www.kiise.or.kr

ISSN 1229-6821

제39권 제11호 통권 제390호

정보과학회지

Communications of the Korean Institute of Information Scientists and Engineers

2021. 11

매니코어 시스템과 소프트웨어

한국정보과학회
Korean Institute of Information Scientists and Engineers

정보과학회지

2021. 11
제39권 제11호
통권 제390호

목차

Communications of KIISE

November 2021

Vol. 39 No. 11
Serial No. 390

Contents

매니코어에서 가상화된 네트워크 서비스 성능 개선

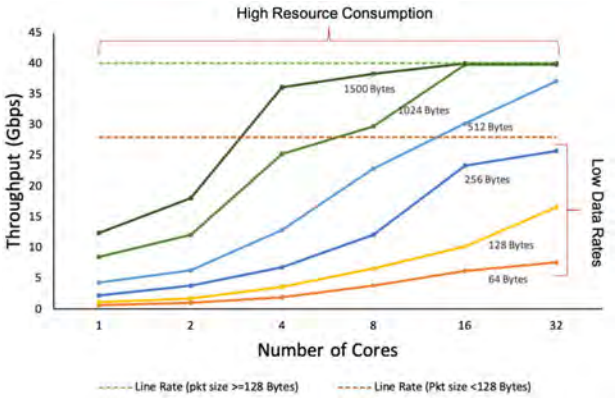
한국전자통신연구원 ▌차승준
데브스택 ▌송정호

1. 서 론

검색, 소셜 네트워크, 전자상거래 등 서비스를 제공하는 데이터센터의 애플리케이션은 일반적으로 수백 개의 마이크로 서비스로 동작한다[1]. 즉, 사용자의 요청은 마이크로 서비스화하기 위해 수천 개의 원격 프로시저 호출(RPC, remote procedure call)로 바뀌어 처리된다. 이러한 환경에서 빠른 통신 및 서비스를 지원하기 위해 40G/100G를 지원하는 네트워크 어댑터를 사용한다. 또한, 기존의 TCP/IP 처리 외에도 RDMA(remote direct memory access) 기능을 지원하는 네트워크 카드가 도입되고 있다.

스위치, 라우터, 미들박스와 같은 네트워크 장비들은 특정 하드웨어에서 최적화된 소프트웨어를 사용하여 고성능 패킷 처리가 가능하다. 하지만, 현재의 데이터센터는 수천 개의 가상머신과 컨테이너를 통해 서비스를 제공하기 때문에 물리적인 네트워크와 상관없이 자체적인 토폴로지를 구성하고 네트워크를 지원한다. 따라서 하드웨어 기반의 네트워크 성능 향상은 데이터센터에서의 유연성과 확장성을 충족시키기 어렵다.

데이터센터에서 유연성과 확장성을 지원하기 위해 네트워크 기능 가상화(NFV, network function virtualization)가 연구되었다. 네트워크 기능 가상화는 하드웨어로 제공되는 네트워크 기능을 소프트웨어로 구현한 기술로 일반 소프트웨어와 같이 운영체제 커널의 기능을 사용하여 통신을 제공한다. 하지만 현재 사용되는 네트워크 및 프로토콜 스택으로 인해 데이터센터에서 많은 대기시간이 발생한다. 데이터센터의 시스템 소프트웨어에 의해 대기시간이 랙 사이에는 66%, 랙 안에서는 81% 발생한다[2]. 이는 네트워크 및 프로토콜 스택이 빠른 통신에 최적화되지 않았기 때문은 오버헤드가 발생하여 네트워크 성능을 모두 사용하지 못하기 때문이다. 이와 같은 환경에서 높은 네트워크 성능을 얻기 위해서는 더 많은 코어 자원이 필요하다.



〈자료〉 참고문헌 [11]의 그림을 재구성

그림 1 코어 수와 패킷 크기에 대한 패킷 처리 성능 차이

그림 1에서는 40G 네트워크 환경에서 코어 수와 패킷 크기에 따른 네트워크 성능을 보인다[3]. 패킷 크기가 작은 경우(256바이트 이하) 코어 수에 상관없이 네트워크의 성능을 지원하지 한다. 패킷 크기가 큰 경우(256바이트 초과)에는 적은 코어(4코어 미만)의 경우 네트워크의 성능을 여전히 지원하지 못하지만, 많은 코어(4코어 이상)의 경우에는 네트워크의 모든 성능을 지원할 수 있음을 확인하였다.

엑사스케일 컴퓨팅 환경이 발전함에 따라 이를 지원하는 프로세서도 기존 수십 개의 코어로 구성된 멀티코어에서 수백~수천 개의 코어로 구성된 매니코어 환경으로 발전하고 있다. 프로세서의 발전과 함께 컴퓨팅 시스템이 점점 더 강력해짐에도 불구하고, 빠른 네트워크의 지원하기 위해 많은 코어를 사용해야 하기 때문에 연산에 활용되는 코어는 작아진다. 매니코어 환경에서 빠른 네트워크를 지원하기 위해 소프트웨어 수준에서의 최적화 연구가 필요하다. 본 논문에

† 이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임[No. 2014-3-00035, 매니코어 기반 초고성능 스케일러블 OS 기초연구(차세대 OS 기초연구센터)]

서는 이와 같은 소프트웨어 수준의 최적화를 연구한 커널 네트워크 스택의 최적화, 프로토콜 스택의 최적화, SFC 최적화에 관한 기술을 소개한다.

2. 최신 동향 및 관련 연구 기술

네트워크 서비스 성능을 개선하기 위한 연구는 1) 커널 일괄 처리, 2) 커널 우회 3) 확장의 세 가지로 구분된다. 커널 일괄 처리 방법은 기존 커널 프로토콜에서 사용하는 것으로, 관련된 시스템콜을 한꺼번에 배치처리하는 방식으로 시스템콜의 오버헤드를 줄이는 방법이다. 최신 커널 프로토콜 스택인 IX[4]와 ZygOS[5]에서 적응형 일괄 처리를 포함하는 이벤트 기반 시스템콜 인터페이스를 지원한다. 하지만 이러한 일괄 처리 기법은 시스템콜의 오버헤드를 줄일 수 있지만, 오히려 지연시간을 증가시키는 단점이 있다.

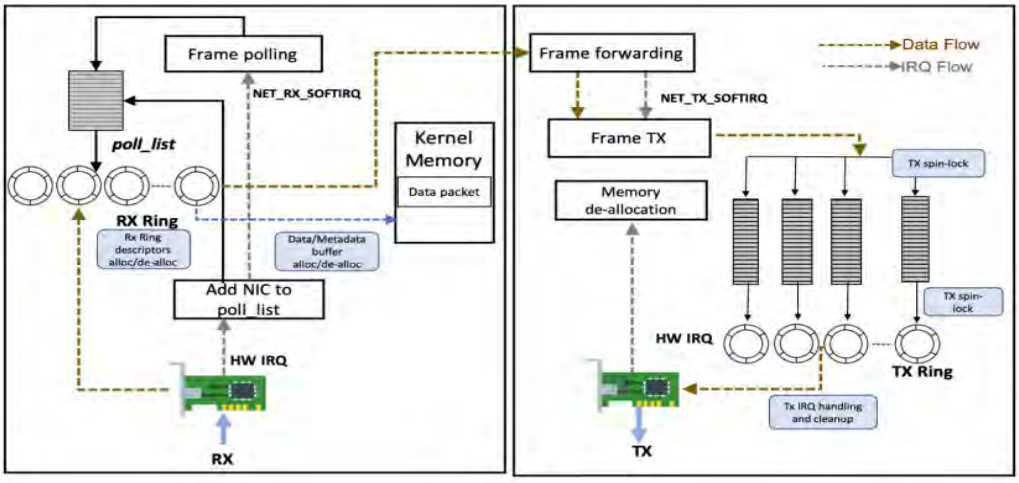
커널 우회 방법은 관련된 시스템콜을 처리하는데 커널의 거치지 않고 직접 디바이스에 접근하는 기법으로, 대표적으로 DPDK(mTCP)[6]가 있다. 이는 시스템콜을 처리하는데 발생하는 지연시간을 줄일 수 있지만, 커널의 대기시간을 증가시키며, 네트워크 데이터의 캐시를 유지하지 못하는 단점을 가진다. 확장 방식은 기존에 구현된 커널의 기능을 확장하는 것이다. Plexus[7], ASH[8], Exokernel[9]에서 이미 커널 기능을 확장하여 적용하였지만, 확장성 지원이 어렵고 보안에 취약하다는 문제가 발생할 수 있다. 또한, 추가되는 하드웨어에 대해 소프트웨어 기능을 다시 구현하거나, 새로운 시스템을 개발해야 하는 문제점이 있다.

최근 많은 수의 코어를 지원하는 프로세서 기술과 높은 데이터 전송률을 지원하는 네트워크 장비의 발전하는 상황에서 네트워크 기능의 병렬화를 지원한다면 전송 속도의 성능을 지원할 수 있다[10]. 또한, iptables, netfilter, ipsec, cgroups 등과 같은 기존 커널이 가진 성숙한 기능을 사용한다면 가상화된 네트워크 기능을 더욱 쉽고 유연하게 정의할 수 있다. 따라서, 네트워크 서비스의 성능 개선을 위해 소프트웨어적인 개선이 필요하다.

3. 네트워크 스택 개선을 통한 패킷 처리 고속화 기법

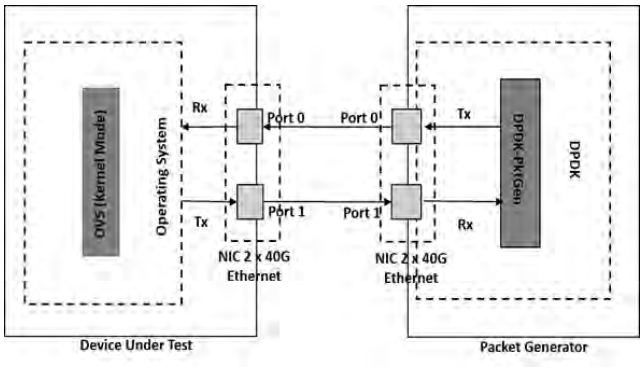
리눅스에서의 TCP/IP 스택은 그림 2와 같이 메타 데이터와 데이터 버퍼 할당 및 삭제 오버헤드, 패킷 송신 큐의 잠금 오버헤드와 빈번한 송신 인터럽트 처리로 병목 원인이 된다. mKPAC(kernel packet processing for manycore systems)은 40G 이상의 고속 네트워크에서 커널의 TCP/IP 스택에서 발생하는 병목 요인을 제거하여 커널 패킷 처리를 고속화한 연구이다 [11].

mKPAC에서 네트워크 스택의 병목을 제거하기 위해 수신(Rx)/송신(Tx) 데이터 및 메타 데이터 버퍼 할당 및 해제, 송신(Tx)/수신(Rx) 링 디스크립터 관리, 송신(Tx) 스핀락, 송신(Tx) 인터럽트 처리 및 정리(cleanup)를 개선하여, 네트워크 스택에서 발생하는 오버헤드를 줄이고 패킷 처리 성능을 개선하였다. 또한, 실험을 위해 그림 3의 오픈소스 기반 패킷 생성기 테스트베드를 구축하였다. mKPAC은 DPDK와 달리, 네트워크 서비스 제품 환경에서 이미 견고하고 안정성이 입증된 리눅스 커널의 기능을 활용한다.



〈자료〉 참고문헌 [3]의 그림을 재구성

그림 2 리눅스 커널에서의 패킷 송신(Rx)/수신(Tx) 절차 및 관련된 오버헤드



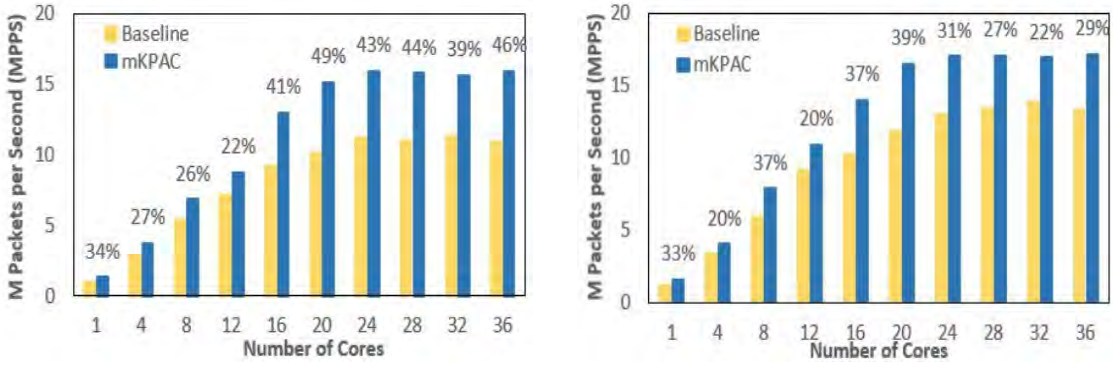
1) 하드웨어 구성
- Intel(R) Xeon(R) CPU E7-8870v2 @ 2.30GHz
- 120코어
- 32KB L1 캐시, 256KB L2 캐시, 30MB L3 캐시
- 인텔 40G Ethernet dual port

2) 소프트웨어 구성
- 리눅스 커널 4.4.1
- 커널 OVS

3) 패킷 생성기 (Packet Generator)
- 192코어
- DPDK-PktGen
- 50 Gbps UDP 트래픽

〈자료〉 참고문헌 [3]의 그림을 재구성

그림 3 40G 패킷 생성기



(a) 64바이트 패킷

(b) 128바이트 패킷

〈자료〉 참고문헌 [11]의 그림을 재구성

그림 4 코어 수와 패킷 크기 따른 mkpac 성능 비교

테스트베드를 위해 인텔 제온 CPU E7-8870v2 2.3GHz로 15코어, 8소켓으로 구성된 120코어의 매니코어 시스템을 구축하였다. 네트워크를 위해 2개의 40GbE 포트로 구성된 인텔 이더넷 컨트롤러 XL710를 설치하여 데이터의 수신과 발신에 사용하고, 데이터 플레인의 전송을 위해 OVS 커널 모듈을 사용한다. 패킷 생성기는 56코어의 서버로 DPDK-Pktgen을 사용하여 40Gbps의 트래픽을 생성한다.

그림 4는 코어 수와 패킷 크기에 따른 기존 리눅스와의 성능 비교 결과이다. 64바이트 패킷의 경우 기존 리눅스 대비 37% 성능이 향상됨을 확인할 수 있었다. 또한, 코어당 지연시간은 64바이트 패킷의 경우 23%, 128바이트 패킷의 경우 22% 감소한 것으로 측정되었다. 이러한 결과를 통해 커널을 우회하거니 특수 하드웨어 지원 없이 매니코어를 활용할 경우 성능이 향상될 수 있음을 보인 것이다. 패킷 처리 성능은 매니코어 시스템의 다른 오버헤드 및 확장성 문제를 해결하면 더욱 향상될 수 있다.

그림 5는 서로 다른 패킷 크기를 기준으로 리눅스와 비교하여 최대 속도를 얻는 데 필요한 코어 수를

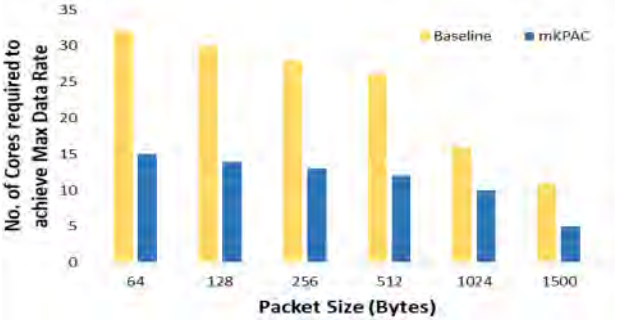
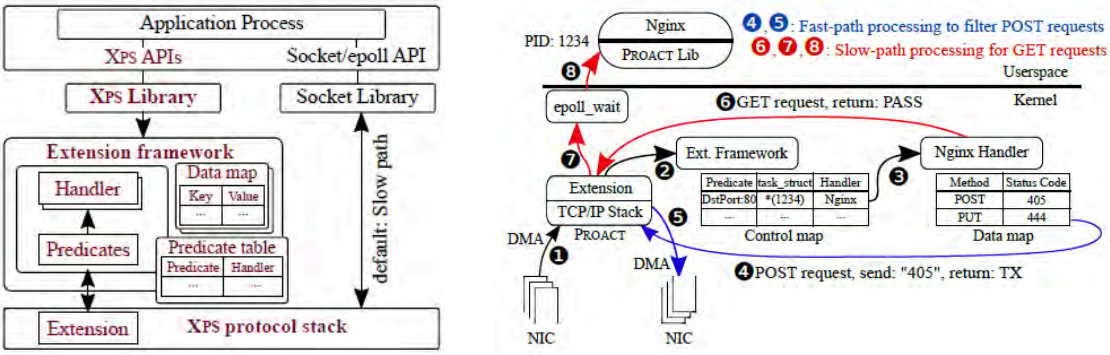


그림 5 송수신 기능 개선으로 자원 효율화

비교한 결과이다. 모든 패킷 치수에서 mKPAC이 50% 이상 필요한 코어 수가 적음을 확인하였다. 이를 통해 특히 매니코어 시스템에서 네트워크의 고속화를 위해 많은 코어를 사용하지 않음으로써, 연산에 더 많은 코어를 활용할 수 있음을 의미한다. 이는 전체 시스템 관점에서 응용 프로그램의 성능을 향상시킬 수 있다.

4. 프로토콜 스택 개선을 통한 성능 개선 연구

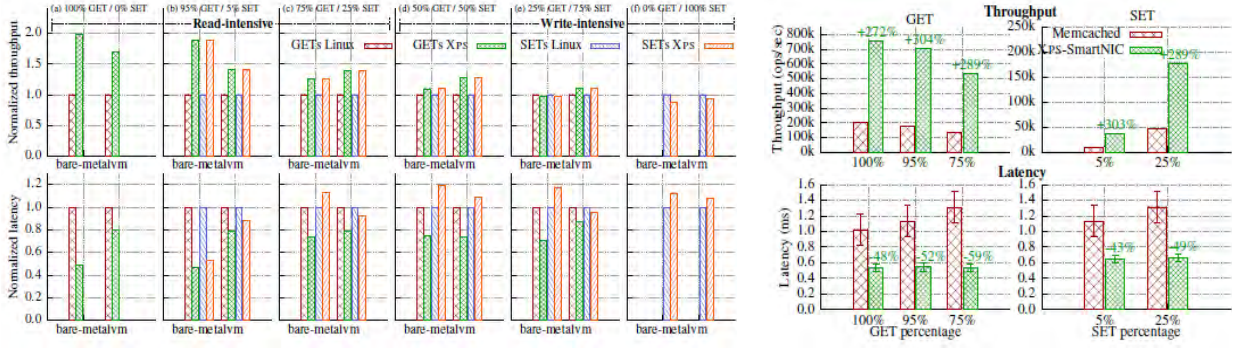
네트워크 통신을 위해 현재 TCP/IP의 소켓 인터페



(a) XPS의 구성 요소와 인터페이스 (b) Nginx에서 패킷 흐름 예제

〈자료〉 참고문헌 [12]의 그림을 재구성

그림 6 XPS의 구조와 웹서비스 흐름도



(a) read/write 성능 비교 (b) Memcached 성능 비교

〈자료〉 참고문헌 [12]의 그림을 재구성

그림 7 XPS 성능 실험 결과

이스를 사용하면 epoll() 및 read()/write()의 소켓 인터페이스 자체인 오버헤드와, 콜드 네트워크 데이터에 접근할 때의 캐시 미스로 인한 오버헤드뿐만 아닐, 커널 페이지테이블 격리(KPTI, kernel page table isolation)와 같은 커널 기능으로 인해 오버헤드가 발생한다. XPS (eXtensible Protocol Stack)은 TCP와 UDP와 같은 프로토콜 스택에서 발생하는 소켓 오버헤드를 줄이기 위해 연구이다 [12].

그림 6은 XPS의 구조와 웹서비스의 흐름을 나타낸다. XPS는 라이브러리, 확장 프레임워크, 프로토콜 스택의 세 가지로 구성된다. 확장 프레임워크는 예측 테이블(predicate table)과 패킷 처리 프레임워크가 포함되어 있으며, 애플리케이션 핸들러(handler)를 추가할 수 있다. 이를 통해 프로토콜 스택에서의 오버헤드는 데이터 복사, 모드 스위칭, 캐시 미스 등에서 발생하며, XPS는 이를 해결하여 시간에 민감한 연산을 프로토콜 처리 이후에 즉시 실행할 수 있는 추상화된 기능을 제공한다. 이는 일반적인 운영체제의 프로토콜 스택이나, mTCP와 같은 유저 스페이스 프로토콜 스택, 최근의 스마트 NIC에도 적용이 가능하다. 특히 다른 모든 연산은 기존의 소켓 계층을 사용하면서 시간에 민감한 작업을 위해 기능을 확장하였다.

XPS는 키밸류 스코어에서의 캐싱, 웹서버에서 HTTP 요청의 제한 및 필터링, 분산 시스템에서 하트비트 처리의 3가지 실제 서비스를 통해 성능을 확인하였다. 그림 7은 XPS를 웹서비스와 DB 서비스에서 실험한 결과이다. XPS는 Redis 키밸류 스토어[13]에서 처리량을 98.1% 향상시키고 지연시간이 73.3% 줄어듦을 확인하였다. Nginx 웹서버[14]의 경우 최대 2.2배와 82%까지 성능 향상을 보였다. IX[4]와 ZygOS[5]에 비해 XPS-mTCP는 Redis의 처리량이 최대 50.1% 향상되며, 꼬리 응답시간이 63.5% 향상시킨다.

5. 서비스 기능 체이닝 구성 시 네트워크 서비스 간 성능 개선

네트워크 기능 가상화(NFV, network function virtualization)는 네트워크 기능들을 전용 하드웨어 장비 대신 소프트웨어 기반의 범용 서버에 가상화하는 기술로, 그림 8과

같이 가상 네트워크 기능 요소(VNF, virtual network functions)들로 구성된다. 이처럼 가상화를 기반으로 소프트웨어적으로 VNF를 구성하므로, 운용하고자 하는 구성 목적에 따라 네트워크 기능들을 유연하게 구성할 수 있으며, 이에 따라 하드웨어를 직접 구성하지 않아도 되기 때문에 전체 구성 비용이 절감된다.

NFV 기술이 도입되면서 네트워크 서비스 기능이 전달 경로상에 존재하는 형태가 아니라 자신이 필요로 하는 네트워크 서비스 기능이 존재하는 곳을 경유하는 형태로 되게 된다. 이와 같이 특정 서비스를 위해 필요한 서비스 기능들과 이들 간의 적용 순서를 추상화한 것이 서비스 기능 체이닝(SFC, service function chaining)이다. 가상화 기반으로 소프트웨어적으로 VNF 구성할 경우, 네트워크 성능 저하 및 CPU 성능 과부하가 생길 수 있게 되는데, 이를 해결하면 SFC 전체 네트워크 스택의 성능을 향상시킬 수 있다.

VNF는 하드웨어로 고정되지 않고 소프트웨어적으로 유연하게 구성할 수 있는 장점이 있지만, 커널 네트워크 스택에서 처리되어 사용자 프로그램을 통해 네트워크 기능을 처리해야 하므로 네트워크 성능 저하되고, CPU 성능에 과부하가 발생한다. 이러한 VNF의 성능적 문제점을 해결하려고 했던 연구로는 eVNF[15, 16]가 있으며, eVNF는 eBPF(extended Berkeley Packet Filter)의 네트워크 버전인 XDP(eXpress Data Path, 그림 9)로 VNF를 구현하여 VNF의 네트워크 성능 저하 및 CPU 성능 과부하의 문제점을 해결하였다.

단일 루트 I/O 가상화(SR-IOV, single root I/O virtualization)은 하나의 네트워크 디바이스를 가상화 기술을 통해 여러 개의 네트워크 카드로 만들어 가상머신에서 사용하게 하는 기술이다. SR-IOV를 사용하면 가상머신이 호스트 머신의 네트워크 인터페이스 카드에 직접 연결된 것처럼 동작하여 I/O 성능이 향상된다. 본 연구에서는 가상머신 환경에서 eVNF 기반으로 SFC를 구성하는 데 있어서 SR-IOV 기술을 적용하여 가상화 단계에서 발생할 수 있는 네트워크 성능 저하를 최소한으로 함으로서, NFV의 유연함과 구성 비용 절감 효과를 가져오면서 성능 저하는 최소한 줄일 수 있는 방안을 제시한다(그림 10).

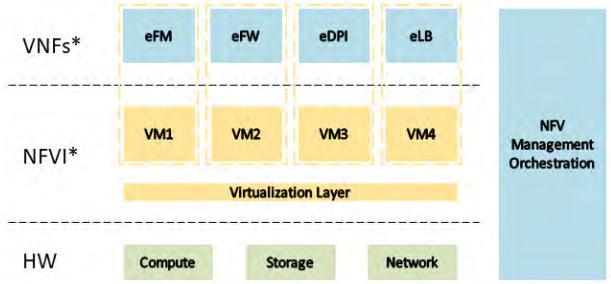
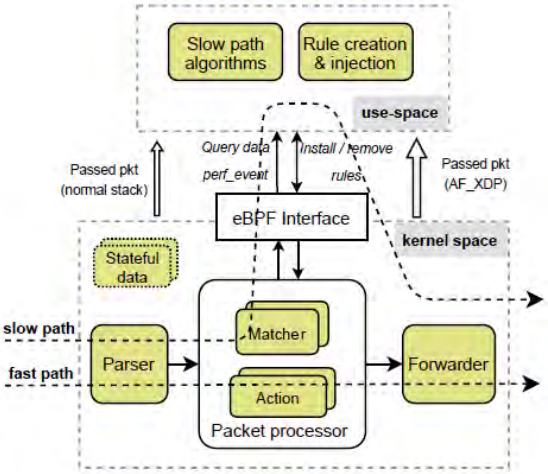


그림 8 NFV 아키텍처



〈자료〉 참고문헌 [16]의 그림을 재구성

그림 9 eVNF: XDP를 적용한 VNF

eVNF와 SR-IOV를 적용한 SFC의 성능 개선을 확인하기 위해 그림 11과 같이 테스트베드를 구축하였다. 테스트베드 1은 SFC를 구성한 시스템으로, eVNF 기반으로 방화벽(eFW), 플로우모니터(eFM), 로드밸런서(eLB)로 SFC를 구상하였다. 프론트엔드에는 SFC, 백엔드에는 웹서버가 동작한다. 테스트베드 2는 클라이언트 노드로 패킷을 생성하여 전달한다. 이를 위한 벤치마크로는 iperf3[17]과 wrk[18]를 사용하였다. 테스트베드는 인텔 XL710 40G 네트워크 카드 기반으로 연결하였다. 비교를 위해 VNF 기반 SFC에서 방화벽은 리눅스의 iptables를 사용하였고, 플로우모니터는 eFM을 참고하여 구현하였다.

그림 12는 벤치마크별로 성능을 실험한 결과이다. iperf3 벤치마크에서는 eVNF 기반 SFC는 VNF 기반 SFC에 비해 네트워크 성능이 26%~331% 향상되었으며, 17%~62%의 CPU 사용률 감소를 확인하였다. wrk 벤치마크에서는 37%~361%의 네트워크 성능 향상 및 13%~31%의 CPU 사용률 감소를 확인하였다. 두 벤치마크 모두 eVNF기반 SFC에서는 모든 vCPU(1개~8개)에서 비슷한 네트워크 성능을 확인하였지만, VNF 기반 SFC에서는 vCPU 개수에 따라 큰 폭의 네트워크 성능 저하를 확인하였다. 1개의 vCPU 실험에서 가장 높은 네트워크 성능 차이, 2개의 vCPU 실험에서 가장 높은 CPU 사용량 차이가 있었다. 이러한 성능 차

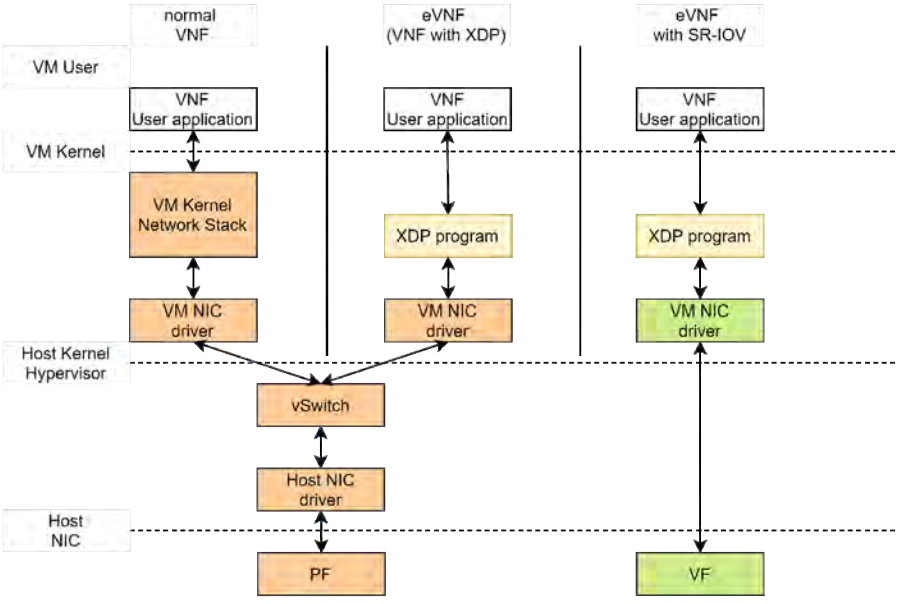


그림 10 가상 머신 환경에서의 VNF 구성별 구조도

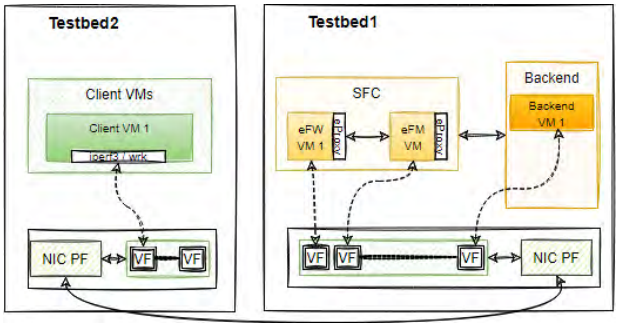
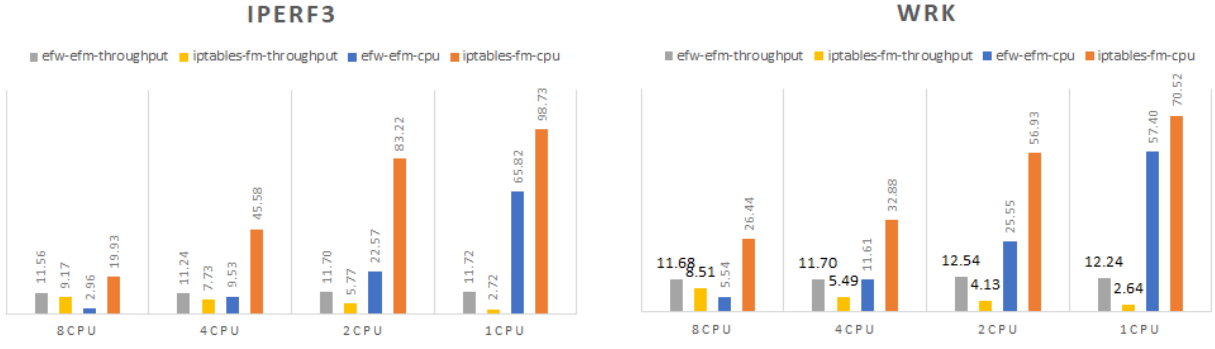


그림 11 SFC 실험 환경

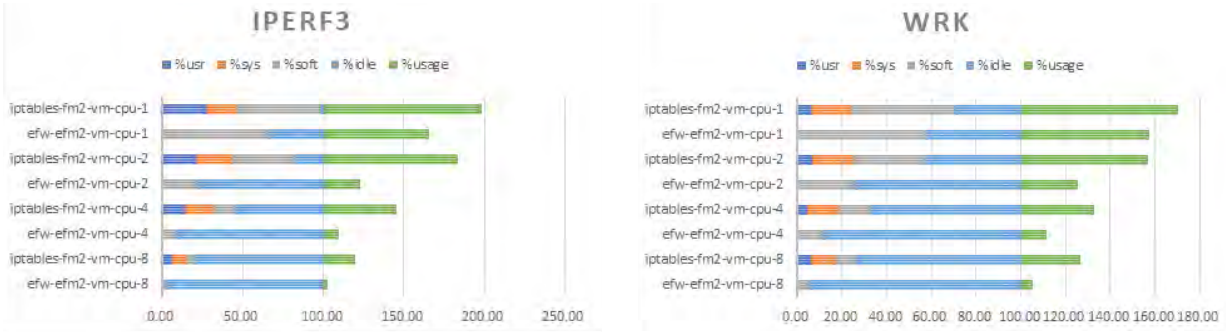
이는 커널 네트워크 스택을 거치지 않는 XDP의 특성 때문인데, SFC를 거쳐 가는 패킷이 해당 VNF가 구현된 가상머신의 커널 네트워크 스택을 거치지 않고 네트워크 드라이버 단에서 처리되어 진행되기 때문이다.

그림 13은 CPU 사용량을 세분화하여 분석한 결과이다. eVNF와 VNF의 가장 큰 차이는 %sys(커널 영역) %usr(사용자 영역)의 CPU 사용량의 존재 여부다. XDP를 적용한 eVNF의 경우 패킷이 네트워크 드라이버 단에서 바로 처리되기 때문에 커널 영역 및 유저 영역에서의 CPU가 사용되지 않는다. 이 차이로 eVNF기반 SFC에서는 비교적 적은 vCPU에서도 일정한 네트워크 성능을 내면서도 CPU 사용량이 낮게 되는 결과를 보여 준다. 이를 통해 매니코어 시스템에서 네트워크의 성능을 보장하기 위해 많은 코어를 사용하지 않음으로써, 연산에 더 많은 코어를 활용할 수 있음을 의미한다. 이를 통해 매니코어 시스템의 전체적인 성능을 향상시킬 수 있다.



(a) iperf3 벤치마크 (b) wrk 벤치마크

그림 12 iperf3과 wrk 벤치마크를 통한 네트워크, CPU 성능 비교 결과



(a) iperf3 벤치마크 (b) wrk 벤치마크

그림 13 iperf3과 wrk 벤치마크에서 CPU 사용률 비교 결과

5. 결 론

소프트웨어 기반의 네트워크 성능 가속화 기술은 다양한 분야에 활용될 수 있다. 많이 사용되는 응용인 웹서비스, 데이터베이스 분야에서는 서비스 지연시간을 줄이면서 최적의 성능으로 서비스를 제공할 수 있다. 연구된 네트워크 가상화 기술은 최근 주목받고 있는 분야인 에지 컴퓨팅과 NFV 플랫폼 성능 향상에 활용할 수 있다.

참고문헌

[1] Starting the Avalanche, https://netflixtechblog.com/starting-the-avalanche-640e69b14a06

[2] Gao, Peter X., et al. “Network requirements for resource disaggregation.” 12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016.

[3] Ramkeen, et al. “FENCE: Fast, ExteNsible, and ConsolidatEd Framework for Intelligent Big Data Processing.” IEEE Access 8 (2020): 125423-125437.

[4] Belay, Adam, et al. “IX: A Protected Dataplane Operating System for High Throughput and Low Latency.” 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014.

[5] Prekas G., Marios K., and Edouard B.. “Zygos: Achieving low tail latency for microsecond-scale networked tasks.” Proceedings of the 26th Symposium on Operating Systems Principles. 2017.

[6] DPDK, http://www.intel.com/content/www/us/en/communications/data-plane-development-kit.html, 2017.

[7] Fiuczynski, Marc E., and Brian N. Bershad. “An Extensible Protocol Architecture for Application-Specific Networking.” USENIX Annual Technical Conference. 1996.

[8] Wallach, Deborah A., Dawson R. Engler, and M. Frans Kaashoek. “ASHs: Application-specific handlers for high-performance messaging.” Conference proceedings on Applications, technologies, architectures, and protocols for computer communications. 1996.

[9] Engler, Dawson R., M. Frans Kaashoek, and James O'Toole Jr. “Exokernel: An operating system architecture for application-level resource management.” ACM SIGOPS Operating Systems Review 29.5 (1995): 251-266.

[10] Boyd-Wickizer, Silas, et al. “An Analysis of Linux Scalability to Many Cores.” OSDI. Vol. 10. No. 13. 2010.

[11] Ramneek, Kumar M., Taesoo K., and Sungin J. "mKPAC: Kernel packet processing for manycore systems." Proceedings of the 19th International Middleware Conference (Posters). 2018.

[12] Kumar M. “Taming latency in data center applications.” Diss. Georgia Institute of Technology, 2019.

[13] Redis Key-Value Store, https://redis.io/, 2017.

[14] Nginx Web Cache, https://www.nginx.com/resources/wiki/, 2017.

[15] Van Tu, Nguyen, Jae-Hyoung Yoo, and James Won-Ki H. “Accelerating Virtual Network Functions with Fast-Slow Path Architecture using eXpress Data Path.” IEEE Transactions on Network and Service Management 17.3 (2020): 1474-1486.

[16] eVNF, https://github.com/dpnm-ni/ni-evnf, 2020.

[17] Mortimer M. “iperf3 Documentation.” 2018.

[18] Yasukata K., et al. “StackMap: Low-Latency Networking with the OS Stack and Dedicated NICs.” 2016 USENIX Annual Technical Conference (USENIX ATC 16). 2016.

약 력

차 승 준

2006 충남대학교 컴퓨터공학과 졸업(학사)
2013 충남대학교 대학원 컴퓨터공학과 졸업(석사, 박사)
2011~2013 충남대학교 소프트웨어연구소
2013~현재 한국전자통신연구원 선임연구원
관심분야 : 시스템 소프트웨어, 운영체제, 데이터베이스, 클라우드컴퓨팅
Email : seungjunn@etri.re.kr

송 정 호

2009 KAIST 전산학과 졸업(학사)
2011 KAIST 전산학과 졸업(석사)
2020 KAIST 전산학과(박사수료)
2020~현재 데브스택 플랫폼 연구실
관심분야 : 클라우드 컴퓨팅, 시스템 소프트웨어, 시스템 보안, 내장형 시스템
Email : jhsong@devstack.co.kr