Journal of Korea Multimedia Society Vol. 8, No. 11, November 2005(pp. 1483-1495)

표준 소켓 인터페이스에 대한 바이너리 호환성을 제공하는 TOE 지원 모듈의 설계 및 구현

강동재[†], 김재열[††], 김강호[†††], 정성인[††††]

요 약

멀티미디어 데이터를 위한 스트리밍 기술이나 차세대 기술로서 고려되는 유비쿼터스 관련 기술들의 급격한 발달로 네트워크를 사용하는 대용량 데이터 서버들이 증가하는 추세이다. 대용량 데이터의 전송에 있어서 TCP/IP 프로토콜의 처리는 CPU에 많은 부하를 야기시키며 이를 해결하기 위한 방안의 하나로 TOE 디바이스를 적용하고 있다. 하지만 기존의 TOE 지원 모듈의 경우, 해당 TOE 디바이스에 의존적인 인터페이스의 지원으로 인하여 기존의 네트워크 어플리케이션이 TOE의 잇점을 얻기 위해서는 해당 TOE 디바이스가 제공하는 인터페이스를 사용하여 기존 프로그램을 수정하거나 재 컴파일하는 과정을 수행해야 한다는 단점을 갖는다. 본 논문에서는 상기 문제점을 해결하기 위하여 표준 소켓 인터페이스에 대한 바이너리 호환성을 제공하는 TOE 지원 모듈의 설계 및 구현을 제안한다. 본 논문에서 제안하는 TOE 지원 모듈은 리눅스의 네트워크 프로토콜 레이어에 구현되며 기존의 네트워크 어플리케이션이 수정없이 TOE 디바이스의 잇점을 사용할 수 있도록 하고 일반 NIC과 TOE 디바이스의 동시 사용을 지원한다.

Design and Implementation of TOE Module Supporting Binary Compatibility for Standard Socket Interfaces

Dong-Jae Kang[†], Chei-Yeol Kim[††], Kang-Ho Kim[†††], Sung-In Jung[††††]

ABSTRACT

TCP/IP is the most commonly used protocol to communicate among servers, and is used in a wide range of applications. Unfortunately, Data transmission through TCP/IP places a very heavy burden on host CPUs. And it hardly makes another job to be processed. So, TOE(TCP/IP Offload Engine) is considered in many servers. But, most of TOE modules tends to not support binary compatibility for standard socket interfaces. So, it has problems that existing applications should be modified and recompiled to get advantage of TOE device. In this paper, to resolve upper problems, we suppose design and implementation of TOE module supporting binary compatibility for standard socket interfaces. Also, it can make a usage of multiple TOEs and NICs simultaneously.

Key words: TOE(TCP/IP 오프로드 엔진), Binary Compatibility(바이너리 호환성), Standard Socket Interface(표준 소켓 인터페이스)

※ 교신저자(Corresponding Author) : 강동재, 주소 : 대전시 유성구 가정동 161(305-350), 전화 : 042)860-1561, FAX : 042)860-6699, E-mail : djkang@etri.re.kr
접수일 : 2005년 3월 31일, 완료일 : 2005년 7월 6일
[†] 한국전자통신연구원 인터넷서버그룹 연구원
[††] 한국전자통신연구원, 선임연구원
(E-mail : gauri@etri.re.kr)
[†††] 한국전자통신연구원, 선임연구원
(E-mail : khk@etri.re.kr)
[††††] 한국전자통신연구원 시스템소프트웨어연구팀장(책임연구원)
(E-mail : sijung@etri.re.kr)

1. 서 론

최근 멀티미디어 데이터와 같은 대용량 데이터의 네트워크 전송이 증가함에 따라 전송 성능의 향상을 위한 많은 방안들이 고려되어 지고 있다. 특히, TCP/IP의 처리 비용은 일반적인 웹서버에서 네트워크 처리 비용의 50% 이상을 차지하고 있다.[1]

서버에서 TCP/IP 처리 비용를 줄이기 위한 방법으로 TOE(TCP/IP Offload Engine) 디바이스의 도입을 고려하고 있으나,[1] 기존 TOE 지원 모듈은 비표준 인터페이스 지원이나 새로운 어드레스 패밀리의 추가로 인하여 커널이나 기존 어플리케이션이 수정 없이 적용되기 어려운 환경을 제공하여 왔다.[2,8] 따라서 본 논문에서는 표준 소켓 인터페이스에 대한 바이너리 호환성을 제공하는 TOE 지원 모듈에 대한 설계 및 구현을 제안한다.

본 논문에서 언급하는 TOE 디바이스는 TCP/IP 프로토콜 스택을 내장하고 있는 전용 네트워크 디바이스를 의미하며 TOE 지원 모듈은 TOE 디바이스를 지원하기 위하여 커널에 설치되는 소프트웨어 부분을 의미한다. 제안하는 TOE 지원 모듈은 응용 프로그램이 일반 네트워크 인터페이스 카드(NIC)를 사용하는 것과 동일한 방법으로 표준 소켓 인터페이스를 통하여 TOE 디바이스를 사용할 수 있도록 하며 기존 소켓 프로그램이 수정 또는 재컴파일 없이 TOE 디바이스를 사용할 수 있도록 바이너리 호환성(binary compatibility)을 제공한다. 본 논문에서 제안하는 TOE 지원 모듈의 설계 및 구현에 있어서 고려사항은 아래와 같다.

- 기존 리눅스 운영체제가 제공하는 표준 소켓 인터페이스들의 지원.
- 기존 소켓 응용 프로그램들에 대한 바이너리 수준의 호환성 제공.
- 여러 개의 NIC과 TOE 디바이스들의 동시 지원.
- 다양한 형태의 TCP/IP 오프로드 엔진들을 위한 확장 가능한 구조 지원.
- 커널에서 불필요한 데이터 복사의 제거를 위한 어플리케이션 버퍼와 TOE 디바이스의 메모리 사이의 무복제(zero-copy) 기능 지원.

본 논문에서 기술하는 TOE 지원 모듈은 상기와 같은 요구사항을 만족하며 TCP/IP에 대한 시스템의 처리 부분을 TOE 디바이스로 오프로드 함으로서 시스템의 CPU 가용성을 향상시키고 해당 시스템에서 추가적인 작업 수행을 가능하게 한다.

본 논문의 2장에서는 관련 연구인 OSF에 대한 소개 및 표준 소켓 인터페이스의 지원에 대하여 살펴보고, 3장에서는 제안하는 TOE 지원 모듈에 대한 구조 및 주요 기능에 대하여 설명한다. 4장에서는 구현된 TOE 지원 모듈의 오프로드 성능 및 커널에 미치는 영향의 분석을 위한 테스트 수행 및 결과를 기술하고, 마지막으로, 5장에서는 TOE 지원 모듈에 대한 정리 및 향후 요구사항을 기술한다.

2. 관련연구

2.1 Offload Socket Framework(OSF)

현재 TOE와 같은 네트워크 가속기를 사용하는 시스템 구조에 대한 연구가 다수 진행중이며 OSF는 상기 구조의 지원을 위한 대표적인 프레임워크이다.[2,8] OSF는 네트워크 상에 원격의 시스템과 통신을 하는데 있어서 전송 성능의 잇점을 얻기 위하여 일반 NIC의 지원과 함께 TOE나 InfiniBand 디바이스 제공하는 구조이다.[2]

OSF의 프레임워크 구조는 그림 1과 같으며 새로운 프로토콜인 AF_INET_OFFLOAD를 정의하여 새롭게 추가된 네트워크 프로토콜 레이어를 통하여 TOE 및 InfiniBand 디바이스의 사용을 지원한다. 하지만, 새로운 프로토콜 패밀리의 지원은 TOE나

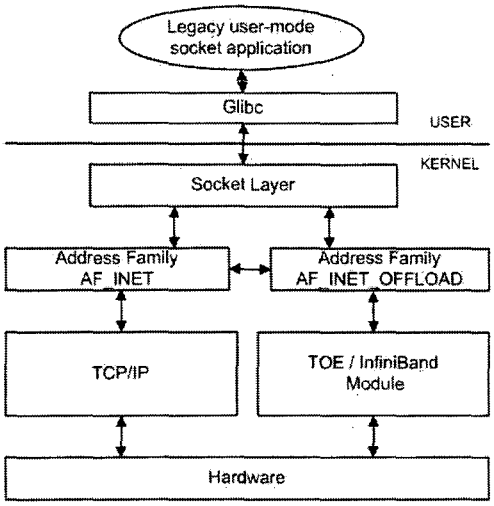


그림 1. OSF 프레임워크의 구조

InfiniBand 디바이스가 제공하는 성능상의 잇점을 사용하기 위하여 기존 어플리케이션의 수정이 요구되는 단점을 갖는다. 국내의 KSOVIA[8] 역시 유사한 구조를 갖는 시스템으로 동일한 단점을 갖는 구조이다. 따라서, 본 논문에서는 기존 어플리케이션이 수정이나 재컴파일 없이 TOE 디바이스를 사용할 수 있도록 하기 위하여 표준 소켓 인터페이스에 대한 바이너리 호환성을 제공하는 TOE 지원 모듈의 설계 및 구현을 수행한다.

2.2 표준 소켓 인터페이스 지원

TOE를 비롯한 VIA, IB(Infini Band)와 같은 네트워크 디바이스에 대한 표준 소켓 인터페이스 지원에 대한 연구는 OSF(Offload Socket Framework)[2] 및 국내의 KSOVIA[11]를 비롯하여 현재까지 다수의 연구가 수행되어 왔다. 상기 연구는 호스트에 장착되는 다양한 네트워크 가속 디바이스에 대한 표준 소켓 인터페이스를 지원함으로서 사용자 편리성과 고성능 네트워크 전송을 구현한다. 이와 같은 표준 소켓 인터페이스를 지원하기 위해서는 기존 어플리케이션을 위한 인터페이스나 커널에 대한 수정이 요구되며 가능한 방식은 아래와 같다.

- *glibc*와 같은 소켓 인터페이스를 구현하는 라이브러리 수정
- AF_XXX 와 같은 새로운 *address family*의 추가 및 해당 프로토콜 구현
- 사용자 수준에서 소켓 호출로의 hooking 지원을 위한 리눅스 로더 수정
- *strace(1), ltrace(1)*에서 사용하는 call intercept 방식 사용

기존 연구 및 결과물은 구현 용이성 및 기존 리눅스의 구현 특성을 고려하여 새로운 *address family*의 추가 방식을 가장 많이 사용하고 있다. 하지만, 새로운 *address family*의 추가는 표준 소켓 인터페이스에 대한 지원은 가능하지만, 기존 어플리케이션에 대한 바이너리 호환성을 지원하지 못한다는 단점을 갖는다.

따라서 본 논문에서 제안하는 TOE 지원 모듈에서는 표준 소켓 인터페이스의 지원과 더불어 기존 어플리케이션에 대한 바이너리 호환성의 지원을 위하여 새로운 *address family*를 추가하지 않으며, 적절한 프로토콜로의 분기를 위하여 별도의 테이블을 유지한다.

3. TOE 지원 모듈

본 장에서는 표준 소켓 인터페이스에 대한 바이너리 호환성을 지원하는 TOE 지원 모듈의 설계 및 구현에 대하여 기술한다. 먼저, 시스템의 전체 구조를 살펴보고, 구현을 위하여 요구되는 주요 자료 구조 및 각 소프트웨어 계층에서의 설계 내용을 다루도록 한다.

3.1 TOE 지원 모듈의 구조

TOE 지원 모듈은 서론에서 기술한 요구사항을 만족시키기 위하여 그림 2와 같이, 세 부분의 소프트웨어 계층으로 구성하며 각각은 소켓 분기 계층(*TSSL*), 오프로드 프로토콜 계층(*TOPL*) 및 디바이스 드라이버 계층(*TDDL*)으로 정의한다.

TOE 지원 모듈은 기존 네트워크 응용 프로그램인 TELNET, FTP 및 웹 브라우저와 같은 프로그램이 수정 없이 TOE 디바이스를 사용할 수 있도록 있도록 해야 한다.

따라서 TOE 지원 모듈이 기존 TCP/IP 프로토콜과 동일한 프로토콜 패밀리 식별자를 가지면서도, 기존의 커널 코드의 수정 없이 동작하기 위해서는 사용자가 요청한 소켓 시스템 호출을 적절한 TCP/IP 프로토콜 스택(TOE를 위한 TOE 지원 모듈 또는 기존 TCP/IP 스택)으로 전달해 줄 수 있는 새로운 계층이

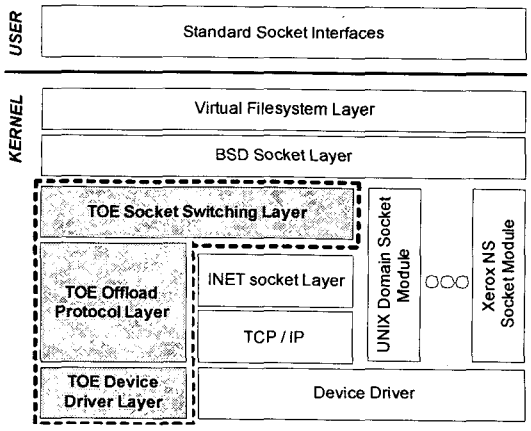


그림 2. TOE 지원 모듈의 구성

추가되어야 한다. 이 계층을 소켓 분기 계층(*TSSL*)이라고 정의한다. 즉, 응용 프로그램에 의한 소켓 호출을 오프로드 프로토콜 계층 또는 기존 INET 프로토콜 계층으로의 분기를 결정해주는 부분이다. 오프로드 프로토콜 계층은 소켓 호출이 TOE 디바이스를 대상으로 하는 경우에 사용되며, 소켓 호출의 수행을 위하여 프로토콜의 의미에 맞도록 디바이스 드라이버 함수를 호출하는 부분이다. 소켓 분기 계층의 수행 요청은 INET 수준의 함수이므로 디바이스 드라이버 함수와는 상당한 의미 차이가 있다. 따라서 그 차이를 구현한 계층이 오프로드 프로토콜 계층 부분이다. 디바이스 드라이버 계층은 오프로드 프로토콜 계층의 요청을 TOE 디바이스에 전달하는 부분으로 디바이스와 TOE 지원 모듈 사이의 통신을 처리하는 계층이다. 상기 기술한 TOE 지원 모듈을 리눅스 네트워크 아키텍처에 적용하여 그림으로 도시하면 그림 2와 같다. 그림에서 TOE 지원 모듈은 굵은 점선으로 표시된 부분이다.

3.2 주요 데이터 구조

TOE 지원 모듈의 주요 자료구조는 *struct tssl_inet_sock*, *struct net_proto_family* 및 *struct proto_ops* 등이 있다. *tssl_inet_sock* 구조체는 소켓 분기 계층 아래의 여러 프로토콜 스택을 구분하기 위하여 사용되며 커널 구조체인 *struct inet_sock* 의 확장된 형태로 구현된다. 즉, 기존의 *struct inet_sock* 을 첫 번째 멤버 변수로 갖는 상위 구조체로서 TOE 지원 모듈을 위한 추가적인 여러 멤버 변수들과 프로토콜 스택을 구분하기 위한 필드를 갖는다. *net_proto_family*와 *proto_ops*는 기존 커널의 자료 구조이며 *net_proto_family*는 프로토콜 패밀리 식별자(*PF_INET*)와 해당 프로토콜 패밀리의 소켓 생성 함수 *CREATE()* 에 대한 포인터를 갖는 구조체이다. 커널은 새로운 소켓을 생성할 때마다 BSD 소켓 계층의 *net_families[]* 배열을 통해 *net_proto_family*에 접근하여 해당 *CREATE* 함수를 수행함으로서 새로운 소켓을 생성한다. *proto_ops*는 소켓 분기 계층의 프로토콜 처리 함수에 대한 포인터로 구성되며, 모든 TOE 지원 모듈의 소켓들은 *proto_ops*에 대한 포인터를 가지고 있기 때문에 TCP/IP 관련 함수들을 수행할 수 있다. *proto_ops*에 설정된 TOE 지원 모듈 관련 함수들은 내부적으로 호스트의 INET 관련 함수를 호출하거나, 오프로드 프로토콜 계층의 함수들을 호출한다. 이외에 중요한 자료 구조로는 *struct proto_stacks*이 있으며 INET과 TOPL 계층 사이의 분기를 결정하기 위한 구조체이다. 일반 NIC과 TOE의 동시 사용 시에 별도의 프로토콜 패밀리를 지정하지 않고 표준 소켓 인터페이스에 대한 바이너리 호환성을 지원하기 위한 구조체이다.

3.3 소켓 분기 계층(TSSL)

본 절에서는 소켓 분기 계층의 주요 설계 내용을 기술한다. 소켓 분기 계층은 TOE 지원 모듈이 기존 TCP/IP 프로토콜과 동일한 프로토콜 패밀리 식별자를 가지면서 커널 코드의 수정 없이 동작하기 위한 계층이다. 커널은 새로운 소켓을 생성할 때마다 BSD 소켓 계층의 net_families 구조체에 접근하여 create로 등록된 함수를 호출함으로서 소켓을 생성한다. 따라서, net_family의 구조에 TOE 지원 모듈을 위한 create함수를 등록하며 생성되는 소켓에 TOE 지원 모듈을 위한 proto_ops 함수들을 할당함으로서 어플리케이션의 소켓 관련 요청을 TOE 지원 모듈로 스위칭 할 수 있게 된다.

가. 소켓의 생성 및 삭제

TOE 지원 모듈은 소켓 생성을 호출에 대하여 기존 TCP/IP 스택을 위한 소켓 외에 TOE 지원 모듈을 위한 추가적인 소켓(*tssl_inet_sock*)을 생성한다(그림 3에서 점선으로 둘러싸인 부분). 두 개의 소켓 구조는 일반 NIC과 TOE 디바이스를 동시에 지원하기 위한 구조로서 생성 시에는 TOPL과 INET 중에서 어느 프로토콜을 사용할 지 결정이 되지 않기 때문에 양쪽 프로토콜을 모두 고려한다.

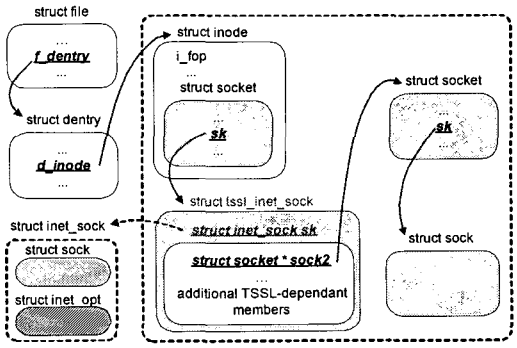


그림 3. TOE 지원 모듈을 위한 소켓 구조

소켓의 생성은 TOE 지원 모듈을 위한 소켓 구조(*tssl_inet_sock*)를 생성한 후, INET 프로토콜을 위한 소켓을 *inet_create*를 이용하여 생성한다. *CREATE* 수행이 완료되면 커널 내에는 그림 3과 같이, 두 개의 소켓 구조체가 생성되며, 각각의 소켓 구조체에 연결된 *struct sock* 구조가 생성된다. (INET을 위한 *struct sock* 구조체와 TOPL을 위한 *struct tssl_inet_sock* 구조체). 또한 두 개의 소켓 구조체내의 *ops* 필드에는 일반 TCP/IP 스택 관련 함수 포인터(*struct proto_ops inet_stream_ops*)와 TOE 지원 모듈을 위한 함수 포인터(*struct proto_ops tssl_stream_ops*)가 셋팅한다.

소켓의 삭제 시에는 INET 프로토콜 스택을 위한 일반 소켓의 삭제를 처리하는 *inet_release*의 호출과 더불어 TOE 지원 모듈을 위한 소켓 구조의 삭제를 위한 추가적인 처리가 수행된다.

나. 표준 소켓 인터페이스에 대한 바이너리 호환성의 지원

본 논문에서 제안하는 TOE 지원 모듈은 기존 어플리케이션이 수정없이 TOE 모듈을 사용할 수 있도록 표준 소켓 인터페이스에 대한 바이너리 호환성을 지원하며, 그림 4는 어플리케이션에서의 소켓 함수 호출 시에 일반 NIC과 TOE 디바이스로의 제어 흐름을 보여준다.

BSD 소켓 인터페이스에 대한 호출은 일반 INET 프로토콜 계층과 TOPL 계층으로의 적절한 분기를 위하여 추가된 소켓 분기 계층의 인터페이스(*tssl_xxx*)를 호출하며 소켓 분기 계층에서의 판단에 의하여 TOPL 프로토콜 계층 또는 INET 프로토콜 계층으로 분기가 이루어진다. TOPL 또는 INET으로 제어가 분기된 후에는 지정된 루틴을 따라서 INET 또는 TOPL을 위한 하위 계층인 TDDL 또는 TCP/ IP 프로토콜 계층의 함수들을 수행한다.

TOE 지원 모듈은 상기와 같이 표준 소켓 인터페이스를 지원하며 기존의 소켓 프로그램들이 수정 또는 재컴파일 없이 TOE 디바이스를 사용할 수 있도록 바이너리 수준의 호환성을 제공하기 위하여 *proto_stacks*라는 구조를 유지한다. *proto_stacks*는 프로토콜 스택의 정보를 관리하기 위한 구조체이며 그림 5와 같은 구조를 갖는다.

프로토콜 스택의 정보는 네트워크 인터페이스의 정보(*tssl_interface*)와 라우팅 정보(*tssl_route*)로 구성되며 해당 디바이스(NIC 또는 TOE)를 구분하기 위한 *interface_id* 필드를 갖는다. 하나의 *proto_stacks*는 독립된 TCP/IP 프로토콜 정보를 관리한다. 예를 들어, 호스트에 일반 NIC이 3개 존재하고 TOE 디바이스가 2개 존재하는 경우, 모두 3개의 *proto_stacks* 구조체가 생성된다. 3개의 NIC은 호스트의 동일한 TCP/IP 프로토콜 스택을 사용하므로 하나의 *proto_stacks* 구조체에 의하여 관리되며, 각각의 NIC은 *tssl_interface*와 *tssl_route* 구조체로 관리되고 동일한 *proto_stacks* 구조체에 연결된다. 하지만, TOE 디바이스는 각 디바이스마다 별도의 TCP/IP 프로토콜 스택을 유지하므로 TOE 디바이스 마다 별도의 *proto_stacks* 구조체가 생성된다. *proto_stacks* 구조체는 소켓의 멤버 변수로서 사용되며 소켓에 관련된 *proto_stacks* 포인터가 기록되어져 제어의 분기 시에 참조된다.

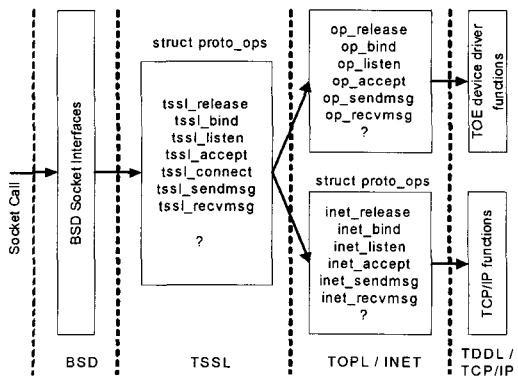


그림 4. 표준 소켓 인터페이스 호출 시의 분기

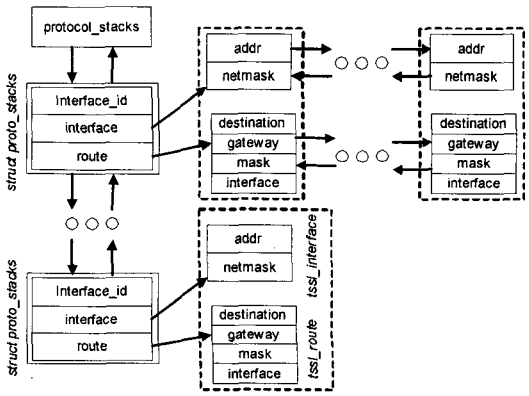


그림 5. proto_stacks의 자료구조

TOE 지원 모듈이 사용되는 시스템에서 응용 프로그램이 소켓 인터페이스를 호출하는 경우, INET과 TOPL로의 분기에 대하여 살펴보자. 분기에 대한 결정은 소켓의 *BIND* 인터페이스 호출 시점에 결정된다. 특정 주소(특정 디바이스)로 *BIND*가 호출되는 경우, 해당 소켓에 BIND된 주소와 관련된 프로토콜 정보(*proto_stacks)*가 할당된다. 동일 소켓에 대한 이후 연산은 할당된 *proto_stacks* 정보를 참조하여 INET 및 TOPL로 분기가 수행된다. 하지만, 다수 어플리케이션의 경우, INADDR_ANY 상태로 *BIND*를 호출하므로 우선적으로 BIND를 수행할 디바이스를 결정하기 위하여 우선순위를 지정해야 한다. 본 논문에서는 TOE 디바이스가 선택의 우선순위를 갖는다는 가정 하에서 기술한다.

*SEND, RECEIVE*와 같은 소켓 연산이 요청되면, 소켓이 사용할 프로토콜 스택의 판단을 위하여 소켓 구조(*tssl_inet_sock*)의 멤버인 *proto_stacks*와 *port* 값을 참조한다. TSSL에서는 소켓 인터페이스가 호출될 때, 적당한 프로토콜 스택으로의 분기를 수행하게 된다. 이때, 분기를 위하여 *port*와 *proto_stacks*(ps)를 참고하며 표 1은 *port*와 *proto_stacks*(ps)에 따른 소켓의 BIND 상태를 결정하기 위한 테이블이다. 표 1에서 "Not bound"의 경우는 INADDR_ANY의 옵션으로 먼저 BIND를 수행한다. "bound to INADDR_ANY"의 상태로 BIND된 경우에는 *proto_stacks*에 등록된 가용한 모든 일반 NIC과 TOE 디바이스들을 대상으로 우선순위에 따라서 순차적으로 연산이 수행되며 가장 먼저 성공하는 프로토콜 스택이 선택된다. BIND가 되어 있는 경우는 *proto_stacks*의 *interface_id*와 *port* 값을 참조하여 분기가 수행된다. TOE 또는 NIC이 단독으로 사용되는 경우, *proto_stacks*에는 해당 디바이스에 대한 프로토콜 스택만이 등록이 되어 TOE 전용 또는 NIC 전용으로 사용된다. 상기와 같이, TOE 지원 모듈은 특정 프로토콜 패밀리를 정의하지 않고, *proto_stacks* 라는 분기 정보를 유지함으로서 기존 어플리케이션들에 대한 표준 소켓 인터페이스에 대한 바이너리 호환성을 제공한다.

라. 데이터 송신 및 수신

본 절에서는 TOE 디바이스와 일반 NIC의 동시 사용 시에 데이터 송신 및 수신에 대한 처리 과정을 기술한다. 데이터 송신 및 수신을 위한 인터페이스는 *send, sendto, sendmsg*와 같은 데이터 송신 시스템 콜이나 *recv, recvfrom, recvmsg*와 같은 데이터 수신 시스템 콜에 의해 호출되는 기능이다. *tssl_sendmsg*는 소켓이 특정 IP 주소로 BIND된 소켓이면 해당 프로토콜 스택의 *sendmsg*함수를 호출한다(특정 일반 NIC 으로 BIND 된 경우이면, *inet_sendmsg*를 호출하고, TOE 디바이스 중 하나로 BIND 되었으면 오프로드 프로토콜 스택의 *op_sendmsg*를 호출). 만일, BIND되지 않은 소켓이면 *get_port* 함수를 호출하여 포트를 할당하고, 할당 받은 포트번호에 대하여 INADDR_ANY로 BIND를 수행한다. INADDR_ANY인 경우, *proto_stacks*의 우선순위에 따른 정보를 참조하여 해당 프로토콜의 SEND 호출을 수행한다. 이때, 먼저 성공하는 *proto_stacks*의 정보가 소켓에 할당되며 이후 데이터 전송에 사용된다. 소켓의 BIND 상태에 대한 판단은 표 1에서 보여주는 바와 같이 *port* 및 *proto_stacks*의 값에 근거하며 *tssl_recvmsg*의 경우도 동일한 알고리즘으로 동작한다. TOE 지원 모듈을 위한 소켓 구조체(*tsslsk*)의 *dst_addr*와 *dst_port*는 BIND되지 않았거나 INADDR_ANY로 BIND된 소켓에서 가장 최근에 *sendmsg*를 수행했을 때의 목적지의 주소와 포트번호를 지정하고, *send_ps*는 가장 최근에 *sendmsg* 를 수행하여 성공한 *proto_stacks*에 대한 정보를 지정한다. 상기 멤버 변수들은 이후에 동일한 목적지에 대해 데이터의 송신 및 수신 요청이 수행될 때 신속한 수행을 위한 정보들이다.

아래의 코드는 *tssl_sendmsg*의 pseudo code이다.

표 1. port and *proto_stacks*(ps) 값에 따른 소켓의 bind 상태 ps : *proto_stacks*

port status	port != null			Port == null
ps status	ps == null	ps != null		*
		Interface_id == NIC	Interface_id == TOE	
bind status	bound to INADDR_ANY	bound to Local NIC(INET)	bound to TOE(TOPL)	Not bound

```
BEGIN tssl_sendmsg(Socket, Msg, Size, Scm)
        SET tsslsk to sk member of Socket
        SET Addr to Source Address of Socket
        IF Port of Socket is zero THEN /* not bound */
                CALL get_port with INET Socket
                SET tsslsk→src_port to new port value
                SET Addr to INADDR_ANY
                FOR each TOE in proto_stacks
                        CALL bind with new port value
                END FOR
        END IF
        IF Addr is Specail IP THEN/*bound to Special IP*/
                IF Addr is NIC's address THEN
                        CALL sendmsg for INET
                ELSE IF Addr is TOE's address THEN
                        CALL sendmsg for TOPL
                END IF
        ELSE IF Addr is INADDR_ANY THEN/* bount to INADDR_ANY */
        IF Dest Address of Msg is tsslsk→st_addr THEN
                        CALL sendmsg for send_ps→interface_id
                ELSE THEN
                        FOR each proto_stacks
                                IF ps→interface_id is NIC THEN
                                        CALL sendmsg for INET
                                ELSE IF ps→interface_id is TOE THEN
                                        CALL sendmsg for TOPL
                                END IF
                                IF sendmsg is successful THEN
                                        SET send_ps, dst_addr, dst_port to current values
                                        CALL return
                                END IF
                        END FOR
                END IF
        END IF
        CALL return
END tssl_sendmsg
```

3.4 오프로드 프로토콜 계층(TOPL)

오프로드 포로토콜 계층은 TOE 디바이스로 분기가 결정된 이후에 수행되는 계층이며 소켓 인터페이스를 수행하기 위하여 프로토콜의 의미에 맞도록 TOE 디바이스 드라이버의 함수를 호출한다. 오프로드 프로토콜 계층에서의 주요한 동작은 소켓 관련 요청에 대한 디바이스 드라이버의 함수 호출과 인터럽트의 발생으로 인한 처리의 구현이며 데이터의 송신 및 수신시에 빠른 처리를 위하여 TOE 디바이스의 메모리와 사용자 버퍼 사이의 무복제 기능을 구현한다.

가. 소켓 명령의 처리

그림 6은 오프로드 프로토콜 계층에서의 명령 처리 과정을 보여주는 그림이다. 그림 6에서 소켓 리스트의 소켓들은 소켓 생성시(*CREATE, ACCEPT*)에 리스트에 추가되며 삭제(RELEASE) 시에 리스트에서 제거된다.

소켓 분기 계층(TSSL)으로부터 전달된 명령은 TOPL에서 호출하는 TDDL 계층의 인터페이스를 통하여 TOE 디바이스로 전달된다. 해당 프로세스는 등록된 명령이 TOE 디바이스에서 처리되어 완료 시점까지 명령과 관련된 *tssl_inet_sock* 구조의 대기 큐

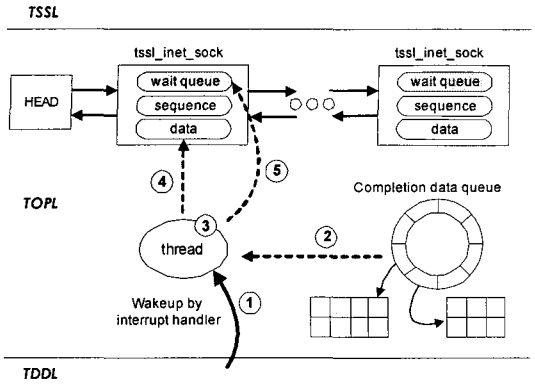


그림 6. 오프로드 프로토콜 계층(TOPL)의 동작 방식

에서 대기한다. TOE 디바이스에서 명령의 처리가 완료되면 인터럽트를 발생시키며 인터럽트 핸들러는 인터럽트 발생 시점까지 완료된 결과 데이터를 그림 6의 완료 데이터 큐(*Completion data queue*)에 등록한다. 이후 TOPL의 처리는 아래와 같은 순서로 수행된다.

(1) 인터럽트 핸들러는 TOPL의 쓰레드를 wakeup 시킨다.
(2) 쓰레드는 완료 데이터 큐(Completion data queue)로부터 결과 데이터들을 가져온다.
(3) 소켓의 *id*와 *sequence* 번호에 근거하여 해당 소켓을 리스트에서 검색한다.
(4) 선택된 소켓의 *data* 필드에 결과 데이터를 등록한다.
(5) 해당 소켓의 대기 큐에서 명령의 완료를 기다리는 프로세스를 wakeup 시킨다.

앞에서 기술한 처리가 완료되면 깨어난 프로세스는 결과 값과 함께 완료를 상위 계층에 통보함으로서 수행이 종료된다.

나. TOE 메모리와 사용자 버퍼 사이의 무복제 지원

오프로드 프로토콜 계층은 데이터 송신 및 수신시에 커널에서의 불필요한 데이터 복사를 피하기 위하여 TOE 디바이스의 메모리와 사용자 버퍼 사이의 무복제 기능를 지원하다. 이것은 사용자 영역 버퍼에 대한 물리 주소를 TOE 디바이스에 전달함으로서 TOE 디바이스에 DMA를 위한 메모리 영역과 사용자 버퍼 영역 사이에 무복제 데이터 교환을 가능하게 한다. TOE 지원 모듈은 페이지 단위의 무복제 데이터 교환을 지원하며 데이터에 대한 x byte-align 주소 제한을 갖는 디바이스도 고려한다. 그림 7은 무복제 데이터 교환 기능 지원을 위한 과정이며 4byte align 제약을 갖는 경우의 예이다.

데이터 전송을 위한 소켓 명령(*SEND, RECEIVE*) 시의 인자인 사용자 영역 버퍼에 대한 주소는 그림 7에서와 같이, 페이지 단위의 물리 주소로 변환이 되어 *sg_list*라는 구조체에 쓰여진다. 커널에서는 커널 및 사용자 영역의 주소에 모두 접근이 가능하므로 사용자 영역의 버퍼 주소를 획득할 수 있으며 획득한 사용자 영역 버퍼의 논리 주소는 페이지 테이블을 검색하여 물리 주소를 얻을 수 있다. 하지만, TOE 디바이스가 x-byte 단위의 주소 제약을 갖는 경우, 사용자 버퍼의 시작 주소가 x byte-aligned 주소가 아닌 경우, x 바이트 단위의 주소로 변환하기 위하여 처음 시작 주소로부터 x byte-aligned 주소가 시작되기 이전의 영역($< x$ byte)은 커널로 복사를 한다. 커널에 복사된 x 바이트 단위의 주소는 물리주소로 변환하여 *sg_list* 에 기록한다. 이러한 과정을 통하여 생성된 *sg_list*가 TOE 디바이스로 전달됨으로서 사용자 영역 버퍼와 TOE 디바이스의 메모리 사이에 무복제 데이터 교환이 가능하다.

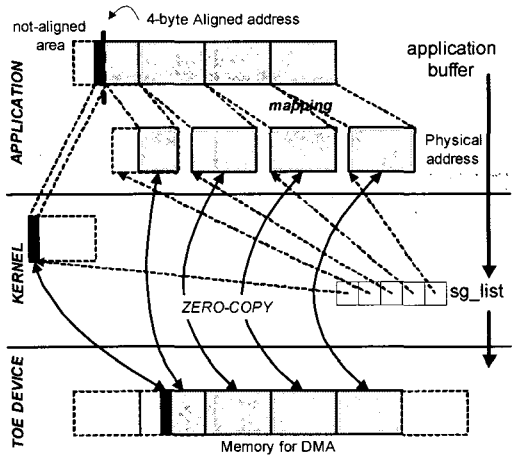


그림 7. 어플리케이션 버퍼와 TOE 디바이스 사이의 무복제 데이터 교환

3.5 디바이스 드라이버 계층(TDDL)

디바이스 드라이버 계층은 TOE 디바이스와 직접적으로 통신하는 부분이므로 안정성을 가장 중요한

요소로 하는 소프트웨어 부분이며 TOE 디바이스와의 메시지 교환과 하드웨어 인터럽트를 사용하여 구현된다.

디바이스 드라이버는 그림 8과 같이 TOE 디바이스와의 통신을 위하여 4개의 원형 큐를 가지며 이를 도어벨(doorbell)이라 한다. 도어벨은 TOE 디바이스의 메모리 영역에 위치하며 디바이스 드라이버와 미리 약속된 주소 지정 방식에 의하여 TOE 지원 모듈과의 공유가 가능하다. 도어벨은 데이터 송신 및 수신을 위한 송신 도어벨(snd_doorbell) 및 수신 도어벨(rcv_doorbell), 데이터의 송신 및 수신 이외의 명령을 위한 명령 도어벨(cmd_doorbell)과 요청의 완료시에 결과를 커널에 전달하기 위한 완료 도어벨(cpl_doorbell)의 4개이다. 상기 4개의 도어벨은 TOE 디바이스와 TOE 지원 모듈 사이에 각각 2개의 레지스터인 헤드, 테일 값을 공유한다. TOE 지원 모듈은 명령 및 송, 수신 도어벨에 요청을 등록하고 인터럽트 발생시, 완료 도어벨로부터 데이터를 가져오고, TOE 디바이스는 명령 및 송, 수신 도어벨에 대한 폴링을 수행함으로서 데이터의 등록을 파악한다. TOE 디바이스로부터 처리가 완료된 결과는 완료 도어벨에 등록되며, 등록 후 인터럽트의 발생을 통하여 커널에 통보한다.

TOE 디바이스가 발생하는 인터럽트 상황은 주요하게 3가지이며 도어벨이 풀(full)인 경우, 하나의 명령 수행이 완료된 경우 또는 미리 지정된 개수의 명령이 완료된 경우에 발생하도록 하며 이는 적용하는 어플리케이션의 특성에 따라서 설정이 가능하도록 지원한다.

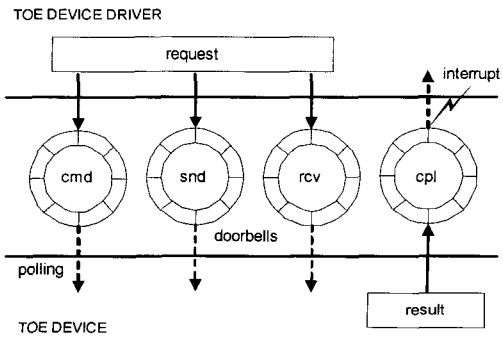


그림 8. TOE 디바이스와 TOE 지원 모듈 사이의 통신을 위한 도어벨(doorbell) 메커니즘

4. 실험 및 결과

본 장에서는 앞에서 제안한 TOE 지원 모듈에 대한 성능 실험의 결과 및 분석 내용을 기술한다.

4.1 실험 환경

본 논문에서 제시한 TOE 지원 모듈의 주요한 목적은 사용자에게 표준 소켓 인터페이스에 대한 바이너리 호환성을 제공하면서 기존 커널에서 수행되었던 TCP/IP 네트워크 프로토콜의 처리 부분을 TOE 디바이스로 오프로드 시키는 것이다.

따라서 본 장에서는 데이터 송신 및 수신시에 TOE 지원 모듈이 TCP/IP 네트워크 프로토콜을 통한 처리 부분을 어느 정도 오프로드 시킬 수 있는지에 대한 시험을 수행하도록 한다. 또한 본문에서 설명한 바와 같이, TOE 지원 모듈의 구현을 위해서는 소켓 구조의 확장된 형태인 tssl_inet_sock 이라는 구조를 커널에 유지해야 한다. 따라서 커널에서 추가적인 구조의 생성 및 운용을 위해서 어느 정도의 영향을 받고 있는지에 대한 측정을 추가적으로 수행한다.

기존에 구현된 TOE 지원 모듈의 경우, 특정 TOE 디바이스를 대상으로 하고 있으며, 알려진 성능 시험의 결과는 TOE 지원 모듈에 대한 성능이 아닌 특정 TOE 디바이스에 대한 성능이므로 본 논문에서 제안한 TOE 지원 모듈의 성능 시험 결과와 비교하는 것은 의미가 없다. 따라서 본 논문에서는 제안한 TOE 지원 모듈에 대한 테스트는 커널에서의 TCP/IP 프로토콜의 처리 시간과 TOE 모듈에서의 처리 시간을 비교하여 제안한 TOE 지원 모듈이 디바이스로 오프로드 할 수 있는 처리 시간을 제시하도록 한다.

그림 9는 TOE 지원모듈에 대한 오프로드 성능을 측정하기 위한 실험 방식을 도시한다. TCP/IP 프로토콜 처리 및 TOE 모듈의 프로토콜 처리를 통한 전송 및 수신 성능은 순수하게 커널에서 소요되는 시간만을 측정하기 위하여 그림 9와 같이 어플리케이션에서 디바이스 레이어 이전까지의 처리 시간만을 측정한다.

본 테스트를 수행하기 위한 환경은 Intel pentium IV-2.6G CPU 및 128M 메모리를 장착한 linux (fedora core II) kernel-2.6.8.1를 사용하는 시스템에서 수행하였다.

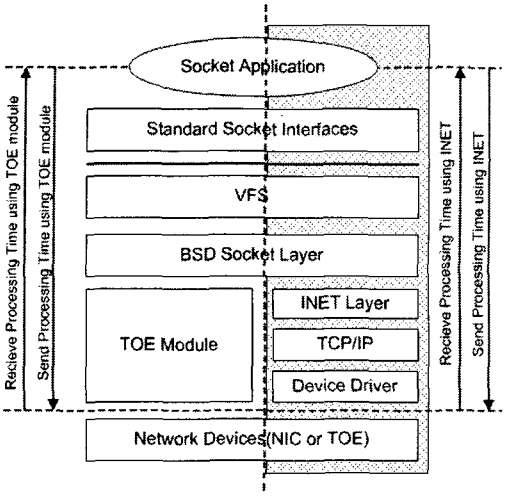


그림 9. TOE module의 오프로드 성능 테스트 방법

4.2 데이터 송신 및 수신 비용의 오프로드

본 절에서는 TOE 지원 모듈이 데이터 송신 및 수신시에 TCP/IP 프로토콜 처리 비용을 어느 정도까지 TOE 디바이스로 오프로드 시킬 수 있는지를 실험한다. 커널 내에서의 데이터의 송신 및 수신을 위한 처리 성능 실험은 그림 9와 같이 수행한다.

그림 10은 일반 INET을 통하여 데이터를 송,수신하는 경우와 TOE 지원 모듈의 TOPL을 통하여 데이터를 송, 수신 하는 경우에 커널에서의 처리 시간을 측정한 결과를 나타낸다. 각 경우의 처리 시간(processing time)은 데이터의 송신 및 수신 시에 커널에서의 처리에 요구되는 시간을 의미하며 count는 동일한 실험의 반복횟수를 나타낸다.

그림 10의 데이터 전송시의 성능 그래프는 *send* 인터페이스의 호출 시에 INET과 TOPL을 통한 커널에서의 처리 비용을 도시한 그림이다. 그림에서 보여주는 바와 같이, INET을 통한 데이터의 전송 시 커널에서 처리 시간은 평균 83.2usec 가 소요되며, TOPL을 통한 데이터의 송신 처리 시간은 46.2μsec가 소요되었다. 따라서 44.5% 가량의 처리 비용을 TOE 디바이스로 오프로드 할 수 있음을 나타낸다. 그림 10의 데이터의 수신시의 성능 그래프는 *receive* 인터페이스의 호출 시에 INET과 TOPL을 통한 커널 처리 비용을 보여준다. INET 계층을 통한 데이터의 수신시 커널에서의 처리 시간은 평균 73.3usec가 소요되며 TOPL을 통한 데이터 수신의 처리 비용은 평균 49.2μsec가 소요되었다. 즉, 데이터 수신의 경우, TOE 지원 모듈을 사용할 경우에 일반 INET 프로토을 사용할 경우의 32.9% 가량의 처리 비용을 TOE 디바이스로 오프로드 할 수 있음을 보여준다. 이상과 같이, 본 논문에서 제안한 TOE 지원 모듈은 일반 네트워크에서의 데이터 송, 수신을 위한 커널에서의 처리 비용을 32~45%가량 오프로드 시킬 수 있으며 오프로드 되어진 처리 비용 만큼 커널이 다른 작업을 수행할 수 있도록 함으로서 서버의 가용성을 높이고 더 많은 클라이언트들의 요청을 수용할 수 있게 된다.

4.3 소켓의 생성 및 삭제에 대한 비용

TOE 지원 모듈은 본론에서 기술한 바와 같이 기존 네트워크 프로토콜 스택에서 사용하는 소켓 구조와 다른 확장된 소켓 구조를 사용한다. 따라서 데이터의 네트워크 전송을 위하여 사용하는 소켓의 생성 및 삭제 시에 기존 소켓에 대한 처리 이외에 TOE 디바이스를 위한 추가적인 구조의 생성 및 운용을 위한 작업이 요구된다.

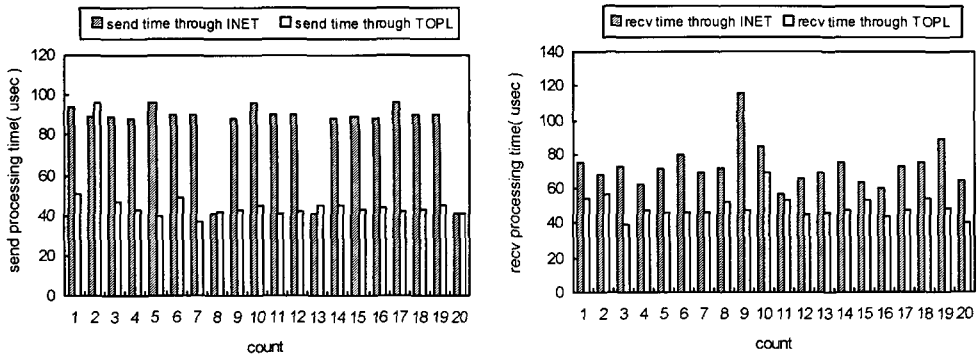


그림 10. INET과 TOPL을 통한 데이터의 송신 및 수신 처리 비용의 비교

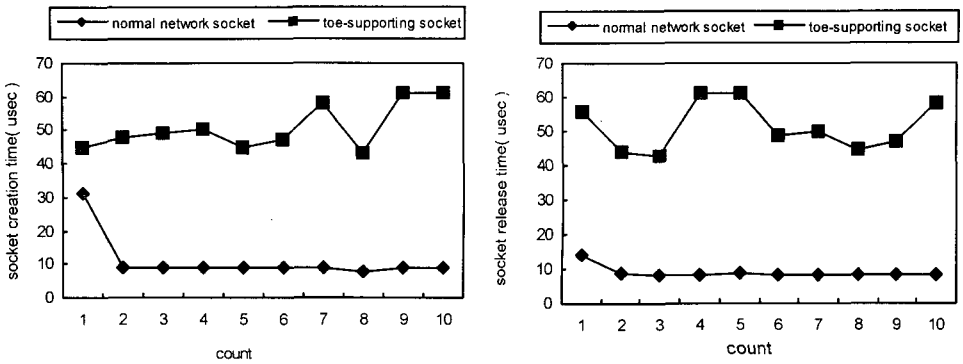


그림 11. 일반 네트워크 소켓과 TOE 지원 모듈을 위한 소켓의 생성 및 삭제 비용

본 실험에서는 어플리케이션 수준에서 소켓 생성 및 삭제 인터페이스를 호출하고 각각의 시스템 콜이 처리 되어 결과값을 반환할 때까지의 어플리케이션 응답시간을 측정함으로서 TOE 지원 모듈을 위한 소켓의 생성 및 삭제에 대한 추가 비용을 측정한다. 그림 11은 TOE 지원 모듈이 지원되지 않는 경우, 일반 소켓에 대한 생성 및 삭제와 TOE 지원 모듈의 지원 시에 소켓 생성 및 삭제에 대한 처리 시간을 측정한 결과를 그래프로 도시한 그림이다.

일반 네트워크 소켓의 생성 시간은 평균 11.1μsec가 소요되었으며 삭제를 위한 어플리케이션 응답 시간은 8.8μsec가 소요되었다. 반면에, TOE-지원 소켓의 경우, 소켓 생성에 소요되는 시간은 50.7μsec이고 소켓 삭제를 위한 응답시간은 51.4μsec가 소요되었다. 결과적으로, 본 논문에서 제안한 TOE 지원 모듈의 소켓 생성 및 삭제를 위한 비용은 일반 네트워크 소켓에 비교하여, 소켓 생성은 4.57배, 소켓 삭제는 5.84배의 처리 비용이 요구되었다.

상기와 같은 결과는 TOE 지원 소켓이 일반 네트워크 소켓 이외의 별도의 추가적인 소켓 생성 및 삭제가 요구되고, TOPL내의 소켓 리스트에서 별도로 관리되며 추가적인 필드에 대한 정보의 할당 및 삭제 비용 등이 요구되기 때문이다. 이러한 추가 비용 등은 추후에 소켓의 생성 및 삭제 부분에 대한 최적화를 통하여 일반 소켓의 생성 및 삭제 비용에 근접할 수 있을 것으로 추산된다.

5. 결론 및 향후 연구

본 논문에서 제안한 TOE 지원 모듈은 표준 소켓 인터페이스에 대한 바이너리 호환성을 지원하여 기존 어플리케이션들이 수정 없이 TOE 디바이스의 잇점을 사용할 수 있도록 지원하였으며 일반 NIC카드와 TOE 디바이스의 동시 사용을 가능하게 하였다. 구현한 TOE 지원 모듈은 새로운 프로토콜 패밀리의 추가 없이 IP 기반으로 디바이스를 선택할 수 있도록 함으로서 상위 어플리케이션들에게 디바이스의 종류에 상관없이 일관된 인터페이스를 사용할 수 있는 유연한 구조를 지원할 수 있다. 또한, TOE 지원 모듈은 TCP/IP에 대한 호스트 시스템의 처리 부분을 TOE 디바이스로 오프로드 함으로서 데이터의 송 수신시에 기존 서버에서의 처리 비용을 30~40% 가량 오프로드 할 수 있음을 시험을 통하여 살펴보았다. 하지만, TOE 지원 모듈의 경우, 소켓 생성 및 삭제 시의 비용이 크므로 소켓의 생성 및 삭제가 적고 상대적으로 데이터의 송 수신량이 많은 응용에 적합하다.

향후 연구로는 TOE 지원 모듈의 소켓 처리 비용의 감소를 위한 최적화 작업이 요구되며, 다수의 TOE 카드 및 일반 NIC카드가 공존하는 경우, 카드간의 load balancing을 지원하기 위한 정책에 대한 연구가 요구된다.

참 고 문 헌

[1] Boon S. Ang, "An Evaluation of an Attempt at Offloading TCP/IP Protocol Processing onto an i9 60RN-based iNIC," *Computer Systems and Technology Laboratory HP Laboratories*. 2001.

[2] Intel, "Architectural Specification Offload Sockets Framework and Sockets Direct Protocol (SDP) High Level Design," *IBM Software Architecture, infiniband.sourceforge.net*, 2002.

[3] Jeffrey C. and Mogu,l, "TCP Offload is a Dumb Idea Whose Time Has Come. Proceedings of HotOS IX," *The 9th Workshop on Hot Topics in Operating Systems, USENIX*, 2003.

[4] "Advantages of a TCP/IP Offload ASIC," *Adaptec*, Inc, 2002.

[5] Hardware Platform Research Team, "Doorbell Mechanism for Network Protocol Acceleration Hardware version1.0," *Digital Home Division, ETRI*, 2004. 10.

[6] Hardware Platform Research Team, "Specification of TOE Software," *ETRI*, 2004.

[7] System SW Research Team, "OS Subsystem Kernel Block Specification ver1.0," *ETRI*, 2003.

[8] 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*, Vol. 15, Issue 7~8, 2003.

[9] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg, "Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer," *in Proceeding 21st Annual Int. Symp. Comp. Arch., IEEE/ACM*, Apr. 1994, pp 142-153.

[10] H.K. Jerry Chu and Vivek Kashyap, "Transmission of IP over InfiniBand," *Internet Draft, IETF*, Jan. 2005.

[11] Gary R. Wright and W. Richard Stevens, *TCP/IP Illustrated*, Vol. 2-The Implementation, Addison-Wesley, 1995.

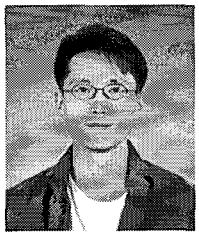
[12] David Mosberger, Larry L. Peterson, and Sean O'Malley, "Analysis of Technique to Improve Protocol Processing Latency," *Proc. of Conference on Applications, Technologies, Architectures, and Protocols for computer communications(SIGCOMM'96)*, August 1996.

[13] Jeff Chase, Andrew Gallatin, and Ken Yocum, "End-System Optimization for High-Speed TCP," June, 2000.(http://www.cs.duke.edu/ari/publications/publications.html)

[14] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley, "Afterburner: Architectural Support for High Performance Protocols," *IEEE Network Magazine*, 7(4), 1995.

[15] J. Brustoloni, "Interoperation of Copy Avoidance in Network and File I/O," *In Proc. INFOCOM 99*, Mar. 1999.

[16] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing," *Proc. 15th ACM symposium on Operating System Principles*, 1995.



강 동 재

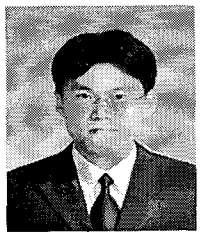
1999년 인하대학교 전자계산공학 학사

1999년~2001년 인하대학교 전자계산공학 석사

2004년~현재 인하대학교 컴퓨터.정보공학 박사과정

2001년~현재 한국전자통신연구원 인터넷서버그룹 연구원

관심분야 : 데이터베이스, 네트워크 연결형 자료저장시스템, 원격 시스템 관리, 공개SW(리눅스)기반기술.



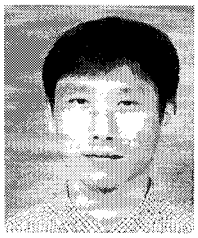
김 재 열

1999년 경북대학교 전자공학과 학사

2001년 경북대학교 전자공학과 석사

2001년~현재 한국전자통신연구원, 선임연구원

관심 분야 : 시스템 소프트웨어, 리눅스 운영체제, 파일 시스템



김 강 호

1993년 경북대학교 전자계산학과 학사
1996년 경북대학교 컴퓨터과학과 석사
1996년~현재 한국전자통신연구원, 선임연구원

관심 분야 : 시스템 소프트웨어, 리눅스 운영체제, 공개소프트웨어

정 성 인

1987년 부산대학교 전자계산학과(이학사)
1989년 부산대학교 전자계산학과(이학석사)
2003년 충남대학교 컴퓨터공학과(공학박사수료)
1999년~2000년 미국 SCO 방문연구원
1989년~현재 한국전자통신연구원 시스템소프트웨어 연구팀장(책임연구원)
관심분야 : 운영체제커널, 공개S/W, 시스템관리, 고가용성, 컴퓨터구조