논문번호 12-37B-09-11

한국통신학회논문지 '12-09 Vol.37B No.09
http://dx.doi.org/10.7840/kics.2012.37B.9.814

# 40 Gb/s 실시간 플로우 관리 네트워크 프로세서 구현

두 경 환◆, 이 범 철*, 김 환 우**

# Implementation of 40 Gb/s Network Processor of Wire-Speed Flow Management

Kyeong-Hwan Doo◆, Bhum-Cheol Lee*, Whan-Woo Kim**

요 약

본 논문에서는 하드웨어 기반의 플로우 수락 제어 알고리즘(FAC)을 이용하여 실시간 플로우 관리가 가능한 네트워크 프로세서인 옴니플로우 프로세서를 제안한다. 옴니플로우 프로세서는 플로우 연결 설정 및 해제를 실시간으로 처리하므로 플로우 업데이트 주기를 짧게 설정할 수 있고, 이 주기 내에 입력되지 않는 패킷들이 속하는 플로우의 연결을 해제함으로써 실제 유효한 플로우만을 효율적으로 관리할 수 있다. 그러므로 FAC를 통해 TCP 뿐만 아니라 UDP 응용서비스의 전송 신뢰성을 높힐 수 있다. 이 프로세서는 65nm CMOS 공정에 의해 총 2천5백만 게이트 용량의 칩으로 제작되었으며, 패킷 처리를 위한 32개의 RISC 코어를 이용하여 최대 동작 주파수가 555MHz 일 때 40Gb/s의 처리 성능을 갖는다.

**Key Words** : 플로우 프로세서, 네트워크 프로세서, 멀티코어, 수락 제어, 병렬 처리

ABSTRACT

We propose a network processor called an OmniFlow processor capable of wire-speed flow management by a hardware-based flow admission control(FAC) in this paper. Because the OmniFlow processor can set up and release a wire-speed connection for flows, the update period of flows can be set to a short time, and only active flows can be effectively managed by terminating a flow that does not have a packet transmitted within this period. Therefore, the FAC can be used to provide a reliable transmission of UDP as well as TCP applications. This processor is fabricated in 65nm CMOS technology, and total gate count is 25 million. It has 40 Gb/s throughput performance in using the 32 RISC cores when maximum operating frequency is 555MHz.

## Ⅰ. 서 론

네트워크의 사용량이 증가하고 사용 형태가 다양화됨에 따라 서비스 수준별 패킷 처리 요구 및 악의적 공격에 대한 방어 등의 추가적 기능에 대한 요구가 증대되고 있다. 특히 P2P 등의 대규모 대역폭을 요구하는 애플리케이션이 증가함에 따라 전체 네트워크의 트래픽이 급속도로 증가하고 있다. 패킷 스위칭 네트워크에서 패킷의 플로우는 동일한 프로토콜을 사용하고, 동일한 목적지 인터넷 프로토콜

※ 본 연구는 방송통신위원회의 미래인터넷 원천기술개발사업 연구결과로 수행되었음 (KCA-2012-12-921-05-001)
◆ 주저자 : 한국전자통신연구원 광인터넷연구부, khdoo@etri.re.kr, 정회원
* 한국전자통신연구원 광인터넷연구부, 정회원
** 충남대학교 전자공학과, wwkim@cnu.ac.kr, 종신회원
논문번호 : KICS2012-07-307, 접수일자 : 2012년 7월 6일, 최종논문접수일자 : 2012년 9월 3일

(IP) 주소 및 포트와 특정 소스 IP 주소 및 포트 사이의 패킷 흐름으로 정의될 수 있다[1,2]. OSI 7 모델 계층 4에서 플로우를 제어하기 위해 사용되는 두 가지 프로토콜에는 TCP와 UDP가 있다. TCP 제어하의 웹 세션, FTP 연결, 전자 상거래 등과 같은 서비스는 패킷 레벨이 아닌 플로우 또는 세션 레벨에서 패킷이 손실되면 패킷 재전송을 통해 플로우의 무결성을 보장 받을 수 있는 트래픽 제어 및 관리가 가능하다. 반면 UDP는 연결 설정 절차 없이 단방향 통신을 허용하는 간단한 전송 모델로서 패킷의 무결성, 순서 등을 보장할 수 없기 때문에 패킷 손실에 덜 민감하고 시간 지연이 적은 IPTV, VoIP, 온라인 게임 등에 적용된다. 그러므로 지금까지의 연결 수락 제어(Connection Admission Control)에 대한 연구와 구현은 TCP 플로우에 대해 제한적으로 이루어져 왔다[3,4]. 계층 2에서 이루어지는 MPLS(Multi Protocol label Switching) 프로토콜을 통해 UDP를 포함한 모든 인터넷 프로토콜의 연결설정이 가능하지만 연결 설정 속도가 매우 느리기 때문에 짧은 플로우에 대해서는 효과적이지 않으며, 플로우별로 고정된 대역이 할당되므로 링크 이용률이 낮은 단점이 있다. 일반적인 라우터는 플로우를 구분하지 않기 때문에 각각의 패킷을 독립적으로 처리한다. 따라서 플로우 레벨의 백오프 메커니즘이 없는 UDP 플로우에 혼잡이 발생하면 플로우 구별 없이 패킷이 무작위로 폐기되므로 음성 서비스의 잡음 또는 비디오 화면 깨짐과 같은 품질 저하의 결과를 초래하게 되며, 동일한 품질의 클래스 내에서는 어떤 UDP 플로우도 완전한 무결성이 보장 될 수 없다.

본 논문에서는 플로우 단위로 QoS를 보장하고, 플로우 연결 제어(FAC)를 하드웨어 기반으로 구현한 옴니플로우 프로세서를 제안한다. FAC를 이용하여 UDP 플로우의 연결을 설정하고, 혼잡 상황에서 새로운 플로우를 거절함으로써 현재 서비스되고 있는 다른 플로우 서비스의 패킷 손실을 방지한다. 하드웨어 기반 FAC를 통해 플로우 정보 테이블 내에 등록되어 있지 않은 플로우를 실시간 속도로 연결 설정하고, 사용자가 정한 임의의 주기 동안 감시하여 전송된 패킷이 없는 플로우의 연결을 해지한다. 실시간 서비스 FAC 방법을 이용하면 서비스 중인 플로우의 라우팅 경로를 분석할 수 있으므로 DDoS 공격을 효과적으로 검출하고 방어할 수 있다[7].

본 논문은 다음과 같이 구성된다. 2장에서 플로우 프로세서의 구조와 기능을 설명하고, 3장에서는 시뮬레이션을 이용하여 대역 이용률 및 패킷 손실률 관점에서 기존 모델과 비교하여 제안된 FAC를 적용한 시스템의 성능을 평가한다. 4장에서는 구현된 옴니플로우 프로세서 칩의 특징을 보여주고, 마지막으로 5장에서 결론을 맺는다.

## Ⅱ. 플로우 프로세서의 구조와 기능

### 2.1. 병렬 패킷 처리

네트워크 프로세서는 패킷의 패턴 및 주소 검색, 큐 관리, 스케쥴링 등의 패킷 처리 기능을 고속으로 처리하는 네트워크 패킷 처리 전용 소자로서 범용 프로세서와 같이 소프트웨어에 의한 프로그래밍이 가능한 특징을 가지고 있다. 일반적으로 네트워크 프로세서는 처리 성능의 향상을 위하여 멀티프로세서 구조로 구현된다. 식 (1)인 Amdahl의 법칙[8]에 따르면 프로세서 처리 성능($S$)은 프로세서 코어의 수($N$)에 따라 비례적으로 증가하지만, 선형적으로 증가하지 못하고 일정 수준에서 포화되며, 병렬 처리 프로그램의 비율($f_p, 0 \le f_p \le 1$)에 더 큰 영향을 받는다. 따라서 처리 성능을 높이기 위해서는 병렬 처리 기술이 필요하지만, 효율적인 병렬처리를 위해서는 각 프로세서에 부하를 적절히 할당하는 방법이 병행되어야 한다.

$$S = \frac{1}{(1 - f_p + f_p/N)} \quad (1)$$

패킷 처리의 병렬화를 위해서는, 기본적으로 입력되는 패킷 헤더를 각 프로세서에 순차적으로 할당하는 작업이 선행되어야 한다. 그림 1은 일반적인 네트워크 프로세서와 플로우 프로세서의 패킷 처리 순서를 보여 주고 있다. 그림 1에서 A 플로우에 속하는 n번째 도착 패킷을 An으로 표시하며, 6개의 플로우를 A에서 C로 표시한다. 패킷 프로세서는 3개의 멀티프로세서(MP1, MP2, MP3)로 구성되며, 각 프로세서는 2개의 스레드(thread)(Ta, Tb)를 포함하고 있다. 스레드는 프로세스 내부에 존재하는 수행 경로로서, 일련의 실행 코드의 실행 작업이 이루어지는 부분이다. 일반적인 네트워크 프로세서는 그림 1 (a)와 같이 도착하는 패킷의 순서대로 각각의 패킷을 독립적인 프로세서 스레드에 할당하고, 처리가 끝난 순서에 따라서 패킷이 출력되므로 동일한 플로우에 속하는 패킷 A2는 A1보다 먼저 출력될 수 있다. 이는 패킷의 처리 속도, 각 스레드의 메모리 사용 요청 대기 시간 차이 등의 영향에 의

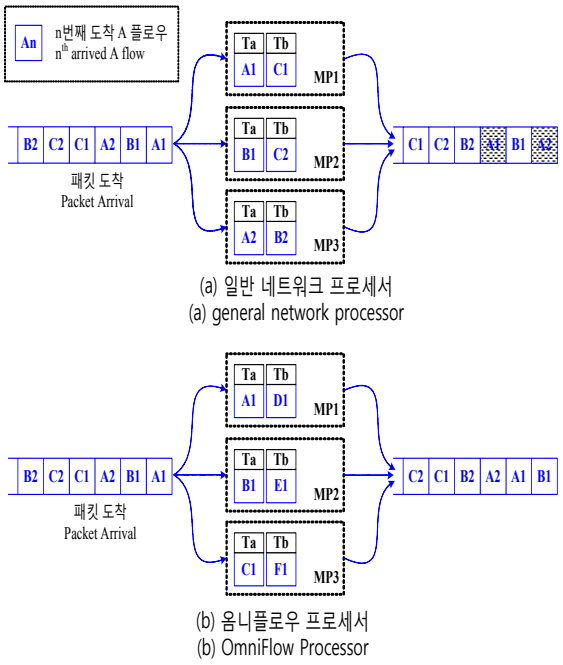


그림 1. 멀티 프로세서 패킷 처리 순서
Fig. 1. Packet processing order of a multi-processor.

해 발생된다. 따라서 네트워크 프로세서에서는 처리되는 패킷의 순서를 유지하기 위한 별도의 방법을 사용해야 한다. 대표적인 예로 시스코 사의 병렬 처리 방식은 패킷에서 플로우를 생성하여 플로우 및 프로세스 단위로 복수 프로세서에 배정하고, 복수의 프로세서 내에서 각각의 순서를 재배열 한다[9,10].

반면 옴니플로우 프로세서는 패킷들의 플로우를 구별하여 동일한 플로우에 속하는 패킷들은 동일한 프로세서의 스레드에 할당한다. 그림 1 (b)는 플로우 기반의 병렬 패킷처리 절차를 설명하기 위한 그림이다. A플로우에 속하는 모든 패킷들은 MP1의 스레드 Ta에 할당된다. 따라서 MP1의 Ta에서 A1이 처리되고 있을 때 다른 프로세서가 빈 상태라 하더라도 A2는 MP1의 Ta에서 A1의 처리가 완료될 때 까지 대기한다. 이와 같이 플로우별로 패킷이 분류되어 동일한 플로우는 동일한 프로세서의 스레드에 할당되므로 서로 다른 프로세서에 할당된 패킷들의 순서가 뒤바뀌어도 재정렬할 필요가 없다.

### 2.2. 옴니플로우 프로세서 구조

옴니플로우 프로세서는 그림 2와 같이 5개의 주요 블록인 물리 인터페이스 블럭(FHP, PHY_Rx, ESch, PHY_Tx), 스위치 인터페이스 블록(ISch, SF_Rx, SF_Tx), 데이터 브리지, 컨텍스트 스케줄러(ConSch), 그리고 프로세서 어레이(PA)로 구성된다.

데이터 브리지는 물리 인터페이스 블록 또는 스

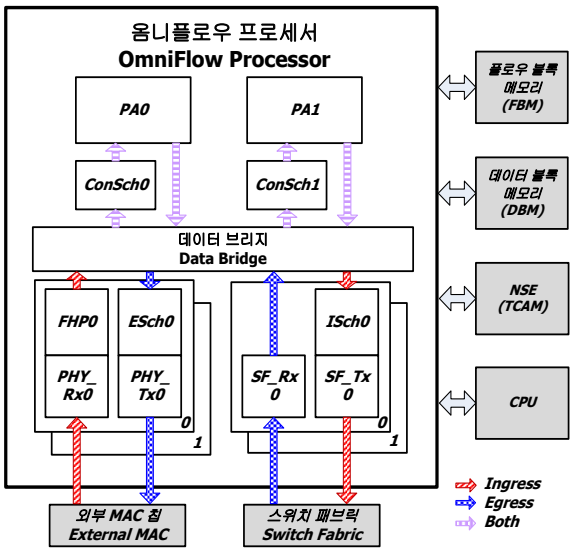


그림 2. 옴니플로우 프로세서 구조
Fig. 2. The structure of OmniFlow Processor.

위치 인터페이스 블럭을 PA0 또는 PA1에 선택적으로 연결시키는 역할을 한다. 입력 패킷은 데이터 경로에 따라서 인그레스(Ingress) 방향과 이그레스(Egress)방향으로 구분되어 처리되는데 일반적으로 외부 MAC 칩에서 스위치 패브릭 방향을 인그레스, 그리고 스위치 패브릭에서 외부 MAC 칩 방향을 이그레스라고 한다.

인그레스 경로에서 외부 MAC 소자로부터 SPI4.2 인터페이스를 통해 패킷을 수신한 PHY_Rx 블록은 입력 패킷을 48바이트 단위로 나누어서, 첫 번째 부분 패킷(세그먼트)을 제외한 나머지 부분을 데이터 블록 메모리에 저장한다. 4개의 RISC 코어로 구성된 가변 헤더 분석기(FHP)는 마이크로코드를 사용하여 세그먼트에서 해쉬키(48바이트)와 해쉬값(22비트)을 생성한다. 콘텍스트 스케줄러(ConSch)는 앞 절인 2.1에서 설명한 병렬 처리 방식대로 해쉬키와 해쉬값을 이용하여 플로우를 식별하고 패킷이 처리될 프로세서를 결정한 후 해당 프로세서로 세그먼트를 전송하는 역할을 수행한다. PA는 16개의 RISC 코어로 구성된 멀티스레드 패킷 프로세서로서 각 프로세서는 2개의 콘텍스트(스레드)로 구성되며, 2,048개의 명령어를 저장할 수 있는 내부 메모리를 가지고 있다. 모든 멀티프로세서는 플로우 상태 정보를 저장하기 위한 플로우 블록 메모리와 패킷을 저장하는 데이터 블록 메모리, 그리고 룩업 테이블을 저장하는 NSE 등의 외부 메모리를 직접적으로 액세스할 수 있다. PA의 주요 역할 중 하나는 QoS 요구 조건을 만족하도록 패킷을 스케줄링

하는 것이다. 옴니플로우 프로세서는 캘린더 기반 스케줄링 메커니즘[6]을 사용하기 때문에 각 패킷은 정해진 타임 슬롯에 맞춰서 출력되며, 패킷의 출력 시간($TTS$)은 식 (2)와 같이 결정된다.

$$TTS(us) = \max\left[\left\{T_{prev} + \frac{num\_pkt\_bytes \times C}{Service_{rate}}\right\}, T_{current}\right] \quad (2)$$

마지막 패킷이 출력된 시간($T_{prev}$)를 기준으로 패킷 바이트 수($num\_pkt\_bytes$)와 해당 서비스가 서비스 되어야 하는 트래픽 속도($Service_{rate}$)을 반영하여 현재 패킷의 출력 예정 시간이 결정된다. 이 때 계산된 출력 예정 시간이 현재 시간($T_{current}$)보다 적을 때는 현재 시간이 출력 시간이 된다. 트래픽 속도에 서비스 등급별 가중치가 별도 적용되며, 상수 $C$는 $msec$ 단위 계산에 필요한 상수 값으로써 $2^{13}$ ($\simeq 8{,}000$)로 정의한다. 일례로, 식 (2)에서 $Service_{rate}$가 64kb/s, 패킷사이즈가 300바이트이며, $T_{prev}$는 0일 때 서비스 예상 출력 시간은 37,500 us 로 결정되므로 이 경우에는 매 37.5 ms마다 패킷이 출력되어야 한다.

PA에서 출력된 세그먼트는 인그레스 스케줄러(ISch)로 전송된다. 스케줄러는 데이터 블록 메모리의 스케줄러 테이블인 캘린더 큐 내에 포인터를 저장하고, 출력 시간에 맞춰 메모리에 있는 패킷을 읽어서 스위치 인터페이스(SF_Tx)를 통해 스위치로 출력한다.

이그레스 경로는 인그레스의 반대 경로로서, 스위치 인터페이스(SF_Rx)를 통해 외부 스위치 패브릭으로부터 이그레스 패킷을 수신하면 데이터 블록 메모리에 첫 번째 패킷 부분을 제외한 나머지 패킷을 저장한다. PA는 패킷을 처리한 후에 스케줄링 정보와 함께 이그레스 스케줄러(ESch)로 패킷을 전송한다. 인그레스 스케줄러 처럼 이그레스 스케줄러는 출력시간과 큐 넘버 등과 같은 스케줄링 정보를 이용하여 패킷을 스케줄링한 후 물리 인터페이스(PHY_Tx)를 통해 외부 MAC 소자로 출력한다.

### 2.3. 플로우 상태 관리

옴니플로우 프로세서는 실시간 속도의 플로우 연결 설정이 가능하다. 그림 3은 연결 설정 및 패킷 포워딩 절차를 보여준다. 옴니플로우 프로세서는 해쉬키와 해쉬값을 플로우 식별자로 사용한다. 해쉬키

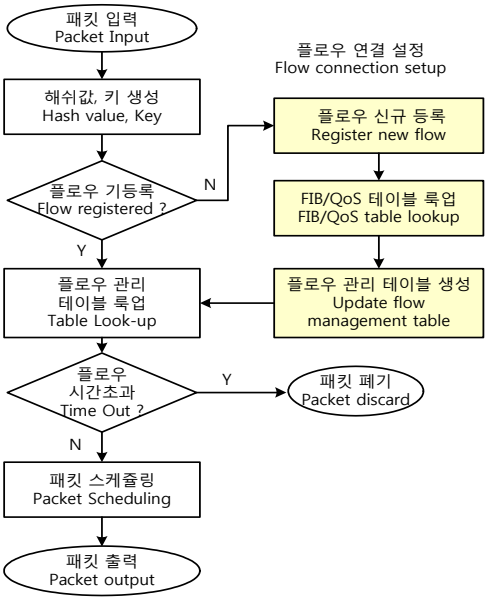


그림 3. 플로우 연결 설정 과정 및 관리
Fig. 3. Flow connection setup and management.

는 패킷의 헤더 또는 사용자에 의해 정해진 패킷의 특정 부분에서 추출된다. PA는 해쉬키를 주소로 사용하는 플로우 상태 테이블에 플로우를 등록하며, 입력된 모든 플로우의 등록 여부를 확인한다. 플로우가 등록되어 있지 않다면 플로우 연결 설정 절차를 통해 신규 플로우로 등록된다. 새로운 플로우 정보를 생성하기 위하여 포워딩 정보 베이스(FIB)와 QoS 테이블 등을 각각 룩업하여, 플로우에 해당되는 정보를 모아서 플로우 관리 테이블에 저장한다. 플로우 관리 테이블은 플로우의 목적지 포트 정보와 플로우 스케줄링 정보를 저장하고 있으므로 플로우 상태 테이블에 등록된 플로우는 플로우 관리 테이블 만을 룩업하여 패킷의 포워딩 포트를 결정하고 플로우에 적합한 스케줄링 규칙에 따른 관리가 가능하다. 또한 각 플로우의 관리 테이블에는 플로우 유지 시간(lifetime)이 기록된다. 각 플로우의 유지 시간은 각각 다르게 설정될 수 있으며 유지 시간 동안 해당 플로우의 패킷이 수신되지 않으면 해당 플로우의 패킷은 폐기되며, 플로우 상태 및 관리 테이블에서 삭제된다. 플로우 유지 시간을 짧게 설정하면 실제 서비스 되는 패킷의 플로우를 실시간적으로 감시할 수 있으므로 DDoS 공격을 효과적으로 방지할 수 있는 보안 시스템 및 플로우의 QoS를 보장하는 빠른 FAC에 적용될 수 있다[7]. 포워딩 정보 베이스, QoS 그리고 플로우 상태 및 관리 테이블은 모두 1G DDR2 SDRAM으로 구성된 플로우 블록 메모리에 포함되며, 이 메모리는 32개

의 멀티프로세서에 의해 액세스가 가능하다. 플로우 상태 테이블은 최대 600만 플로우 엔트리를 관리할 수 있으며, 각 엔트리는 해쉬 정보, 플로우 유지시간, 통계, 그리고 플로우 관리 테이블의 엔트리 해당 포인터 등을 포함한 64바이트 플로우 데이터로 구성된다.

### 2.4. 플로우 수락 제어

연결 수락 제어(CAC)를 지원하는 일반적인 라우터는 RSVP(서비스 예약 프로토콜), MPLS-LSP 등의 프로토콜을 사용하여 미리 서비스를 요구하는 플로우에 대해 경로를 설정하고 대역을 예약하는 방식을 사용하여, 서비스 등급 협약(SLA)을 기준으로 새로운 플로우를 수락할 것인지 혹은 거절할 것인지를 결정한다. 따라서 채널 이용률보다는 셀 손실율, 평균 지연 시간 등을 향상 시키는데 주목적이 있다. 이런 대역 예약 방식은 플로우의 업데이트 주기가 매우 길며, 관리할 수 있는 플로우 수가 많지 않다. SLA에는 서비스 등급, 협약된 플로우 대역 등의 플로우 정보가 포함된다. 일반적인 라우터의 연결 수락 제어 방식은 네트워크상에서 측정되는 실제 트래픽을 고려하지 않고, 미리 알고 있는 새로운 플로우의 최대 셀전송율(PCR: Peak Cell Rate) 대역을 이미 수락된 대역에 더한 값이 가용한 대역의 기준치를 넘게 되면 플로우는 거절된다. 이 때는 새로운 플로우의 실제 트래픽 양이 매우 적은 경우라 하더라도 거절되는 상황이 발생할 수 있다.

반면에 옴니플로우 프로세서를 사용한 플로우 라우터는 실제 트래픽을 고려한 플로우별 연결 수락 제어(FAC)를 지원한다. 그러므로 수락된 플로우의 평균 입력 속도의 총 합이 플로우 라우터의 가용 대역의 문턱값을 초과한 경우에 새로운 플로우를 허용하지 않는다. 또한 폭주가 발생하더라도 플로우 라우터의 수율을 조절하기 위하여 SLA 협약에 의해서 특정 플로우의 패킷만을 선택적으로 폐기할 수 있다. 옴니플로우 프로세서는 하드웨어 기반으로 플로우를 식별, 제어 및 관리할 수 있는 플로우 프로세서로서 실시간적으로 매 짧은 주기(예) 1, 2, 3 sec)마다 최대 600만 플로우의 등록과 삭제가 가능하다. 따라서 현재 활동 플로우의 실 가용 채널량을 모니터링하여 FAC를 수행한다.

즉, 옴니플로우 프로세서는 식 (3)과 같은 조건을 만족할 때 새로운 플로우를 수락한다. 식 (3)에서 전체 링크 용량은 $C$ b/s, $\mu$는 수락 결정 문턱 경계값 ($0 < \mu \leq 1$), 플로우 $m$의 평균 속도는 $r_m$ b/s, 새로운 플로우는 $k$, 그리고 이미 수락된 플로우 수를 $p$라 할 때, 현재의 트래픽 부하와 새로운 플로우 속도의 합이 전체 링크 용량의 문턱 경계값보다 크면 새로운 플로우가 거절된다. 이 때 측정된 현재의 트래픽 부하는 수락된 플로우 속도 총 합이 된다.

$$\sum_{m=0}^{p-1} r_m + r_k \leq \mu C \qquad (3)$$

플로우 수락율을 높이기 위해서는 빠르고 정확한 트래픽 측정과 사용되지 않는 불필요한 플로우는 신속하게 삭제되어야 한다. 왜냐하면 플로우 업데이트 주기가 길면 문턱 경계 값을 순간적으로 초과한 경우라 하더라도 일정한 시간동안 플로우를 수락하지 못하므로 성능 저하의 원인이 되기 때문이다.

## Ⅲ. 플로우 수락 제어 모의 실험

본 논문에서는 2.4에서 설명한 CAC 기능을 지원하는 기존 라우터(Legacy 모델)와 플로우 라우터(FAC 모델)를 비교하여 CAC 관점에서 링크 대역 이용률의 성능을 평가한다. Legacy 모델은 실제 트래픽을 고려하지 않고 미리 협약된 대역을 기준으로 새로운 플로우의 수락 및 거절을 결정한다. 따라서 시뮬레이션에서 사용되는 Legacy 모델은 Diffserv 방식의 트래픽 제어 방식을 지원하기 위하여 MPLS의 LSP(Label Switched Path)를 설정하기 위해 최대 셀 전송율(PCR) 기반의 CAC를 수행한다고 가정한다[12]. 그림 4는 시뮬레이션을 위한 시스템 구성을 나타낸다. 두 시스템은 동일한 조건의 시뮬레이션 환경 및 파라미터를 사용한다.

시스템 당 최대 플로우 수는 32개, 각 플로우 당 버퍼 사이즈는 4Kx256바이트, 그리고 256 바이트의 고정 크기의 패킷을 사용한다. 패킷 도착 시간은

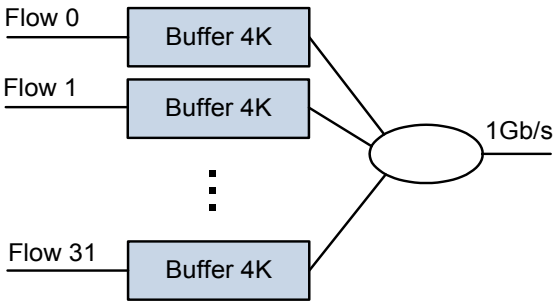


그림 4. 시뮬레이션을 위한 시스템 구성
Fig. 4. System Configuration for simulation.

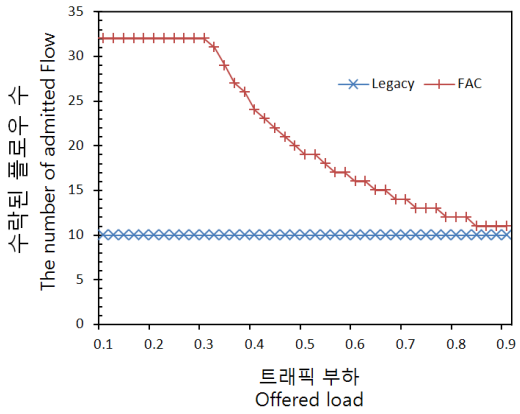


그림 5. 트래픽 부하에 따른 수락된 플로우의 수
Fig. 5. The number of admitted flows.

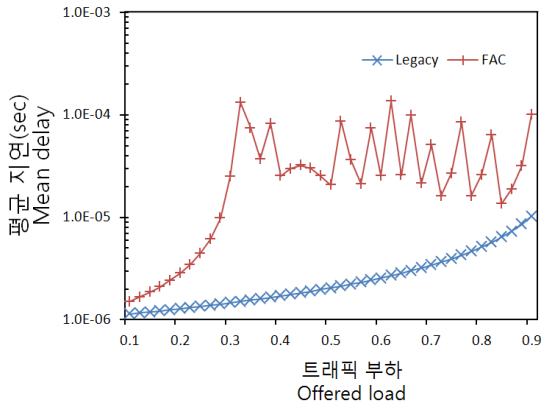


그림 6. 패킷 평균 지연 시간
Fig. 6. Mean packet delay.

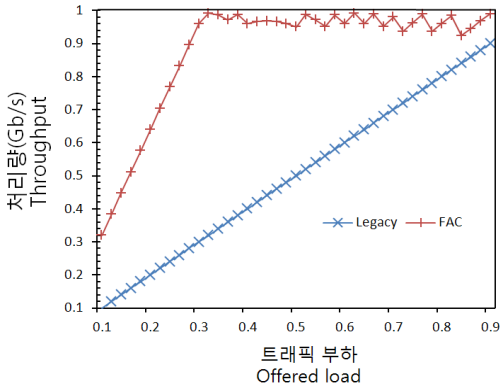


그림 7. 패킷 처리량
Fig. 7. Packet throughput.

Poisson 도착 프로세스 분포를 따른다. 가용한 링크 대역은 1Gb/s 이며 각 플로우는 최대 100Mb/s까지 가변된다. 각 플로우의 입력 속도를 최대 속도인 100Mb/s의 10%에서 90%까지 증가시키며, 패킷 처리량(링크 이용율), 평균 지연, 그리고 수락된 플로우 수를 측정한다. 수락된 플로우에 대한 패킷 손실은 없다고 가정하며, C++ 언어를 사용하여 시뮬레이션을 실행하였다.

그림 5는 인가된 트래픽 부하의 증가에 따른 수락된 플로우 수의 관계를 나타내었다. 그림 5에서 Legacy 모델은 10개의 플로우에 대한 협약된 총 대역이 최대 출력 속도와 동일하므로 트래픽 상황에 상관없이 10개의 플로우만을 수락한다. 그러므로 실제 트래픽 부하가 낮다 하더라도 더 많은 플로우를 수락하지 않는다.

FAC 모델에서 인가된 트래픽 부하가 약 0.3이 되었을 때 트래픽 폭주로 인해 패킷 손실이 발생할 수 있다. 따라서 서비스되는 플로우의 패킷 손실을 방지하기 위하여 수락된 플로우의 수를 제한하므로 수락된 플로우 수는 부하가 증가할수록 단계적으로 감소한다.

그림 6은 라우터에 입력된 패킷이 출력될 때 까지 걸린 평균 지연시간을 나타낸다. FAC 모델의 지연 시간은 부하가 0.3 이후에 증가와 감소를 반복하는데 그 이유는 수락된 플로우 수가 하나씩 감소할 때 마다 지연 시간이 감소하기 때문에 발생한다. 수락된 플로우 수가 하나씩 감소할 때 마다 폭주 상태는 일시적으로 해소되며 트래픽이 점점 증가하여 폭주 상태가 다시 발생하기 전까지 패킷은 빠르게 처리된다.

그림 7에서 두 시스템의 패킷 처리량은 수락된 플로우의 차이에 따라 결정되므로 FAC 모델이 우수함을 보여준다. Legacy 모델에서는 트래픽이 증가함에 따라서 비례적으로 처리량도 증가하지만 플로우 당 트래픽 부하가 낮을 때 패킷 처리량은 높지 않다. 반면에 FAC 모델의 패킷 처리량은 1Gb/s까지 빠르게 증가하며, 트래픽 부하가 0.3 이상부터 수락된 플로우 수가 하나씩 감소할 때마다 처리량이 일시적으로 감소하기 때문에 처리량은 증가와 감소를 반복한다. 본 시뮬레이션의 결과인 그림 5~7 결과를 볼 때 동일한 QoS를 제공할 때 FAC 모델이 Legacy 모델 보다 수락된 플로우 수가 많으며, 이에 따라 처리량 또한 우수함을 알 수 있다.

본 실험은 실시간 평균값에 따라 플로우 수락이 이루어지므로, 순간적으로 트래픽이 증가하면 수락된 플로우에 대해서도 트래픽 혼잡이 발생할 수 있다. 이 경우에는 신규 플로우의 수락을 차단하거나 특정한 플로우를 선별적으로 폐기할 수 있으므로 손실되는 패킷의 플로우 수를 최소화 할 수 있다. 또한 패킷의 손실을 막기 위하여서는 채널 이용률

이 낮아지더라도 순간적인 트래픽 증가를 고려하여 수락 문턱값($\mu$)을 낮출 수 있다. 플로우 라우터는 연결 설정이 매우 빠르게 진행되므로 짧은 플로우 유지 시간을 갖는 플로우인 경우에는 패킷 손실이 발생하지 않고 수락 플로우 수를 늘릴 수 있으므로 패킷 처리량을 증가시킬 수 있다.

## Ⅳ. 옴니플로우 프로세서 칩 구현

옴니플로우 프로세서는 Verilog HDL로 설계되었다. 논리 합성과 배치 및 라우팅 작업을 거친 후 DFT용 회로를 삽입하고 TSMC 사의 65 nm 공정에 의해 칩으로 제작 되었다. 옴니플로우 프로세서에서 사용된 TCAM, PLL, 메모리, 입출력 셀 등은 별도로 제작된 라이브러리이며, 일부 블록은 보안 설계를 위해 하드코어 매크로가 반영되었다. 표 1은 옴니플로우 프로세서의 칩의 주요 특징을 나타내며, 그림 8은 제작된 옴니플로우 프로세서를 주요 블록별 라이브러리로 표시한 칩 전체 배치 도면이다.

표 1. 옴니플로우 프로세서 칩의 특징
Table 1. Hardware Specification of OmniFlow Processor

| 구분<br>specification | 옴니플로우 프로세서<br>OmniFlow Processor |
|---|---|
| 사용자 인터페이스<br>User Interface | SPI4.2 x 2 |
| PA | 64bits RISC x 32 |
| FHP | 128bits RISC x 8 |
| Data Block Memory | 2G Bytes |
| Flow Memory | 1G Bytes |
| Operating Frequency | 555MHz |
| 용량<br>Logic Capacity | 25M gates |
| | 17.6Mbits memory |
| 신호수<br>the number of signals | 1,518 |
| Package type | 50x50 $mm^2$<br>2,401HFCBGA |
| Die size | 17x17 $mm^2$ |
| Power consumption | 40W |
| 공정 Manufactory | TSMC 65nm CMOS |

## Ⅴ. 결 론

본 논문에서는 실시간으로 연결을 설정할 수 있는 FAC 방법 및 이를 하드웨어로 구현한 옴니플로우 프로세서를 제안하였다. 각 사용자 별로 설정한

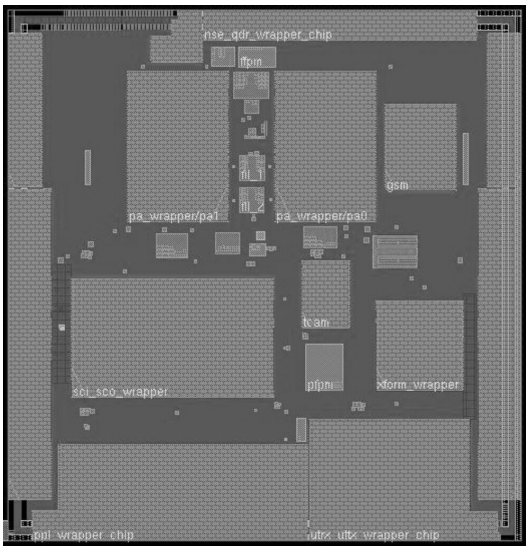


그림 8. 옴니플로우 프로세서 칩 배치도
Fig. 8. The Layout of OmniFlow processor.

특정 시간 동안 전송된 패킷이 없는 플로우의 연결을 해지 할 수 있기 때문에 옴니플로우 프로세서는 짧은 시간 동안 네트워크에서 사용되는 플로우를 효과적으로 관리할 수 있다. 이와 같은 실시간 플로우 관리 기능은 폭주 상황에서 UDP/TCP 플로우에 대한 전송 신뢰성을 보장하고 보안 및 미래 네트워크 서비스를 제공하는데 유용할 것으로 생각된다.

옴니플로우 프로세서는 65nm CMOS 기술로 제작되었으며, 40개의 멀티 코어 프로세서(32개의 패킷 처리 코어와 8개의 패킷헤더 분석 코어)를 내장하고 있으므로 555MHz 동작 주파수로 최대 40 Gb/s의 처리 성능을 나타낸다. 특히 내부 브리지를 통해 각 기능 블록을 연결하는 구조로 설계되어 확장성이 뛰어나므로 쉽게 성능을 향상 시킬 수 있다. 시뮬레이션을 통해 옴니플로우를 사용한 플로우 라우터가 기존 시스템 보다 높은 처리량과 낮은 패킷 손실율을 나타냄을 보였다. 앞으로 옴니플로우 프로세서가 새로운 미래 네트워크 서비스를 창출하는데 일조할 것으로 기대한다.

## 참 고 문 헌

[1] N. Brownlee, C. Mills, and G. Ruth, Traffic Flow Measurement: Architecture, *IETF* RFC 2722, Oct 1999.

[2] J. Rajahalme, A. Conta, B. Carpenter, et al., IPv6 Flow Label Specification, *IETF* RFC 3697, Mar. 2004.

[3] A.Kumar, M. Hegde, S.V.R. Anand et al., “Nonintrusive TCP Connection Admission Control for Bandwidth Management of an Internet Access Link,” *IEEE Comm. Magazine*, May 2000.

[4] R. Mortier, I. Pratt, C. Clark et al., “Implicit Admission Control,” *IEEE Journal on Selected Areas in Comm.*, Dec. 2000.

[5] J.W. Roberts and S. Oueslati-Boulahia, “Quality of Service by Flow Aware Networking,” *Phil. Trans. Royal Soc. London*, 2000.

[6] N.S. Ko, S.B. Hong, K.H Lee et al., "Quality-of-Service Mechanisms for Flow-Based Routers," *ETRI Journal*, vol. 30, no. 2, Apr. 2008, pp.183-193.

[7] HeeKyong Yi, PyungKoo Park, Seungwook Min et al., "DDoS Detection Algorithm Using the Bidirectional Session," *CN2011*, vol. 160, Jun. 2011, pp.191-203.

[8] G. M. Amdahl, "Validity of the single-processor approach to achieving large scale computing capabilities," *ACM Press*, vol. 30, 1967, pp483-485

[9] S. Govind, R.Govindarajan, and Joy Kuri, “Packet Reordering in Network Processors,” *Parallel and Distributed Proc. Symposium (IPDPS 2007)*, IEEE International, Mar. 2007.

[10] “The Cisco QuantumFlow Processor: Cisco’s Next Generation Network Processor,” *Cisco Syst.*, 2008

[11] V. Paxson, "End-to-end Internet packet dynamics," *IEEE/ACM Trans. Networking*, vol. 7, no. 3, Jun. 1999, pp.277-292.

[12] “Junos OS MPLS Applications Configuration Guide R12.1,” *Juniper Networks*, Mar. 2012

[13] J.W. Park, W.Y.Chung, etc., “Study of Parallel Network Processor using Global Cache,” KICS, vol.36, no.1, Jan. 2011.

**두 경 환** (Kyeong-Hwan Doo)

1996년 2월 전북대학교 전자공학과 학사
1998년 2월 전북대학교 전자공학과 석사
2007년 3월~현재 충남대학교 전기정보통신공학부 박사과정
2000년~현재 한국전자통신연구원 선임연구원
<관심분야> 플로우 프로세서, 초고속 디지털 통신, 광가입자망 기술

**이 범 철** (Bhum-Cheol Lee)

1981년 2월 경희대학교 전자공학과 학사
1983년 2월 연세대학교 전자공학과 석사
1997년 8월 연세대학교 전자공학과 박사
1983년~현재 한국전자통신연구원 차세대통신연구부문 개방형스위치연구팀장
<관심분야> 네트워크 가상화, 플로우 프로세서, SDN

**김 환 우** (Whan-Woo Kim)

1977년 2월 서울대학교 전자공학과 학사
1979년 2월 한국과학기술원 전기 및 전자공학과 석사
1988년 6월 University Of Utah 전자공학과 박사
1980년 6월~현재 충남대학교 전기정보통신공학부 교수
<관심분야> 디지털 신호처리, 초고속 디지털 통신