Implementation of Offloading the iSCSI and TCP/IP Protocol onto Host Bus Adapter

Han-Chiang Chen
Industrial Technology Research Institute
jolly@itri.org.tw

Zheng-Ji Wu
Industrial Technology Research Institute
wujumper@itri.org.tw

Zhong-Zhen Wu
Industrial Technology Research Institute
zzwu@itri.org.tw

Abstract

The iSCSI storage can provide high speed, easy management and low cost advantages to satisfy the requirements of small-size IT departments. Usually, the servers running iSCSI protocols suffer from the heavy IO processing and therefore require an adapter card relieving the host CPU load. In this paper, we proposed an effective method which attempts to offload the processing of iSCSI and TCP/IP onto our designed host bus adapter card. The adapter card uses embedded Linux as operating system to perform iSCSI functions and hardware-accelerating CRC module to optimize the performance of iSCSI. The experimental results show that the iSCSI card can provide good performance with less host CPU load.

1. Introduction

With the increasing demand for network service, such as web-based Emails, enterprise database, network phone, audio/video and others, the need for information and storage equipments grows tremendously in recent years. Among data storage technologies, Storage Area Networks (SANs), providing considerable benefits in terms of performance, flexibility, scalability, security and centralized data management, have gained the attention of many IT professionals. The main idea of SANs is to interconnect servers and storage devices via a network topology so that SANs resources can be efficiently allocated, shared and managed. Because of the advantages of SANs, more and more companies, governments and academic organizations deploy SANs to achieve highly reliable data storing and data protection. Today, the frequently used transmission technology in SANs is Fibre Channel (FC) [1-5], an industry standard designed for high performance, low-latency, reliable block-level data transportation between servers and storage devices. Currently, its maximum data rate can be up to 4Gb/s. Despite the advantages of high speed and low latency, the infrastructure cost for FC SANs can be so expensive that most small- to medium-size IT departments cannot afford the expenses. As presented in [14], even in a small, eight-node FC SAN, this can amount to over $40,000USD in switches, cables, and host bus adapters for the servers.

In comparison to FC SANs requiring a new, different and expensive network infrastructure, IP SANs run on an existing IP network which can provide a significant cost saving in hardware acquisition, implementation, training and operation. To make IP SANs available, more and more IP storage protocols, such as Internet SCSI Protocols (iSCSI) [8], Fibre Channel over IP (FCIP) [3], Internet Fibre Channel Protocol (iFCP) have been proposed to enable storage data transportation over an IP network. Among these protocols, iSCSI [6-13] is a newly developed storage network transport protocol which encapsulates SCSI packets in TCP packets and then transmits the TCP packets using IP network. Fig. 1 illustrates an IP SAN environment running iSCSI protocol. The storage devices provide block-based data and the servers see the remote storage devices as if they were locally-attached SCSI drives via iSCSI protocol.

Usually, the processing of TCP/IP and iSCSI over Ethernet can be easily accomplished by software running on the CPU of the servers or storage devices. For example, a server running Microsoft iSCSI initiator software can access the information of an IP based storage device over Ethernet. However, this software based implementation may suffer from the problem of heavy CPU processing load on I/O and networking, especially when Ethernet technology moves forward 10Gps or above network speed. Therefore, the objective in the paper is to research an effective methodology which attempts to offload the TCP/IP and iSCSI processing from the server CPU to host bus adapters. To achieve this goal, we have developed a 1Gbps FPGA based iSCSI HBA card having embedded Linux as its OS and the iSCSI and TCP/IP protocol running on it. Also, the computation intensive components,

e.g. CRC module, are specifically implemented by hardware to enhance the system performance. In addition, the driver for the host to operate the card is ready to use. With the offloading of the iSCSI and TCP/IP, the experimental results show that the offloading method can obtain larger reduction on CPU utilization compared to the pure software implementation.

The rest of the paper is organized as follows. Section 2 introduces the iSCSI and offloading method in more details. In section 3, we discuss the hardware, software and drivers design for the iSCSI HBA. The experimental results are shown in Section 4 followed by conclusions.

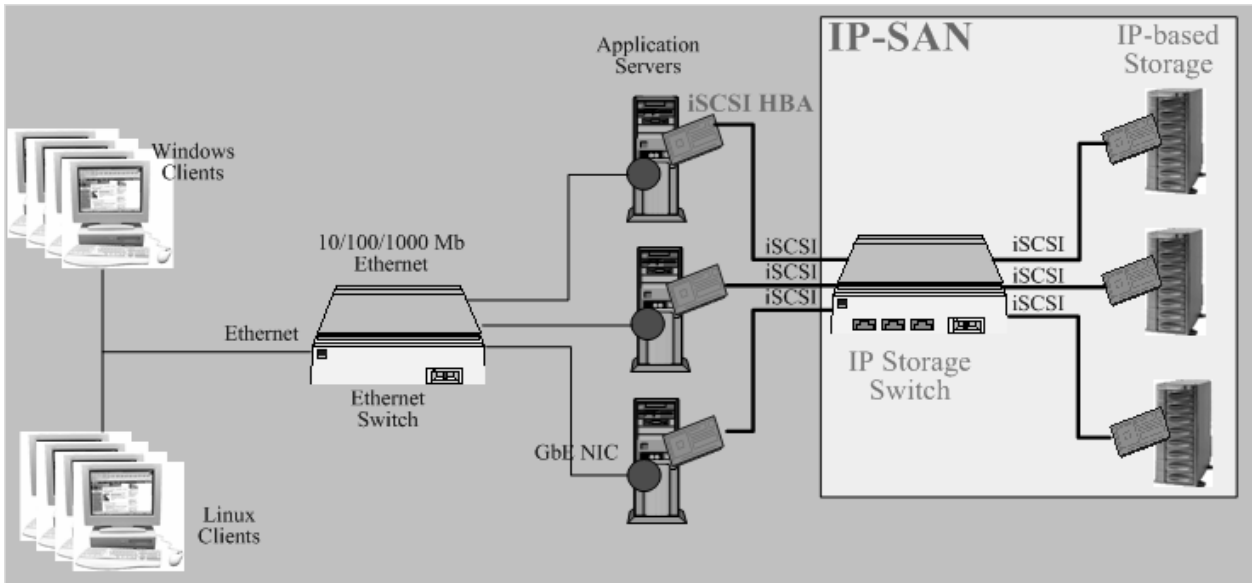

Figure 1. iSCSI Storage Network for IP-SAN.

2. iSCSI offloading methodology

2.1. iSCSI PDU types

Before we discuss the technique of offloading iSCSI protocol, we would like to define some commonly used terminology in the standard. In an iSCSI based SAN environment, an *initiator* (server) issues SCSI requests, which is encapsulated into iSCSI packets, referred to as *protocol data units*, *PDUs*. These PDUs are then transmitted over TCP/IP network to another remote *target* storage device through an iSCSI storage switch/gateway. The target returns the SCSI response/data by a form of iSCSI messages after analyzing the request iSCSI PDU received from TCP/IP network. A pair of initiator and target is called an iSCSI *session*.

iSCSI PDUs consist of an iSCSI header and iSCSI data, where the data length is stored within the iSCSI PDU header. The most commonly used iSCSI PDU types [8, 9] are:

- iSCSI Command/Response
- Data In/Out
- Ready to Transfer (R2T)
- Login Request/Response
- SNACK PDU

We briefly describe the scenarios of reading and writing data of an initiator from/to a target. The iSCSI Command PDU is used to transfer a SCSI command from an initiator to a target. If the initiator requires writing data to the target, it will first issue an iSCSI Write command to the target. Then, the target informs the initiator which part of the data to be transferred by sending the initiator an R2T PDU. When receiving the R2T PDUs from the target, the initiator sends the target the SCSI data encapsulated into one or more Data Out PDUs. On the other hand, if the initiator wants to read data from the target, when an iSCSI Read command received the target will send the requested data to the initiator in the form of iSCSI Data In PDUs. Upon completion of the entire data transfer, the target sends the initiator a SCSI Response PDU indicating successful completion of the command or any error condition detected. For each SCSI Command PDU there is only a corresponding SCSI Response PDU, but possibly multiple (or no) Data PDUs. iSCSI Data and Response

PDUs must be sent over the same TCP connection in which their corresponding iSCSI Command PDU was issued.

2.2. iSCSI offloading method

We now present the proposed offloading method. Basically, the main idea of offloading iSCSI is to offload the execution of network card driver, TCP/IP and iSCSI driver from the server CPU to Host Bus Adapter (HBA), handling all iSCSI and TCP/IP functions, such as Login, session establishing, and data transferring in the Full Feature Phase. We illustrate the idea by the Fig. 2. This will lead to several significant advantages. First, the offloading method greatly reduces the CPU load, and therefore can have more CPU resource reserved to other applications within the server. Moreover, due to the acceleration of computation-oriented modules by hardware, the performance can be further promoted in terms of throughput and speed.

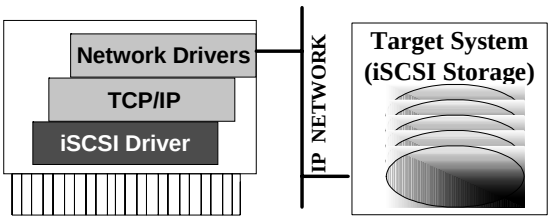

Figure 2. Offloading iSCSI protocol into HBA.

In the point of view of a server, this HBA card can be regarded to as a generic SCSI card but can remotely access the storage devices. The only thing that we need to do is to write a host bus interface driver and then can easily communicate with the HBA via the driver. The driver is responsible for delivering SCSI request messages to the HBA through host bus interface and replying the SCSI response messages from the HBA to upper layer applications.

The block diagram for our 1G CCL iSCSI HBA is shown in Fig. 3. The HBA architecture consists of several essential hardware components, including Xilinx Vitrex II PRO FPGA, RS-232 UART console, GPIOs, 64MB flash memory, 128MB SDRAM, PCI-X Bridge and one Gigabit Ethernet PHY. The Xilinx FPGA provides an embedded hardware platform to implement user-designed logic, and includes two PowerPC 405 CPUs allowing the embedded OS running on them. The system architecture for FPGA design will be discussed later. The kernel image of the embedded OS, stored in the flash memory, is responsible for dealing with the iSCSI and TCP/IP protocol. The PCI-X Bridge provides a set of control/status interface registers allowing the driver communicate with the embedded OS. The booting flow of the embedded OS is described as follows. When the machine powers on, the PPC405 CPU in the FPGA will execute the boot code for loading the kernel image from the flash memory to the SDRAM. Then the PPC405 un-compresses the linux kernel image and jumps to the start of kernel code to perform the functions of TCP/IP, iSCSI protocols and all hardware drivers. In the future, the processing of TCP/IP and iSCSI can be further hardwarized as TOE (TCP/IP Offloading Engine) and iSHB (iSCSI Hardware Block) to enable higher speed data transmission.

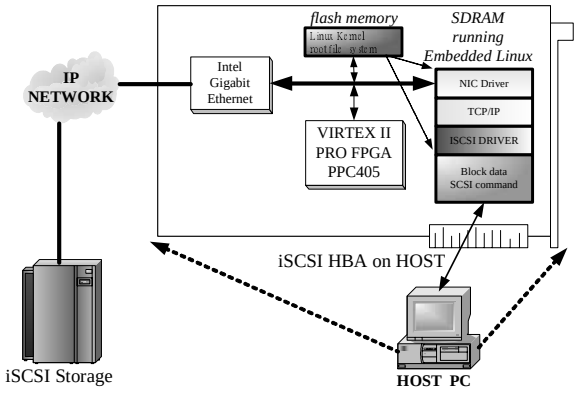

Figure 3. The Embedded system for processing iSCSI and TCP/IP.

3. iSCSI host bus adapter design

3.1. iSCSI adapter hardware design

For constructing an iSCSI IP SAN as in Fig. 1, the application servers and storage devices can be easily changed to iSCSI-compatible interface by plugging our iSCSI HBA onto peripheral interface slots. The complete design of the iSCSI adapter consists of two important issues: FPAG hardware design and the porting of embedded OS for the hardware. In the hardware design of the adapter, we rapidly develop a fundamental hardware platform using the embedded cad tools provided by Xilinx. Based on the Xilinx V2Pro FPGA platform, a commercial embedded linux MontaVista is then ported to manage the hardware resources and perform the functions of TCP/IP protocol stacks. Furthermore, we also must insert the compiled module of iSCSI initiator driver into the embedded linux kernel to support the iSCSI functions.

Now we present the hardware design of the adapter. The architecture of our iSCSI adapter is shown in Fig. 4. There are three digital clock managers (DCMs) generating different clock speed 66MHz, 100MHz, 125MHz and 300MHz respectively. The 66MHz clock is for the PCI devices, 125MHz clock for the gigabit Ethernet devices, 300MHz for CPU clock and 100MHz clock for other devices. There also include two types of bus structures: Peripheral Local Bus (PLB) and On-chip Peripheral Bus (OPB) specified by IBM CoreConnect standard. The PLB, running at 100MHz 64 bit data width, provides a high speed bus structure of interconnected PPC 405, SDRAM controller, 1 gigabit Ethernet MAC controller, BlockRAM controller, DMA/CRC controller and local-to-PLB bridge. The flash, UART and interrupt controllers are connected on a slower OPB bus, running 100MHz 32-bit width. The PLB and OPB devices communicate with via PLB-to-OPB Bridge.

The Xilinx tools can help us quickly develop a hardware platform and all hardware components except user-designed local-to-PLB bridge can be automatically generated. We must design the local-to-PLB bridge to allow the host applications to communicate with our adapter. The PCI host PLX9656 chip of C Mode transfers SCSI messages between PCI bus and 66MHz local bus, and our local-to-PLB bridge then moves the data from local bus to PLB bus. Note that this bridge contains the interface control and status registers and supports burst mode for C mode local bus and PLB bus for DMA data transmission.

The iSCSI adapter works as follows. Consider one scenario the host performs the SCSI Write request to the adapter. Support the OS has been loaded and executed in SDRAM as described in Section 2.2. First, the host driver checks the status of the adapter and transmits the SCSI Write command to the local-to-PLB bridge and the SCSI data to the SDRAM in DMA mode if the adapter is not under use. After the requested SCSI command/data are prepared in the adapter, the driver sets the SCSI Write flag of the control register to start the encapsulation of iSCSI packets and network transmission over an IP network. If the requested SCSI write is successfully completed, the adapter returns response messages by an interrupt notification. This is the similar case to the scenario of the SCSI read. We would like to mention that this initial-version adapter performs the iSCSI and TCP/IP functions based on the embedded OS, and as a result, still suffers from the drawbacks of traditional TCP/IP protocols, such as memory copy, pipeline flush, etc. We can implement hardware-accelerating modules such as TOE and iSCSI Hardware Block to promote the system performance in the future.

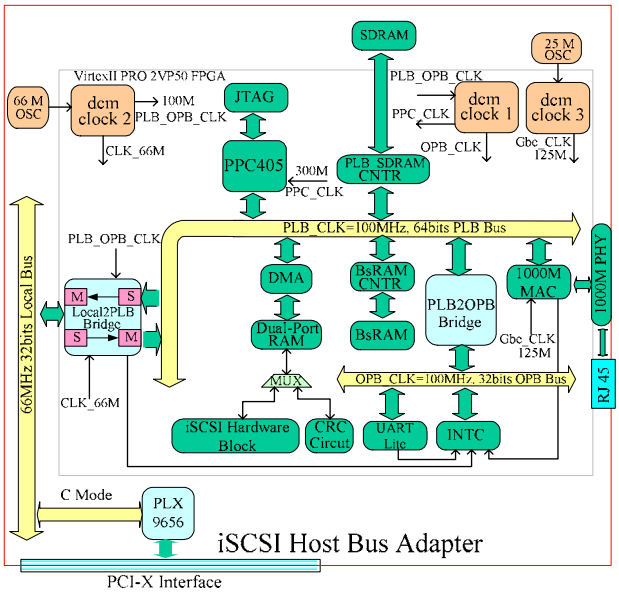

Figure 4. Architecture design of iSCSI adapter.

3.2. iSCSI HBA driver design

After the iSCSI adapter is successfully finished, we have to implement an adapter driver to allow the host to communicate with the adapter. As shown in Fig. 5, the HBA driver receives the SCSI read/write requests from the SCSI layers and commands the adapter to complete the requested work.

Here we present the flow of our driver design step by step. Before testing the driver, we must make sure the iSCSI adapter can work correctly. The flow of developing/testing the driver is as follows.

Step 1: Write a simple adapter driver to allow the host OS (e.g. Linux) to access the HBA SDRAM, PCI configuration registers, DMA control registers and interrupt register through PCI-X bus. We must assure that the driver can move data in the bi-directional with DMA controller and can receive an interrupt notification.

Step 2: Check whether or not the iSCSI HBA can successfully connect to remote iSCSI Target to do login and full feature work.

Step 3: Test whether or not the iSCSI HBA can receive an interrupt message from the host driver. If successful, this means the HBA can receive the commands from the host, and then can transmit the iSCSI packets over network.

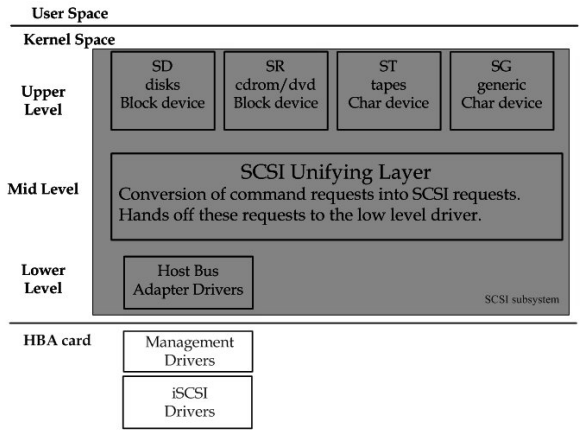

Figure 5. The iSCSI HBA Driver interface.

Step 4: Based on Step 1, we further insert the code for handling the SCSI requests from SCSI-mid level and interrupt service routine for receiving the SCSI response and data from the adapter.

Step 5: if the iSCSI Hardware Block and TOE were used to accelerate the iSCSI operation in the future, we must modify the iSCSI and TCP/IP protocols in the HBA to control the hardware circuit.

4. Experimental results

We have implemented our 1G CCL iSCSI HBA Prototype as in Fig. 6. The test environment includes an initiator plugging our iSCSI HBA and a target storage server based a Gigabit Ethernet network. The initiator platform is with Intel Celeron 1.7GHz CPU, 512M DDR SDRAM, and Gigabit Ethernet running RedHat Linux 9.0. The target platform is running RedHat Linux 9.0 based on Iwill DP533-S Server System, with Intel Xeon 2.8GHz CPU, Intel 82544 Gigabit Ethernet, 512MB DDR SDRAM, and 36GB Ultra SCSI hard disk. The target uses the UNH-iSCSI target driver [9] to perform software based iSCSI functions.

We compare three cases performing iSCSI protocol to show the performance and CPU overhead of the initiator platform. In the first case, the software based iSCSI initiator (UNH-iSCSI) [9] is adopted to perform iSCSI without plugging iSCSI HBAs. The second and third methods perform iSCSI initiator functions plugging the commercial Qlogic iSCSI HBA and our CCL iSCSI. Table 1 shows the results of data throughput and CPU utilization measured by Linux TIObench tool in these cases. For example, the software based method achieves roughly 16MB/s data rate with 20.9% CPU load and our method obtains 11MB/s with smaller CPU load 1.2%. The experimental result shows that the software-based method can obtain better throughput results with high performance CPU, but cause larger CPU load. Though the HBAs cannot have large improvement on data throughput under high speed CPU, they still have benefits on relieving the CPU load, especially in 10Gps network speed. On the other hand, the processing of TCP/IP and iSCSI by embedded OS still suffers from the inefficient data movement. Therefore, the HBAs require more sophisticated hardware architecture to promote the system performance.

Table 2 reports FPGA resource utilization of our 1G iSCSI HBA. Here one PPC405 CPU in FPGA is used to run the embedded linux, TCP/IP and iSCSI Initiator driver for offloading work. The other CPU and unused FPGA resources can be reserved for TOE design in the future work.

Table 1. Throughput and CPU utilization measured by Linux TIObench

		Pure Software	Qlogic QLA-4010C	1G CCL iSCSI HBA
Without CRC	Write (MB/s)	15.85	11.35	10.85
	Read (MB/s)	10.55	18.75	10.19
	Write CPU (%)	20.9	3.5	1.2
	Read CPU (%)	9.3	3.8	2.1
With CRC	Write (MB/s)	11.38	3.78	3.75
	Read (MB/s)	6.55	3.98	3.46
	Write CPU (%)	29.6	1.2	1.1
	Read CPU (%)	18.4	2.7	0.9

Table 2. Device utilization summary

XILINX VIRTEX II PRO FPGA XC2VP50	Used	Total	Utilization
External IOBs	215	692	31%

LOCed IOBs	206	215	95%
PPC405s	1	2	50%
RAMB16s	79	232	34%
SLICEs	5444	23616	23%
BUFGMUXs	7	16	43%
DCMs	4	8	50%

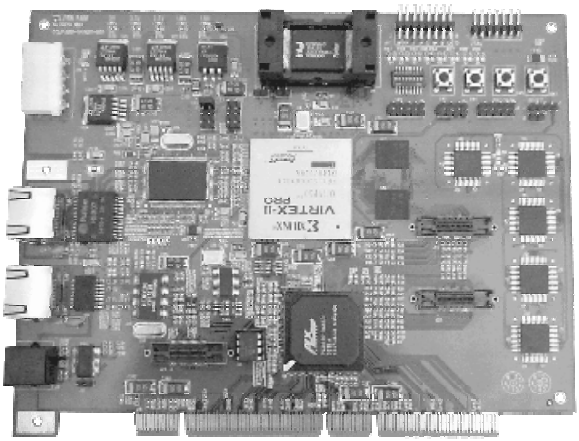

Figure 6. iSCSI HBA Prototype.

5. Conclusions

In this paper, we have introduced an effective method of offloading the processing of iSCSI and TCP/IP onto our 1G CCL iSCSI HBA. The iSCSI HBA can significantly reduce the HOST CPU utilization for application server and IP-based storage server. We also add the hardware-accelerating module (CRC) to enhance the system performance. With this iSCSI HBA, one can easily build an IP SAN with cheaper expense. In the future work, to enable high-speed data throughput, we should implement TCP/IP Offload Engine to further improve the processing of TCP/IP and iSCSI.

Acknowledgement

The authors acknowledge the contributions by many hardware and software engineers at CCLP300/ITRI who design the printed circuit board for 1G iSCSI HBA, port embedded linux and do final test. Our special thanks give to MOEA (Ministry of Economic Affairs) who support to the plan "IP based Storage Adapter Technology" B34BSP3100 in FY94.

References

[1] Fibre channel storage area network design for an acoustic camera system with 1.6 Gbits/s bandwidth Zhang Hong; Koay Teong Beng. Proceedings of IEEE Region 10 International Conference on Volume: 1 , 2001

[2] John R.Health and Peter J.Yakutis. High-Speed Area Networks Using a Fibre Channel Arbitrated Loop Interconnect. IEEE Network, March/April 2000, 51-56.

[3] “Fibre Channel Over TCP/IP (FCIP),” Internet draft, draft-ietf-ips-fcovertcpip-12.txt, 2002.

[4] Huseyin Simitci, Chris Malakapalli, Vamsi Gunturu XIOtech Corporation ”Evaluation of SCSI Over TCP/IP and SCSI Over Fibre Channel Connections” IEEE 2001

[5] Robert W.Kembel and Horst L.Truestedt. Fibre Channel Arbitrrated Loop, the Fibre Channel Consultane Series. Northwest Learning Associates, Inc. 2000

[6] Stephen Aiken, Dirk Grunwald, Andrew R. Pleszkun “A Performance Analysis of the iSCSI Protocol” Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS’03)

[7] Kalman Z. Meth, Julian Satran IBM Haifa Research Laboratory Haifa, Israel ”Design the iSCSI Protocol” Proceedings of the IEEE/11 the NASA Goddard Conference on Mass Storage Systems and Technologies (MSS’03)

[8] RFC 3720 - Internet Small Computer Systems Interface http://www.faqs.org/rfcs/rfc3720.html

[9] UNH-iSCSI consortium http://www.iol.unh.edu/consortiums/iscsi/

[10] X. He, Q. Yang, and M. Zhang, ”Introducing SCSI-To-IP Cache for Storage Area Networks”, International conference on Parallel Processing (ICPP’2002), August 2002.

[11] R. Hernandez, C. Kion, and G. Cole, ”IP Storage Networking: IBM NAS and iSCSI Solutions,”’ Redbooks Publications (IBM), 2001.

[12] Xubin He, Qing Yang, and Ming Zhang “A Caching Strategy to Improve iSCSI Performance” Proceedings of the 27th nnual IEEE Conference on Local Computer Networks (LCN’02)

[13] Kalman Z. Meth and Julian Satran, IBM Haifa Research Lab “Features of the iSCSI Protocol” IEEE Communications Magazine • August 2003

[14] Sanz, Inc., “Designing an IP Storage Network”, http://www.iscsistorage.com/a/wp2.htm