
b) a sequence reception timeout (no status) - to be dealt with as specified in section 8.5
c) a header digest error which manifests as a sequence reception timeout, or a sequence error - to be dealt with as specified in section 8.5

Please Note that an initiator SHOULD NOT request a missing R2T by a SNACK after a timeout to avoid a race; this action is better left to the target.

Both the timeout values to be used by the initiator and target are outside the scope of this document.

### 8.11.2 Recovery Within-connection

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

a) Requests not acknowledged for a long time. Requests are acknowledged explicitly through ExpCmdSN or implicitly by receiving data and/or status. The initiator MAY reissue non-acknowledged commands as specified in section 8.1.
b) Lost iSCSI numbered Response recognized by either receiving it with a data digest error or receiving a Response PDU with a higher StatSN than expected. In the first case, digest error handling is done as specified in section 8.4, and in the second case, sequence error handling is done as specified in section 8.5.

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

a) Status/Response not acknowledged for a long time. The target MAY issue a NOP-IN, with the P bit set to 1 or 0, which indicates in the StatSN field the next status number it is going to issue.  This helps the initiator detect missing StatSN and issue a SNACK-status.

Both the timeout values to be used by the initiator and target are outside the scope of this document.

### 8.11.3 Connection Recovery

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

a) TCP connection failure. The initiator MUST close the connection following which it 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 (section 3.5.1).

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

N.B. As an alternative to Logout and reissue commands, the initiator MAY instead reset the target and terminate all outstanding commands with a service response indicating Delivery Subsystem Failure. The initiator MUST perform one of the two actions.

b) Receiving an Asynchronous Message indicating one or all connections in a session had 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 -

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

### 8.11.4 Session Recovery

Session recovery is to 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 closing of all TCP connections, aborting at target all executing and queued tasks for the given initiator, terminating at initiator all outstanding SCSI commands with an appropriate SCSI service response and restarting a session on a new connection set (TCP connection establishment and login on all new connections). It is to be noted that neither Reserve-Release managed SCSI reservations ("Regular" reservations) nor Persistent SCSI reservations are affected by session failures. Regular SCSI means are to be used to handle these reservations.

### 8.12 Error Recovery Hierarchy

The error recovery classes and features described so far are organized into a hierarchy for ease of understanding and to limit the myriad of implementation possibilities, thus significantly contributing to highly interoperable implementations. The attributes of this hierarchy are:

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 picture represents the error recovery hierarchy.

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

The following table lists the error recovery capabilities expected of implementations supporting each error recovery level.

| ErrorRecoveryLevel | Associated Error recovery capabilities |
|---|---|
| 0 | Session recovery class (8.11.4) |
| 1 | Digest failure recovery (Note below) |
| 2 | Connection recovery class (8.11.3) |

Note: Digest failure recovery is comprised of two recovery classes - Within-Connection recovery class (8.11.2) and Within-Command recovery class (8.11.1).

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 |

# 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 was 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 2.5.3.

## 9.1 Multiple Network Adapters

The iSCSI protocol allows multiple connections, not all of which need 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 insure 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 iSCSI Name and ISID/TSID use

The designers of the iSCSI protocol envision there should be 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.  This 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 that are coordinated as a single iSCSI Node, it is expected that an entity representing the iSCSI Node should manage session identifier resources (to enforce both the TSID and ISID RULES). This assignment can be accomplished in a number of ways.  For iSCSI initiator devices, it is expected that operating systems will provide standard APIs that will inform these devices of the iSCSI Node Name and manage ISIDs for the purpose of coordination of sessions.

Recognizing that these APIs are not available today, the following are recommendations for separate session coordinators within a node ("Session coordinators" are the components, either hardware or software, that handle session semantics within Portal Groups).

Because all portal groups of one iSCSI Node (initiator or target) must share the same name, the iSCSI Name should be a configurable parameter to the session coordinators across all portal groups of that node. A session coordinator is responsible for presenting the iSCSI Name of the node at login and in response to certain SCSI inquiries.

For an iSCSI Initiator, the ISID values used by the session coordinators should be configurable parameters and the ISID namespace should be partitioned among the Initiator Portal Groups. This allows each Initiator Portal Group to act independently of other portal groups when selecting an ISID for a login; this facilitates enforcement of the ISID RULE (see 2.5.3) at the initiator.

For an iSCSI Target, the TSID values used by the session coordinators should be configurable parameters and the TSID namespace should be partitioned among the Target Portal Groups. This allows each Target Portal Group to act independently of other portal groups when selecting a TSID for a login response; this best facilitates the enforcement of the TSID RULE (see 2.5.3) at the target.

## 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 presence of errors. As iSCSI can have many commands in-flight between initiator and target iSCSI mandates support for ACA.

## 9.3 Task Management Commands and Immediate Delivery

A task management command may reach the target and, in the case immediate delivery was requested, be executed before all of the tasks it was meant to act upon have been delivered or even reached the target.

It is assumed that, while pending delivery from iSCSI to SCSI at the target, commands are kept in an iSCSI queue at both the initiator and the target and that the target queue contains both commands and "holes" (placeholders for commands not received yet).

The following general mechanism can be used to achieve the effect of ordered delivery for task management commands while enabling the "urgent" delivery that some of them imply and immediate execution of the task management commands. The mechanism manages discarding commands while they are in the iSCSI layer at the target and prevents these discarded commands from being delivered to the target's SCSI layer. The initiator must keep a record of these commands to determine which will not receive a response. The target does not generate a response to a command that is aborted while in the iSCSI layer. The "barrier list" described in the following sections is a list containing information relating to all task management commands marked for immediate delivery.

At the Initiator when a relevant task management command marked for immediate delivery is issued:

a) if ExpCmdSN is equal to CmdSN (there are no commands in the queue) skip to step c;
b) mark all pending commands with a CmdSN field between the current ExpCmdSN and the current CmdSN as candidates for cleanup and retain CmdSN of the task management command in a "barrier list";
c) send the task management command for immediate delivery to the target.

Note: for clarity, the barrier list contains "items" and the command queue contains "entries"

At initiator when updating ExpCmdSN:

a) if the "barrier list" is empty or ExpCmdSN is less than the CmdSN of the oldest item in the barrier list then skip to step d;
b) remove the oldest barrier list item, and remove and silently discard all entries marked for cleanup having a CmdSN field less than ExpCmdSN;
c) go to step a;
d) release all queued entries between the old and new ExpCmdSN from the queue. Note: Any entries that had been marked as a candidate for cleanup have now been delivered by the target to its SCSI layer. The SCSI layer will have to determine if they are to be aborted.

At the target when receiving a relevant task management command for immediate delivery:

a) if ExpCmdSN is equal to CmdSN skip to step c;

b) mark all pending entries (commands received and placeholders) with a CmdSN field between ExpCmdSN and the current CmdSN as candidates for cleanup and retain CmdSN in a "barrier list" including the referenced LUN (or an ALL marker);

c) send the task management command to SCSI for immediate execution.

At target when updating ExpCmdSN (releasing ordered commands to SCSI):

a) if the "barrier list" is empty or ExpCmdSN is less than the oldest item in the barrier list then skip to step d;

b) remove the oldest barrier list item and evaluate all queued entries that have a CmdSN field less than ExpCmdSN, removing and silently discarding each queued command that meets the criteria for cleanup candidacy (as specified by the task management function);

c) go to step a;

d) release all queued entries between the old and new ExpCmdSN from the queue.

Note that this scheme will withstand connection recovery.

The following table details the candidates for cleanup:

| No | Function | Candidacy Selection |
|---|---|---|
| 1 | Abort Task | The task that are identified by the Referenced Task Tag Field and initiator |
| 2 | Abort Task Set | All tasks associated with the specified LUN and initiator. |
| 3 | Clear ACA | No entries are marked for candidacy. |
| 4 | Clear Task Set | All tasks associated with the specified LUN and initiator. For all other initiators all tasks at LUN with no regard to order. |

## 9.4 Synch and steering layer and performance

Although a synch and steering layer is optional, an initiator/target without synch and steering working against a target/initiator demanding 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. 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 (pretending as another identity, message insertion, deletion, modification and replaying) and passive attacks (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 to be provided by an iSCSI implementation.

### 10.1 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 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 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].

### 10.2 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 and 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 minimal advantage as possible by inspecting the authentication phase PDUs. In this spirit, 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 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.  In addition, the CHAP authentication method (specified in Appendix A) is vulnerable to off-line dictionary attack. CHAP SHOULD NOT is used without additional protection in environments where this attack is a concern.  Underlying IPsec, encryption provides protection against this attack.

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

## 10.3 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 on using IPSec for iSCSI are given in [SEC-IPS].

### 10.3.1 Data Integrity and Authentication

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

An iSCSI compliant initiator or target MUST provide data integrity and authentication by implementing IPSec [RFC2401] with ESP [RFC2406] with the following iSCSI specific requirements:

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

The ESP anti-replay service MUST also be implemented.

### 10.3.2 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 [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).

The NULL encryption algorithm MUST also be implemented.

### 10.3.3 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].

When IPSec is used, each iSCSI TCP connection within an iSCSI session MUST be protected by a separate IKE Phase 2 SA.
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 (MUST NOT use the IP Subnet or IP Address Range formats).

IKE main mode with pre-shared key authentication method SHOULD NOT be used (the only practical usage under this configuration is a group pre-shared key which creates vulnerability to man-in-the-middle attack).

## 11. IANA Considerations

There will be a well-known port for iSCSI connections. This well-known port will be registered with IANA.

## 12. 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/proposedmodes/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 (Cam)

[Castagnoli93] G. Castagnoli, S. Braeuer and M. Herrman "Optimization of Cyclic Redundancy-Check Codes with 24 and 32 Parity Bits", IEEE Transact. on Communications, Vol. 41, No. 6, June 1993

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

[NDT] M. Bakke & team, draft-ietf-ips-iscsi-name-disc-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-01.txt
[SPC] NCITS.351:200, SCSI-3 Primary Commands (SPC)
[SPC3]T10/1416-D, SCSI-3 Primary Commands (SPC)

## 13. Author's 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. iSCSI Security and Integrity

## 01 Security Keys and Values

The parameters (keys) negotiated for security are:

- Digests (HeaderDigest, DataDigest)
- Authentication method (AuthMethod)

Digests enable checking end-to-end data integrity beyond the integrity checks provided by the link layers and covering the whole communication path including all elements that may change the network level PDUs like routers, switches, proxies, etc.

The following table lists cyclic integrity checksums that can be negotiated for the digests and MUST be implemented by every iSCSI initiator and target. Note that these digest options have only 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.

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 - start with byte 0 bit 0 continue byte 1 bit 0 etc. (Big Endian on bytes / Little Endian on bits)
- the CRC register is initialized with all 1s (equivalent to complementing the first 32 bits of the message)
- 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 and divided by G(x)- the generator polynomial - producing 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. ( whenever examples are given the value to be specified in examples follows the same rules of representation as the rest of this document)
- a receiver of a "good" segment (data or header) built using the generator 0x11edc6f41 will end-up having in the CRC register the value 0x1c2d19ed (this a register value and not a word as outlined in this draft)

Proprietary algorithms MAY also be negotiated for data authentication and integrity digests.

The following table details authentication 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, SPKM2 are defined in [RFC2025], SRP is defined in [RFC2945] and CHAP is defined in [RFC1994].

Initiator and target MUST implement SRP.

## 02 Authentication

The authentication exchange authenticates the initiator to the target, and optionally the target to the initiator.  Authentication is not mandatory and is distinct from the data integrity exchange.

The authentication methods to be used are KRB5, SPKM1, SPKM2, SRP, CHAP, or proprietary.

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 return an error. 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].

KRB_AP_REQ, KRB_AP_REP are large binary items.

For SPKM1,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 which may cause the peer to re-generate and resend his 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 (by [RFC2025]) to send SPKM-DEL token, it will, instead, send a Login "login reject" message and terminate the connection. If the initiator needs to send SPKM-DEL token, it will just abort the connection.

In what follows, 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 abort 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].

All the SPKM-* tokens are large binary items and their binary length (not the encoded length) MUST not exceed 2048 bytes.

For SRP [RFC2945], the initiator MUST use:

```
   SRP_U=<user> TargetAuth=yes   /* or TargetAuth=no */
```

The target MUST either return an error or reply with:

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

The initiator MUST continue with:

```
   SRP_A=<A>
```

The target MUST either return an error or reply with:

```
   SRP_B=<B>
```

The initiator MUST either abort or continue with:

```
   SRP_M=<M>
```

If the initiator authentication fails, the target MUST return an error. 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)>
```

Where U, N, g, s, A, B, M and H(A | M | K) are defined in [RFC2945]. U is a text string, N,g,s,A,B,M and H(A | M | K) are numbers.

For CHAP [RFC1994], the initiator MUST use:

```
   CHAP_A=<A1,A2...>
```

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

The target MUST either return an error 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 either with:

```
CHAP_N=<N> CHAP_R=<R>
```

or, if he requires target authentication, with:

```
CHAP_N=<N> CHAP_R=<R> CHAP_I=<I> CHAP_C=<C>
```

If the initiator authentication fails, the target MUST return an error. Otherwise, if the initiator required target authentication, the target MUST reply with

```
CHAP_N=<N> CHAP_R=<R>
```

Where N, (A,A1,A2), I, C, R are (correspondingly) the Name, Algorithm, Identifier, Challenge and Response as defined in [RFC1994]. N is a text string, A,A1,A2,I are numbers and C,R are large binaries encoded as hexadecimal strings.

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.

## 03 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
    HeaderDigest=CRC32C,none
    DataDigest=CRC32C,none
    AuthMethod=KRB5,SRP,none

T-> Login (CSG,NSG=0,1 T=0)
    HeaderDigest=CRC32C
    DataDigest=CRC32C
    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 proceeds.

From this point on, any Login command and each PDU thereafter MUST have CRC32C digests for the header and the data.

The initiator may proceed:

```
I-> Login (CSG,NSG=1,3 T=0)
    ... iSCSI Operational Parameters

T-> Login (CSG,NSG=1,3 T=0)
    ... 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 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 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
           HeaderDigest=CRC32C,none
           DataDigest=CRC32C,none
           AuthMethod=SRP,KRB5,none

       T-> Login-PR (CSG,NSG=0,1 T=0)
           HeaderDigest=CRC32C
           DataDigest=CRC32C
           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)

       From this point on, any Login command and each PDU thereafter
       MUST have CRC32C digests for the header and the data.

       I-> Login (CSG,NSG=1,3 T=0)
           ... iSCSI parameters

       T-> Login (CSG,NSG=1,3 T=0)
           ... iSCSI parameters

       . . .

       T-> Login (CSG,NSG=1,3 T=1)"login accept"
           TargetName=iqn.1999-07.com.acme.diskarray.sn.88
```

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
           HeaderDigest=CRC32C,none
           DataDigest=CRC32C, none
           AuthMethod=SPKM1,KRB5,none

       T-> Login (CSG,NSG=0,1 T=0)
```

```
         HeaderDigest=CRC32C
         DataDigest=CRC32C
         AuthMethod=SPKM1

     I-> Login (CSG,NSG=0,1 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=0,1 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)
```

From this point on, any Login command and each PDU thereafter MUST have CRC32C digests for the header and the data.

The initiator may proceed:

```
     I-> Login  (CSG,NSG=1,3 T=0) ... iSCSI parameters
     T-> Login  (CSG,NSG=1,3 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"
         TargetName=iqn.1999-07.com.acme.diskarray.sn.88
```

If the target authentication by the initiator is not successful, the initiator terminates the connection (without responding to the Login SPKM_REP_TI message).

If the initiator authentication by the target is not successful, the target responds with:

```
      T-> Login "login reject"

      instead of the Login "proceed and change stage" message, and
      terminates the connection.


   In the next example, the initiator and target authenticate each other
   via SPKM2:

      I-> Login (CSG,NSG=0,1 T=0)
          InitiatorName=iqn.1999-07.com.os.hostid.77
          TargetName=iqn.1999-07.com.acme.diskarray.sn.88
    HeaderDigest= CRC32C,none
          DataDigest=CRC32C,none
          AuthMethod=SPKM1,SPKM2

      T-> Login-PR (CSG,NSG=0,1 T=0)
          HeaderDigest=CRC32C
          DataDigest=CRC32C
          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)

      From this point on, any Login command and each PDU thereafter
      MUST have CRC32C digests for the header and the data.

      The initiator may proceed:

      I-> Login (CSG,NSG=1,3 T=0)
          ... iSCSI parameters

      T-> Login (CSG,NSG=1,3 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"
          TargetName=iqn.1999-07.com.acme.diskarray.sn.88
```

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
       HeaderDigest=CRC32C,none
       DataDigest=CRC32C,none
       AuthMethod=KRB5,SRP,none

   T-> Login-PR  (CSG,NSG=0,1 T=0)
       HeaderDigest=CRC32C
       DataDigest=CRC32C
       AuthMethod=SRP

   I-> Login (CSG,NSG=0,1 T=0)
       SRP_U=<user>
       TargetAuth=yes

   T-> Login (CSG,NSG=0,1 T=0)
       SRP_N=<N>
       SRP_g=<g>
       SRP_s=<s>

   I-> Login (CSG,NSG=0,1 T=0)
       SRP_A=<A>

   T-> Login (CSG,NSG=0,1 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.

From this point on, any Login command and each PDU thereafter MUST have CRC32C digests for the header and the data.

```
I-> Login (CSG,NSG=1,3 T=0)
    ... iSCSI parameters

T-> Login (CSG,NSG=1,3 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"
    TargetName=iqn.1999-07.com.acme.diskarray.sn.88
```

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
    HeaderDigest=CRC32C,none
    DataDigest=CRC32C,none
    AuthMethod=KRB5,SRP,none

T-> Login-PR (CSG,NSG=0,1 T=0)
    HeaderDigest=CRC32C
    DataDigest=CRC32C
    AuthMethod=SRP

I-> Login (CSG,NSG=0,1 T=0)
    SRP_U=<user>
    TargetAuth=no
```

```
      T-> Login (CSG,NSG=0,1 T=0)
          SRP_N=<N>
          SRP_g=<g>
          SRP_s=<s>

      I-> Login (CSG,NSG=0,1 T=0)
          SRP_A=<A>

      T-> Login (CSG,NSG=0,1 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)
```

From this point on, any Login command and each PDU thereafter MUST have CRC32C digests for the header and the data.

```
      I-> Login (CSG,NSG=1,3 T=0)
          ... iSCSI parameters

      T-> Login (CSG,NSG=1,3 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"
          TargetName=iqn.1999-07.com.acme.diskarray.sn.88
```

In the next example the initiator and target authenticate each other 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
          HeaderDigest=CRC32C,none
          DataDigest=CRC32C,none
```

```
    AuthMethod=KRB5,CHAP,none

T-> Login-PR (CSG,NSG=0,1 T=0)
    HeaderDigest=CRC32C
    DataDigest=CRC32C
    AuthMethod=CHAP

I-> Login (CSG,NSG=0,1 T=0)
    CHAP_A=<A1,A2>

T-> Login (CSG,NSG=0,1 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.

From this point on, any Login command and each PDU thereafter MUST have CRC32C digests for the header and the data.

```
I-> Login (CSG,NSG=1,3 T=0)
    ... iSCSI parameters
T-> Login (CSG,NSG=1,3 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"
    TargetName=iqn.1999-07.com.acme.diskarray.sn.88
```

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
    HeaderDigest=CRC32C,none
    DataDigest=CRC32C,none
    AuthMethod=KRB5,CHAP,none

T-> Login-PR (CSG,NSG=0,1 T=0)
    HeaderDigest=CRC32C
    DataDigest=CRC32C
    AuthMethod=CHAP

I-> Login (CSG,NSG=0,1 T=0)
    CHAP_A=<A1,A2>

T-> Login (CSG,NSG=0,1 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=1,3 T=0)
    ... iSCSI parameters
```

```
T-> Login (CSG,NSG=1,3 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"
    TargetName=iqn.1999-07.com.acme.diskarray.sn.88
```

In the next example, the initiator does not offer any security/integrity parameters, so it may offer iSCSI parameters on the Login PDU with the F 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"
    TargetName=iqn.1999-07.com.acme.diskarray.sn.88
    ... ISCSI parameters
```

In the next example, the initiator does offer security/integrity 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
    HeaderDigest=CRC32C,none
    DataDigest=CRC32C,none
    AuthMethod:KRB5,SRP,none

T-> Login-PR (CSG,NSG=0,1 T=1)
    HeaderDigest=none
    DataDigest=none
    AuthMethod=none

I-> Login (CSG,NSG=1,3 T=0)
    ... iSCSI parameters

T-> Login (CSG,NSG=1,3 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"
            TargetName=iqn.1999-07.com.acme.diskarray.sn.88
```

## Appendix B. Examples

### 04 Read Operation Example

| Initiator Function | PDU Type | Target Function |
|---|---|---|
| Command request (read) | SCSI Command (READ)>>> | |
| | | Prepare Data Transfer |
| Receive Data | <<< SCSI Data-in | Send Data |
| Receive Data | <<< SCSI Data-in | Send Data |
| Receive Data | <<< SCSI Data-in | Send Data |
| | <<< SCSI Response | Send Status and Sense |
| Command Complete | | |

## 05 Write Operation Example

| Initiator Function | PDU Type | Target Function |
|---|---|---|
| Command request (write) | SCSI Command (WRITE)>>> | Receive command and queue it |
| | | Process old commands |
| | <<< R2T | Ready to process WRITE command |
| Send Data | SCSI Data-out >>> | Receive Data |
| | <<< R2T | Ready for data |
| | <<< R2T | Ready for data |
| Send Data | SCSI Data-out >>> | Receive Data |
| Send Data | SCSI Data-out >>> | Receive Data |
| | <<< SCSI Response | Send Status and Sense |
| Command Complete | | |

## 06 R2TSN/DataSN use examples

Output (write) data DataSN/R2TSN Example

| Initiator Function | PDU Type & Content | Target Function |
|---|---|---|
| Command request (write) | SCSI Command (WRITE)>>> | Receive command and queue it |
| | | Process old commands |
| | <<< R2T<br>R2TSN = 0 | Ready for data |
| | <<< R2T<br>R2TSN = 1 | Ready for more data |
| Send Data for R2TSN 0 | SCSI Data-out >>><br>DataSN = 0, F=0 | Receive Data |
| Send Data for R2TSN 0 | SCSI Data-out >>><br>DataSN = 1, F=1 | Receive Data |
| Send Data for R2TSN 1 | SCSI Data >>><br>DataSN = 0, F=1 | Receive Data |
| | <<< SCSI Response<br>ExpDataSN = 0 | Send Status and Sense |
| Command Complete | | |

Input (read) data DataSN Example

| Initiator Function | PDU Type | Target Function |
|---|---|---|
| Command request (read) | SCSI Command (READ)>>> | |
| | | Prepare Data Transfer |
| Receive Data | <<< SCSI Data-in DataSN = 0, F=0 | Send Data |
| Receive Data | <<< SCSI Data-in DataSN = 1, F=0 | Send Data |
| Receive Data | <<< SCSI Data-in DataSN = 2, F=1 | Send Data |
| | <<< SCSI Response ExpDataSN = 3 | Send Status and Sense |
| Command Complete | | |

Bidirectional DataSN Example

| Initiator Function | PDU Type | Target Function |
|---|---|---|
| Command request<br>(Read-Write) | SCSI Command >>><br>Read-Write | |
| | | Process old commands |
| | <<< R2T<br>R2TSN = 0 | Ready to process<br>WRITE command |
| * Receive Data | <<< SCSI Data-in<br>DataSN = 0, F=0 | Send Data |
| * Receive Data | <<< SCSI Data-in<br>DataSN = 1, F=1 | Send Data |
| * Send Data<br>for R2TSN 0 | SCSI Data-out >>><br>DataSN = 0, F=1 | Receive Data |
| | <<< SCSI Response<br>ExpDataSN = 2 | Send Status and Sense |
| Command Complete | | |

*) 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

| Initiator Function | PDU Type & Content | Target Function |
|---|---|---|
| Command request<br>(write)<br>+ immediate data | SCSI Command (WRITE)>>><br>F=0 | Receive command<br>and data<br>and queue it |
| Send Unsolicited<br>Data | SCSI Write Data >>><br>DataSN = 0, F=1 | Receive more Data |
| | | Process old commands |
| | <<< R2T<br>R2TSN = 0 | Ready for more data |
| Send Data<br>for R2TSN 0 | SCSI Write Data >>><br>DataSN = 0, F=1 | Receive Data |
| | <<< SCSI Response<br>ExpDataSN = 0 | Send Status and Sense |
| Command Complete | | |

## 07 CRC Examples

N.B. all Values are Hexadecimal

32 bytes of zeroes:

```
  Byte:        0  1  2  3

     0:       00 00 00 00
   ...
    28:       00 00 00 00

    CRC:      aa 36 91 8a
```

32 bytes of ones:

```
  Byte:        0  1  2  3

     0:       ff ff ff ff
```

```
      ...
       28:       ff ff ff ff

      CRC:       43 ab a8 62

  32 bytes of incrementing 00..1f:

     Byte:        0  1  2  3

        0:       00 01 02 03
      ...
       28:       1c 1d 1e 1f

      CRC:       4e 79 dd 46

  32 bytes of decrementing 1f..00:

     Byte:        0  1  2  3

        0:       1f 1e 1d 1c
      ...
       28:       03 02 01 00

      CRC:       5c db 3f 11
```

## Appendix C. Sync and Steering with Fixed Interval Markers

This appendix presents a simple scheme for synchronization (PDU boundary retrieval). It uses markers including synchronization information placed at fixed intervals in the TCP stream.

A Marker consists of:

```
Byte /    0       |       1       |       2       |       3       |
    /             |               |               |               |
   |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
   +---------------+---------------+---------------+---------------+
  0| Next-iSCSI-PDU-start pointer - copy #1                        |
   +---------------+---------------+---------------+---------------+
  4| Next-iSCSI-PDU-start pointer - copy #2                        |
   +---------------+---------------+---------------+---------------+
```

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 offset is counted from the marker end to the beginning of the next header. The marker uses two copies of the pointer so that a marker spanning a TCP packet boundary should leave at least one valid copy in one of the packets.

The offset to the next iSCSI PDU header is counted in terms of the TCP stream data. Anything counted in the TCP sequence-number is counted for the offset. Specifically this includes any bytes "inserted" in the TCP stream by an UFL and it excludes any other markers inserted between the one we are examining and the next PDU header. 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.

### 08 Markers At Fixed Intervals

At fixed intervals in the TCP byte stream, a marker is inserted. Each end of the iSCSI session specifies during login the interval at which it is willing to receive the marker or disables the marker altogether. 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 TCP stream data. Anything counted in the TCP sequence-number is counted for the interval and the initial marker-less interval. Specifically this includes any bytes "inserted" in the TCP stream by an UFL.

When reduced to iSCSI terms markers MUST indicate the offset to a 4-byte word boundary in the stream. The last 2 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.

## 09 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.

# Appendix D. Login/Text Operational Keys

ISID and TSID form collectively the SSID (session id). A TSID of zero indicates a leading connection. Some session specific parameters MUST be carried only on the leading connection and cannot be changed after the leading connection login (e.g., MaxConnections, the maximum number of connections). This holds even for a single connection session with regard to connection restart. The keys that fall into this category have the use defined as LO (Leading Only).

Keys that can be used only during login have the use defined as IO (initialize only) while those that can be used in both the login phase and full feature phase have the use defined as ALL.

Keys that can be used only during full feature phase have the use defined as FFPO (full feature phase only).

Unless explicitly stated otherwise, all key=value pairs specified here are session specific.

## 10 MaxConnections

Use: LO
Who can send: Initiator and Target

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

Default is 1.

Initiator and target negotiate the maximum number of connections requested/acceptable. The lower of the 2 numbers is selected.

## 11 SendTargets

Use: FFPO
Who can send: Initiator

For a complete description see Appendix E.

## 12 TargetName

Use: LO by initiator ALL by target, Declarative
Who can send: Initiator and Target

TargetName=<iSCSI-Name>

Examples:

```
   TargetName=iqn.1993-11.com.disk-vendor.diskarrays.sn.45678
   TargetName=eui.020000023B040506
```

This key MUST be provided by the initiator of the TCP connection to the remote endpoint before the end of the login phase. The iSCSI Target Name specifies the worldwide unique name of the target. The TargetName key may also be returned by the "SendTargets" text request (and that is its only use when issued by a target).

## 13 InitiatorName

Use: LO, Declarative
Who can send: Initiator

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
```

This key MUST be provided by the initiator of the TCP connection to the remote endpoint before the end of the login phase. The Initiator key enables the initiator to identify itself to the remote endpoint. The use of the group name "iSCSI" is interpreted as "other side of TCP connection". The target may silently ignore this key if it does not support it, and does not need to track or verify which initiators use it.  A target that supports this field may use it to allow or deny access to an initiator.

## 14 TargetAlias

Use: ALL
Who can send: Target

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.

### 15 InitiatorAlias

Use: ALL
Who can send: Initiator

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.

### 16 TargetAddress

Use: ALL
Who can send: Target

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

If the TargetAddress is being 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 more fully described in Appendix E.

17 FMarker

Use: IO
Who can send: 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 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.

18 RFMarkInt

Use: IO
Who can send: Initiator and Target

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

This is a connection specific parameter.

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

Default is 2048.

### 19 SFMarkInt

Use: IO
Who can send: Initiator and Target

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

This is a connection specific parameter.

Indicates at what interval (in 4-byte words) the sender accepts to send the markers. The number MUST be within the range required by the receiver.  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 "pure" payload between markers).

Default is 2048.

### 20 InitialR2T

Use: ALL
Who can send: Initiator and Target

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. Once InitialR2T has been set to 'no', it cannot be set back to 'yes'. Note that only the first outgoing data burst (immediate data and/or or separate PDUs) can be sent unsolicited by an R2T.

### 21 BidiInitialR2T

Use: ALL
Who can send: Initiator and Target

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'. Note that only the first outgoing data burst (immediate data and/or or separate PDUs) can be sent unsolicited by an R2T.

### 22 ImmediateData

Use: ALL
Who can send: Initiator and Target

ImmediateData=<yes|no>

Default is yes.
Result function is AND.

Initiator and target negotiate support for immediate data. If initiator or target wants to turn immediate data off they have to state that. ImmediateData can be turned on if both 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 |

### 23 DataPDULength

Use: LO
Who can send: Initiator and Target

DataPDULength=<number-0-to-(2**15-1)>

Default is 16 units.

Initiator and target negotiate the maximum data payload supported for SCSI command or data PDUs in units of 512 bytes. The minimum of the 2 numbers is selected.

A value of 0 indicates no limit.

24 MaxBurstSize

Use: LO
Who can send: Initiator and Target

MaxBurstSize=<number-0-to-(2**15-1)>

Default is 512 units.

Initiator and target negotiate maximum SCSI data payload in an 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) in units of 512 bytes.

A value of 0 indicates no limit (the largest possible number).

The minimum of the 2 numbers is selected.

25 FirstBurstSize

Use: LO
Who can send: Initiator and Target

FirstBurstSize=<number-from-0-to-(2**15-1)>

Default is 128 units.

Initiator and target negotiate the maximum amount of unsolicited data an iSCSI initiator may send to the target, during the execution of a single SCSI command, in units of 512 bytes.

A value of 0 indicates no limit (the largest possible number).

The minimum of the 2 numbers is selected.

FirstBurstSize MUST NOT exceeds MaximumBurstSize.

26 LogoutLoginMaxTime

Use: LO
Who can send: Initiator and Target

LogoutLoginMaxTime=<number-from-2-3600>

Default is 3.

Initiator and target negotiate the maximum time in seconds before which 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 minimum of the 2 values is selected.

27 MaxOutstandingR2T

Use: LO
Who can send: Initiator and Target

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

The default is 1.

Initiator and target negotiate the maximum number of outstanding R2Ts per task. The minimum of the two values is selected.

28 DataPDUInOrder

Use: LO
Who can send: Initiator and Target

DataPDUInOrder=<yes|no>

The 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.

29 DataSequenceInOrder

Use: LO
Who can send: Initiator and Target

DataSequenceInOrder=<yes|no>

The 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 no, Data Sequences MUST be transferred using continuously increasing offsets except for error recovery.

30 ErrorRecoveryLevel

Use: LO
Who can send: Initiator and Target

ErrorRecoveryLevel=<0 to 2>

Default is 0.

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 to them some new ones.

In the description of recovery mechanisms, certain recovery classes are specified. Section 8.12 describes the mapping between the classes and the levels.

31 SessionType

Use: LO, Declarative
Who can send: Initiator

SessionType= <discovery|normal>

Default is Normal.

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

A discovery session indicates to the Target that the only purpose of this Session is discovery. The only command accepted by a target in this type of session is a text request with a SendTargets key.

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

## 32 The Vendor Specific Key Format

Use: ALL
Who can send: Initiator and Target

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 it is suggested to use the reversed DNS-name as a prefix to the key-proper.

## Appendix E. SendTargets operation

To reduce the amount of configuration required on an initiator, iSCSI provides the SendTargets text request. This command is sent by the initiator 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 need not be specified. Otherwise, the session is a normal, operational session. The SendTargets command MUST be sent only during the full feature phase of a normal or discovery session.

A system containing 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 shall 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>

If no target name is specified, the session should respond with addresses only 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 containing a list of targets and 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 Appendix D-12.

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 contain no 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 "Naming and Addressing" section.

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.
iSCSI addresses belonging with the same portal group tag support spanning multiple-connection sessions across this set of addresses.
iSCSI addresses that do not support multiple-connection sessions with other addresses must have their own unique portal group tag.

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 subsequent SendTargets commands to discover new targets.

Examples:

This example is the SendTargets response from a single target that has no other interface ports.

Initiator sends text request containing:

```
   SendTargets=all
```

Target sends text response containing:

```
   TargetName=iqn.1993-11.com.acme.diskarray.sn.8675309
```

Note that all it really 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. Here's 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
```

Note that 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.

Also note that 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 been returned as well.

The next text response shows a target that supports spanning sessions across multiple addresses, indicating this using 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.

Note that this SendTargets response does not indicate whether .49 supports multiple connections per session; this is communicated via the MaxConnections text key upon login to the target.

# Appendix F. SCSI Alias designation formats

## 33 Format codes

The SCSI command set (see [SPC3]) defines an alias mechanism (CHANGE ALIASES and REPORT ALIASES commands) for mapping long identifiers (such as iSCSI Names) into shorter values for use in parameter data, such as third party commands.

This clause defines the alias entry formats and codes used in these commands to designate iSCSI devices or ports. As noted in 1.1, the protocol identifier used in these formats shall be set to 0x05 (see [SPC3]) and the format code values are defined in the following table:

| Format Code | Description | Max Length (bytes) | Content |
|---|---|---|---|
| 00h | iSCSI Name | 256 | Name in UTF-8 format (null terminated with pad). See Y.2. |
| 01h | iSCSI Name with IPv4 address | 268 | Name in UTF-8 format (null terminated with pad), binary IPv4 address, binary TCP port, binary Internet Protocol Number. See Y.3 |
| 02h | iSCSI Name with IPName | 520 | Name in UTF-8 format (null terminated), IPName (null terminated with pad), binary TCP port, binary Internet Protocol Number. See Y.4 |
| 01h | iSCSI Name with IPv6 address | 268 | Name in UTF-8 format (null terminated with pad), binary IPv6 address, binary TCP port, binary Internet Protocol Number. See Y.5 |
| 04-FFh | reserved | n/a | n/a |

In all cases, if the length is not a multiple of 4, then zero to three pad bytes are added (as indicated).

A designation that contains no IP addressing information or contains IP addressing information that does not address the named SCSI device may require the SCSI logical unit device server to have access to a name server or to other discovery protocols to resolve the given iSCSI Name to an IP address through which the device server may establish iSCSI Login.

## 34 iSCSI Name designation format

The following table describes the iSCSI Name designation format.

```
 Byte /    0       |       1       |       2       |       3       |
    /              |               |               |               |
   |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
   +---------------+---------------+---------------+---------------+
  0| iSCSI Name + Null (00h) + pad (0) (if necessary)              |
  +/                                                               /
   +---------------+---------------+---------------+---------------+
  x
```

The iSCSI Name field shall contain the iSCSI Name of an iSCSI Node.

A Null (00h) byte shall terminate the iSCSI Name.

Zero to three bytes set to zero shall be appended as padding so that the total length of the designation is a multiple of four. The pad field shall be ignored.

An iSCSI Name designation is valid if the device server has access to a SCSI domain containing an IP network and there exists an iSCSI Node on that network with the specified iSCSI Name.

## 35 iSCSI Name with binary IPv4 address designation format

The following table describes the iSCSI Name with IPv4 address designation format.

```
     Byte /    0       |       1       |       2       |       3       |
        /              |               |               |               |
       |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
       +---------------+---------------+---------------+---------------+
      0| iSCSI Name + Null (00h) + pad (0) (if necessary)              |
      +/                                                               /
       +---------------+---------------+---------------+---------------+
      x| IPv4 address                                                  |
       +---------------+---------------+---------------+---------------+
    x+4| Reserved                      | Port Number                   |
       +---------------+---------------+---------------+---------------+
    x+8| Reserved                      | Internet Protocol Number      |
       +---------------+---------------+---------------+---------------+
   x+12
```

The iSCSI Name field shall contain the iSCSI Name of an iSCSI Node.

A Null (00h) byte shall terminate the iSCSI Name.

Zero to three bytes set to zero shall be appended as padding so that the total length of the designation is a multiple of four. The pad field shall be ignored.

The IPv4 Address field shall contain an IPv4 address. See [RFC791] for a description of IPv4 addresses.

The Port Number field shall contain a port number. See [RFC790] for a description of port numbers.

The internet protocol number field shall contain an Internet protocol number. See [RFC790] for a description of Internet protocol numbers.

An iSCSI Name with IPv4 address designation is valid if the device server has access to a SCSI domain containing an IP network and there exists an iSCSI Node on that network with the specified iSCSI Name. The IPv4 address, port number and internet protocol number provided in the designation may be used by a device server for addressing to discover and establish communication with the named iSCSI Node.

Alternatively, the device server may use other protocol specific or vendor specific methods to discover and establish communication with the named iSCSI Node.

## 36 iSCSI Name with IPname designation format

The following table describes the iSCSI Name with IPname designation format.

```
  Byte /    0       |       1       |       2       |       3       |
      /             |               |               |               |
     |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
     +---------------+---------------+---------------+---------------+
    0| iSCSI Name + Null (00h) + IPname + Null (00h) +               |
    +/       pad (0) (if necessary)                                  /
     +---------------+---------------+---------------+---------------+
    x| Reserved                      | Port Number                   |
     +---------------+---------------+---------------+---------------+
  x+4| Reserved                      | Internet Protocol Number      |
     +---------------+---------------+---------------+---------------+
  x+8
```

The iSCSI name field shall contain the iSCSI Name of an iSCSI Node. See [NDT] for a description of iSCSI Names. The iSCSI name field shall not include a byte set to 00h.

A Null (00h) byte shall terminate the iSCSI Name.

The IPname field shall contain a Internet protocol domain name. See [RFC1035] for a description of domain names.

A Null (00h) byte shall terminate the Internet protocol domain name.

Zero to three bytes set to zero shall be appended as padding so that the total length of the designation is a multiple of four. The pad field shall be ignored.

An iSCSI Name with IPname designation is valid if the device server has access to a SCSI domain containing an IP network and there exists an iSCSI Node on that network with the specified iSCSI Name. The domain name, port number and internet protocol number provided in the designation may be used by a device server for addressing to discover and establish communication with the named iSCSI Node. Alternatively, the device server may use other protocol specific or vendor specific methods to discover and establish communication with the named iSCSI Node.

## 37 iSCSI Name with binary IPv6 address designation format

The following table describes the iSCSI Name with IPv6 address designation format.

```
  Byte /    0       |       1       |       2       |       3       |
     /              |               |               |               |
    |7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
    +---------------+---------------+---------------+---------------+
   0| iSCSI Name + Null (00h) + pad (0) (if necessary)              |
   +/                                                               /
    +---------------+---------------+---------------+---------------+
   x| IPv6 address                                                  |
   +/                                                               /
    +---------------+---------------+---------------+---------------+
x+16| Reserved                      | Port Number                   |
    +---------------+---------------+---------------+---------------+
x+20| Reserved                      | Internet Protocol Number      |
    +---------------+---------------+---------------+---------------+
x+24
```

The iSCSI Name field shall contain the iSCSI Name of an iSCSI Node.

A Null (00h) byte shall terminate the iSCSI Name.

Zero to three bytes set to zero shall be appended as padding so that the total length of the designation is a multiple of four. The pad field shall be ignored.

The IPv6 Address field shall contain an IPv6 address. See [RFC2373] for a description of IPv6 addresses.

The Port Number field shall contain a port number. See [RFC790] for a description of port numbers.

The Internet Protocol Number field shall contain an Internet protocol number. See [RFC790] for a description of Internet protocol numbers.

An iSCSI Name with IPv6 address designation is valid if the device server has access to a SCSI domain containing an IP network and there exists an iSCSI Node on that network with the specified iSCSI Name. The IPv6 address, port number and internet protocol number provided in the designation may be used by a device server for addressing to discover and establish communication with the named iSCSI Node.

Alternatively, the device server may use other protocol specific or vendor specific methods to discover and establish communication with the named iSCSI Node.

## Appendix G. 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, and each of the recovery classes described has initiator procedures as well as target procedures. Readers may please note that 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 like recovery of Synchronization on a header digest error are considered out-of-scope in these algorithms. In this particular example, header digest error may lead to connection recovery if sync and steering layer is not implemented.

It may also be noted that these algorithms strive to convey the iSCSI error recovery concepts in simplest terms, and are not designed to be optimal.

## 38 General Data structure and procedure description

This section defines the procedures and data structures that are commonly used by all the error recovery algorithms. Please note that 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 {
        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 PerformConnectionRecovery;
   };

   struct Session {
           int NumConnections;
           int ISID;
           int TSID;
           int NextCmdSN;
           int Maxconnections;
           Boolean FailoverSupport;
           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);
```

## 39 Within-command error recovery algorithms

### 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 - 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 carrying StatSN - SCSI Response, Text Response etc.

## 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 = "Delivery subsystem failure";
         }
    } else {
          TCB.Reason = "Delivery subsystem failure";
    }
    if (TCB.Reason = "Delivery subsystem failure") {
          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 (current PDU has F-bit set) {
          TCB.FbitReceived = TRUE;
      }
      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 ) {
          if (TCB.StatusXferd is TRUE and
               (TCB.FbitReceived is TRUE or
                        task is already considered failed)) {
                     SCSI-Task-Completion(TCB);
          }
   }
}
```

## 3  Target algorithms

```
   Receive-a-In-PDU(Connection, CurrentPDU)
   {
     check-basic-validity(CurrentPDU);
     if (Header-Digest-Bad) {
         Build-And-Send-Reject(Connection, CurrentPDU,
                                      Header-Digest-Error);
         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) {
                Decrement TCB.ActiveR2Ts.
                if (current PDU is unsolicited and
                       data received is less than I/O size and
                         data received is less than FirstBurstSize) {
                    send-recovery-R2T = TRUE;
                    Note the missing data PDUs 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 = "Delivery subsystem failure";
            }
         }
         if (TCB.ActiveR2Ts = 0) {
            Build-And-Send-Status(Connection, TCB);
         }
     } else if (CurrentPDU.type = SNACK) {
         if (this is data retransmission request) {
            if (TargetDataSNACKSupported) {
                 if (the request is satisfiable) {
                       Build-And-Send-A-Data-Burst(CurrentPDU, TCB);
                 } else {
                       TCB.Reason = "SNACK Rejected";
                 }
            } else {
                 TCB.Reason = "SNACK Rejected";
            }
            if (TCB.Reason = "SNACK Rejected") {
                 Build-And-Send-Reject(Connection, CurrentPDU,
                                                     Data-SNACK-Reject);
                 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 = "Delivery subsystem failure";
          if (TCB.ActiveR2Ts = 0) {
             Build-And-Send-Status(Connection, TCB);
          }
      }
    }
```

## 40 Within-connection recovery algorithms

### 4 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,
                                       Retrybit);
   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 deal only 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 retransmission of unacknowledged commands in more cases than is necessary.  This will not however affect the correctness of operation since the target is required to discard the duplicate CmdSNs.

The procedure Build-And-Send-Async is defined in Connection recovery algorithms.
The procedure Status-Expect-Timeout-Handler describes how initiators may proactively attempt to retrieve Status if they choose to. This procedure is assumed to be triggered much before the standard ULP timeout.

## 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.PerformConnectionRecovery = TRUE;
          }
       } else {
          Connection.PerformConnectionRecovery = TRUE;
       }
       if (Connection.PerformConnectionRecovery is TRUE) {
          Start-Timer(Connection-Recovery-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, Retrybit);
    }
}

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);
        }
    }
}
```

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,
                                    0, TargetConnectionRecoveryTimeout);
       }
   }
```

5  Connection recovery algorithms

3. 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,
                                                    bits to set);
Connection-Recovery-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, hung connection while the connection is in the full-feature phase, are all assumed to be asynchronously signaled to iSCSI layer using the Transport_Exception_Handler procedure.

## 4. 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.iSCSIEvent = LogoutRequest) or
                (CurrentPDU.iSCSIEvent = ConnectionDropped)) {
          Retrieve the AffectedConnection for CurrentPDU.Parameter1.
          AffectedConnection.State = ASYNC_MSG_RCVD;
          AffectedConnection.PerformConnectionRecovery = TRUE;
          Start-Timer(Connection-Recovery-Handler,
                        AffectedConnection, CurrentPDU.Parameter2);
        }
    } else if (CurrentPDU.type = LogoutResponse) {
        Retrieve the RecoveryConnection for CurrentPDU.CID.
        if (CurrentPDU.Response = failure) {
           RecoveryConnection.State = RECOVERY_START;
           Start-Timer(Connection-Resource-Timeout-Handler,
                       RecoveryConnection, InitiatorRecoveryTimeout);
        } else {
            RecoveryConnection.State = FREE;
        }
    } else if (CurrentPDU.type = LoginResponse) {
         if (this is a response to an implicit Logout) {
            Retrieve the RecoveryConnection.
            if (successful) {
                RecoveryConnection.State = FREE;
                Connection.State = LOGGED_IN;
            } else {
                 RecoveryConnection.State = RECOVERY_START;
                 DestroyTransportConnection(Connection);
                 Start-Timer(Connection-Resource-Timeout-Handler,
                       RecoveryConnection, InitiatorRecoveryTimeout);
            }
         }
    } else { /* REST UNRELATED TO CONNECTION-RECOVERY,
              * NOT SHOWN */
    }
    if (RecoveryConnection.State = FREE) {
       for (each command that was active on RecoveryConnection) {
            NewConnection = Pick-A-Logged-In-Connection(Session);
            Build-And-Send-Command(NewConnection, TCB, Retrybit);
        }
    }
}

Connection-Recovery-Handler(Connection)
{
    Retrieve Session from Connection.
    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.PerformConnectionRecovery = TRUE;
       if (the event is an unexpected transport disconnect) {
           Connection.State = XPT_CLEANUP;
       } else {
           Connection.State = RECOVERY_START;
       }
       Start-Timer(Connection-Recovery-Handler, Connection, 0);
   }
```

## 5. Target algorithms

```
   Receive-a-In-PDU(Connection, CurrentPDU)
   {
       check-basic-validity(CurrentPDU);
       if (Header-Digest-Bad) {
         Build-And-Send-Reject(Connection, CurrentPDU,
                                         Header-Digest-Error);
         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 RecoveryConnection from CurrentPDU.CID).
              for (each command active on RecoveryConnection) {
                   Quiesce-And-Prepare-for-New-Allegiance(Session, TCB);
                   TCB.CurrentlyAllegiant = FALSE;
              }
              Cleanup-Connection-State(RecoveryConnection);
              if ((quiescing successful) and (cleanup successful)) {
                   Build-And-Send-Logout-Response(Connection,
                                       RecoveryConnection.CID, Sucess);
              } else {
                   Build-And-Send-Logout-Response(Connection,
                                       RecoveryConnection.CID, Failure);
              }
          }
       } else if (CurrentPDU.type = TaskManagement) {
            if (CurrentPDU.function = "TaskReassign") {
                  if (Session.FailoverSupport is not TRUE) {
                     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.PerformConnectionRecovery = TRUE;
       if (the event is an unexpected transport disconnect) {
           Connection.State = XPT_CLEANUP;
       } else {
           Connection.State = RECOVERY_START;
       }
       Start-Timer(Connection-Resource-Timeout-Handler, Connection,
                                 TargetConnectionRecoveryTimeout);
       if (this Session has full-feature phase connections left) {
            DifferentConnection = Pick-A-Logged-In-Connection(Session);
            Build-And-Send-Async(DifferentConnection, DroppedConnection,
                                    0, TargetConnectionRecoveryTimeout);
       }
   }
```

## Full Copyright Statement