Performance Improvement of iSCSI Remote Storage Access through Optimization of Multiple Layers

Masato Oguchi, Reika Higa*
Ochanomizu University, Tokyo, Japan
Email: oguchi@computer.org, reika@ogl.is.ocha.ac.jp
*currently with NTT DOCOMO, INC.

Kosuke Matsubara, Takao Okamawari
Softbank Telecom Corporation, Tokyo, Japan
Email: {kosuke.matsubara, takao.okamawari}@tm.softbank.co.jp

Saneyasu Yamaguchi
Kogakuin University, Tokyo, Japan
Email: sane@cc.kogakuin.ac.jp

***Abstract*— iSCSI is one of the most popular protocols among the storage area network (SAN) working on an IP network. iSCSI is a standard to encapsulate a SCSI command into a TCP/IP packet, thus we can make an access to a storage system with commoditized IP devices only. In this paper, iSCSI sequential write access for a remote backup system is focused. It is well known that iSCSI experiences performance degradation under a high latency environment. The storage access by iSCSI has multiple hierarchical protocols. Due to this complicated structure, it is very difficult to identify the performance bottleneck that causes the degradation. Therefore, system tools which can analyze the complicated layered structure are required. We have developed system tools that enable us to monitor the hierarchical protocols. As a result, we have identified the cause of the iSCSI performance degradation problem. We then fixed the problem and confirmed that one can obtain the performance close to the theoretical limit.**

***Index Terms*—iSCSI, SAN, TCP, Kernel, Optimization, Performance**

I. Introduction

Recently, with the rapid spread of broadband networks and the improved technologies of computer system, a large volume of data is stored and managed in a storage system in many business fields. In addition, remote data backup is regarded as an essential system to protect important data from a natural disaster or a terrorist attack. High speed data access technology to a storage device over long distance is a key to realize the remote backup system.

However, due to rapid increase of a volume of data, the storage management cost is one of the most serious issues of storage systems. Storage Area Network (SAN) is a high-speed network that connects multiple storage devices to servers. Because SAN allows the storage to be consolidated and managed in a centralized manner, it is widely used in storage area for an efficient management of many storage devices. FC-SAN, which is widely used already, connects servers and storage with Fibre Channel.

Due to defects in FC-SAN including its hardware costs and a distance limitation (up to about 10km), significant barriers exist in the introduction of FC-SAN. For this reason, IP-SAN configured with inexpensive Ethernet and TCP/IP is introduced. One of the candidates for the technology is iSCSI [1], by which one can use SCSI protocol over an IP network. iSCSI encapsulates a SCSI command, which is widely used in Direct Attached Storage (DAS), within a TCP/IP packet and transports the volume of data between server (Initiator) and storage (Target). In the future, since the Gigabit and 10 Gigabit class lines are expected to be more popular by development of the Internet, iSCSI will be effective furthermore.

However, iSCSI has an extremely complicated structure, i.e. "SCSI/iSCSI/TCP/IP/Ethernet", as shown in Figure 1. In addition, since it transmits the burst data, the degradation of the performance is remarkable in the high latency environment. It is pointed out in previous papers that the iSCSI has a problem when it is used over long distance, i.e. high latency environment due to the complicated protocol stack [2,3]. To mitigate the problem, several methods to improve the iSCSI throughput were proposed [2-4]. In our previous work [3], we found that the obtained performance for iSCSI sequential write is far below than the theoretically expected value in the high latency environment.

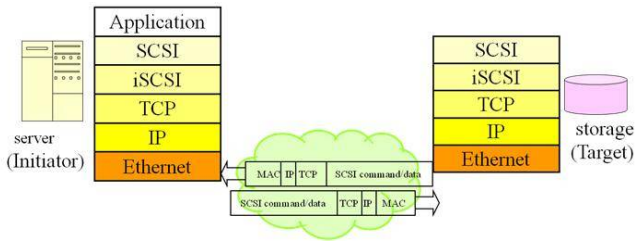

Figure 1. Configuration of iSCSI

doi:10.4304/jsw.8.3.538-546

Basically, iSCSI throughput cannot reach beyond the limit performance of TCP/IP which is laid at the lower layer. Thus, in iSCSI remote storage access, optimization not only in iSCSI layer but also in other layers is expected to improve iSCSI performance.

In this work, we have developed several tools to analyze the system to find the cause of the performance degradation. In the case of remote backup, the amount of writing data is more than that of reading. In addition, it should sometimes be assumed that only a non-customized system can be used for iSCSI Target. Thus in our research work, iSCSI software at Initiator is optimized. By the optimization, we have achieved extremely good performance of iSCSI sequential access in a high latency environment.

The rest of the paper is organized as follows. Section II describes a research background. Basic optimization of iSCSI acces is shown in Section III. Section IV describes preparation for analysis including an introduction of our system tools. Through Section V to VIII, the cause of the performance degradation is analyzed from various points of view. Finally, Section IX presents the conclusion.

II. BACKGROUND OF OUR RESEARCH

A. Experimental Environment

In our previous work [3], we applied various techniques, including parameter optimization for iSCSI, TCP, and network interface card, to improve the iSCSI write access performance under high latency environment. [4, 5] The schematic diagram of our experimental setup is shown in Figure 2. We have investigated a point to point iSCSI connection. Initiator and Target are workstations which have 1.6GHz Quad Core Intel Xeon, respectively. They are connected with Gigabit Ethernet and TCP/IP connection is established between them. As Target storage, SAS disks are used with RAID0 configuration. As an operating system, Linux 2.6.18-8 is used. As for iSCSI software, open iSCSI [6] and iSCSI Enterprise Target [7] are used. We have artificially inserted delay between Initiator and Target by using a network simulator.

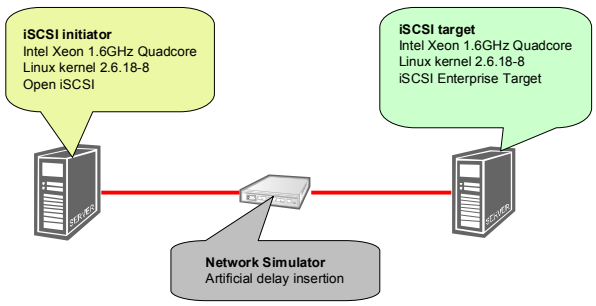

Figure 2. Schematic diagram of experimental setup

B. TCP Congestion Window Control Algorithms

TCP is used as transport layer in iSCSI. TCP packet transmission is controled by window size. There are two window sizes, that is, advertisement window and congestion window. Advertisement window is a parameter notified from a receiver to a sender, which informs the remained buffer size of the receiver. On the other hand, congestion window is a parameter controlled by a sender, which restrict the number of packets in order to avoid network congestion.

Basically, TCP congestion window is controlled by slow start phase when a packet transmission begins. This increases the size of congestion window as an exponential manner. As a result, the volume of traffic increases and congestion might be caused. In order to prevent the congestion, when the size of congestion window exceeds slow start threshold, the congestion window becomes to be controlled by congestion avoidance phase, in which the size of congestion window increases as a linear manner. After an error is detected, the size of congestion window decreases drastically. By repeating these phases, the behavior of the congestion window becomes saw tooth pattern usually.

The state transition of Linux TCP is shown in Figure 3. In Linux TCP, when the condition is normal, the congestion window is increased. On the other hand, the condition is judged as abnormal if an error occurs, and the congestion window is decreased in such a case. The causes of errors include Local Congestion (CWR) which means the buffer of sender's device driver is full, receiving duplicated ACK or SACK (Recovery), and detecting timeout (Loss). In addition, in the case of Linux TCP implementation, once the congestion window is set during a communication, it is not changed unless the larger volume of data than it is transmitted, and throughput becomes constant in this case.

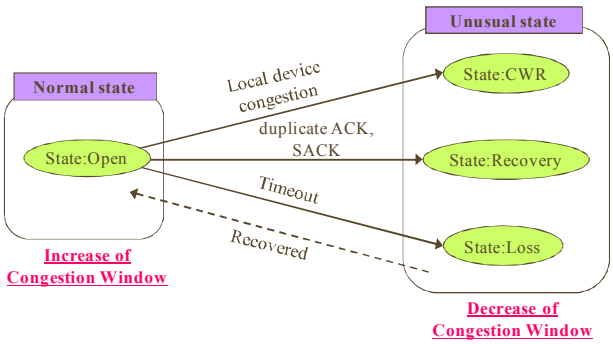

Figure 3. State transition of Linux TCP

There are various congestion window control algorithms designed for TCP. In Linux TCP, some of them are implemented, and the algorithm can be changed among them only with a command. They include Reno, Binary Increase Congestion control (BIC), Westwood, and Hamilton TCP (H-TCP).

Reno is a classic algorithm, and plenty of algorithms are designed based on Reno. Reno detects congestion with packet loss, and the available bandwidth is decided based on the value when a packet loss occurs. That is to say, when a sender receives three consecutive duplicated ACKs, this is judged as a packet loss, and the congestion

window becomes half. The congestion window increases one on every RTT, afterward. Thus, the congestion window becomes larger gradually, and falls sharply in Reno.

BIC is a default algorithm in our experimental environment. BIC executes binary search to find an available bandwidth, while general TCP congestion control performs linear search.

Westwood is designed for an environment in which a packet loss occurs frequently. This is developed based on Reno, optimized for wireless communications.

H-TCP is recommended suitable for a broadband and long-delay environment. This algorithm is designed to recover quickly to be the original state after congestion.

C. Process of TCP Transmission

In the case of TCP implemented in Linux OS, socket buffers which store the sending data are connected to a queue and wait to be processed as shown in Figure 4. Socket buffer cannot be freed until the time of receiving ACK.

Sending queue is a member of sk_write_queue in sock structure defined in the kernel source code. A pointer of sk_send_head points the socket buffer whose data will be sent next. In the socket buffer, a part between a queue head and sk_send_head is in a state whose data has been sent out but cannot be freed because ACK is not received yet. Socket buffers behind the point of sk_send_head are in a queue whose data will be sent out afterward. sk_send_head is shifted when each segment is transmitted. In this paper, the state of queue length (between the queue head and the tail) is discussed in the section VII.

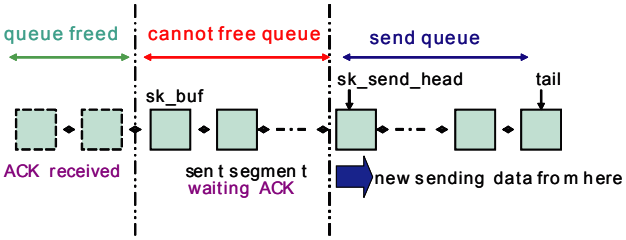

Figure 4. TCP Socket Buffer

III. BASIC OPTIMIZATION

A. iSCSI Layer Optimization

iSCSI layer has a lot of parameters[4]. Thay are exchanged between Initiator and Target during the negotiation phase. In our research work, we have optimized iSCSI parameters at first. The optimized parameters, compared with default settings, are shown in Table 1.

Table 1. Configuration of iSCSI

	Default	Optimized
Target side		
InitialR2T	Yes	No
ImmediateData	No	Yes
FirstBurstLength	65536	1048576
MaxBurstLength	262144	1048576
MaxRecvDataSegmentLength	8192	1048576
Initiator side		
node.conn[0].iscsi.MaxRecvDataSegmentLength	131072	1048576
node.session.iscsi.FirstBurstLength	262144	1048576

B. Ethernet Layer Optimization

In order to optimize iSCSI remote storage access further, NIC parameters of Ethernet are optimized. Intel PRO/1000PT is used as NICs in our experiment. The device driver of this NIC provides parameters for optimization. The optimized NIC parameter settings are shown in Table 2.

Table 2. Optimized NIC parameter settings

	Setting
Target side	
MTU	9000
Initiator side	
MTU	9000
Txqueuelen	3000
TxDescriptors	4096
FlowControl	Disabled

C. Congestion Window Control Algorithms

In order to investigate the effectiveness TCP congestion window control algorithms, several algorithms are compared in the case of socket communication. As TCP congestion window control algorithm, Reno, BIC, Westwood, and H-TCP are used for the comparison. The result of comparison shown in Figure 5. Measured time is 1000sec in this case.

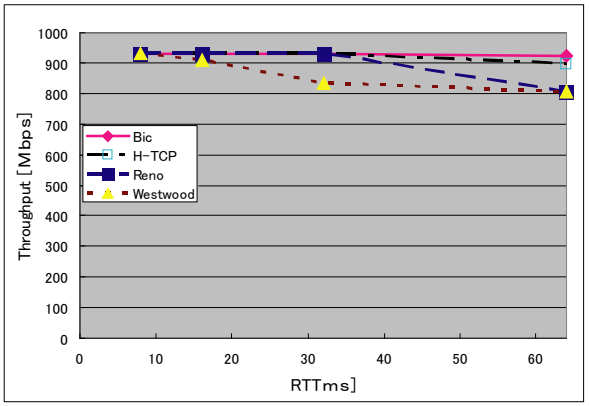

Figure 5. Throughput comparison using various algorithms (1000[s])

Figure 6 shows the result of the same experiment with 30sec measured time, in order to investigate the influence of the beginning phase of each algorithm.

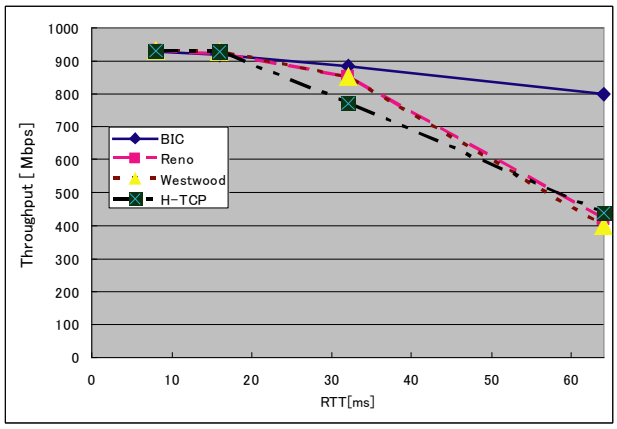

Figure 6. Throughput comparison using various algorithms (30[s])

According to Figure 5, the difference of algorithms can be observed in a high latency environment. Figure 6 shows this difference is mostly caused in the beginning phase of each algorithm.

Although the difference of throughput among congestion window control algorithms has been observed in the case of socket communication, throughput is almost the same with all algorithms in the case of iSCSI access. Therefore, we have used the default BIC algorithm in the following iSCSI experiments.

D. Result of Basic Optimization

Figure 7 shows the result of basic optimization, in which iSCSI layer optimization is most effective. The theoretical limit of the throughput is also given in the figure. Iperf [8] is used for the measurement of performance in socket access, and Network Recorder, protocol analyzer of ClearSight Network [9], is used for the measurement of performance in iSCSI storage access. In our experiment, the advertised window is set as the size enough for the communication.

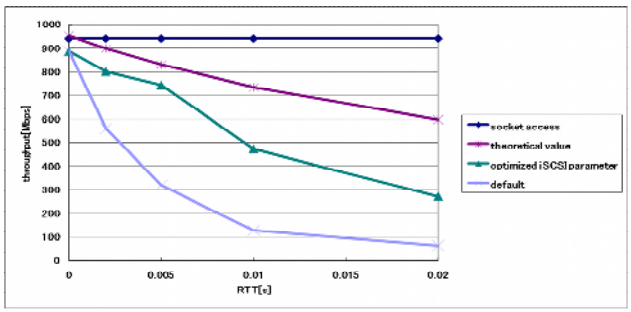

Figure 7. iSCSI sequential write throughput in high latency environment

One can see in Figure 7 that the iSCSI performance is dramatically improved by the techniques and the obtained throughput is in good agreement with the theoretical value if round trip time (RTT) is less than 5ms. However, even if the optimization is used, the throughput is getting significantly worse than the theoretical value once RTT exceeds the 5ms limit. We have not understand why there is the degradation in the high latency area. We will solve the question in the following sections.

IV. PREPARATION FOR ANALYSIS

A. System Tools for Analysis

To identify the exact cause of the degradation, we have developed several tools. We developed a kernel monitor tool, a packet monitor tool, and data analysis tool, as shown in Figure 8. The kernel monitor tool allows us to monitor kernel parameters at Initiator. The kernel parameters include TCP congestion window (CWND) size, advertisement window size, and socket buffer queue size. Generally, user programs cannot recognize the size of CWND and socket buffer queue because the size is a parameter controlled in a Kernel space of an operating system. Therefore we have inserted monitor functions in TCP source code and implemented a recording mechanism of TCP parameters within a Linux Kernel memory space, so that they are accessible from User space. With this mechanism, we can confirm TCP parameters by reading a special file for accessing Kernel memory space.

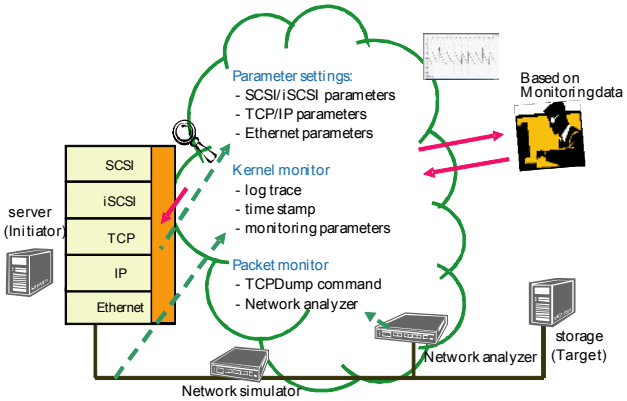

Figure 8. An overview of system tools for analysis

The packet monitor tool captures the packets received and sent out at the Initiator. The data analysis tool can process the information obtained from the kernel and packet monitor tools. In this way, we can directly or indirectly observe the procedures of various communication protocols that are executed during iSCSI transmission. In the followings, we will explain the analyses by means of the tools to identify the exact cause of the iSCSI throughput degradation under high latency environment.

B. Model of iSCSI Sequential Write Access

The bottleneck that causes iSCSI performance degradation in a higher latency environment has been examined as follows:

First, "dd" command has been used and the packets of iSCSI block access have been analyzed by using protocol analyzer. However, packets of various block sizes have been observed in the network in this case. Therefore, for the verification process of our model, "sg_dd" command is used [10]. While there is compatibility between "sg_dd" command and "dd" command, "sg_dd" command can access target with the specified size of the block at SCSI level in iSCSI access. By the reconfiguration of our kernel, "sg_dd" command enables us to access target with the size of the 4MB maximum. In accordance with the change, in iSCSI parameters, values of both FirstBurstLength and MaxBurstLength are set to 4,194,304. In our experimental system, we have measured throughput when the access block size is 2MB and 4MB, and RTT is 0-50ms.

Figure 9 shows the model of iSCSI write access sequence. Ta means data transfer time of sending from the first packet to the last one. Tb is time to prepare a packet to inform the end of writing on Target side to Initiator side. Tc is an interval until the following write is executed. Ta, Tb, Tc, and RTT have been measured by the network analyzer with each RTT set at the network simulator. We have analyzed of the factor of the performance degradation in higher latency by measuring each times.

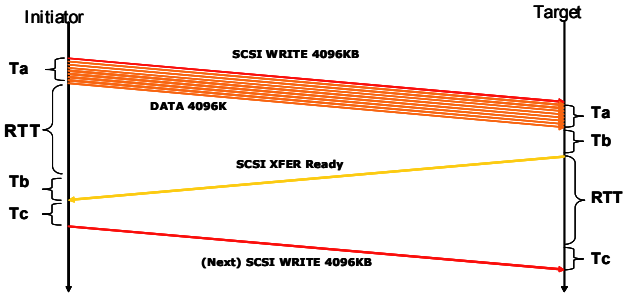

Figure 9. iSCSI write access sequence

According to a measurement result, Tb and Tc are constant mostly and RTT is almost the same with the value set at the network simulator. However, as Figure 10 shows, Ta increases in proportion to RTT. Therefore, the reason why the performance of iSCSI access decreases below the theoretical value is that the data transfer time of sending packets is proportional to RTT, which is constant generally. Thus, what happens in data transfer time should be examined in detail.

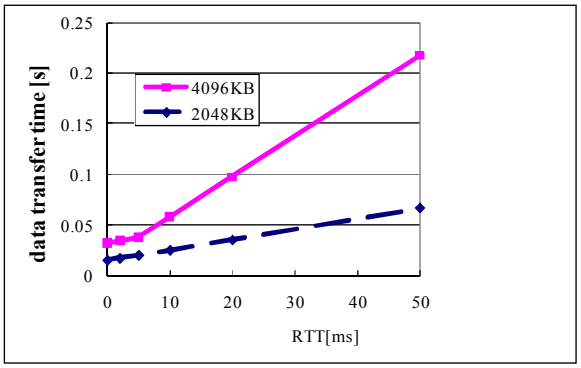

Figure 10. Data transfer time in iSCSI write access

V. ANALYSIS OF PACKET TRANSMISSION

As we described in the previous subsection, the factor of iSCSI performance degradation is owing to data transfer time in a higher latency environment. Thus, what happens during data transfer time is examined in detail by using network analyzer.

A. Analysis of Packets on Initiator

We have observed packets sent from Initiator to Target with the network analyzer. In this experiment, we have set 20ms RTT and 4MB iSCSI access block size. Figure 11 shows the result. The horizontal axis indicates time, and the vertical axis indicates the number of transmission packets. In this case, packets with "write10" command and "dataout" command are also shown in the graph for comparison. The vertical axis of "write10" and "dataout" command means nothing but only timing. According to Figure 11, it is observed that after packets are transferred consecutively within short time, the transmission of packet suspends temporarily. This behavior is not observed in normal socket access.

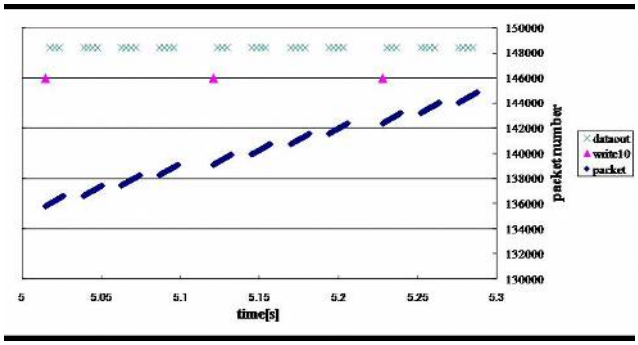

Figure 11. Analysis of packet on Initiator

One sequence of Figure 11 is enlarged and shown in Figure 12. From Figure 12, the following matters are observed: After packets are transferred consecutively within short time, the transmission of packet suspends temporarily. After constant time, the consecutive transmission of packets resumes again. Those intervals are about 20ms, which is equal to RTT.

When RTT is changed, the intervals are equal to RTT again. Therefore, the interval should be always equal to RTT.

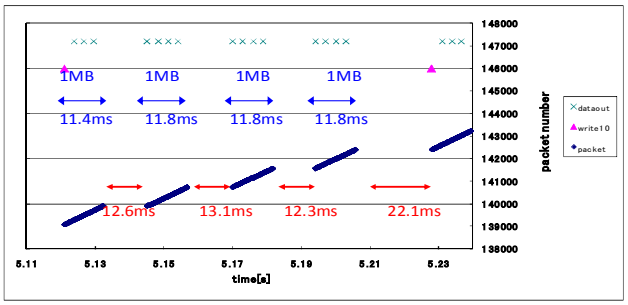

Figure 12. An enlarged part of Figure 11

B. Discussion of Packets Transmission Analysis

It has been observed that only TCP ACKs are received before the restart of data transmission. According to this observation, TCP ACKs should bring about the restart of data transmission.

We have set 20ms RTT and 4MB iSCSI access, and the packets of iSCSI write access have been analyzed with network analyzer. As a result, it is observed that after packets are transferred consecutively with short time, the transmission of packets suspends temporarily. The transmission of packets is intermittent. Therefore, the lack of CWND is naturally suspected to suspend the transmission of packets. Next, in the following section, CWND is examined with our kernel monitor.

VI. ANALYSIS OF WINDOW SIZE WITH KERNEL MONITOR

A. Analysis of TCP Advertisement Window

If the advertisement window of TCP is not large enough in high latency environment, TCP throughput is limited and the gap shown in Figure 12 might be created. However, it is revealed by the kernel monitor tool that the size of the advertisement window is 5MB. This means that the size of the advertisement window is wide enough. Therefore, we can conclude that the cause of the gap is not advertisement window.

B. Analysis of TCP Congestion Window

It is known that the values of throughput and CWND have a close relationship. We have used the TCP CWND monitor tool and "tcpdump" command, and observed the relationship between CWND and the amount of packets. In this experiment, we have set 20ms RTT and 4MB iSCSI access block size. Figure 13 shows the result.

In order to transfer 4MB data efficiently, about 3,000 CWNDs is necessary. However, Figure 13 shows CWND is 1,200. The measured CWND is smaller than the value which can transfer 4MB packets effectively.

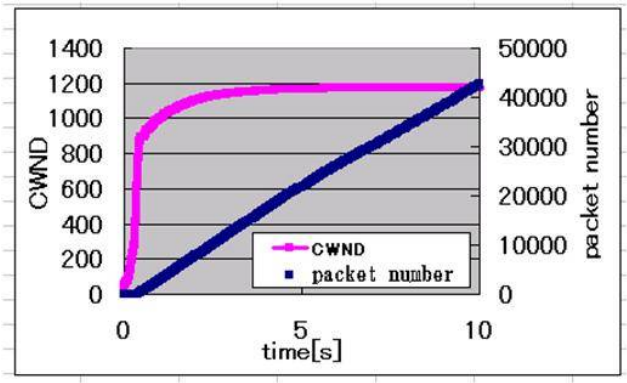

Figure 13. Analysis of CWND using kernel monitor

According to Figure 13, it seems the cause of the intermittent packet transmission is that the measured CWND is smaller than the value which can transfer 4MB packets effectively. However, from the reference of Figure 12 and 13, we can find this is not the cause.

The amount of transferred packets in one sequence is about 700 as shown in Figure 12, while the value of CWNDs is 1,200 as shown in Figure 13. Thus, the amount of transferred packets in one sequence is not enough size that consumes CWNDs. In addition, it is examined whether CWND really remains, by measuring the amount of packets on the fly. As a result, maximum volume of packets on the network is 1.1MB which is smaller than 1.8MB, the amount of packets that consumes 1,200 CWNDs. This means CWND is not exhausted.

After all, even though CWND is not exhausted, packet transmission is not continuous in the case of iSCSI access.

VII. ANALYSIS OF TCP SOCKET BUFFER

From previous analyses, the cause of intermittent packet transmission is neither CWNDs nor advertised windows. Therefore, we have analyzed TCP socket buffer next.

A. The Comparison of Queue Length in iSCSI Access and Socket Access

In the case of socket access, performance is kept to be equal even in a high latency environment. However, in the case of iSCSI access, performance is degraded in the same environment. Therefore, the behavior of queue of TCP socket buffer is analyzed by comparing the cases of iSCSI access and socket access using kernel monitor. RTT is set to be 20ms and 32ms, and access block size is set to be 4MB. The advertised window is set to be enough size to transfer data. In the case of measuring throughput of socket access, Iperf is used. In the case of measuring throughput of iSCSI access, "sg_dd" command is used.

The Figures from 14 to 17 show the comparisons of the number of transferred packets measured using "tcpdump" command with the queue length of socket buffer measured using kernel monitor in Initiator. The horizontal axis indicates time, the first vertical axis

indicates queue length, and the second vertical axis indicates the number of transferred packets.

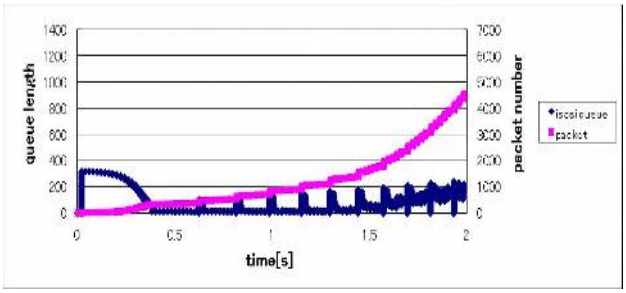

Figure 14. Socket buffer queue length for iSCSI access (RTT = 20ms)

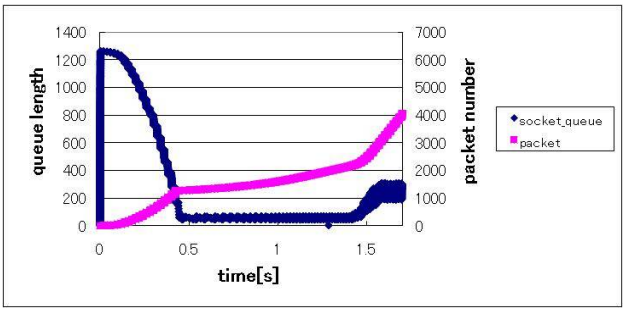

Figure 15. Socket buffer queue length for socket access (RTT = 20ms)

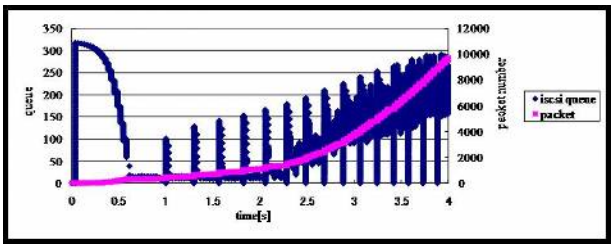

Figure 16. Socket buffer queue length for iSCSI access (RTT = 32ms)

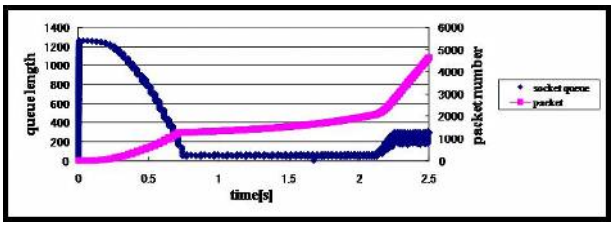

Figure 17. Socket buffer queue length for socket access (RTT = 32ms)

B. The Comparison of Queue Length when RTT = 20ms

From Figure 14, in the case of iSCSI access, it is confirmed that the maximum queue length is about 300 and it is kept between 0 and 300 in the steady state. According to Figure 15, on the other hand, it is confirmed that maximum queue length is about 1300 in the case of socket access, and it is kept between 200 and 300 in the steady state. Thus, an obvious difference is observed between the behavior of TCP socket buffer in iSCSI access and socket access.

C. The Comparison of Queue Length when RTT = 32ms

From Figure 16, in 32ms RTT also, it is confirmed that maximum queue length is about 300 in the case of iSCSI access, and it is kept between 0 and 300 in the steady state. On the other hand, from Figure 17, it is confirmed that maximum queue length is about 1300 in the case of socket access, and it is kept between 200 and 300 in the steady state.

Because of the same behavior is observed in 20ms RTT and 32ms RTT, we can conclude that the behavior of TCP socket buffer in the case of iSCSI access and socket access should be different.

D. Detailed Analysis of Queue in iSCSI Access

In this subsection, the behavior of queue of socket buffer in the case of iSCSI access is analyzed in detail. In Figure 18 and 19, we investigate it with ACK packets. In this case, the vertical axis of ACK means nothing but only timing.

Figure 18 shows the behavior of queue in iSCSI access in the steady state when RTT is 20ms. This is one sequence of 4MB iSCSI access. Received ACK makes queue start to increase and packet transmission restart. In this case, packet transmission suspends after growth of queue stops.

After suspend of packet transmission, ACK from Target is received and a part of queue is freed. Because of the increased queue, packets can be transferred again. However, queue comes to reach the limit again and suspend to send packet.

On the other hand, the state of queue in socket access is shown in Figure 19. Although the queue stops to increase when it reaches to 1,300, packets transmission is always continuous. This is considered to be the cause of performance difference between the case of iSCSI access and that of socket access in a high latency environment.

Therefore, in iSCSI access, available memory for TCP socket buffer in the kernel must be smaller than that of socket access. As a result, in iSCSI remote storage burst access, the waste of socket buffer makes the intermittent transmission of data packets.

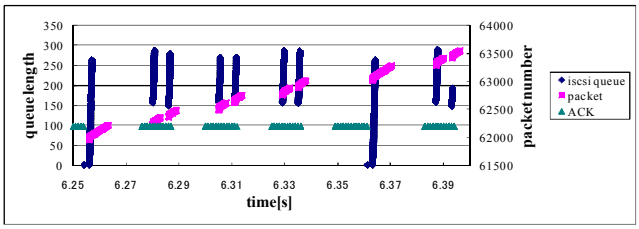

Figure 18. An enlarged part of Figure 14

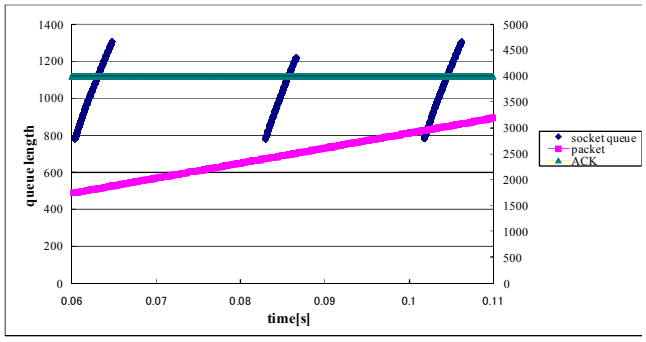

Figure 19. An enlarged part of Figure 15

VIII. ANALYSIS OF KERNEL CODE

A. An Overview of Kernel Code Analysis

With our system tool, it is possible to record a time stamp inside a kernel code during the execution of data transmission. That is to say, it is possible to recognize where in the kernel code is executed during data transmission and how long it takes to execute the specific part of the kernel code. Therefore, with this system tool, we have analyzed which part of kernel code is executed in both cases of iSCSI access and socket access, and where the cause of the suspension of data transmission is. After the discovery of the location of problem, we have inserted a software module for optimization of parameters, in order to resolve the problem and achieve good performance.

B. Identification of the Position of Problem

A part of Linux kernel code executed in data transmission is shown in Figure 20. In both socket access and iSCSI access, the execution on the kernel code leads to timeout, in the function of schedule() shown in Figure 20. In the case of socket access, schedule() function in line 1415 is executed, while in the case of iSCSI access, schedule() function in line 1439 is executed and it leads to timeout waiting, and it is finally awaked by ACK after RTT. Unnecessary timeout is also observed in iSCSI access.

Thus we have traced the kernel code after the schedule() function is called with the system tool. This is shown in Figure 21.

```
1400 fastcall signed long __sched schedule_timeout(signed long timeout)
1401 {
1402 struct timer_list timer;
1403 unsigned long expire;
1404
1405 switch (timeout)
1406 {
1407 case MAX_SCHEDULE_TIMEOUT:
1415 schedule();                                   Socket access
1416 goto out;
1417 default:
1425 if (timeout < 0)
1426 {
1427 printk(KERN_ERR "schedule_timeout: wrong timeout "
1428 "value %lx from %p\n", timeout,
1429 __builtin_return_address(0));
1430 current->state = TASK_RUNNING;
1431 goto out;
1432 }
1433 }
1435 expire = timeout + jiffies;
1436
1437 setup_timer(&timer, process_timeout, (unsigned long)current);
1438 __mod_timer(&timer, expire);                  iSCSI access
1439 schedule();
1440 del_singleshot_timer_sync(&timer);
1441
1442 timeout = expire - jiffies;
1443
1444 out:
```

Figure 20. Analysis of kernel source code

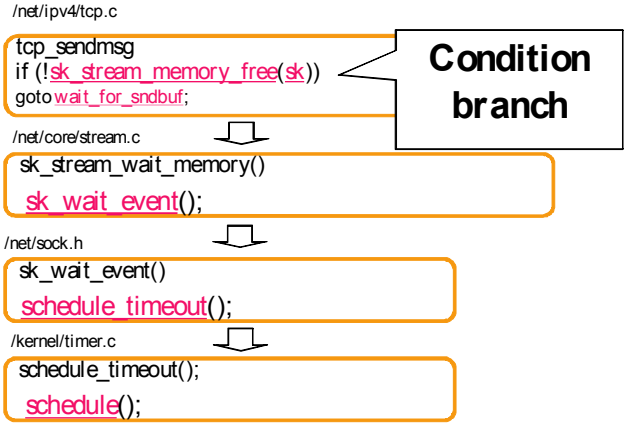

Figure 21. Kernel code trace

According to the trace log, the schedule() function is called as a result of a conditional branch, in which sk_stream_memory_free() function is included. As shown in Figure 22, this function just compares parameters of sk_wmem_queued and sk_send_buf and returns the larger one. Since we have identified the location of conditional branch that leads to timeout, we have inserted an appropriate software module for optimization of parameters, in order to avoid timeout waiting.

```
linux-v2.6.18.5/include/net/sock.h#L446
static inline int sk_stream_memory_free(struct sock *sk)
{
 return sk->sk_wmem_queued < sk->sk_sndbuf;
}
```

Figure 22. Conditional branch that leads to timeout waiting

C. Performance Evaluation

We have measured the throughput for the fixed iSCSI access, i.e. iSCSI sequential write with the "optimized socket buffer". The result given in Figure 23 clearly shows that the performance is greatly improved compared to the previous work. Figure 24 and 25 are the diagrams for the fixed iSCSI access that are corresponding to Figure 12 and 14 respectively. We have confirmed that the gap is shortened and the socket buffer size is not limited any more.

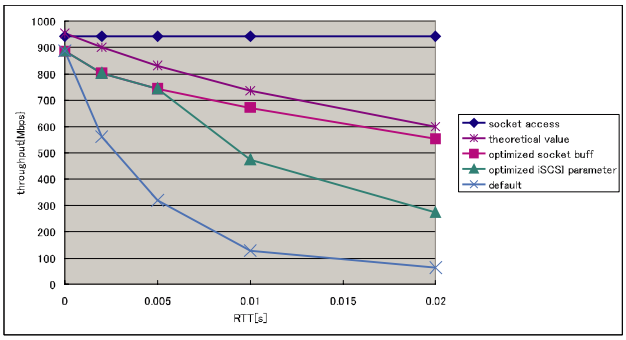

Figure 23. Improvement of data sending rate in fixed iSCSI access

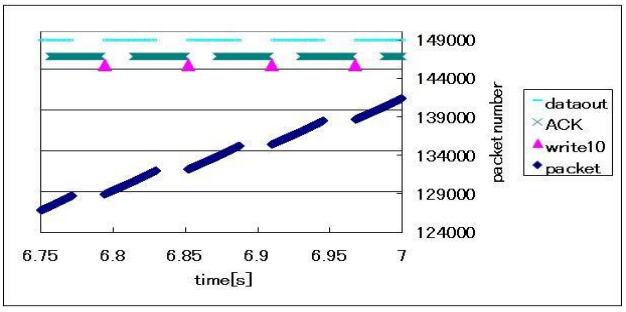

Figure 24. Data sending rate in fixed iSCSI access

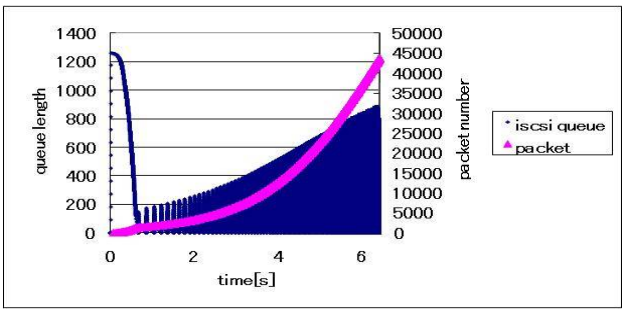

Figure 25. Scoket buffer queue length for fixed iSCSI access

IX. CONCLUSION

In this paper, we have developed system tools which enable us to analyze the detailed behavior of communication protocols by analyzing kernel parameters as well as packets dispatched and received. With the help of these tools, we identified the cause of the problem for iSCSI access over long distance. Thus, we have added a necessary software module and achieved a very high speed iSCSI sequential access, which can be easily applicable to various remote data backup services.

Note that the applicability of the developed tools is not limited to the iSCSI remote access. We believe that the tools are powerful enough to monitor and analyze a broad range of communication protocols for performance improvement as well as trouble shooting.

ACKNOWLEDGMENT

Part of this work is based on the research funded by Japan Science and Technology Agency in 2007 and 2008. We dedicate this paper to the late Mr. Takahiro Abe who had worked together and greatly helped us.

REFERENCES

[1] Satran, J. et al.: Internet Small Computer Systems Interface (iSCSI), IETF RFC 3720, http://www.ietf.org/rfc/rfc3720.txt, (2004)
[2] Toyoda, M., Yamaguchi, S. and Oguchi, M.: TCP congestion window controll on an iSCSI read access in a long latency environment, Proc. 16th IASTED International Conference on Communication Systemd and Applications (CSA 2005), pp. 170-175, (2005).
[3] Higa, R., Matsubara, K., Okamawari, T., Yamaguchi, S. and Oguchi, M.: Optimization of iSCSI Remote Storage Access through Multiple Layers, The 3rd International Workshop on Telecommunication Networking, Applications and Systems (TeNAS'2009) in conjunction with The IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA-09), pp.612-617, (2009).
[4] Shastry, Y., Klotz, S. and Russell R.: Evaluating the effect of iSCSI protocol parameters on performance, IASTED International conference on Parallel and Distributed Computing and Networks (PDCS 2005), Innsbruck, Austria, pp. 15-17 (2005).
[5] Cillendo E.: Tuning Red Hat Enterprise Linux on IBM e-server x-series servers, IBM Redpaper, (2005).
[6] Open iSCSI, http://www.open-iscsi.org.
[7] The iSCSI Enterprise Target project, http://iscsitarget.sourceforge.net
[8] Iperf, http://dast.nlanr.net/Projects/Iperf/
[9] ClearSight Network, http://www.toyo.co.jp/clearsight/product/analyzer.html
[10] The Linux sg3_utils package, http://sg.danny.cz/sg/sg3_utils.html