ISSN 2277-3061

Estimation of Performance Improvement Derived fromTCP/IP Offload Engine with Software Emulation

Takamichi Nishijima, Nobuhiro Yokoi, Yoichi Nakamoto, Hiroyuki Ohsaki

Graduate School of Information Science and Technology,
Osaka University, Suita, Osaka 565-0871, Japan
t-nisijm@ist.osaka-u.ac.jp
Hitachi, Ltd., Yokohama Research Laboratory, 292 Yoshida-cho,
Totsuka-ku, Yokohama, Kanagawa 244-0817, Japan
nobuhiro.yokoi.eh@hitachi.com
Hitachi, Ltd., Yokohama Research Laboratory, 292 Yoshida-cho,
Totsuka-ku, Yokohama, Kanagawa 244-0817, Japan
yoichi.nakamoto.ef@hitachi.com
Department of Informatics School of Science and Technology,
Kwansei Gakuin University 2-1 Gakuen,
Sanda, Hyogo 669-1337, Japan
ohsaki@kwansei.ac.jp

ABSTRACT

Many TOE (TCP/IP Offload Engine) devices have been developed and installed in high-end systems. Although there have been many analysis of the effectiveness of TOE devices, there remain open issues related to them. For example, it has not been clarified which part of TCP/IP processing should be performed with hardware and which with software, or how end-to-end TCP/IP performance is affected by the introduction of a TOE device. In this paper, we propose ***VOSE (Virtual Offloading with Software Emulation)***, which is a technique for measuring TCP/IP performance improvements derived from different type of TOE devices without implementing TOE prototypes really. VOSE enables ***virtual offloading*** without requiring a hardware TOE device by virtually emulating TOE processing on both source and destination end hosts. For demonstrating the effectiveness of VOSE, we apply VOSE to the TCP checksum and IPsec protocol. We extensively examine the accuracy of virtual offloading with VOSE, by comparing performance (i.e., end-to-end performance and CPU processing overhead) between VOSE and a dedicated TOE device. Moreover, we estimate performance improvement that are derived from several TOE devices of IPsec and combinations of those devices, by applying VOSE to header authenticating and payload encryption in IPsec protocol. Consequently, we show that performance improvements which are derived from TOE devices can be estimated correctly.

Indexing terms/Keywords

Hardware Offloading; TOE (TCP/IP Offload Engine); Network Performance; Virtual Software Emulation; Virtual Offloading.

Academic Discipline And Sub-Disciplines

Network Performance; Protocols; Sensors; Networking theory and technologies

SUBJECT CLASSIFICATION

Performance Estimation; Software Emulation; Hardware Offloading

TYPE (METHOD/APPROACH)

Approach

Council for Innovative Research

Peer Review Research Publishing System

Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY

Vol 12, No.1

editor@cirworld.com

www.cirworld.com, member.cirworld.com

1. Introduction

End system protocol processing is becoming a bottleneck in end-to-end communication because of rapid increases in the speed and bandwidth of networks such as 10 Gigabit Ethernet® [1]. Since the performance of end-to-end communication is limited by the processing speed of the bottleneck, such bottlenecks must be remedied to increase the speed of communication networks. Most internet traffic is transferred by TCP (Transmission Control Protocol) [2] and its derivatives[3,4]. Hence, improved TCP protocol processing would result in a significant network performance gain [5,6]. According to past experimental verifications, with the exception of very high-speed networks, 1 bit/s protocol processing generally requires about a 1 Hz processor [7]. This generalization implies that sufficient protocol processing to fully utilize the capacity of 10 Gigabit Ethernet with a processor that operates at several gigahertz would be challenging.

TOE (TCP/IP Offload Engine) has been studied by many researchers as a means of accelerating protocol processing. TOE is a technology that uses a hardware accelerator to perform heavy TCP/IP processing in end systems. There have been many performance studies of TOE devices [8-17], and many TOE devices have been developed and installed, particularly in high-end systems [18,19]. There remain, however, open issues related to TOE devices. For example, it has not been clarified which part of TCP/IP processing should be performed with hardware and which with software, or how end-to-end TCP/IP performance is affected by the introduction of a TOE device.

In TOE design, the amount of CPU processing load reduction alone is not a good measure. TOE design must also take into account improvements in end-to-end network performance (e.g., throughput and latency) resulting from the introduction of TOE [20,21]. For instance, it is known that memory copies between the user and kernel spaces and packet checksum calculations are bottlenecks in TCP processing [5,22]. CPU processing load reduction resulting from the introduction of a TOE device can be easily measured using modern profilers. However, measurement of end-to-end network performance improvements generally requires actual experiments with a TOE implementation. Namely, in TOE design, to measure performance improvements derived from introducing a TOE device, it is necessary to implement a TOE prototype really. A method for measuring the effectiveness of protocol offloading with a specific TOE device that does not require installation of the device itself would significantly reduce the cost of TOE design and development.

In this paper, we therefore propose VOSE (Virtual Offloading with Software Emulation), which is a technique for measuring TCP/IP performance improvements derived from different type of TOE devices without implementing TOE devices really. VOSE enables offloading without hardware by virtually emulating TOE processing (e.g., bypassing software protocol processing without violating protocol consistency) on both the source and destination end hosts. VOSE realizes protocol processing offloading by utilizing symmetry of protocol processing between the sender and the receiver while preserving its integrity.

In addition, for demonstrating the effectiveness of VOSE, we apply VOSE to the TCP checksum and IPsec protocol. First, we apply VOSE to the TCP checksum in a Linux® kernel. We extensively examine the accuracy of virtual offloading with VOSE by comparing performance (i.e., end-to-end performance and CPU processing overhead) between virtual offloading with VOSE and hardware offloading with a dedicated TOE device. Second, we apply VOSE to header authenticating and payload encryption in IPsec protocol. We estimate the performance improvement that is derived from several TOE devices of IPsec and combinations of the TOE devices because TOE devices of IPsec have not been evaluated sufficiently yet.

One of the important contributions of this work is as follows; VOSE enables to measure performance improvements derived from introducing a TOE device, without implementing TOE prototypes really. More specifically, unlike simulators, the improvements in real environments (i.e., operating environments) can be measured by using VOSE because VOSE enables measurements using real computers, networks, applications and protocols. This leads to 1) minimizing costs of design and development, 2) minimizing development periods and 3) optimizing TOE performance. 1) VOSE enables that TOE designers elucidate bottlenecks of various TCP/IP processings by trial and error and implement a TOE device that offloads the bottlenecks only. Therefore, VOSE realizes the maximum effect at the minimum cost of design and development. 2) VOSE eliminates the need for implementing many TOE prototypes to satisfy performance requirements and enables testing various patterns of TCP/IP offloads in a short period. 3) VOSE enables optimizing the balance between software and hardware processings because TOE designers measure performance of the various patterns of offloadings and clarify processings that should be performed with software and the processings that should be performed with hardware.

The construction of this paper is as follows: Section 2 summarizes previous work related to TCP/IP processing loads and TOE. Section 3 gives an overview of TOE, and Section 4 describes VOSE. Section 5 describes the TCP checksum calculations experiments: end-to-end performance measurements of virtual offloading with VOSE versus hardware offloading with TOE, and system loads. Section 6 describes the performance improvement caused by introduction of several types of TOE devices for header authenticating and payload encryption in IPsec protocol. Finally, Section 7 summarizes this paper and discusses directions for future study.

2. Related Work

Many studies have measured TCP processing loads, and in particular there are several studies that measure processing loads in relatively high-speed networks exceeding 1 Gbit/s [6,7,23].

Chase *et al.* surveyed high-speed methods related to the TCP/IP stack and experimentally evaluated their effectiveness[23]. They targeted overhead mitigation at the packet level (using large frames, mitigating the number of

device interruptions) and the byte level (integration of copy avoidance, checksum offloading, copy processing, and checksum calculations). Chase *et al.* show that integrating copy processing and checksum calculations was ineffective, and that copy avoidance was effective (approximately a 50-100% improvement in throughput). They also found that performing checksum calculations with hardware was effective, giving an approximately 30% improvement in throughput.

Foong *et al.* measured the processing load of the TCP/IP stack under Linux kernel 2.4 [7]. In particular, they investigated the scalability of the TCP/IP stack with CPU clock speed, and found that the convention that 1 bit/s protocol processing requires 1 Hz of processing power fails when the CPU clock exceeds 1 GHz. Foong *et al.* showed that the balance of sender and receiver loads changes with data size, and that checksum offloading was effective only for large data sizes, thus making increasing TCP/IP stack speeds via this method a complex task.

Although many other experimental evaluations of the effectiveness of TOE exist [8-16], most measure performance under a specific environment. For instance, Fukuju and Ishihara used a Toshiba TOE and measured the throughput when offloading IPsec processing to the TOE [8]. Ghadia evaluated the throughput and delay when processing all aspects of TCP/IP by hardware, and showed that when doing so throughput more than doubles, while delay is reduced approximately 20% [16]. Jang *et al.* evaluated performance when establishing connections and controlling flow by software, and processing packet buffers management, TCP headers, and ACK packets by hardware, and found that doing so reduces CPU load to 1/5 or less [13]. Benjamin and Patrick compared throughput when performing TCP checksum calculations on a TOE versus processing solely by software, and found that the TOE approximately doubled throughput [11].

Thus, many studies have demonstrated the effectiveness of TOE. Each of these studies, however, measures performance when offloading TCP/IP processing under specific experimental environments. Moreover, there has been insufficient examination of how TCP/IP protocol processing should be divided between hardware and software to maximize performance gains.

Westrelin *et al.* measured the TOE offloading effect using software emulation under symmetrical type multiprocessing, a computer design by which multiple processors can share and manage physical memory [17]. Doing so, they realized processing equivalent to hardware by performing part of TCP/IP handling on processors other than the main processors. However, because this method uses code particular to Solaris® to perform processing with the regular core, this method cannot be used with other operating systems. Moreover, because this method connects processor boards over the PCI® bus, there is overhead associated with data transport between memories. Because there are limits to the number of cores and amount of physical memory that symmetrical type multiprocessing can use, TOE performance can be mimicked only to a certain degree. Therefore, there remains a need for methods of offloading effect measuring performance improvement with the introduction of a TOE device, regardless of the type of computer used or experimental environment.

3. TOE (TCP/IP Offload Engine)

TOE is a technology that uses a hardware accelerator to perform heavy TCP/IP processing in end systems [12-16]. In conventional systems, TCP/IP protocol processing is performed by the host computer's CPU. Under TOE, however, a part of TCP/IP processing is directly performed by hardware without CPU intervention.

Below, as an example, we explain TOE operation when offloading TCP checksum calculations. Figure 1 shows how introducing TOE changes the TCP/IP processing flow. Figure 1 (a) is the processing flow when performing all TCP/IP processing by software. In the transport and network layers, several TCP/IP activities such as flow control and checksum addition are performed on data received from the application layer. Generally, all of these processes are performed by the CPU. As the network transmission speed increases, the number of CPU memory accesses will also increase, in turn increasing CPU load.

In contrast, Figure 1 (b) shows the processing flow when offloading checksum calculations to TOE. In this case, dedicated hardware performs checksum calculations in place of the CPU, alleviating the need for memory access and mitigating increased CPU load.

TCP/IP processing tasks that are generally suited to hardware offloading have the following characteristics:

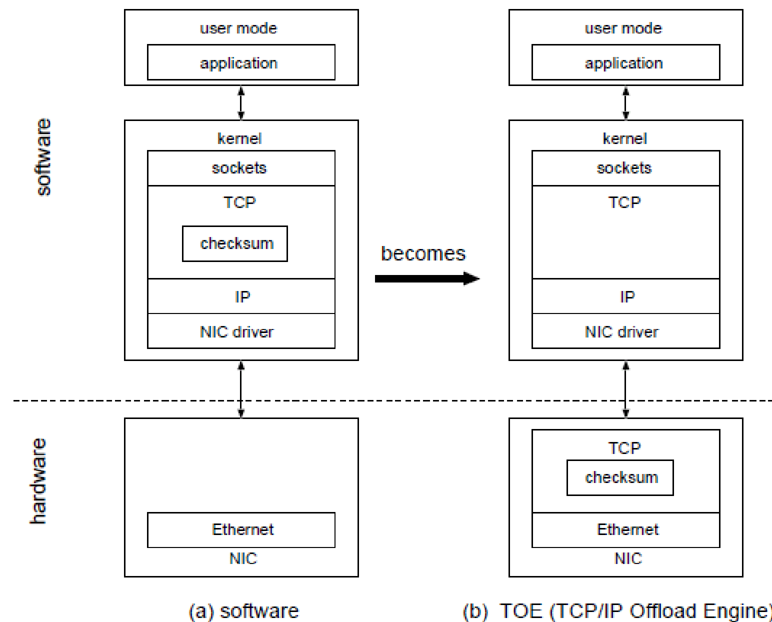


Fig1. Changes in TCP/IP processing under TOE: (a) the flow when performing all TCP processing in software, and (b) the processing flow when offloading checksum calculations to TOE

(1) Processing suited to hardware

Some data processing is better performed by dedicated hardware than by a general purpose CPU. CPUs perform many operations in succession at high speeds. Dedicated hardware, on the other hand, can be advantageous when processing can be performed in parallel. Examples of such processing includes checksum calculations and handling of fixed length data, such as encryption and decryption. In the case of TCP/IP processing, therefore, tasks such as TCP checksum calculations, IPsec encryption and decryption, and memory copies are considered to be suited to hardware processing.

(2) Processing that increases with increased transmission speed

Some processing that is not problematic in a low-speed network can become an issue as network speed increases. For example, the burden of payload processing increases with longer packet length, faster transmission speed, and larger frame size. Specific examples from TCP/IP processing include TCP payload checksum calculations and IPsec encryption and decryption.

(3) Processing that requires real-time execution

This characteristic refers to processing that must complete within a guaranteed time. When dedicated hardware is used, hardware processing is not affected by other processing, allowing completion within a constant time. In contrast, because processing time can vary with CPU status or memory conditions, software processing cannot guarantee completion times. Video and audio streaming protocols are examples of processing that requires real-time execution. However, TCP/IP networks are best-effort networks where packet delivery is not guaranteed, and end-to-end delays may occur. Hence, processing that requires real-time execution under TCP/IP are limited to, for example, retransmission timers.

Processing with the above characteristics is the main target for TOE offloading. In the following section, we propose and evaluate a method of measuring end-to-end performance gains when such processes are performed on hardware.

4. Proposal of VOSE (Virtual Offloading with Software Emulation) for Estimating Performance Improvement with TCP/IP Protocol Offloading

The fundamental idea of VOSE is partial TCP/IP processing of sender or receiver data by virtualized hardware under software emulation. Since TCP/IP is an end-to-end communication protocol, the bulk of TCP/IP processing is symmetrical on both sender and receiver sides.

VOSE emulates offloading of protocol processing by utilizing this symmetry. For instance, offloading is replaced by short-time sleep in the transport layer, in a form where the integrity of the processing is maintained between sender and receiver (Figure 2).

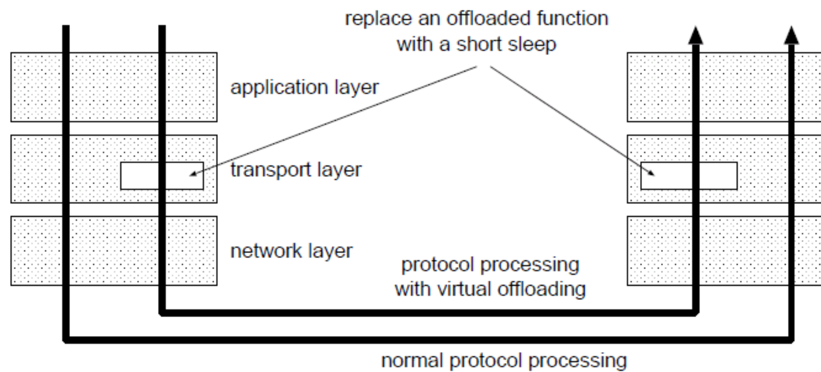


Figure 3. VOSE realizes virtual offloading of protocol processing by utilizing the symmetry of protocol processing between the sender and the receiver while preserving its integrity.

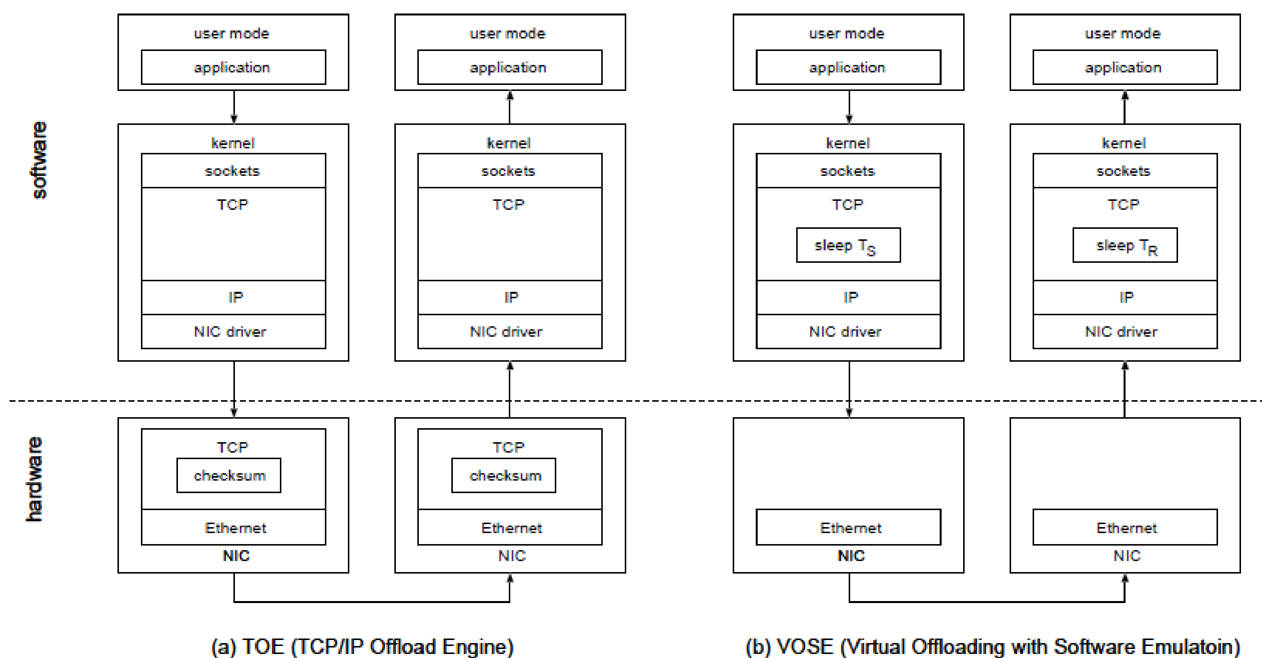


Figure 3. Changes in TCP/IP processing by VOSE: (a) the processing flow in the case of offloading checksum calculations to TOE, and (b) the processing flow in the case of virtually offloading checksum calculations.

VOSE can emulate arbitrary TOE devices with different processing speeds by changing the duration of a short sleep. Thus, with VOSE, a TOE designer can measure the overall performance with different types of TOE devices without developing and implementing the TOE hardware.

However, because TCP/IP requires processing of multiple transport factors, it is necessary to consider which processes should be offloaded. We classify protocol processing according to the following three types:

(4) Symmetrical

Symmetrical processing involves processing between sender and receiver. Examples include TCP checksum addition and verification by the sender and the receiver, packet order control, and management of multiple connections.

(5) Independent

Independent processing is performed by either sender or receiver. An example is memory data copies from user space to the kernel space by the sender, or data copies from the kernel space to the user space by the receiver.

(6) Cooperative

Cooperative processing is performed by sender and receiver together. Examples include congestion control such as window control based on ACK packets from the receiver, and flow control to the network based on the advertising window from the receiver.

Of these, virtual offloading is possible for symmetrical protocol processing by bypassing symmetrical sender and receiver protocol processing simultaneously.

ISSN 2277-3061

Independent protocol processing is completed by sender or receiver, so virtual offloading is possible if integrity in protocol processing are maintained.

Cooperative protocol processing, however, may affect other processing, so the potential for virtual offloading and how it should be performed depends on the particular processing performed.

It should be noted that most *heavy* protocol processing are classified as either symmetrical or independent, which can be virtually offloaded with VOSE. Cooperative protocol processing needs *cooperation* between the sender and the receiver, which results in slow processing. So, TOE devices for cooperative processing are rarely required.

In what follows, we use TCP checksum calculations (an example of symmetrical processing) as an example of virtual offloading. In the transport layer, TCP checksums are used to detect bit errors in the TCP packet headers and payloads. When sender sends a packet, checksums are calculated for the pseudo TCP header, the TCP header, and the TCP payload, and the value is stored in the TCP header [2]. When receiver receives the packet, the checksums are calculated again and compared with the value in the header to ensure that the packet is not corrupt.

TCP checksum calculations are performed strictly in the transport layer. Hence, if, as shown in Figure 3 (b), sender and receiver checksum calculations are replaced by sleep operations of T_S [s] and T_R [s], respectively, those calculations can be considered to have been virtually offloaded (Figure 3). How T_S and T_R should be configured depends on the TOE architecture, the access speed of memory and the bus, and interrupt processing of the operating system. Nonetheless, the theoretical maximum performance of TOE can be investigated by setting T_S and T_R to 0.

Recall that VOSE is not specific to virtual offloading of TCP checksum. Instead, as we have explained above, it can be used different types of protocol processing. For instance, application of VOSE to other protocol processing than TCP checksum will be discussed in Section 6.

Thereby, we can perform communication normally between hosts in which virtual offloading is carried out by VOSE, allowing us to then measure the effects of TCP/IP offloading.

5. Evaluation of VOSE

In this section, we evaluate the accuracy of virtual offloading with VOSE. We apply VOSE to the TCP checksum calculations in a Linux kernel. We compare end-to-end performance and system loads between virtual and hardware offloading.

5.1 Experimental environments

To evaluate the accuracy of virtual offloading with VOSE, we perform experiments by carrying out virtual offloading of TCP checksum calculations. Hardware offloading of checksum calculations has already been carried out by NIC (Network Interface Card). We can therefore evaluate the accuracy of virtual offloading with VOSE by comparing experimental results with NIC hardware offloading.

In the experiments, we use two experimental environments as follows.

- Experimental environment A

  Two computers are connected by Gigabit Ethernet. The computers with a Intel® Pentium® 4 3.03 GHz CPU and 2 Gbyte RAM are used. A NIC installed to the computers is an Intel PRO/1000® Network Driver.

- Experimental environment B

  Two computers are connected by 10 Gigabit Ethernet. The computers with an Intel Core i7® 3.2 GHz CPU and 3 Gbyte RAM are used. The processor is operated with only a single core active. A NIC installed to the computers is an Intel PRO/10G® Network Driver.

```
        rdtsc
        addl $clocks_to_sleep, eax
        adcl $0, edx
        movl eax, ecx
        movl edx, ebx
loop:
        subl ecx, eax
        sbbl ebx, edx
        js   loop
```

Fig 4. Realization of the busy loop using the RDTSC command: A loop is carried out for the specified number of cycles using the IA32/IA64 processor RDTSC instruction, which waits for $clocks_to_sleep clocks.

The operating system in both environments is Debian® GNU/Linux 5.0.2 (Linux kernel 2.6.30). Each experiment is conducted with hardware offloading of TCP checksum calculations was carried out by TOE on the NIC, and with virtual offloading carried out by VOSE. To eliminate the effects of offloading processing other than TCP checksum calculations, we turn off with TSO (TCP Segmentation Offload), GSO (Generic Segmentation Offload), and LRO (Large Receive Offload).

In the experiments, we use iperf as benchmarking software for continuously transferring data for 60 [s]. We measure CPU utilization for 60 [s] using the information in **/proc/stat**. We repeat the experiments 10 times, and then measure average values of throughput and CPU utilization and calculate their 95% confidence intervals.

5.2 Virtual offloading of TCP checksum calculations

We explain how TCP checksum calculations in the Linux kernel version 2.6.30 can be virtually offloaded with VOSE.

Virtual offloading of TCP checksum calculations can be realized by replacing **tcp_v4_send_check()** of the sender and **tcp_v4_checksum_init()** of the receiver functions with a short sleep of T_S and T_R in both the sender and the receiver, respectively. To bypass TCP checksum calculations in **csum_and_copy_from_user()** of the sender, we replace that function call with function call of **copy_from_user()** in the sender. To bypass TCP checksum calculations in **tcp_v4_checksum_init()** of the receiver, **skb->ip_summed** is set to **CHECKSUM_UNNEC-ESSARY**, and **__tcp_checksum_complete()** is not performed.

We realize a sleep processing using a busy loop of CPU. To realize the busy loop, the loop is carried out for the specified number of cycles using the IA32® and IA64® Intel processor RDTSC instruction (Fig. 4).

We program a busy loop by using an inline assembler to realize the sleep processing of T_S and T_R. Note that each loop requires approximately 24 clock cycles to process and that the granularity of sleep times is specified during the busy loop.

5.3 Results

First, Figs. 5 and 6 show the end-to-end performance (effective throughput) with virtual offloading by VOSE (labeled as 'VOSE') in the Gigabit Ethernet (experimental environment A) and the 10 Gigabit Ethernet (experimental environment B) when the CPU operating frequency of the sender and receiver is varied. For comparison purposes, the end-to-end performance with TOE (labeled as 'Hardware') without TOE (labeled as 'Software') are also shown in the figure. In these experiments, we set T_S and T_R to 0.

Figures 5 and 6 show that throughput mostly coincides in both the case where checksum calculations are offloaded by VOSE and the case where calculations are offloaded by hardware. The average error was below 1% over Gigabit Ethernet, and below 3% over 10 Gigabit Ethernet. These observations indicate that VOSE correctly emulates the offloading effect of TCP checksum calculations.

Next, we focus on CPU utilization of the sender (Figs. 7 and 8). Figure 7 shows that CPU utilization of the sender over Gigabit Ethernet mostly coincides both when checksum calculations are offloaded by VOSE and when that are performed by hardware. In both cases the average error is 3% or less. Thus we find that emulation of the offloading effect is correctly carried out by VOSE. Over 10 Gigabit Ethernet (Fig. 8), when virtually offloading with VOSE, the sender CPU utilization is 19 % less on average than the hardware offloading.

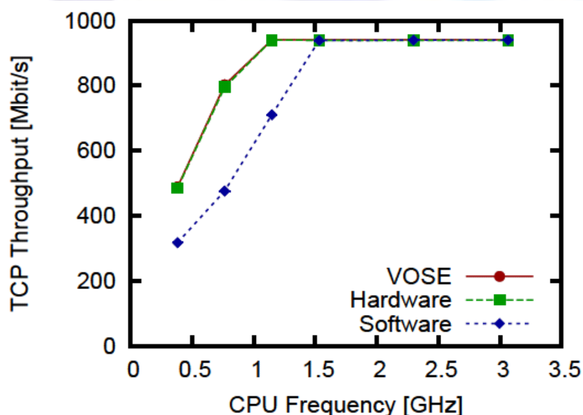


Fig 5. End-to-end performance (effective throughput) when changing the sender and receiver CPU operating frequency over Gigabit Ethernet (experimental environment A)

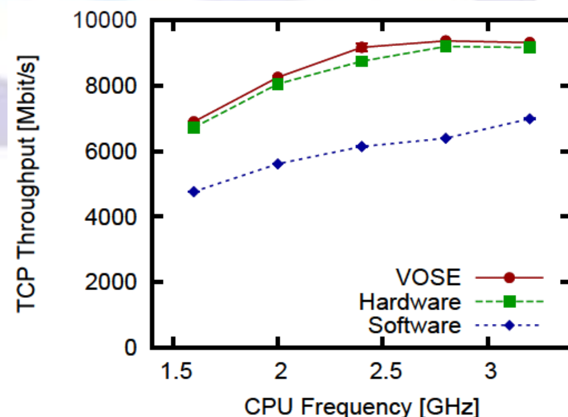


Fig 6. End-to-end performance (effective throughput) when changing the sender and receiver CPU operating frequency over 10 Gigabit Ethernet (experimental environment B)

ISSN 2277-3061

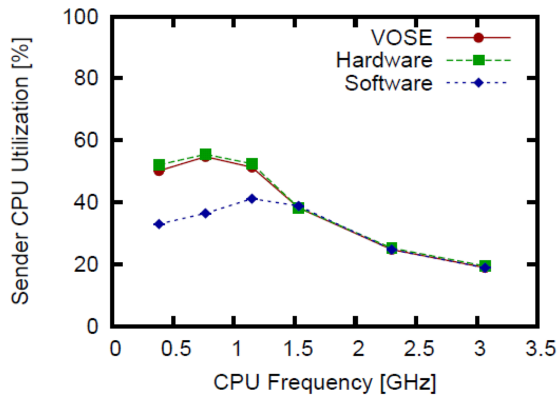


Fig 7. CPU utilization of the sender when changing the sender and receiver CPU operating frequency over Gigabit Ethernet (experimental environment A)

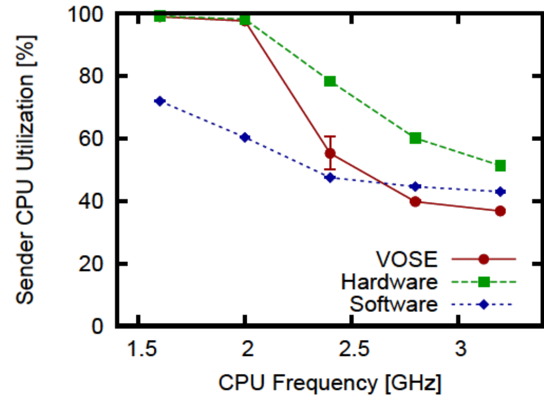


Fig 8. CPU utilization of the sender when changing the sender and receiver CPU operating frequency over 10 Gigabit Ethernet (experimental environment B)

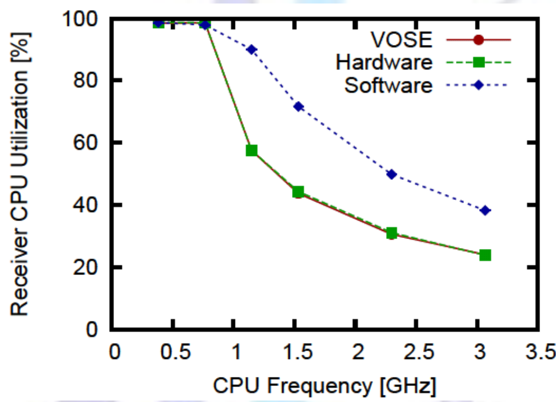


Fig 9. CPU utilization of the receiver when changing the sender and receiver CPU operating frequency over Gigabit Ethernet (experimental environment A)

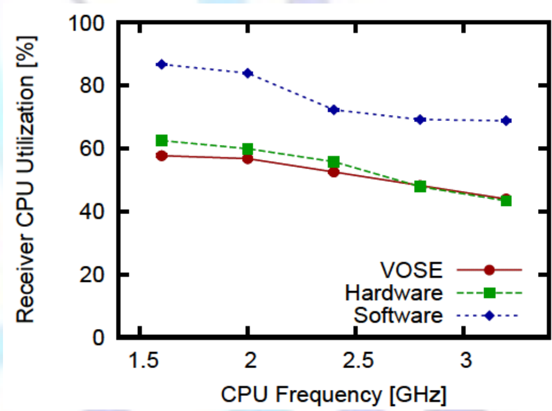


Fig 10. CPU utilization of the receiver when changing the sender and receiver CPU operating frequency over Gigabit Ethernet (experimental environment A)

Tab 1. The profiling result of the sender when carrying out a hardware offloading with TOE in 3.2 GHz

samples	%	symbol name
171700	26.3	copy_from_user()
90286	13.8	ixgbe_clean_tx_irq()
70741	10.9	ixgbe_xmit_frame()
32196	4.9	kmem_cache_alloc()
31117	4.8	__kmalloc()
30769	4.7	kfree()
21116	3.2	kmem_cache_free()
19136	2.9	ixgbe_intr()
14830	2.3	cache_alloc_refill()

ISSN 2277-3061

Tab2. The profiling result of the sender when carrying out a virtual offloading with VOSE in 3.2 GHz

samples	%	symbol name
169239	27.9	copy_from_user()
57922	9.5	ixgbe_clean_tx_irq()
56400	9.3	ixgbe_xmit_frame()
32694	5.4	__kmalloc()
31417	5.2	kmem_cache_alloc()
30547	5.0	kfree()
21537	3.5	kmem_cache_free()
18950	3.1	ixgbe_intr()
18055	3.0	cache_alloc_refill()

Figures 9 and 10 show that CPU utilization of the receiver mostly coincides, both when checksum calculations are offloaded by VOSE and when that are performed by hardware. When operating over Gigabit Ethernet, both error rates averaged below 1%. Over 10 Gigabit Ethernet, both error rates averaged below 5%. These observations indicate that VOSE correctly emulates the offloading effect.

Over 10 Gigabit Ethernet, the sender CPU utilization when virtually offloading the TCP checksum calculations by VOSE is less than the case where hardware offloading is performed. In these experiments, TCP checksum calculations in the sender and receiver is simply bypassed in the virtual offloading in VOSE ($T_S = T_R = 0$). Hence, in the virtual offloading by VOSE, the call overhead of TOE produced in a hardware offloading is not contained. Hence, the sender CPU utilization in the case of the virtual offloading by VOSE serves as a small value from the value in the case of a hardware offloading.

To confirm the call overhead of TOE of the sender, we compared the profiling result of the sender when carrying out a hardware offloading with TOE in 3.2 GHz (Tab. 1) with the result when carrying out a virtual offloading (Tab. 2) using the profiler [24]. We describe only the processings whose rate in the whole exceeds 1 % among results. Results show that the total amount of processings (i.e., samples in Tabs. 1 and 2) of **ixgbe_clean_tx_irq()** and **ixgbe_xmit_frame()** decreases when a virtual offloading is carried out using VOSE. These functions are contained in an Intel 10GbE driver and are the transportation processing to hardware. From these observations, when carrying out a virtual offloading, we find that the load of these transportation processing to hardware is not contained in CPU utilization of the sender (VOSE in Fig. 8).

If required, we guess that a more exact emulation becomes possible by configuring the value of T_S and T_R appropriately according to the call overhead of TOE produced in a hardware offloading. In other words, we guess that a more exact emulation becomes possible by configuring T_S and T_R so that the amount of processings between VOSE and hardware may coincide.

On the contrary, if the processing time of TOE hardware is converged to 0, we expect that the performance with a TOE hardware offloading approaches the performance with a virtual offloading by VOSE. In other words, supposing a TOE designer creates TOE hardware with the processing speed near 0, we expect that throughput improves to the value of the throughput of VOSE in Fig. 6. Similarly, we expect that CPU utilizations decrease to the value of the CPU utilizations of VOSE in Figs. 8 and 10.

5.4 Observation

We next describe the offloading effect of performing TCP checksum calculations over Gigabit Ethernet (experimental environment A). First, we explain the offloading effects of TOE from the results of Sec. 5.3. Next, we apply VOSE to TCP checksum calculations, and estimate the performance improvement caused by introduction of TOE device.

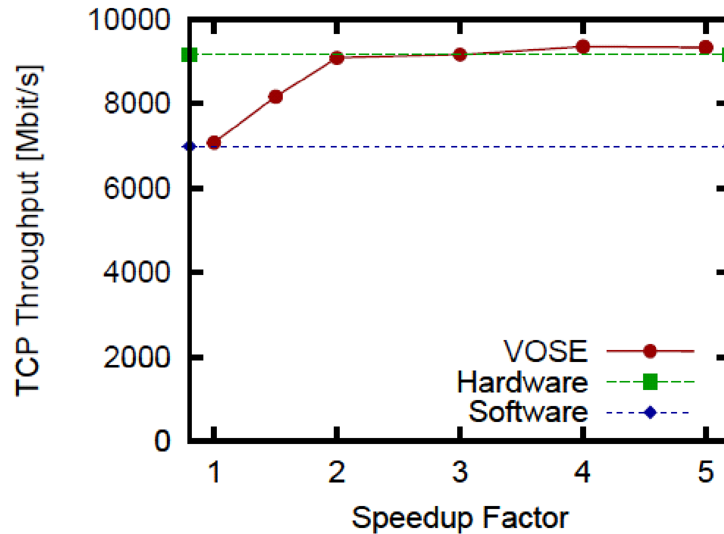


Figure 11. The end-to-end performance (effective throughput) when changing the time T_S and T_R of sleep processing

First, from the results of Sec. 5.3, we compare the case where offloading TCP checksum calculations is carried out by hardware, the case where simulated offloading is carried out by VOSE, or the case where the processing is performed entirely by software.

Figure 5 shows that offloading the TCP checksum calculations is effective when the CPU frequency is low. Specifically, when the CPU frequency is under 1.5 GHz, throughput improves significantly (approximately 1.3 to 1.7 times) in Gigabit Ethernet by offloading the checksum calculations. When CPU frequencies exceed 1.5 GHz, however, throughput does not improve. These results indicate that offloading TCP checksum calculations is effective in Gigabit Ethernet when CPU frequency is less than 1.5 GHz.

Figure 9 shows that offloading TCP checksum calculations significantly reduces CPU loads at the receiver. For instance, when compared to the case where the CPU frequency is above 1 GHz and offloading is not performed, hardware offloading or virtual offloading reduces CPU utilization by approximately 35 to 45%. When the CPU frequency is 1.5 GHz or less, however, hardware offloading or virtual offloading increases CPU loads at the sender shown in Fig. 7. The increase in such CPU loads originates in throughput having improved (Fig.5). These results show that the receiver of the offloading of TCP checksum calculations is more effective than the sender.

Moreover, as mentioned above, if CPU frequencies exceed 1.5 GHz, it is possible to use up the bandwidth even if offloading the TCP checksum calculations is not performed. However, the point of introducing TOE is not only increasing throughput, but also decreasing CPU load. Hence, even if CPU frequency is 1.5 GHz or more, hardware offloading of TCP checksum calculations is effective.

Next, with VOSE-enabled Linux kernel, we experimentally estimate the performance improvement caused by introduction of TOE device for TCP checksum calculations. Figure 11 shows the end-to-end performance (effective throughput) with TOE device in the experimental environment B (i.e., 10 Gigabit Ethernet, Core i7 3.2 [GHz] CPU) when the speedup factor of TOE devices is varied. The speedup factor is the ratio of the TOE processing speed to that of the software processing speed.

We calclates T_S and T_R as follows. In what follows, $T_{S\ software}$ is the processing time of software in the sender, and $T_{R\ software}$ is the processing time of software in the receiver.

$$T_S = \frac{T_{S_{software}}}{\text{speedup factor}}$$

$$T_R = \frac{T_{R_{software}}}{\text{speedup factor}}$$

For instance, when a speedup factor was 2, we set T_S and T_R to the half of the processing time of the software in the sender and receiver, respectively. The processing time of the software was measured by counting the average number of CPU clocks spent in **tcp_v4_send_check()** and **tcp_v4_checksum_init()**. Specifically, RDTSC operations were embedded at the start and the end of **tcp_v4_send_check()** and **tcp_v4_checksum_init()**, and the difference in the number of CPU clocks were measured. For comparison purposes, the end-to-end performance with TOE (labeled as `Hardware') and without TOE (labeled as `Software') are also shown in the figure.

Figure 11 shows that throughput increases large when a speedup factor is set to 1.5 - 2. However, throughput hardly increases when the speedup factor is 3 or more. From these observations, we find that we should use TOE that finishes processing of TCP checksum calculations within the half of the processing time of the software to increase throughput sufficiently.

Consequently, VOSE enables TOE designers to know the overall performance improvements which are derived from introducing a TOE device in advance. Moreover, VOSE enables TOE designers to know in advance how much processing speed of TOE hardware is required for obtaining overall required performance. This leads to minimizing costs of design and minimizing development periods. For instance, TOE designers can elucidate bottlenecks of various TCP/IP processings by trial and error and implement a TOE device that offloads the bottlenecks only. Moreover, TOE designers can test various patterns of TCP/IP offloads in a short period. Because the need for really implementing many TOE prototypes to satisfy performance requirements is eliminated, VOSE reduces the cost and periods in designing TOE devices.

6. Application of VOSE to IPsec Protocol

In this section, to demonstrate the effectiveness of VOSE, we apply VOSE to header authenticating and payload encryption in IPsec protocol, and estimate the performance improvement caused by introduction of several types of TOE devices.

6.1 Virtual offloading of IPsec protocol

IPsec is a protocol suite for realizing secure communication by authenticating and encrypting every IP packet [25-27]. IPsec is basically composed of three elements: the header authentication with AH (Authentication Header), the payload encryption with ESP (Encapsulating Security Payload), and the key exchange with IKE (Internet Key Exchange protocol).

IPsec supports variety of hashing and encryption algorithms for header authentication and payload encryption, respectively. It has been known that hashing and encryption algorithms are computing intensive. Hence, packet processing in the IPsec protocol easily becomes the performance bottleneck between end systems.

In what follows, we explain how header authentication and payload encryption of the IPsec protocol in the Linux kernel version 2.6.30 can be virtually offloaded with VOSE. Among several hashing and encryption algorithms implemented in the Linux kernel, we use rather modern algorithms: SHA-1 (Secure Hash Algorithm 1) [28] for header authentication in AH and payload encryption in ESP, and AES (Advanced Encryption Standard) with CBC (Ciphertext Block Chaining) [29] for payload encryption in ESP.

- SHA-1 virtual offloading

  Virtual offloading of header authentication and payload encryption with SHA-1 can be realized by replacing **sha1_update()** function with a short sleep of $T_{S\ SHA\text{-}1}$ and $T_{R\ SHA\text{-}1}$ in both the sender and the receiver, respectively.

- AES virtual offloading

  Virtual offloading of payload encryption with AES can be realized by replacing **aes_encrypt()** and **aes_decrypt()** functions with a short sleep of $T_{S\ AES}$ and $T_{R\ AES}$ in both the sender and the receiver, respectively.

- CBC virtual offloading

  Virtual offloading of payload encryption with CBC can be realized by replacing **crypto_cbc_encrypt()** and **crypto_cbc_decrypt()** functions with a short sleep of $T_{S\ CBC}$ and $T_{R\ CBC}$ in both the sender and the receiver, respectively.

6.2 Experiment

With VOSE-enabled Linux kernel, we experimentally estimate the performance improvement caused by introduction of several types of TOE devices for the IPsec protocol.

We compare performances of five types of TOE devices: AES-only, AES-CBC, SHA1-only, SHA1-AES, and SHA1-AES-CBC, each which corresponds to the case of AES-only offloading, AES and CBC offloading, SHA-1-only offloading, SHA-1 and AES offloading, and full offloading. Note that selection of protocol processings to be offloaded is a design choice of the TOE device. In general, offloading more protocol processings should result in better performance, but as well result in a more expensive TOE device. The crucial problem is to choose the best combination of protocol processings to be offloaded, under a given cost and hardware constraint and performance requirements. As we will see in the following experiments, VOSE is quite effective for estimating performance improvements caused by different types of TOE devices *without* implementing multiple types of TOE devices.

Figure 12 shows the end-to-end performance with different types of TOE devices in the experimental environment B (i.e., 10 Gigabit Ethernet, Core i7 3.2 [GHz] CPU) when the speedup factor of TOE devices is varied. Recall that the speedup factor is the ratio of the TOE processing speed to that of the software processing speed. Note that ``inf'' in the x-axis denotes the infinite speedup factor (i.e., $T_S = T_R = 0$). For comparison purposes, the end-to-end performance without TOE (labeled as `Software') is also shown in the figure.

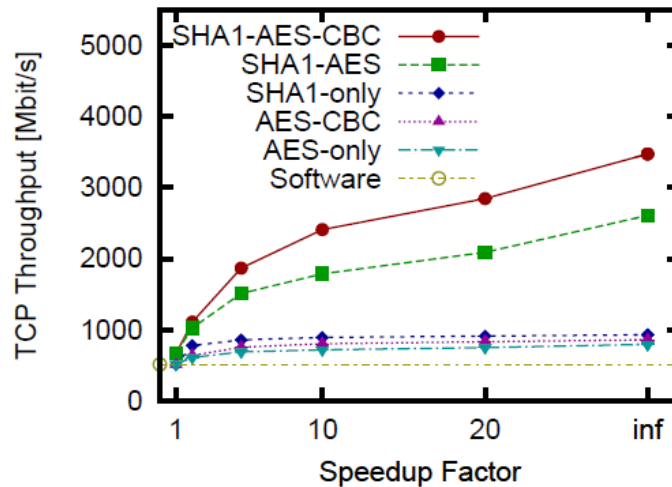


Fig 12. The end-to-end performance (effective throughput) of five types of TOE devices when changing the time of sleep processing

Figure 12 quantitatively show the effectiveness of TOE devices for the header authentication and the payload encryption of the IPsec protocol. This figure clearly indicates that SHA-1 is the heaviest protocol processing among SHA-1, AES, and CBC, and that SHA-1 offloading is absolutely necessary to achieve more than 1 [Gbit/s] throughput. Interestingly, this figure also indicates that SHA-1 alone is not the performance bottleneck of end-to-end communication. For instance, the throughputs with SHA1-AES-CBC and SHA1-AES are high, but throughputs with other TOE devices, including SHA-1-only TOE device, are all less than 1 [Gbit/s]. This means that offloading the heaviest protocol processing alone is not sufficient for achieving high performance. Instead, we need to choose an appropriate combination of protocol processings to be offloaded to the TOE device. This demonstrates that VOSE is quite effective for quantitatively comparing the performances of different types of TOE devices *without* implementing real TOE devices.

Moreover, VOSE tells how fast the TOE device for every protocol processing should be. In Fig. 12, the speedup factor is varied from 1 to infinity. This figure shows that increasing the processing speed of the hardwared does not always contribute to improve the end-to-end performance. For instance, the curve labeled `SHA1-AES-CBC' always increases as the speedup factor increases, which means that increasing the hardware processing speed contributes to the performance improvement when all of SHA-1, AES, and CBC are offloaded onto the TOE device. However, somewhat surprisingly, the curve labeled `SHA-1-only' saturates when the speedup factor is around 10, and further increasing the speedup factor does not improve the end-to-end performance at all. In other words, providing very fast hardwared (i.e., more than ten times faster than the software processing) on the TOE device is simply worthless.

As we have observed, TOE design under several constraint and performance requirements are quite complex and difficult task. Such complexity and difficulty are resulted from non-linearity of the protocol offloading. Namely, the performance improvement caused by multiple-protocol offloading is hard to predict from performance improvements caused by individual-protocol offloadings. For instance, the performance improvement caused by SHA-1 and AES offloading (SHA1-AES in Fig. 12) is hard to predict from performance improvements caused by individual SHA-1 and AES offloadings (SHA1-only and AES-only in Fig. 12. Also, the performance improvement caused by increasing the speedup factor (i.e., providing a faster hardware) changes non-linearly and it is heavily dependent on the protocol processing to be offloaded. As we have demonstrated, VOSE dramatically reduces the burden of TOE designers. With VOSE, TOE designers can easily and flexibly examine the effectiveness of several combinations of TOE designs with experiments.

Consequently, VOSE enables TOE designers to measure performance of the various patterns of offloadings and clarify processings that should be performed with software and the processings that should be performed with hardware, without experiments with a real TOE implementation. Therefore, VOSE optimizes the TOE performance because TOE designers can design the TOE hardware that considered the appropriate balance between the scale of TOE hardware and processing speed.

7. Conclusion

In this paper, we have proposed VOSE, which is a technique for measuring TCP/IP performance improvements derived from different type of TOE devices without implementing TOE devices really. VOSE enables virtual offloading without requiring a hardware TOE device by virtually emulating TOE processing (e.g., bypassing software protocol processing without violating protocol consistency) at both the source and destination end hosts.

The accuracy of virtual offloading with VOSE was thoroughly examined in terms of end-to-end performance and CPU processing overhead. Specifically, we applied VOSE to the TCP checksum calculations in a Linux kernel, and have compared the performance and processing overhead between hardware offloading with a dedicated TOE and virtual offloading with VOSE. Consequently, we have shown that performance improvements which are derived from TOE devices can be estimated correctly.

Moreover, we have applied VOSE to header authenticating and payload encryption in IPsec protocol. We have estimated the performance improvement which are derived from several types of TOE devices of IPsec.

Areas for future study include the virtual offloading of protocol processing other than symmetrical processings and the effectiveness verification of VOSE using a multicore computer.

Reference

[1] Z.Z. Wu and H.C. Chen, "Design and implementation of TCP/IP offload engine system over gigabit ethernet," Proceedings of the 15th International Conference on Computer Communications and Networks (ICCCN 2006)., pp.245–250, Oct. 2006.

[2] J. Postel, "Transmission control protocol," Request for Comments (RFC) 793, Sept. 1981.

[3] S. Floyd, "A report on some recent developments in TCP congestion control," IEEE Communications Magazine, vol.39, no.4, pp.84–90, June 2001.

[4] A. Mark and F. Aaron, "On the eective evaluation of TCP," ACM SIGCOMM Computer Communication Review, vol.29, no.12, pp.59–70, Oct. 1999.

[5] D.D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," IEEE Communications Magazine, vol.27, pp.23–29, June 1989.

[6] N. Bierbaum, "Mpi and embedded tcp/ip gigabit ethernet cluster computing," Proceedings of the 27th Annual IEEE Conference on Local Computer Networks, pp.733–734, Nov. 2002.

[7] A.P. Foong, T.R. Hu, H.H. Hum, J.P. Patwardhan, and G.J. Regnier, "TCP performance re-visited," Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2003), pp.70–79, March 2003.

[8] K.M. Yasuhiro Fukuju and T. Ishihara, "Evalution of hardware-based IPsec processing method for embedded devices," IEICE Technical Report, vol.107, no.378, pp.79–84, Dec. 2007.

[9] W. Feng, P. Balaji, C. Baron, L.N. Bhuyan, and D.K. Panda, "Performance characterization of a 10-gigabit ethernet TOE," Proceedings of the 13th Symposium on High Performance Interconnects, pp.58–63, Aug. 2005.

[10] C.S. Ha, J.H. Lee, D.S. Leem, M.S. Park, and B.Y. Choi, "Asic design of IPsec hard-ware accelerator for network security," Proceedings of the 2004 IEEE Asia-Pacific Conference on Advanced System Integrated Circuits (AP-ASIC 2004)., pp.168–171, Aug. 2004.

[11] W. Benjamin and C. Patrick, "Network I/O acceleration in heterogeneous multicore processors," Proceedings of the 14th IEEE Symposium on High-Performance Interconnects, pp.9–14, Aug. 2006.

[12] D.W. Kim, W.O. Kwon, K. Park, and S.W. Kim, "Internet protocol engine in TCP/IP ooading engine," Proceedings of the 10th International Conference on Advanced Communication Technology(ICACT 2008), pp.270–275, Feb. 2008.

[13] H. Jang, S.H. Chung, and S.C. Oh, "Implementation of a hybrid TCP/IP offoad engine prototype," Proceedings of the 10th Asia Pacific Computer Systems Architecture Conference(ACSAC 2005), pp.464–477, Oct. 2005.

[14] H. Jang, S.H. Chung, and D.H. Yoo, "Design and implementation of a protocol offoad engine for TCP/IP and remote direct memory access based on hardware/software coprocessing," Microprocessors and Microsystems archive, vol.33, no.5-6, pp.333–342, Aug. 2009.

[15] H. Jang, S.H. Chung, D.K. Kim, and Y.S. Lee, "An efficient architecture for a TCP offoad engine based on hardware/software co-design,"Journal of Information Science and Engineering, vol.27, no.2, pp.493–509, March 2011.

[16] H. Ghadia, "Benefits of full TCP/IP offoad (TOE) for NFS services," Proceedings of the 2003 NFS Industry Conference, Sept. 2003.

[17] R. Westrelin, N. Fugier, E. Nordmark, K. Kunze, and E. Lemoine, "Studying network protocol offoad with emulation: approach and preliminary results," Proceedings of the 12th IEEE International Symposium on High Performance Interconnects, pp.84–90, Aug. 2004.

[18] "Safexcel." http://safenet-inc.com/products/chips/index.asp.

[19] "Ixp4xx." http://www.intel.com/design/network/products/npfamily/ixp4xx.htm.

[20] J.C. Mogul, "TCP offoad is a dumb idea whose time has come," Proceedings of the Ninth workshop on hot topics in Operating Systems, May 2003.

[21] D.X. Wei, P. Cao, and S.H. Low, "Time for a TCP benchmark suite?," 2005.

[22] R. Braden, D. Borman, and C. Partridge, "Computing the internet checksum," ACM SIGCOMM Computer Communication Review, vol.19, no.2, pp.86–94, April 1989.

[23] J. Chase, A. Gallatin, and K. Yocum, "End-system optimisation for high-speed TCP," IEEE Communications, vol.39, no.4, pp.68–74, April 2001.

[24] J. Levon, "Oprofile - a system profiler for Linux."http://oprofile.sourceforge.net/.

[25] S. Kent and K. Seo, "Security architecture for the internet protocol,"Request for Comments (RFC) 4301, Dec. 2005.

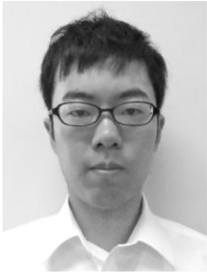
[26] S. Kent, "IP authentication header," Request for Comments (RFC) 4302, Dec. 2005.

[27] S. Kent, "IP encapsulating security payload (ESP)," Request for Comments (RFC) 4303, Dec. 2005.

[28] C. Madson and R. Glenn, "The use of HMAC-SHA-1-96 within ESP and AH," Request for Comments (RFC) 2404, Nov. 1998.

[29] S. Frankel, R. Glenn, and S. Kelly, "The AES-CBC cipher algorithm and its use with IPsec," Request for Comments (RFC) 3602, Sept. 2003.

Author' biography with Photo



received Bachelor of Information Science and Master of Information Science degrees from Osaka University in 2009 and 2011, respectively. He has been a graduate student of the Graduate School of Information Science and Technology, Osaka University since April 2011. He is a member of IEEE and IEICE.

received the B.E. degree from the Chitose Institute of Science and Technology University, Japan, in 2005, and the M.E. degree in Graduate School of Information Science and Technology from Hokkaido University, Japan, in 2007. He is currently working as a Researcher at the Hitachi, Ltd., Yokohama Research Laboratory, Japan. His research interests are data processing hardware architectures for high-speed networks.

received the B.E. and the M.E. degrees in Mechanical Engineering from Tokyo University, Japan, in 2003. He is currently working as a Researcher at the Hitachi, Ltd., Yokohama Research Laboratory, Japan. His research interests are new communication systems for the Smart City.

received the M. E. degree in the Information and Computer Sciences from Osaka University, Osaka, Japan, in 1995. He also received the Ph. D. degree from Osaka University, Osaka, Japan, in 1997. He is currently a professor at Department of Informatics, School of Science and Technology, Kwansei Gakuin University, Japan. His research work is in the area of design, modeling, and control of large-scale communication networks. He is a member of IEEE, IEICE, and IPSJ. His e-mail address is ohsaki@kwansei.ac.jp

Ethernet is a registered trademark of Xerox Corp.

Linux is a registered trademark of Linus Torvalds.

Solaris is a registered trademark of Sun Microsystems, Inc.

PCI Express is a registered trademark of PCI-SIG Corporation.

Intel, IA32, IA64, Intel Pentium, and Intel Core i7 are registered trademarks of Intel Corp.

Debian is a registered trademark of software in the Public Interest, Inc.

All other trademarks are the property of their respective owners.