임베디드 시스템상에서 Lightweight TCP/IP를 이용한 TCP/IP Offload Engine의 구현

(Implementation of a TCP/IP Offload Engine Using Lightweight TCP/IP on an Embedded System)

윤 인 수 † 정 상 화 †† 최 봉 식 ††† 전 용 태 ††††
(In-Su Yoon) (Sang-Hwa Chung) (Bong-Sik Choi) (Yong-Tae Jun)

요 약 현재 네트워크 기술이 기가비트급의 속도를 넘어 급속히 발전하고 있다. 그러나 호스트에서 TCP/IP를 처리하는 기존의 방식은 고속 네트워크 환경에서 호스트 CPU에 많은 부하를 야기한다. 이러한 문제를 해결하기 위해 네트워크 어댑터에서 TCP/IP를 처리하는 TCP/IP Offload Engine(TOE)에 대한 연구가 최근 활발히 진행되고 있다. 본 논문에서는 두 가지의 소프트웨어 기반 TOE를 기가비트 이더넷 환경 하에서 개발하였다. 하나는 임베디드 리눅스를 사용하여 구현한 TOE이고, 다른 하나는 Lightweight TCP/IP(lwIP)를 사용하여 구현한 TOE이다. 임베디드 리눅스를 사용한 TOE는 문맥 전환(context switch), 프로세스 대기 및 활성화 그리고 운영체제 자체의 부하로 인하여 62Mbps의 낮은 대역폭을 보였다. 본 논문에서는 임베디드 리눅스를 사용한 TOE의 성능을 개선하기 위하여 운영체제 없이 lwIP를 이용하여 TOE를 구현하였다. 그리고 이러한 lwIP를 이용한 TOE의 성능을 높이기 위하여 lwIP의 메모리 복사를 제거하고, 지연 ACK 기능과 TCP Segmentation Offload(TSO)기능을 추가하였으며, lwIP가 큰 데이타를 전송할 수 있도록 수정하였다. 그 결과, lwIP를 이용한 TOE는 194Mbps의 대역폭을 보였다.

키워드 : TOE, 임베디드 시스템, TCP/IP, 기가비트 이더넷

Abstract The speed of present-day network technology exceeds a gigabit and is developing rapidly. When using TCP/IP in these high-speed networks, a high load is incurred in processing TCP/IP protocol in a host CPU. To solve this problem, research has been carried out into TCP/IP Offload Engine (TOE). The TOE processes TCP/IP on a network adapter instead of using a host CPU; this reduces the processing burden on the host CPU. In this paper, we developed two software-based TOEs. One is the TOE implementation using an embedded Linux. The other is the TOE implementation using Lightweight TCP/IP (lwIP). The TOE using an embedded Linux did not have the bandwidth more than 62Mbps. To overcome the poor performance of the TOE using an embedded Linux, we ported the lwIP to the embedded system and enhanced the lwIP for the high performance. We eliminated the memory copy overhead of the lwIP. We added a delayed ACK and a TCP Segmentation Offload (TSO) features to the lwIP and modified the default parameters of the lwIP for large data transfer. With the aid of these modifications, the TOE using the modified lwIP shows a bandwidth of 194 Mbps.

Key words : TOE, Embedded System, TCP/IP, Gigabit Ethernet

· 본 연구는 정보통신연구진흥원지원 기초기술연구지원사업(과제번호: 05-기초-083)으로 수행한 연구결과입니다.
† 학생회원 : 부산대학교 컴퓨터공학과
isyoon@pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수/컴퓨터및정보통신연구소 연구원
shchung@pusan.ac.kr
(Corresponding author임)
††† 비 회 원 : 삼성전자 무선사업부 연구원
guyver3@pusan.ac.kr
†††† 비 회 원 : 부산대학교 컴퓨터공학과
ytjun@pusan.ac.kr
논문접수 : 2006년 1월 15일
심사완료 : 2006년 4월 7일

1. 서 론

현재 이더넷을 기반으로 하는 네트워크 기술은 1 Gbps의 대역폭이 일반화되고 있으며, 10 Gbps의 대역폭이 실용화 단계에 있다. 그러나 호스트에서 TCP/IP와 같은 통신 프로토콜을 처리하는 기존의 방식은 네트워크가 고속화될수록 호스트 CPU에 많은 부하를 준다. 결국 이는 전체 시스템의 성능을 저하시킨다[1]. 이러한 문제점을 극복하기 위해 네트워크 어댑터에서 TCP/IP

를 처리하는 TOE(TCP/IP Offload Engine) 기술에 대한 연구가 활발히 진행되고 있다.

TOE는 크게 두 가지 방법으로 개발이 이루어지고 있다. 첫 번째 방법은 TCP/IP를 하드웨어적으로 처리하는 전용 칩(ASIC)을 개발하는 방법이다[2]. 이 방법은 통신 성능이 우수한 반면, 하드웨어로 구현되었기 때문에 추후 발생할 수 있는 TCP/IP의 변경사항들 및 TCP/IP를 기반으로 하는 새로운 상위수준 프로토콜을 추가하기 어렵다. 즉, 구조의 유연성 측면에서 단점을 보인다. 두 번째 방법은 임베디드 프로세서를 사용하여 소프트웨어적으로 TCP/IP를 처리하는 방법이다[3]. 이 방법은 하드웨어 방식에 비해 구현이 쉽고 유연성이 뛰어난 장점을 가지지만 상대적으로 성능이 떨어진다는 단점을 가진다.

소프트웨어 기반 TOE는 임베디드 운영체제를 이용하여 구현하는 방법과 운영체제 없이 TCP/IP를 처리하는 전용 프로그램을 이용하여 구현하는 방법이 있다. 임베디드 운영체제를 이용하는 방법은 커널내의 TCP/IP 스택과 기존의 소켓 라이브러리를 사용하여 TOE를 구현할 수 있다. 그러나 운영체제를 사용함으로써 발생하는 부하가 크기 때문에 높은 성능을 내지 못한다[3]. 이러한 문제점을 해결하기 위해 운영체제 없이 TCP/IP를 처리하는 전용 프로그램을 이용하여 TOE를 구현할 수 있다. 본 논문에서는 이러한 두 가지 방식의 소프트웨어 기반 TOE를 구현하였다. 임베디드 운영체제로는 리눅스를 사용하여 구현하였고, 운영체제 없이 TOE를 구현하기 위해 lwIP를 사용하였다. 또한 lwIP를 개선하여 성능을 향상시켰다. 실험을 통하여 두 가지 방식의 TOE 성능을 비교, 분석하였다.

2. 관련연구

하드웨어 기반 TOE의 최초 구현 사례로는 Alacritech사의 SLIC(Session-Layer Interface Control) 기술을 채택한 네트워크 어댑터들이 있다[4]. 이 어댑터들은 TCP/IP를 이용한 데이타 전송부분만을 Offload하고, TCP 연결 설정등과 같은 작업들은 여전히 호스트에서 처리한다. 최근의 하드웨어 기반 TOE는 Chelsio사의 T210이 있다. 이는 어댑터에 여러 개의 하드웨어 유닛들을 내장해서 병렬로 TCP/IP를 처리하는 구조이다[2]. 데이타 전송부분의 Offload외에 TCP 연결 설정 및 혼잡제어 기능 등을 어댑터에서 추가로 수행한다. 서론에서도 언급하였듯이, 이러한 하드웨어 기반 TOE들은 성능이 높은 반면, 구조의 유연성 측면에서 단점을 보인다.

소프트웨어 기반 TOE는 크게 두 가지의 방법으로 구현할 수 있다. 첫 번째 방법은 리눅스와 VxWorks 등과 같이 커널에 포함된 TCP/IP와 소켓 라이브러리를 이용하여 구현하는 것이다. 이러한 방법은 문맥 전환, 프로세스 대기 및 활성화 그리고 운영체제 자체의 부하로 인해 성능이 높지 못하다. 두 번째 방법은 운영체제 없이 TCP/IP를 처리하는 전용 소프트웨어를 사용하여 TOE를 구현하는 것이다. 이러한 전용 소프트웨어들 중에서 소스코드가 공개되어 있는 대표적인 것으로 lwIP (Lightweight TCP/IP)[5]를 들 수 있다. 원래 lwIP는 자원이 제한된 소형 임베디드 시스템을 위하여 개발되었다[6]. 따라서 lwIP의 주요 목적은 메모리 사용량의 최소화에 있다. lwIP의 주요 구성요소는 크게 4가지이다. 첫 번째는 IP, ICMP, UDP, TCP와 같은 프로토콜 처리 모듈이다. 두 번째는 메모리 관리 시스템, 세 번째는 하위 네트워크 인터페이스 함수들이다. 마지막으로 lwIP는 운영체제 에뮬레이션 계층을 가지고 있다. 운영체제 에뮬레이션 계층은 lwIP가 운영체제가 있는 환경에 적용될 때, lwIP와 운영체제간의 인터페이스를 담당한다. 만약 lwIP가 운영체제 상에서 동작한다면, 개발자는 lwIP에서 제공하는 상위수준의 API를 가지고 BSD 소켓 API를 사용하는 것과 유사한 방법으로 TCP/IP를 사용할 수 있다. 본 연구에서는 운영체제 없이 lwIP를 사용하여 TOE를 구현하였기 때문에 이러한 운영체제 에뮬레이션 계층을 사용하지 않았으며, 상위 수준의 API가 아닌 raw API를 사용하였다.

3. 소프트웨어 기반 TOE의 구현

본 장에서는 본 연구에서 구현한 두 가지 방식의 소프트웨어 기반 TOE를 설명한다. 첫 번째는 임베디드 리눅스를 이용한 TOE의 구현이고, 두 번째는 운영체제 없이 lwIP를 이용한 TOE의 구현이다. 본 장의 순서는 다음과 같다. 먼저 TOE를 개발하기 위한 개발환경과 호스트 측의 구현 사항을 설명한다. 다음으로 임베디드 리눅스를 이용한 TOE를 설명하며, 이어서 lwIP를 이용한 TOE를 설명한다. 마지막으로 성능을 개선하기 위해 lwIP를 수정한 방법을 설명한다.

3.1 TOE의 개발 환경

TOE를 구현하기 위한 타켓 보드로서 Cyclone Microsystems사의 PCI-730 카드[7]를 사용하였다. PCI-730 카드는 Intel사의 600MHz XScale CPU와 Intel사의 82544 기가비트 이더넷 칩을 탑재하였다. 그림 1은 PCI-730 카드의 블록 다이어그램을 나타낸다.

3.2 호스트 측의 구현

기존에 작성된 소켓 응용 프로그램들이 호스트 커널의 TCP/IP를 거치지 않고 바로 TOE를 사용하기 위해

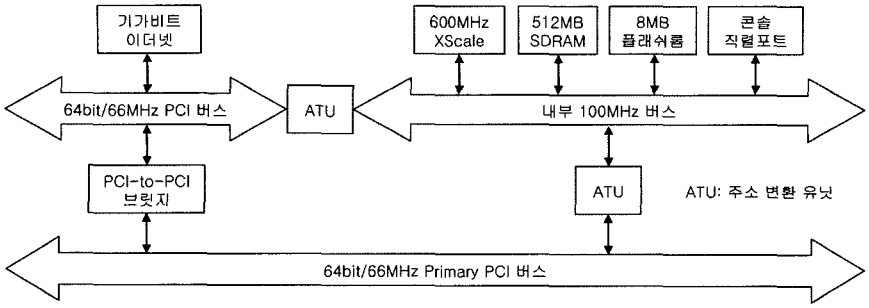

그림 1 PCI-730의 블록 다이어그램

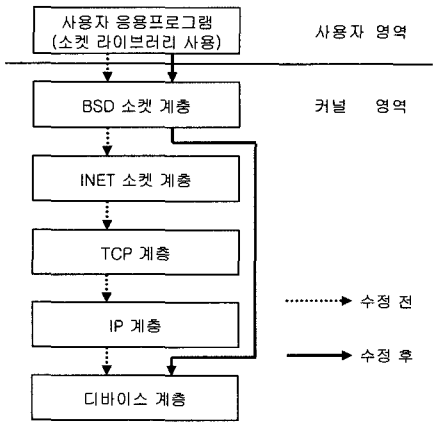

그림 2 호스트 커널의 수정 전후의 TCP/IP 스택 구조도

서는 호스트 커널의 수정이 필요하다. 이를 위해 본 연구에서는 호스트 컴퓨터의 리눅스 커널 버전 2.4.27을 수정하였다. 수정된 리눅스 커널 소스는 *include/linux/* 디렉터리의 *netdevice.h, socket.h* 파일과 *net/* 디렉터리에 있는 *socket.c* 파일이다. 그림 2는 수정하기 전후의 TCP/IP 프로토콜 계층구조를 비교하고 있다. 그림 2에서 보는 바와 같이 사용자 응용프로그램은 기존의 소켓 라이브러리를 그대로 사용한다. 그리고 BSD 계층에서는 소켓 함수들의 수행을 커널을 거치지 않고 바로 디바이스 계층으로 넘긴다. 기존 소켓 응용 프로그램들이 TOE를 사용하기 위해서는, 소켓을 생성하는 socket() 함수의 인자에 TOE를 사용한다고 설정해주면 된다.

3.3 임베디드 리눅스를 이용한 TOE의 구현

임베디드 리눅스를 이용하여 TOE를 구현하기 위해 본 연구에서는 *임베디드 응용 프로그램*과 *임베디드 커널 모듈*을 구현하였다. 임베디드 커널 모듈은 호스트와 임베디드 응용프로그램간의 인터페이스를 제어하는 역할을 한다. 임베디드 응용 프로그램은 소켓 라이브러리를 사용하여 작성되었으며, 임베디드 리눅스와 함께 TCP/IP를 처리한다.

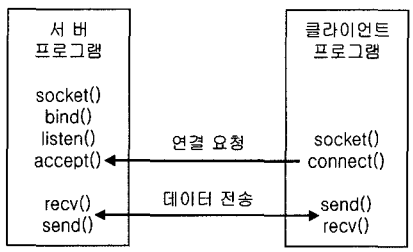

그림 3 일반적인 TCP/IP 서버, 클라이언트 모델

그림 3은 TCP/IP를 사용하는 일반적인 서버와 클라이언트 모델을 보여준다. 소켓 라이브러리 함수들은 크게 두 부분으로 나눌 수 있다. 하나는 send()와 recv()와 같은 데이타 전송과 관련된 함수들이고, 나머지 하나는 socket(), listen(), accept()등과 같은 TCP/IP 연결과 관련된 함수들이다. 본 장에서는 이와 같이 두 가지로 나누어서 동작을 설명한다.

3.3.1 소켓 라이브러리의 연결 관련 함수 처리

그림 4는 소켓을 생성할 때 사용되는 socket() 함수의 처리과정을 보여준다. 그림 4의 각각의 원 문자에 해당하는 설명을 아래에서 한다.

호스트의 사용자 응용프로그램이 *socket()*을 호출하면, BSD 소켓 계층은 디바이스 드라이버에 구현된 *toe_socket()* 함수를 호출한다(①). 디바이스 드라이버는 임베디드 커널 모듈에 소켓 함수를 구분할 수 있는 구분자와 처리에 필요한 정보를 전달함과 동시에 인터럽트를 보낸다(②). 디바이스 드라이버는 *toe_socket()*을 호출한 사용자 프로세스를 대기 상태로 만든다. 인터럽트를 받은 임베디드 커널 모듈은 임베디드 응용 프로그램을 깨운다(③). 깨워진 임베디드 응용 프로그램은 주어진 구분자와 처리에 필요한 정보를 가지고 *socket()* 함수를 수행한다(④). *socket()*의 수행이 끝난 후, 임베디드 응용 프로그램은 *socket()*의 결과값을 *ioctl()* 함수를 통해 임베디드 커널 모듈에 전달한다(⑤). 임베디드 커널 모듈은 호스트의 디바이스 드라이버에 결과값과

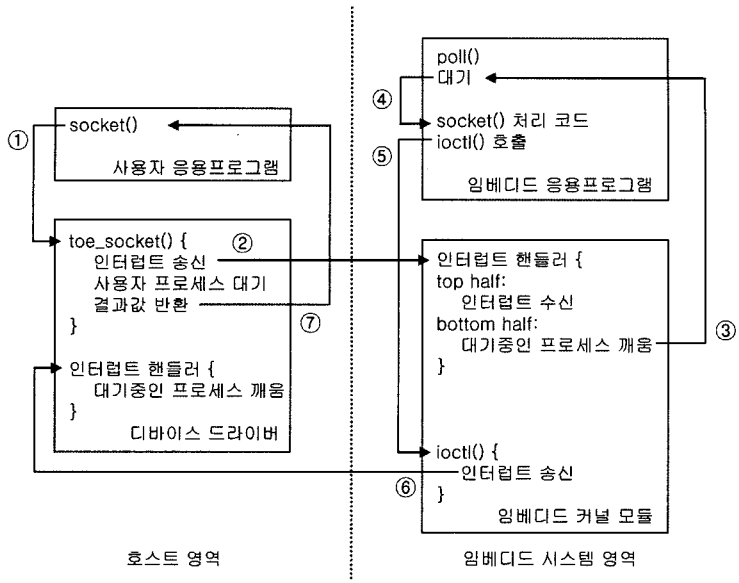

그림 4 임베디드 리눅스를 이용한 TOE의 연결 관련 함수 처리도

함께 인터럽트를 보낸다(⑥). 이 인터럽트는 대기 상태에 있던 호스트 디바이스 드라이버의 *toe_socket()* 함수를 깨운다. 마지막으로 *toe_socket()* 함수는 호스트 측의 사용자 응용프로그램에게 결과값을 전달한다(⑦).

3.3.2 소켓 라이브러리의 데이타 전송 함수 처리

그림 5는 데이타를 송신할 때 사용되는 send() 함수의 처리과정을 보여준다. 데이타 전송 함수를 처리하는 과정은 연결 관련 함수를 처리하는 과정과 유사하다. 단지 호스트 영역에 있는 데이타를 임베디드 시스템에 전달하기 위하여 두 가지 작업이 추가된다. 하나는 호스트 사용자 영역 메모리의 정보를 임베디드 커널 모듈에 전달하는 것이고, 나머지 하나는 이러한 메모리 정보를 통해 카드 측에서 DMA를 하는 것이다.

호스트의 사용자 응용프로그램이 *send()*를 호출하면

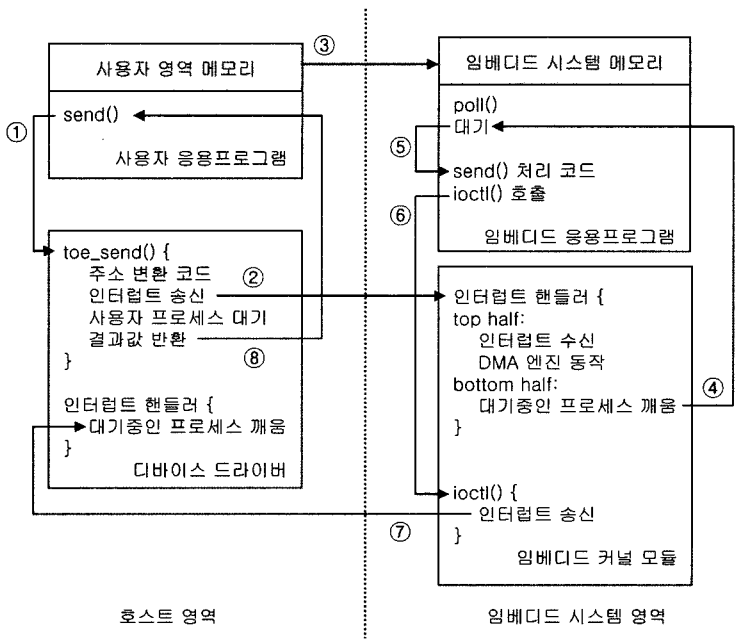

그림 5 임베디드 리눅스를 이용한 TOE의 데이타 전송 함수 처리도

디바이스 드라이버에 구현된 *toe_send()* 함수를 호출한다(①). 디바이스 드라이버는 사용자 영역 메모리의 가상 주소를 가지고 물리적인 주소와 길이의 쌍들을 구한다. 구해진 쌍들을 디바이스 드라이버는 임베디드 커널 모듈에 전달하고 인터럽트를 보낸다(②). 임베디드 커널 모듈의 인터럽트 핸들러에서는 호스트가 보낸 주소 정보를 DMA 컨트롤러에 전달한다. DMA 컨트롤러는 사용자 영역 메모리에 있는 데이타를 임베디드 시스템 메모리로 전송한다(③). 이후 임베디드 커널 모듈은 임베디드 응용 프로그램을 깨운다(④). 깨워진 임베디드 응용 프로그램은 *send()* 함수를 통해서 임베디드 시스템에 있는 데이타를 전송한다(⑤). 데이타 전송 후, 결과값을 전달하는 과정은 연결 관련 함수 처리과정과 동일하다.

3.4 lwIP를 이용한 TOE의 구현

본 절에서는 운영체제를 사용하지 않는 TOE의 구현을 기술하도록 하겠다. 리눅스를 이용하여 TOE를 구현할 경우, 리눅스를 통해 TCP/IP 처리를 할 수 있었다. 하지만 운영체제를 사용하지 않고 TOE를 구현하기 위해서는 TCP/IP를 위한 전용 소프트웨어가 필요하다. 본 논문에서는 TCP/IP를 위한 전용 소프트웨어로 lwIP를 이용하여 TOE를 구현하였다.

3.4.1 소켓 라이브러리 함수 처리

그림 6은 운영체제 없이 lwIP를 이용하여 구현한 TOE의 소켓 라이브러리 함수의 처리과정을 보여준다. 호스트 측의 구조와 동작 과정은 앞에서 기술한 임베디드 리눅스를 이용한 TOE의 호스트 측과 동일하다. 그림 6에서 보이듯이 lwIP를 이용한 TOE는 운영체제를 이용하지 않기 때문에 임베디드 시스템 영역은 응용 프로그램으로만 구성된다. 따라서 임베디드 응용 프로그램은 프로세스의 대기 및 깨우는 작업 없이 동작한다.

그림 6은 lwIP를 이용한 TOE의 소켓 라이브러리 함수의 처리과정을 보여주고 있다. *while* 루프를 돌고 있는 임베디드 응용 프로그램은 호스트에서 보내온 인터럽트에 의해 인터럽트 핸들러로 제어권을 넘긴다(①). 인터럽트 핸들러에서는 인터럽트와 함께 호스트에서 전달한 구분자와 소켓 함수 처리에 필요한 정보들을 저장한다. 만약 호스트 측에서 *send()* 명령을 전달한 경우, DMA를 통해 사용자 영역 메모리를 임베디드 시스템 메모리로 전송하는 작업이 추가된다. 인터럽트 핸들러의 동작이 종료되면 프로그램의 제어권은 다시 *while* 루프로 넘어간다(②). *while* 루프에서는 앞서 저장된 구분자와 소켓 함수 처리에 필요한 정보들을 가지고 lwIP의 raw API를 사용하여 소켓 함수들을 처리한다. 처리 후의 결과값은 인터럽트와 함께 호스트에 전달된다(③).

3.5 성능 향상을 위한 lwIP의 개선

첫째, lwIP와 이더넷 디바이스 드라이버간의 메모리 복사 문제가 있다. 그림 7에서 보이듯이 lwIP는 패킷 처리를 위해서 전용 패킷 버퍼를 가지고 있다. 또한 이더넷 디바이스 드라이버도 수신된 패킷을 저장하기 위한 버퍼를 따로 가지고 있다. 기존의 lwIP는 이더넷 디바이스 드라이버의 버퍼에서 lwIP의 패킷 버퍼로 복사하도록 구현되어 있다. 본 연구에서는 이러한 메모리 복사를 없애기 위하여 lwIP의 이더넷 계층과의 인터페이스 및 이더넷 디바이스 드라이버를 수정하였다. 즉, 수신된 패킷들이 lwIP의 패킷 버퍼로 DMA되도록 구현하였다.

둘째, 기존 lwIP의 ACK(acknowledgement) 전송 방식은 매 데이타 패킷을 수신할 때마다 ACK를 전송한다. 이러한 방식은 수신되는 데이타가 많을수록 임베디드 시스템에 부하를 가중시킨다. 본 연구에서는 이러한 문제점을 개선하기 위하여 lwIP의 ACK 전송방식을 지연 ACK(delayed ACK) 방식으로 수정하였다. 구현된 지연 ACK는 다음 두 가지 조건 중 하나를 만족할 때

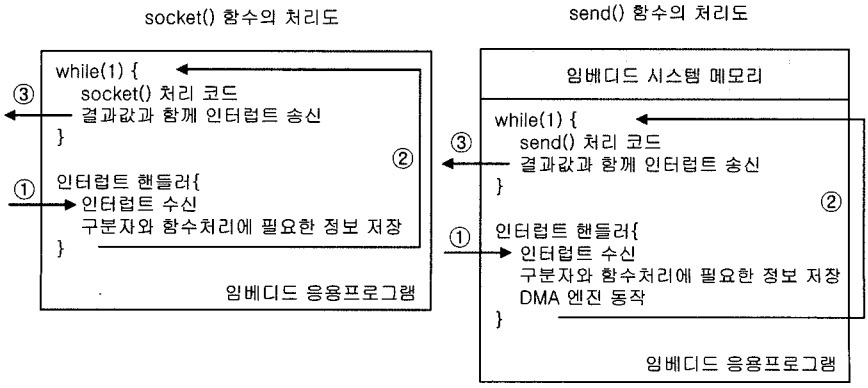

그림 6 lwIP를 이용한 TOE의 소켓 라이브러리 함수 처리도

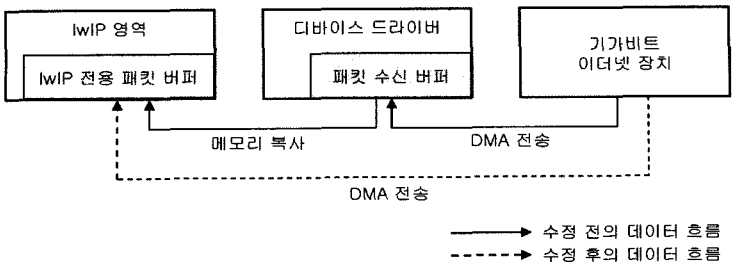

그림 7 lwIP를 이용한 TOE의 메모리 복사 제거

발생한다. 첫째, 두 개의 패킷을 수신하였을 때, 둘째, 마지막 패킷을 받은 지 500ms가 경과했을 때이다. 이러한 지연 ACK 방법은 기존 lwIP의 ACK 전송 방식보다 ACK 패킷의 수를 줄일 수 있어 성능 면에서 이점을 볼 수 있다.

셋째, 기존 lwIP는 자원이 적은 소형 임베디드 시스템을 위하여 개발되었다. 따라서 lwIP 내부의 패킷 버퍼 크기, 최대 세그먼트 크기(Maximum Segment Size) 및 TCP 윈도우 크기가 일반적인 인터넷 환경에서 사용되고 있는 값들보다 적게 설정되어 있다. 본 연구에서는 이러한 디폴트 값을 수정하여 lwIP가 대용량의 데이타를 처리할 수 있도록 하였다. 아래 표 1은 디폴트 값과 본 연구에서 수정한 값을 나타낸다.

표 1 대용량 데이타 전송을 위해서 수정한 lwIP의 설정값

	lwIP의 디폴트 값	수정한 값
패킷 버퍼 크기	128바이트	1514바이트
최대 세그먼트 크기	128바이트	1460바이트
TCP 윈도우 크기	2048바이트	64K바이트

표 1에서 수정한 값들은 현재 리눅스 운영체제가 사용하고 있는 값들을 바탕으로 수정하였다. TCP 윈도우 크기를 리눅스 운영체제가 사용하고 있는 크기인 64K바이트로 수정하였다. 패킷 버퍼의 크기는 일반적인 이더넷 패킷 크기인 1514바이트로 수정하였으며, 최대 세그먼트 크기는 수정한 패킷 버퍼의 크기에서 헤더 크기 54바이트를 뺀 1460바이트로 수정하였다.

마지막으로, 본 연구에서는 TCP Segmentation Offload(TSO) 기능을 lwIP에 추가하였다. 이더넷이 한 번에 보낼 수 있는 최대 크기보다 큰 데이타가 상위 계층에서 내려오면, 이를 이더넷이 보낼 수 있는 단위로 나누는 작업이 필요하다. 이러한 작업을 세그멘테이션이라고 부른다. 세그멘테이션은 일반적으로 TCP 계층에서 수행되는데, 이를 이더넷 칩에서 담당하도록 하는 것을 TCP Segmentation Offload(TSO)라고 부른다. 이러한 TSO는 임베디드 프로세서가 처리해야 할 일을 이더넷 칩에서 처리함으로써 프로세서의 부담을 덜어줄 수 있다. 따라서 본 연구에서는 이더넷 칩이 TSO 기능을 지원할 때, lwIP가 이를 사용할 수 있도록 수정하였다.

4. 실 험

본 연구에서 구현한 임베디드 리눅스를 이용한 TOE와 lwIP를 이용한 TOE의 성능을 측정하였다. 첫째, 일반 기가비트 어댑터를 사용하였을 때와 본 논문에서 구현한 TOE를 사용하였을 때의 CPU 사용률을 측정하였다. 둘째, 구현한 두 TOE의 대역폭을 측정하여 성능을 비교하였다. 셋째, 본 연구에서 개선한 lwIP의 성능 변화를 측정하였다.

실험 환경으로 1.8GHz Intel Xeon 프로세서, 512MB 메인 메모리 및 64bit/66MHz PCI 슬롯을 가진 두 대의 컴퓨터를 사용하였다. 3COM사의 SuperStack3 스위치를 사용하여 노드들을 연결하였다. 호스트 컴퓨터의 운영체제로는 리눅스 커널 2.4.27을 사용하였다. 성능 비교를 위한 일반 기가비트 이더넷 어댑터로는 Intel사의 PRO/1000MT Server Adapter를 사용하였다. 실험 방법은 호스트의 사용자 응용 프로그램에서 send() 함수의 호출로부터 데이타가 원격 노드의 수신버퍼에 들어갈 때까지의 지연시간과 대역폭을 측정하였다. 측정치는 4바이트에서 256K 바이트까지의 데이타 전송을 100회 반복한 평균치이다.

4.1 호스트 CPU의 사용률

그림 8에서 보이듯이 일반 기가비트 이더넷 어댑터를 장착한 시스템의 경우 호스트 CPU 사용률은 데이타 크기에 따라 30~70% 정도인 반면에, TOE를 장착한 시스템의 경우 CPU 사용률은 약 6% 이하로 측정되었다. 데이타 크기가 커질수록 호스트에서보다 TOE에서의 작업 시간이 더 많아지기 때문에 TOE의 CPU 사용률은 거의 0%에 가까워진다.

4.2 임베디드 리눅스를 이용한 TOE와 lwIP를 이용한 TOE와의 성능 비교

측정에 사용된 lwIP는 3.5절에서 설명한 메모리 복사의 제거, 지연 ACK 및 TSO의 사용, 그리고 큰 데이타 전송을 위해 인자 수정이 된 것이다.

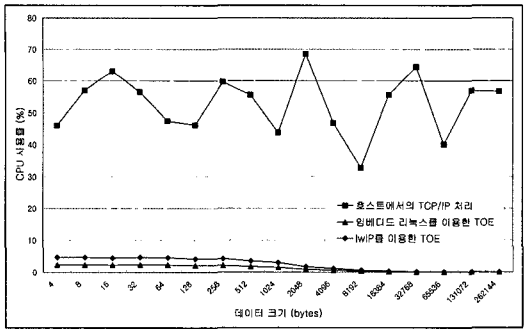

그림 8 CPU 사용률

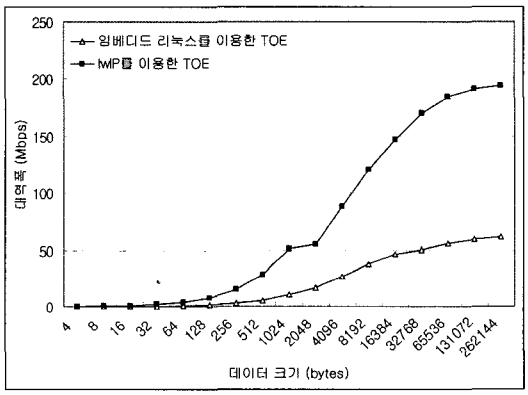

그림 9 임베디드 리눅스를 이용한 TOE와 lwIP를 이용한 TOE의 대역폭

그림 9에서 보는 바와 같이, 임베디드 리눅스를 이용한 TOE의 경우 대역폭이 62Mbps로 낮은 성능을 보였다. 이러한 낮은 성능의 원인은 임베디드 응용 프로그램과 임베디드 커널간의 문맥 전환, 프로세스의 대기 및 깨우는 작업, 그리고 운영체제의 자체 부하가 존재하기 때문이다[3]. 반면에 lwIP를 이용한 TOE의 경우 위와 같은 부하가 존재하지 않기 때문에 실험 결과에서처럼, 임베디드 리눅스를 이용한 TOE 대역폭의 3배인 194Mbps의 대역폭을 보이고 있다.

본 논문에서 구현한 lwIP를 이용한 TOE는 약 600Mbps의 대역폭을 가지는 일반 기가비트 이더넷보다는 낮은 성능을 보인다. 그러나 194Mbps의 대역폭을 가지고 있고 CPU 사용률 측면에서 월등히 우수한 성능을 보이고 있다. 또한 현재 인터넷 망의 대역폭이 100Mbps를 넘지 않는 환경에서 lwIP를 이용한 TOE는 네트워크의 대역폭을 충분히 활용하면서 일반 기가비트 이더넷 어댑터를 사용할 때보다 호스트 CPU의 부담을 줄일 수 있다는 장점을 가진다.

4.3 lwIP의 개선을 통한 TOE의 성능

본 절에서는 3.5절에서 제시한 lwIP의 개선 방법들을 각각 적용하였을 때의 성능을 측정하였다. 우선 기존 lwIP의 지연시간과 대역폭을 측정하였다. 그 후, 대용량 데이타 전송을 위한 lwIP의 인자 수정, 수신 시의 메모리 복사 제거, 지연 ACK 및 TSO 방식을 차례로 적용하여 각각의 지연시간과 대역폭을 측정하였다.

표 2는 각각의 개선 방법을 적용하였을 때 lwIP를 이용한 TOE의 최소 지연시간을 보여준다. 수정을 하지 않은 lwIP를 이용한 경우 지연시간은 183.19μs를 보였다. 대용량 데이타 전송을 위해서 인자를 수정하였을 경우에는 지연시간에 있어서 변화를 보이지 않았다. 최소 지연시간은 4바이트의 데이타를 전송할 때 측정한 값이므로, 대용량 데이타를 위한 인자 수정은 최소 지연시간에 영향을 미치지 않았다. 메모리 복사 제거를 위한 수정을 하였을 경우 최소 지연시간은 174.88μs을 보였고, 마지막으로 지연 ACK 방식을 적용하면 126.15μs의 최소 지연시간을 보였다.

표 2 lwIP의 개선을 통한 TOE의 최소 지연시간

lwIP의 개선 방법	지연시간(μs)
수정을 하지 않은 lwIP	183.19
lwIP + 인자 수정	183.13
lwIP + 인자 수정 + 메모리 복사 제거	174.88
lwIP + 인자 수정 + 메모리 복사 제거 + 지연 ACK	126.15

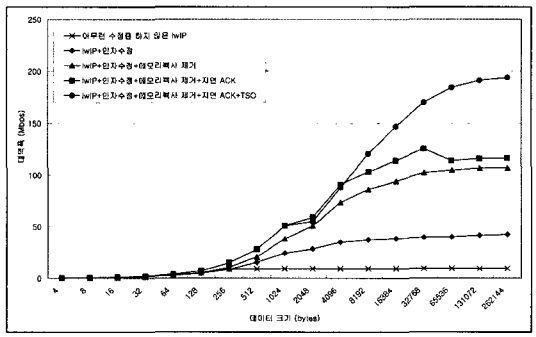

그림 10 lwIP의 개선을 통한 TOE의 대역폭

그림 10은 각각의 개선 방법을 적용하였을 때, lwIP를 이용한 TOE의 대역폭을 보여준다. 수정을 하지 않은 lwIP를 이용하였을 경우 대역폭은 9.6Mbps이었다. 대용량 데이타를 위한 인자 수정을 하였을 경우 42.6Mbps의 대역폭을 보였으며, 메모리 복사 제거를 위한 수정을 하였을 경우 대역폭은 107.2Mbps이었다. 지연 ACK 방식을 적용하면 대역폭은 125.4Mbps로 측정되었다. 마지막으로 TSO 기능을 lwIP에 적용하였을 때, 대역폭은 194Mbps였다. 각각의 개선 방법을 적용함에 따라 증가된 대역폭의 수치를 통해서 메모리 복사를 제

거한 경우와 TSO 기능을 적용하였을 때가 다른 개선 방법과 비교하여 많은 성능 향상이 있었음을 알 수 있다.

5. 결론 및 향후 과제

본 논문에서는 두 가지의 소프트웨어 기반 TOE를 구현하였다. 임베디드 리눅스를 이용한 TOE는 운영체제를 이용함으로써 가지는 문맥 전환, 프로세스의 대기 및 깨우기와 같은 작업으로 인해 높은 성능을 낼 수 없었다. 이러한 문제를 해결하기 위하여 본 논문에서는 운영체제 없이 lwIP를 이용하여 TOE를 구현하였다. 그리고 lwIP를 이용한 TOE의 성능향상을 위하여 lwIP의 문제점을 분석하여 이를 개선하였다. 실험을 통하여 본 논문에서 구현한 두 TOE의 성능을 측정하였다. 일반 기가비트 이더넷 어댑터를 사용한 경우 호스트의 CPU 사용률이 30~70% 보였으나, 구현한 두 TOE의 CPU 사용률은 6%를 넘지 않았다. 임베디드 리눅스를 이용한 TOE의 경우 대역폭은 약 62Mbps이었고, 본 연구에서 개선한 lwIP를 이용한 TOE의 경우 그의 3배인 194Mbps의 대역폭을 보였다. 또한 lwIP의 각 개선 방법에 따른 성능 변화를 실험을 통해 밝혔다. 이러한 lwIP를 이용한 TOE는 낮은 CPU 사용률과 약 200Mbps의 대역폭으로 현재의 인터넷 환경에서 호스트의 컴퓨팅 파워를 보장해줄 수 있다. 향후 본 연구진은 구현한 lwIP를 이용한 TOE의 다중 연결 및 혼잡 상황에서의 성능을 실험을 통하여 측정할 것이며, 필요 시 lwIP의 혼잡 제어 알고리즘을 개선할 것이다. 또한 현재 lwIP를 이용한 TOE가 가지는 194Mbps의 성능을 개선하기 위해, 독자적인 TOE용 TCP/IP 스택을 개발하여 450Mbps 이상의 대역폭을 낼 수 있는 TOE를 개발하는 것을 목표로 하고 있다.

참 고 문 헌

[1] N. Bierbaum, "MPI and Embedded TCP/IP Gigabit Ethernet Cluster Computing," Proc. of the 27th Annual IEEE Conference on Local Computer Networks, pp. 733 - 734, 2002.

[2] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, D. K. Panda, "Performance Characterization of a 10-Gigabit Ethernet TOE," Proc. of the 13th IEEE Symposium on High-Performance Interconnects, August 2005.

[3] In-Su Yoon and Sang-Hwa Chung, "Implementation and Analysis of TCP/IP Offload Engine and RDMA Transfer Mechanisms on an Embedded System," Lecture Note in Computer Science, Vol. 3740, pp. 818-830, October 2005.

[4] SLIC Technology Overview [online]. Available: http://www.alacritech.com/html/tech_review.shtml

[5] A Lightweight TCP/IP stack [online]. Available: http://savannah.nongnu.org/projects/lwip

[6] A. Dunkels, "Full TCP/IP for 8-Bit Architectures," Proc. of the First International Conference on Mobile Applications, Systems and Services, May 2003.

[7] PCI-730: Intelligent Gigabit Ethernet Controller [online]. Available: http://www.cyclone.com/products/pci730.php

윤 인 수

2001년 부산대학교 컴퓨터공학과 학사
2001년~현재 부산대학교 컴퓨터공학과 석박사 통합과정. 관심분야는 컴퓨터 구조, 클러스터 시스템, RDMA, TOE

정 상 화

정보과학회논문지 : 시스템 및 이론
제 33 권 제 5 호 참조

최 봉 식

2004년 부산대학교 컴퓨터공학과 학사
2006년 부산대학교 컴퓨터공학과 석사
2006년~현재 삼성전자 무선사업부 연구원. 관심분야는 컴퓨터 구조, 클러스터 시스템, TOE

전 용 태

2005년 울산대학교 컴퓨터공학과 학사
2005년~현재 부산대학교 컴퓨터공학과 석사 과정. 관심분야는 컴퓨터 구조, 클러스터 시스템, RDMA, TOE