네트워크 프로세서 기반의 프로토콜 오프로드 엔진 구현 및 성능측정

김주홍○ 김대영 조혜영 정성인*
한국정보통신대학교, 한국전자통신연구원*
{scarlet○, kimd, hycho}@icu.ac.kr, sijung@etri.re.kr*

Implementation and evaluation of Network processor based protocol offload engine

Joohong Kim○ Daeyoung Kim, Hyeyoung Cho, Sung-In Jung*
Dept. of Computer Engineering, Information and Communications University, ETRI*

요 약

본 논문에서는 최근에 급속도로 증가하고 있는 네트워크 트래픽으로 인한 종단서버 시스템의 네트워크 정합부분에서 발생하는 병목현상을 해소하기위한 방안으로 네트워크 프로세서 기반의 프로토콜 오프로드 엔진을 제안한다. 지금까지 프로토콜 재설계, 제로복사 등의 소프트웨어적인 방법으로 시도되던 종단서버 시스템의 네트워크 입출력 처리를 네트워크 프로세서를 이용한 프로토콜 오프로드엔진을 사용하여 처리함으로써 네트워크 입,출력 처리부의 성능을 향상시키고자 한다. 오프로드엔진은 프로그램에 의해 쉽게 재사용, 수정이 용이한 구조로 설계하였다. 본 논문에서는 인텔의 IXP1200를 탑재한 PCI 네트워크 정합장치 개발 보드를 사용하여 프로토콜 오프로드 엔진을 구현하고 그 성능을 측정하였다.

1. 서 론

최근 수십년간 네트워크 전송속도는 매우 빠른 속도로 증가하였고, CPU의 처리속도 증가 추세를 추월하였으며, 현재 백본의 네트워크 전송속도는 초당 테라비트를 능가하고 있다. 이러한 네트워크 전송속도의 향상과 함께 급증하고 있는 멀티미디어 서비스 수요로 인해 서비스를 제공하는 종단서버 시스템에서는 심각한 병목현상이 발생하고 있다. 이는 멀티미디어 스트리밍 서버와 같은 종단서버 시스템의 네트워크 패킷 처리부분이 순수하게 소프트웨어로 구현되어 있어, 증가하는 네트워크 트래픽을 처리하기 위해서 시스템의 CPU가 많은 프로세싱 타임을 소모하기 때문이다. 종단 서버시스템의 CPU에서 발생하는 네트워크 입,출력 병모현상을 줄이기 위한 방안들이 많이 연구되어 왔고, 그 접근 방식에 따라 크게 소프트웨어적인 방법과 하드웨어적인 방식이 있다. 먼저 소프트웨어적인 방법으로는 jumbo frame, TCP segmentation, Copy On Write등과 같은 기술이 있으며, 하드웨어적인 방식은 기존의 종단서버 응용프로세스로 동작하는 프로토콜 스택을 네트워크 정합장치가 처리하게 하는 프로토콜 오프로드 방식이 있다. 여기서 효율적인 프로토콜 처리를 위한 하드웨어를 프로토콜 오프로드 엔진이라고 부른다. [2][3] 본 논문에서는 고성능과 프로그래밍상의 유연성을 제공하기 위하여 네트워크 프로세서를 사용한 오프로드 엔진 구현을 시도 하였으며, 특히 네트워크 프로세서 중 프로그래밍성이 가장 우수한 칩으로 알려진 인텔의 IXP1200 프로세서를 이용하여 프로토콜 오프로드 엔진을 구현하고, 각 모듈의 성능을 측정하였다. 본 논문의 구성은 제 2장 IXP기반의 프로토콜 오프로드엔진에서 프로토콜 오프로드 엔진의 설계 및 구조에 관하여 살펴보고, 3장에서 구현 및 성능 측정 결과에 관해 설명을 하고 제 4장에서 결론과 향후계획을 설명한다.

2. IXP 기반의 프로토콜 오프로드 엔진

인텔의 IXP프로세서는 컨트롤과 관리를 담당하는 스트롱암 코어 프로세서와 패킷 프로세싱을 위해 특화된 구조를 가진 6개의 RISC 구조의 마이크로 엔진 프로세서로 구성된다. 각 마이크로 엔진은 4개의 하드웨어 쓰레드를 멀티 프로세싱할 수 있는 구조로 되어 있으며 각 쓰레드는 프로그래밍에 의해 그 기능을 변경할 수 있다. [1][5]

2.1 IXP 기반의 UDP/IP 프로토콜 스택 구조

그림 1.의 왼쪽은 기존의 리눅스 프로토콜 스택을, 오른쪽은 IXP기반의 프로토콜 오프로드 엔진을 사용할 경우 변경된 프로토콜 스택을 나타낸다. IXP기반의 프로토콜 스택에서 SCP(Socket Control Protocol)계층은 IP 주소에 따라 일반적인 네트워크 카드와 IXP기반의 네트워크 카드로 패킷을 분기시키는 역할과 응용프로세서의 프로세스 ID와 포트정보, 각종 통계정보를 네트워크 카드와 유기적으로 관리하는 역할을 한다.

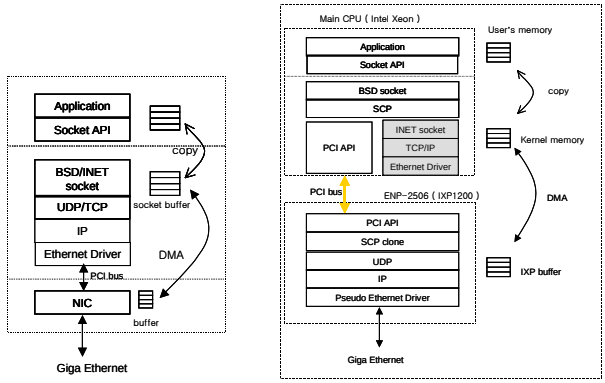


그림 1. IXP 기반의 UDP/IP 프로토콜 스택

2.2 IXP 기반 Modular Architecture

IXP를 이용한 프로토콜 오프로드 엔진 구현을 위해 각 프로토콜 계층을 하나의 프레임워크 단위로 정의 하였다. 이러한 구조는 새로운 기능의 구현이나 기존에 구현된 프레임워크의 수정시 쉽게 기능을 추가하거나 수정할 수 있는 장점이 있다. 각 프레임

워크는 크게 프로토콜의 컨트롤과 관리를 담당하는 Control Plane 과 프로토콜의 패킷 프로세싱을 담당하는 Data Plane 으로 구성된다. Control Plane은 IXP의 스트롱암 코어 응용 프로세스로 동작하며, Data Plane은 마이크로 엔진 응용 프로세스로 동작한다. 아래 그림 2.는 IXP를 기반으로한 UDP/IP 프로토콜 프레임워크 구조를 보여주고 있다. 그림에서 보듯이 각 프레임워크는 SDRAM, SRAM, Scratch 메모리의 데이터를 리소스 관리 모듈을 통해 공유할 수 있는 구조이다.

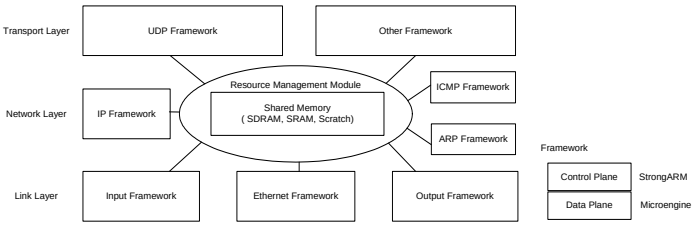
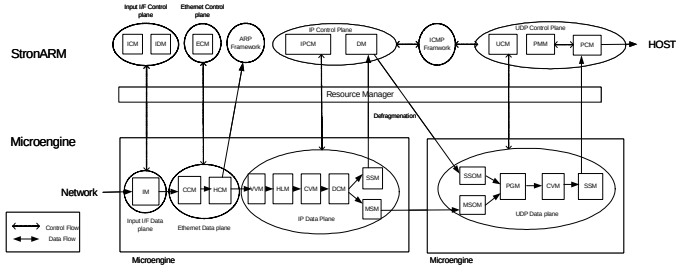


그림 2. IXP 기반 Modular Architecture

2.2 IXP 기반 UDP/IP 프로토콜 오프로드 엔진

2.1절에서 살펴본 바와 같이 각각의 프로토콜은 프레임워크 단위로 설계하였고, 각 프레임워크는 다시 작은 기능 모듈로 구성되어진다. 그림 3.과 그림 4.는 각 프로토콜 프레임워크를 IXP 기반으로 설계한 구조를 보여주고 있다. 하나의 프레임워크는 스트롱암에서 작동하는 컨트롤, 관리 모듈들과 마이크로 엔진에서 실행되고 있는 패킷처리 모듈들이 유기적으로 관계하여 동작하는 구조이다.

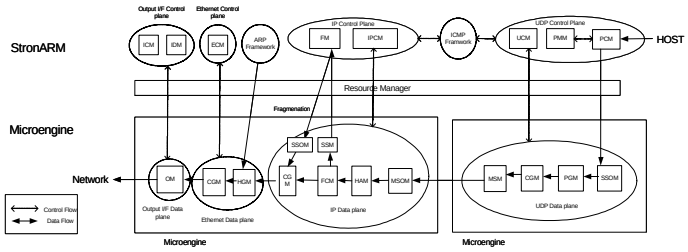


ICM: Interface Control Module
IDM: Interface Driver Module
IM: Input Interface Module
ECM: Ethernet Control Module
CCM: CRC Check Module
HCM: Header,Type Check Module
IPCM: IP Control Module
DM: Defragmentation Module
VVM: Version Verifier Module
HLM: Header Length Check Module
CVM: Checksum Verifier Module
DCM: Destination Check Module
SSM: StrongARM Sink Module
MSM: Microengine Sink Module
UCM: UDP Control Module
PMM: Protocol Management Module
PCM: PCI Control Module
SSOM: StrongARM Source Module
MSOM: Microengine Source Module
PGM: Pseudo header Generate Module
CVM: Checksum Verifier Module
SSM: StrongARM Sink Module

그림 3. IXP기반 UDP/IP 오프로드 엔진 - 수신부

송,수신된 패킷은 그림과 같이 마이크로 엔진내의 화살표를 따라 이동하며 프로토콜 처리를 하게된다. 이때 에러나 예외사항이 발생하였을 때는 패킷 제어권을 스트롱암 응용프로그램에 넘겨 자원의 해제나 에러메시지 송출등을 하게된다. 또한, 패킷 fragmentation과 같은 복잡한 데이터 조작이나 ARP같은 비교적 패킷 프로세싱 오버로드가 작은 작업은 스트롱암 응용프로그램만으로 동작하도록 하였다.

마이크로 엔진에 각각 4개의 하드웨어 쓰레드를 프로그래밍 할 수 있으며, 수신시 그림 3.에서와 같이 인터페이스와 IP 프레임워크를 두개의 마이크로 엔진을 사용하여 총 8개의 하드웨어 쓰레드가 동시에 패킷 수신 및 IP계층 프로토콜 수행을 하도록 하였으며, UDP프로토콜 처리를 위하여 총 4개의 하드웨어 쓰레드를 할당하였다. 송신시에는 그림 4. 에서와 같이 UDP처리를 위하여 총 4개의 하드웨어 쓰레드를 할당하였고, 송신 인터페이스와 IP처리를 위해 총 3개의 쓰레드를 할당하였다. 송신시에는 패킷 버퍼링을 위한 특수한 하드웨어가 지원된다.



ICM: Interface Control Module
IDM: Interface Driver Module
OM: Output Interface Module
ECM: Ethernet Control Module
CGM: CRC Generate Module
HGM: Header Generate Module
IPCM: IP Control Module
FM: Fragmentation Module
CGM: Checksum Generate Module
FCM: Fragmentation Check Module
HAM: Header Adding Module
SSOM: StrongARM Source Module
SSM: StrongARM Sink Module
MSOM: Microengine Source Module
UCM: UDP Control Module
PMM: Protocol Management Module
PCM: PCI Control Module
MSM: Microengine Sink Module
CGM: Checksum Generate Module
PGM: Pseudo header Generate Module
SSOM: StrongARM Source Module

그림 4. IXP기반 UDP/IP 오프로드 엔진 - 송신부

마이크로 엔진과 엔진사이에는 버퍼링을 통한 패킷 전달이 이루어지며, 이러한 패킷 전달시 패킷 손실이 발생하지 않도록 각 모듈의 성능 측정과 예측을 통해 쓰레드를 할당하였다.

3. 구현 및 성능측정

앞장에서 설계한 UDP/IP 오프로드 엔진을 구현하기 위해서 IXP1200을 탑재하고 있는 Radisys사의 ENP-2506p PCI 기바이더넷 보드를 이용하여 구현하였다.

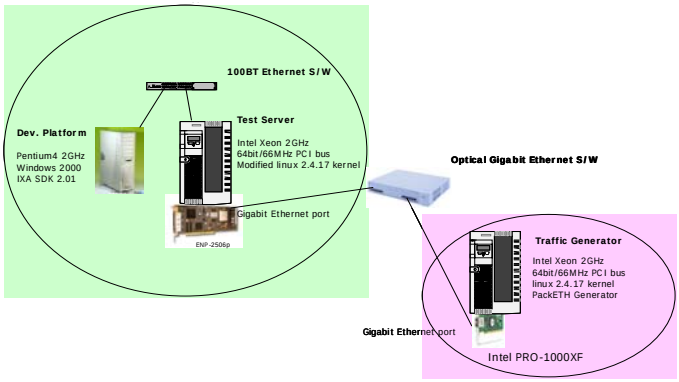


그림 5. 성능측정을 위한 시스템 구성

3.1 성능측정 환경 설정

ENP 2506p 보드의 IXP1200에 구현된 UDP/IP 오프로드 엔진의 성능 측정을 위하여 그림 5.와 같은 측정환경을 사용하였다.

- ENP 2506p

각 마이크로엔진은 232Mhz로 동작하며, 하나의 하드웨어 쓰레드만을 동작 시켰다. 측정을 위하여 CYCLE_CNT 레지스터를 이용하였다. CYCLE_CNT 레지스터는 232Mhz로 동작하는 IXP의 코어 클럭 수를 저장한다.

- Packet generation server

패킷 생성을 위하여 오픈 프로젝트인 packETH를 이용하였다. [4] 성능측정을 위하여 64, 125, 250, 500, 1000, 1250, 1500 bytes의 다양한 크기의 패킷을 생성하였다. 패킷 크기는 이더넷 데이터 영역의 크기이며, UDP/IP 패킷을 포함한다.
각각의 패킷 크기에 1500 개의 패킷을 생성하여 테스트한 결과를 평균하였고, 패킷 생성속도는 1000Base-T 라인속도에서 큰 패킷을 처리하는데 걸리는 패킷당 최대 프로세싱 타임으로 알려진 12.14 마이크로 초 속도로 패킷을 생성하였다.

3.2 UDP/IP data plane 모듈 성능측정 결과

그림 6.의 왼쪽 그림은 IP data plane에서 소모한 총 클럭수를 보여준다. IP plane은 패킷 크기에 따라 소모된 클럭수가 변동이 없음을 알 수 있다. 이는 IP는 프로토콜 처리에 있어서 헤더 부분만을 프로세싱하는데 기인한다. 즉, IP 프로토콜 프로세싱을 위해서는 패킷크기에 관련없이 IP의 헤더 부분만을 마이크로엔진내로 읽어 들이면 되기 때문이다.

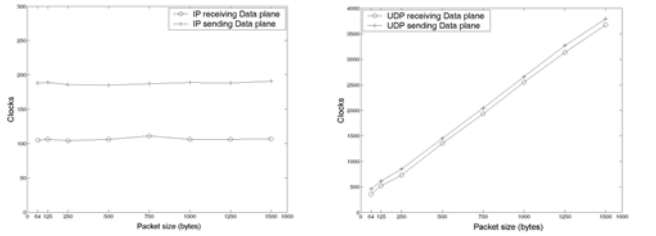


그림 6. UDP/IP data plane 성능측정결과

이에 반해 UDP data plane 처리에 있어서는 패킷 사이즈에 비례하여, 소모된 총 클럭수가 증가하는 것을 볼수 있다. 이는 UDP의 체크섬이 UDP 사용자 데이터 영역까지 포함하기 때문이다. 예를 들면, 250바이트 데이터의 체크섬을 위해7번의 SDARM read 명령이 사용되고, 1500bytes 데이터에는 총 46번의 SDRAM read 명령이 사용되어진다.

표 1. UDP/IP data plane의 프로세싱 타임측정

Packet Size (byte) / Processing time(μs)	64	125	250	500	750	1000	1250	1500
IP receiving Plane	0.45	0.46	0.45	0.46	0.48	0.46	0.46	0.46
UDP receiving plane	1.53	2.25	3.16	5.84	8.34	11.02	13.51	15.83
IP sending plane	0.81	0.81	0.80	0.80	0.81	0.81	0.81	0.82
UDP sending plane	1.98	2.64	3.66	6.25	8.82	11.47	14.09	16.35

표 1.은 각 데이터 처리블록이 소모한 사이클 수를 마이크로 초로 환산한 결과이다.

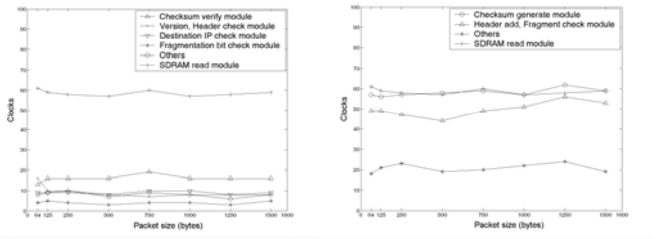


그림 7. IP 블록의 모듈별 프로세싱 타임 측정

그림 7.는 IP처리 블록의 모듈별 프로세싱 타임을 측정한 결과이다. 그림에서 알수 있듯이 패킷처리를 위해 버퍼로부터 데이터를 읽어 들이는 SDRAM read 오프레이션이 가장많은 시간을 소모하는 것을 알 수 있다.

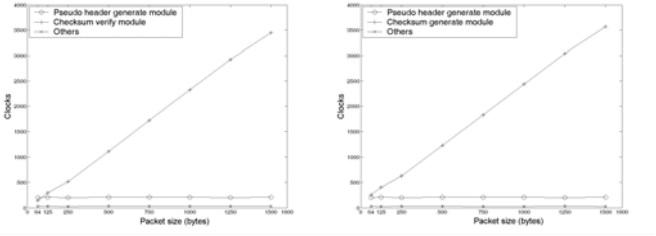


그림 8. UDP 블록의 모듈별 프로세싱 타임 측정

그림 8.은 UDP 처리블록의 모듈별 프로세싱 타임측정 결과이다. 체크섬 처리모듈이 가장 많은 시간을 소모하고 있으며, 이는 체크섬을 위해 사용자 데이터를 버퍼로부터 읽어들이는 데 많은 SDRAM 오프레이션이 사용되기 때문이다.

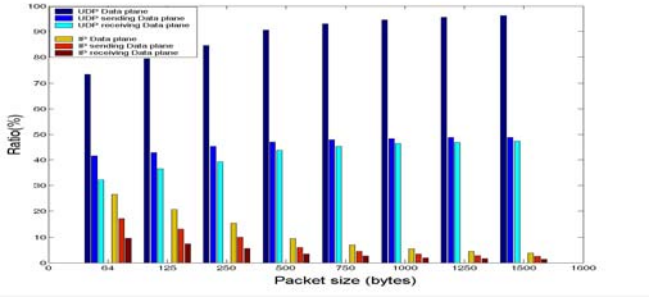


그림 9. 각 data plane의 상대적 프로세싱 타임측정

그림 9.에서 보듯이 64byte의 패킷을 처리할 경우 UDP data plane이 약74% IP data plane이 약26%를 차지했으나, 1500byte의 패킷을 처리할 경우 UDP data plane이 약 95%, IP data plane이 약 5%를 차지했다. 즉, 패킷의 크기가 클수록 UDP data plane의 시간소비 비중이 높음을 알 수 있다.

3.3 UDP/IP data plane 예상 전송속도 측정

3.2 절 표1에서 측정한 각 프로토콜의 프로세싱타임과 2.2절에서 살펴본 쓰레드 할당정책을 사용하여 각 data plane의 예상전송속도를 다음의 식으로 계산하였다.

$$DataRate(Gbps) = \frac{PacketSize(byte) * 8bit * TheNumberOfThread}{ProcessingTimePerPacket(\mu s) * 10^3}$$

단, 하드웨어 쓰레드간의 문맥교환시간은 이론적으로 0이다.

표 2. 각 data plane의 예상 전송속도

Packet size (byte) / Data rates (Gbps)	64	125	250	500	750	1000	1250	1500
I/F plane	1.20	1.25	1.27	1.25	1.31	1.33	1.34	1.35
IP receiving Plane	9.10	17.39	35.56	69.57	100.00	139.13	173.91	208.70
UDP receiving plane	1.34	1.78	2.53	2.74	2.88	2.90	2.96	3.03
IP sending plane	1.90	3.70	7.50	15.00	22.22	29.63	37.04	43.90
UDP sending plane	1.03	1.52	2.19	2.56	2.72	2.79	2.84	2.97

결과에서 알 수 있듯이 2.2절에서 사용한 마이크로 엔진 할당정책을 썻을 때, 64byte 패킷크기를 제외한 패킷 전송 시 I/F&IP 처리 마이크로 엔진과 UDP처리 마이크로 엔진 사이에 전송속도 지연으로 패킷 손실은 발생하지 않음을 알 수 있다.

4. 결론 및 향후 계획

본 논문에서 설계한 IXP기반의 UDP/IP 프로토콜 오프로드엔진의 모듈별 성능측정 결과를 토대로, 보다 향상된 성능을 가지는 차세대 네트워크 프로세서 기반 하에서 고성능의 프로토콜 오프로드 엔진의 구현이 가능함을 알 수 있었다. 고성능의 프로토콜 오프로드 엔진 구현을 위해서 앞으로 현재 고려치 않은 TCP 프로토콜 오프로드 엔진의 구현 및 차세대 네트워크 프로세서 구조에 적합한 마이크로 엔진 할당기법의 연구가 필요하다.

참고문헌

[1] 김주홍, 조혜영, 성종우, 정성인, 김대영, “ 네트워크 프로세서를 활용한 리눅스용 고속 네트워크 프로토콜 설계 및 구현”, 정보과학회 추계학술발표대회 제 29 권 2 호, 628- 630 Page, 2002.10

[2] Burd, D, Zero-copy interfacing to TCP/IP, *Dr. Dobb's Journal*, Volume 20, Issue 9, Pages 68, 70, 72, 74, 76, 78, 106, 108-110, 1995.

[3] Phil Dykstra, Gigabit Ethernet Jumbo Frames And why you should care, WareOnEarth Communications, Inc. 20 December 1999

[4] Ethernet packet generator project, http://sourceforge.net/.

[5] Intel Corporation, Intel IXP1200 Hardware Reference Manual, 2001.

[6] Intel Corporation, Intel IXA SDK ACE Programming Framework: SDK2.01 Tutorial, 2001