임베디드 시스템에 기반한 TCP/IP Offload Engine과 RDMA 메커니즘의 구현

윤인수°, 정상화

{isyoon°, shchung}@pusan.ac.kr

Implementation of a TCP/IP Offload Engine and RDMA Mechanism Based on an Embedded Systems

In-Su Yoon°, Sang-Hwa Chung

Department of Computer Engineering, Pusan National University

요 약

기가비트의 속도를 넘는 고속 네트워크상에서 TCP/IP를 사용할 경우, 호스트 CPU에서 TCP/IP를 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 연구에서는 임베디드 시스템에 리눅스를 사용하여, TOE와 Remote Direct Memory Access(RDMA) 메커니즘을 구현하였고 그 동작 방식을 보인다. 실험을 통해 구현된 메커니즘들을 검증하였으며, 각 구간별로 소요시간을 측정하였다. 본 논문에서는 이러한 실험 결과를 바탕으로, 추후 기가비트 환경에 적합한 TOE 및 RDMA 메커니즘의 구현 방안을 제안한다.

1. 서 론

호스트에서 TCP/IP를 처리하는 방식은, 통신 대역폭이 빠르게 증가할수록 시스템 자원을 많이 소비하며, 이로 인해 전체 시스템의 성능을 저하시킨다. 이러한 문제점을 해결하기 위해, 네트워크 어댑터에서 TCP/IP를 처리하는 TOE에 대한 연구가 활발히 진행되고 있다. 그러나 TOE도 TCP/IP 스택을 이동하면서 발생하는 데이터 복사 문제를 여전히 지니고 있으며, 고속 TCP/IP 통신에서 이러한 오버헤드는 상당히 크다. 이의 해결을 위해 무 복사로 데이터를 전송할 수 있는 RDMA 메커니즘이 RDMA 컨소시엄[1]에 의해 제안되었다. 이는 원격 노드의 사용자 메모리로 패킷의 데이터를 바로 DMA할 수 있게 하며, 이러한 RDMA와 TOE가 결합한 네트워크 인터페이스 카드(NIC)를 RDMA enabled NIC(RNIC)이라고 부른다.

본 연구에서는 임베디드 시스템에 리눅스를 사용하여 TOE와 RDMA를 구현하였으며, 기존 소켓 응용 프로그램들이 수정 없이 TOE와 RDMA를 이용할 수 있도록 하였다. 실험 결과, 임베디드 시스템에 리눅스를 사용하여 상기 메커니즘들을 구현한 방식은 높은 지연시간을 가졌다. 본 논문에서는 구현된 메커니즘들의 각 구간별로 소요시간을 측정하여 그 원인을 분석하였으며, 이를 바탕으로 추후 기가비트 환경에 적합한 TOE 및 RNIC의 구현 방안을 제안한다.

2. 관련 연구

TOE의 구현 방향은 크게 네트워크 어댑터에 범용 프로세서를 탑재하여 TCP/IP를 처리하는 소프트웨어적 방법과 ASIC과 같은 별도의 하드웨어로 TCP/IP를 처리하는 하드웨어적 방법으로 나뉜다. 소프트웨어 방식으로 구현한 사례로는 인텔사의 PRO1000T IP Storage Adapter를 들 수 있다. 이것은 임베디드 리눅스와 같은 OS를 사용하여 TCP/IP를 처리하는 것이 아니라, OS없이 동작할 수 있는 독자적인 프로그램을 개발하여 TCP/IP를 처리한다. 하드웨어 방식으로 구현한 대표적 사례로는 Alacritech사의 Session Layer Interface Control(SLIC) 기술과 Adaptec사의 ASIC에 기반한 TOE NAC 7711을 들 수 있다. 또한 InfiniBand나 VIA에 RDMA를 구현한 것[2][3]은 있으나, TOE와 RDMA가 결합한 형태인 RNIC은 학계에 보고된 바 없다. 이러한 TOE 제품들은 전용 프로그램 및 ASIC을 사용하여 TCP/IP를 처리하고 있어 그 통신 성능은 우수하다. 그러나 TOE상에 새로운 응용 계층을 추가하기 힘들며, IPv6와 같은 새로운 TCP/IP 구조를 적용시키기 힘들다는, 구조의 유연성 측면에서의 단점을 지닌다. 따라서 본 논문에서는 현재 여러 임베디드 시스템에 적용되고 있는 리눅스를 TOE 및 RDMA를 구현하는데 사용하였고, 실험을 통해 그 성능을 측정, 분석하였다.

3. TOE와 RDMA 메커니즘의 구현

소켓 라이브러리를 이용하여 작성된 기존 TCP/IP 응용 프로그램들이 수정 없이 TOE와 RDMA를 이용할 수 있도록 구현하였으며, 이를 위해 호스트의 커널을 수정하였다. 또한 TOE 및 RDMA 메커니즘을 임베디드 커널 모듈과 응용 프로그램으로 카드에 구현하였다.

3.1 호스트 측 구현 사항

TOE를 사용하고자 하는 기존 소켓 라이브러리 함수들은, 호스트의 TCP/IP 스택을 거치지 않고, 바로 BSD 소켓 계층에서 디바이스 계층을 호출한다. 호출된 디바이스(TOE)는 소켓 함수를 구별하여 관련 TCP/IP 작업을 수행한 후, 그 결과를 반환하게 된다. 이를 위해 호스트의 2.4.27 버전의 리눅스 커널을 수정하였으며, 그림 1은 수정 전, 후의 TCP/IP 처리 순서를 나타낸다.

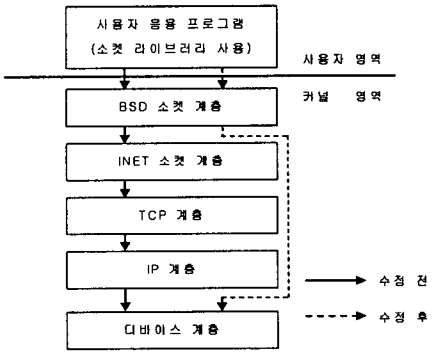

그림 1 커널 수정 전, 후의 TCP/IP 처리 순서

수정한 부분은 커널 소스 트리의 *include/linux* 디렉터리 안의 *netdevice.h* 파일과 *socket.h* 파일, *net/* 디렉터리 안의

이 논문은 교육인적자원부 지방연구중심대학육성사업(차세대 물류IT기술연구사업단)의 지원에 의하여 연구되었음

socket.c 파일을 수정하였다.

3.2 카드 측 구현 사항

600MHz의 XScale과 기가비트 인터페이스를 가지고 있는 Cyclone사의 PCI-730 카드[4]를 이용하였다. PCI-730용 패치가 적용된 2.4.18 버전의 리눅스 커널을 임베디드 운영체제로 하였으며, XScale용 공개 컴파일러를 이용하여 임베디드 커널 모듈 및 임베디드 응용 프로그램을 작성하였다.

3.2.1 TOE 동작 메커니즘

임베디드 응용 프로그램이 TCP/IP 처리를 담당하며, 이러한 응용 프로그램과 호스트와의 연결을 임베디드 커널 모듈이 담당하도록 구현하였다. TOE를 이용한 send() 소켓 함수 처리 메커니즘을 그림 2에 나타낸다.

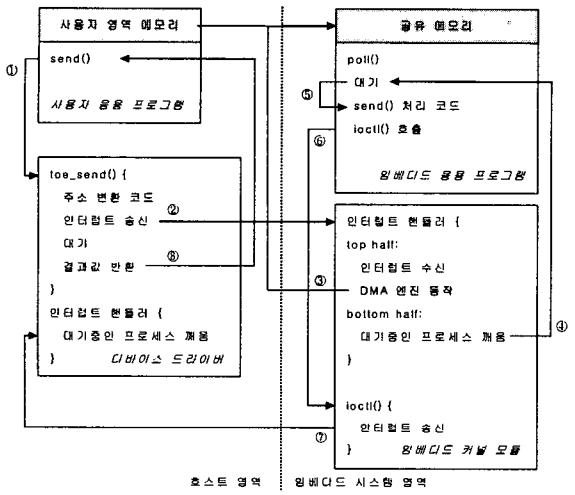

그림 2 TOE를 이용한 send() 소켓 함수 처리도

send() 함수 호출(①) 시, 사용자 메모리 영역의 물리 주소와 길이정보만을 구해 커널 모듈로 알려주고 인터럽트를 건다(②). 커널 모듈은 이 정보를 이용해, 임베디드 응용 프로그램과 공유하고 있는 메모리로 데이터를 DMA해 온다(③). 이 후 커널 모듈은 대기 중인 응용 프로그램을 깨우고(④), 응용 프로그램은 공유 메모리에 있는 내용을 send() 함수를 통해 원격 노드로 전송(⑤)한다. 그 결과는 ioctl()을 통해 커널 모듈로 전달(⑥)되고, 커널 모듈은 이를 인터럽트를 통해 호스트에 알린다(⑦). 마지막으로 호스트의 디바이스 드라이버는 사용자 응용 프로그램을 깨워 결과값을 반환(⑧)하고 작업을 완료하게 된다. recv() 함수는 그림 2와 유사한 순서에 의해 처리되며, socket(), bind(), listen() 등과 같은 함수들은 상기 처리 순서에서 데이터를 DMA해오는 부분을 제외하고 같은 순서에 의해 처리된다.

3.2.2 RDMA 동작 메커니즘

본 연구에서는 send()/recv() 소켓 함수를 RDMA write를 통해 처리하였고, 그림 3은 이러한 RDMA write의 메커니즘을 보인다. RDMA 패킷을 만드는 부분과 RDMA 패킷을 수신하였을 때 호출될 수 있는 RDMA 패킷 핸들러를 커널 모듈 내에 구현하였다. 이렇게 독자적인 패킷처리 메커니즘을 커널 모듈 내에 구현함으로써 TOE의 임베디드 커널 모듈과 임베디드 응용 프로그램으로 나누어 send(), recv() 함수를 처리하는 방식에 비해 문맥전환(context switching) 오버헤드를 없앨 수 있었다.

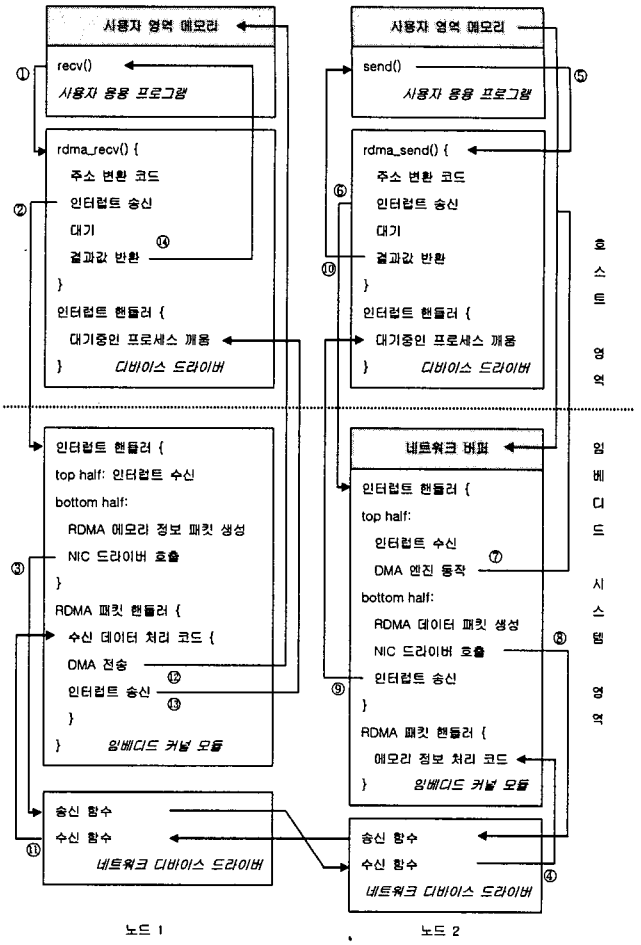

그림 3 RDMA write를 이용한 기존 send()/recv()를 처리도

동작과정을 설명하면, 우선 노드 1에서 recv() 함수를 호출(①)한 후, 디바이스 드라이버에서 사용자 영역의 메모리에 대한 물리 주소와 길이정보를 구해 커널 모듈로 알려주는 방식은 그림 2에서 나타낸 것과 동일하다(②). 이 후 커널 모듈의 인터럽트 핸들러는 이러한 정보를 바탕으로 소켓 버퍼를 생성하고 자신과 원격노드의 MAC 주소를 가지고 패킷 헤더를 만든다. 패킷의 데이터 부분은 디바이스 드라이버가 내려 준 물리 주소와 길이 정보들을 기입하여 만든다. 이를 본 논문에서는 *RDMA 메모리 정보 패킷*이라고 부른다.

이렇게 만들어진 패킷을 하위 네트워크 디바이스 드라이버로 전달(③)하면, 네트워크 카드는 RDMA 메모리 정보 패킷을 노드 2로 전송하게 된다. RDMA 메모리 정보 패킷을 받은 노드 2는 이를 커널 모듈에서 저장, 관리한다(④). 노드 2의 호스트에서 send() 함수를 호출(⑤)한 후, 패킷 헤더를 만드는 부분까지는 앞 단락과 동일하게 동작한다. 이 후 노드 2는 *RDMA 데이터 패킷*을 만들기 위해 커널 모듈에서 관리하고 있는 노드 1의 메모리 정보와 사용자 영역에서 가져온 데이터를 패킷의 데이터 부분에 추가하고 이를 노드 1으로 전송(⑧)한다. RDMA 데이터 패킷을 받은(⑪) 노드 1은 RDMA 데이터

패킷에 기입된 메모리 정보를 바탕으로 패킷에 들어있는 데이터를 호스트의 사용자 영역에 있는 수신 버퍼영역으로 바로 DMA 전송(⑫)한다. 이러한 RDMA 전송 메커니즘은 패킷에 데이터와 함께 메모리 정보도 같이 보냄으로써 네트워크 혼잡에 의한 패킷 드랍 시 혹은 패킷이 순서대로 오지 않을 시에도 패킷내의 메모리 정보를 보고 사용자의 수신 버퍼로 직접 데이터를 전송할 수 있다는 이점을 지닌다.

4. 실험 및 분석

실험에는 PCI-730 카드를 장착한 1.8GHz 펜티엄 IV 서버 2대가 사용되었으며, 3COM의 SuperStack3 스위치로 연결하였다. 지연시간 측정은 send() 함수의 호출로부터 데이터가 원격 노드의 수신버퍼에 들어갈 때까지의 시간을 측정하였다. 각 구간을 나누어 소요 시간을 측정하였으며, 측정치는 4바이트의 데이터 전송을 100회 반복한 평균치이다. 표 1은 TOE및 RDMA의 각 구간별 소요 시간과 지연시간을 나타낸다. 표 1에서 T는 TOE를 R은 RDMA를 나타내며, 시간의 단위는 ㎲이다.

표 1 TOE및 RDMA의 각 구간별 소요시간 및 지연시간

항 목	T	R	시간
호스트 드라이버에서의 주소 변환	O	O	4
인터럽트 처리 (호스트↔카드)	O	O	10
DMA로 메모리 정보와 데이터 전송	O	O	37
top half에서 bottom half로 진입대기시간	O	O	31
임베디드 응용 프로그램 깨움	O		31
TCP/IP 처리 (송, 수신)	O		249
RDMA 패킷 처리 (송, 수신)		O	68
네트워크드라이버의 패킷처리 (송, 수신)	O	O	84
send() 완료 후, ioctl()함수호출 (송신측)	O		27
recv() 완료 후, ioctl()함수호출 (수신측)	O		27
DMA로 수신된 데이터를 호스트로 전송	O	O	33
TOE의 지연시간			533
RDMA의 지연시간			267

TOE는 임베디드 응용 프로그램을 사용하여 TCP/IP를 처리하고, RDMA는 독자적인 패킷 처리 메커니즘을 가진다. TOE와 RDMA가 호스트와 인터페이스하는 메커니즘은 동일하다. 표 1에서 TOE의 지연시간이 533㎲라는 높은 지연시간을 가지는데 비해, RDMA는 267㎲라는 상대적으로 낮은 지연시간을 가진다. 이는 임베디드 응용 프로그램과 커널 모듈 사이를 오가며 TCP/IP를 처리하는 TOE가 가지는 문맥전환 오버헤드를 RDMA가 없앴기 때문이다. 본 논문에서는 표 1의 실험결과와 관련 연구[5]를 바탕으로 TOE의 지연시간을 줄일 수 있는 방안을 제안하고 그 예측치를 제시한다.

첫째, 임베디드 응용 프로그램에서 처리하는 작업을 커널 모듈에서 하게 한다면 TOE의 문맥전환 오버헤드를 없앨 수 있다. 이 경우 표 1의 진하게 표시된 시간들을 없앨 수 있다. 또한 관련 연구[5]에 의하면 전체 TCP/IP 처리시간에서 약 17%를 차지하는, 응용 프로그램 영역에서 커널 영역으로 데이터를 복사하는 시간을 없앨 수 있다. 이 시간들이 줄어들면 406㎲의 지연시간을 가질 수 있다[1]. 둘째, 리눅스와 같은 OS를 사용하지 않고 TOE를 구현하는 것이다. 이를 위해서 Lightweight TCP/IP stack[6]처럼 커널과 분리된 TCP/IP 코드와 커널 없이 카드를 구동시킬 프로그램 작성이 필요하다. 이 경우, 리눅스의 TCP/IP와 디바이스 계층의 분리로 인한 두 계층 간 데이터 복사 오버헤드를 없앨 수 있으며, 리눅스와 같은 범용 OS가 시스템 운용을 위해 사용하고 있는 복잡한 알고리즘 및 자료구조가 간단해질 수 있다. 관련 연구[5]는 이러한 복사와 OS 오버헤드를 각각 32%, 20%를 차지한다고 밝히고 있다. 이 경우 276㎲까지 지연시간을 줄일 수 있다[2]. 마지막으로, 본 연구에 쓰인 하드웨어는 600MHz CPU와 100MHz로 동작하는 내부 버스를 가지고 있어, 매 캐시미스 발생 시, 6클럭을 쉬어야 한다. 이는 메모리 참조가 많은 TCP/IP를 처리할 때는 성능에 주요한 영향을 미친다. 따라서 성능이 우수한 800MHz XScale과 333MHz로 동작하는 내부 버스를 가지는 최신 칩[7]을 사용하게 된다면, 좀 더 지연시간을 줄일 수 있을 것으로 예상한다.

따라서 본 논문에서는 위 단락에서 제안한 방법을 사용하여 TOE의 지연시간을 줄이고, TOE의 수신 측 데이터 복사문제를 본 연구에서 구현한 RDMA로 해결한다면 200㎲ 초반의 지연시간과 400Mbps 이상의 대역폭을 가지는 소프트웨어 방식의 TOE 및 RNIC을 구현할 수 있을 것으로 기대한다.

5. 결론 및 향후 과제

본 논문에서는 PCI-730카드와 임베디드 리눅스를 이용하여 TOE와 RDMA 메커니즘을 구현하였으며, 그 동작 방식을 보였다. 실험 결과 TOE와 RDMA는 각각 533㎲, 267㎲라는 최소 지연시간을 보였으며, 이를 각 구간별로 나누어 시간을 측정, 분석하였다. 이를 바탕으로 임베디드 응용 프로그램을 사용하지 않고 TOE를 구현하는 방식과 임베디드 리눅스와 같은 범용 OS없이 TOE 및 RNIC을 구현하는 방식이, 본 논문에서 구현한 시스템과 비교하여 어느 정도 지연시간을 줄일 수 있는가를 밝히고 있다. 향후 과제로는 본 연구에 사용한 PCI-730 카드보다 상위버전의 하드웨어상에서 OS를 사용하지 않는 TOE 및 RNIC을 개발할 예정이다.

6. 참고 문헌

[1] http://www.rdmaconsortium.org/home
[2] J. Liu, D. K. Panda, and M. Banikazemi, "Evaluating the Impact of RDMA on Storage I/O over InfiniBand", SAN-03 Workshop (in conjunction with HPCA), Feb. 2004.
[3] Hermann Hellwagner and Matthias Ohlenroth, "VI architecture communication features and performance on the Gigabit cluster LAN", Future Generation Computer Systems, Vol. 18, Issue 3, January 2002.
[4] http://www.cyclone.com/products/pci730.htm
[5] David D. Clark, Van Jacobson, John Romkey, and Howard Salwen, "An Analysis of TCP Processing Overhead", IEEE Communications, Vol. 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

[1] 406=533-{(31+27+27)+(249*0.17)}
[2] 276=406-249*(0.32+0.2)