논문 2010-47TC-7-17

# 미래인터넷 OpenFlow 테스트베드 구축을 위한 NetFPGA기반 캡슐레이터 구현 및 성능평가

## ( NetFPGA based capsulator Implementation and its performance evaluation for Future Internet OpenFlow Testbed )

최 윤 철*, 민 석 홍*, 김 병 철**, 이 재 용**, 김 대 영**

( Yun Chul Choi, Seok-Hong Min, Byung Chul Kim, Jae Yong Lee, and Dae Young Kim )

### 요 약

패킷 기반의 TCP/IP 프로토콜 기반으로 구축된 인터넷 환경은 30년 넘게 사용되어 왔으나, 향후 통신환경의 급격한 변화와 다양한 사용자 요구사항의 증대로 인해 프로토콜 확장의 제약으로 인한 근본적인 문제점을 나타나게 될 것이다. 이를 해결하기 위해 Clean Slate 설계 방법에 기반을 둔 미래인터넷 연구가 진행되고 있고, 이를 실험하고 검증하기 위한 대규모 테스트베드 구축이 이루어질 것이다. 이를 위한 오픈 플로우 스위치 기술은 기존에 포설된 네트워크 장비를 활용하면서, 인터넷 트래픽에 영향을 주지 않고 독립적인 프로토콜을 시험할 수 있도록 하는 새로운 기술로 제안되었다. 국내에서도 테스트베드 구축의 한 방법으로 NetFPGA기반 오픈 플로우 스위치를 활용한 망구성이 연구되고 있으며 이러한 구조에서 인터넷망을 통한 오픈 플로우 스위치 간 연결이 이루어지기 위해서는 오픈 플로우 스위치 간 논리적인 터널링이 제공되어야 한다. 이에 대한 해결책으로 본 논문에서는 NetFPGA 기반의 캡슐레이터를 구현하여 국내연구망인 KOREN에 구현된 여러 오픈 플로우 사이트 간에 MAC in IP 터널링이 이루어 질 수 있도록 하였고 이의 성능을 측정하였다. 측정 결과 기존 소프트웨어 기반의 캡슐레이터에 비해 성능이 향상되었음을 확인하였고, 미래인터넷 기술을 실험할 수 있는 테스트 베드로 활용할 수 있음을 보였다.

### Abstract

Current TCP/IP-based Internet architecture has been used for over 30 years, however it will confront with fundamental problems due to new protocol extension limitation since communication environments will change drastically and various user requirements will be emerging in near future. To solve these problems, major countries have started Future Internet researches based on clean slate approach and they will deploy large-scale testbed to experiment and verify new functions. OpenFlow switch technology has been proposed as a new experimental technology for independent protocol that can utilized the legacy network devices and does not interfere with the production Internet traffic. Korea also started Future Internet testbed project called FIRST and OpenFlow switch with NetFPGA card will be used to deploy this testbed. To interconnect distributed testbed using OpenFlow switches, logical tunnel should be established by encapsulating MAC frame inside a unicast IP packet between OpenFlow switches because OpenFlow switches are not directly connected. In this paper, we have implemented a NetFPGA-based that performs MAC in IP tunneling between various OpenFlow switch sites implemented in domestic research network KOREN. The performance evaluation shows that the NetFPGA-based capsulator reveals better performance than the software-based tunneling and it can be utilized as a testbed for experimentation of Future Internet technologies.

**Keywords**: Future Internet, Testbed, OpenFlow, NetFPGA, Capsulator

* 학생회원, ** 평생회원, 충남대학교 정보통신공학과(Chungnam National University)

※ 본 논문은 "미래인터넷 인프라를 위한 가상화 지원 프로그래머블 플랫폼 및 핵심원천 기술개발" 과제 (2009-F-050-01)에 대한 결과물 중 일부분으로 연구개발 업무에 도움을 주신 분께 감사드립니다.
접수일자: 2010년1월11일, 수정완료일: 2010년7월14일

## Ⅰ. 서 론

오늘날의 인터넷 개념은 미 국방성의 ARPA (Advanced Research Projects Agency)에서 전술적인 개념의 컴퓨터 네트워크인 ARPAnet을 개발한 것이 시초가 되었다. 이 네트워크는 네트워크 일부분의 오동작이 생겨도 우회경로를 통하여 지속적인 통신이 가능하게 하며 이기종의 컴퓨터, 혹은 기기들이 상호 접속할 수 있는 매우 획기적인 네트워크였다. 이후 1974년 인터넷의 개념이 정립되고, 1978년에는 IP(Internet Protocol), 80년대 이후 BGP(Border Gateway Protocol) 등의 각종 라우팅 프로토콜이 개발되면서 기업과 개인의 비즈니스 및 일상생활 에서 사용 가능한 인터넷의 모습으로 발전하게 되었다[1].

그러나 최근에는 다양한 사용자들의 요구 사항의 증대로 기존의 이메일, 웹 서핑, 파일 전송 같은 전송 프로그램 들 뿐만이 아니라 P2P(Peer to Peer)기반의 파일 공유 프로그램, IPTV, VoIP 등과 같은 새로운 응용 프로그램들도 등장하게 되었다. 이러한 응용 프로그램들 외에도 전자 상거래, 전자정부, 원격교육 등을 위한 서비스들도 속속 개발되면서 인터넷 환경이 급속도로 바뀌고 서비스도 제공자 중심에서 사용자 중심으로 변화되었다. 이러한 변화를 수용하기 위해서는 기존 인터넷 환경의 확장성, 보완성, 이동성을 보완하는 네트워크 구조와 프로토콜이 필요하게 되었다. 이를 위해 기존의 네트워크 환경을 유지하면서 변화를 수용하는 것은 네트워크 복잡도를 증가시키고 새로운 프로토콜의 보급을 더디게 하였다. 이러한 이유로 실험자가 자유롭게 프로그래밍 가능하고 확장성과 유연성을 지닌 새로운 구조의 네트워크와 프로토콜에 대한 연구가 진행되게 되었고, 이를 실험하고 검증하기 위해서는 전국적인 테스트베드가 필요하게 되었다.

테스트베드 구축을 위해 제안된 기술 중에는 인터넷 트래픽에 영향을 주지 않고 독립적인 프로토콜을 시험할 수 있도록 제안된 오픈 플로우[3] 스위치 기술과, 프로그램 가능한 하드웨어 로직으로 구성되어 네트워크 로직을 연구하고 설계하기에 적합한 NetFPGA 플랫폼[4]등이 있다. 국내에서는 이를 활용한 PC기반의 전국적인 단일 망 테스트베드 구축이 FIRST@PC 프로젝트[2]라는 이름으로 진행되고 있는데, 이를 위해서는 인터넷 망을 통한 로컬 테스트베드 간 연동이 요구된다. 이때 로컬 테스트베드에서 나오는 MAC 프레임 전체를 인터넷 망을 통하여 터널링 하기위해서는 NetFPGA기반의 오픈 플로우 스위치 간 트래픽의 전달이 필요하다. 즉 오픈 플로우 스위치를 통해 연결된 호스트들은 동일한 망 주소를 사용해 특정 스위치 포트에 연결된 것처럼 동작해야 하므로 인터넷 망을 통해 연결된 오픈 플로우 스위치들 간에 로컬 트래픽을 encapsulation 하여 전달하는 것이 필요하다. 본 논문에서는 이를 위해 하드웨어 기반의 NetFPGA플랫폼에 MAC in IP 캡슐레이터를 구현하였으면 이것이 소프트웨어 기반의 캡슐레이터에 비해 성능이 향상되었음을 확인하였다.

본 논문의 Ⅱ장에서는 미래 인터넷 로컬 테스트베드 구축을 위한 플로우 기반의 오픈 플로우 프로토콜[3]과 프로그램 가능한 NetFPGA[4]플랫폼을 소개하고, 이를 활용한 NetFPGA기반 오픈 플로우 스위치[5]에 대해서 기술하였다. Ⅲ장에는 로컬 테스트베드 간 연동을 위한 MAC in IP 캡슐레이터[6]를 하드웨어로 설계하고 구현하여 기존 소프트웨어 방식의 캡슐레이터에 비해 성능이 향상되었음을 확인하였다. 마지막으로 Ⅳ장에서는 결론과 향후 연구 계획을 기술함으로 본 논문을 마무리 한다.

## Ⅱ. 관련 연구

### 1. 오픈 플로우 프로토콜

오픈 플로우[3]는 스위치와 라우터에 플로우기반 사용자 테이블을 생성하고 이들 간 통신에 필요한 표준화된 프로토콜을 제공하여 사용자가 원하는 형태의 프로그램을 작성 가능하도록 도와준다. 이를 활용하면 네트워크 관리자에게는 연구용 플로우와 일반 플로우를 분리하여 관리할 수 있게 도와주고, 개발자에게는 새로운 라우팅 프로토콜, 보완 모델, 새로운 주소 방식 같은 자신의 테

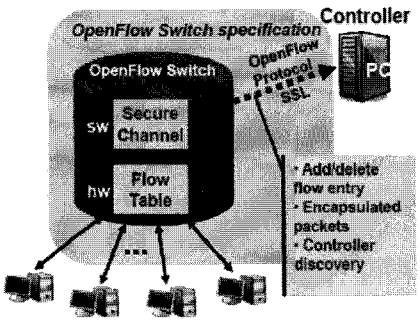



그림 1. 오픈 플로우 스위치 구성요소[7]
Fig. 1. OpenFlow switch components.[7]

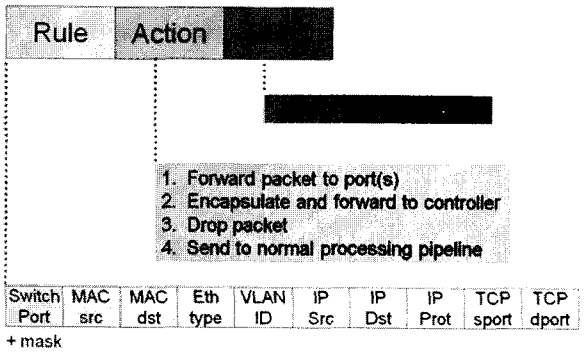



그림 2. 오픈 플로우 스위치 플로우 테이블[8]
Fig. 2. OpenFlow Switch FlowTable[8].

스트용 플로우를 제어할 수 있게 지원해준다.

오픈 플로우 스위치는 그림 1과 같이 플로우 테이블, 보안 채널, 오픈 플로우 프로토콜 등으로 구성 되어 있다. 플로우 테이블에는 각각의 플로우들이 어디에 속해 있는지, 어떻게 처리할 지가 나타나 있다. 보안 채널을 통하여서 외부에 있는 컨트롤러와 연결을 맺고 플로우에 대한 실행명령을 주고받는다. 오픈 플로우 프로토콜은 컨트롤러와 스위치가 통신하는 표준화된 프로토콜이다.

플로우 테이블은 그림 2와 같이 규칙, 실행명령, 상태정보 3가지로 구성 되고, 패킷의 헤더 정보 중에서 10-tuple(switch 포트, VLAN id, MAC, IP, TCP)을 이용하여 플로우를 구분하며, 실행명령에 따라서 패킷을 처리한다. 다음 4가지는 오픈 플로우 스위치에서 제공하는 실행명령이다.

- 특정 포트로 플로우 패킷을 전달한다.
- 패킷을 encapsulation 해서 컨트롤러에 전달한다.
- 문제를 발생시키는 특정 플로우 패킷은 버린다.
- 상용 망과의 연동을 위해 일반 패킷은 통과시킨다.
-

오픈 플로우 스위치와 컨트롤러가 주고받는 프로토콜의 메시지 타입은 그림 3과 같다. 오픈 플로우와 컨트롤러가 처음 연결 되면 Hello 메시지와 Features_equest, Features_reply 메시지를 주고받는다. 이를 통하여 스위치 정보 교환이 이루어지고, 이후에 새로운 플로우 패킷이 오픈 플로우 스위치에 도착하면 Packet_in 메시지가 컨트롤러에 전달되고, 그에 따른 실행명령이 Packet_out 메시지를 통하여 오픈 플로우 스위치에 전달된다. 플로우에 대한 정보가 수정 된 경우

```
enum ofp_type {
    /* Immutable messages. */
    OFPT_HELLO,                 /* Symmetric message */
    OFPT_ERROR,                 /* Symmetric message */
    OFPT_ECHO_REQUEST,          /* Symmetric message */
    OFPT_ECHO_REPLY,            /* Symmetric message */
    OFPT_VENDOR,                /* Symmetric message */

    /* Switch configuration messages. */
    OFPT_FEATURES_REQUEST,      /* Controller/switch message */
    OFPT_FEATURES_REPLY,        /* Controller/switch message */
    OFPT_GET_CONFIG_REQUEST,    /* Controller/switch message */
    OFPT_GET_CONFIG_REPLY,      /* Controller/switch message */
    OFPT_SET_CONFIG,            /* Controller/switch message */

    /* Asynchronous messages. */
    OFPT_PACKET_IN,             /* Async message */
    OFPT_FLOW_EXPIRED,          /* Async message */
    OFPT_PORT_STATUS,           /* Async message */

    /* Controller command messages. */
    OFPT_PACKET_OUT,            /* Controller/switch message */
    OFPT_FLOW_MOD,              /* Controller/switch message */
    OFPT_PORT_MOD,              /* Controller/switch message */

    /* Statistics messages. */
    OFPT_STATS_REQUEST,         /* Controller/switch message */
    OFPT_STATS_REPLY            /* Controller/switch message */
};
```

그림 3. 오픈 플로우 프로토콜 메시지 타입[9]
Fig. 3. Message type of OpenFlow protocol[9].

기존의 플로우 테이블을 변경하라는 Flow_mod 메시지를 전달한다.

## 2. NetFPGA 플랫폼

NetFPGA[4]는 프로그램 가능한 하드웨어 로직을 활용하여 네트워크 하드웨어 구조나 라우터 디자인을 배우고 연구하기 적합한 플랫폼이다. NetFPGA의 구성은 그림 4와 같이 사용자 정의 네트워크 로직에 사용되는 Xilinx Virtex-2 Pro FPGA(Field Programmable Gate Array)와 호스트 인터페이스와 PCI(Peripheral Component Interconnect)버스를 통하여 통신할 때 이용되는 Xilinx Spartan FPGA, output queue에 이용되는 4.5MB SRAM(Static RAM), 64MB DDR(Double

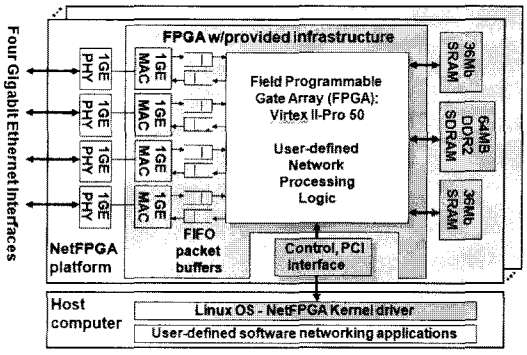



그림 4. NetFPGA 블록 다이어그램[10]
Fig. 4. NetFPGA Block diagram[10].

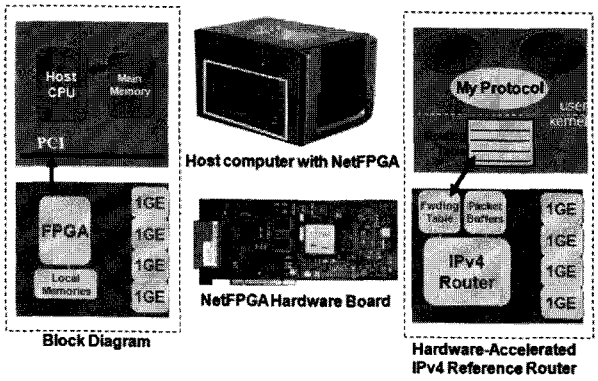



그림 5. NetFPGA 와 호스트 PC 간 인터페이스[4]
Fig. 5. Interface between NetFGPA and Host PC[4].

Date Rate)2 RAM, NetFPGA 병렬연결을 위한 두 개의 SATA(Serial ATA)포트, 외부 데이터 입출력을 위한 4개의 기가비트 이더넷 포트로 이루어져 있다.

표준 PC의 여분의 PCI 슬롯에 NetFPGA카드를 추가하고 NetFPGA드라이버와 관련 프로그램(Java, Perl, python, CAD tool)을 설치하면 NetFPGA 플랫폼이 완성된다. NetFPGA 플랫폼에서 NetFPGA 와 호스트 PC와의 통신 구조는 그림 5와 같이 기가비트 이더넷 포트에서 들어오는 패킷을 FPGA 모듈로 보내고 이를 처리하여 PCI 인터페이스를 통하여 호스트 CPU로 보내거나 4개의 기가비트 이더넷 포트로 내보낸다. 만약 그림 5와 같이 NetFPGA 플랫폼을 라우터로 활용하는 경우에는 라우팅 로직이나 사용자 레벨에서 작성된 라우팅 테이블을 이용하여 포워딩 테이블을 작성하고 이 테이블에 따라 기가비트 인터넷 포트로 들어오는 패킷을 내보낸다.

그림 6과 같은 NetFPGA 하드웨어 기본 구조에서 user data path 라는 영역을 수정하여 필요한 기능을 구현할 수 있다. 여기서 패킷의 전달과정을 살펴보면 패킷은 호스트와 연결되는 CPU RxQ(RxQueue)또는 이더넷 인터페이스와 연결되는 MAC(Media Access Controllers) RxQ에서 CPU TxQ(TxQueue), MAC TxQ로 전달된다. 그림 7과 같이 RxQ에서 생성된 모듈 헤더가 패킷 앞에 추가 되어 패킷 단위로 데이터를 전송한다. 그 후 input arbiter는 서비스 할 RxQ를 선택하여 ouput port lookup 모듈로 패킷을 전달한다. Ouput port lookup 모듈에서는 내부로직에 의해서 TxQ 포트를 선택하고, 모듈 헤더의 DST PORT ONE-HOT 부분에 기록하여, output queues 모듈로 패킷을 전달한다. Ouput queues 모듈에서는 모듈 헤더의

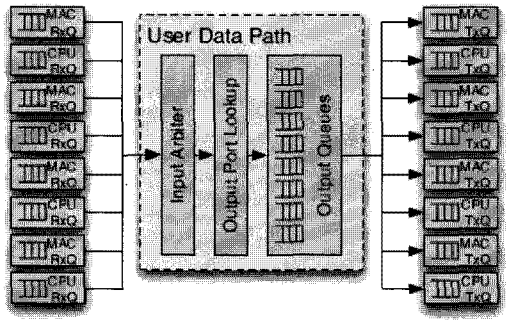



그림 6. NetFPGA 하드웨어 로직 구조[11]
Fig. 6. NetFPGA hardware logic architecture[11].

| CTRL BUS | DATA BUS | | | |
|---|---|---|---|---|
| Bits 7-0 | Bits 63-48 | Bits 47-32 | Bits 31-16 | Bits 15-0 |
| 0xFF | DST PORT ONE-HOT | WORD LENGTH | SRC PORT BINARY | BYTE LENGTH |

그림 7. NetFPGA user data path 모듈 헤더[4]
Fig. 7. NetFPGA user data path module header[4].

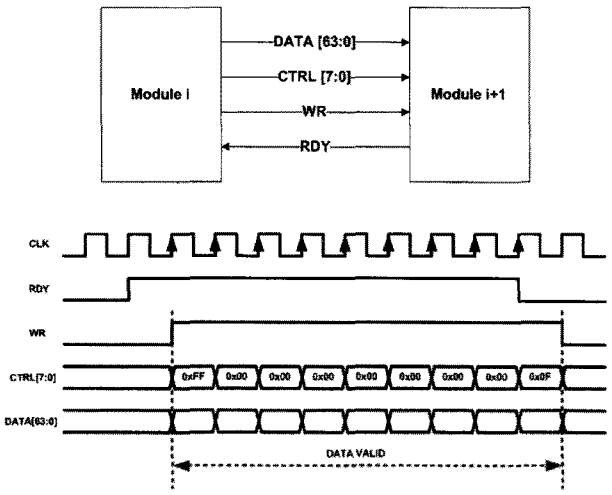



그림 8. NetFPGA 하드웨어 모듈 간 통신[4]
Fig. 8. Communication between NetFPGA hardware modules[4].

DST PORT ONE-HOT 부분을 보고 해당 포트의 TxQ 로 패킷을 전달한다.

NetFPGA 하드웨어 로직을 구성하는 모듈(RxQ, TxQ, user data path) 간 데이터 통신은 그림 8과 같이 125MHz 클럭을 포함하여 5가지(CLK, RDY, WR, CTRL[7:0], DATA[63:0]) 신호를 통하여 전달된다. Module i+1에서 데이터를 수신할 수 있으면, RDY 신호를 module i로 보내고, module i는 전송 할 데이터가 있으면 WR 신호와 함께 DATA 신호, CTRL 신호를 module i+1에 전송한다. DATA 신호에는 패킷이나 모듈 헤더를 담고 있고, CTRL 신호는 모듈 간 데이터 전

송 시 필요한 제어신호를 나타낸다. CTRL신호가 0xFF 이면 DATA 신호의 정보가 모듈 헤더임을 가리키고, 0x00은 패킷을 그리고 그 이외 값은 패킷이 끝나는 부분을 나타낸다. 여기서 데이터의 전송은 클럭이 올라가는 부분에서 이루어지고, 데이터 수신은 클럭이 내려가는 부분에서 이루어진다.

NetFPGA에서 데이터의 전달은 모듈 간 파이프라인을 통해서 이루어지는데 전달되는 데이터를 사용자 계층에서 손쉽게 제어하고, NetFPGA 관련 네트워크 값을 설정하거나 하드웨어 값을 디버깅하기 위해서는 사용자 인터페이스가 요구된다. 이를 위해서는 제어 가능한 하드웨어 레지스터, 카운터 등이 필요하다. 이러한 하드웨어 정보를 사용자 계층에 표시하거나, 수정하기 위해서는 하드웨어와 소프트웨어 간에 공통된 레지스터 인터페이스가 필요하다. 이러한 이유로 하드웨어 레지스터 주소와 소프트웨어 레지스터 주소 연결(mapping)이 필요하고, 이를 활용하면 레지스터 입출력에 대한 결과가 연결된 레지스터 저장 공간(memory-mapped registers)에 출력된다. NetFPGA에 사용 되는 레지스터 주소는 표 1과 같이 나누어진다.

레지스터 모듈 각각은 체인(chain) 형태로 결합되어 그림 9과 같이 링 구조를 이루고 있다. 임의의 모듈에

표 1. NetFPGA 레지스터 주소 영역[4]
Table 1. NetFGPA register address space[4].

| Block | Module | Size | Address Range |
|---|---|---|---|
| CPCI | | 4MB | 0x0000000 - 0x03FFFFF |
| NF2 | | 12MB | 0x0400000 - 0x0FFFFFF |
| | MDIO | 256KB | 0x0440000 - 0x047FFFF |
| | DMA | 256KB | 0x0480000 - 0x04CFFFF |
| | ... ..... | | |
| | MAC | 1M | 0x0600000 - 0x06FFFFF |
| | CPU | 1M | 0x0700000 - 0x07FFFFF |
| | 4MB block 2 | 4M | 0x0800000 - 0x0BFFFFF |
| | 4MB block 3 | 4M | 0x0C00000 - 0x0FFFFFF |
| SRAM | | 16MB | 0x1000000 - 0x1FFFFFF |
| NF2(user data path) | | 32MB | 0x2000000 - 0x3FFFFFF |
| DRAM | | 64MB | 0x4000000 - 0x7FFFFFF |

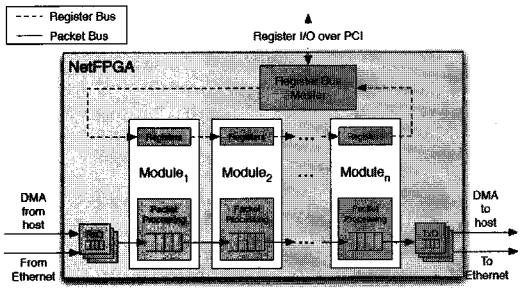



그림 9. NetFPGA 레지스터 데이터 경로[11]
Fig. 9. NetFPGA register data path[11].

서 초기화를 하거나 요청에 대한 응답이 이루어져서 Register Bus Master에 전달되면 PCI 인터페이스를 통하여 소프트웨어에 전달된다. 이와 반대로 소프트웨어에서 임의의 레지스터모듈에 대한 요청이 발생하면 PCI 인터페이스로 전달되고 Register Bus Master 에서 레지스터 모듈로 전달한다. 예를 들어 소프트웨어에서 임의의 NetFPGA 레지스터 값을 읽어 오라는 요청이 레지스터 모듈로 들어오면 자신이 처리 가능한지를 판단하여 다음 레지스터 모듈로 요청을 전달하거나 자신이 처리한다. 처리하는 경우에는 레지스터 모듈 데이터 신호를 통하여 정보를 내보내고, 요청메시지를 수정하여 요청이 처리되었음을 전달한다.

## 3. NetFPGA기반 오픈 플로우 스위치[5]

NetFPGA기본 구조에서 오픈 플로우 스위치에 필요한 기능을 담당하는 모듈들을 user data path 영역에 적용시키면 NetFPGA기반 오픈 플로우 스위치가 된다. NetFPGA의 기본 구조에서 사용되는 RxQ, TxQ, input arbiter, output queues 모듈의 기능과 모듈 헤더를 이용한 모듈 간 통신은 그대로 사용하고, output port lookup 모듈은 오픈 플로우 스위치의 기능을 담당할 수

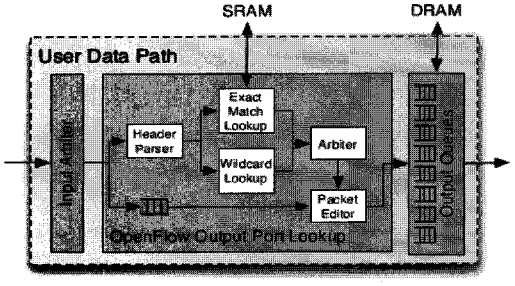



그림 10. NetFPGA 오픈 플로우 스위치 내부 블록도[5]
Fig. 10. Internal block diagram of NetFPGA OpenFlow switch[5].

있게 그림 10과 같이 구성되었다. Output port lookup 모듈의 각 기능 블록은 다음과 같다.

- Header parser: 패킷으로부터 원하는 10-tuple 정보를 추출하여 연결시킨 후 결과를 출력한다.
- Exact Match Lookup: TCAM(Ternary Content-addressable memory)을 확인하여 플로우 엔트리(flow entry)와 입력 정보가 일치하는 지를 확인하고 결과를 출력한다.
- Wildcard Lookup: TCAM을 확인하여 플로우 엔트리에서 입력 정보의 원하는 부분이 일치 하는 지를 확인하고 결과를 출력한다.
- Arbiter: 두 개의 입력 중 하나를 선택하여 결과를 출력한다.
- Packet Editer: 두 개의 입력을 가지고 있다. 하나는 패킷 정보에 대한 입력을 받고 다른 하나는 플로우 엔트리와 패킷 10-tuple의 비교 값이다. 두 값을 입력 받아 플로우 엔트리를 수정하거나, 오픈 플로우 프로토콜에 지정된 실행명령에 따라서 패킷을 수정하여 결과 출력한다.

## III. 미래 인터넷 테스트베드 구현

### 1. 테스트베드 구현 방안

국내 미래 인터넷 테스트베드를 구축하기 위해서 NetFPGA기반 오픈 플로우 스위치를 활용하여 로컬 테스트 환경을 구축하고 이를 기반으로 하여 전국적인 테스트베드를 만드는 것이 하나의 방안으로 연구되고 있다. 이를 위해서는 인터넷망을 통한 로컬 테스트베드 간 연동이 요구되며, 이를 위한 해결책으로 본 논문에서는 MAC in IP 터널링 방법을 활용하였다. MAC in IP 터널링 방법은 로컬 테스트베드에서 나오는 MAC 프레임 전체를 하나의 데이터로 인식하여 송신단의 캡슐레이터에서 목적지 캡슐레이터의 인터페이스 주소를 목적지 IP주소로 하는 IP 헤더를 추가하여 인터넷망을 통해 수신 단 캡슐레이터로 전달하는 것이다. 수신측 테스트베드에서는 전달받은 패킷의 IP 헤더를 캡슐레이터에서 제거하여 원래 MAC 프레임을 내부 테스트베드에 전달하기 때문에 오픈 플로우 스위치에 연결된 호스트들은 인터넷으로 연결되어 있음에도 불구하고 마치 동일한 스위치에 연결된 것 같이 트래픽을 주고받게 되는 것이다. 이러한 과정을 수행하기 위해서 그림 11와

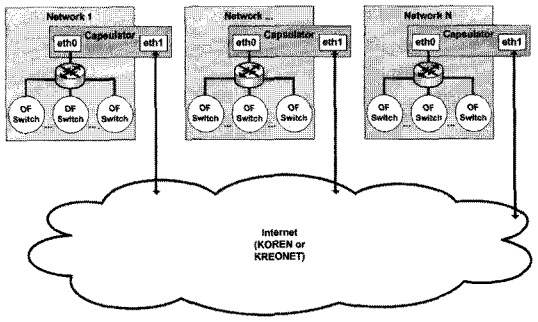



그림 11. 테스트베드 구성도[6]
Fig. 11. Testbed topology[6].

같이 두 개의 이더넷 포트 중 하나는 테스트베드와 연결하고, 다른 하나는 인터넷망에 연결하면 캡슐레이터 기능을 해당 포트 트래픽에 적용하여 송, 수신 되도록 전체 망을 구성하였다.

### 2. 캡슐레이터 구현

NetFPGA user data path 하드웨어 로직 간 데이터 통신은 64bit 로 이루어지므로 그림 12와 같이 IP encapsulation 헤더 포맷을 64bit 로 구성하였다. 송신지와 목적지 MAC, IP 주소는 레지스터를 통하여 사용자로부터 입력을 받도록 하였고, encapsulation 헤더는 패킷 데이터부분에 해당하므로 ctrl 신호를 0x00으로 설정하였다. 이때 송신 IP주소는 전송하는 캡슐레이터의 IP주소, 수신 IP 는 수신하는 캡슐레이터의 IP주소가 된다.

user data path 통신과정에서 패킷 길이, 들어온 포트, 나가는 포트가 명시된 모듈 헤더를 주고받게 된다. 모듈헤더에서 패킷 길이를 나타내는 부분은 모듈 중간에 패킷 길이가 변경되면 수정해주어야 하는데 캡슐레이터 모듈의 경우 패킷에 캡슐레이터 MAC, IP, UDP 헤더가 추가되거나 삭제되는 과정에서 패킷 길이에 변화가 생기므로 모듈헤더에서 패킷의 길이를 나타내는 부분을 수정해 주어야 한다.

캡슐레이터 헤더를 추가하는 encapsulation 모듈의

| CTRL | user_data_path | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Bits 63-56 | Bits 55-48 | Bits 47-40 | Bits 39-32 | Bits 31-24 | Bits 23-16 | Bits 15-8 | Bits 7-0 |
| 0x00 | mac_dst 48 | | | | | | mac_src_hi 16 | |
| 0x00 | mac_src_lo 32 | | | | mac_ethertype 16 | | ip_ver 4<br>ip_header_len | ip_ToS 8 |
| 0x00 | ip_total_length 16 | | ip_id 16 | | ip_flags 3<br>ip_flag_offset 13 | | ip_TTL | ip_prot 8 |
| 0x00 | ip_header_checksum 16 | | ip_src 32 | | | | ip_dst_hi 16 | |
| 0x00 | ip_dst_lo 16 | | udp_src 16 | | udp_dst 16 | | udp_length 16 | |
| 0x00 | udp_checksum 16 | | capsulator_reserved 48 | | | | | |

그림 12. 캡슐레이터 헤더 포맷
Fig. 12. Capsulator header format.

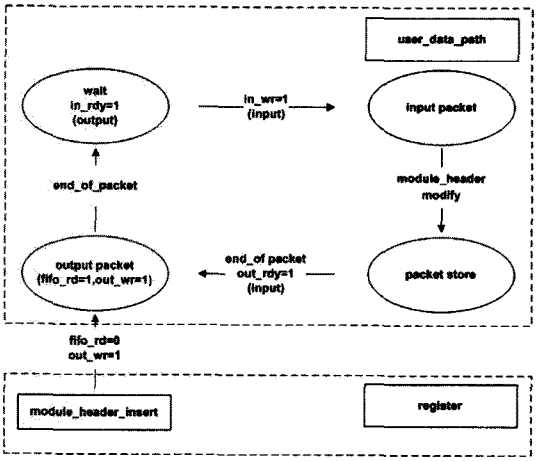



그림 13. encapsulation 모듈의 상태도
Fig. 13. State diagram for encapsulation module.

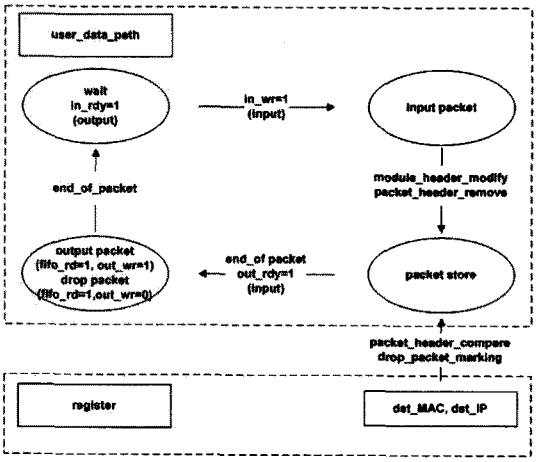



그림 14. decapsulation 모듈의 상태도
Fig. 14. State diagram for decapsulation module.

상태도는 그림 13과 같다. 처음에 in_rdy를 1로 설정하고 대기 상태에 있다가 in_wr가 1로 바뀌고 데이터가 들어오면 모듈 헤더의 패킷 길이를 나타내는 부분을 수정하여 큐에 저장한다. 패킷 한 개가 큐에 다 저장되고, out_rdy가 1로 바뀌면 패킷을 다음 모듈에 전달한다. 전달되는 과정에서 ctrl 신호를 이용하여 모듈 헤더를 구분하고 모듈 헤더 다음에 캡슐레이터 헤더를 추가하여 다음 모듈에 패킷을 전달하고 다시 대기상태로 들어간다.

캡슐레이터 헤더를 제거하는 decapsulation 모듈의 상태도는 그림 14와 같다. 먼저 in_rdy를 1로 설정하고 대기 상태에서 패킷이 들어오는 것을 기다리다가 in_wr가 1로 바뀌면서 패킷 데이터가 들어오면 모듈 헤더를 수정하고 캡슐레이터 헤더를 삭제한 후 패킷의 나머지 부분은 큐에 저장한다. 이때 레지스터에 저장된 송신지 MAC, IP, PORT 와 패킷의 목적지 MAC, IP, PORT

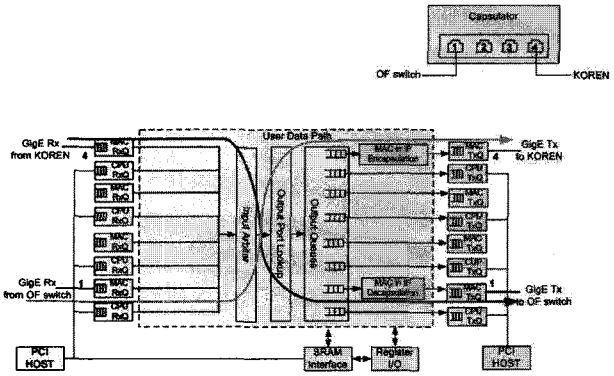



그림 15. 캡슐레이터 user data path
Fig. 15. Capsulator user data path.

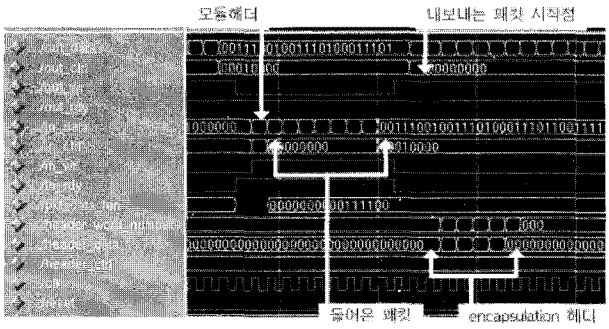



그림 16. 캡슐레이터 모듈 시뮬레이션
Fig. 16. Capsulator module simulation.

를 비교하여 값이 다르면 버리는 패킷으로 표시한다. 패킷 한 개가 큐에 다 저장되고, out_rdy가 1로 바뀌면 패킷에 표시된 정보를 이용하여 다음 모듈에 전달하거나, 패킷을 버린다. 패킷이 전달되면 다시 대기상태에 들어간다.

캡슐레이터 모듈을 NetFPGA user data path에서 그림 15와 같이 output queue 와 MAC TxQ 사이에 연결한다. 인터넷 망과 연결되는 4번 포트에는 encapsulation 모듈을 연결하고 오픈 플로우 스위치와 연결되는 1번 포트에는 decapsulation 모듈과 연결한다. Output port lookup 모듈을 수정하여 1번 MAC 으로 들어오는 패킷을 4번 MAC으로 내보내고 4번 MAC 으로 들어오는 패킷을 1번 MAC 으로 내보낸다. Modelsim 프로그램을 활용하여 작성된 캡슐레이터 모듈에 대한 시뮬레이션을 수행하였다. 그림 16과 같이 캡슐레이터 모듈에서 들어오는 패킷에 대하여 새로운 헤더가 추가되는 것을 확인하였다.

## 3. NetFPGA 캡슐레이터 성능 평가

로컬 테스트베드 간 연동을 위해 구현된 NetFPGA기

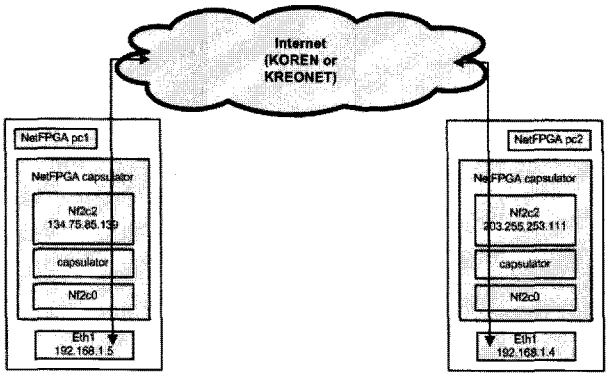



그림 17. 로컬 테스트베드 구성도
Fig. 17. Local testbed topology.

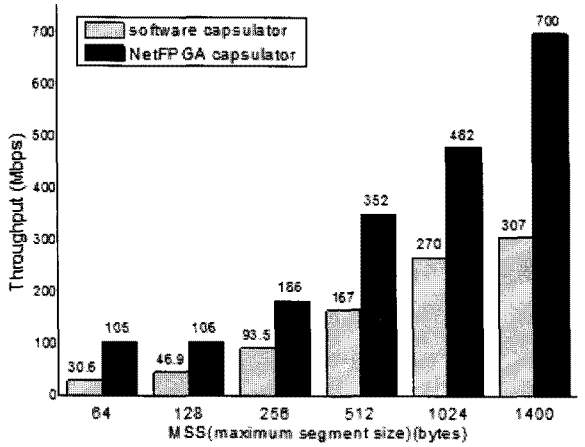



그림 18. 캡슐레이터 로컬 테스트 결과[13]
Fig. 18. Capsulator local test result[13].

반의 하드웨어 캡슐레이터의 성능을 평가하기 위해 그림 17과 같이 NetFPGA 플랫폼을 활용하여 로컬 테스트 환경을 구성하였다.

NetFPGA PC 1번과 2번을 KREONET망(134.75.85.13x)에 연결하고 iperf[12] 프로그램을 서버와 클라이언트로 각각 동작 시켰다. Iperf 프로그램에서 옵션 '-M'을 이용하면 TCP MSS(maximum segment size)변경이 가능하다. MSS 사이즈를 64, 128, 256, 512, 1024, 1400 byte로 변경하면서 각각 3600초씩 동작시켜 TCP 대역폭을 측정해 본 결과[13]는 그림 18과 같다. NetFPGA 기반 캡슐레이터는 하드웨어 가속화를 활용함으로서 기존 소프트웨어 캡슐레이터에 비해 전달 성능이 크게 향상된 것을 알 수 있다.

다음은 그림 17에서 NetFPGA PC 1번에 KREONET망(134.75.85.138)을 연결하고 PC 2번에 KOREN망(203.255.253.111)를 연결하여 ping 프로그램을 이용하여 캡슐레이터 동작여부를 확인하였다. 양단에 iperf 프로그램을 실행시키고 MSS를 1400 byte로 설정하여 60초간 TCP 대역폭을 측정한 결과 599Mbps 가 나오는 것을 확인하였다. 이 값은 캡슐레이터의 오버헤드만을 고려한 로컬 테스트(700Mbps)에 비해 조금 낮은 결과인데 이는 KREONET, KOREN 망에서의 지연시간 및 망 상태에 따른 결과로 판단된다.

## IV. 결론 및 향후 연구

본 논문에서는 플로우 기반의 오픈 플로우 프로토콜과 프로그램 가능한 NetFPGA시스템을 활용한 미래인터넷 테스트베드 구축 방안을 소개하였다. 오픈 플로우 로컬 테스트베드를 구축하고 인터넷 망을 통하여 전국적인 단일 테스트베드로 확장하기 위해 MAC in IP 터널링 기능을 하는 캡슐레이터를 설계하였다.

구현된 하드웨어 기반의 캡슐레이터를 사용해 최대 전송 성능을 측정한 결과 기존 소프트웨어 캡슐레이터에 비해 성능이 향상되었음을 알 수 있었다. 또한 NetFPGA 캡슐레이터가 갖는 전송 가능한 최대 패킷 사이즈를 계산하고 실제 망 환경에서도 동일하게 나타남을 확인하였다.

구현된 캡슐레이터를 이용하면 대용량 멀티미디어 전송도 가능한 대역폭을 보장해주지만 전국적인 단일 테스트베드 구축 시 점대점(point to point)환경 밖에 지원하지 못하는 문제점과 오픈 플로우 스위치 외에 또 다른 장비가 추가로 필요하게 되어 하드웨어 리소스가 낭비되는 문제가 있다. 이에 현재 연구를 NetFPGA 캡슐레이터 모듈에 MAC learning이 가능한 브릿지 모듈을 구현하여 멀티 포인트 환경을 지원하는 연구와 기존 NetFPGA 기반 오픈 플로우 스위치에 캡슐레이터 모듈을 추가하는 OpenFlow + capsulator 을 구현하는 연구를 진행 중에 있다.

## 참 고 문 헌

[1] 박종호, 서승우, "미래인터넷 연구동향", 한국통신학회지(정보와통신) 제24권 제10호, pp. 44-50 2007. 10.
[2] Jinho Hahm, Bongtae Kim, and Kyungpyo Jeon, "The study of Future Internet platfom in ETRI", The Magazine of the IEEK, Vol.36, No.3, March, 2009.
[3] OpenFlow, http://www.openflowswitch.org/

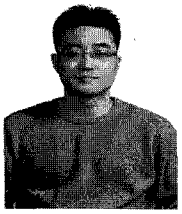
[4] NetFPGA, http://www.netfpga.org/
[5] Jad Naous, David Erickson, Adam Covington, Guido Appenzeller, and Nick McKeown, "Implementing an OpenFlow Switch on the NetFPGA platform", ANCS'08, San Jose, CA, USA, November 6-7, 2008.
[6] Capsulator, http://www.openflowswitch.org/wk/index.php/Capsulator
[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. ACM Computer Communications Review, 38(2):69~74, 2008.
[8] Nick McKeown, OpenFlow, http://www.openflowswitch.org/documents/OpenFlow.ppt
[9] Brandon Heller, "OpenFlowSwitch Specification", http://www.openflowswitch.org/documents/openflow-spec-v0.8.9.pdf
[10] Glen Gibb, John W. Lockwood, Jad Naous, Paul Hartke, and Nick McKeown, "NetFPGA -- Open Platform for Teaching How to Build Gigabit-rate Network Switches and Routers", IEEE Transactions on Education, 2008.
[11] Jad Naous, Glen Gibb, Sara Bolouki, and Nick McKeown, "NetFPGA: Reusable Router Architecture for Experimental Research", SIGCOMM PRESTO Workshop, Seattle, WA, August 2008.
[12] iperf, http://www.noc.ucf.edu/Tools/Iperf/
[13] 최윤철, 김승주, 이재용, 김병철, 김대영, "NetFPGA 기반 MAC in IP capsulator 구현", 2009 추계종합학술발표회, 2009. 11.

## 저 자 소 개


**최 윤 철(학생회원)**

2007년 충남대학교 전자전파 정보통신공학과 학사
2007년~2010년 충남대학교 전자전파정보통신공학과 석사
2010년~현재 충남대학교 전자전파정보통신공학과 박사과정

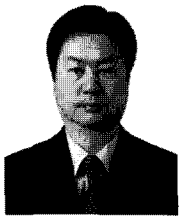
<주관심분야 : 이동통신 네트워크, 데이터 통신, 테스트 베드>



**민 석 홍(학생회원)**

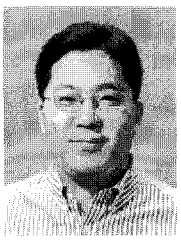
2005년 공주대학교 전기전자정보공학과 석사
2004년 한국전자통신연구원 BcN 시험기술팀 위촉연구원
2005년 디지피아(주) 방송장비팀 연구원
2006년~2009년(주)엠티아이 연구2실 전임연구원/대리
2010년~현재 충남대학교 전자전파정보통신공학과 박사과정

<주관심분야 : 데이터 통신, NetFPGA>



**김 대 영(평생회원)**

1975년 서울대학교 전자공학과 학사
1977년 한국과학기술원 통신공학 석사
1983년 한국과학기술원 통신공학 박사
1983년~현재 충남대학교 정보통신공학부 교수
현재 미래인터넷 포럼(FIF) 부의장, APAN 부의장, ISO/IEC JTC1/SC6 의장



**김 병 철(평생회원)-교신저자**

1988년 서울대학교 전자공학과 학사
1990년 한국과학기술원 전기 및 전자공학과 석사
1996년 한국과학기술원 전기 및 전자공학과 박사
1993년~1999년 삼성전자 CDMA 개발팀
1999년~현재 충남대학교 정보통신공학부 부교수

<주관심 분야 : 이동인터넷, 이동통신 네트워크, 데이터통신>

**이 재 용(평생회원)**

1988년 서울대학교 전자공학과 학사
1990년 한국과학기술원 전기 및 전자공학과 석사
1995년 한국과학기술원 전기 및 전자공학과 박사
1990년~1995년 디지콤 정보통신 연구소 선임연구원
1995년~현재 충남대학교 정보통신공학부 교수

<주관심분야 : 초고속통신, 네트워크 성능분석>