논문 16-41-10-22 The Journal of Korean Institute of Communications and Information Sciences '16-10 Vol.41 No.10
http://dx.doi.org/10.7840/kics.2016.41.10.1290

캐리어 급 주소 변환기(NAT)의 설계 및 구현

이 문 상◆, 이 치 영°, 김 우 태*, 이 영 우*

Design and Implementation of Carrier-Grade Network Address Translation (NAT)

Moon-Sang Lee◆, Chiyoung Lee°, Wootae Kim*, Young-Woo Lee*

요 약

최근 들어, 유무선 망의 종단 사이에 위치하여 다양한 네트워크 기능을 제공하는 미들박스 서비스가 일반화되고 있다. 특히, 클라우드 컴퓨팅 분야의 가상화 기술이 네트워크 분야에 적용되면서 네트워크 가상화가 빠르게 진행되고, 가상 네트워크 장비들을 유연하게 연결하여 민첩한 네트워크 서비스를 제공하는 플랫폼들이 연구되고 있다. 본 논문에서는 캐리어 급 미들박스 서비스를 제공하기 위한 필수 요소들을 살펴보고, 범용서버에서 캐리어 급 네트워크 주소 변환 서비스를 제공하기 위한 프레임워크의 설계와 구현에 대해 기술한다. 실험 결과에 의하면, 제안하는 프레임워크는 기존의 리눅스 커널에서 제공하는 주소 변환 기능보다 15.5배 빠른 성능을 보인다.

Key Words : NAT, Carrier-Grade, DPDK, Packet Processing

ABSTRACT

Recently, various middle box services have been developed and applied to provide network functions to end nodes of the network. Especially, network virtualization is increasingly proceeding by applying the virtualization technologies of cloud computing field to network field, and network platforms for various flexible services are being developed to connect among the virtual network devices. Carrier-grade Network Address Translation (CGNAT) is also one of these flexible network services. This paper designs and implements the DPDK-based CGNAT framework that provides flexibility and maximizes address translation throughput. Our framework achieves 15.5 times higher throughput than the address translation service by Linux kernel.

Ⅰ. 서 론

미들박스는 통신에 참여하는 양 종단 사이에 위치하여 특정 기능을 수행하는 네트워크 장치를 통칭한다. 예를 들어, 기업의 사내외 망을 연결하여 보안을 제공하는 방화벽, 클라우드의 전단에 위치하여 백엔드 노드들의 부하를 분산시키는 로드밸런서, 연결 경로상의 중간에서 파일 및 컨텐츠의 접근 속도를 향상시키는 캐쉬/프락시, 사설망과 공용 망 사이에 위치하여 사설 IP주소를 공용 IP 주소로 변환하는 네트워크 주소 변환기 등을 포함한다. RFC 3234[1]에서는 22종의 미들박스 서비스들을 정의하고 각 미들박스 종류의 특징을 기술하고 있다. 본 논문에서는 초고속 네트워크에서 캐리어 급 주소변환(CGNAT: Carrier-Grade Network Address Translation)을 제공하기 위한 프레임워크에 대해 기술한다.

네트워크 주소변환(NAT: Network Address Translation)은 사설 IP 주소를 공용 IP 주소로 변환하

◆ First Author : KT Institute of Convergence Technology, moonsang.lee@kt.com, 정회원
° Corresponding Author : KT Institute of Convergence Technology, chiyng.lee@kt.com, 정회원
* KT Institute of Convergence Technology, utae.kim@kt.com, young-woo.lee@kt.com, 종신회원
논문번호 : KICS2016-07-147, Received July 12, 2016; Revised October 4, 2016; Accepted October 5, 2016

는 기능이다. 네트워크 단말의 증가로 인해 부족해진 공용 IPv4 주소 공간을 효율적으로 확장하거나 네트워크 주소공간을 분리하여 보안성을 향상시키기 위한 목적으로 사용되며, 통상적으로 가입자 사이트의 공유기 또는 Wi-Fi AP(Access Point)에서 수행되어 왔다. 최근에는 IoT 기술의 확산으로 인해 각종 센서들을 포함하는 많은 단말이 네트워크를 통해 연결되기 때문에 이들의 식별을 위해 주소변환을 활용한다[2]. 캐리어급 주소변환 서비스는 통신사에서 운용하는 장비에서 모든 가입자 패킷에 대한 주소변환을 수행하며, RFC 6888[3]에서는 캐리어급 주소변환을 위한 기본 요구사항들을 기술하고 있다. 기본적으로 가입자 사이트와 통신사 주소변환 장비 사이는 L2 계층으로 연결되어 있다고 가정하고, 통신사는 IEEE 802.1ad[4] 표준 태그를 이용하여 가입자를 구분하거나 가입자간 간섭을 제거할 수 있다.

일반적으로, 캐리어 급 주소변환은 수많은 가입자의 모든 패킷에 대해 수행되어야 하기 때문에 높은 패킷 처리율이 요구된다. 전통적인 리눅스 서버의 네트워크 처리 성능으로는 이 요구를 만족할 수 없기 때문에 리눅스 서버의 네트워크 성능을 극대화하는 것이 필요하다. 본 논문에서는 캐리어 급 주소변환 서비스를 제공하기 위한 CGNAT 프레임워크를 설계하고, x86 기반의 범용 서버에서 실행 가능한 프로토타입을 구현한다. 이를 위해, 제안하는 CGNAT 프레임워크는 DPDK를 이용하여 패킷 전달 성능을 극대화하고, 파이프라인 구조로 구현하여 패킷 당 프로세싱 성능을 최대화한다. 10Gbps 링크에서 단방향 패킷에 대한 주소변환에 따른 패킷 처리량(Throughput)을 측정한 본 논문의 실험에 따르면, 리눅스 커널의 iptables를 사용하여 주소변환을 수행한 경우에 비해 본 논문의 CGNAT 프로토타입 처리율이 패킷 크기 전 구간에 대해 2~15배 향상되었다.

Ⅱ. 데이터 평면 가속 기술

리눅스 커널의 TCP/IP 스택에 기반한 패킷 송수신은 인터럽트 처리, 문맥교환(Context Switch) 및 스케줄링, 빈번한 시스템 콜과 데이터 복사 등으로 인한 부하가 발생한다. 이를 개선하기 위해 UIO(Userspace IO)[5], NAPI(New API)[6], 폴링(Polling)을 사용한 고속 패킷 처리 기술들이 연구되어 왔으며, 오픈소스 프로젝트로 진행 중인DPDK(Data Plane Development Kit)[7]가 대표적인 예이다.

DPDK는 Intel 아키텍처 기반 고성능 패킷 처리에 최적화를 위한 시스템 소프트웨어로, 어플리케이션 프로세스가 하드웨어에 직접 접근할 수 있는 API 라이브러리와 프레임워크를 제공한다. 이는 리눅스 커널의 느린 네트워크 스택 처리를 우회함으로써 커널을 통해 통신할 때 발생하는 부하를 줄여 패킷 처리 성능을 향상시킨다. 이를 위해 미리 할당해 놓은 유저 레벨의 메모리 영역을 패킷 버퍼로 사용한다. 이는 유저 레벨 메모리에 NIC이 직접 패킷을 저장하고, 유저 프로세스가 직접 패킷을 읽어가기 때문에 커널의 개입을 막고 각 패킷에 대한 제로-카피(Zero-Copy)를 제공한다. 또한, 패킷을 저장하기 위한 메모리 공간을 할당할 때, 일반 페이지(4KB)보다 큰 페이지(2MB)를 이용하기 때문에 메모리 페이지 할당 횟수와 TLB Miss를 줄이고 패킷들을 인접한 메모리 공간에 배치할 수 있어 메모리 접근 속도와 캐쉬 활용도를 향상시킨다.

DPDK는 고속 패킷 처리를 위해 PMD(Poll-Mode Driver)를 제공한다. 이는 패킷 송수신 시 폴링 방식을 사용하여 인터럽트 발생으로 인한 오버헤드를 제거한다. 기존의 인터럽트 방식은 패킷 송수신 시 프로세서에게 인터럽트 신호를 보내 동작 중인 프로세스를 중단시키고 패킷을 송수신하도록 한다. 이는 패킷 송수신 여부의 확인을 위해 주기적으로 프로세서가 확인하지 않아도 된다는 장점이 있는 반면, 인터럽트 신호에 의한 프로세스 교체로 인해 문맥교환 오버헤드가 발생한다. 특히, 많은 패킷의 송수신이 발생하는 네트워크 장비에서는 인터럽트 처리로 인해 다른 프로세스가 동작하지 못하여 수신된 패킷의 처리가 지연되거나, 프로세스 간 문맥교환으로 캐쉬가 오염되어 메모리 접근 성능이 낮아진다. 따라서 DPDK는 인터럽트 처리로 인한 오버헤드를 제거하기 위해 폴링 방식을 사용한다.

또한, DPDK는 소프트웨어 기반의 고속 패킷 처리 기술이기 때문에 고가의 장비가 없어도 동작할 수 있는 장점이 있다. 이는 초기 설비 투자비용이 적다는 장점 외에도, 하드웨어 장비의 노후화로 인한 시설 교체 시에도 적은 비용으로 이전 가능하다.

이러한 특성들로 인해, DPDK는 캐리어 급 주소변환에 필수적인 네트워크 성능 극대화를 달성할 수 있다. 또한, DPDK는 이미 다양한 소프트웨어 기반 네트워크 기술들[8-10]에서 채택되어 사용되기 때문에 기술적인 지원을 받기 쉽다는 부가적인 장점이 있어 개발 및 관리 비용 감소에 도움이 된다. 따라서 DPDK는 데이터 평면 가속 기술로 사용되기에 적합하다.

네트워크 가상화를 위한 오픈소스 플랫폼인 OPNFV[11]는 통신사의 캐리어급 네트워크 서비스를

제공하기 위한 통합 플랫폼으로서 그림 1과 같은 플랫폼 스택(현재 버전: Brahmaputra)을 제공한다. ETSI NFV 표준명세에 따라 작성된 네트워크 단위 기능들은 OPNFV 상에서 가상화되고 상호 연결됨으로써 복잡한 네트워크 서비스를 유연하게 제공할 수 있다. 전통적인 TCP/IP 스택으로 다수 가입자로부터 발생하는 수많은 패킷들을 서비스 할 경우 심각한 성능 저하에 직면할 수 있기 때문에 DPDK 또는 ODP(Open Data Plane)[12]와 같은 데이터 평면 가속화 계층을 통해 고속 처리 성능을 제공하고, 성능에 직접적인 영향이 없는 제어 평면의 경우 기존의 TCP/IP 스택을 사용하여 통신할 수 있다.

또 다른 데이터 평면 가속 기술로, 리눅스 커널의 TCP/IP 스택을 최적화하고 각 패킷의 처리 작업(라우팅, 주소변환, 필터링)을 별도의 프로세싱 유닛으로 오프로딩하는 방식이 있다. RouteBricks[13]는 소프트웨어 라우터의 성능을 향상시키기 위해 4대의 서버를 클러스터로 구성하여 하나의 라우터로 동작하게 한다. 이는 다수의 서버에서 병렬적으로 여러 패킷의 경로 탐색을 동시에 수행함으로써 데이터 평면의 성능을 향상시킨다. PacketShader[14]는 패킷 라우팅과 IPsec 처리를 GPGPU(General Purpose Graphic Processing Unit)에서 병렬적으로 수행한다. GPGPU는 수백 개의 프로세싱 코어를 가지고 있기 때문에 이들을 이용하여 많은 수의 패킷을 한꺼번에 처리할 수 있다. 하지만, 이 방식들은 네트워크의 부하가 높아짐에 따라 추가적인 프로세싱 유닛을 필요로 하기 때문에 한 서버에서의 패킷 송수신 능력을 최대화한 DPDK에 비해 유연성이 떨어지는 단점이 있다.

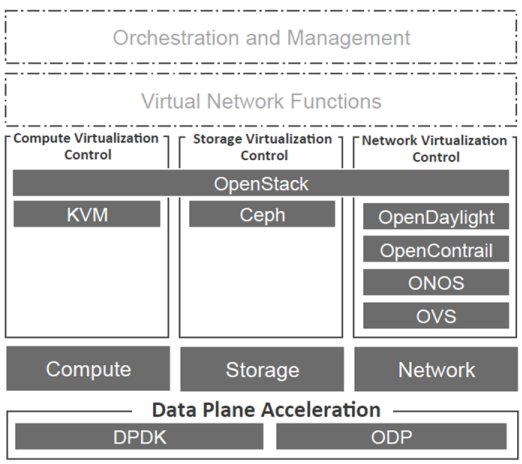

그림 1. OPNFV Brahmaputra 플랫폼의 구조
Fig. 1. OPNFV Brahmanputra platform architecture

Ⅲ. CGNAT 프로토타입 구현

3.1 CGNAT 프레임워크의 특징

제안하는 CGNAT 프레임워크의 목표는 수많은 가입자의 패킷을 처리할 수 있는 높은 패킷 처리율을 제공하는 것이다. 이를 위해, 본 논문은 DPDK에 의한 패킷 전달 속도 향상과 파이프라인 구조에 의한 병렬적 처리를 통해 기존에 비해 성능을 극대화한다.

DPDK는 리눅스 커널이 가진 복잡한 범용 네트워크 스택의 패킷 처리 루틴들을 우회하여, 패킷을 CGNAT 프레임워크로 직접 전달하는 역할을 한다. 일반적으로 리눅스 커널의 네트워크 스택은 범용성을 고려하여 구현되었기 때문에 CGNAT와 같은 미들박스형 네트워크 장비에 불필요한 다양한 네트워크 기능들이 포함되어 있다. 이들이 리눅스 커널의 패킷 처리 성능을 크게 저하시킨다는 것은 기존의 연구들을 통해 밝혀진 사실이다[8,15-17]. 또한, 리눅스 커널의 네트워크 스택은 매 패킷 수신마다 네트워크 스택이 사용하는 커널 레벨 메모리와 CGNAT 프레임워크이 사용하는 유저 레벨 메모리 사이의 패킷 복사를 수행해야 하고, 패킷을 저장하기 위한 소켓 버퍼의 할당과 해제가 반복되어 패킷 처리 성능이 낮아진다[14]. 이로 인해, 리눅스의 패킷 전달(forward) 성능은 최대 14.88Mpps(Million packets per second)의 패킷 전송 능력을 가진 10G NIC에 대해 약 0.7Mpps에 불과하다. 따라서 제안하는 CGNAT 프레임워크는 DPDK를 이용하여 패킷을 리눅스 커널을 거치지 않고 수신하여 느린 패킷 전달 성능을 극복한다.

파이프라인 구조는 CGNAT 프레임워크의 패킷 처리를 병렬화하여 패킷의 처리 지연을 최소화한다. CGNAT는 수신한 모든 패킷의 주소를 변환하기 때문에 패킷 수가 늘어날수록 CPU에 의한 작업량이 증가한다. 파이프라인 구조는 주소변환에 필요한 작업을 세분화하여 하나의 패킷을 처리하는 동안 다른 패킷의 작업을 동시에 수행하도록 한다. 이를 통해, 각 패킷의 대기 시간을 최소화하여 전체적인 패킷 처리 능력을 향상시킨다.

3.2 CGNAT 프레임워크의 구성 요소와 동작

CGNAT는 다수의 가입자들로부터 전달된 패킷들의 주소를 변환하여 외부 인터넷 망으로 전달하는 역할을 수행한다. 이를 위해, CGNAT 서버는 다수의 접속 단말 별로 사설 IP 주소를 할당하고, 해당 단말들이 외부 망을 접속할 때 사용할 공용 IP 주소를 관리한다. 사설 IP 주소는 각 가입자가 가진 접속 단말마

다 할당이 되고, 공용 IP 주소는 외부 망에 연결된 DHCP 서버로부터 할당 받아 각 가입자 별로 맵핑된다. 따라서 특정 가입자 사이트의 단말들이 사용하는 다수의 사설 IP 주소들은 공용 IP 주소 하나를 공유한다.

그림 2는 본 논문에서 구현한 CGNAT 서버의 주요 소프트웨어 구성을 나타낸다. DHCP 동작을 위해서 ISC(Internet Systems Consortium)에서 오픈소스로 배포하는 DHCP 참조 구현(Reference Implementation)을 사용하였다. Private DHCPD는 가입자 단말에게 사설 IP 주소를 할당하기 위한 DHCP 서버이고, Public DHCP Agent는 가입자 단말이 외부 망에서 인식될 공용 IP 주소를 할당받기 위한 DHCP 클라이언트이다. NAT Daemon은 사설 IP 주소와 공용 IP 주소 간 매핑을 관리하고, 공용 IP 주소의 요청(Request), 갱신(Renew), 및 해제(Release)를 제어한다. CGNAT 서버의 모든 패킷은 DPDK를 통해 송수신되며, 패킷의 목적지가 CGNAT 서버일 경우에는 DPDK KNI(Kernel NIC Interface)를 통해 CGNAT 서버의 리눅스 커널에게 전달한다.

NAT Daemon은 {사설 IP 주소, 단말 MAC 주소} → {공용 IP 주소, 가입자 사이트별 가상 MAC 주소}로 변환하기 위한 매핑과 세션 유지를 위한 5-튜플(출발지 IP, 목적지 IP, 출발지 포트 번호, 목적지 포트 번호, 프로토콜 번호)별 연결 관리를 제공한다. 가입자 사이트별 가상 MAC 주소는 SER(Service Edge Router)에서 가입자별 트래픽 관리에 사용되고, 5-튜플별 연결 관리는 의도하지 않은 패킷이 외부 망으로부터 유입되는 것을 차단하기 위한 용도(Address and Port Dependent Filtering)로 사용된다.

NAT Daemon에 의한 네트워크 주소변환은 패킷의 전송 방향에 따라 달라진다. 내부 망에서 외부 망으로 나가는 패킷의 경우는 L2 헤더의 출발지 MAC 주소와 L3 헤더의 출발지 IP 주소가 변경되고, 외부 망에서 내부 망으로 들어오는 패킷의 경우는 L2 헤더의 목적지 MAC 주소와 L3 헤더의 목적지 IP 주소가 변경된다. 또한, L4 계층의 포트 충돌이 발생할 경우 출발지 포트 번호가 변경될 수 있다. 또한, 주소변환 과정에서 L2 헤더와 L3 헤더의 변경이 발생하므로 각 헤더의 CRC 값(Checksum)을 갱신해야 하며, L4 헤더의 CRC 값이 출발지와 목적지 IP 주소에 의존성이 있으므로 UDP/TCP 같은 L4 프로토콜 종류에 따라 L4 헤더의 CRC 값 갱신이 필요하다.

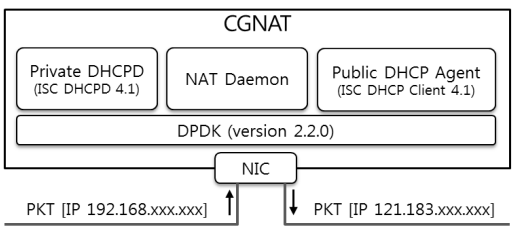

그림 2. CGNAT 서버의 소프트웨어 구성도
Fig 2. Software architecture of CGNAT server

그림 3은 CGNAT 서버의 전체적인 제어 흐름을 상세하게 나타낸다. 이 제어 흐름은 초기화와 데이터 전달로 구분할 수 있다. 초기화는 가입자의 단말이 처음 네트워크에 접속하여 주소를 할당받는 단계로, 해당 단말이 네트워크에 연결되면 Private DHCPD에게 IP 주소 할당 요청을 보낸다. 이 요청을 받은 Private DHCPD는 사설 IP를 단말에게 할당한다. 이와 동시에, CGNAT 서버에 있는 Private DHCP lease file에 새로 할당된 사설 IP와 단말의 MAC 주소 등의 부가 정보를 업데이트한다. NAT Daemon은 주기적으로 Private DHCP lease file을 감시하여 사설 IP 주소 업데이트 사실을 확인하고, 해당 사설 IP 주소와 맵핑할 공용 IP 주소를 요청한다. 이때, 이미 공용 IP 주소를 할당받은 가입자라면 해당 가입자의 기존 공용 IP 주소를 단말에게 할당하고, 새로운 가입자이면 Public DHCP server에게 새로운 공용 IP 주소를 요청한다. 이후, 할당받은 공용 IP 주소와 사설 IP 주소를 맵핑하여 NAT 테이블에 새로 등록하고, 초기화를 완료한다.

데이터 전달 과정은 초기화 완료 후, 단말이 네트워크를 통해 데이터 패킷을 전송하면서 시작된다. 데이터 패킷을 수신한 NAT Daemon은 사설 IP 주소를 키(Key)로 이용해서 NAT 테이블로부터 공용 IP 주소와 가입자 사이트별 가상 MAC 주소를 탐색한다. 맵핑된 주소를 발견하면, 수신 데이터 패킷의 출발지의 IP 주소와 MAC 주소를 탐색에서 발견한 공용 IP 주소와 가상 MAC 주소로 변경하고, 5-튜플별 연결 관리 테이블에 등록한 후에 외부 망으로 데이터 패킷을 전송한다. 이후, 외부에 위치한 데이터 서버 측에서는 패킷의 출발지 주소에 기록된 공용 IP 주소로 응답을 보낸다. 이 응답 패킷을 수신한 NAT Daemon은 5-튜플별 연결 관리 테이블을 탐색하여 유입이 허용되어 있는지를 확인한다. 만약 허용된 패킷이면 5-튜플별 연결 관리 테이블로부터 사설 IP 주소와 단말의 MAC 주소를 얻어와 응답 패킷의 목적지 IP와 MAC 주소를 변경한 후에 단말로 전송하고, 허용되지 않은 패킷이면 드랍 처리한다.

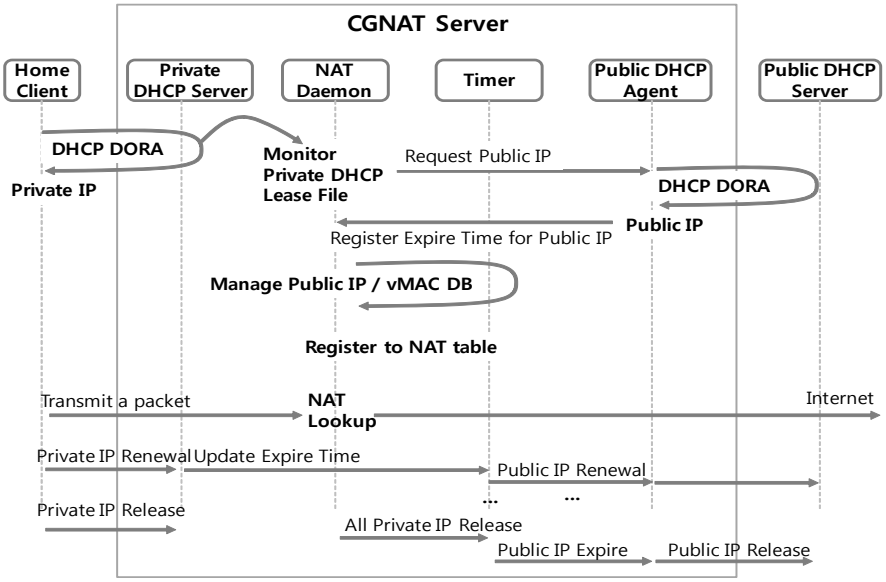

그림 3. CGNAT 서버의 제어 흐름
Fig. 3. Control flow of the CGNAT server

3.3 CGNAT 프레임워크의 구현

본 논문에서는 Intel Xeon CPU를 사용하는 리눅스 서버에서 CGNAT 프레임워크를 구현 하였고, 고속 패킷 처리를 위해 데이터 평면으로 DPDK를 사용하였다. DPDK에서 제시하는 두 가지 구현 모델인 순차적 처리 모델(Run-to-Completion Model)과 파이프라인 모델(Pipeline Framework)에 따라 각각 구현을 진행하여 구현 방법에 따른 성능 비교 대상으로 사용하였다.

CGNAT 서버는 네트워크 주소변환을 위해 그림 4에 표시된 5가지 동작을 순차적으로 수행한다. ①의 단계는 패킷을 수신하여 버퍼에 저장하는 단계이고, ②는 패킷의 유효성을 검사하는 단계이다. CGNAT 서버로 패킷이 수신되면 패킷 종류에 따라 유효성 검사를 먼저 진행하여 유효한 패킷만 주소변환을 수행하고, CRC 값 오류가 있는 패킷들은 유효하지 않은 것으로 간주하여 드랍 처리된다. 이후, ③ 단계에서는

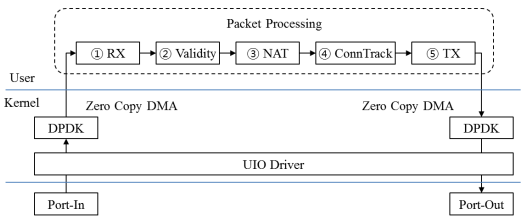

그림 4. CGNAT에서의 순차적 패킷 처리
Fig. 4. Sequential packet processing in CGNAT

주소 변환을 수행한다. 주소 변환에 필요한 사설 IP 주소와 공용 IP 주소 간 매핑과 실제MAC 주소와 가상 MAC 주소 간 매핑 관계는 3장에서 기술한 DHCP 서버를 통해 사전에 NAT 테이블에 등록된다. ④ 단계는 패킷의 출발지에서 목적지로 연결된 세션을 유지한다. 주소가 변환된 패킷은 세션 관리를 위해 ConnTrack 테이블에 5-튜플로 등록하여 응답 패킷의 목적지 주소가 공용IP 주소에서 사설 IP 주소로 변환될 수 있도록 준비한다. ⑤ 단계에서는 주소 변환으로 인해 변경된 헤더 값을 반영하여 L2~L4 각 계층의 헤더의 CRC 값을 재계산하고 이를 각 계층 헤더에 추가한 후, 외부로 나가는 물리적인 NIC 포트를 향해 패킷을 전송한다.

패킷의 수신과 발신은 DPDK에서 제공하는 API를 사용함으로써 리눅스 커널을 우회하여 네트워크 인터페이스 메모리와 CGNAT 프로세스의 메모리 사이에서 DMA로 직접 이동되기 때문에 패킷 버퍼 복사가 발생하지 않는다. 또한, 패킷 송수신을 폴링 방식으로 수행하기 때문에 인터럽트가 발생하지 않고, UIO를 통해 NIC의 레지스터를 직접 접근하므로 패킷 송수신 명령어 전달이나 실행 상태 확인을 위한 시스템 콜이 필요하지 않다.

순차적 패킷 처리는 5단계의 동작들이 완료되어야 다음 패킷의 처리를 시작하기 때문에 다수의 패킷들이 단기간 내 수신될 경우 패킷들이 수신 버퍼에서 대기하는 시간이 증가하고 수신 버퍼에 가용한 공간이

없는 경우 패킷 드랍이 발생할 수 있다. 따라서 5단계를 분할하여 패킷 수신만 수행하는 단계, 수신 버퍼에 있는 패킷에 대해 주소변환만 수행하는 단계, 주소변환이 완료된 패킷을 발신만 하는 단계를 병렬적으로 수행되도록 하는 것이 성능 향상에 도움이 될 수 있다.

그림 5는 주소 변환 5단계를 파이프라인으로 처리하는 과정을 도식화한 것이다. 각 파이프라인은 입력 버퍼로부터 패킷을 수신하여 주어진 단계만 수행하고 출력 버퍼를 통해 다른 파이프라인에게 패킷을 전달한다. 파이프라인 ①은 패킷을 수신하여 자신의 출력 버퍼를 채우는 작업만 반복하고, 파이프라인 ②는 입력 버퍼(①의 출력 버퍼)에 존재하는 패킷들에 대해 유효성 검사를 수행한 후 통과된 패킷만 출력 버퍼에 저장한다. 파이프라인 ③은 입력 버퍼(②의 출력 버퍼)에서 패킷을 꺼내 주소 변환 후 출력 버퍼에 저장하고, ④는 입력 버퍼(③의 출력 버퍼)의 패킷들에 대해 5-튜플을 ConnTrack 테이블에 등록하는 작업을 수행한 후 출력 버퍼에 저장한다. 파이프라인 ⑤는 입력 버퍼(④의 출력 버퍼)의 패킷들에 대해 CRC를 재계산하고 주소변환이 완료된 패킷들을 발신한다. 이때, 각 파이프라인은 다른 파이프라인의 동작 완료를 기다리지 않고, 자신에게 할당된 작업을 반복적으로 수행한다. 따라서 파이프라인 패킷 처리는 순차적 패킷 처리에 비해 병렬성이 증가하여 특정 시점에 패킷 수신, 주소 변환, 패킷 발신이 동시에 처리될 수 있다.

파이프라인 패킷 처리는 각 파이프라인 단계 사이를 연결하는 패킷 큐를 거쳐야 하기 때문에 순차적 패킷 처리에 비해 메모리 접근이 증가할 수 있지만, 병렬적인 패킷 처리를 통해 패킷의 대기 시간을 단축할 수 있으므로 전체 지연시간이 감소하는 효과가 있다. 특히, 각 파이프라인 단계의 부하에 따라 ①~⑤를 적절히 분할함으로써 유연한 성능 최적화가 가능하다.

Ⅳ. 성능 평가

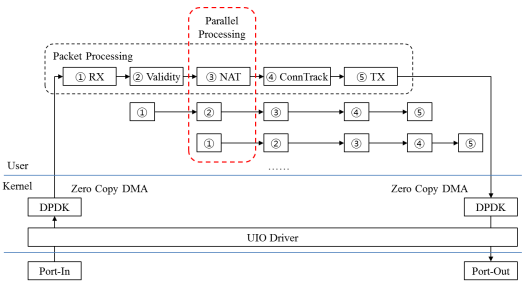

그림 5. CGNAT에서의 파이프라인 패킷 처리
Fig. 5. Pipeline packet processing in CGNAT

4.1 실험 환경

순차적 패킷 처리 CGNAT와 파이프라인 패킷 처리 CGNAT의 성능 평가를 위해, 표 1과 같은 서버를 이용해 CGNAT 서버를 구축하였다. 프로세서는 인텔 Xeon 프로세서 2개로 구성되고, 각 프로세서는 4개의 코어를 갖고 있다. NIC은 DPDK를 지원하기 위해 인텔 82599 칩셋을 사용하는 10G NIC 하나를 사용한다. 이 NIC은 2개의 물리적인 포트로 구성되고, 각 포트는 단말과 연결되도록 하였다.

그림 6은 실험에서 사용한 CGNAT와 단말(client)의 연결을 도식화한 것이다. 패킷 생성 시 단말에서의 지연 시간을 최소화하기 위해 단말에서도 DPDK를 활용한다. 이를 위해 DPDK 기반의 Pktgen[18]을 동작시켜 패킷을 생성한다. 이 구성에서 단말의 0번 포트(P0)에서 전송한 상향 패킷((a)Upstream)은 CGNAT를 거쳐 단말의 1번 포트(P1)로 수신되고, 단말의 P1에서 전송한 하향 패킷((b)Downstream)은 CGNAT를 거쳐 단말의 P0로 수신된다.

표 2는 상향/하향 패킷의 처리를 위해 CGNAT가 수행해야 하는 작업을 나타낸다. 상향 패킷은 단말에

표 1. 성능 평가를 위한 CGNAT 서버의 사양
Table 1. Server specification for evaluation of CGNAT

Components	Specifications
Processor	Intel Xeon E5520 2.27GHz
NIC	Intel 82599EB 10G Ethernet
PCI Bus	x8 PCI Express 2
Data Plane	DPDK v2.2.0
CGNAT OS	CentOS7 (Kernel 3.10.0)

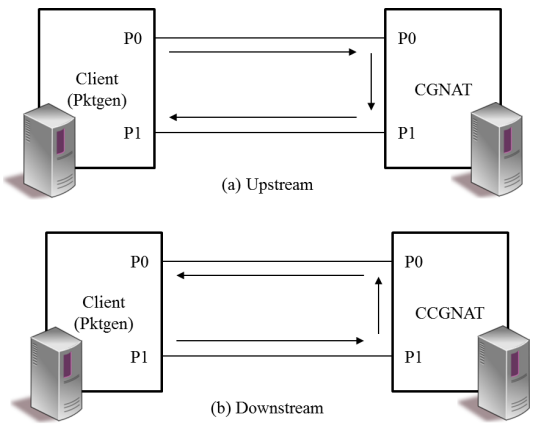

그림 6. 성능 평가를 위한 CGNAT 서버와 단말의 연결도
Fig. 6. Connection between CGNAT server and client for evaluation

표 2. 패킷 진행 방향 별 수행 작업
Table 2. Actions for packet direction

Upstream	Downstream
NAT Table Lookup	ConnTrack Table Lookup
Change of source address	Change of destination address
Registration of 5-tuple into ConnTrack Table	---

서 외부 네트워크(예: 인터넷)로 전송되는 패킷이므로, 주소 변환 테이블(NAT 테이블)을 탐색하여 패킷의 출발지 IP 주소와 출발지 MAC 주소를 각각 공용 IP 주소와 가상 MAC 주소로 변경하고, 목적지와의 세션을 유지하기 위해서 5-튜플 연결 관리 테이블(ConnTrack 테이블)에 해당 패킷의 5-튜플 정보를 등록하는 작업을 수행한다. 하향 패킷은 외부 네트워크에서 단말로 전달되는 패킷으로, ConnTrack 테이블을 탐색하여 사설 IP 주소와 단말의 MAC 주소를 찾고, 패킷의 목적지 IP 주소와 목적지 MAC 주소를 사설 IP 주소와 단말의 MAC 주소로 변경하는 작업을 한다.

4.2 주소변환에 따른 CGNAT의 패킷 처리량

성능 평가를 위해 리눅스 커널(iptables)을 이용한 방법(Kernel), 순차적 패킷 처리를 구현한 방법(uNAT), 파이프라인 패킷 처리를 구현한 방법(uNATp), 단순히 L2 계층 포워딩을 구현한 DPDK 예제코드(L2FWD)를 대상으로 다양한 패킷 크기에 대한 주소변환을 수행했을 때 패킷 처리량을 측정하였다. L2 프레임의 프리앰블(Preamble)과 프레임 간 간격(Inter-Frame Gap)을 고려하면, 64 바이트 패킷에 대한 이론상 최대 대역폭은 7.61Gbps이지만, 프로세서, 메모리, PCI 버스에서 지연이 발생할 수 있으므로 실험에 사용된 하드웨어 사양에 따라 이론상 최대 대역폭보다 낮은 값이 측정될 수 있다.

그림 7은 상향 패킷들의 주소 변환 성능을 측정한 결과 그래프이다. L2FWD은 실험에서 사용한 하드웨어 사양으로 처리할 수 있는 최대 처리율의 상한선(Upper Bound)을 의미하고, Kernel은 주소변환에 따른 패킷 처리량의 하한선(Lower Bound)을 의미한다. Theoretical B/W는 이론상 최대 대역폭을 의미하고, 각 패킷 크기에 대해 프리앰블과 프레임 간 간격을 포함하여 계산한 수치이다.

이 실험 결과에 따르면, 패킷 크기가 증가할수록 상향 패킷들에 대한 처리량이 증가한다. 동일한 10Gbps 대역폭 내에서 패킷 크기가 작을수록 패킷 개수가 많

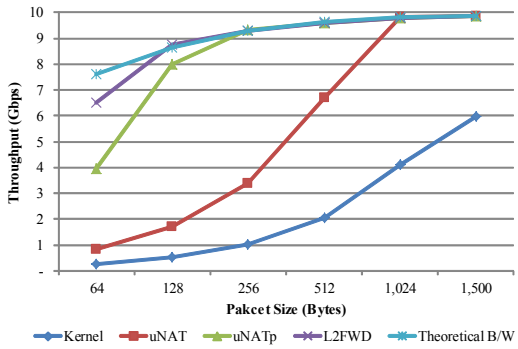

그림 7. CGNAT의 상향 패킷에 대한 패킷 크기별 성능
Fig. 7. Performance of CGNAT for upstream packets

아지기 때문에 CGNAT에서 처리해야 하는 주소변환의 부하가 증가하며, 패킷 크기가 증가할수록 주소변환의 부하가 감소하여 링크 대역폭에 근접한 성능을 나타낸다. 파이프라인 패킷 처리를 구현한 uNATp는, 패킷 크기가 256 바이트 이상인 경우 10Gbps 링크의 이론상 최대 대역폭(Theoretical B/W)에 근접한 성능을 나타낸다. 64바이트 패킷의 경우, uNATp는 리눅스 커널을 이용한 주소변환에 비해 15.5배 높은 처리량을 달성한다.

그림 8은 그림 7과 동일한 환경에서 하향 패킷들의 주소변환 성능을 측정한 결과이다. 하향 패킷들은 상향 패킷의 주소 변환보다 수행해야 할 작업량이 적기 때문에, uNAT와 uNATp 모두 전체적으로 성능이 향상되어 uNAT의 경우도 256 바이트의 패킷에서 이론상 최대 대역폭을 만족한다. L2FWD는 주소변환을 위한 처리 과정 없이 수신한 패킷을 지정된 포트로 내보내는 작업만 수행하기 때문에 상향과 하향의 성능이 동일하게 나타난다. 실험 결과에 의하면, 64 바이트 패킷에 대해, uNATp는 리눅스 커널 대비 14.7배,

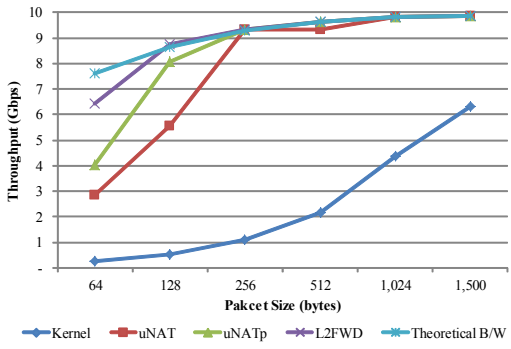

그림 8. CGNAT의 하향 패킷에 대한 패킷 크기별 성능
Fig. 8. Performance of CGNAT for downstream packets

uNAT는 10.4배의 패킷 처리량 향상을 보인다.

또한, 제안하는 uNATp는 DPDK와 파이프라인 구조를 통해, 패킷 프로세싱의 작업량 변화에 대해 안정적으로 높은 성능을 제공한다. 그림 7과 그림 8에 의하면, uNATp는 상향과 하향 패킷의 성능이 유사한 반면, uNAT는 상향에 비해 하향 패킷의 성능이 향상된 것을 볼 수 있다. 이는 하향 패킷에 대한 작업이 상향 패킷에 비해 적기 때문이다. 상향 패킷은 공용 IP망을 통해 패킷을 전달하기 위해, NAT 테이블을 검색하여 해당 단말에 맵핑된 공용 IP와 MAC 주소를 찾고, 출발지 주소를 해당 공용 주소로 변환하며, 세션 유지를 위해 ConnTrack 테이블 엔트리를 생성하는 등의 작업이 필요하다. 반면, 하향 패킷은 ConnTrack 테이블을 탐색하여 단말의 원 IP와 MAC 주소를 찾아 패킷의 목적지 주소를 변환하는 작업만 수행한다. 따라서 순차처리 구조인 uNAT에서는 작업량이 많은 상향 패킷들의 대기 시간이 길어져 전체 성능이 저하되기 때문에 상향과 하향의 성능 차이가 크게 나타난다. 반면, 파이프라인 구조의 uNATp는 병렬적으로 패킷을 처리하기 때문에 작업량이 많은 상향 패킷에 대해서도 대기 시간이 짧아 상향과 하향의 성능이 유사하게 나타난다.

4.3 프로세서 코어 할당에 따른 성능 변화

본 논문에서 구성한 CGNAT는 소프트웨어이기 때문에 이를 구동하는 서버의 하드웨어적인 능력(h/w capacity)에 의해 성능 변화가 나타난다. 특히, 폴링 기반 패킷 송수신과 잦은 테이블 탐색 (예: NAT, ConnTrack)이 발생하므로, 이를 처리할 프로세서의 능력이 중요하다. 또한, uNATp는 파이프라인 구조이기 때문에 각 파이프라인에 할당된 프로세서의 상태에 따라 주소변환 처리 성능이 변화한다. 이는 데이터센터의 서버 자원 할당[19,20]과 동일한 문제이다. 예를 들어, 하나의 프로세서를 두 파이프라인이 사용한다면, 두 파이프라인의 동작을 위해 프로세서에서 스케줄링을 수행해야 한다. 이는 파이프라인 간 문맥교환을 발생시켜 성능을 저하시킬 수 있다. 또한, 2개 이상의 프로세서를 갖는 대부분의 서버는 NUMA(Non-Uniform Memory Access)[21] 구조로 구성되기 때문에 어떤 프로세서, 메모리, PCI 장치를 사용하는가에 따라 이들 사이의 데이터 전달 성능이 달라진다. 예를 들어, 0번 NUMA 소켓(NUMA socket)에 속한 프로세서가 1번 NUMA 소켓에 속한 메모리를 접근하면 데이터 전달 경로가 길어져 성능이 낮아진다. 따라서 본 논문은 프로세서 할당에 따른 성능 차이를 실험을 통해서 보임으로써, 최적의 프로세서 할당을 위한 기초 자료를 마련한다.

그림 9는 표 3에 기술된 프로세서 코어 할당에 따라 측정한 uNATp의 성능을 나타낸다. 표 3의 ‘s’는 NUMA 소켓을, ‘c’는 프로세서 코어를, ‘h’는 하이퍼쓰레드(Hyper-thread)를 의미한다. 예를 들어, s0c1은 0번 NUMA 소켓의 코어 1번을 의미하고, s0c1h는 0번 NUMA 소켓의 코어 1번에서 하이퍼쓰레드에 의해 생성된 논리적인 코어를 사용한다는 의미이다. Core set 1은 모두 다른 물리 코어를 할당 받았지만, TX 파이프라인이 다른 NUMA 소켓의 코어를 사용한다. Core set 2와 3은 각각 RX와 TX, Validity와 NAT가 동일한 물리 코어를 할당한 경우이다.

이들에 대해 성능을 측정한 결과, 그림 9와 같이 모두 다른 패킷 처리량을 보였다. Core set 1은 다른 NUMA 소켓의 코어를 사용하기 때문에 메모리 접근 속도가 낮아져 가장 낮은 처리율을 보인다. 반면, Core set 2와 3을 비교해보면, 같은 NUMA 소켓을 사용하더라도 같은 물리 코어를 사용하는 파이프라인들의 작업량에 따라 성능이 달라지는 것을 확인할 수 있다.

본 논문은 파이프라인의 작업량에 따른 성능 변화를 더 자세히 알아보기 위해서, 같은 NUMA 소켓의

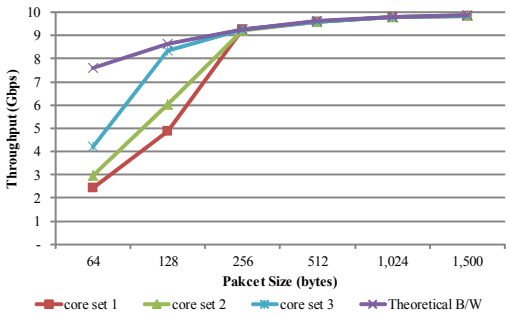

그림 9. 프로세서 코어 별 uNATp의 성능 변화
Fig. 9. Performance of uNATp by processor allocations

표 3. 프로세서 코어 할당 정보
Table 3. Allocated processor core for each pipeline

	RX	Validity	NAT	ConnTrack	TX
Core set 1	s0c0	s0c1	s0c2	s0c3	s1c1
Core set 2	s0c0	s0c1	s0c2	s0c3	s0c0h
Core set 3	s0c0	s0c1	s0c1h	s0c2	s0c3

물리 코어만 활용하는 환경에서 각 파이프라인의 조합을 바꾸면서 동일한 물리 코어를 할당하는 실험을 진행하였다. 이 실험에서 사용한 물리 코어의 할당 정보는 표 4와 같다. 표에 나타난 "Val+NAT"와 같은 이름은 코어 할당 정보를 구분하기 위한 것으로, 같은 물리 코어를 사용하는 파이프라인의 이름을 의미한다. 즉, "Val+NAT"는 Validity와 NAT 파이프라인이 같은 물리 코어를 사용함을 의미한다. 또한, 이 실험에서는 하이퍼쓰레드에 의한 논리적인 코어의 영향을 제거하기 위해 하이퍼쓰레드 기능을 비활성화한 상태에서 진행하였다. 따라서 표에서 강조된 코어 정보들은 실제로 물리 코어를 함께 사용하는 경우를 의미한다.

그림 10은 표 4의 코어 할당 정보에 따라 파이프라인을 동작했을 때 주소변환에 따른 패킷 처리량을 나타낸다. 실험에 사용한 패킷의 크기는 패킷의 최소 크기인 64 바이트이고, 하향 패킷에서 NAT 파이프라인을 거치지 않기 때문에 모든 파이프라인을 통과하는 상향 패킷만을 대상으로 한다.

실험 결과에 의하면, Val+NAT의 경우, 4.3Gbps로 가장 높은 패킷 처리량을 보인다. 이는 Validity 파이프라인의 작업량이 다른 파이프라인에 비해 적기 때문이다. Validity 파이프라인은 패킷의 특정 헤더 부분(예: CRC)을 읽는 역할만 하기 때문에 테이블 등록이나 패킷 헤더 수정 등의 작업이 없어 작업량이 다른 파이프라인에 비해 적다. 반면, 가장 낮은 처리율을 보인 경우는 NAT+TX이다. 이는 TX의 작업량이 다른 파이프라인보다 많기 때문에 나타나는 현상이다. TX 파이프라인은 수정된 헤더 정보를 바탕으로 CRC를 재계산하고 변경된 CRC를 다시 패킷 헤더에 추가하는 작업을 하기 때문에 프로세서 사용량이 많다. 이로 인해, 다른 파이프라인과 TX가 같은 프로세서에서 동작하게 되면 전체적으로 주소변환에 따른 패킷 처리량이 낮아진다. 이처럼, 파이프라인 구조의 CGNAT는 기존의 커널이나 순차적 처리 구조에 비해 높은 주소 변환 성능을 달성할 수 있는 반면, 물리적인 프로세서의 할당에 따라 성능 차이가 발생한다. 따라서 각 파이프라인의 특성과 작업량을 고려하여 프로세서를 할당하는 정책이 뒷받침되어야 한다.

표 4. 동일 NUMA 소켓에서의 코어 할당
Table 4. Processor core allocation in the same NUMA socket

	RX	Validity	NAT	ConnTrack	TX
Val+NAT	s0c0	s0c1	s0c1	s0c2	s0c3
Val+CT	s0c0	s0c1	s0c2	s0c1	s0c3
Val+TX	s0c0	s0c1	s0c2	s0c3	s0c1
NAT+CT	s0c0	s0c1	s0c2	s0c2	s0c3
NAT+TX	s0c0	s0c1	s0c2	s0c3	s0c2
RX+Val	s0c0	s0c0	s0c1	s0c2	s0c3
RX+NAT	s0c0	s0c1	s0c0	s0c2	s0c3
RX+TX	s0c0	s0c1	s0c2	s0c3	s0c0

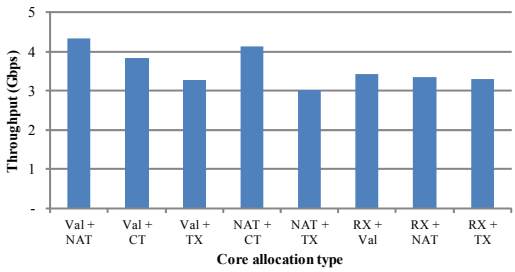

그림 10. 동일 코어 상에서 파이프라인 부하에 따른 성능 변화
Fig. 10. Performance by loads of pipelines on the same physical core

V. 결론 및 향후 계획

본 논문에서는 캐리어급 주소변환을 위한 프레임워크를 설계하고 프로토타입 구현을 통해 성능 평가를 수행하였다. 전통적인 리눅스 커널(iptables)을 사용하는 경우에 비해 최대 15.5배의 패킷 처리율 향상이 관찰되었고, 프로세서 할당 실험을 통해 파이프라인에 대한 프로세서의 할당에 따라 전체적인 CGNAT의 성능이 변화한다는 사실을 확인할 수 있었다.

향후에는 네트워크 가상화 기술을 적용하여 가상머신(Virtual Machine)상에서 실행되는 CGNAT 서버의 성능을 평가하고 하이퍼바이저 계층에서 제공할 수 있는 성능 개선 방법에 관한 연구를 진행할 계획이다.

References

[1] B. Carpenter, *Middleboxes: taxonomy and issues*, RFC 3234, 2002.

[2] J. Yang, H. Park, Y. Kim, and J. Choi, "A virtual object hosting technology for IoT device controlling on wireless AP's," *J. KICS*, vol. 39, no. 2, pp. 164-172, Feb. 2014.

[3] S. Perreault, Ed., *Common requirements for carrier-grade NATs (CGNs)*, RFC 6888, 2013.

[4] T. Jeffree, *IEEE draft standard for local and*

metropolitan area networks, virtual bridged local area networks, amendment 4: Provider bridges, IEEE 802.1ad, 2005.

[5] B. Leslie, P. Chubb, N. Fitzroy-Dale, S. Gotz, C. Gray, L. Macpherson, D. Potts, Y. Shen, K. Elphinstone, and G. Heiser, "User-level device drivers: achieved performance," *J. Comput. Sci. Technol.*, vol. 20, no. 5, pp. 654-664, Sept. 2005.

[6] J. H. Salim, R. Olsson, and A. Kuznetsov, "Beyond softnet," in *Proc. Linux 2.5 Kernel Developers Summit*, San Jose, CA, USA, Mar. 2001.

[7] DPDK, Retrieved July 5, 2016, from http://dpdk.org.

[8] J. Hwang, K. K. Ramakrishnan, and T. Wood. "NetVM: high performance and flexible networking using virtualization on commodity platforms," *IEEE Trans. Netw. Serv. Management*, vol. 12, no. 1, pp. 34-47, 2015.

[9] G. Pongrácz, L. Molnar, and Z. L. Kis, "Removing roadblocks from SDN: OpenFlow software switch performance on Intel DPDK," in *Proc. 2nd Eur. Wksp. Softw. Defined Netw.*, pp. 62-67, Berlin, Germany, Oct. 2013.

[10] I. Cerrato, M. Annarumma, and F. Risso, "Supporting fine-grained network functions through Intel DPDK," in 3*rd Eur. Wksp. Softw. Defined Netw.*, pp. 1-6, Budapest, Hungary, Sept. 2014.

[11] OPNFV, Retrieved July 5, 2016, from https://www.opnfv.org.

[12] ODP, Retrieved July 5, 2016, from http://www.opendataplane.org.

[13] M. Dobrescu, N. Egi, K. Argyraki, B. G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "Routebricks: exploiting parallelism to scale software routers," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Principles*, pp. 15-28, Big Sky, Montana, USA, Oct. 2009.

[14] S. Han, K. Jang, K. Park, and S. Moon, "Packetshader: a gpu-accelerated software router," *ACM SIGCOMM Computer Commun. Rev.*, vol. 40, no. 4, pp. 195-206, Oct. 2010.

[15] T. Brecht, G. J. Janakiraman, B. Lynn, V. Saletore, and Y. Turner, "Evaluating network processing efficiency with processor partitioning and asynchronous I/O," *ACM SIGOPS Operating Syst. Rev.*, vol. 40, no. 4, pp. 265-278, Oct. 2006.

[16] R. Bolla and R. Bruschi, "PC-based software routers: high performance and application service support," in *Proc. ACM Wksp. Programmable routers for extensible services of tomorrow*, pp. 27-32, Seattle, WA, USA, Aug. 2008.

[17] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proc. 11th USENIX Symp. NSDI 14*, pp. 459-473, Seattle, WA, USA, Apr. 2014.

[18] The Pktgen Application, Retrieved July 6 2016, from https://pktgen.readthedocs.io/en/latest/.

[19] T. Ahn, Y. Kim, and S. Lee, "Dynamic resource allocation in distributed cloud computing," *J. KICS*, vol. 38, no. 7, pp. 512-518, Jul. 2013.

[20] H. Kim and H. Kim, "Control algorithm for virtual machine-level fairness in virtualized cloud data center," *J. KICS*, vol. 38, no. 6, pp. 512-520, Jun. 2013.

[21] A. Drebes, A. Pop, K. Heydemann, N. Drach, and A. Cohen, "NUMA-aware scheduling and memory allocation for data-flow task-parallel applications," in Proc. *21st ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, no. 44, Barcelona, Spain, Mar. 2016.

이 문 상 (Moon-Sang Lee)

1996년 : KAIST 전산학과 학사
1998년 : KAIST 전산학과 석사
2006년 : KAIST 전산학과 박사
2002년~2003년 : EPG 슈퍼컴퓨팅 연구소 책임연구원
2005년~2012년 : 삼성전자 DMC 연구소 책임연구원
2012년~현재 : KT 융합기술원 책임연구원
<관심분야> 클라우드, 가상화, NFV, SDN

김 우 태 (Wootae Kim)

1999년 : 경북대학교 전자공학과 학사
2001년 : 경북대학교 전자공학과 석사
2002년~현재 : KT 융합기술원 수석연구원
<관심분야> Virtual Edge Infra, NFV, Netconf/YANG 모델링

이 치 영 (Chiyoung Lee)

2006년 : 충남대학교 전기전자통신공학부 컴퓨터전공 학사
2008년 : 고려대학교 컴퓨터학과 석사
2015년 : 고려대학교 컴퓨터전파통신공학과 박사
2016년~현재 : KT 융합기술원 전임연구원
<관심분야> 네트워크 가상화, NFV, vEdge

이 영 우 (Young-Woo Lee)

1984년 : 숭실대학교 전자공학과 학사
1990년 : 숭실대학교 전자공학과 석사
2005년 : 충남대학교 컴퓨터공학과 박사
1990년~현재 : KT 융합기술원 상무
<관심분야> NMS, SDN, NFV, 클라우드