IXP1200 네트워크 프로세서를 이용한 Communication Kernel의 구현

황광섭[0] 백성찬 박우진 정영환 안순신
고려대학교 전자공학과
{hanriver, scpaik, progress, youngh, sunshin}@dsys.korea.ac.kr

The implementation of the Communication Kernel on IXP1200

Kwang-Seop Hwang[0] Sung-Chan Paik Woo-Jin Park Sun-Shin An
Computer Network Lab. Dept. of Electronics Eng., Korea University

요 약

인터넷의 급격한 성장과 함께 네트워크 서비스에 대한 사용자의 요구도 점점 증대되고 있다. 이러한 시장의 요구에 빠르게 대응하고 새로운 특징에 대한 시스템의 수정과 보완이 용이하게 되도록 고안된 것이 네트워크 프로세서이며, 본 논문에서는 인텔사의 IXP1200 네트워크 프로세서를 이용하여 기본적인 IP패킷 포워딩기능을 수행하는 Communication kernel을 구현한다. 우리의 구현에서는 8개의 slow port와 1개의 fast port가 하나의 queue를 공유하며, Receive thread가 이 queue를 공유하도록 한다. Communication kernel은 receive scheduler, receive thread, transmit scheduler, transmit thread의 네 개의 모듈로 구성되어진다.

1. 서 론

초기 인터넷 디바이스는 소프트웨어 기반의 간단한 구조를 가지고 있었으나 최근의 많은 사용자와 멀티미디어 서비스를 지원하기에는 역부족이었다. 그래서 나온 방법이 ASIC에 의한 하드웨어적인 방법이었고 최근까지도 역시 이 방법이 사용되고 있다. 그러나 이는 고성능인데 반해, 일단 설계되면 수정이 불가능해서 새로운 특징을 시스템에 첨가하기 위해서는 많은 시간이 소요된다. 이러한 단점의 극복을 위해 탄생한 것이 네트워크 프로세서이며, 프로그램밍이 가능한 소프트웨어적인 방법을 사용하여 라우터, 브리지, layer2 스위치 등의 다양한 네트워크 디바이스들을 시장의 요구에 빠르게 대응하고 새로운 특징에 대한 시스템의 수정과 보완이 용이하게끔 하는 장점이 있다. 또한, 병렬처리, 멀티쓰레드 등의 기술을 채용하여 성능이 계속 증대되고 있으며, 이러한 네트워크 프로세서를 이용한 네트워크 및 통신장비는 날로 확대될 전망이다. 본 논문에서는 IXP1200 네트워크 프로세서를 이용하여 라우터 구현을 위한 초기단계로서 기본적인 패킷의 포워딩 기능을 수행하는 Communication kernel을 구현한다.

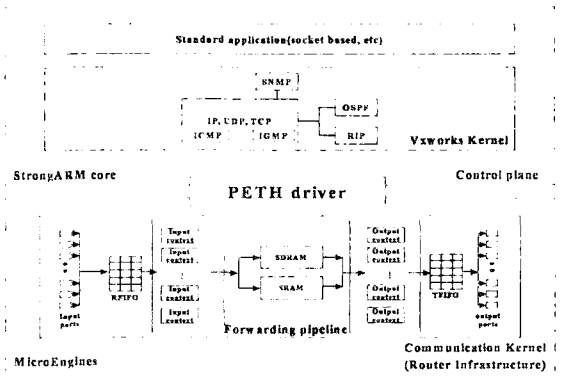
2. 관련연구

2.1 IXP1200 네트워크 프로세서를 이용한 IPv4 라우터의 구조

그림 1에서 보여지는 바와 같이 IXP1200 network processor를 이용한 IPv4 라우터의 전체적인 구성은 크게 MicroEngine의 Communication kernel과 StrongARM core의 control plane으로 구성된다. MicroEngine의 Communication kernel은 기본적인 패킷 포워딩 기능을 수행하고 예외 패킷(e.g. ICMP, IP option, ARP 등)은 StrongARM core의 control plane으로 보내어져 처리된다. 그리고 Communication kernel과 control plane의 논리적인 인터페이스는 PETH(Pseudo Ethernet) 드라이버이다.

2.2 Communication Kernel

Communication kernel은 라인 스피드로 패킷을 처리해야 하고,



[그림1] IXP1200을 이용한 IPv4 라우터의 구조

IP 헤더의 유효성 검사, TTL감소, 체크섬의 재계산, 적절한 출력포트의 선택의 기능을 수행해야 한다. Communication kernel은 forwarding pipeline으로 구성된다. 입력포트로부터 패킷을 받고 패킷의 처리과정을 수행하고 패킷을 큐에 저장하며 출력포트로 패킷을 보낸다. IXP1200은 기본 데이터 유닛으로 64바이트 MAC Packet(MPKT)을 사용하며 입력포트로 수신되는 패킷은 MPKT로 분할되어 처리된다.

- Input Processing

Receive scheduler thread는 입력포트의 receive ready flags를 확인하여 새로운 패킷의 도착을 확인하고 receive thread를 선택하여 패킷 처리를 수행하도록 알려준다. 입력포트에 수신된 64바이트 MAC Packet (MPKT)을 RFIFO로 임시저장하고, 일반적인 IP인 경우 헤더의 수정을 거친 후 하나의 완전한 패킷을 출력을 위해 SDRAM에 저장한다.

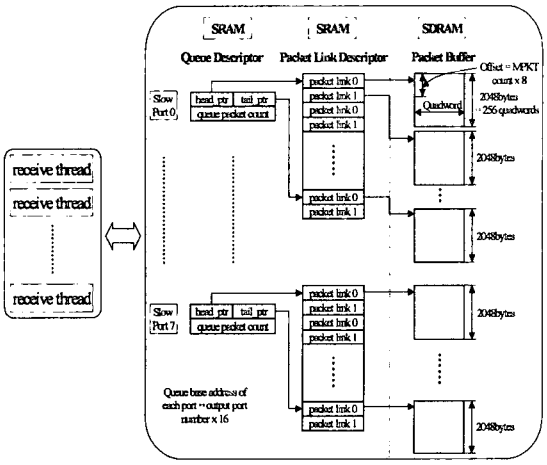
- Output processing

Transmit scheduler는 각각의 출력포트에 대해 출력포트로의 패킷의 전송 순서를 결정하고 transmit thread에게 그것을 알린다.

Transmit thread는 transmit scheduler에게서 받은 메시지를 읽고 출력 포트로 패킷을 전송한다. 이때 패킷은 SDRAM에서 TFIFO로 이동하게 되고 이 과정은 하드웨어에 의해서 자동으로 이루어진다.

• **Queuing discipline**

8개의 slow port와 1개의 fast port는 하나의 queue를 가진다. receive thread는 이 queue를 공유한다. 그림 2는 Transmit queue의 구조를 보여준다. SDRAM의 Packet Buffer는 2KB 데이터 크기로 구성되어지며, 각각의 buffer는 최고 값 1518byte frame을 수용할 수 있다. Transmit Queue는 Queue Descriptor에 의하여 정의되며, SRAM에서 유지된다. Queue Descriptor에는 Transmit Queue의 처음과 마지막 MPKT에 대한 포인터와 MPKT의 개수에 대한 정보가 포함되며, 각각의 패킷은 Packet Link Descriptor의 linked list로서 유지된다. Receive thread는 패킷의 전송이 발생할 때 마다 transmit scheduler에게 알려주며, transmit thread는 Queue Descriptor와 Packet Link Descriptor를 읽어서 패킷을 전송한다.



[그림 2] Transmit Queue의 구조

3. Communication Kernel의 구현

3.1 Microprogram의 전체 구성

우리의 IXP1200 네트워크 프로세서는 8개의 slow port (8*100Mbps)와 2개의 fast port (2*1Gbps)로 이루어져 있으며 본 논문의 라우터 설계에서는 8개의 slow port와 1개의 fast port를 사용한다. Forwarding Engine은 Receive Scheduler, Receive Thread, Transmit Scheduler, Transmit Thread의 4가지 모듈로 구성된다.

3.2 Input Processing을 위한 receive code의 동작

그림 3은 receive code의 동작 순서를 보여준다.

① Receive scheduler는 THREAD_DONE 레지스터 값을 읽고 idle receive thread의 정보를 갱신한다.

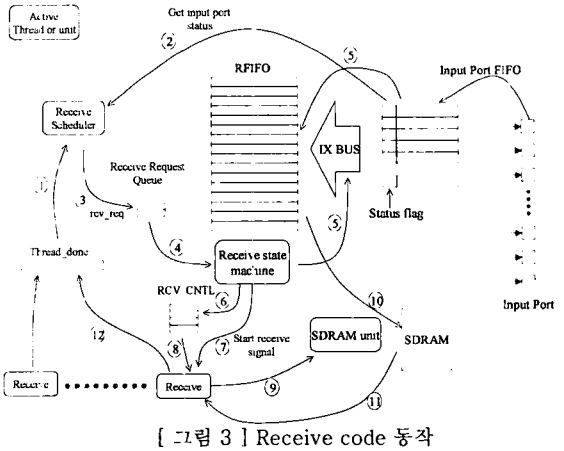
② Receive scheduler는 패킷이 수신된 포트를 찾기 위해 RCV_RDY_LO, RCV_RDY_HI 레지스터 값을 읽는다.

③ Receive scheduler는 서비스될 포트를 결정하고 receive request를 RCV_REQ 레지스터에 쓴다.

④ Receive state machine에서 RCV_REQ 값을 읽는다.

⑤ Receive state machine은 입력포트로부터 RFIFO로 패킷을 이동시킨다.

⑥ Receive state machine은 RCV_CNTL 레지스터에 패킷 정보를 쓴다.



[그림 3] Receive code 동작

⑦ Receive state machine은 receive scheduler에서 명시한 receive thread로 START RECEIVE signal을 보낸다.

⑧ START RECEIVE signal를 받은 receive thread는 RCV_CNTL레지스터 값을 읽어서 처리해야 할 패킷 정보를 얻는다.

⑨ receive thread는 목적지 주소와 RFIFO element number를 가지는 RFIFO read 명령을 SDRAM unit으로 보낸다.

⑩ SDRAM unit은 할당된 주소에 해당하는 SDRAM의 위치로 패킷을 이동시킨다.

⑪ SDRAM unit이 이동을 완료하면 receive thread에게 signal을 보낸다.

⑫ receive thread는 THREAD_DONE 레지스터에 idle로 표시하고 다시 START_RECEIVE signal을 기다린다.

3.3 Input Processing을 위한 pseudo-code

그림 4는 각각의 MPKT를 받았을 때 수행 과정에 대한 pseudo-code이다. *p*는 포트 넘버, *e*는 RFIFO의 인덱스, *MP_addr*은 MPKT이 저장된 메모리 주소, *reg_mpkt _data*는 MPKT를 유지하고 있는 MicroEngine의 레지스터, *state*는 MPKT의 처리방법에 대한 정보가 포함된 데이터 구조이다.

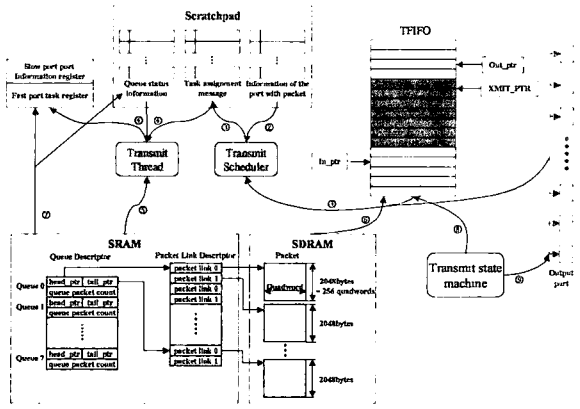
```
Input_Processing_Loop:
1 if (!(thread_rdy(t) && port_rdy(p)) goto Input_Processing_Loop
2 choose RFIFO(e)
3 send receive request
4 load RFIFO(e)
5 MP_addr = calculate MP buff_addr & packet_link_addr
6 copy reg_MP_data <- RFIFO(e)
7 state=protocol_processing(reg_MP_data)and route_lookup(dest_addr)
8 copy reg_MP_data -> SDRAM(MP_data)
9 if End_Of_Packet
10 enqueue(state, queue_state)
11 goto Input_Processing_Loop
```

[그림 4] Input processing을 수행하는 context의 pseudo-code

pseudo-code의 1~4 line은 receive scheduler의 기능이다. 어느 포트(p)에 새로운 MPKT를 수신했는지를 확인하고, idle thread를 선택하여 RFIFO(e)를 할당하며, receive thread에게 receive request를 한다. Load동작은 Receive-State-Machine이 MPKT을 포트 버퍼로부터 RFIFO로 복사하는 것이다.

일단 MPKT가 RFIFO안으로 들어오면 MicroEngine은 MPKT을 자신의 레지스터로 복사한다. 레지스터에 복사된 MPKT에 대해 protocol processing(헤더 검증, TTL감소, 체크섬 재계산, 목적지 MAC 주소 세팅)을 한 후, route lookup을 수행한다. 본 논문의 구현에서의 lookup algorithm은 IP longest prefix match lookup방법을 사용한다. Core Route Table Manager에 의해 쓰여진 table을 사용하며 이는 hi64K entry table과 trie block lookups의 dual lookup을 사용한다. 최악의 경우에 5 SRAM reference signal이 소모된다. protocol processing과 route lookup을 한 후 MPKT는 레지스터에서 SDRAM으로 복사된다. 만약 MPKT이 들어온 패킷의 최초의 것이거나 패킷이 하나의 MPKT이면 패킷의 헤더가 포함되어있는 것으로 가정하고, 패킷 처리과정의 결과는 패킷의 destination queue로 명기한다. 예외 패킷은 일반적인 Output processing대신 StrongARM에 의해 처리될 수 있도록 queue에 저장한다

3.4 Output Processing을 위한 transmit code의 동작



[그림 5] Transmit code 동작

①Transmit scheduler는 포트의 ready 정보, TFIFO transmit pointer(XMIT_PTR)정보를 얻고 사용 가능한 TFIFO element를 조사한다.
② Transmit scheduler는 scratchpad memory에서 패킷을 가진 포트정보를 가져온다. 이 정보는 receive thread가 패킷을 enqueue시킬때 쓴 정보이다. Transmit scheduler는 서비스될 포트를 찾고, TFIFO와 포트를 연결시키며, queue를 선택한다.
③ Transmit scheduler는 scratchpad memory에 포트, TFIFO element, queue정보를 가지는 task assignment message를 쓴다. Slow port와 fast port에 따라 다른 정보가 들어간다.
④ Transmit thread는 scratchpad memory에서 task assignment message를 읽어서 slow port information 레지스터와 fast port task 레지스터에 저장한다.
⑤ Transmit thread는 SRAM에서 queue descriptor, packet link descriptor를 읽는다.
⑥ Transmit thread는 MPKT를 SDRAM에서 TFIFO로 보내고 control 정보를 TFIFO control field에 기록한다.
⑦ Transmit thread는 포트 status와 fast task를 갱신한다.
⑧ Transmit state machine(TMS)는 TFIFO validate bit을 검사한다.
⑨ TFIFO validate bit가 세팅되어 있으면 TMS는 MPKT를 출력포트로 보낸다.

3.5 Output Processing

그림 6는 패킷 전송에 대한 pseudo-code를 보여준다. 코드의 1~2 line은 transmit scheduler의 기능이다. Transmit schedueIr는 출력포트의 ready상태를 결정하고 TFIFO를 선택하고 TFIFO와 출력포트를 연결시킨다. 이 정보를 task assignment message에 저장하고 transmit thread에게 전달한다.

```
Output_Processing_Loop:
 1 choose TFIFO(e) and port_rdy(p)
 2 bind TFIFO(e) and port_rdy(p)
 3 read task assignment message
 4 if SOP
 5 qid = select_queue()
 6 state = dequeue(qid)
 7 copy SDRAM[MP_addr] -> TFIFO(e)
 8 write TFIFO control field
 9 start_transfer(p,e)
10 goto Oupput_Processing_Loop
```

[그림 6] Output processing을 수행하는 context의 pseudo-code

Transmit thread는 task assignment message를 읽는다. 각각의 출력포트에 대해 select_queue operation은 포트와 관련된 queue의 set중 비어있지 않은 queue를 선택한다. packet descripor는 선택된 queue로부터 dequeue된다. 패킷의 각 MPKT에 대한 MPKT의 SDRAM 주소가 계산되고, 사용 가능한 TFIFO slot이 선택되어 MPKT가 SDRAM으로부터 TFIFO로 복사된다. MPKT의 정보는 TFIFO control field에 저장되고, Transmit-State-Machine은 패킷을 출력포트로 보낸다.

4. 결론 및 향후연구

본 논문에서는 인텔사의 네트워크 프로세서인 IXP1200를 이용하여 Communication kernel을 구현하였다. IXP1200를 사용하는 IPv4라우터의 구조는 Communication kernel과 control plane으로 구성되며, Communication kernel은 기본적인 IP패킷 포워딩 기능을 수행하며 예외적인 패킷은 control panel로 보내어 처리되도록 한다. Communication kernel은 receive scheduler, receive thread, transmit scheduler, transmit thread의 4가지 모듈로 구성된다. 본 논문의 구현에서 전송률은 2.01Mpps로 측정되었으며, 이론적인 최고치인 2.67Mpps와는 다소 차이를 보였다. 이는 receive context들이 하나의 queue를 공유하여 queue사용에 경쟁이 일어나기 때문이다. 또 한가지 이유로는 MicroEngine에서 4가지 모듈이 나누어져 있기 때문이다. 이는 비효율적인 parallel resource의 사용을 불러 일으킨다. 따라서, 향후에는 IXP1200 MicroEngine에서의 parallel resource 관리에 대한 연구가 필요하다.

그리고, RFC1812에서 정의된 라우터의 기능을 만족하기 위해서 예외적인 패킷 처리 부분을 Microprogram에 첨가해야 하고, OSPF나 RIP과 같은 프로토콜을 사용하여 동적으로 route table의 갱신이 가능하도록 구현해야 하는 과제가 남아있다.

5. 참조 문헌

[1] "IXP 1200 Hardware Reference Manual" Intel, June, 2001
[2] "IXP 1200 Microcode Programmar's Reference Manual", Intel, June, 2001
[3] " IXP1200 MicroSoftware Reference Manual" Intel, June, 2001
[4] RFC 1812 "Requirements for IP Version 4 Routers"