Design of the iSCSI Protocol

Kalman Z. Meth, Julian Satran
IBM Haifa Research Laboratory
Haifa, Israel
{meth,satran}@il.ibm.com

Abstract

The iSCSI protocol enables accessing SCSI I/O devices over an IP network. TCP is used as a transport for SCSI I/O commands. We describe the design considerations and decisions in defining the iSCSI protocol: why we use TCP, how multiple TCP connections can be used to increase performance and reliability, why we require allegiance of a command to a particular TCP connection, the importance of Direct Data Placement, various levels and complexity of error recovery, security and naming issues.

1.0. Introduction

The iSCSI protocol [1] is a transport for SCSI over TCP/IP [2] [3]. SAM-2 [4] defines an architecture model for SCSI transports, and iSCSI defines such a transport on top of TCP/IP. Other SCSI transports include SCSI Serial [5] and Fibre Channel Protocol (FCP) [6] [7]. Until recently standard IP protocol infrastructure (i.e. Ethernet) could not provide the necessary high bandwidth and low latency needed for storage access. With the recent advances in Ethernet technology, it is now practical (from a performance perspective) to access storage devices over an IP network. 1 Gigabit Ethernet is now widely available and is competitive with 1 and 2 Gigabit Fibre Channel, and 10 Gigabit Ethernet will soon also be available. Similar to FCP, iSCSI allows storage to be accessed over a Storage Area Network (SAN), allowing shared access to storage. A major advantage of iSCSI over FCP is that iSCSI can run over standard, off-the-shelf network components, such as Ethernet. A network that incorporates iSCSI SANs need use only a single kind of network infrastructure (Ethernet) for both data and storage traffic, whereas use of FCP requires a separate kind of infrastructure (Fibre Channel) for the storage. Furthermore, iSCSI (TCP) based SANs can extend over arbitrary distances, and are not subject to distance limitations that currently limit FCP. See FIGURE 1.

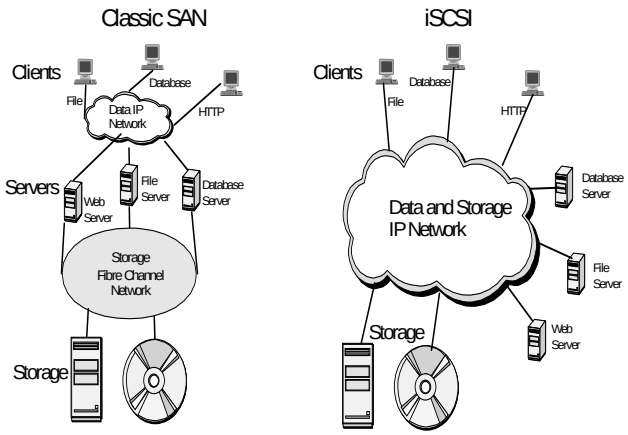

FIGURE 1. Classic SAN vs. iSCSI

iSCSI defines its own packets that are referred to as iSCSI Protocol Data Units (PDUs). iSCSI PDUs consist of a header and possible data, where the data length is specified within the iSCSI PDU header. Since iSCSI runs on top of TCP/IP, we have a layering of protocol levels. An iSCSI PDU is sent as the contents of one or more TCP packets. The protocol layering looks as depicted in FIGURE 2.

Ethernet Header	IP Header	TCP Header	iSCSI Header	iSCSI Data

FIGURE 2. Protocol Layering

Since iSCSI is designed to run on an IP network, iSCSI can take advantage of existing features and tools that were already developed for IP networks. The very use of TCP utilizes TCP's features of guaranteed in-order delivery of data and congestion control. IPSec [8] can be leveraged to provide security of an iSCSI SAN, whereas a new security mechanism would have to be developed for Fibre Channel. SLP (Service Location Protocol) [9] can be used by iSCSI to discover iSCSI

entities on the network. Thus in addition to iSCSI running on standard, cheaper, off-the-shelf hardware, iSCSI also benefits from using existing standard IP based tools and services.

2.0. Design issues

Some of the design decisions were strongly influenced by the design team's perception of how iSCSI would eventually be used. iSCSI was designed to allow efficient hardware and software implementations to access I/O devices attached over any IP network. iSCSI was also designed for a wide variety of environments and applications including local and remote storage access, local and remote mirroring, local and remote backup/restore. It was assumed that TCP/IP acceleration adapters and even iSCSI adapters would become prevalent, and it would be strongly desirable to define the protocol to allow high-performance adapter implementations. Mechanisms were therefore included to overcome various anticipated problems, such as maintaining high bandwidth despite frequently dropped packets. Care was taken to not limit the application of iSCSI to disks; mechanisms are provided for various types of SCSI devices, especially tapes, for which it is inconvenient and perhaps prohibitive to cancel and restart commands. Effort was made to exploit existing IP protocol suite infrastructure whenever possible, thus avoiding the need to re-design all levels of the protocol stack and related tools.

2.1. Why TCP?

Although attempts have been made to define SCSI over UDP, SCSI over IP, and even SCSI directly over Ethernet, the designers of iSCSI decided that it was best to define SCSI over TCP. There are several reasons to use TCP:

- TCP is a reliable connection protocol that works over a variety of physical media and interconnect topologies, and is implemented on a wide variety of machines.
- It is field proven and scalable, offers an end-to-end connection model independent of the underlying network.
- It is probably going to be well supported on underlying networks for some time in the future.

TCP has a mechanism to acknowledge all TCP packets that are received and to resend packets that are not acknowledged within a certain time-out period. Thus iSCSI packets sent over TCP that may get lost during delivery are automatically resent by TCP. If iSCSI were defined on top of a protocol that is not reliable and in-order, then iSCSI would have had to provide these services itself, since SCSI traffic must be reliable and is expected to be ordered. We would also have to provide a congestion control mechanism, which is also already provided by TCP. TCP was chosen also over the SCTP protocol [10] because of TCP's wide acceptance and its proven track record over many years. Although TCP has additional features that are not needed for transport of SCSI in general, the designers of iSCSI felt that the benefits of using an existing, well-tested and understood transport like TCP justified its use.

2.2. Sessions with multiple connections

The iSCSI entity corresponding to an I_T_NEXUS (Initiator - Target nexus) is an iSCSI session. Since TCP is used as the transport for iSCSI, iSCSI initiators are connected to iSCSI targets using TCP connections. It might be impossible to achieve the full bandwidth capability of the underlying physical transport by using a single TCP connection (possibly due to the TCP window size and the round-trip-time of TCP acknowledgements over long distances). Some protocols, like SCTP, automatically and transparently distribute their traffic over a number of connections in order to achieve the full bandwidth. There may also be several physical interconnects (i.e. separate cables) connecting the initiator and target, and it would be most desirable to aggregate and simultaneously utilize all such existing physical connections. TCP does not provide traffic aggregation nor does it provide its own failover mechanism. An iSCSI session is therefore defined to be a collection of one or more TCP connections connecting an initiator to a target. The TCP connections of a session may span several physical interconnects. See FIGURE 3.

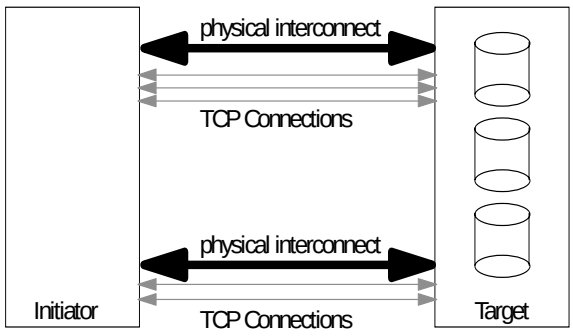

FIGURE 3. An iSCSI session consisting of multiple TCP connections over several physical interconnects.

Mechanisms are provided within iSCSI to efficiently spread its traffic over multiple TCP connections and to recover from connection failures. An iSCSI session may thus utilize multiple TCP connections to achieve higher utilization of the available bandwidth and/or for failover capabilities in case one of the physical interconnects becomes detached. An iSCSI session can remain active as long as it is still possible to establish at least one connection on any path connecting the initiator to the target.

In FCP fabrics, there may be multiple paths in the fabric through which an initiator is connected to a target. This causes complications for an initiator, because it appears think it has multiple devices, when in fact it simply has multiple paths to the same device. Some Fibre Channel vendors provide (expensive) multipath packages to deal with this issue. In iSCSI the problem is minimized because a single session connects the initiator to the target, even if there are multiple physical paths between them. The initiator sees a single image of all of its devices that exist on the target. In order to have multiple paths to a device in the same sense as with an FCP fabric, there would have to be multiple sessions between the initiator and target, and this is usually not needed.

2.3. Symmetric vs. asymmetric model

One of the issues that the designers grappled with was whether data should travel over the same connections as commands and control information. One approach is to send all SCSI commands, SCSI responses, and task management information over a *control* channel, while all data transfers go over separate *data* channels. We refer to this approach as the *asymmetric* model, since there would be different types of connections: control and data. Another approach is to transfer data over the same connection on which the corresponding command was sent. We call this *allegiance* of a command to a TCP connection. In this approach, all connections are treated equally, and we refer to it as the *symmetric* model. A major concern with the symmetric model is the possibility of filling up a connection with data from some command, closing the TCP window, and then being unable to deliver a high-priority task management request over the same channel. In the asymmetric model, only commands and task management requests are sent over the control channel. These are all relatively short messages, so the TCP window would almost never be closed, and the control channel would not block. A major drawback of the asymmetric model occurs when using iSCSI adapters. If the connections for a control channel and its data channels are handled by different iSCSI adapters, then the handling of a command would require the interaction between different iSCSI adapters, possibly through some software on the host machine. This was deemed most undesirable. In order to allow for complete solutions contained within individual iSCSI adapters, the symmetric model was adopted.

2.4. Uniform headers

Different iSCSI PDU headers contain different numbers of fields and field lengths, based on the particular information needed for that type of PDU. It would have been possible to have variable length PDU headers for the various types of PDUs. The length of the PDU header could be derived from the PDU opcode or from a length field that would be in a fixed position. However, such a scheme would require two read operations to read each PDU header, thus incurring a significant performance penalty, especially for software implementations. It was therefore decided to use a uniform-sized PDU header for all standard operations. The SCSI CDB is also a variable length field. In order to maintain a uniform-sized PDU header, the largest standard CDB size (16 bytes) was accounted for in the standard size PDU header. The iSCSI header for a typical SCSI command is reproduced (from [1]) in FIGURE 4.

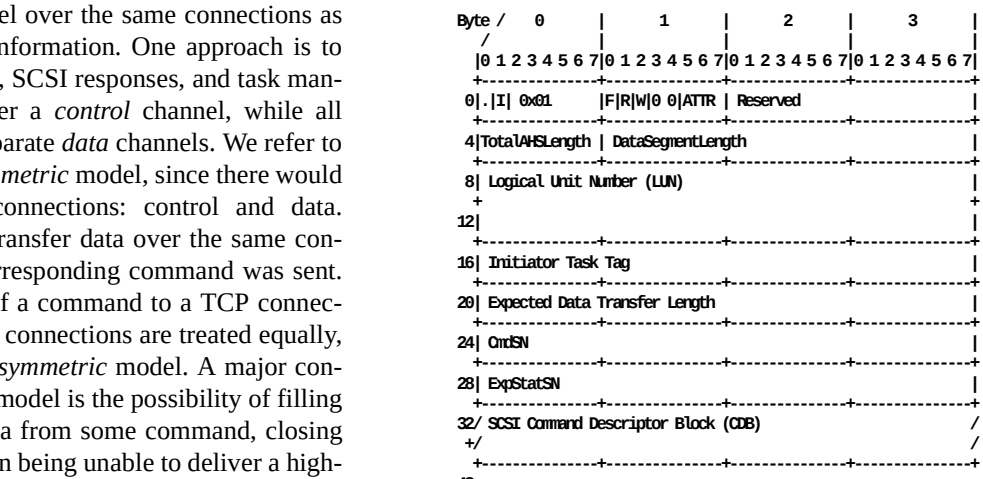

FIGURE 4. An iSCSI header for a SCSI command.

If a larger PDU is required (possibly because of a longer CDB), a special Additional Header Segment feature is included to specify a longer PDU header length. Only in these exceptional cases would a second read

operation be required to read the entire header. In the normal case, only a single fixed-size read operation is needed to read an iSCSI PDU header.

2.5. Direct data placement

In typical TCP implementations, data that arrives on a TCP connection is first copied into temporary buffers. The TCP driver then examines the connection identification information (source and destination addresses and port numbers) to determine the intended receiver of the data. The data is then copied into the receiver's buffers. For SCSI data, there might be many pending SCSI commands at any given instant, and the received data must typically be copied into the specific buffer that was provided by the SCSI layer for the particular command. This entire procedure might require the receiving host to copy the data a number of times before the data ends up in its final destination buffer. Such copies require a significant amount of CPU and memory bus usage that would adversely affect the system performance. It is therefore most desirable to be able to place the data in its final destination with a minimum number of copies.

Ideally, we would have liked to use a generic, well-established Direct Data Placement mechanism. However, no such mechanism existed in the IP suite of protocols. Therefore, in parallel to the iSCSI protocol, an RDMA (Remote Direct Memory Access) mechanism was also proposed to the IETF (Internet Engineering Task Force, responsible for the IP family of protocols). The RDMA proposal did not make quick enough progress towards standardization, so iSCSI could not depend on such a general mechanism being available. It was therefore necessary to add some Direct Data Placement support infrastructure directly into iSCSI. iSCSI Data PDU headers contain sufficient information to allow an iSCSI adapter to perform Direct Data Placement. The information provided in an iSCSI Data PDU header include: a transfer tag to identify the SCSI command and its corresponding buffer, a byte offset relative to the beginning of the corresponding buffer, and a data length parameter indicating the number of bytes being transferred in the current data packet. This information is sufficient to enable direct placement of the arriving data into pre-registered SCSI provided buffers. An iSCSI adapter that performs both TCP and iSCSI processing on the adapter will have sufficient information in the TCP and iSCSI headers to place arriving iSCSI data directly into the appropriate SCSI buffers without having to copy the data into additional temporary buffers on the host machine.

An iSCSI implementation can easily exploit a generic RDMA, whenever it becomes available.

2.6. Unsolicited and immediate data

The SCSI protocol has a command phase followed by a data transfer phase and a response phase. The data transfer phase is often driven by the target; the target specifies which data it is ready to process. Historically, some target devices would optimize their command processing based on where a disk head happened to be stationed. Data for a particular command need not be transferred in order. The target might request sectors of data in a non-contiguous order, based on internal target considerations.

With network-storage and RAID controllers with large caches, it is still beneficial in many cases for the target to choose what data should be transferred to/from the initiator, based on availability of memory or other resources. It should be noted, however, that extra messages are required for the target to inform the initiator which data to send. For large data transfers this is quite acceptable. For short data transfers, where only a single block of data is to be transferred, we end up paying the overhead of an extra message and the resulting round-trip time delay. This penalty grows with the network-distance between the initiator and target.

In order to reduce this performance penalty, iSCSI allows the initiator and target to agree upon a maximum data transfer length that may be sent *unsolicited*. A target may keep available some number of free buffers to accommodate up to some amount of unsolicited data. On any particular command, an initiator may send up to the agreed maximum amount of data without the target having to first explicitly request the data to be transferred. Once the maximum length for unsolicited has been reached, the target sends *Request to Transfer (R2T)* messages to the initiator specifying which data may now be transferred. The target provides a transfer tag with each R2T to allow Direct Data Placement of the data when it arrives.

Ordinarily, data is sent in iSCSI Data PDUs in the data transfer phase. If the amount of data to be transferred from initiator to target is so small that it will fit in a single iSCSI packet, then the data may be sent as *immediate* data. This means that the data is sent together with the SCSI command in the same iSCSI PDU, thus collapsing the command and data phases. Additionally, even if the data will extend beyond a single iSCSI PDU, the first part of the data may be sent together with the iSCSI Command PDU as immediate data, up to the maximum agreed size.

2.7. iSCSI recovery

The considerations in defining iSCSI Error Recovery mechanisms centered on the anticipated problems that would occur as a result of running on non-reliable IP based networks. Although infrastructure for Local Area Networks (LANs) has significantly improved in recent years so as to not require special iSCSI level recovery, problems do occur on long distance data transfer. We concentrate here on two particular problems:

- TCP connections do occasionally break and must be re-established.
- The TCP error detection mechanism (16 bit checksum) is not sufficiently reliable for storage data.

The iSCSI protocol defines mechanisms to overcome each of these types of errors, without having to fail a corresponding SCSI command. Simple iSCSI implementations may opt to not implement the advanced iSCSI level recovery, and may simply fail any pending SCSI command that is affected by an encountered TCP problem.

2.7.1. Session recovery. The most basic kind of recovery is called *session* recovery. In session recovery, whenever any kind of error is detected, the entire iSCSI session is terminated. All TCP connections connecting the initiator to the target are closed, and all pending SCSI commands are completed with an appropriate error status. A new iSCSI session is then established between the initiator and target, with new TCP connections. The upper level SCSI protocol may re-issue the failed commands, and these commands will be transported over the new TCP connections of the new session. iSCSI Session Recovery thus essentially passes the recovery on to the SCSI level to re-issue failed SCSI commands. This is the most basic kind of recovery that all iSCSI implementations must provide.

2.7.2. Digest failure recovery. TCP defines a 16-bit checksum to detect corruption of TCP packets. The checksum used by TCP was considered to be too weak for the requirements of storage. There are also instances where the TCP checksum does not protect iSCSI data, as when data is corrupted while being transferred on a PCI bus or while in memory. The iSCSI protocol therefore defines a 32-bit CRC digest on iSCSI packets in order to detect data corruption on an end-to-end basis. CRCs can be used on iSCSI PDU headers and/or data. If the iSCSI driver detects that data arrived with an invalid CRC digest, the data packet must be rejected. The command corresponding to the corrupted data can then be completed with an appropriate error indication. Alternatively, the iSCSI protocol defines a means by which the receiver of the corrupted data can inform the sender to resend a previously sent packet of data. The data transfer for a SCSI command may occur in several data transfer packets. It is often not necessary to resend all of the data for the entire SCSI command, as the receiver can specify exactly which packet of data is required.

2.7.3. Connection recovery. It sometimes occurs that a TCP connection is broken. This may occur as a result of some kind of transient exceptional condition on the network that prevents data from reaching its destination in a timely manner. If on the broken TCP connection there was a SCSI command that was delivered to an iSCSI target but for which a response had not yet been received by the initiator, the SCSI command could remain pending until the upper level SCSI driver decides that the command has timed out and tries to abort the affected SCSI command. The SCSI level time-out might be tens of seconds, all which time an application is waiting for its I/O operation to complete. Upon detection of a broken TCP connection, the iSCSI driver can either immediately complete the pending command with an appropriate error indication, or it can attempt to transfer the SCSI command to another TCP connection. If necessary, the iSCSI initiator driver can establish another TCP connection to the target, and the iSCSI initiator driver can inform the target that the allegiance of the SCSI command is being changed to another TCP connection. The target can then proceed to continue processing the SCSI command on the new TCP connection. The upper level SCSI driver remains unaware that a new TCP connection has been established and that the command has been transferred to the new connection.

We thus have several possible classes of recovery of increasing complexity: session recovery, digest-failure recovery, connection recovery. For some environments (such as local LANs with disks), it is sufficient to implement only session recovery, whereby upon any error, the iSCSI session is terminated and all pending SCSI commands are completed with an error indication. The upper level SCSI driver can then re-issue the failed commands that will be transported over the TCP connections of a new iSCSI session. For some environments (such as with tape commands where it is rather expensive to restart commands, or for long distance backups where communication problems are more likely) it will be more appropriate to implement a more complex level of recovery: digest failure recovery and/or connection recovery.

2.8. Message framing

It would be desirable if iSCSI PDUs could be easily marked within the TCP stream. One of the problems with the defined PDU scheme occurs when an iSCSI PDU header becomes corrupted, as detected by an iSCSI PDU header CRC mismatch. If an iSCSI header becomes corrupted, we have no way of knowing for sure where the next iSCSI PDU begins, since the length of the current PDU is stored in the corrupted header. The obvious remedy of closing the TCP connection and performing connection recovery might be overly expensive and disruptive. It would be best if we could simply discard the corrupted iSCSI PDU without having to close the connection, and continue processing on the same connection. In order to accomplish this, we need to know where the next iSCSI PDU begins, relative to the current (corrupted) PDU. Various ideas were proposed in order to accomplish this:

- Insist that the beginning of each TCP packet coincide with the beginning of an iSCSI PDU.
- Set a fixed length for all iSCSI PDUs.
- Use the urgent feature of TCP to indicate where the next iSCSI PDU begins in the TCP stream.

All of these suggestions were rejected either for imposing non-standard usage on TCP or because the proposed solution would be overly inefficient for normal operations.

Some type of generic framing capability will probably eventually be added to the TCP/IP family of protocols, and iSCSI will then be able to exploit it. In the meantime, the iSCSI specification suggests a scheme using markers to indicate where future iSCSI PDU headers are to appear in the TCP stream. By agreement between initiator and target, a marker, at fixed byte intervals in an iSCSI TCP stream, may be inserted to indicate where the next iSCSI PDU begins in the TCP stream. If a digest error is detected in an iSCSI PDU header, all data up to the PDU indicated by the marker is discarded. PDU processing continues from the next valid PDU indicated by the marker. A message is issued to request from the connecting party to resend the PDUs that were discarded. This proposed mechanism of markers does not in any way interfere with ordinary implementations or semantics of TCP.

2.9. Security

In Directly Attached Storage and in early Fibre Channel SANs, there was no perceived need for special security, as the storage was physically isolated from potential security threats. With the advent of IP attached storage, there is now the possibility of storage and other network traffic sharing a common network infrastructure. There is therefore a need to provide some kind of security within the framework of the iSCSI protocol. Some environments may require encryption (as when transferring sensitive data over an open network), while other environments may not require any special security enforcement (as on a private, physically secure, IP SAN). iSCSI therefore allows for different levels of security:

- No security.
- Authentication of communicating parties (initiator and target).
- Encryption.

Since iSCSI runs over IP, it is natural to leverage existing security mechanisms in the IP family of protocols. Thus IPSec [8] can be used to encrypt all iSCSI traffic over particular TCP connections.

2.10. Naming

Another issue that the designers of the iSCSI protocol grappled with was with naming. How should iSCSI entities be named? What iSCSI entities should be named?

Borrowing from other internet protocols, iSCSI uses a URL-like scheme to name targets. iSCSI names are meant to be global, similar to World Wide Names used by Fibre Channel. An iSCSI entity might have its IP address changed but retain its name. In fact, multiple IP addresses may map to a single iSCSI name, and multiple iSCSI names might map to a single IP address. An iSCSI entity is identified by its name and not by its address(es). This allows for easier handling of iSCSI names by proxies, gateways, Network Address Translation boxes, firewalls, etc. iSCSI names should be World-Wide unique. Typical iSCSI names might look like the following.

iqn.2001-04.com.acme:storage.disk2.sys1.xyz

The prefix *iqn* stands for *iSCSI Qualified Name*. The above named device was produced by the company that owned the domain name *acme.com* during *2001-04*. This may be followed by a character string, as deemed appropriate by the domain-name owner, to further qualify the name of the particular device.

2.11. Parameter negotiation

iSCSI initiators and targets can negotiate a number of parameters such as maximum PDU length, desired authentication method, number of TCP connections in the session. It was assumed that it was impossible to foresee all of the possible parameters that various implemen-

tations would want to implement. It was assumed that additional parameters would be introduced as implementations discover other useful parameters that could be negotiated. The iSCSI specification therefore specifies the parameters that were identified, and defines a mechanism to define new parameters that can be negotiated between cooperating implementations. All parameters are negotiated using Text keys, allowing vendors to introduce new (and/or proprietary) keys for special features. Any iSCSI entity that does not recognize a proprietary key may safely ignore the unknown key. All implementations are required to support the basic defined set of parameter keys. The parameter negotiation mechanism is thus extensible, and the set of negotiated parameters can be broadened as iSCSI gains acceptance.

3.0. How is iSCSI being used?

As mentioned above, iSCSI is seen as a cheaper alternative to Fibre Channel SANs. Because of its high cost, Fibre Channel SANs were previously considered to be limited to large data centers. iSCSI now allows for lower cost network access to shared data, and brings the SAN to the low-end market. Low-end iSCSI storage controllers have already appeared on the market, as have iSCSI tape backup systems.

Of course, previous investments in Fibre Channel infrastructure will continue to be utilized. iSCSI gateways have appeared to bridge existing Fibre Channel devices to Ethernet. Additionally, in parallel to iSCSI, two additional IP-based protocols have been introduced to connect Fibre Channel SANs over IP [11], [12].

4.0. Summary

We discussed above some of the issues that were addressed in designing the iSCSI protocol. On each issue there were alternate suggestions, each with its pros and cons. We discussed the reasons for the design choices on each of these issues. One of the main recurring themes was to make use of the existing IP protocol family to the greatest extent possible, without requiring any changes to existing protocols. As new protocols and features are added to the IP protocol family, these can also be leveraged by new iSCSI implementations.

5.0. Acknowledgements

Many people contributed ideas to the definition of the iSCSI protocol. We would especially like to thank those who participated in the Design Team and in our early phone calls and meetings including: Efri Zeidner, Costa Sapuntzakis, Mallikarjun Chadalapaka, Daniel Smith, Ofer Biran, Jim Hafner, John Hufferd, Mark Bakke, Randy Haagens, Matt Wakeley, Luciano Dalle Ore, Paul Von Stamwitz, Prasenjit Sarkar, Meir Toledano, John Dowdy, Steve Legg, Alain Azagury, Dave Nagle, David Black, John Matze, Steve DeGroote, Mark Schrandt, Gabi Hecht, Robert Snively, Brian Forbes, Nelson Nachum, Uri Elzur.

References

[1] Julian Satran, et al, iSCSI (Internet SCSI), draft-ietf-ips-iscsi-19.txt (Nov., 2002); see ietf.org/html.charters/ips-charter.html or www.haifa.il.ibm.com/satran/ips

[2] RFC791, internet Protocol (IP), DARPA Internet Program, Protocol Specification, Sep 1981; see http://ietf.org/rfc.html.

[3] RFC793, Transmission Control Protocol (TCP), DARPA Internet Program, Protocol Specification, Sep 1981; see http://ietf.org/rfc.html.

[4] SAM2, SCSI Architecture Model – 2, T10 Technical Committee NCITS (National Committee for Information Technology Standards), T10, Project 1157-D, Revision 23, 16 Mar 2002.

[5] SBP-2, Serial Bus Protocol - 2, ANSI NCITS.325-1999.

[6] FCP, SCSI-3 Fibre Channel Protocol, ANSI X3.269-1996.

[7] Benner, A. Fibre Channel: Gigabit Communications and I/O for Computer Networks, McGraw Hill, New York, 1996.

[8] RFC2401, S. Kent, R. Atkinson, Security Architecture for the Internet Protocol (IPSec), November 1998; see http://ietf.org/rfc.html.

[9] RFC2165, Service Location Protocol (SLP), June 1997; see http://ietf.org/rfc.html.

[10] RFC2960, R. Stewart, et al, Stream Control Transmission Protocol (SCTP), October 2000; see http://ietf.org/rfc.html.

[11] M Rajagopal, et al, Fibre Channel over TCP/IP (FCIP), draft-ietf-ips-fcovertcpip-12.txt (August, 2002); see ietf.org/html.charters/ips-charter.html or www.haifa.il.ibm.com/satran/ips.

[12] Charles Monia, et al, iFCP - A Protocol for Internet Fibre Channel Storage Networking, draft-ietf-ips-ifcp-13.txt (August, 2002); see ietf.org/html.charters/ips-charter.html or www.haifa.il.ibm.com/satran/ips.