iSCSI Analysis System and Performance Improvement of iSCSI Sequential Access in High Latency Networks

Saneyasu Yamaguchi[1], Masato Oguchi[2], and Masaru Kitsuregawa[1]

[1] The University of Tokyo, 4-6-1 Komaba Meguro-Ku, Tokyo Japan
{sane,kitsure}@tkl.iis.u-tokyo.ac.jp
[2] Ochanomizu University, 2-1-1 Otsuka Bunkyo-ku, Tokyo Japan
oguchi@computer.org

Abstract. IP-SAN and iSCSI are expected to remedy the problems of FC-based SAN. iSCSI has a structure of multilayer protocols. A typical configuration of the protocols to realize this system is as follows: SCSI over iSCSI over TCP/IP over Ethernet. Thus, in order to improve the performance of the system, it is necessary to precisely analyze the complicated behavior of each layer. In this paper, we present an IP-SAN analysis tool that monitors each of these layers from different viewpoints. By using this analysis tool, we experimentally demonstrate that the performance of iSCSI storage access can be significantly improved by more than 60 times.

1 Introduction

The size of data processed by computer systems is increasing rapidly; thus, the large maintenance costs of storage systems have become one of the crucial issues for current computer systems. Storage consolidation using a Storage Area Network (SAN) is one of the most efficient solutions to this problem, and it has been implemented in many computer systems. However, the current-generation SAN based on FC has few demerits; for example, 1) the number of FC engineers is small, 2) the installation cost of FC-SAN is high, 3) the FC has distance limitation, and 4) the interoperability of the FC is not necessarily high. The next-generation SAN based on IP (IP-SAN) is expected to remedy these issues. The IP-SAN employs commodity technologies for a network infrastructure, including Ethernet and TCP/IP. One of the promising standard data transfer protocols of IP-SAN is iSCSI [1], which was approved by the IETF [2] in February 2003. However, the problems of low performance and high CPU utilization have been identified as the demerits of IP-SAN [3–5]. Thus, improving its performance and maintaining low CPU utilization [3, 6] are the critical issues regarding IP-SAN. In this paper, we discuss the performance issues of IP-SAN.

iSCSI is a protocol through which the SCSI protocol is transferred over TCP/IP; thus, the protocol stack of IP-SAN is "SCSI over iSCSI over TCP/IP over Ethernet." In order to improve the performance of iSCSI storage access, detailed information of all these layers is required because each of them may

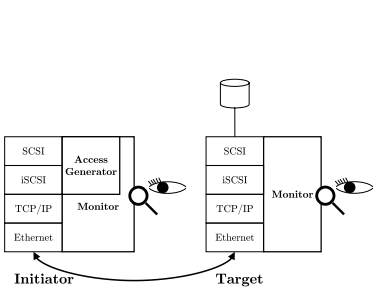

Fig. 1. Overview of the iSCSI analysis system

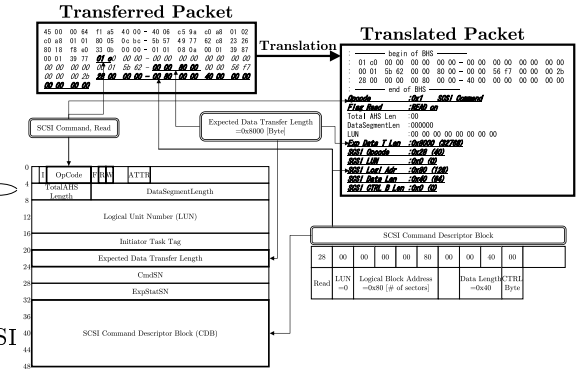

Fig. 2. iSCSI read command as an example of protocol translation

have an influence on the end-to-end performance. We propose an iSCSI analysis system and demonstrate that it can identify the causes for the decline in performance, which can be improved by resolving them.

The remainder of this paper is organized as follows. Section 2 introduces the iSCSI analysis system. Section 3 describes an actual application of this system and its role in improving the performance improvement. Section 4 mentions related work and compares them with our research. Finally, section 5 concludes this research and projects future work.

2 iSCSI Analysis System

In this section, we explain our iSCSI monitoring system. This system can monitor the internal states of each layer in an IP-SAN protocol stack. We developed an iSCSI analysis system by inserting the monitoring code into these layers. The functions of this monitoring system are as follows: 1) protocol translation (SCSI, iSCSI, and TCP/IP); 2) visualization of packet transmission with timeline; 3) monitoring behavior of the TCP flow control; 4) detection of packet loss; and 5) generation of the iSCSI storage access with a pseudo iSCSI initiator driver. An overview of our analyzing system is shown in Figure 1. We have discussed these functions in detail in the following subsections.

2.1 Protocol Translation

In this subsection, we describe the function of protocol translation in our analyzing system. With this function, the recorded iSCSI traffic data can be translated into human-readable text data. The iSCSI PDU format and an example of the translation are shown in Figure 2. In the case of "SCSI Command Read," the iSCSI PDU has a format as shown in the lower left part of Figure 2. An example of the hexadecimal dump of a transferred Ethernet packet with 100 byte data in an actual iSCSI storage access is shown in the "Transferred Packet" at the

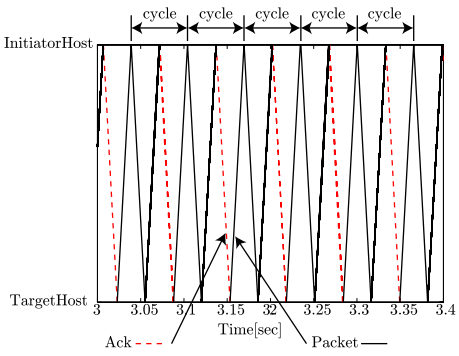

Fig. 3. Packet Transmission A

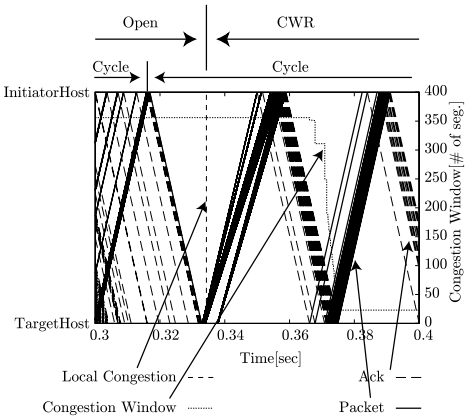

Fig. 4. Packet Transmission B

upper left part of Figure 2. The first 52 bytes written in a smaller font (bytes 1-52) construct the IP and TCP headers, and the remaining 48 bytes written in italics (bytes 53-100) construct the iSCSI PDU.

The translation function translates each of these fields into human-readable values and sections of the translated texts are shown in the upper right part of Figure 2[3]. In this case, it can be observed that the packet implies SCSI Command Read with a data length of 32 KB.

The SCSI CDB field with a size of 16 bytes is allocated in the range from bytes 33-48 in the iSCSI PDU. The fields in the range from bytes 43-48 in the PDU are padding data because the length of the SCSI CDB is 10 bytes in our experimental environment. The "Data Segment Length" and "Expected Data Transfer Length" in the iSCSI PDU are of one byte each while that of the SCSI CDB is 512 bytes. Consequently, both the "Expected Data Transfer Length" in the iSCSI header, which is 0x8000 [bytes], and the "Data Length" in the SCSI CDB, which is 0x40 [512 Bytes], are of 32 [KB]. As shown in this section, our analyzing system enables easy understanding of the transferred data using the iSCSI protocol.

2.2 Visualization of Packets Transmission

The transferred packets in the network can be visualized on a timeline with the visualization function. An example of visualized packet transmissions is shown in Figure 3. It shows the packet transmission of the iSCSI sequential read when the one-way latency time is 16 ms and the block size in the iSCSI PDU ("block size in the PDU" will be discussed in Section 3.3) is 32 KB. There are 5 cycles of "SCSI Command Read" iSCSI PDU and "Data-in" PDU with a data unit of 32 KB for the read command (this sequential read cycle is termed as "Seq. Read Cycle"). Figure 3 shows that a significant amount of idle time results while waiting for the network I/O. Further, the network utilization is considerably low.

[3] TCP/IP header and some fields in iSCSI PDU are omitted in Figure 2.

Figure 4 shows the packet transmission and the states of the TCP flow control of the iSCSI sequential read when the one-way latency time is 16 ms and the block size in the iSCSI PDU is 4 MB. The figure indicates the size of the TCP congestion window, transition of the state machine of Linux TCP implementation, and events that occurred in the TCP while implementing in kernel space. In this case, the figure shows the iSCSI PDUs of the "SCSI Command Read" and "Data-in" for the read command. It can be observed in the figure that the target attempted to transmit a large amount of data on receiving the read command leading to congestion of the local device. The TCP implementation then reduced the size of the TCP congestion window.

2.3 Monitoring TCP flow Control

The TCP implementation has a flow-controlling function. In most cases, since the TCP implementation functions in the kernel space, users are not allowed to monitor its behavior. The monitoring functions of the TCP flow control in our analyzing system enables the monitoring of this behavior in user space by adding a monitoring code into the TCP implementation. This is the only function that depends on system implementation in the iSCSI analysis system. Our TCP flow control monitoring system is implemented with Linux TCP implementation. With this function, the size of the TCP congestion window, various events in the TCP implementation such as congestion detection, and the state transition of the Linux TCP can be monitored from the user space. Linux TCP is implemented as a state machine. The state transition shown in Figure 4 (transition from "`Open`" to "`CWR`") is monitored by this function. The TCP flow control has a direct influence on iSCSI performance; thus, it is important to consider this function for improving iSCSI performance.

2.4 iSCSI Access Generation

The analyzing system also has a function for iSCSI access generation. The iSCSI driver is usually installed as a SCSI HBA driver. It is then driven by a generic SCSI driver and other OS implementations. Consequently, the users cannot create the iSCSI PDU based on their requirements. In the case of Linux, since a raw device and certain driver implementations divide a single block issued by the I/O command into multiple small blocks, the users cannot issue an iSCSI read command with a large block size. This limitation does not originate from the iSCSI protocol but from the OS implementation. The iSCSI access generation function enables the iSCSI access without the OS limitation affecting it. The generator directly establishes a TCP/IP connection with iSCSI target implementation using a socket API and transmits and receives the iSCSI PDU according to the iSCSI protocol. With this generator, the users can create the iSCSI PDU based on their requirements and can measure the performance of the iSCSI storage access.

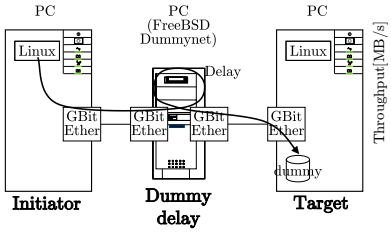

Fig. 5. Experimental setup

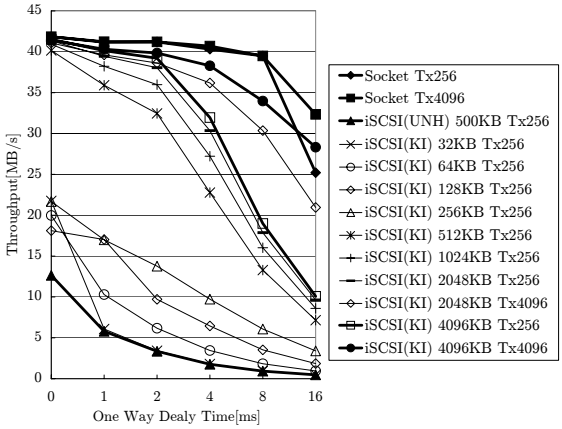

Fig. 6. Experimental result: iSCSI sequential read throughput

3 Performance Improvement

In this section, we present the performance evaluation of iSCSI sequential storage accesses in a long delayed network and the results of the analysis from our system. We also identify the causes of performance decline. In addition, we demonstrate how performance can be improved.

3.1 Experimental Setup

In this subsection, we describe an experiment conducted to evaluate the iSCSI performance and its environment.

We evaluated the performance of the iSCSI storage access in a heavily delayed network by performing the following experiment. The experimental system is shown in Figure 5. The iSCSI initiator and iSCSI target are constructed using PCs. A network delay emulator, which is constructed by FreeBSD dummynet [7], is placed between the initiator and the target. Further, the initiator and target establish a TCP connection over the dummynet in order that a simulated delayed network is realized between them. The initiator and dummynet are connected with a cross cable of 1 gigabit Ethernet. Further, the dummynet and target are also connected with a cross cable. Both the initiator and target are constructed by a Linux OS, and the dummynet is constructed by FreeBSD. The detailed specifications of the initiator and the target PCs are as follows: CPU Pentium 4 2.80 GHz; main memory 1 GB; OS Linux 2.4.18-3; and NIC gigabit Ethernet card Intel PRO/1000 XT Server adapter. The detailed specifications of the dummynet PC are as follows: CPU Pentium 4 1.5 GHz, main memory 128 MB, OS FreeBSD 4.5-RELEASE, and NIC Intel PRO/1000 XT Server Adapter $\times$ 2.

We employed the iSCSI implementation, which is distributed by the interoperability laboratory in the University of New Hampshire [8, 9]. This iSCSI implementation is termed as "UNH" implementation. The detailed specifications

and configurations of the iSCSI implementation used for the evaluation are as follows: iSCSI initiator and target: UNH IOL Draft 18 reference implementation ver. 3; iSCSI MaxRecvDataSegmentLength, iSCSI MaxBurstLength, and iSCSI FirstBurstLength: 16777215 bytes.

The initiator establishes the iSCSI connection with the target, and a benchmark software is implemented to measure the performance. The benchmark software iterates by issuing system call `read()` to the raw iSCSI device on the initiator OS. This call is single-threaded. The size of the TCP advertised window is 2 MB. The iSCSI target runs in memory mode. It can be regarded as a storage device with exceptional performance.

3.2 Basic Performance Measurement

"Socket Tx256" and "iSCSI (UNH) 500 KB Tx256" shown in Figure 6 are obtained by the experiment described in Section 3.1. We refer to this experiment as "Exp. A." The horizontal axis represents the one-way delay time between the initiator host and the target host. The vertical axis represents the measured throughput. "0 ms" implies that the network delay was not intentionally generated by the dummynet. In this case, the network delay with the physical device is approximately 100 μs. The "iSCSI (UNH) 500 KB Tx256" shown in the figure represents the throughput of the iSCSI sequential read, while the "Socket Tx256" represents the throughput of simple socket communication by means of which the initiator and target hosts establish the TCP/IP connection and transmit data through the socket API in the same environment. This simple socket connection is referred to as "pure socket." The block size specified at system call `read()` is 500 KB, and this block size is termed as "System Call Block Size." The maximum throughput of the "pure socket" is approximately 40 [MB/s] because this value represents the performance limit of the dummynet PC.

The results obtained indicate the following: 1) The iSCSI performance is severely low although considerably high-performance socket communication is achieved in the same environment and 2) The iSCSI performance decreases as network latency increases. In the following subsections, the reasons for the performance decline of the iSCSI are discussed.

3.3 Analysis of iSCSI Access

In this subsection, we present the analysis of the iSCSI storage access, as discussed in Section 3.2. Figure 2 shows the results of the protocol translation of the iSCSI PDU in the experiment. Figure 3 shows the visualized packet transmission in the experiment.

The translation results indicate that the "SCSI Command Read" PDUs with 32-KB "PDU Block Size" were issued even though the benchmark software issued the system call `read()` with a System Call Block Size of 500 KB. With several OS implementations, the issued system calls are transmitted to the network through the block or character devices, SCSI generic driver, iSCSI driver, TCP/IP implementation, and Ethernet device driver. Such calls are rarely transmitted to the network without being modified by these drivers. In other words,

the "System Call Block Size" is not always equal to the "PDU Block Size." It can also be observed that in our experimental environment, the block size of the issued system call `read()` is divided into multiple 32-KB block reads and transmitted to the target host.

The visualized figure (Figure 3) shows that when a large amount of time in the "Seq. Read Cycle" is expended in waiting for the network I/O, the network utilization is considerably low.

3.4 Discussion of Performance Decline

The analysis in Section 3.3 demonstrated that the System Call Block Size was divided into multiple small (i.e., 32 KB) "PDU Block Size"; thus, the network utilization is significantly low. The poor network utilization can be considered as the most critical reason for the decline in performance. As shown in Figure 3, the throughput of the iSCSI sequential read can be modeled as follows:

$$\frac{\text{PDU Block Size}}{4 \times \text{OneWayDelay} + \dfrac{\text{PDU Block Size}}{\text{LowerLayerThroughput}}} \tag{1}$$

A "Non-Idle ratio," which is the ratio of the time spent for sending data to the total "Seq. Read Cycle" time, can be modeled as follows:

$$\frac{\dfrac{\text{PDU Block Size}}{\text{LowerLayerThroughput}}}{4 \times \text{OneWayDelay} + \dfrac{\text{PDU Block Size}}{\text{LowerLayerThroughput}}} \tag{2}$$

In these models,`LowerLayerThroughput` represents the throughput of the pure socket. Consequently, the idle ratio with a 32-KB "PDU Block Size" is 84% for a one-way delay of 1 ms; 91%, 2 ms; 95%, 4 ms; 97%, 8 ms; and 98%, 16 ms. By using the analyzing system, we can identify that the reason for the performance decline is poor network utilization caused by a small block size.

The division of the block size is not only due to the limitation of the iSCSI protocol specification but also due to Linux OS implementation. As a result, we measured the performance of the essential iSCSI storage access, which is not restricted by any specific OS implementation, by using our iSCSI access generator. The experimental results are indicated in Figure 6 as "iSCSI(KI) nKB Tx256". The experiments were carried out in the same environment and termed as "Exp. B."

"iSCSI(UNH) 500 KB Tx256" in the figure represents the performance with the iSCSI sequential read access under the UNH iSCSI implementation. The System Call Block Size is 500 KB ("PDU Block Size" is 32 KB, as mentioned previously). The lines labeled as "iSCSI(KI)" represent the performance using our pseudo iSCSI initiator. Their block sizes are recorded in the labels. The "Socket Tx256" indicates the performance of the pure socket. "Tx" is mentioned in section 3.5.

The following can be obtained from the results. 1) The performance of the iSCSI increased significantly by increasing the read block size. 2) The performance obtained by the iSCSI access with a large block is not sufficiently high.

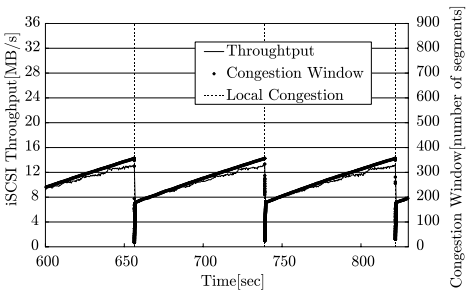

Fig. 7. Transitions of TCP Congestion Window (iSCSI)

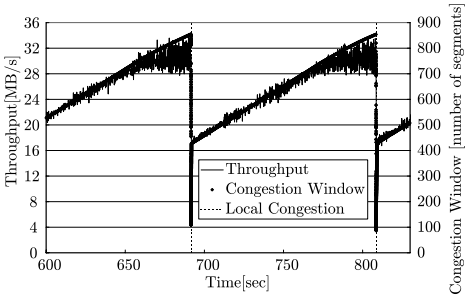

Fig. 8. Transitions of TCP Congestion Window (Socket)

In the case of 16-ms one way delay, the throughput of "iSCSI(KI) 4 MB Tx256" is 10.1 [MB/s], while that of "iSCSI(UNH) Tx256" is 0.47 [MB/s]. The performance improved more than 20 times when the block size was increased. However, the throughput of the pure socket was 25.2 [MB/s], and the performance of the iSCSI is less than half that of the socket.

3.5 Analysis without Block Division

In this subsection, we analyze the behavior of the iSCSI storage access with a large block size.

The transitions of the throughput and size of the TCP congestion window of "iSCSI(KI) 4MB Tx256" and the "Socket Tx256" for a one-way delay of 16 ms are shown in Figure 7 and Figure 8, respectively. First, we found that the obtained throughput and the size of the TCP congestion window vary synchronously. The TCP implementation restricts the output throughput below the $\frac{\text{CongestionWindowSize}}{\text{RoundTripTime}}$. Second, Figure 7 shows that local device congestion occurs when the congestion window size is approximately 350 segments and this size cannot be exceeded for the iSCSI access sequential read. However, the congestion window size was approximately 850 segments in the case of the socket communication. The burstness of traffic progressively weakens due to self-clocking of the TCP during the pure socket communication. On the other hand, in the case of the iSCSI access, the iSCSI driver is independently synchronized using the SCSI Command Read and SCSI Response at the iSCSI layer. This results in extremely bursty traffic that is generated when the target returns the "Data-in" PDU for the "SCSI Command Read" PDU, as shown in Figure 4. As a result, the traffic burstness persists. Therefore, the iSCSI traffic patterns easily cause congestion and the throughput is restricted by the TCP implementation.

In this case, the reason for the performance decline can be attributed to the congestion in the local device and TCP flow control resulting from bursty iSCSI traffic. The local device congestion occurs when the packet descriptors are depleted in the local network interface card. This congestion can be avoided by improving the tolerance of the NIC to bursty traffic by enlarging the buffer

size of the NIC device driver. The number of packet buffers in the device driver of the NICs used in this experiment environment (refer to Section 3.1) can be regulated from 80 to 4096. In "Exp. B" described in Section 3.4, they were set to the default value of 256.

The measured throughput with 4096 NIC device driver buffers are represented as lines that are labeled as "Tx4096" in Figure 6; this experiment is referred to as "Exp. C." In the labels in Figure 6, "Tx" represents the number of packets that the device driver can buffer. Further improvement in the performance was obtained by avoiding the local congestion. In the case of 4-MB block size and 16-ms one way delay, the performance improved 2.81 times. The performance in Exp. C improved 60.5 times that in Exp. A for a one-way delay of 16 ms. The performance of the pure socket, which can be considered as the performance limit of our experiment environment, was also improved by avoiding the local congestion. In Exp. C, the performance decline by adopting the iSCSI protocol was 12% below the system performance limit for a one-way delay of 16 ms, while it was 60% in Exp. B. Thus, a performance comparable to the system limit can be achieved in the iSCSI storage access.

4 Related Work

Several studies have presented the performance evaluation of IP-SAN using iSCSI [3, 4, 10–13].

Sarkar et al. [4] avoided the performance evaluation of the iSCSI. In particular, this study paid attention to the CPU utilization of the iSCSI storage access since it is extremely crucial for IP-SAN. They experimented with the performances of the iSCSI storage accesses using various block sizes in a LAN environment. The work demonstrated that the TCP/IP processing consumed considerable CPU resources. Further, they showed that the CPU utilization reached 100% at the peak throughput with block size of 64 KB.

Radkov et al. [13] presented a detailed comparison of the NFS and iSCSI. Further, the comparison is very broad in scope. Their discussion not only includes the performance and CPU utilization but also the number of network messages. Both the micro- and macrobenchmarks were executed in some configurations such as warm cache or cold cache and several network delays. It was shown that the iSCSI and NFS are comparable for data-intensive workloads, while the former outperforms the latter for meta-data intensive workloads.

These studies are obtained by executing various workloads outside the IP-SAN system. Consequently, these studies do not reveal accurate behaviors inside IP-SAN systems. Our work presents very exact behaviors inside the IP-SAN system including those in kernel space, for example TCP flow controlling. As far as we know, there is no published report discussing the performance of the iSCSI through the examination of the TCP/IP behavior, particularly the congestion window size and receive window size. This type of monitoring is a novel feature of our work. In addition, we have also identified the causes for the performance decline by employing the proposed system while the existing studies reveal the experimental results. This is also a novel feature of our work.

5 Conclusion

In this paper, we proposed and implemented an iSCSI storage access analysis system and demonstrated that the iSCSI performance can be significantly improved by detailed analysis using the proposed system and resolving the issues identified by the system. The proposed system can point out the reasons for the decline in performance. In our experiment, the performance improved more than 60 times and was comparable to the system limit performance.

In future work, our objectives are as follows: 1) to measure the performance of the iSCSI access using a real storage device; 2) to analyze not only single-threaded sequential read access but also write access, random access; and multiple access, and 3) to analyze the iSCSI storage access of some applications such as DBMS.

References

1. J. Satran et al. Internet Small Computer Systems Interface (iSCSI). `http://www.ietf.org/rfc/rfc3720.txt` , April 2004.
2. IETF Home Page. `http://www.ietf.org/` , 2004.
3. Prasenjit Sarkar, Sandeep Uttamchandani, and Kaladhar Voruganti. Storage over IP: When Does Hardware Support help? In *Proc. FAST 2003, USENIX Conference on File and Storage Technologies*, March 2003.
4. Prasenjit Sarkar and Kaladhar Voruganti. IP Storage: The Challenge Ahead. In *Proc. of Tenth NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2002.
5. Fujita Tomonori and Ogawara Masanori. Analisys fo iSCSI Target Software. In *SACSIS (Symposium on Advanced Computing Systems and Infrastructures) 2004*, April 2004. (in Japanese).
6. Jeffrey C. Mogul. Tcp offload is a dumb idea whose time has come. In *9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, May 2003.
7. L. Rizzo. dummynet. `http://info.iet.unipi.it/~luigi/ip_dummynet/` , 2004.
8. University of new hampshire interoperability lab. `http://www.iol.unh.edu/` , 2004.
9. iSCSI reference implementation. `http://www.iol.unh.edu/consortiums/iscsi/downloads.html` , 2004.
10. Stephen Aiken, Dirk Grunwald, and Andy Pleszkun. A Performance Analysis of the iSCSI Protocol. In *IEEE/NASA MSST2003 Twentieth IEEE/Eleventh NASA Goddard Conference on Mass Storage Systems and Technologies*, April 2003.
11. Yingping Lu and David H. C. Du. Performance Study of iSCSI-Based Storage Subsystems. *IEEE Communications Magazine*, August 2003.
12. Wee Teck Ng, Bruce Hilly Elizabeth Shriver, Eran Gabber, and Banu Ozden. Obtaining High Performance for Storage Outsourcing. In *Proc. FAST 2002, USENIX Conference on File and Storage Technologies*, pages 145–158, January 2002.
13. Peter Radkov, Li Yin, Pawan Goyal, Prasenjit Sarkar, and Prashant Shenoy. A performance Comparison of NFS and iSCSI for IP-Networked Storage. In *Proc. FAST 2004, USENIX Conference on File and Storage Technologies*, March 2004.