임베디드 시스템에 기반한 TCP/IP Offload Engine 구현 및 분석[1)]

윤인수° 정상화
부산대학교 컴퓨터공학과
{isyoon°, shchung}@pusan.ac.kr

Implementation and Analysis of a TCP/IP Offload Engine on an Embedded System

In-Su Yoon° and Sang-Hwa Chung
Department of Computer Engineering, Pusan National University

요 약

현재 네트워크 기술은 기가비트급의 속도를 넘어 급속히 발전하고 있다. 이러한 고속 네트워크상에서 TCP/IP를 사용할 경우, 호스트 CPU에서 TCP/IP 프로토콜을 처리하는데 많은 부하가 발생한다. 이러한 문제를 해결하기 위해 최근 네트워크 어댑터에서 TCP/IP를 처리하는 TCP/IP Offload Engine(TOE)에 대한 연구가 활발히 진행되고 있다. 본 논문에서는 임베디드 시스템과 리눅스를 사용하여 TOE를 구현하였으며, 그 동작 메커니즘을 보인다. 실험 결과 및 분석을 통해 임베디드 시스템에 리눅스를 활용한 TOE는 상당한 오버헤드를 가지고 있는 것으로 나타났으며, 이를 바탕으로 추후 기가비트 환경에 맞는 TOE 구현 시 이러한 오버헤드를 극복할 수 있는 방안을 제안한다.

1. 서 론

현재 통신 대역폭은 수 Gbps급으로 빠르게 증가하고 있다. 이러한 상황에서 기존의 호스트 컴퓨터에서 TCP/IP와 같은 통신 프로토콜을 처리하는 방식은, 호스트 CPU에 부하를 증가시키고, 이로 인해 전체 시스템의 성능을 저하시킨다. 이러한 문제점들을 해결하는 방안으로서 네트워크 어댑터에서 TCP/IP 프로토콜을 처리하는 TCP/IP Offload Engine(TOE) 기술에 대한 연구가 근래에 활발히 진행되고 있다.

본 논문에서는 이러한 TOE를 임베디드 시스템과 리눅스를 사용하여 구현하였다. 실험 결과 임베디드 시스템에 리눅스를 활용한 TOE는 상당한 오버헤드를 가지고 있는 것으로 나타났다. 본 연구진은 실험결과를 분석하였고, 이를 바탕으로 추후 TOE를 구현할 시 이러한 오버헤드를 극복할 수 있는 방안을 제안한다.

2. 관련 연구

지금까지 TCP/IP의 오버헤드를 분석[1]하고 문제점을 해결하기 위한 연구[2]가 다양하게 이루어져 왔으며, 최근에는 TOE 기술에 대한 연구가 활발히 진행되고 있다.

지금까지 TOE의 구현 방향은 크게 네트워크 어댑터에 범용 내장형 프로세서를 탑재하여 TCP/IP를 처리하는 소프트웨어적인 구현 방안과 ASIC과 같은 별도의 하드웨어로 TCP/IP를 처리하는 하드웨어적인 구현 방안으로 나뉘어져 왔다. 소프트웨어 방식으로 구현한 대표적인 사례로는 인텔사의 PRO1000T IP Storage Adapter[3]를 들 수 있다. 이 제품은 기가비트 이더넷 어댑터에 인텔사의 80200 StrongARM (200MHz) 내장형 프로세서를 장착하여 TCP/IP와 iSCSI 프로토콜을 처리한다. 그리고 하드웨어 방식으로 구현한 대표적인 사례로는 Alacritech 사의 SLIC(session-layer interface control) 기술[4]과 이를 채택한 네트워크 어댑터들을 들 수 있다.

3. TOE 구현

TOE 구현을 위해서는 우선 호스트쪽 커널을 수정하여 호스트의 커널이 TCP/IP 프로토콜 스택을 처리하지 않게 해야 하며, TOE 기능을 담당하는 카드의 구현이 또한 필요하다. 3.1절과 3.2절에서 그 구현을 각각 보인다.

3.1 호스트측 구현 사항

이 절에서는 TCP/IP로 작성된 기존의 응용 프로그램들이 수정 없이 어떻게 TOE를 사용할 수 있는가에 대해 설명한다. 이를 위해 본 연구진은 기존 2.4.27 버전의 리눅스 커널을 수정하였으며, 그림 1은 TCP/IP를 사용할 때 거치는 프로토콜 계층의 수정 전, 후를 비교한다.

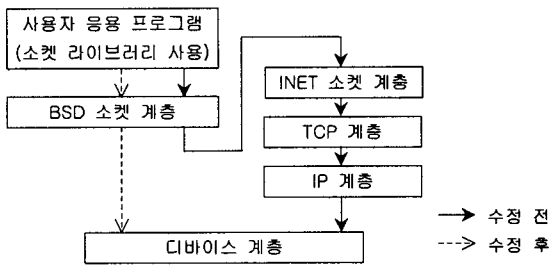

그림 1 커널 수정 전, 후의 TCP/IP 처리 과정 비교도

사용자는 기존의 코드를 수정할 필요 없이, 소켓을 생성하는

1) 본 연구는 정부통신연구진흥원지원 기초기술연구지원사업(과제번호:04-기초-040)으로 수행한 연구결과입니다.

부분의 코드만 TOE를 이용한다고 정의해 주면 된다. 이러한 코드들은 수행 중 BSD 소켓 계층에서 하위 INET, TCP, IP계층들을 거치지 않고 바로 디바이스 계층을 호출하며, 호출된 디바이스가 TCP/IP를 처리하여 그 결과를 되돌려준다.

3.2 카드측 구현 사항

본 연구진은 구현을 위해 Cyclone사의 PCI-730 기가비트 이더넷 카드를 이용하였다. 그림 2는 PCI-730의 블록 다이어그램을 나타낸다.

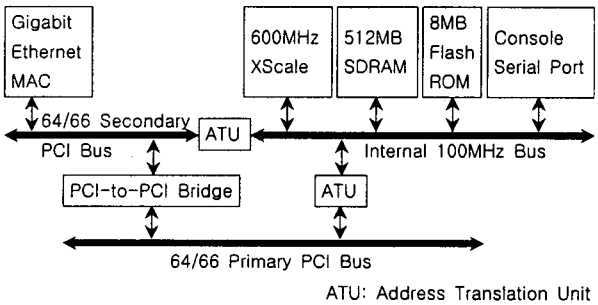

그림 2 PCI-730의 블록 다이어그램

본 연구진은 PCI-730용 패치가 적용된 2.4.18 버전의 리눅스 커널을 임베디드 운영체제로 하였으며, TOE 메커니즘의 구현을 위해 XScale용 공개 컴파일러를 이용하여 커널 모듈 및 임베디드 응용 프로그램을 작성하였다.

3.3 TOE의 동작 메커니즘

임베디드 리눅스 상에서 동작하는 응용 프로그램이 TCP/IP의 처리를 담당하며, 이러한 응용 프로그램과 호스트와의 연결을 임베디드 커널 모듈이 담당한다. 그림 3은 TCP/IP를 이용할 때의 전형적인 서버, 클라이언트의 동작 과정을 나타내며, 그림 4와 5는 이를 본 연구진이 구현한 TOE에서는 어떻게 처리하는지를 나타낸다.

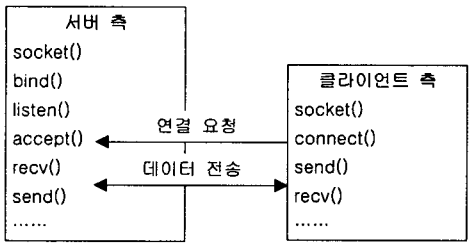

그림 3 전형적인 서버, 클라이언트 동작 모델

그림 3에서 보이듯이 소켓 라이브러리를 이루는 함수들은 연결에 필요한 함수들과 데이터 전송에 필요한 send(), recv() 함수들로 구분할 수 있다. 그림 4는 연결에 필요한 함수들이 본 연구진이 구현한 임베디드 시스템에서 어떻게 처리되는지를 나타낸다.

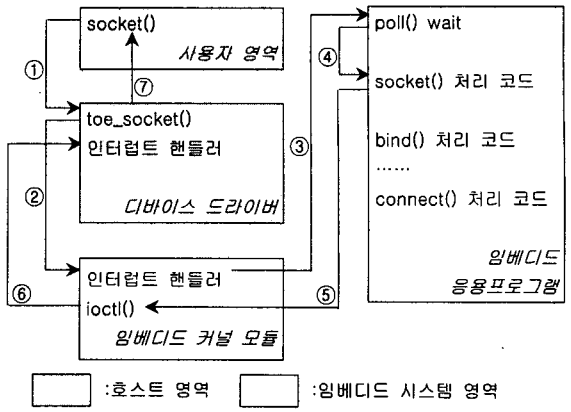

그림 4 TCP/IP 연결과 관련된 함수들의 처리도

그림 4는 소켓 생성 시에 쓰이는 socket() 함수가 어떤 식으로 처리되는지를 나타낸다. bind(), listen()등과 같은 함수들도 같은 방식으로 처리된다. 호스트의 응용 프로그램이 socket() 함수를 호출하면 BSD 소켓 계층에서 디바이스 드라이버에 구현된 toe_socket() 함수를 바로 호출한다. toe_socket()함수는 소켓 생성 시에 필요한 파라미터들을 임베디드 커널 모듈로 전달하고 인터럽트를 건 후 대기한다. 인터럽트를 받은 커널 모듈은 대기 중인 임베디드 응용 프로그램을 깨우고, 깨워 진 응용 프로그램은 처리해야 할 소켓 라이브러리 함수가 무엇인지에 따라 해당 코드를 수행한 후, 결과 값을 ioctl()을 통해 모듈로 전달한다. 모듈이 호스트의 디바이스 드라이버에 결과 값을 쓴 후 인터럽트를 걸면, 대기 중인 toe_socket() 함수는 깨어나고, 처리 결과를 호스트 응용 프로그램으로 보내고 종료한다. 데이터 전송에 쓰이는 send(), recv()와 같은 함수는 위와 같은 과정 외에 전송하고자 하는 사용자 영역의 메모리 정보를 커널 모듈로 전달하는 과정과 DMA하는 부분이 추가된다. 이를 그림 5에 나타낸다.

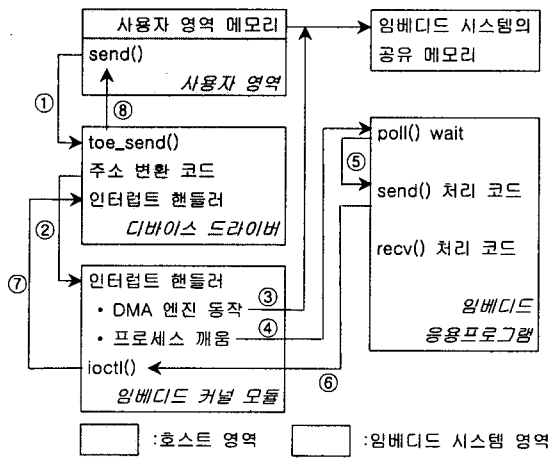

그림 5 데이터 전송과 관련된 함수들의 처리도

그림 5는 send() 함수가 어떻게 처리되는지를 나타낸다. send() 함수 호출 시, 디바이스 드라이버는 사용자 영역의 메모리를 복사하지 않고 전송하고자 하는 메모리 영역의 물리 주소와 길이정보만을 구해 커널 모듈로 알려주고 인터럽트를 건다. 커널 모듈은 이 정보를 DMA엔진에 실어 호스트 메모리의 데이터를 임베디드 응용 프로그램과 공유하고 있는 메모리로 가져오게 된다. 이 후 커널 모듈은 대기 중인 응용 프로그램을 깨우고, 응용 프로그램은 공유 메모리에 있는 내용을 send() 함수를 통해 원격노드로 전송하게 된다. 전송 후 호스트로 결과 값의 전달 메커니즘은 그림 4와 동일하다. 임베디드 커널 모듈과 응용 프로그램 사이에 공유 메모리를 둠으로써 사용자 영역->디바이스 드라이버->임베디드 커널 모듈->임베디드 응용 프로그램으로 발생하는 3번의 복사를 한 번의 DMA를 통해 해결할 수 있었다. recv() 함수 역시 그림 5와 유사한 차례를 따라 처리된다.

4. 실험 및 분석

본 장에서는 임베디드 시스템을 활용하여 TOE 전송 메커니즘을 구현할 시, 각 구간에서 걸리는 시간을 측정하여, 어느 부분이 얼마만큼의 오버헤드를 지니는지 밝힌다. 그리고 이를 분석하여 TOE 구현 시, 적절한 모델을 제시한다.

실험에는 PCI-730 카드를 장착한 1.8GHz 펜티엄 IV 서버 2대가 사용되었으며, 3COM의 SuperStack3 스위치로 연결하였다. 지연시간 측정은 send() 함수의 호출로부터 데이터가 원격노드의 수신버퍼에 들어갈 때까지의 각 구간 시간을 측정하였다. 측정치는 4바이트의 데이터 전송을 100회 반복한 평균치이다.

표 1 각 구간 별 지연 시간 분석 (단위 ㎲)

항목	지연 시간	백분율
호스트에서의 주소 변환	4	1%
인터럽트 처리 (호스트↔카드)	10	2%
DMA로 메모리 정보와 데이터 전송	37	7%
인터럽트 핸들러의 나머지 코드 수행	31	6%
임베디드 응용 프로그램 깨움	31	6%
TCP/IP 프로토콜 처리 (송, 수신)	249	47%
이더넷 드라이버의 패킷 처리 (송, 수신)	84	16%
send() 완료 후 ioctl()함수 호출(송신 측)	27	5%
recv() 완료 후 ioctl()함수 호출(수신 측)	27	5%
DMA로 수신된 데이터를 호스트로 전송	33	6%
합계	533	100%

표 1에서 보이듯이 533㎲라는 지연시간으로는 기가비트 환경에 TOE를 적용하기 힘들다. 이러한 긴 지연시간을 줄이기 위해 임베디드 응용 프로그램이 하는 일들을 임베디드 커널 모듈로 옮기고 임베디드 응용 프로그램을 없애는 방안을 고려해 보았다. 이 경우 표 1의 진하게 표시한 시간들은 발생하지 않는다. 또한 관련 연구[5]에 의하면 응용 프로그램 영역에서 커널 영역으로 데이터를 복사하는 시간이 TCP/IP 프로토콜을 처리하는 시간에서 약 17%(42㎲)를 차지한다. 이러한 시간들을 533㎲에서 제하게 되면 375㎲의 지연시간을 가지며 이는 본 연구에서 구현한 것보다 약 30%의 시간을 줄여줄 수 있다.

또한 TOE를 구현할 시, 리눅스와 같은 범용 OS를 사용하지 않는 방법이 있다. 리눅스는 TCP/IP 계층과 디바이스 계층의 분리로 인한 두 계층 간 데이터 복사 오버헤드를 가지고 있다. 관련 연구[5]는 이러한 오버헤드가 약 32%(80㎲)를 차지하며 TCP/IP를 처리하는 도중에 발생하는 OS측 오버헤드가 약 20%(67㎲)를 차지한다고 밝히고 있다. 본 연구에서 사용한 PCI-730 카드에 리눅스를 사용하지 않고 TOE를 구현하기 위해서는 Lightweight TCP/IP stack (lwIP)[6]과 같은, 커널과 분리된 독자적인 TCP/IP 코드와 카드를 커널 없이 동작시키기 위한 프로그램이 필요하다. 이렇게 구현할 시, 약 228㎲의 지연시간을 가질 것으로 예상되며, CPU 파워와 복사 오버헤드에 의존적인 TCP/IP를 본 논문에서 쓰인 PCI-730 카드보다 성능이 우수한 800MHz XScale과 333MHz로 동작하는 내부 버스를 가지는 최신 칩[7]을 사용하여 처리하게 한다면 기가비트 환경에 맞는 200㎲ 초반의 지연시간을 가지는 TOE가 구현될 것으로 예상한다.

5. 결론 및 향후 과제

본 논문에서는 PCI-730 기가비트 이더넷 카드와 임베디드 리눅스를 이용하여 TOE를 구현하였으며, 그 동작 메커니즘을 제시하였다. 실험 결과 533㎲라는 높은 지연시간이 나왔으며, 이를 각 구간별로 나누어 시간을 측정, 분석하였다. 이를 바탕으로 임베디드 커널 모듈에서 TOE를 구현하는 방식과 임베디드 리눅스 없이 TOE를 구현하는 방식이 본 논문에서 구현한 시스템과 비교하여 어느 정도 지연시간을 줄일 수 있는가를 밝히고 있다. 향후 과제로는 본 연구에서 사용한 PCI-730카드보다 상위버전의 하드웨어 상에 임베디드 리눅스와 같은 범용 OS를 사용하지 않는 TOE를 개발할 예정이다.

6. 참고 문헌

[1] J. Kay, J. Pasquale, "Profiling and reducing processing overheads in TCP/IP", IEEE/ACM Transactions on Networking, Vol. 4, No. 6, pp. 817-828, Dec. 1996.

[2] H. Bilic, Y. Birk, I. Chirashnya, and Z. Machulsky, "Deferred segmentation for wire-speed transmission of large TCP frames over standard GbE networks", Hot Interconnects 9, pp. 81-85, Aug. 2001.

[3] http://support.intel.com/support/network/adapter/iscsi/tip/index.htm

[4] http://www.alacritech.com/html/tech_review.html

[5] David D. Clark, Van Jacobson, John Romkey, and Howard Salwen, "An Analysis of TCP Processing Overhead", IEEE Communications, Volume 27, No. 6, pp. 23-29, June 1989.

[6] Adam Dunkels, "Full TCP/IP for 8-Bit Architectures". In Proceedings of the first international conference on mobile applications, systems and services, San Francisco, California, May 2003.

[7] http://www.intel.com/design/iio/iop333.htm