소켓 인터페이스의 바이너리 호환성을 제공하는 TCP/IP Offload Engine 용 Linux 커널 모듈

오수철*, 김성운*
*한국전자통신연구원
e-mail : {ponylife, ksw}@etri.re.kr

Linux Kernel Module for TCP/IP Offload Engine Supporting Binary Compatibility of Socket Interface

Soo-Cheol Oh*, Seong-Woon Kim*
*Electronics and Communications Research Institute

요 약

기존의 컴퓨터 시스템에서는 인터넷의 대표적인 통신 프로토콜인 TCP/IP 가 호스트 CPU 에서 처리되는데, 이는 호스트 CPU 에 많은 부하(load)를 발생시켜 전체 시스템의 성능을 저하시키는 문제를 야기한다. 최근 이러한 문제점을 해결하는 방안으로서 네트워크 어댑터에서 TCP/IP 를 처리하는 TOE(TCP/IP Offload Engine)에 대한 연구가 활발히 진행되고 있다. 이러한 TOE 가 성공적으로 컴퓨터 시스템에 적용되는 위해서는 이를 지원하는 운영체제용 커널 모듈의 개발이 필요하며, 커널 모듈은 기존의 TCP/IP 를 위한 소켓 인터페이스를 바이너리 수준에서 호환성을 제공해야 한다. 따라서, 본 논문에서는 Linux 시스템에서 소켓 인터페이스에 대한 바이너리 수준의 호환성을 제공하는 TOE 용 커널 모듈을 제안하고 개발하였다. 또한, 실험의 통하여 TOE 커널 모듈이 CPU 에 부하를 거의 발생시키지 않음을 확인하였다.

1. 서론

인터넷을 비롯한 많은 분야에서 널리 사용되고 있는 Ethernet 기술은 이미 1 Gbps(Gigabit per second) 대역폭의 Gigabit Ethernet 을 넘어 10 Gigabit Ethernet 기술의 표준화가 이루어졌다. Ethernet 상에서 사용되는 대표적인 통신 프로토콜인 TCP/IP 는 일반적으로 호스트 CPU 에서 처리되는데, 이는 호스트 CPU 에 프로토콜을 처리하는 부하를 유발하여 전체 시스템의 성능을 떨어트린다. 특히 네트워크의 물리적인 성능이 발전함에 따라 TCP/IP 를 사용한 통신에서 프로토콜 처리의 부하가 더욱 커지고 있다 [1].

이러한 문제를 해결하는 방안으로서 TCP/IP 프로토콜의 처리를 호스트 CPU 가 아닌 네트워크 어댑터에서 전담하는 TOE(TCP/IP Offload Engine) 기술에 대한 연구가 진행되고 있다. 네트워크 어댑터 상에서 TCP/IP 를 처리하게 되면 호스트 CPU 에 가해지는 부하가 줄어들고, 호스트 CPU 가 프로토콜 처리 이외의 실질적인 작업에 전념함으로써 전체 시스템의 성능이 향상되는 효과를 얻을 수 있다. 또한 호스트 CPU 에서 실행되는 작업량이 늘어나더라도 통신의 성능을 계속 유지할 수 있게 된다.

TOE 가 성공적으로 컴퓨터 시스템에 적용되기 위해서는 이를 지원하는 TOE 커널 모듈의 개발이 필수적이다. 이러한 TOE 용 커널 모듈의 개발에 있어 가장 중요한 점은 TCP/IP 를 위한 기존의 소켓 인터페이스를 바이너리 수준에서 호환성을 제공해야 하는 것이다. 이러한 바이너리 수준의 호환성을 제공함으로써 TCP/IP 에 기반한 기존의 응용 프로그램을 수정 및 재컴파일하지 않고도 TOE 를 활용할 수 있게 된다. 이러한 요구 사항과 더불어 TOE 커널 모듈 개발에 요구되는 사항을 정리해 보면 다음과 같다.

- 기존 소켓 인터페이스와의 바이너리 호환성을 유지함으로써 기존 응용 프로그램의 수정 및 재컴파일없이 응용 프로그램을 수행할 수 있어야 한다.
- 새로운 프로토콜 family 를 정의하지 않아야 한다.
- 기존 운영체제의 커널 수정을 최소화 해야 하고, 이를 위해서 TOE 커널 모듈을 모듈 형태로 개발한다.
- TOE 의 목적은 호스트 CPU 의 utilization 을 최소화 하는 것이다. 따라서 TOE 커널 모듈을 경량화 해야 한다.

본 논문에서는 TOE 를 위한 운영체제로 Linux 를 선택하였으며, 위와 같은 요구 조건을 만족시키는 TOE 커널 모듈을 개발하고자 한다.

2. 배경연구

TOE 를 포함한 네트워크 가속 기술에 대한 대표적인 연구로 OSF (Offload Socket Framework)[2]를 들 수 있다. OSF 는 TOE 와 Infiniband 기반 네트워크 어댑터를 지원하는 프레임워크이다. 이러한 OSF 는 소켓 인터페이스를 지원하기 위해서 새로운 프로토콜 family 를 추가하였다. 또한 [3]에서도 VIA 통신 프로토콜에 소켓 인터페이스를 지원하기 위해서 새로운 프로토콜 family 를 추가하는 방식을 적용하였다. 기존의 TCP/IP 기반 통신에서는 PF_INET 이라는 프로토콜 family 를 사용하며, 모든 응용 프로그램이 PF_INET 을 사용하도록 작성되어 있다. 그러나, 새로운 프로토콜 family 가 정의된다면 기존의 응용 프로그램들은 새로운 프로토콜 family 를 사용하기 위해서 수정되고 재컴파일되어야 하는 문제가 발생한다. 따라서 이러한 방식은 본 논문의 기본 요구 사항과 부합하지 않는다.

3. TOE 커널 모듈의 위치

(그림 1)은 기존의 Linux 에서 제공되는 TCP/IP 를 위한 프로토콜 스택의 구조를 보여준다. 최상위 계층(layer)에는 BSD 소켓(socket)을 지원하는 BSD 소켓 계층이 자리잡고 있으며, 사용자가 UNIX 표준의 소켓 인터페이스를 통해 TCP/IP 를 비롯한 다양한 통신 프로토콜을 사용할 수 있도록 지원한다. BSD 소켓 계층의 아래에는 INET 소켓 계층이 자리잡고 있으며, 실제 통신에 사용되는 프로토콜(TCP, UDP, raw IP)에 대한 abstraction 을 제공한다. INET 계층의 하부에는 TCP/UDP 계층이 있고, 그 밑에 IP 계층이 위치한다. 커널의 프로토콜 스택 하부에는 네트워크 어댑터의 디바이스 드라이버가 자리잡고 있다.

TOE 는 TCP/IP 를 offload 하는 것임으로 TOE 커널모듈은 (그림 1)에서 TCP/IP 계층의 상위에 위치해야 하며, TOE 커널 모듈은 (그림 1)에서 표시된 ①, ② 그리고 ③ 중의 한 곳에 추가될 수 있다.

① 응용 프로그램과 BSD 계층 사이에 위치

기존의 응용 프로그램은 BSD 계층의 소켓 인터페이스를 사용하여 작성되어 있다. 그러나, 응용 프로그램 바로 아래쪽에 TOE 커널 모듈이 위치할 경우, TOE 용 운영체제 소프트웨어는 TOE 사용을 위한 전용 인터페이스를 응용 프로그램에 제공해야 한다. 결국, 이것은 기존의 응용프로그램이 소켓인터페이스를 사용할 수 없도록 만들며, 응용 프로그램이 TOE 를 사용하기 위해서는 TOE 전용 인터페이스를 사용하여 프로그램을 수정해야 하는 문제가 발생한다. 이것은 서론에서 언급한 TOE 커널 모듈의 요구조건에 부합하지 않은 것이다. 따라서, 응용 프로그램과 BSD 계층 사이에 TOE 커널 모듈을 위치시키는 것은 불가능하다

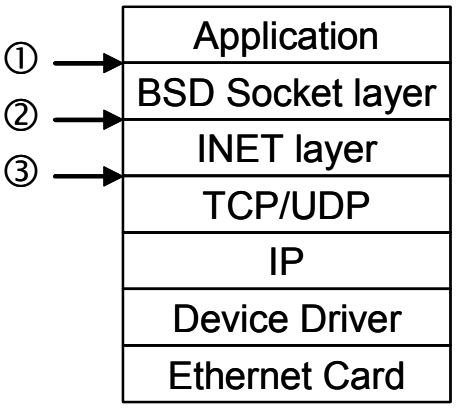
② BSD 계층과 INET 계층 사이에 위치

Linux 에서 소켓을 사용한 모든 통신은 BSD 계층을 사용하여 이루어 진다. 따라서 BSD 계층과 INET 계층 사이에 TOE 커널 모듈이 위치할 경우, 기존의 응용 프로그램이 소켓 인터페이스를 사용할 수 있다. 또한 기존 프로그램의 수정 및 재컴파일없이 바이너리 수준의 호환성을 제공할 수 있다.

③ INET 과 TCP 계층 사이에 위치

두번째 경우와 같이 BSD 계층 하위에 TOE 커널 모듈이 위치함으로 바이너리 호환성 문제를 해결할 수 있다. 소켓을 사용한 통신을 수행하기 전에 먼저 소켓을 생성한다. 이때 Linux 시스템은 크게 2 가지 data 구조체를 생성한다. 첫번째는 TCP/IP 와 관련없이 소켓을 사용한 통신에 필요한 범용적인 데이터를 가지는 socket 구조체 이다. 두번째는 TCP/IP 를 사용한 통신에 필요한 데이터를 가지는 sock 구조체이다. BSD 계층에서는 socket 구조체만을 사용하지만 INET 계층에서는 TCP/IP 와 관련있는 sock 구조체를 사용한다. 그러나, TOE 에서는 TCP/IP 가 TOE 네트워크 어댑터에서 처리되기 때문에 호스트 CPU 에서 TCP/IP 와 관련 있는 sock 구조체를 사용하는 것은 불필요한 작업에 시간을 소요하는 것이 된다. 즉 기능에는 문제가 없지만 불필요한 시간을 소모하게 된다.

이와 같이 위에서 언급한 3 가지 경우를 모두 고려했을 경우, TOE 커널 모듈을 BSD 계층과 INET 계층 사이에 위치시키는 것이 가장 적절하다고 판단된다.

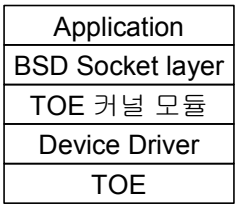


(그림 1) TCP/IP 프로토콜 스택의 구조

4. TOE 커널 모듈의 구현

3 장에서 분석한 결과에 따라서 TOE 커널 모듈을 포함하는 프로토콜 스택의 구조는 (그림 2)와 같다. 응용 프로그램에서 소켓 인터페이스를 사용하여 TCP/IP 에 기반한 통신명령을 내리면 BSD 계층을 통하여 TOE 커널 모듈로 전송된다. 이러한 TOE 커널 모듈은 TOE 용 device driver 를 통하여 TCP/IP 를 처리하는 TOE 로 통신 명령을 전송하고, TOE 를 이를 받아들여 TCP/IP 통신을 수행한다.

3 장의 (그림 1)에서는 BSD 계층에서 INET 계층을 호출하였지만, 본 논문에서는 BSD 계층이 INET 계층이 아닌 TOE 커널 모듈을 호출할 수 있도록 하였다. 또한 TOE 커널 모듈은 Linux 의 모듈 형태로 구현함으로써 기존 커널의 수정을 최소화하려고 하였다. BSD 계층이 INET 계층을 호출하는 과정은 소켓 생성함수와 이 외의 다른 함수들이 다른 메커니즘을 사용함으로 이를 분리하여 4.1 절과 4.2 절에서 설명하겠다.

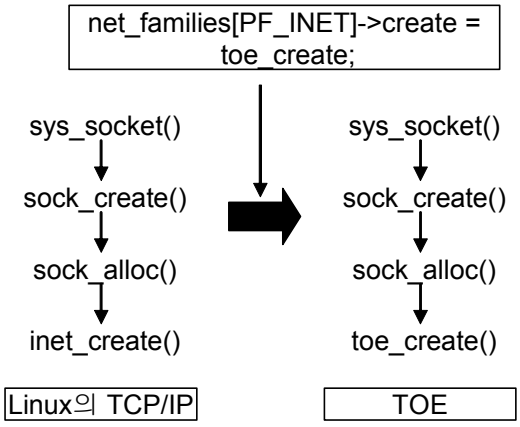


(그림 2) TOE 지원 프로토콜 스택의 구조

4.1 소켓 생성 함수

(그림 3)은 기존의 Linux TCP/IP 에서의 소켓 생성과정과 TOE 커널 모듈을 적용했을 때의 소켓 생성과정을 보여준다. 모든 socket API 호출은 Linux 커널의 sys_socketcall() 함수를 호출하고, 이 함수는 각 socket API 에 맞는 커널 내부의 함수를 호출한다. Socket 생성 API 인 socket()함수가 호출되면 sys_socketcall()을 통하여 sys_socket()을 호출한다. 이 함수는 (그림 3)의 왼쪽편과 같이 여러 함수를 통하여 INET 계층의 진입점인 inet_create()함수를 호출한다. 본 논문의 TOE 커널 모듈은 (그림 2) 에서 설명한 바와 같이 BSD 계층 아래에 위치함으로 inet_create 에 대한 호출을 본 시스템을 위한 socket 생성함수로 대체할 필요가 있다.

sock_alloc()함수에서 inet_create()함수로의 호출은 함수 포인터를 사용하여 이루어진다. 이러한 함수에 대한 포인터는 커널 내부의 net_families 라는 구조체에 저장되어 있다. net_families 구조체는 각 프로토콜 family 에 대한 데이터를 저장하고 있으며, 멤버(member) 중의 하나인 create 는 각 프로토콜 family 에서 소켓 생성을 수행하는 INET 계층함수에 대한 포인터를 지정하고 있다. 따라서, TCP/IP 를 위한 프로토콜 family 인 PF_INET 의 경우, net_families[PF_INET]->create 에 inet_create()함수가 지정되어 있다. 따라서 본 논문에서는 TOE 커널 모듈의 loading 시 net_families[PF_INET]->create 에 TOE 를 위한 소켓 생성함수인 toe_create()로 대체 지정하였다.



(그림 3) 소켓 생성 과정

4.2 소켓 생성 이외의 함수

소켓을 생성한 후 각 소켓에 대한 bind(), send(), receive()등의 socket API 가 호출될 때 이들에 대한 호출도 TOE 커널 모듈로 전달되도록 해야한다. 이러한 작업은 소켓 생성시 생성되는 socket 구조체의 멤버인 ops 를 수정함으로써 이루어진다. ops 는 struct proto_ops 타입으로, 각 socket 에 대한 bind()와 send()같은 socket call 이 발생했을 때 BSD 계층이 호출하는 INET 계층 함수에 대한 함수 포인터를 지정하고 있다. 이러한 ops 값은 소켓 생성시에 설정되며, TOE 커널 모듈의 소켓 생성함수인 toe_create()에서 ops 값을 TOE 를 위한 함수들로 대체하였다. (그림 4)

예를 들어 Linux 의 TCP/IP 스택을 사용하는 send 함수의 경우, (그림 4)와 같이 inet_sendmsg 가 지정되어 있다. 본 논문에서는 이를 TOE 커널 모듈의 send 함수인 toe_sendmsg 함수로 교체하였다.

Linux의 TCP/IP	struct proto_ops ops = {
	.family = PF_INET,
	.owner = THIS_MODULE,
	.release = inet_release,
	.bind = inet_bind,
	.connect = inet_stream_connect,
	……
	.sendmsg = inet_sendmsg,
	.recvmsg = sock_common_recvmsg,
	};

TOE	struct proto_ops ops = {
	.family = PF_INET,
	.owner = THIS_MODULE,
	.release = toe_release,
	.bind = toe_bind,
	.connect = toe_stream_connect,
	……
	.sendmsg = toe_sendmsg,
	.recvmsg = toe_recvmsg,
	};

(그림 4) struct proto_ops 의 함수 포인터

4.3 Linux Kernel 의 수정

4.1 절과 4.2 절에서 설명된 사항을 수행하기 위해서는 Linux Kernel 의 수정이 필요하다. 4.1 절에서 net_famiels[PF_INET]->create 에 TOE 에서 지정하는 소켓 생성함수를 지정한다고 했었다. Linux 커널의 socket.c 를 보면 net_families 구조체를 정의하고 있다. 본 논문에서는 커널 외부의 TOE 커널 모듈에서 net_families 구조체를 사용해야 함으로 (그림 5)와 같이 변수 정의 앞의 static 을 제거하고 EXPORT_SYMBOL 을 추가하였다.

수정전	static struct net_proto_family *net_families[NPROTO];
수정후	struct net_proto_family *net_families[NPROTO]; EXPORT_SYMBOL(net_families);

(그림 5) Linux Kernel 의 수정

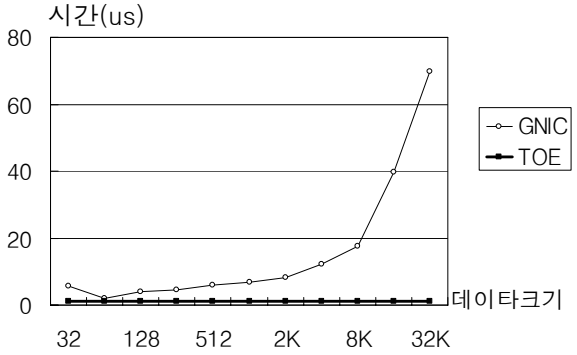
5. 실험

본 실험에서는 Linux 의 TCP/IP 과 TOE 커널 모듈을 사용하였을 때의 성능을 측정하였다. 실험에 사용된 Linux PC 는 Pentium 2GHz CPU, 512MB 메모리, Intel 1000/Pro Gigabit Ethernet Card 를 장착하고 있으며, Linux Kernel 은 2.6.15 버전을 사용하였다.

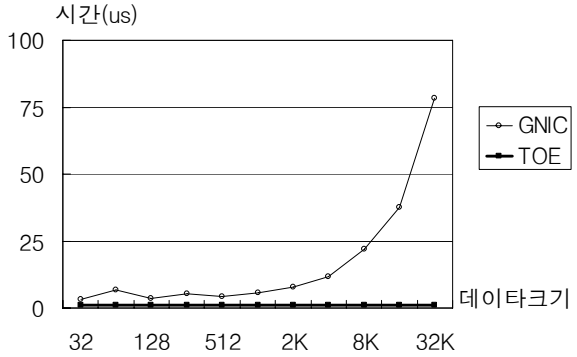
(그림 5)는 데이터 크기 변화에 따라서 send, receive 시에 소요되는 시간을 측정하였다. GNIC 은 Linux 의 TCP/IP 를 사용할 때, 응용 프로그램에서 IP 계층까지 소요되는 시간을 측정하였으며, TOE 는 응용 프로그램에서 TOE 커널 모듈까지 소요되는 시간을 측정하였다.

(그림 5)를 보면 TOE 커널 모듈을 사용한 경우가 Linux 의 TCP/IP 를 사용한 경우보다 훨씬 성능이 좋은 것을 알 수 있다. 이것은 TCP/IP 의 처리를 TOE 에서 담당함으로 TOE 커널 모듈에서 처리해야 할 일이 거의 대부분 감소되었기 때문이다. 또한, GNIC 의 경우 데이터 크기가 증가할수록 송수신에 소요되는 시간이 증가하는데 반하여 TOE 의 경우는 데이터 크기에 관계없이 송수신에 소요되는 시간이 일정함을 알 수 있다. Linux TCP/IP 의 경우, 송수신할 데이터를 커널 내부의 버퍼로 복사함으로 데이터 크기증가에 따라서 송수신 시간이 증가한다. 반면에, TOE 의 경우, TOE 커널 모듈은 응용 프로그램에서 호출된 소켓 call 을 TOE 네트워크 어댑터로 전달하며, 실제 송수신될 데이터는 커널을 거치지 않고 DMA 를 사용하여 TOE 네트워크 어댑터로 바로 전송되기 때문이다.

또한 본 실험에서는 실험에 사용될 테스트 프로그램을 소켓 인터페이스를 사용하여 작성하였으며, GNIC 과 TOE 실험을 진행할 때 테스트 프로그램을 수정 및 재컴파일 하지 않고 동일한 바이너리 코드를 사용함으로써 논 본문의 서론에서 언급한 소켓 인터페이스에 대한 바이너리 호환성을 제공하였다. 또한, (그림 5)의 실험결과에서 보는 바와 같이 TOE 커널 모듈에서 소요되는 시간을 최소화 하였다.



(a) send



(b) receive

(그림 5) TOE 커널 모듈의 송수신시간 비교

6. 결론

본 논문에서는 Linux 시스템의 소켓 인터페이스에 대해서 응용 프로그램의 수정 및 재컴파일을 요구하지 않는 바이너리 수준의 호환성을 제공하는 TOE 용 커널 모듈을 개발하였다. TOE 커널 모듈은 Linux 의 모듈형태로 개발하였으며, 기존 커널의 수정을 최소화하였다. 또한, 실험을 통하여 TOE 커널 모듈이 TCP/IP 를 처리하는 CPU 의 부하를 제거하였으며, TOE 커널 모듈을 경량화함으로써 CPU 에 TOE 커널 모듈로 인하 부하를 거의 발생시키지 않음을 확인하였다. 앞으로, 본 연구진에서 개발중인 TOE 네트워크 어댑터의 개발이 완료되면, TOE 네트워크 어댑터를 포함한 전체 시스템의 성능 측정을 진행할 것이다.

참고문헌

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

[2] “Offload Sockets Framework and Sockets Direct Procotol High Level Design”, Intel, 2002.

[3] J.-S. Kim, K. Kim, S.-I. Jung, and S. Ha, “Design and Implementation of a User-level Sockets Layer over Virtual Interface Architecture”, Concurrency and Computation, Practice and Experience, Volume 15, Issue 7-8, 2003.