DESIGN AND IMPLEMENTATION OF A SOFTWARE PROTOTYPE FOR STORAGE AREA NETWORK PROTOCOL EVALUATION

ASHISH PALEKAR
ANSHUL CHADDA
NARENDRAN GANAPATHY
ROBERT RUSSELL

InterOperability Laboratory
University of New Hampshire
Durham, New Hampshire 03824, USA

E-mail: apalekar@brocade.com, {achadda,ng3,rdr}@iol.unh.edu

ABSTRACT

In the past few years, storage area networks have gained popularity as a cost-effective means of providing enterprise-wide access to massive amounts of storage. A key idea in this development has been the replacement of the data bus that connects a host computer to a storage device with a high-speed data network. The first step in this direction was the replacement of the traditional SCSI bus with a Fibre Channel network that transported SCSI commands and data over greater distances at Gigabit per second speeds. A more recent development has been the emergence of protocols intended to transport SCSI commands and data over TCP/IP networks, and hence, over the global Internet.

This paper describes the design and implementation of a software prototype for evaluating several different storage area network transport protocols. A testbed has been developed that enables Linux PCs playing the roles of SCSI targets and SCSI initiators to interact over different transport networks using different protocols. The first implementation is for three different configurations: SCSI over Fibre Channel, SCSI over TCP/IP using the SCSI Encapsulation Protocol (SEP), and SCSI over TCP/IP using the iSCSI protocol currently under development by IETF.

Keywords: storage area networks, network protocols, Internet SCSI, iSCSI.

1 INTRODUCTION

The widespread adoption of the World Wide Web and e-commerce has created a huge demand for vast storage repositories that provide access 24 hours a day, 7 days a week. This demand has overwhelmed traditional storage mechanisms, and has prompted the development of promising new technologies.

One of these ideas is the “Storage Area Network”, or SAN [1]. The basic observation leading to development of the SAN concept is the following. Traditionally storage devices were connected to local computers via buses, such as that defined by the Small Computer Systems Interface, or SCSI. These buses had severe limitations on the distance between the accessing host and the accessed storage device (on the order of 25 meters maximum), on the number of devices which could be attached to the bus (7 or 15), and on the speed with which data could be transferred. They also caused access bottlenecks because only the host computer connected to the bus could access the storage. In order to access the storage, other computers would have to first contact that host computer and get it to act as a server on their behalf. On such a server, the speed at which the server accesses data from the storage device is comparable to, or lower than, the speed at which the server is able to transmit this data to other computers. Such a server could easily get swamped, especially as the amount of storage it was serving grew.

Therefore, the key idea behind SAN development is to replace this traditional data bus with a high-speed data network, such that the storage device is directly attached to that network and can interact with multiple host computers via that network. This not only eliminates the server bottleneck described above, but with the advent of gigabit-per-second and higher network capacity, actually improves the data transfer rates and decreases the costs.

The first technology to build on this idea was Fibre Channel [2], which uses a gigabit-per-second link to carry SCSI commands and data over distances up to 10 km. This technology requires special hardware adapters on both the client hosts and the target storage devices, and new fiber-optic cabling that meets the Fibre Channel specifications.

In the past year or two, several proposals have been made to leverage the huge existing network infrastructure, which

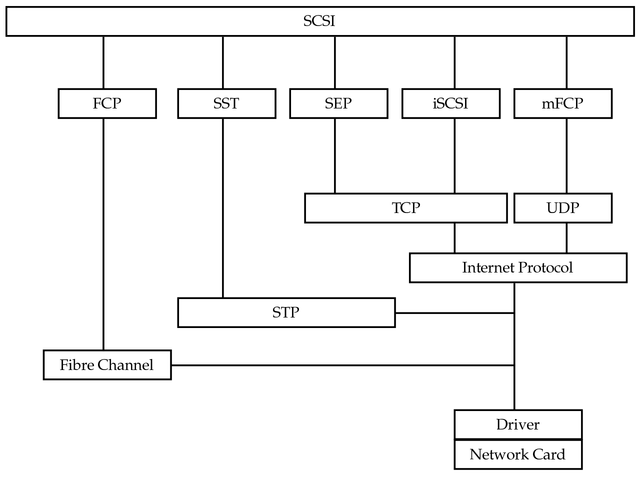

FIGURE 1: Different ways to map SCSI onto a network

is largely built on Ethernet and which forms the basis for the global Internet. As discussed in the next section, there are many ways to do this, but they all involve designing a new protocol that would encapsulate SCSI commands and data for transport over existing protocols, such as IP or TCP or UDP.

In order to evaluate these various protocol proposals, we designed a software testbed that provides both SCSI targets and SCSI initiators that can be interconnected by various networking technologies, primarily 1000 BaseT Ethernet. Using this design, we have implemented a number of the proposed SAN transport protocols in an attempt to evaluate their strengths and weaknesses, especially the completeness of their specifications, and to measure their performance under various conditions. Of course, the ultimate goal for most companies interested in SANs requires a hardware implementation of the protocols, so measurements of our software implementations would only allow relative, not absolute, comparisons. However, we designed the testbed to enable us to incorporate hardware implementations of both initiator and target components as they become available.

2 BACKGROUND

Figure 1 shows in schematic form some of the various ways that have been proposed to map SCSI onto a network. Fibre Channel is the lowest-level technology because it simply replaces the SCSI bus with its own network fiber and replaces the SCSI Host Bus Adapter (HBA) with its own driver.

The Scheduled Transfer Protocol (STP) [3] is a proposal being developed by the ANSI X3T11.1 working group for a protocol that would allow zero-copy message exchange over an unrouted network. SCSI on STP (SST) is another proposed standard being developed by the ANSI X3T10 working group. This defines a mapping of SCSI commands, data transfers and responses on SST sequences that in turn are carried by STP over a network.

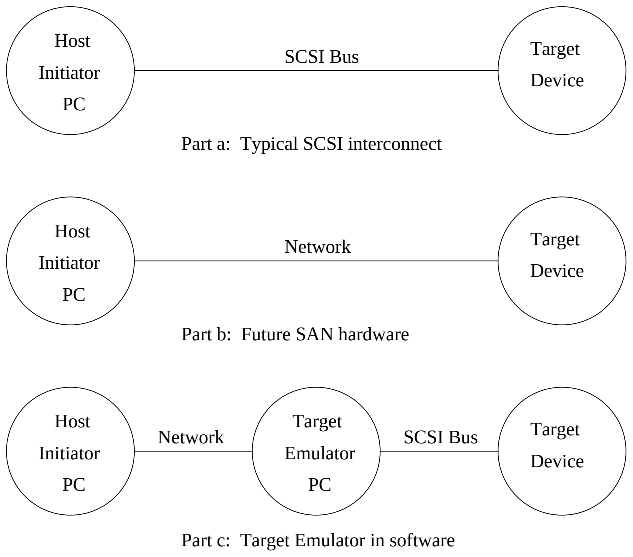

FIGURE 2: Steps in the Evolution of SANs and our SAN Target Emulator

Both the SCSI Encapsulation Protocol (SEP) [4] and the iSCSI Protocol [5] define an encapsulation of SCSI commands and data into protocol data units (PDUs) that are transported over the existing TCP/IP protocol stack. Thus the existing well-understood network infrastructure defined by TCP/IP is not reinvented, as with Fibre Channel and STP.

The Storage over IP (SoIP) [6] proposed by NISHAN systems is a family of protocols to carry storage traffic over IP. One of these, iFCP, runs over TCP, but another, mFCP, runs over the existing UDP protocol. This is different from SEP and iSCSI because it does not depend on the TCP protocol to provide a reliable transport service, but instead utilizes the unreliable UDP protocol. mFCP actually encapsulates the Fibre Channel protocol for delivery by UDP, so that lost UDP packets will be detected and recovered from by Fibre Channel. The expectation is that UDP provides faster message-oriented delivery and is cheaper to implement in hardware than the fairly complex stream-oriented TCP.

We decided to concentrate our initial design efforts on SEP because it was relatively simple and would enable us to develop and test the basic architecture quickly. Our second step was then to utilize this architecture to implement one of the early drafts of iSCSI, since that is actually going to become an international standard, whereas SEP is not.

3 DESIGN

Figure 2 illustrates the high-level approach we took toward our design. Part a of that figure shows a typical SCSI bus interconnection between a host and a device. Part b of

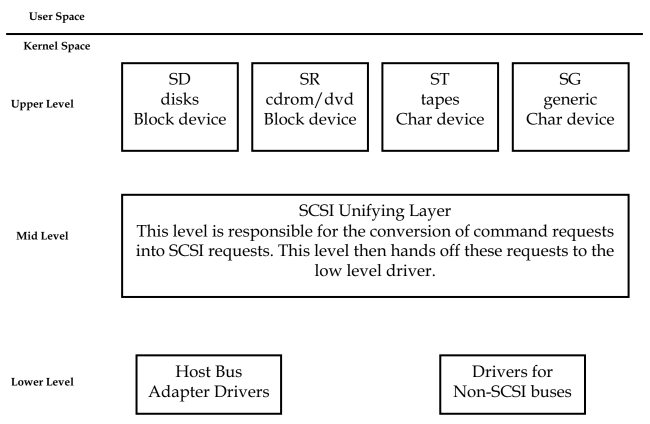

FIGURE 3: Layers in the SCSI subsystem initiator

the figure shows the SCSI bus replaced by a network. In such a configuration, special hardware on both the host and the device would be necessary to map the SCSI command and data sequences onto the network protocol. Except for Fibre Channel, such hardware is not yet widely available, and drivers are not yet written for Linux. However, that situation is changing rapidly. Part c of the figure shows how our software target emulator would be interposed as an interface between the network and the target device so that no new hardware would be required. Our software components on the host and on the target emulator use SEP or iSCSI to encapsulate SCSI commands and data and then transfer them over the network. The target emulator delivers these commands and data “down” to the attached SCSI device using an ordinary SCSI bus, while the host component delivers these commands and data “up” through the host SCSI subsystem to the host application.

Figure 3 shows that the SCSI subsystem on an initiator is traditionally divided into three layers: a upper-level that provides a “read/write” interface (which is traditionally used by a file system); a mid-level that converts the reads and writes into general SCSI commands and data transfer sequences; and a lower-level which is the driver for the HBA to a specific SCSI device and which controls the delivery of the SCSI commands and data through the HBA over the SCSI bus to that device.

Figure 4 shows the internal organization of a typical SCSI initiator and our software Target Emulator. This figure shows how, on the host, the 3 layers in the SCSI subsystem (as illustrated in Figure 3) are situated in kernel space below a traditional file system that in turn is accessed from user-space applications. The Front-End Initiator Driver (FEID) is the host software component that encapsulates the SCSI commands and data and delivers them to the TCP/IP subsystem for transport over the network. We have to write one such component for each SCSI transport protocol (SEP, iSCSI). Such a component for Fibre Channel is already available with the Linux kernel as a SCSI driver for the QLogic ISP 2x00 and the Interphase IPH5526, among others.

On the target emulator, our software is divided into two components, as shown in Figure 4. The lower-level component is called the Front End Target Driver (FETD). There is one such component for each SCSI transport protocol (SEP, iSCSI), and all understanding of the protocol on the target resides entirely within this component. The upper-level component is called the SCSI Target Mid-Level (STML) and is independent of the protocol used in the FETD. The primary purpose of the STML is to process SCSI commands and to hand off the responses generated to the appropriate FETDs. The STML is also responsible for SCSI error handling and for maintaining SCSI state information. In this sense, it represents an abstraction of all the common functions that need to be performed by FETDs written for different SCSI Transport Protocols. This layer enables the design of the Target Emulator to be generic for all SCSI Transport Protocols.

A key component in this design is the specification of two interfaces, shown as API in the figure. One enables the FETD to call STML functions to pass to the STML information and data received over the network from the initiator. The other interface enables data and status information to flow in the other direction.

When commands and/or data are received from the initiator, the FETD unencapsulates them and delivers them to the STML for processing. There are several different configurations for implementing this processing, as discussed in the next section. When data and responses flow in the opposite direction, the STML passes the information on to the appropriate FETD for encapsulation and transport over the network to the initiator.

Part c of Figure 2 illustrates that the design of the Target Emulator allows it to be used as a bridge to SCSI devices. It could also be used as a bridge to non-SCSI devices, such as IDE disks, by providing a SCSI to IDE translation function. In fact, we are currently using it as a bridge between the SEP and iSCSI protocols on a 1000BaseT network from the host initiator and a Fibre Channel interconnect to the target device.

Because some of our tools reside in the kernel and are intended to be used to simulate and test a wide variety of situations, an important component of the design of both the Target FETDs and the Initiator FEIDs is the design of an interface that allows them to be configured dynamically and interactively, (i.e., without rebuilding and rebooting the kernel or reloading a kernel module). Since SEP is a very simple protocol, configuration on a SEP initiator is limited to bringing SCSI devices up or down and assigning these devices to target IP addresses and Logical Unit Numbers (LUNs). However, iSCSI is a much more complex protocol which, in addition to having configuration requirements

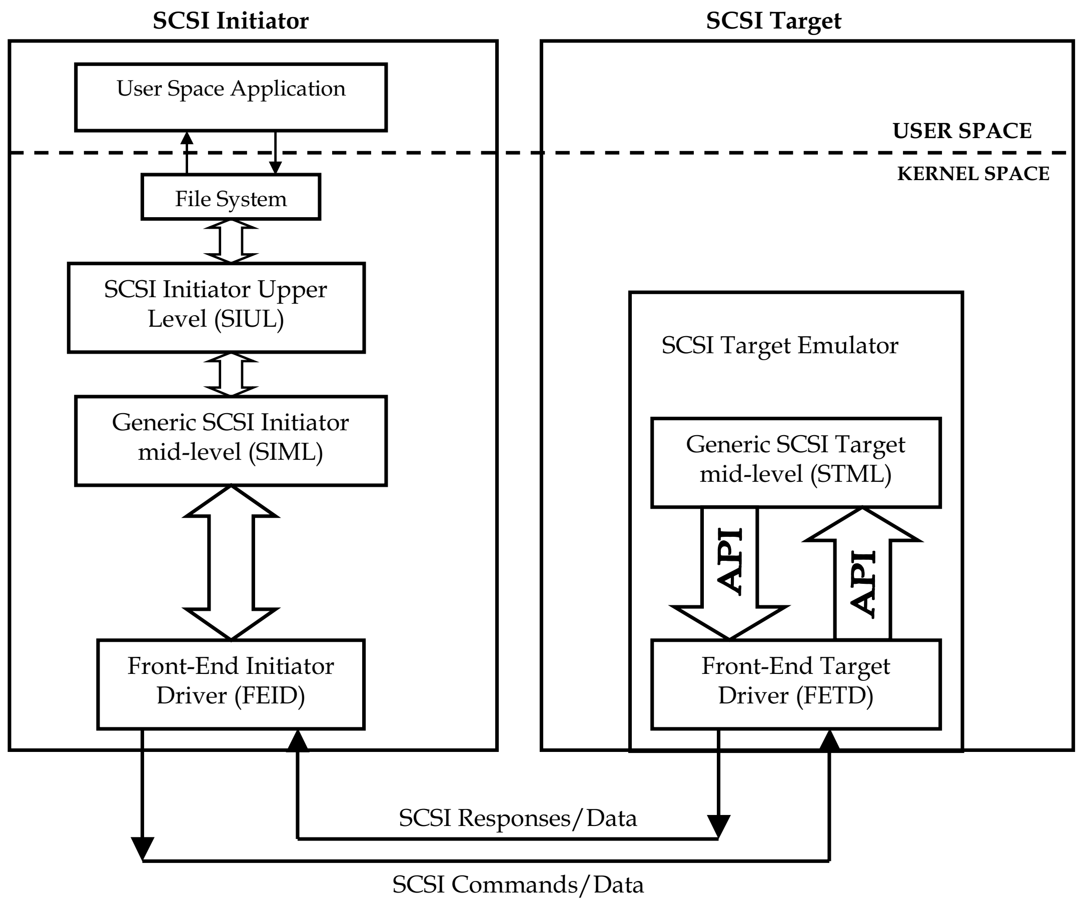
SCSI Initiator
SCSI Target
User Space Application
USER SPACE
KERNEL SPACE
File System
SCSI Initiator Upper Level (SIUL)
Generic SCSI Initiator mid-level (SIML)
Front-End Initiator Driver (FEID)
SCSI Target Emulator
Generic SCSI Target mid-level (STML)
API
API
Front-End Target Driver (FETD)
SCSI Responses/Data
SCSI Commands/Data

FIGURE 4: Internal components in the SCSI Initiator and SCSI Target

similar to those of SEP, also requires the dynamic negotiation between initiator and target of dozens of iSCSI parameters. To deal with this we have developed an interface that allows a user to interactively interrogate and modify these parameters and to control their use during the iSCSI negotiation process. Each FEID and FETD will incorporate interface functions that interact with user-level programs to accomplish these management tasks, as discussed further in section 4. There is one such interface for each FEID and FETD.

4 IMPLEMENTATION

All our coding was done in C for the Linux operating system, and is available at http://www.iol.unh.edu under the terms of the GNU Copyleft agreement.

On the initiator side we have implemented two drivers and have tested drivers for three transport protocols. The drivers for the SEP and iSCSI draft 3 protocols both utilize the existing Linux software TCP/IP stack to communicate over an Alteon ACENIC 1000 BaseT Ethernet card. The driver for the Fibre Channel protocol utilizes a QLogic Fibre Channel HBA that communicates over a Gigabit Fibre Channel link.

We have two separate implementations of the target-side software: one that operates entirely in user space, called the User Space Target Emulator (USTE), and one that operates entirely in kernel space, called the Kernel Space Target Emulator (KSTE). Both are organized into the two part structure described in the previous section, namely an FETD and a target emulator. For the USTE we have implemented the SEP FETD. For the KSTE we have implemented three FETDs: one for SEP, one for iSCSI draft 3 and one for Fibre Channel.

We have several reasons for wanting both a user-space and kernel-space version of the target emulator. We implemented the user-space version first because it was obviously easier to do, and this enabled us to easily refine our design on the target side and debug the initiator side. Once we had this up and working, we used it as a model that made the task of creating a kernel version simpler. Having both USTE and KSTE versions allows us to compare the performance of the two to see if the user-space/kernel-space difference actually produces performance differences (it does, see the next section). In the longer term we expect the kernel-space version to be the reference version, and the user-space version to form the basis for tools to do conformance and interoperability testing. We also plan to do stress testing of SCSI transport protocols implemented in hardware.

Protocol	KSTE	USTE
SEP-9000	35	–
SEP-1500	21	19
iSCSI	19	–
FC	45	–

Table 1: Observed Data Transfer Rates in MBps on 1 Gbps links

Both the USTE and the KSTE can be compiled into three different configurations. The most basic is one in which the SCSI commands are essentially ignored and data is simply transferred between the FETD and host memory without ever going to or from a real SCSI device. This configuration allows us to debug and test an FETD and the transport protocol it is utilizing without actually interacting with any SCSI device. It also allows us to make performance measurements of the transport protocol itself, for example, to see how well it utilizes the available network bandwidth. This target configuration makes it impossible to have a real user-level application on the initiator side, since such an application would expect to be accessing a real file system, not random data.

In the second configuration, the USTE or KSTE transfers data to and from a file in the host file system rather than directly to and from a SCSI device. This allows us to debug and test the SCSI Initiator software without actually having a SCSI disk as the target. Because the target data is stored in a file, it is easier to inspect and verify using standard file system tools on the target than it would be if it were actually stored on a raw disk. It also allows us to have a fully functional initiator in which applications read and write files to the target.

The third configuration is the "production" mode of operation in which the USTE or KSTE actually reads and writes data to a SCSI device.

As discussed in Section 3, an important part of our design was specification of an interactive interface that would allow users to dynamically configure and control the parameters required by SEP and especially iSCSI. This interface was implemented using the "/proc" file system in Linux. This file system enables a user-level program to read and write what looks like a file, but what is really a direct communication channel to a kernel module. Both our FEIDs and FETDs have functions in them to register with the "/proc" file system and to satisfy read and write requests. Among other possibilities, writes to a certain file allow the user to define new parameter settings, and reads from that file give the current settings back to the user.

5 CONCLUSIONS AND FUTURE WORK

This paper has described the design and implementation of a software testbed that enables us to prototype and evaluate various storage area network protocols.

We have organized our software into a number of interacting but independent modules for maximum flexibility. To date we have implemented versions for three protocols: SEP, iSCSI draft 3, and Fibre Channel. Table 1 shows a comparison of the observed data transfer rates for these protocols on 1 Gbps links for both Ethernet and Fibre Channel. This is the rate seen by a test application using the "memory-only" configuration of the target emulators.

Table 1 contains two lines for the SEP implementation, one labeled SEP-1500, the other labeled SEP-9000. The Alteon Acenic Gigabit Ethernet card we are using for these experiments provides an option that enables the sending and receiving of ethernet frames having a payload of up to 9000 bytes (Jumbo frames), instead of the standard limit of 1500 bytes. The line labeled SEP-1500 shows that with a 1500 byte payload the SEP implementation achieves a transfer rate of 21 Megabytes per second (MBps), but by increasing the payload to 9000 bytes the transfer rate increases 66% to 35 MBps, as shown in the line labeled SEP-9000. That such a significant improvement in bandwidth utilization can be attributed solely to increasing the packet size implies that there is a very high per packet overhead. Work is currently underway to determine which components of the protocol stack are the primary contributors to this overhead, and to suggest ways to reduce it.

This table also illustrates that moving the SEP-1500 target emulator from user-space to kernel-space produces a 10% improvement in the observed transfer rate from 19 MBps to 21 MBps. We assume this is due to the extra context switches and memory copying necessitated by the user-space implementation, but work is underway to specifically identify the exact cause.

Obviously the Fibre Channel protocol (labeled FC in Table 1) provides the best results, since it utilizes a hardware HBA on both the initiator and target. The SEP and iSCSI protocols both utilize the standard software TCP/IP stack provided in Linux.

Work is underway to upgrade the iSCSI implementation from draft 3 [5] of the proposed standard to the latest draft, currently draft 6 [7]. This involves a number of changes, including differences in the PDU format, in login and parameter negotiation sequences, task management, and security considerations.

We are also instrumenting our implementations to obtain more precise performance measurements of such important metrics as bandwidth utilization, latency and cpu utilization. We are interested in identifying the bottlenecks in the design and/or implementation of both our software and the proposed protocols, and in feeding this information back into the standardization process.

Finally we are developing an extensive suite of tests for conformance, interoperability, operability and stress testing. To do this requires modifying our existing prototype software so that it can “provoke” certain situations in order to observe the responses of systems under test. Our interactive configuration interface greatly enhances our ability to implement these tests, and we are planning to extend it to other areas of the kernel, such as the TCP/IP stack and the network drivers, in order to be able to create designated test situations, such as dropped packets, bad checksums, etc. We are designing these tests so they can be used on the hardware implementations of iSCSI that are soon to be released.

6 ACKNOWLEDGEMENTS

This work was supported in part by SUN Corporation and EMC Corporation.

REFERENCES

[1] Ravi K. Khattar, Mark S. Murphy, Giulio J. Tarella, and Kyell E. Nystrom. *Introduction to Storage Area Network, SAN*. http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg 245470.pdf, August 1999.

[2] ANSI X2.269-1995. *Information Technology, ’dpANS Fibre Channel Protocol for SCSI (SCSI-FCP)’*, 1995.

[3] dpANS T11.1/Project 1245-D/Rev 3.6. *Information Technology, ’Scheduled Transfer Protocol (ST)’*, January 2000.

[4] Andrew Wilson. *Internet Draft, ’The SCSI Encapsulation Protocol’ (SEP)*. http://www.ietf.org/internet-drafts/draft-wilson-sep-00.txt, May 2000.

[5] Julian Satran and et al. *Internet Draft, ’iSCSI (Internet SCSI)’*. http://www.ietf.org/internet-drafts/draft-ietf-ips-iSCSI-03.txt, January 2001.

[6] Nishan Systems. *An Overview of SoIP – Storage over Internet Protocol*. http://www.nishansystems.com/techlib/technical.html, 2000.

[7] Julian Satran and et al. *Internet Draft, ’iSCSI (Internet SCSI)’*. http://www.haifa.il.ibm.com/satran/ips/draft-ietf-ips-iSCSI-06.txt, January 2001.