JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 27, 1871-1883 (2011)

Implementation of a Software-Based TCP/IP Offload Engine Using Standalone TCP/IP without an Embedded OS*

IN-SU YOON, SANG-HWA CHUNG+ AND YOON-GEUN KWON
Department of Computer Engineering
Pusan National University
Busan, 609-735 Korea

A number of TCP/IP offload engines have been developed to reduce the CPU load of processing TCP/IP, but most of them are implemented in hardware. Although hardware-based TOEs have a high performance, they lack the flexibility to accept changes in the TCP/IP. To preserve flexibility, we implemented a software-based TOE, called HL-TCP 100134 (High-performance Lightweight TCP/IP). The HL-TCP is a standalone TCP/IP without an embedded OS. The TOE using the HL-TCP has features of a zero-copy sending mechanism and an efficient DMA mechanism for TCP retransmission. It also fully utilizes offload features in the Ethernet. Our experimental results show that the TOE using the HL-TCP can achieve a bandwidth of 453 Mbps with almost zero CPU utilization, compared with a general gigabit Ethernet, which has a CPU utilization of approximately 23%.

Keywords: TCP/IP offload engine, TOE, TCP/IP, gigabit Ethernet, embedded systems

1. INTRODUCTION

Ethernet technology is rapidly developing towards a bandwidth of 10 Gbps. In such high-speed networks, the traditional method of TCP/IP processing, in which the host CPU processes the TCP/IP, requires extensive computing power [1]. Consequently, little or no processing resources are left for applications. To solve this problem, studies have been made of a TCP/IP offload engine (TOE), in which the TCP/IP is processed on a network adapter instead of the host CPU. The existing TOEs process the TCP/IP by dedicated hardware. Although these ensure high network performance, they are inflexible. TCP/IP are complex and evolving protocols, and although their most basic operations have not changed significantly since RFC 791 and 793, several enhancements have been made over the years. The current Linux kernel uses BIC TCP for congestion control instead of TCP Reno. New protocols, such as IPv6 and RDMA over TCP/IP [2], have emerged and been proposed. In this environment, the hardware-based TOEs cannot easily accept and implement the new features.

To overcome these disadvantages, a software-based TOE may be an alternative to the hardware-based TOE. Previously, we proposed a software-based TOE using embedded Linux [3], which had the following problems. First, it performs frequently context switching between the embedded Linux kernel and application. Second, the fully featured TCP/IP in the TOE was slow. To solve these problems, the TOE must be implemented without an embedded OS. This method also requires a standalone TCP/IP that

Received May 19, 2010; revised September 17, 2010; accepted October 29, 2010.
Communicated by Ce-Kuen Shieh.
* This work was supported by the grant of the Korean Ministry of Education, Science and Technology (The Regional Core Research Program/Institute of Logistics Information Technology).
+ Corresponding author.

can be used without an OS. There are many standalone TCP/IP implementations [4, 5], but most of these are used to provide Internet access capability. That is, they are not targeted at high-performance network interfaces, such as TOE. Therefore, we implemented the following two standalone TCP/IPs. First, we chose an open source standalone TCP/IP, called a lightweight TCP/IP (lwIP) [4]. To adapt the lwIP to the TOE and improve performance, we modified the lwIP. Second, we developed our own TCP/IP, called a high performance lightweight TCP/IP (HL-TCP). With the two TCP/IPs, we implemented two types of software-based TOEs. In this paper, we describe the design of our software-based TOEs and the improved features. In addition, we present experimental and analytical results of the two TOEs.

2. RELATED WORK

The common approaches to implementation of the TOE are restricted to using dedicated hardware, such as an ASIC [2, 6, 7] or an FPGA [8]. Several companies have also announced 10-gigabit Ethernet products. However, few of these NICs are currently available to the public, and very little actual information has been made available concerning their architectures. Although these hardware-based approaches have high performance, these approaches lack the flexibility to meet the evolving TCP/IP protocol suite and require considerable design complexity.

There has been some previous work on hybrid approaches [9-12] to implementing a TOE. The time-consuming tasks are processed by hardware logic in an FPGA and other tasks are processed by software running on an embedded processor, such as an ARM processor. Wu and Chen's TOE [9] processes IP, ARP and ICMP protocols in hardware and processes TCP by software based on an embedded processor. Kim and Rixner implemented a TOE [13, 14] using a programmable gigabit Ethernet. The Ethernet firmware implemented the TCP/IP. They presented a connection handoff interface [14] between the OS and the Ethernet. Using this interface, the OS can offload a subset of TCP connections to the Ethernet, while the remaining connections are processed on the host CPU. Simulation results showed that the number of CPU cycles spent processing each packet decreases by 16.84% [14] and the web server throughput can be improved by 33.72% [13].

3. EMBEDDED SYSTEMS FOR TOE

We used a Cyclone Microsystems PCI-730 [15] to implement the TOEs. Fig. 1 shows a block diagram of the PCI-730. The PCI-730 has the Intel IOP310 I/O processor chipset, 256 MB SDRAM and a gigabit Ethernet that is based on the Intel 82544. The IOP310 chipset contains a 600 MHz XScale CPU and an Intel 80312 I/O companion chip.

From the TOE point of view, there are several interesting features provided by the PCI-730. First, the messaging unit can be used to transfer a TCP/IP processing request and a processed result of the request between the host computer and the TOE. The second interesting feature is the capability of the PCI-to-PCI bridge. Through the bridge, devices attached to the secondary PCI bus can directly access to the primary PCI bus. Because of

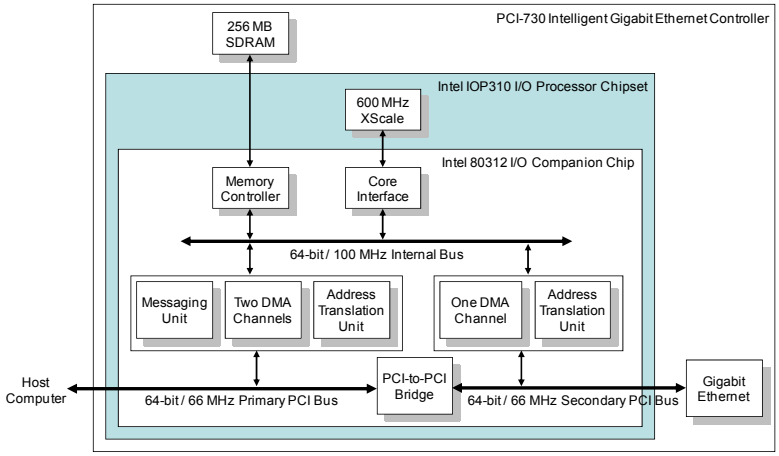


Fig. 1. Block diagram of the PCI-730.

this feature, the Ethernet in the PCI-730 can directly access the host memory for packet data transmission. Also the Intel 82544 has host-offloading features, such as checksum offloading (CSO), TCP segmentation offloading (TSO) and interrupt coalescing. Modern OSes do not execute their own checksum and segmentation codes when the CSO and TSO are available in the Ethernet. The interrupt coalescing, which is a technique that generates a single interrupt for multiple incoming packets, therefore prevents a host from being flooded with too many interrupts. When implementing the TOEs, we fully utilized the offloading features of the Ethernet.

4. EMBEDDED SYSTEMS FOR TOE

4.1 Interface Between Host and TOE

The lwIP and the HL-TCP use the same interface mechanism, as shown in Fig. 2. Most TCP/IP applications use the socket library. The socket library functions can be classified into two categories; namely, nondata transfer functions and data transfer functions. These are handled in a single system call, sys_socketcall. At this level, to bypass the TCP/IP layers in the kernel, we modified the Linux kernel. The sys_socketcall directly calls the toe_socketcall, which is implemented in the TOE device driver. Depending on the type of socket library function, different TCP/IP processing requests are made and pushed into the inbound messaging queue. The request consists of a command, an ID and parameters. The command represents the types of jobs, such as socket creation, binding, send, *etc*. The ID is used to identify the process that posted the job. The parameters contain the data necessary to process the task.

The PCI-730 has a messaging unit that can be used to transfer data between the host system and the embedded system. The system that receives new data is notified by the messaging unit by an interrupt. We used the messaging unit as a queue that accepts a TCP/IP processing request and a processed result of the request. The right half of Fig. 2 shows an example of the processing socket functions through the messaging queue. A

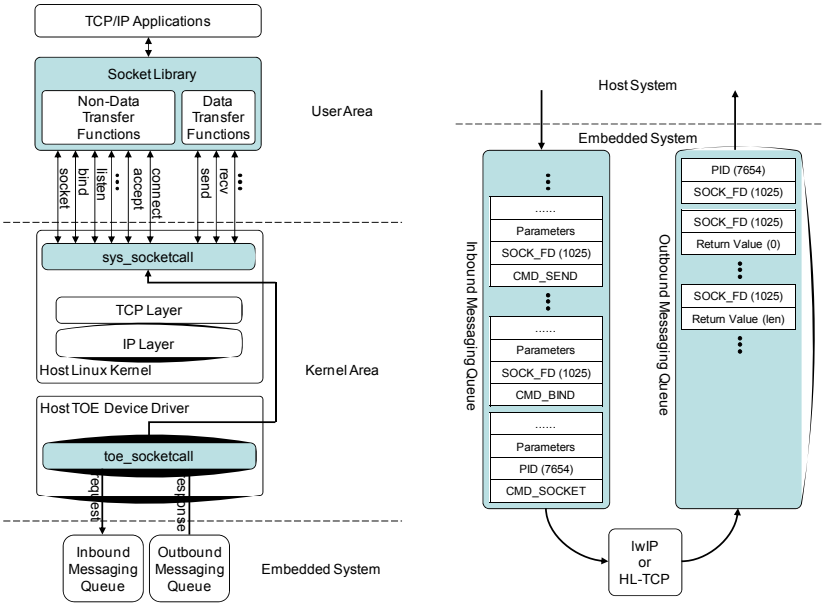


Fig. 2. Interface mechanism between the host and the TOE.

process, which has a process ID (PID) of 7654, requests a socket creation (CMD_SOCKET). The lwIP or HL-TCP processes the request and returns a socket file descriptor (FD) of 1025 with a PID of 7654 through the outbound messaging queue. The process with the PID 7654 is awakened and has the newly created socket FD 1025. Because a process can have multiple sockets and connections, the PID can only be used as a unique identifier in the case of socket creation. Therefore, after creating a new socket, using a socket function, the socket FD is used as a unique identifier instead of the PID. With the socket FD 1025 as a unique identifier, the right half of Fig. 2 shows the processing of a bind and a send function.

4.2 LwIP Improvement

Because the original lwIP was designed for small embedded microcontrollers, it was focused on memory use, not network performance. To improve the performance, we modified the lwIP by the following methods.

First, when receiving a packet, the packet is DMA transferred to the embedded system memory by the Ethernet. After DMA transfer is complete, the lwIP copies the packet to its packet buffer. To eliminate the copy overhead, we modified the network interface layer of the lwIP and the Ethernet driver. By this modification, all received packets are DMA transferred to the packet buffer without a copy.

Second, the original lwIP acknowledges each incoming packet, which increases the embedded system overhead as more packets are transferred. To reduce this overhead, we modified the lwIP to adopt delayed ACK [16]. The delayed ACK is sent for every two packets received or sent after 500 ms passed since the last packet received. This can improve the performance by decreasing the number of ACK packets.

Table 1. Modified buffer sizes of the lwIP to transfer large data.

	Original Size	Modified Size
Packet Buffer Size	128 bytes	1514 bytes
Maximum Segment Size	128 bytes	1460 bytes
TCP Window Size	2 KB	64 KB

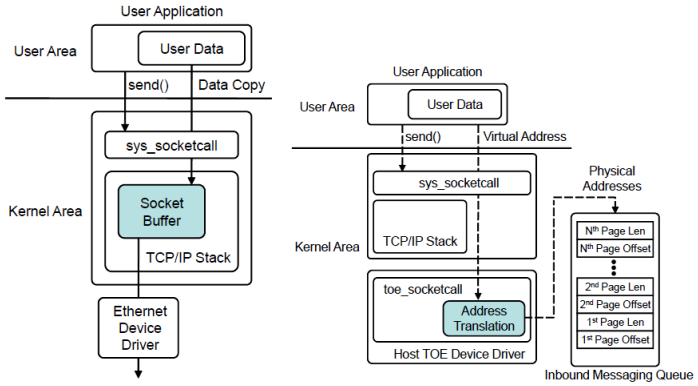
Third, we added the CSO and TSO features to the lwIP. Finally, the original lwIP was developed for a small embedded system that has limited computing resources so its packet buffer size, maximum segment size (MSS) and TCP window size were configured to be smaller than those that are used in the current Internet environment. We adjusted the original sizes of these to transfer large data sets. Table 1 shows both the original and the modified buffer sizes.

4.3 HL-TCP Improvement

Unlike the lwIP, the HL-TCP was originally targeted at TOE. Therefore, all of the improvements as described in section 4.2 were applied in the implementation of the HL-TCP. Moreover, we present a zero-copy sending mechanism and an efficient DMA mechanism for TCP retransmission in this section.

4.3.1 Zero-copy sending mechanism

Fig. 3 (a) shows the processing sequence of the send socket function when using a general gigabit Ethernet. First, the send socket function calls sys_socketcall in the Linux kernel and then copies the user data to the socket buffer. In contrast, the TOE bypasses the Linux TCP/IP stack in the sys_socketcall and calls toe_socketcall in the TOE device driver. And then, it does not copy the user data to the socket buffer but performs an address translation. To translate a virtual address of the user data into a physical address, the get_user_pages and kmap functions in the kernel are used. Finally, the pairs of the starting addresses and lengths of the physical pages are transferred to the TOE using the inbound messaging queue.



(a) General gigabit Ethernet. (b) TOE.

Fig. 3. Comparison between copy and address translation methods.

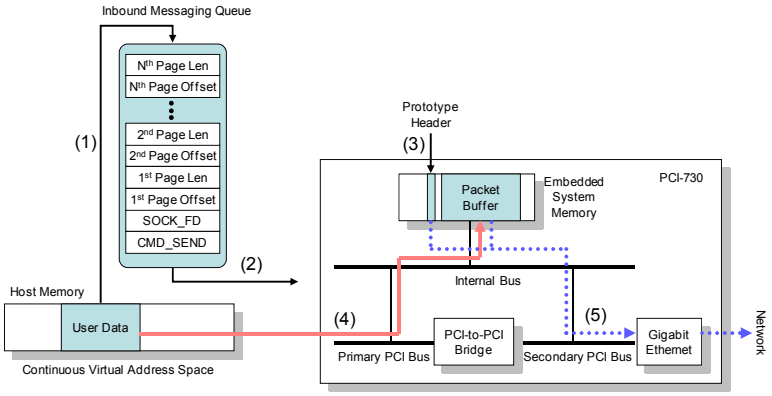


Fig. 4. Packet sending mechanism with data copy.

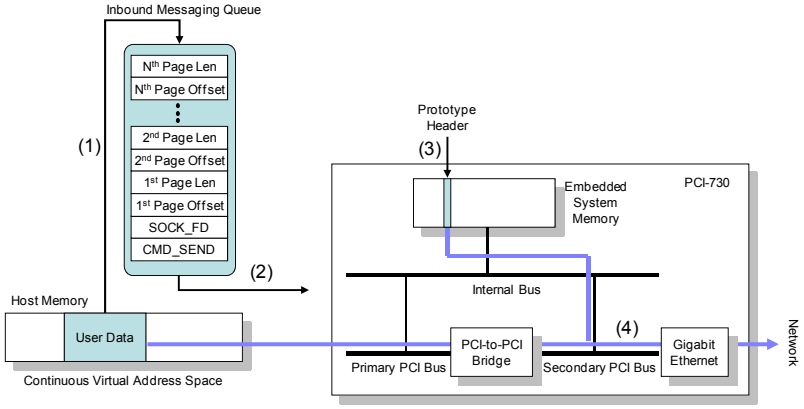


Fig. 5. Zero-copy sending mechanism of the HL-TCP.

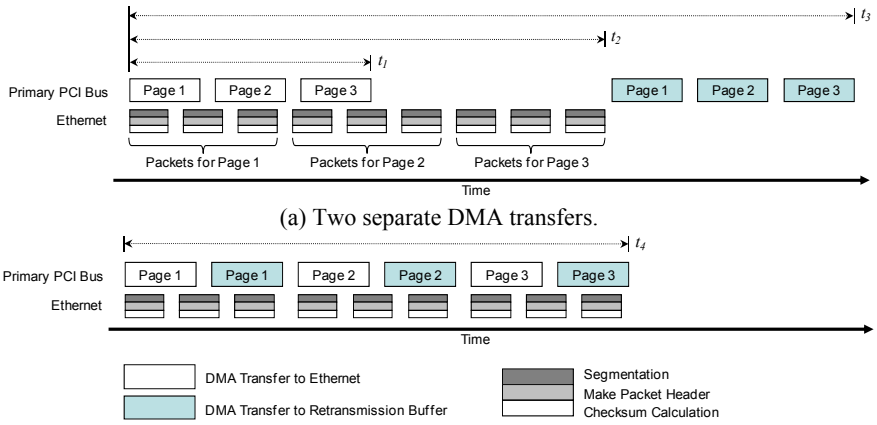
Though we eliminated the kernel copy by bypassing the TCP/IP stack, there exists a copy due to the data structure of the lwIP. The lwIP has a packet buffer, which is similar to the socket buffer in the Linux kernel. The user data have to be copied into the packet buffer first. Then, the copied data are DMA transferred by the Ethernet.

Fig. 4 shows the packet sending mechanism with data copy. After the TOE gets the address and length pairs, it creates a prototype header that is used for the TSO by the Ethernet. After creating the prototype header, the user data are copied into the packet buffer. Finally, the copied data are DMA transferred by the Ethernet, which concurrently performs segmentation and header creation of the data.

To remove this copy, we implemented the HL-TCP without the packet buffer. The address and length pairs are directly used by the DMA engine of the Ethernet and the DMA transactions are performed through the PCI-to-PCI bridge. Fig. 5 shows the mechanism.

4.3.2 Efficient DMA mechanism for TCP retransmission

The method proposed in the previous section is inadequate for TCP retransmission.



(a) Two separate DMA transfers.

(b) Round-robin DMA transfers.

Fig. 6. DMA mechanisms for TCP retransmission.

After sending the data in the host memory, the content of the memory can be changed or swapped out. To solve this problem, we created a retransmission buffer and implemented the HL-TCP to manage it. Fig. 6 shows two different DMA mechanisms when three pages of user data are transferred.

The mechanism using two separate DMA transfers is shown in Fig. 6 (a). The first DMA is triggered by the DMA controller in the Ethernet and the pages are transferred to the Ethernet. Simultaneously, the Ethernet creates and sends out packets using the TSO. Normally, the page size is 4 KB and MSS is 1460 bytes, and a page is divided into three packets. Because creating and sending out the packets requires more time to complete than the DMA read does, it is finished at time t_2, which is longer than time t_1. After sending out the last packet, the second DMA is triggered by the DMA controller in the IOP310 I/O processor chipset. The user data are DMA transferred to the retransmission buffer. This is finished at time t_3. As shown in Fig. 6 (a), this mechanism has an idle time of $t_2 - t_1$.

To utilize the idle time of the primary PCI bus, we implemented the HL-TCP to issue the two DMA transfer requests simultaneously. The DMA requests of the Ethernet and the IOP310 I/O processor chipset are processed in round-robin manner by the PCI bus arbiter. The mechanism using round-robin DMA transfers is shown in Fig. 6 (b) and the two DMA transfers are completed by time t_4. Compared with the method of Fig. 6 (a), our proposed method reduces the completion time by $t_3 - t_4$.

5. EXPERIMENTAL RESULTS

We implemented two TOEs using the lwIP and the HL-TCP based on the PCI-730. As a host system, we used two Pentium IV servers, which have a 1.8 Ghz Intel Xeon processor, 512 MB memory, 64-bit/66 MHz PCI bus and Linux kernel 2.6.18. The two systems were connected using 3COM's SuperStack3 switch. The systems were also equipped with Intel's PRO/1000MT adapter as a general gigabit adapter.

5.1 Performance Enhancements of LwIP and HL-TCP

To measure the performance, we developed a micro benchmark. The micro benchmark was implemented using a standard socket library. It measures the bandwidth and latency from calling the send socket function until the data were placed into the receiver's memory. We repeated each experiment 100 times and averaged the results.

Table 2 shows latencies and bandwidths of the modified lwIP according to the methods described in section 4.2. The latencies of 4-byte data are presented in Table 2 and the bandwidths of 256 KB data are presented because the bandwidths were almost saturated at that size. Because the original lwIP was focused on memory usage, not network performance, it had a latency of 183.19 μs and a very low bandwidth, 10 Mbps. When we increased the sizes of the packet buffer, maximum segment and TCP window for large data transfer; the bandwidth was increased to 43 Mbps, whereas the latency was almost the same as for the original lwIP. Because the minimum latency was measured using 4-byte data, the method did not affect the minimum latency. When we eliminated data copy between the driver and the packet buffer of the lwIP, the bandwidth was increased to 107 Mbps. The latency was 174.88 μs. In the case of the delayed ACK, the bandwidth was increased to 125 Mbps. When we added the TSO and CSO features to the lwIP, the bandwidth was increased to 203 Mbps. The latency was decreased to 126.63 μs because the checksum was calculated by the gigabit Ethernet instead of the lwIP. Eliminating data copy and the TSO feature had the greatest impact on the bandwidth of the TOE.

Table 2. Latencies and bandwidths of the modified lwIP.

Applied Methods	Latency (μs)	Bandwidth (Mbps)
Original lwIP	183.19	10
Increasing Size of Packet Buffer, MSS, TCP Window	183.14	43
Eliminating Data Copy	174.88	107
Delayed ACK	126.15	125
TSO and CSO	126.63	203

Table 3. Latencies and bandwidths of the HL-TCP.

Sending Mechanism	DMA Direction	Latency (μs)	Bandwidth (Mbps)
Copy and Send	Host → TOE (Packet Buffer) TOE (Packet Buffer) → Network	59.45	306
Zero-copy	Host → Network	54.21	350
Simultaneously Send and Copy	Host → Network Host → TOE (Retransmission Buffer)	54.43	345

Table 3 shows the latencies and bandwidths of the HL-TCP according to the sending mechanisms described in section 4.3. The copy and send method in Table 3 copies the user data in the host memory to the packet buffer of the HL-TCP and then the gigabit Ethernet transfers the data in the packet buffer to the network using the TSO and CSO features. It had a latency of 59.45 μs and a bandwidth of 306 Mbps. In the case of zero-copy, the gigabit Ethernet transfers directly from the user data in the host memory to the

network by using one DMA and the PCI-to-PCI bridge unit. The bandwidth was increased to 350 Mbps. The simultaneously send and copy method performs one more DMA compared to the zero-copy method. However, the performance degradation is negligible. As described in section 4.3.2, this is because the DMA transfer to the retransmission buffer is efficiently overlapped with the time when the gigabit Ethernet makes packets and transfers to the network. When we applied the interrupt coalescing feature to the HL-TCP, the bandwidth was increased to 453 Mbps and the latency was 67.23 μs.

5.2 Performance Comparison of the General Gigabit Ethernet and Two TOEs

We measured CPU utilization and bandwidth of the general gigabit Ethernet and the two TOEs. As a benchmark program, we used the Network Protocol Independent Performance Evaluator (NetPIPE) [17] version 4. As shown in Fig. 7, the CPU utilization of the two TOEs was less than 5%, compared with that of the general gigabit Ethernet, which was approximately 17-40%. The CPU utilization of the HL-TCP TOE was almost half that of the lwIP TOE up to 1 KB of data. Because the HL-TCP does not have a packet buffer, unlike the lwIP, the HL-TCP reduces the latency by not executing the buffer management code. As a result, the reduced latency reduces the CPU utilization. However, the CPU utilization of the two TOEs was almost 0% when transferring large data sets. The processing time of the TOE increased with the size of data, whereas the processing time of the host was almost unchanged. The host only performed address translation of the user memory and most TCP/IP processing operations were performed by the TOEs.

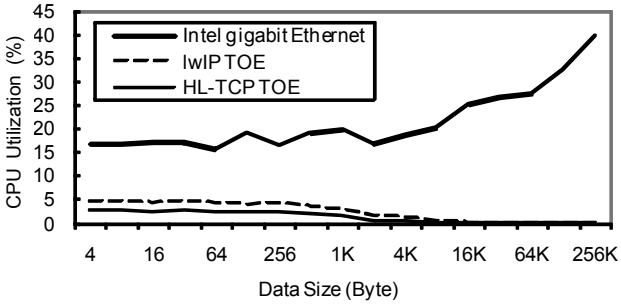


Fig. 7. CPU utilization.

Fig. 8 shows the bandwidth of the two TOEs and the general gigabit Ethernet. The maximum bandwidth of the general gigabit Ethernet was approximately 860 Mbps. Compared with the lwIP and the HL-TCP, it clearly outperforms in terms of bandwidth.

Fig. 9 shows a normalized CPU utilization of the general gigabit Ethernet when it has the same bandwidth as the HL-TCP TOE. In Fig. 8, when the general gigabit Ethernet transfers 13 KB data, it has the same bandwidth of 453 Mbps as the HL-TCP TOE. At the data size of 13 KB, the general gigabit Ethernet has a CPU utilization around 22% in Fig. 7. In this manner, we calculated the normalized CPU utilization of the general gigabit Ethernet for each data size. In Fig. 9, the HL-TCP TOE reduced the CPU utilization by 15.8% on average compared to the general gigabit Ethernet.

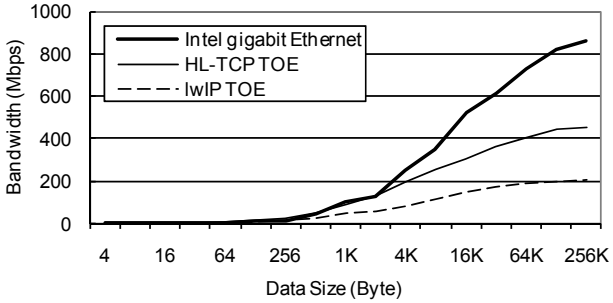


Fig. 8. Bandwidth.

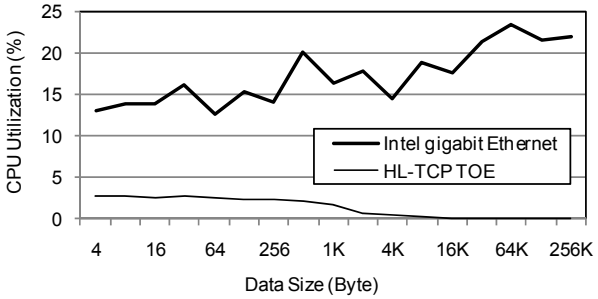


Fig. 9. Normalized CPU utilization.

The TOE is important particularly in a server environment. Servers today use multiple gigabits of bandwidth and high-speed switch. However, the multiple gigabits of bandwidth are not provided to the general users who use the Internet to connect the servers. Nowadays, although the users can purchase the cheap 1 gigabit Ethernet cards, the performance of the commercial network which is used by general user is only about from several dozen bps to 100 Mbps. In the circumstance, lowering CPU utilization to smoothly process multiple user applications is more important than getting more bandwidth from the NIC.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we developed two standalone TCP/IPs that do not need an embedded OS; one is the modified lwIP and the other is the HL-TCP. To improve the performance of the lwIP, we eliminated the data copy between the Ethernet driver and the packet buffer and modified the lwIP to support the TSO and CSO of the Ethernet. The TOE using the modified lwIP had a bandwidth of 203 Mbps. The HL-TCP includes all enhancement features of the modified lwIP. In addition in this paper, we propose the zero-copy sending mechanism and efficient DMA mechanism for TCP retransmission. The zero-copy sending mechanism can be achieved using the TSO and the PCI-to-PCI bridge unit in the embedded system. By overlapping the DMA transactions with the idle time of the PCI bus, the data can be copied efficiently to the embedded system for TCP retransmission. The TOE using the HL-TCP had a bandwidth of 453 Mbps and a latency of 67 μs.

In this paper, we compare the performances of the two TOEs and the general gigabit Ethernet. The experimental results show that the general gigabit Ethernet outperformed the TOEs in terms of bandwidth, but had a higher CPU utilization, 17-40%, compared with the TOEs that have almost 0% CPU utilization.

As a future work, to utilize the flexibility of the HL-TCP TOE, we will implement a security function and QoS policy, which are not supported by the hardware-based TOEs, on the HL-TCP TOE and evaluate the performance characteristics. The recent Intel IOP 342 processor [18] has two processors operating at 1.2 GHz. Using the multi-core processor, we will implement the software-based TOE to process the TCP/IP faster and increase the performance.

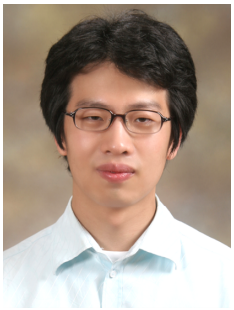
REFERENCES

1. N. Bierbaum, "MPI and embedded TCP/IP gigabit Ethernet cluster computing," in *Proceedings of the 27th Annual IEEE Conference on Local Computer Networks*, 2002, pp. 733-734.
2. D. Dalessandro, P. Wycko, and G. Montry, "Initial performance evaluation of the NetEffect 10 gigabit iWARP adapter," in *Proceedings of the RAIT Workshop at the IEEE Cluster Conference*, 2006, pp. 1-7.
3. I. S. Yoon and S. H. Chung, "Implementation and analysis of TCP/IP offload engine and RDMA transfer mechanisms on an embedded system," in *Proceedings of the 10th Asia-Pacific Computer Systems Architecture Conference*, Vol. 3740, 2005, pp. 818-830.
4. A. Dunkels, "Full TCP/IP for 8-Bit architectures," in *Proceedings of the 1st International Conference on Mobile Systems*, *Applications and Services*, 2003, pp. 85-98.
5. J. Bentham, *TCP/IP Lean: Web Servers for Embedded Systems*, CMP Books, Kansas, 2002.
6. Y. Hoskote, *et al*., "A TCP offload accelerator for 10 Gb/s Ethernet in 90-nm CMOS," *IEEE Journal of Solid-State Circuits*, Vol. 38, 2003, pp. 1866-1875.
7. W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda, "Performance characterization of a 10-gigabit Ethernet TOE," in *Proceedings of the 13th IEEE International Symposium on High-Performance Interconnects*, 2005, pp. 58-63.
8. D. V. Schuehler and J. W. Lockwood, "A modular system for FPGA-based TCP flow processing in high-speed network," in *Proceedings of the 14th International Conference on Field Programmable Logic and its Applications*, Vol. 3203, 2004, pp. 301-310.
9. Z. Z. Wu and H. C. Chen, "Design and implementation of TCP/IP offload engine system over gigabit Ethernet," in *Proceedings of the 15th International Conference on Computer Communications and Networks*, 2006, pp. 245-250.
10. T. H. Liu, H. F. Zhu, C. S. Zhou, and G. R. Chang, "Research and prototype implementation of a TCP/IP offload engine based on the ML403 Xilinx development board," in *Proceedings of the 2nd International Conference on Information and Communication Technologies*, 2006, pp. 3163-3168.
11. H. Jang, S. H. Chung, and S. C. Oh, "Implementation of a hybrid TCP/IP offload engine prototype," in *Proceedings of the 10th Asia-Pacific Computer Systems Archi-*

tecture Conference, Vol. 3740, 2005, pp. 464-477.
12. S. C. Oh, H. Jang, and S. H. Chung, "Analysis of TCP/IP protocol stack for a hybrid TCP/IP offload engine," in *Proceedings of the 5th Parallel and Distributed Computing*: *Applications and Technologies*, Vol. 3320, 2004, pp. 406-409.
13. H. Y. Kim and S. Rixner, "Connection handoff policies for TCP offload network interfaces," in *Proceedings of the Symposium on Operating Systems Design and Implementation*, 2006, pp. 293-306.
14. H. Y. Kim and S. Rixner, "TCP offload through connection handoff," in *Proceedings of the 1st European Conference on Computer Systems*, 2006, pp. 279-290.
15. Cyclone Microsystems PCI-730 Gigabit Ethernet Controller (online), http://www.cyclone.com/products/network_adapters/pci730.php.
16. R. Braden, Requirements for Internet Hosts-Communication Layers, RFC 1122, Network Working Group, Internet Engineering Task Force, October 1989.
17. D. Turner, A. Oline, X. Chen, and T. Benjegerdes, "Integrating new capabilities into NetPIPE," in *Proceedings of the 10th European PVM/MPI Users' Group Meeting*, Vol. 2840, 2003, pp. 37-44.
18. Intel IOP 342 Processor (online), http://www.intel.com/design/iio/iop341_42.htm.

In-Su Yoon received the B.S., and Ph.D. degrees in Computer Engineering from Pusan National University, Busan, Korea, in 2001 and 2009, respectively. He currently works for Samsung Electronics Co., Ltd., Suwon, Korea. His research interests include wireless communications, mobile devices, handover, and mobile operating system.

Sang-Hwa Chung received the B.S. degree in Electrical Engineering from Seoul National University in 1985, the M.S. degree in Computer Engineering from Iowa State University in 1988, and the Ph.D. degree in Computer Engineering from the University of Southern California in 1993. Since 1994, he has been with Pusan National University, where he is currently a Professor in the Department of Computer Engineering, Pusan National University. His research interests include wired/wireless network, RFID system, embedded system, computer architecture, and high-performance computing.



Yoon-Geun Kwon received the B.S. degree in Computer Engineering from Pusan National University, Busan, Korea, in 2009. He is currently doing Master course in the Department of Computer Engineering, Pusan National University. His research interests include RFID system and embedded system.