Implementation and Evaluation of Secure and Optimized IP-SAN Mechanism

Kikuko Kamisaka
Graduate School of Humanities and Sciences
Ochanomizu University
2–1–1, Otsuka, Bunkyo-ku
Tokyo 112–8610, JAPAN
kikuko@ogl.is.ocha.ac.jp

Saneyasu Yamaguchi
Department of Computer Science
and Communication Engineering
Kogakuin University
1–24–2, Nishishinjuku, Shinjuku
Tokyo 163–8677, JAPAN
sane@cc.kogakuin.ac.jp

Masato Oguchi
Department of Information Sciences
Ochanomizu University
2–1–1, Otsuka, Bunkyo-ku
Tokyo 112–8610, JAPAN
oguchi@computer.org

***Abstract*— Since IP-SAN consolidates distributed storage by using TCP/IP networks, it not only reduces management costs but also realizes a large scale storage system on Wide Area Network(WAN). In the case of accessing remote storage through open TCP/IP networks, security has a significant meaning. Although it is possible to employ IPsec encryption on IP-SAN, performance degradation due to inefficiency of encryption processing is concerned when IPsec is used.**

In this paper, we have proposed a method of optimization for encryption processing on IP-SAN. We have also implemented the middleware based on our proposed method, and evaluated the optimized IP-SAN system in regard to sequential write access.

I. Introduction

With the rapid spread of broadband networks, a large volume of data is stored and managed in a storage system in many business fields recently. 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)[1] is a high-speed network connecting multiple storage devices to a server. Because SAN allows the storage to be consolidated and managed in a centralized manner, it is widely used in storage area for efficient management of many storage devices.

Current generation SAN based on Fibre Channel (FC) technology uses high-speed and dedicated networks. Figure 1 shows an overview of FC-SAN. Due to defects in FC-SAN including its hardware costs and distance limitation (until about 10km), there are lots of barriers to the introduction of FC-SAN. In contrast, IP-SAN, the next generation SAN, has emerged recently to make a storage system more reasonable and manageable. IP-SAN is generally established using standard TCP/IP protocols and Ethernet instead of FC and its dedicated protocol of SAN. Figure 2 shows an overview of IP-SAN. Since IP-SAN hardware is inexpensive and there are many engineers for IP-SAN, it can reduce introduction and operational costs drastically. Moreover, IP-SAN provides seamless integration with existing IP networks, it is possible to realize a large scale storage system on a wide area SAN, in which a long distant backup can be performed for disaster planning. On the other hand, since IP-SAN uses TCP/IP networks, issues of low transmission a speed and high CPU load are pointed out[2]. An Internet SCSI (iSCSI) protocol[3], ratified by the IETF in April 2004, is expected to become a dominant IP-SAN protocol in the near future. iSCSI is a block-level data transfer protocol, in which each SCSI Command is

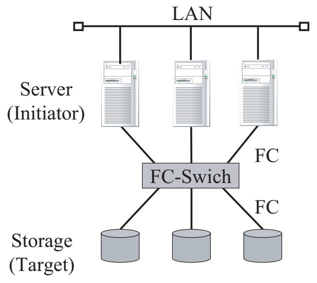

Fig. 1. FC-SAN

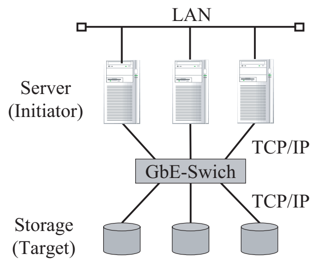

Fig. 2. IP-SAN

! !! !! !! !! !! !! !! !! !! !! !! !! !! !

encapsulated into TCP/IP packets and transferred between a server (initiator) and storage (target) over IP networks.

When storage is accessed through the IP-SAN, using open TCP/IP networks brings new challenges as a security concern. Although we can employ IPsec encryption on IP-SAN, performance degradation caused by sequential encryption processing of IPsec should be concerned in a data-intensive application. Because a trade-off exists between security and performance in regard to IP-SAN, it is necessary to employ a well-balanced data access scheme for a storage system.

We paid attention to inefficiencies in IPsec sequential encryption processing. In this paper, we have proposed the method of optimization for more efficient encryption processing compared with traditional method using IPsec. Moreover, we have implemented the IP-SAN system based on our proposed method, and evaluated our IP-SAN system in regard to sequential write access in a long-latency environment. We have also analyzed the experimental results using throughput modeling. As a consequence, our implemented system optimized for encryption processing has achieved better performance and proved to be more efficient than using IPsec on a long-latency network.

II. Related Work

Several studies about performance evaluation or implementation of IP-SAN have been presented until now. Ng et al.[4] have early studied performance of SCSI over IP and analyzed it in great detail. Sarkar et al.[5] have compared iSCSI software with hardware implementation such as TCP Offload Engine (TOE) and Host Bus Adapter (HBA). As a consequence, although such dedicated hardware is effective for reducing the CPU load, it does not achieve better throughput than that of the software implementation. Aiken et al.[6] have presented

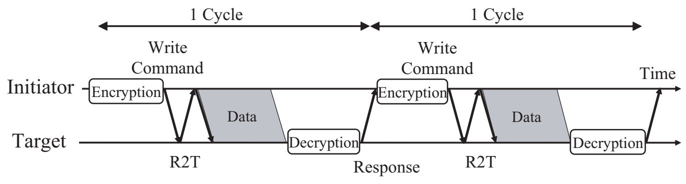

Fig. 3. iSCSI sequential write access with encryption in the upper-layer

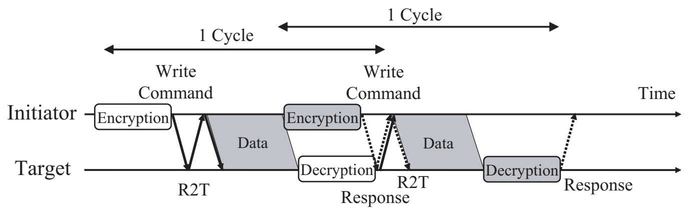

Fig. 4. iSCSI sequential write access with optimization of encryption processing

performance evaluation of iSCSI software on a network with latency. In SAN and WAN environment, they have shown that the iSCSI software implementation achieves almost the same performance with that of FC in the case of a large block size. In this paper, because the focus of our research interest is not only reduction of the CPU load but also improvement of overall performance, we have used software implementation of iSCSI. Gauger et al.[7] have presented a concise modeling and performance evaluation of the iSCSI protocol in TCP/IP-based MAN and WAN networks.

In a security-related work, Gibson et al.[8] have evaluated Network-Attached Secure Disks (NASD). They have proposed and implemented a secure Network SCSI mechanism with dedicated hardware and protocols. Tang et al.[9] have compared IPsec security schemes with SSL using iSCSI software implementation. They mentioned that SSL throughput outperforms that of IPsec in a large block size although the IPsec throughput is higher than that of SSL in a small block size. In addition, they have shown the CPU load of SSL is higher than that of IPsec. Joglekar et al.[10] have proposed two techniques for improving the performance of iSCSI protocol. Their methods are optimization of Cyclic Redundancy Codes (CRCs) are being calculated and iSCSI interacts with the TCP layer. Many papers about IP-SAN or iSCSI performance evaluation are presented until now. However, security techniques such as encryption for improving performance using iSCSI are not discussed.

III. iSCSI Write Sequence with Optimization for Encryption processing

A. *Issues of Applying IPsec on iSCSI Networks*

In order to transfer data to and from the storage securely on an iSCSI network, iSCSI can employ IPsec that offers strong encryption and authentication functions for IP packets. However, the encryption processing triggers performance degradation when mass volume of data should be transferred. Specifically in a long-latency environment, ACK or a SCSI Command takes a long time until it arrives at the other machine. Moreover, IPsec is implemented in IP layer located on the lower-level. If we try to improve the performance of IPsec encryption processing, IP and other codes inside a kernel of operating systems are required to be modified.

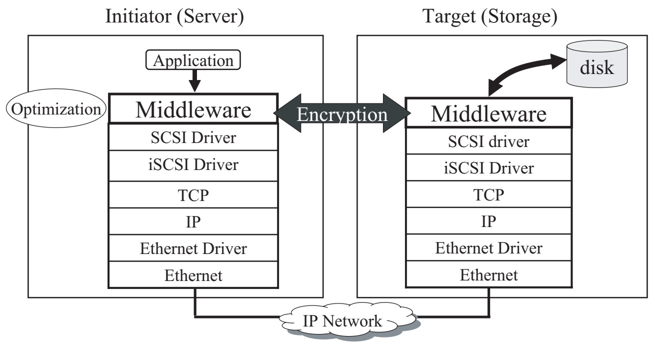

Fig. 5. Our proposed model

B. *Optimized Method for Encryption Processing in the Upper-layer*

We have proposed a method of optimization for encryption processing in the upper-layer instead of using IPsec. Our proposed model is depicted in Figure 5. The iSCSI system is a layered structure consisting of SCSI/iSCSI and TCP/IP. The encryption and optimization are performed in the middleware on top of SCSI layer in our proposed model.

Figure 3 depicts an example of iSCSI sequential write access in the case of encryption in the middleware located on the upper-layer without optimization. First, data is encrypted in the upper-layer and a SCSI Write Command is issued in the SCSI layer of the initiator. Next, a Ready to Transfer (R2T) Command is sent from the target to the initiator when the target becomes ready. After the initiator receives the a Ready to Transfer (R2T) Command from the target, the encrypted data is transferred. At the target, data is decrypted in the upper-layer at the target and written to the target's disk. Finally, the Response Command is sent back to the initiator in the SCSI layer, and a write access cycle with encryption is terminated. The benefit of executing the encryption processing in the upper-layer is that the performance improvement methods can be applied without modifying the implementation of IP layer. Thereby, our proposed method for optimization can be applied easily, so as to improve the performance of secure IP-SAN.

On the other hand, the method optimized for encryption processing is depicted in Figure 4. This method is realized by encryption pre-processing performed in the middleware, as shown in Figure 5. In a sequential access cycle without optimization, while one machine is encrypting or decrypting, waiting time for communications exists at the other machine, as shown in Figure 3. With the optimization of encryption processing in the upper-layer, it is possible to encrypt the next data consecutively during waiting time for communications, as shown in Figure 4. Overlapping the iSCSI sequential access cycle by encryption pre-processing assures effective use of CPU availability, so that the system performance improvement is expected.

IV. Implementation of Optimized Method in Our Middleware

In this paper, we have implemented the method optimized for encryption processing in the upper-layer as the middleware. In the implementation, we have used an open source operating system, Linux as the initiator and the target. As iSCSI implementation also, we have selected open source software, in consideration of low cost and source code availability. Reference implementation developed by the University of

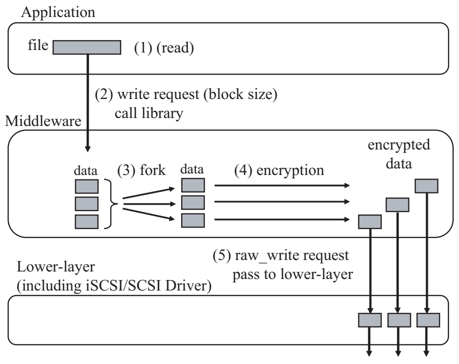

Fig. 6. Middleware in the initiator

New Hampshire InterOperability Laboratory[11] is used for an iSCSI driver.

A. Implementation Details in the Initiator

Our middleware in the initiator has a function of encryption and optimization, implemented on the upper-layer. The middleware is constructed with library functions called from the application. As implementation of encryption, we employ symmetric-key cryptographic algorithm, Triple Data Encryption Standard (3DES), used in IPsec by default. The encryption is executed by calling an encryption function in *libdes* library used in IPsec.

Figure 6 shows our middleware implementation in the case of sequential write access using a raw device of the initiator. As shown in Figure 6, (1) first, a file is read in the benchmark application in the initiator. (2) Next, a write instruction for each data segment with a defined block size is issued as a single process in the application. (3) The data segments are passed from the application to the middleware, and the process that receives the segments in the middleware forks into multiple images in order to perform processing concurrently. (4) Each process calls an encryption function in *libdes* library concurrently. (5) A write request for each encrypted data segment is issued to a raw device in the initiator in parallel. Thereby, the next data segment is encrypted by a forked process in the middleware at the initiator during waiting time; e.g., during the data is decrypted at the target side. Our implemented middleware allows pre-processing of encryption consecutively, so that overall processing time is reduced.

B. Implementation Details in the Target

In contrast, our system in the target does not implement the optimization function. Only the decryption function is implemented in the upper-layer. Our middleware is implemented as a kernel module on the top of SCSI layer as shown in Figure 5, the function of decryption in the upper layer is independent of the existing SCSI layers. Thereby, this enables us to apply the method of performance improvement to an existing system easily without modifying IP and other codes inside a kernel.

Figure 7 shows our middleware implementation in the case of sequential write access. After encrypted data segments are transferred to the target, (1) first, a write request for each incoming data segment is issued at iSCSI/SCSI device driver. (2) Next, each data segment is decrypted in our kernel module. (3) Decrypted data segments are passed to the *handle_cmd* function at iSCSI/SCSI driver and (4) they are written as a

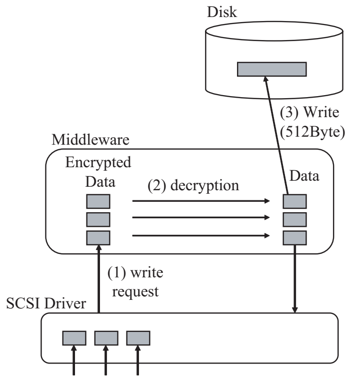

Fig. 7. Middleware in the target

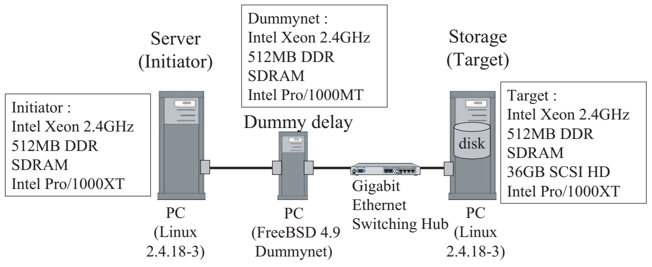

Fig. 8. Experimental setup

plain text in the target's disk sequentially. Thus, our implemented kernel module decrypts data in the upper-layer by intercepting a Write Command at iSCSI/SCSI device driver.

V. Experiment with Our Implemented IP-SAN System

In consideration of an importance of write access in IP-SAN, we have experimentally evaluated sequential write access in our implemented IP-SAN system. Since IP-SAN is expected to be used in a long-distance environment for disaster planning, we have evaluated it in a long-latency environment. Figure 8 shows an experimental setup. A network delay emulator, Dummynet[2][12], is inserted between the initiator (server computer) and the target (storage). One-way delay time between the initiator and the target is changed from 1ms to 64ms. A receiver's TCP window size is 8MB, which is large enough for this environment. In our experiments, FreeS/WAN for Linux[13] is used for a comparison as IPsec implementation. IPsec is set up with a transport mode that encrypts a host-to-host communication, and an ESP protocol is used. Because the benchmark software issues system calls to a raw device rather than a file system, the issued system calls are always transmitted to the SCSI layer in the target side without any cache hit in the initiator side.

A. Evaluation of Encrypted Write Access Throughput

We have evaluated the throughput of sequential write access in our system by comparing it with the performance of IPsec. The number of forked processes for the encryption pre-processing is changed from two to ten. In the rest of this paper, "OP-2", "OP-4", "OP-6", "OP-8" and "OP-10" stand for the number of forked processes for pre-processing in our IP-SAN system.

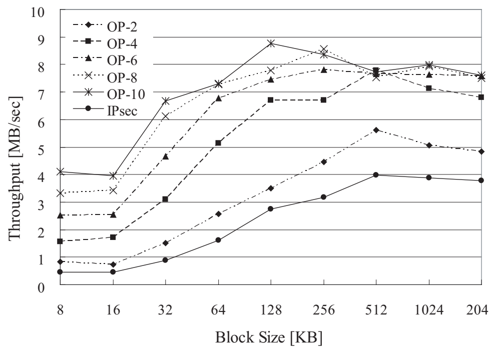

Fig. 9. Throughput on 8ms latency network

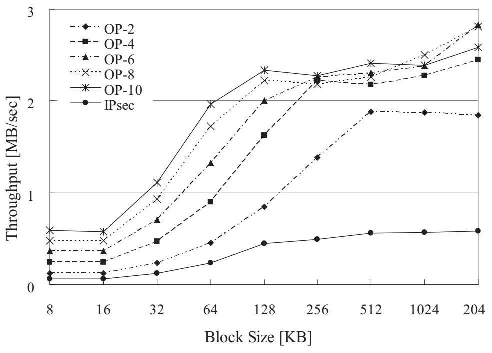

Fig. 10. Throughput on 64ms latency network

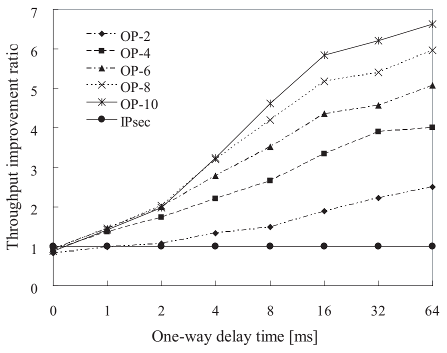

Fig. 11. Throughput improvement ratio

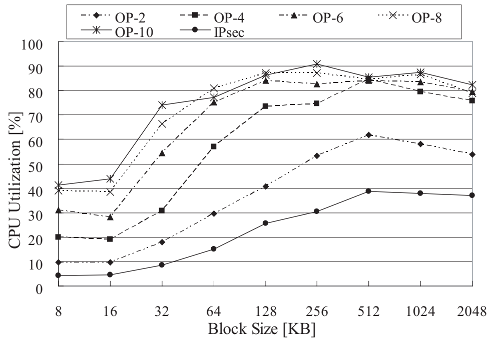

Fig. 12. CPU utilization on 8ms latency network

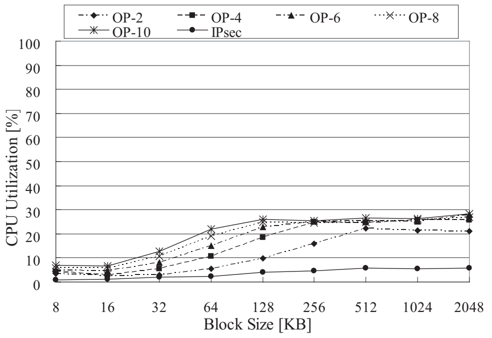

Fig. 13. CPU utilization on 64ms latency network

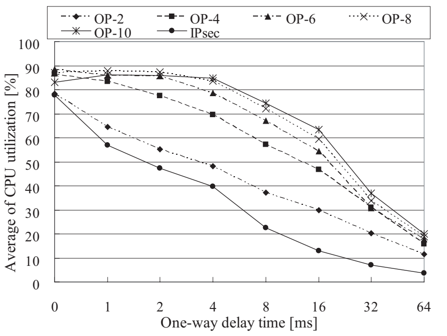

Fig. 14. Average CPU utilization

The experimental results of throughput with 8ms and 64ms delay time are shown in Figure 9 and 10. The IPsec throughput is saturated at the block size of more than about 512KB in all cases. On the contrary, throughput of our system (from OP-2 to OP-10) is much higher than that of IPsec in both block sizes. In addition, in the case of using relatively small block size, throughput is greatly increased with the number of processes for pre-processing in our middleware. In the 8ms latency environment, whereas IPsec throughput is 4.0MB/sec at most, our system achieves up to 8.8MB/sec. In the 64ms latency environment, although IPsec throughput is up to 0.6MB/sec at most, our system achieves up to 2.8MB/sec.

In the experiment, because both our system and IPsec execute encryption in a long-latency environment, the throughput is a low absolute value compared with no encryption case. However, since 10Gigabit Ethernet has already emerged at present, performance of network hardware is expected to improve in the future. In this case, though both the performance of our system and that of IPsec improve, we expect that our system outperforms IPsec as confirmed in the experiment.

Figure 11 shows an average throughput improvement ratio against IPsec in each one-way delay time. Throughput of our system is lower than that of IPsec slightly on the 0ms latency network. However, as one-way delay time becomes longer than 1ms, our system (from OP-4 to OP-10) outperforms IPsec. Our system's throughput (OP-10) on the 1ms latency network achieves about up to 1.5 times of IPsec. There is no significant difference between our system and IPsec in the short-latency network. On the contrary, as one-way delay time increases, the improvement ratio becomes higher. Our system's throughput (OP-10) achieves about up to 4.6 times of IPsec on the 8ms latency network. On the 64ms latency network, our system's throughput achieves about up to 6.6 times of IPsec. This is because the time of available CPU, i.e., waiting time for communication, becomes longer compared with encryption or decryption time as one-way delay time becomes longer. In fact, the advantage of optimized encryption processing increases on a long-latency network by encrypting next data segments consecutively before the previous sequential access cycle is finished. Consequently, our system's throughput (OP-10) achieves from 1.4 to 6.6 times of IPsec.

B. *Evaluation of CPU Utilization of Encrypted Write Access*

The CPU utilization during the experiment is shown in Figure 12 and 13. These figures show CPU utilization in the case of 8ms and 64ms latency network. Figure 14 shows an average CPU utilization in each one-way delay time. Our system's CPU load is higher than that of IPsec in all cases. However, as one-way delay time increases, the CPU utilization becomes smaller. On the 64ms latency network, the average CPU utilization of our system is up to 20% at most, as shown in Figure 14. Our system's CPU on a long-latency network can still afford to process the data concurrently, different from the case with a short-latency network. In the case of a long-latency network, communication time becomes longer compared with encryption or decryption time. Thus, our middleware is effective compared with IPsec on a long-latency network.

VI. Discussion on the Evaluation Experiment

A. Security Concerns

From a security standpoint, we discuss the comparison of our proposed mechanism with IPsec in this subsection.

The ESP protocol in IPsec ensures integrity of data by adding authentication data as an optional extra. Because we have implemented and evaluated our proposed method with particular emphasis on protecting data confidentiality, we do not evaluate it about integrity of data. However, it is easily possible to add a feature to our middleware that calculates message digest in the initiator and the target, and compares two values.

Additionally, if a tunnel mode that encrypts transferred data from gateway to gateway is used in IPsec, it encrypts not only TCP data but also IP header and TCP header in IP layer. In this experiment, because IPsec uses a transport mode, IP header is not encrypted in both our system and IPsec. At the same time, in our middleware, TCP header that includes a port number is not encrypted. However, if we add the authentication feature that prevents masquerading to our middleware, it is possible to avoid the issue. Actually, iSCSI has authentication schemes between the initiator and the target, and it can be decided which authentication schemes is used by negotiation. Examples of them include Kerberos V5, Simple Public-Key GSS-API Mechanism, Secure Remote Password and Challenge Handshake Authentication Protocol (CHAP). Therefore, even if TCP header is not encrypted in our system, it is not a signifficant security problem because the iSCSI authentication scheme can be used.

From an encryption algorithm standpoint, in the year 2000, the US National Institute of Standards and Technology (NIST) announced that Rijndael was selected as Advanced Encryption Standard (AES). While it is possible for various software and hardware to support AES encryption lately, 3DES encryption algorithm is still used in many devices by default at present. In our implementation, 3DES is employed because we consider it practicality in the current environment. However, since it is possible to apply AES to our middleware optimized for encryption processing, we assume that the performance of our system should also be improved in AES.

B. Theoretical Evaluation using Throughput Modeling

We have discussed throughput modeling derived from our proposed method with respect to sequential write access, and compared it with the experimental results of our implemented system. We have modeled the both cases of our middleware; without optimization and with optimization (the number of forked processes for pre-processing is two). First, we explain the throughput modeling in the case of encryption in the upper-layer without optimization. As shown in Figure 3, data is encrypted at the initiator side at first, and a Response Command is eventually returned to the initiator. We refer to this sequence as one cycle time in this subsection. The following Equation (1) is a relational expression of one cycle time (1CYCLE), Round Trip Time (RTT), data transfer time (TRANSFER), encryption time (ENC) and decryption time (DEC).

$$1CYCLE = 2*RTT + TRANSFER + ENC + DEC \quad (1)$$

The following Equation (2) is a relational expression of data transfer time (TRANSFER), data size (DATASIZE) and throughput in the lower layer (SOCKET). SOCKET is

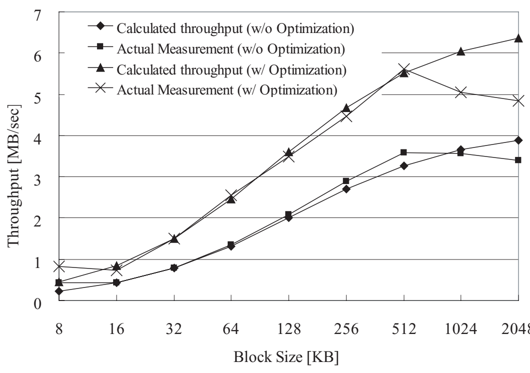

Fig. 15. Calculated throughput and Actual Measurement on 8ms latency network

throughput below iSCSI-layer. This is the value in a simple socket communication without iSCSI.

$$TRANSFER = \frac{DATASIZE}{SOCKET} \quad (2)$$

The 3DES algorithm does almost the same amount of calculation in both encryption and decryption. Therefore, we assume that the encryption time is the same as decryption time. In our experiments, the encryption throughput is 10.82MB/sec.

Thus, the projected throughput of sequential write access in our system without optimization is modeled as the following Equation (3) using block size (BLOCK), encryption throughput (ECN_TH) and decryption throughput (DEC_TH).

$$\text{THROUGHPUT} = \frac{BLOCK}{2*RTT + \frac{BLOCK}{SOCKET} + \frac{BLOCK}{ENC\_TH} + \frac{BLOCK}{DEC\_TH}} \quad (3)$$

On the other hand, based on Figure 4, we model the throughput in the case of encryption with optimization in the upper-layer. The relational expression of throughput in the optimization is modeled as the following Equation (4). Because the next data segment is encrypted during waiting time for communications, decryption time is hidden by encryption time.

$$THROUGHPUT = \frac{BLOCK}{RTT + \frac{BLOCK}{SOCKET} + \frac{BLOCK}{ENC\_TH}} \quad (4)$$

Figure 15 shows calculated values from the Equation (3) and (4), and actual measurements with and without the optimization on a 8ms latency network. From Figure 15, the values of the calculated throughput using the modeling are close enough to the actual measurements, except more than 512KB block size cases. In the case of large block sizes (1MB and 2MB), the actual measurements is saturated due to the limit of processing throughput of Dummynet. Thus, it is demonstrated that the throughput modeling by Equation (3) and (4) is almost correct.

C. Restrictive condition of the number of forked processes

In regard to our proposed method of optimizing for encryption processing, we have considered restrictive condition determined by the number of forked processes in parallel. As one of the ways to determine the most appropriate numbers of forked processes, we have taken into account how much data can be encrypted during a waiting time for communication. First, from Figure 4, the following Equation (5) can be

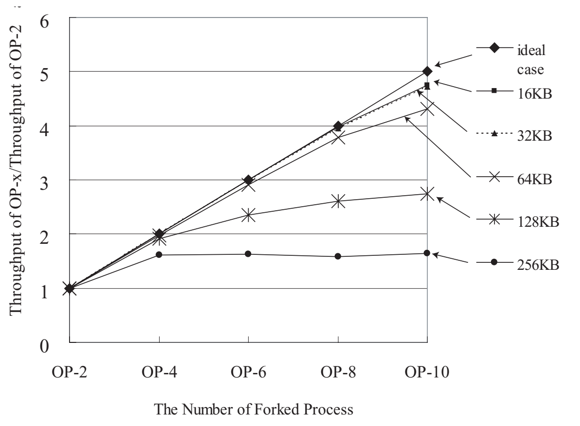

Fig. 16. Throughput ratio against the case of OP-2 on 64ms latency network

extended modeling of preceding subsection. This is a relational expression of Round Trip Time (RTT), data transfer time (TRANSFER), encryption time (ENC) and decryption time (DEC).

$$2 * RTT + DEC = n * (ENC + TRANSFER) \quad (5)$$

The left-hand side of Equation (5) stands for the waiting time for communications in one cycle time. In contrast, the right-hand side of Equation (5) stands for the execution time of encryption processing in parallel. The positive number n indicates the number of forked processes. Equation (6) is derived from Equation (5).

$$n = \frac{2 * RTT + DEC}{ENC + TRANSFER} \quad (6)$$

n in Equation (6) stands for the maximum number of forked processes with which ideal performance improvement can be achieved. In fact, under such a condition, if the number of forked processes is set to n, n times throughput improvement is expected.

Figure 16 shows throughput ratio based on the experimental results (Figure 10) against the case of OP-2 (the forked processes are two) on the 64ms latency network. In Figure 16, "ideal case" indicates ideal values; n times throughput improvement in our system when the number of forked processes is n.

On the 64ms latency network, n in Equation (6) is 15.7 in 32KB block size, 7.9 in 64KB block size and 4.1 in 128KB block size. They are limit values of the number of forked processes until which ideal throughput improvement is expected. In 16KB and 32KB block sizes, the performance improves linearly from OP-2 to OP-10. However, the performance does not improve linearly more than OP-8 (close to 7.9) in 64KB block size, and more than OP-4 (close to 4.1) in 128KB block size. The value of n derived from Equation (6) is almost equal to our experimental results (Figure 16). In this way, the maximum number of forked processes with which ideal performance improvement is represented by Equation (6).

D. Applicability to the realistic network

We have evaluated our system in a long-latency environment using the network delay emulator inserted between the initiator and the target. In this experiment, we have not considered impact of jitter or packet loss in actual wide area networks. However, if TCP retransmission occurs due to packet loss, IPsec that executes encryption in IP layer is required to repeat the encryption of the lost packets. In contrast, in the case of our proposed method, since data segments encrypted in our middleware perched on the TCP layer, the retransmission of the data stored in TCP buffer is performed. In such a case, lost packets are not required to be encrypted again in our system. Therefore, in actual traffic, the performance of our system receives less impact by packet loss than that of IPsec.

VII. Conclusion

In this paper, in order to prevent performance degradation that is caused by IPsec encryption processing, we have implemented middleware based on the proposed method optimized for encryption processing. In our proposed method, transferred data is encrypted in the upper-layer on top of the SCSI layer instead of using IPsec. Thus, the scheme for the performance improvement can be applied without modifying the implementation of IP layer, and efficient secure communications by pre-processing of encryption in the upper-layer are realized. In addition, we have measured the throughput and the CPU utilization of iSCSI sequential write access in a long-latency environment. We have also analyzed the experimental results in our system by modeling the throughput. Our system's throughput achieves 1.4-6.4 times of IPsec on from 1ms to 64ms latency networks. As a consequence, our middleware optimized for encryption processing has achieved better performance and is more efficient than using IPsec on a long-latency network.

In the future, we plan to evaluate our middleware in a more practical environment, such as multiple initiators and targets. We will also investigate the effect of the practical environment on our IP-SAN system.

References

[1] "Storage Networking Industry Association," http://www.snia.org/.
[2] S. Yamaguchi, M. Oguchi, and M. Kitsuregawa, "iSCSI Analysis System and Performance Improvement of Sequential Access in a Long-Latency Environment," in *IEICE Transaction on Information and Systems*, vol. J87-D-I.
[3] "iSCSI Draft," http://www.ietf.org/rfc/rfc3722.txt.
[4] W. T. Ng, B. Hillyer, E. Shriver, E. Gabber, and B. Ozden, "Obtaining High Performance for Storage Outsourcing," in *Proc. FAST 2002, USENIX Conference on File and Storage Technologies*.
[5] P. Sarkar, S. Uttamchandani, and K. Voruganti, "Storage over IP: When Does Hardware Support help?" in *Proc. FAST 2003, USENIX Conference on File and Storage Technologies*.
[6] S. Aiken, D. Grunwald, A. Pleszkun, and J. Willeke, "A Performance Analysis of the iSCSI Protocol," in *Proc. 20th IEEE Symposium on Mass Storage Systems and Technologies (MSS '03)*.
[7] C. Gauger, M. Koehn, S. Gunreben, D. Sass, and S. G. Perez, "Modeling and Performance Evaluation of iSCSI Storage Area Networks over TCP/IP-based MAN and WAN networks," in *Proc. The Second International Conference on Broadband Networks*, vol. 2.
[8] G. A. Gibson, D. F. Nagle, W. C. II, N. Lanza, P. Mazaitis, M. Unangst, and J. Zelenka, "NASD Scalable Storage Systems," in *Proc. Extreme Linux Workshop in the 1999 USENIX*.
[9] S.-Y. Tang, Y.-P. Lu, and D. H. C. Du, "Performance Study of Software-Based iSCSI Security," in *Proc. First International IEEE Security in Storage Workshop*.
[10] A. Joglekar, M. E. Kounavis, and F. L. Berry, "A Scalable and High Performance Software iSCSI Implementation," in *Proc. FAST 2005, USENIX Conference on File and Storage Technologies*.
[11] "InterOperability Lab in the University of New Hampshire," http://www.iol.unh.edu/.
[12] "dummynet," http://info.iet.unipi.it/~luigi/ip_dummynet/.
[13] "FreeS/WAN Project," http://www.freeswan.org/.