논문 19-44-01-22 The Journal of Korean Institute of Communications and Information Sciences '19-01 Vol.44 No.01
https://doi.org/10.7840/kics.2019.44.1.158

컨테이너 네트워킹 기술의 성능비교

박 영 기◆, 양 현 식*, 김 영 한°

Performance Comparison of Container Networking Technologies

Young-ki Park◆, Hyun-sik Yang*, Young-han Kim°

요 약

클라우드 환경에서 기존의 가상머신보다 경량화 되어 마이크로 서비스 및 포그 컴퓨팅 환경에 적합한 컨테이너 기술에 대한 관심이 증대되고 있다. 클라우드 환경에서 컨테이너 서비스 제공을 위한 연구와 함께 컨테이너간 서비스 연결을 위해 CNM(Container Network Model) 및 CNI(Container Network Interface) 구조가 제안되었으며, 이를 기반으로 한 다양한 네트워크 기술들이 개발되고 있다. 최근에는 클라우드와 컨테이너 환경 간 네트워크 연동을 위해 CNI 기반 네트워킹 제공구조가 제안되었으나, 제공환경에 따라 성능 차이가 발생한다. 최적화된 네트워킹 기술 적용을 위해서는 서비스 제공환경에 따른 적합한 네트워킹 구조 설계 및 구현을 통한 성능 검증이 필요하다. 본 논문에서는 클라우드 환경 내 CNI 기반 네트워크 적용을 위한 구조를 설계하고 각 네트워크 서비스 타입별 드라이버 및 가속화 환경에 따른 성능 측정을 하였다. 검증을 위해 오픈스택 및 Kubernetes를 활용하여 환경을 구성하였다. 성능저하 구간 확인 및 원인 분석 결과는 클라우드 환경의 컨테이너 시스템 구축에 반영할 수 있다.

Key Words : Container Network, CNI, CNM, Kubernetes, Kuryr, multus

ABSTRACT

The interest in container technology that it is more suitable for micro service and fog computing since it is lighter than existing virtual machines has been increased in cloud environment. Container Network Model (CNM) and Container Network Interface (CNI) architecture have been proposed for providing container services in the cloud environment, and various network technologies based on them have been developed. Recently, CNI-based networking architecture has been proposed for network interworking between cloud and container environment, but there is a difference in performance depending on the providing environment. To provide the optimized networking technology, it is necessary to verify the performance by designing and implementing a suitable networking architecture according to the service environment. In this paper, we designed network architecture for applying CNI based network, and measured performance according to driver and acceleration technology. We constructed testbed using OpenStack and Kubernetes. The performance degradation section and the cause analysis result can be reflected in the construction of the container system in the cloud environment.

※ 본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 대학ICT연구센터육성지원사업의 연구와 (IITP-2018-2017-0-01633) 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. 2015-0-00575, 글로벌 SDN/NFV 공개 소프트웨어 핵심 모듈/기능 개발)

◆ First Author : (ORCID:0000-0002-4448-7901)Soongsil University School of Electronic Engineering, red1028@ssu.ac.kr, 학생회원
° Corresponding Author : (ORCID:0000-0002-1066-4818)Soongsil University School of Electronic Engineering, younghak@ssu.ac.kr, 종신회원
* (ORCID:0000-0002-5359-4653)Soongsil University School of Electronic Engineering, hyunchic86@ssu.ac.kr, 학생회원

논문번호 : 201810-320-D-RN, Received October 10, 2018; Revised November 28, 2018; Accepted December 2, 2018

Ⅰ. 서 론

클라우드 환경에서 기존의 가상머신보다 경량화 되어 마이크로 서비스 및 포그 컴퓨팅 환경에 적합한 컨테이너기술에 대한 관심이 증대되고 있다[1,2]. 컨테이너 서비스 제공방법은 물리서버에서 직접 제공되는 native 방법 혹은 VM(Virtual Machine) 에서 제공되는 nested 방법이 있다[3]. 최근에는 두 가지 방법을 이용해 동시에 제공하는 방법도 제안되었으며, 이와 함께 네트워킹 구성이 다양해지면서 컨테이너 네트워킹 기술도 중요해 지고 있다. 컨테이너 네트워킹 기술은 CNM(Container Network Model) 및 CNI(Container Network Interface) 기술이 대표적이다[4].

CNM은 컨테이너 엔진을 제공하는 도커에서 제안한 구조로 libnetwork 기반의 네트워크 드라이버에 의한 연결성을 제공한다[5]. CNI는 CoreOS에서 제안한 기술로 Plugin API에 의한 네트워크 연결성을 제공한다[6]. 각 기술은 컨테이너 생성방법 및 운영주체에 따라 달라지며, Docker는 CNM 드라이버를 통해 네트워크 환경을 제공하며, Kubernetes 및 Apache Mesos와 같은 COE(Container Orchestration Engines)의 경우 CNI Plugin API를 통해 컨테이너 네트워크를 제공한다[7,8].

이와 같이 다양한 컨테이너 제공환경에서의 네트워크 성능에 대한 연구가 진행되고 있지만 네트워크 플러그인에 대한 성능 검증을 중심으로 연구가 진행되었다[4,9,10]. 즉, 컨테이너 및 네트워크 브릿지 연결을 위한 veth pair(Virtual Ethernet Interface Pair)기반의 Overlay 네트워크와 VLAN(Virtual LAN)과 같은 커널에서 지원하는 네트워크 타입별 드라이버, 그리고 가속화 지원 및 적용 방법에 대해서 성능 검증이 되지 않았다. 또한, VM기반의 클라우드(이하 Openstack-오픈스택)와 컨테이너 네트워크 연동에 따른 데이터 평면 가속화 네트워크 및 VM기반의 컨테이너 네트워크에 대한 연결성 문제점에 대한 고려가 필요 하다.

이를 위해 본 논문에서는 Openstack과 Kubernetes (이하 K8s) 환경을 통합한 테스트베드를 구축하고, 기존 논문에서 고려되지 않은 네트워크 타입별 드라이버 및 데이터 평면 가속화 네트워크를 적용한 다양한 네트워크 구조에서의 성능평가를 진행 하였다.

본 논문의 연구를 위해 2장에서 컨테이너 시스템 및 컨테이너 네트워크 기술에 대해 연구하고, 이를 기반으로 3장에서는 네트워크 서비스 타입별 컨테이너 네트워크 성능 측정을 위한 클라우드 시스템의 컨테이너 연동구조를 설계하고 구현한다. 마지막으로 본 논문의 실험을 위해 베어메탈 및 VM기반의 컨테이너 네트워크를 적용하고 각 실험의 성능평가 및 분석결과를 통해 마무리 한다.

Ⅱ. 관련 연구

본 장에서는 본 논문에서 제안하는 네트워크 서비스 타입별 드라이버 및 네트워크 가속화 기술에 따른 성능검증을 위해 컨테이너 네트워크 기술을 연구한다. 먼저 컨테이너 네트워크 구조를 연구하고, 이와 함께 오픈스택 환경에서의 컨테이너 서비스 제공을 위한 네트워크 연동 기술을 연구한다.

2.1 VM기반 가상화와 컨테이너 가상화 기술

클라우드 시스템의 핵심 가상화 구성인 VM은 하드웨어 가상화를 통해 독립적인 운영체제가 실행 가능한 구조로, 서비스를 위한 응용프로그램 실행시 필요한 OS(Operating System) 커널과, 바이너리 및 관련 라이브러리를 요구하며, 호스트의 하드웨어 자원의 존도가 높다[11]. 이에 반하여 컨테이너는 응용프로그램을 실행하기 위해 필요한 커널 모듈을 호스트와 공유하고 프로그램이 필요한 바이너리 및 라이브러리만 사용하기 때문에 VM보다 하드웨어 의존도가 낮다[12]. Fig. 1은 클라우드 시스템의 가상화 구조로 하이퍼바이저에 의해 GuestOS를 독립적으로 제공하는 VM기반의 가상화 구조와 컨테이너 엔진을 통해 프로그램 독립화를 제공하는 컨테이너 구조이다[13].

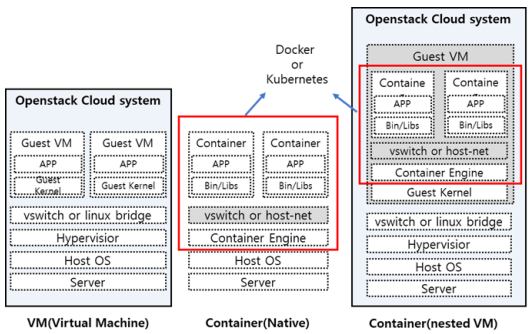


그림 1. VM 및 컨테이너 가상화 구조
Fig. 1. VM and container virtualization architecture

2.2 컨테이너 네트워크 기술

컨테이너를 위한 네트워크 기술은 CNM 및 CNI 구조로 분류된다. CNM은 도커에서 제안한 네트워크 구조로 컨테이너 엔진인 도커 시스템에서 기본적으로 제공되던 네트워크 라이브러리(libnetwork)를 도커 런

타임과 분리시켜 호스트에서 지원하는 다양한 네트워크 드라이버 및 서드파티 드라이버와 연동 가능하게 한 네트워크 모델이다. Fig. 2는 CNM 네트워크 구조이다. (a)의 libnetwork는 CNM의 표준으로 구현된 인터페이스로, 도커 런타임과 네트워크 드라이버간의 연결을 위한 인터페이스를 제공한다. 각 네트워크 드라이버는 컨테이너의 IP 주소를 관리하기 위한 IPAM(IP Address Management) 플러그인과 네트워크 생성 및 삭제를 위한 네트워크 플러그인을 포함하고 있다. 도커 런타임은 libnetwork를 통해 하나 이상의 네트워크 드라이버를 호출하고 컨테이너를 위한 네트워크를 할당 한다. 다중 네트워크 드라이버 호출에 의해 IP가 할당 될 때, 네트워크 드라이버가 컨테이너의 네트워크 네임스페이스에 접근권한이 없기 때문에 IP주소의 충돌이 발생 할 수 있다. 이를 위해 libnetwork에서 IP 충돌방지를 위한 기능을 포함하고 있다.

(b)와 같이 CNM에 의한 네트워크가 생성된 경우 호스트 OS의 Kernel space에서 생성된 네트워크 브릿지와 User space에서 생성된 컨테이너 인터페이스간 연결을 위해 veth pair 인터페이스를 이용해서 연결한다[10,14]. Kernel space에서 관리되는 Host namespace와 User space의 Container namespace(또는 Sandbox라고 지칭)를 연결하기 위한 네트워크 인터페이스를 End-Point 라고 한다[15].

CNI은 CoreOS 프로젝트에서 제안한 네트워크 구조로 rtk(Container Engine with CoreOS-'roket')를 적용한 K8s 등에서 채택되어 사용되고 있다. 기존 CNM에서는 컨테이너 런타임 엔진이 도커 런터임에 종속적이며, libnetwork의 기능 변경 및 추가에 어려움이 있다. 이에 반하여 CNI는 다양한 컨테이너 런타임이 지원되며, 인터페이스 복잡성을 줄이기 위해 컨테이너 네트워크를 위한 추가 및 삭제 명령만을 지원한다. 추가는 컨테이너를 생성 할 때 컨테이너 런타임에 의해 호출되고, 삭제는 컨테이너 인스턴스가 제거될 때 컨테이너 런타임에 의해 호출된다. 그리고 컨테이너 생성시 직접적으로 인터페이스 및 IP를 할당하는 CNM과 달리 CNI는 POD(하나 이상의 컨테이너를 포함하고 있는 그룹) 단위 컨테이너에 인터페이스를 할당하고, 이를 통해 POD 내 컨테이너와 통신이 가능하도록 네트워크를 관리한다. Fig. 3은 CNI 네트워크 구조 이다. 컨테이너 런타임은 먼저 컨테이너에 네트워크 네임스페이스를 할당하고 플러그인 드라이버에 의해 네트워크 인터페이스를 컨테이너의 네임스페이스에 연결하여 IP를 할당한다. 이때 다중 플러그인 드라이버사용시 CNM과 같이 IP 충돌에 대한 문제가 발생 할 수 있다. 이를 위해 CNI에서는 플러그인 드라이버에서 컨테이너 네트워크의 네임스페이스에 대한 접근허용을 통해 충돌을 회피할 수 있다. (b)의 Pod network에서는 기본적으로 하나의 인터페이스를 이용해서 네트워크 데이터를 처리하지만, Intel에서 개발한 Multus Plugin을 사용 할 경우 다중 인터페이스를 사용해서 데이터를 처리 할 수 있다[16].

Fig. 4는 오픈스택 네트워크와 컨테이너 네트워크를 연동하기 위한 Kuryr 컨테이너 네트워크 구조이다. Kuryr는 오픈스택 네트워킹 모듈인 Neutron을 통해 오픈스택에서 관리하는 네트워크를 컨테이너의 네트워크 인터페이스에 할당 하고 관리하기 위한 프로젝트이다[17]. Fig. 4와 같이 Kuryr는 컨테이너의 네트워크 인터페이스 생성을 위해 오픈스택 Neutron에 정의된 컨테이너용 네트워크 대역에서 가상 인터페이스에 대한 생성을 요청 한다. 이를 기반으로 veth pair 인터페이스를 생성하고 Neutron에서 관리되는 호스트측 네트워크 브릿지와 컨테이너의 네트워크 네임스페이스(또는 POD)에 할당함으로써, 데이터 연결을 가능하게 하는 기술이다.

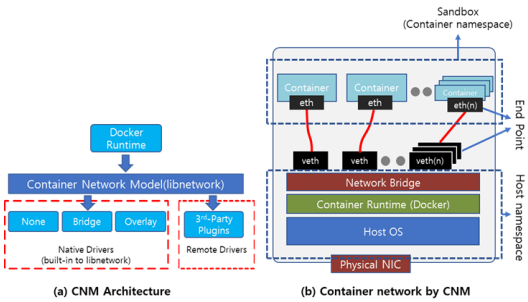


그림 2. CNM에 의한 컨테이너 네트워크 구조
Fig. 2. Container network structure by CNM

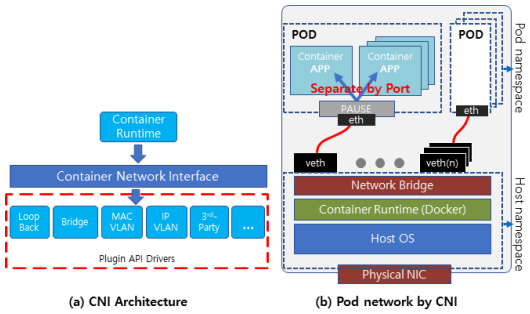


그림 3. CNI에 의한 POD 네트워크 구조
Fig. 3. POD network structure by CNI

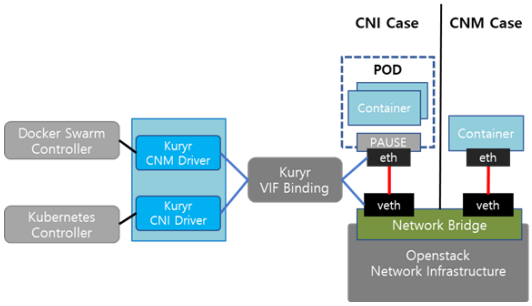


그림 4. Kuryr-CNI에 의한 오픈스택 네트워크와 컨테이너 네트워크 연동 구조
Fig. 4. Openstack and Container network interworking by Kuryr-CNI

2.3 컨테이너 네트워크 가속화 기술

컨테이너 네트워크 가속화 기술에는 기존 VM기반 데이터 평면 가속화기술[11]과 같이 SR-IOV(Single Root I/O Virtualization), DPDK(Data Plane Development Kit), VPP(Vector Packet Processing) 기술이 있으며, Intel에 의해 CNI 플러그인 드라이버가 개발되고 있다[18,19].

Fig. 5의 (a)는 컨테이너 네트워크에 SR-IOV CNI 적용한 구조이다. VM기반 SR-IOV에서는 VM내 네트워크 드라이버와 호스트 네임스페이스의 VF(Virtual Function)와 직접 통신하는 방법을 사용하고 있다. 이와 달리 컨테이너 SR-IOV는 SR-IOV NIC에서 생성한 VF를 POD의 네임스페이스로 VF의 리소스를 할당 한다. 그리고 POD에 할당된 VF 인터페이스는 SR-IOV NIC의 PF(Physical Function)와 연결되어 통신하게 된다. DPDK의 경우 Fig. 5의 (b)와 같이 User space에서 생성된 네트워크 리소스를 POD의 네임스페이스로 공유시킨 후 인터페이스를 생성하게 된다. 이를 위해 POD의 컨테이너는 DPDK 라이브러리가 설치되어 있어야 하며, DPDK APP (Application)에 의해서 네트워크 통신을 하게 된다.

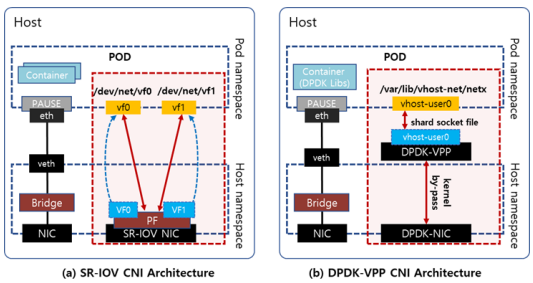


그림 5. 컨테이너 네트워크 가속화 구조
Fig. 5. Container Network Acceleration Architecture

Ⅲ. 성능측정을 위한 시스템 구조 설계

본 장에서는 테스트 베드를 위한 물리호스트 서버 및 VM기반 서버에서 본 논문에서 제안하는 다양한 네트워크 서비스 타입별 데이터 평면 기술을 컨테이너 네트워크에 적용하고 이에 대한 성능평가를 진행하기 위해 다음과 같은 구조를 설계한다. 테스트 베드의 VIM(Virtual Infrastructure Manager)은 오픈스택으로 구성하고, 컨테이너를 위한 COE는 K8s로 구성하였다. 테스트 베드는 3대의 물리서버로 구성하였으며, 한 대의 물리서버에는 VM 및 컨테이너 관리를 위한 컨트롤러 노드로 구성하고 나머지 2대는 VM과 컨테이너가 동일 호스트에서 구동 될 수 있는 컴퓨트 노드로 구성하였다. Fig. 6은 물리호스트에서 구성한 오픈스택과 컨테이너 네트워크 연동 구조이며, 이를 기반으로 다양한 네트워크 구성 및 서비스 타입에 따른 데이터 평면 기술들을 적용하였다.

테스트 베드의 기본적인 컨테이너 네트워크 구조는 Kuryr-CNI를 적용하였다. Kuryr의 POD 네트워크를 위해 오픈스택에서 VLAN 및 VxLAN(Virtual Extensible LAN), 그리고 GRE(Generic Routing Encapsulation) 네트워크 서비스 타입을 가진 3개의 네트워크 대역을 생성하였다. 그리고 POD 의 다중 인터페이스 적용을 위해 Kuryr의 Multi-VIF를 활성화하고, 3개의 네트워크 대역에 대한 정보를 K8s의 네트워크 CRD(Custom Resource Definition)에 등록 하여 POD에서 오픈스택 Neutron 네트워크에 접근 가능하게 하였다. Kuryr-CNI가 적용된 POD에서의 데이터 전송은 POD의 네임스페이스에 할당된 인터페이스와 한 쌍인 veth 인터페이스로 데이터가 전송된다. veth는 OVS(OpenVSwitch)의 통합브릿지(br-int)에 네트워크 서비스 타입별 태그번호로 연결되어 있다. 통합 브릿지에 전달된 데이터는 태그에 따라 네트워크 서비스 타입이 정의되며, VLAN의 경우 외부의 인터페이스로 데이터를 전송하기 전에 VLANID를 추가하여 데이터를 전송한다. 다른 서버에서 VLANID가 일치되는 패킷을 수신하면, 역으로 데이터를 POD로 전송한다. VxLAN 및 GRE의 경우 POD에서 veth를 통해 통합브릿지에 전송되는 과정은 같으나, OVS의 통합브릿지에서 정의된 VxLAN 또는 GRE용 IP를 통해 데이터를 전송한다. 데이터를 전송하기 전에 호스트의 Overlay(VxLAN, GRE) 모듈에 의해 패킷이 캡슐화 되고 전송 된다. flannel[20] 네트워크에서도 POD에서 전송된 데이터가 veth와 매핑된 docker bridge로 데이터가 전달된다. 그리고 docker bridge에 연결된

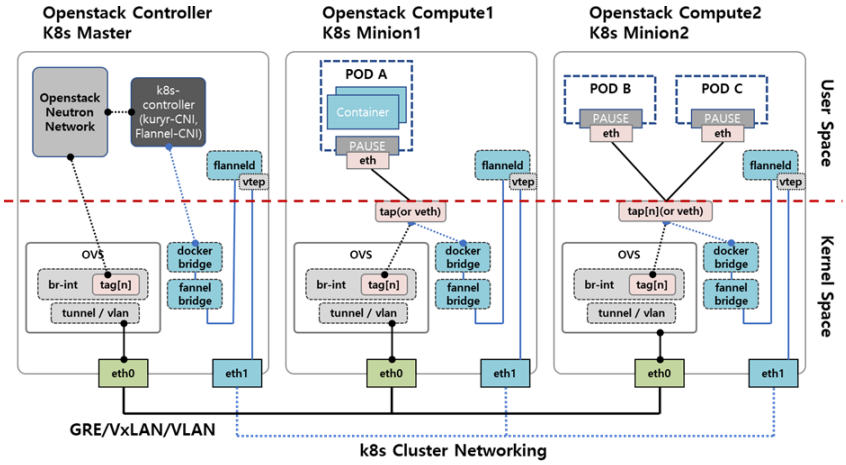


그림 6. 오픈스택 환경에서의 컨테이너 네트워크 연동 구조
Fig. 6. Container network interworking in Openstack environment

flannel bridge를 통해 User Space에 생성된 flannel 데몬에 전달된다. flannel 데몬은 VxLAN 패킷으로 캡슐화 하고 flannel에 정의된 인터페이스를 통해 외부로 전송 한다. MACVLAN 및 IPVLAN[21] 경우 호스트의 네트워크에서 가상화 네트워크에 할당되지 않은 인터페이스를 사용한다. MACVLAN의 경우 호스트의 인터페이스를 부모 인터페이스로 정의 하고, 가상의 MAC 주소를 이용하여 자식 인터페이스를 생성하여 POD에 할당 한다. 데이터 전송은 POD에 할당된 자식 인터페이스에서 호스트의 부모 인터페이스로 직접적으로 전송되며, 부모인터페이스에 의해 외부로 데이터를 보내는 구조이다. IPVLAN도 MACVLAN과 동일하지만 자식 인터페이스를 생성 할 때 부모의 MAC주소를 그대로 이용 하는 부분이 다르다. 그리고 네트워크 가속화 기술인 SR-IOV, OVS-DPDK, VPP를 적용한 환경을 각각 구성하여 실험을 진행 하였다.

Fig. 7은 오픈스택의 각 컴퓨터 노드에서 VM을 생성하고 컨테이너 시스템을 구성한 후 물리서버에 설치된 컨테이너 시스템과 연동하기 위한 네트워크 구성이다. VM내 컨테이너 네트워크는 베어메탈 구조에서의 Kuryr-CNI 및 flannel 네트워크와 동일한 구조로 생성 하였다. 베어메탈 기반의 POD와 VM기반의 POD간 네트워크 연결 및 성능측정을 위해 컴퓨터 노드에서 VM생성시 VM에 할당하는 네트워크는 Overlay 네트워크 및 FLAT, VLAN, 그리고 네트워크 가속화를 적용하여 실험 하였다. 물리서버의 네트워크 구성과 VM내 네트워크 구성을 모두 VLAN으로 구성한 경우, 물리서버에서 생성한 VLAN 패킷을 VM내로 전달하기 위해서는 K8s Minion1의 tap[n](or veth) 포트를 가상의 Trunk Port로 생성 한 후 VM에 할당 하거나, 또는 물리서버의 OVS에 적용된 VLAN Tag-ID값을 VM내의 OVS에 할당된 인터페이스 정보에 VLAN Tag-ID를 추가해야 한다. 베어메탈 및 VM POD간의 데이터 전송은 베어메탈 POD에서 OVS의 통합브릿지에 정의된 veth로 전달되고, veth 태그와 동일한 인터페이스로 데이터를 전송 한다. VM에서 해당 데이터를 수신하면, VM내에 있는 통합 브릿지로 전달되고, 베어메탈 POD에서 생성된 태그와 동일한 태그를 가진 POD의 veth로 데이터가 전달되어 POD에서 데이터를 수신한다. VLAN의 경우 이종간의 시스템 환경을 위해 VLAN 구간에서 Trunk Port 또는 Tag-ID를 맵핑해야 데이터가 전송되지만, VxLAN과 같은 Overlay 네트워크를 사용할 경우 Overlay를 위한 IP 설정만으로 이종간 환경에서 데이터를 송수신 할 수 있다.

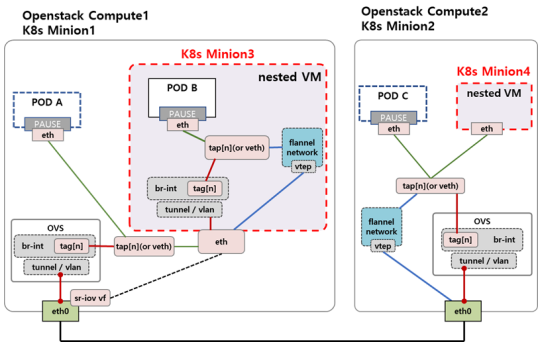


그림 7. VM기반의 컨테이너 네트워크 연동 구조
Fig. 7. VM-based Container network interworking

Ⅳ. 성능 평가

본 장에서는 Fig. 6, 7과 같이 베어메탈기반 컨테이너 시스템과, VM기반 컨테이너 시스템간의 데이터 평면 기술의 성능 비교를 위해 네트워크 처리 속도 향상을 위한 가속화 방법과, 커널에서 지원되는 네트워크 서비스 타입별 구조를 적용하고 성능평가 실시하였다. 성능평가 측정을 위해 2대의 물리호스트에 오픈스택 컴퓨트 노드 환경을 구성하였다. 그리고 각 물리호스트에 컨테이너 환경을 설치하고 오픈스택 네트워크와 연동하였다. VM기반의 컨테이너 시스템 테스트를 위해 각 물리호스트에 VM을 생성하고 물리호스트에서 설치한 컨테이너 시스템 환경과 동일하게 VM에도 동일하게 적용하고 실험을 하였다. 실험을 위한 테스트 시나리오는 BMP2BMP(Baremetal Pod to Baremetal Pod - 베어메탈에서 생성된 컨테이너간 시험), BMP2VMP(Baremetal Pod to VM Pod - 베어메탈에서 생성된 컨테이너와 VM내에서 생성된 컨테이너간 시험), VMP2VMP(VM Pod to VM Pod - VM내에서 생성된 컨테이너간 시험) 방법을 사용하였다. 이때 모든 테스트 시나리오 방법에 동일서버 및 다른 서버에서의 성능측정 방법을 포함한다.

Fig. 8은 베어메탈 및 VM 기반의 컨테이너 시스템에서 POD간의 네트워크 트래픽에 대한 성능측정 구조로, 실험을 위한 하드웨어 및 시스템 환경은 Table. 1과 같다. POD의 컨테이너 내에서 성능측정을 하기 위해 네트워크 성능측정을 위한 Bandwidth tool 및 Socket Latency tool과 컨테이너의 확장성에 따른 성능 측정을 위한 MPI(Message Passing Interface) tool을 설치해서 컨테이너 기본 이미지를 생성 하였다[22-24].

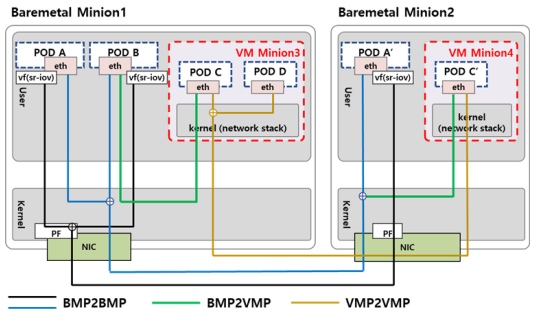


그림 8. POD간의 네트워크 성능측정을 위한 데이터 흐름
Fig. 8. Data flow for measuring network performance between PODs

4.1 베어메탈기반 컨테이너 네트워크 성능 비교

Fig. 9는 Fig. 6의 시스템 설계에 기반을 둔 다양한 네트워크 서비스를 적용한 구조이다. POD 컨테이너간의 성능측정을 위해 호스트 서버의 네트워크 브릿

표 1. 테스트베드 시스템을 위한 규격
Table 1. Specification of Test-bed system

NODE	Classification		Specification
Baremetal (Master Minion1 Minion2)	CPU		Intel(R) Xeon(R) Gold 6148 2.40GHz * 2
	MEMORY		DDR4 2400 MHz 32GB * 6
	SR-IOV NIC		Mellanox ConnectX-5 (40G SFP+)
VM (Minion3 Minion4)	CPU		Virtualized CPU * 8 (apply host-model)
	MEMORY		Virtualized MEM * 32GB
	NIC		vhost-net and sr-iov vf, vhost-user
System Software	OS		Ubuntu 16.04 Server LTS
	Clous OS		Openstack queens by Devstack
	COE		kubernetes v1.9.0 and docker 18.06
	CNI		default cni plugin driver and kuryr, flannel, sr-iov, vshot-user, multus
Testbed	container image		ubuntu 16.04 (modified)
	bandwidth tool		iperf or iperf3 (https://iperf.fr), netperf
	latency tool		sockperf (https://github.com/Mellanox/sockperf)
	CASE	BMP2BMP	Baremetal Pod ⇔ Baremetal Pod (local or remote)
		BMP2VMP	Baremetal Pod ⇔ VM Pod and Baremetal Pod ⇔ Baremetal ⇔ VM Pod
		VMP2VMP	VM Pod ⇔ Baremetal ⇔ VM Pod and VM Pod ⇔ BM⇔ BM ⇔ VM Pod

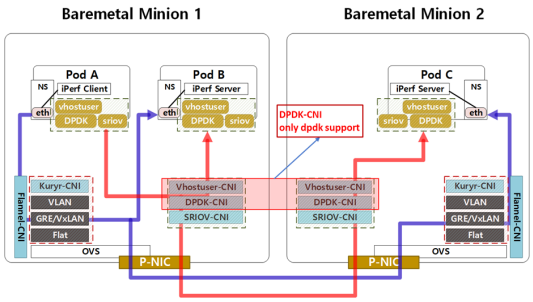


그림 9. 베어메탈 기반 POD간의 네트워크 성능측정을 위한 구조
Fig. 9. Network structure for BMP2BMP measurement

지는 OVS와 flannel1 그리고 가속화를 위한 SR-IOV, vhost-user를 적용 하였다. 네트워크 서비스 타입별 적용을 위해 OVS에서는 VLAN, FLAT, Overlay(VxLAN, GRE) 네트워크를 적용하였으며, 호스트 네이티브 네트워크는 MACVLAN, IPVLAN을 적용하여 실험을 진행하였다.

Fig. 10은 한 대의 베어메탈에서 BMP2BMP 실험을 진행한 결과이다.

Fig. 10은 한 대의 베어메탈에서 BMP2BMP 실험

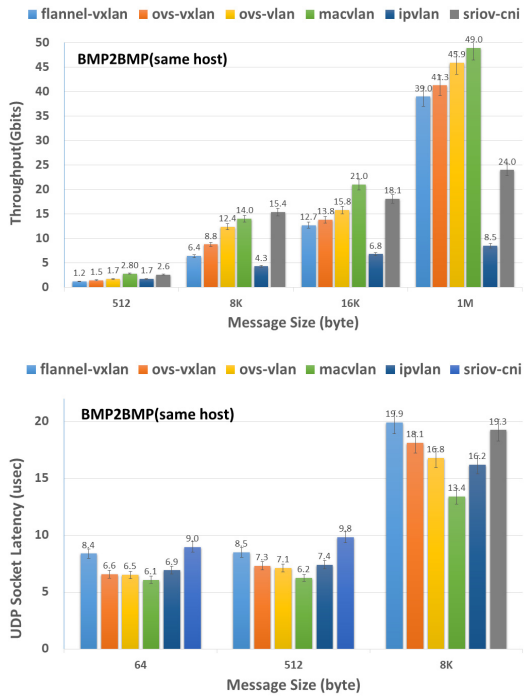


그림 10. 동일 호스트에서 베어메탈기반의 POD간 네트워크 성능 측정
Fig. 10. Measure network performance between BMP2BMP on the same-host

을 진행한 결과이다. Overlay 네트워크를 사용한 경우 K8s의 기본 네트워크인 flannel보다 오픈스택의 기본 네트워크인 ovs-vxlan or gre를 사용한 경우 약 10% 높은 처리량을 확인하였는데, 이는 Fig. 6의 flannel 네트워크의 경우 컨테이너에서 생성된 패킷이 docker bridge와 flannel bridge를 통해 전달되는 구조로 성능 저하가 발생한다. VLAN 실험결과 MACVLAN을 적용한 경우 가장 높은 처리량을 보였다. MACVLAN의 경우 호스트의 네트워크 인터페이스와 컨테이너의 네트워크 인터페이스간 1:N으로 직접적인 구성을 이루고 있으며, 호스트 OS의 관점에서는 네임스페이스로 구분된 하나의 물리 인터페이스로 인식되고 호스트 OS의 네트워크 스택을 이용해서 패킷 라우팅이 되기 때문에 호스트의 성능에 따라 처리량이 높아 질 수 있다. ovs-vlan의 경우 컨테이너의 네트워크 인터페이스와 OVS의 네트워크 브릿지에 생성되는 veth pair인터페이스의 오버헤드로 인해 MACVLAN 보다 약 20% 성능이 낮게 측정 되었다. IPVLAN(l2 mode)의 경우 MACVLAN과 비슷한 방법으로 컨테이너 네트워킹을 지원하지만, Ingress 및 Egress 대한 모든 패킷 처리를 호스트 및 컨테이너에서 각각 네트워크 스택을 사용하여 패킷 포워딩을 처리하기 때문에 커널의 네트워크 사용에 대한 오버헤드로 네트워크 처리량이 가장 낮게 측정되었다. 네트워크 가속화를 위한 SR-IOV를 적용한 경우 16KB 이하의 메시지 처리에서 높은 처리량을 보여 주지만, 그 이상의 메시지를 사용한 경우 IPVLAN을 제외한 다른 네트워크 서비스 보다 낮은 결과를 확인하였다. 이는 데이터가 컨테이너에 할당된 VF에서 호스트의 PF에서 관리되는 SR-IOV 네트워크 카드의 Legacy Switch로 전달되어 처리되고, 네트워크 카드에 정의된 Link Speed에 종속되기 때문에 동일 호스트 커널에서 처리되는 패킷 스위칭보다 성능이 낮게 측정된 부분으로 분석된다. 한 대의 물리 호스트에서 가상화 네트워크를 지원하는 경우 물리호스트의 CPU에 의한 호스트의 커널에서 처리되는 네트워크 스택의 성능 측정이기 때문에 네트워크 하드웨어에서 처리 되는 SR-IOV 보다 더 높은 값이 측정될 수 있다. vhost-user 기술인 DPDK 및 VPP의 경우 컨테이너에서 DPDK APP에 의해 처리되는 부분으로, 현재 정상적인 성능측정 방법을 적용할 수 없으며, 앞의 측정방법과 상이하기 때문에 컨테이너 내에서의 네트워크 처리량 측정에서 제외 하였다.

Fig. 11은 두 대의 베어메탈 서버에서 컨테이너 간 실험을 진행한 결과이다. 16KB 이하의 메시지 사이즈 실험에서 한 대의 서버에서 실험한 결과와 마찬가

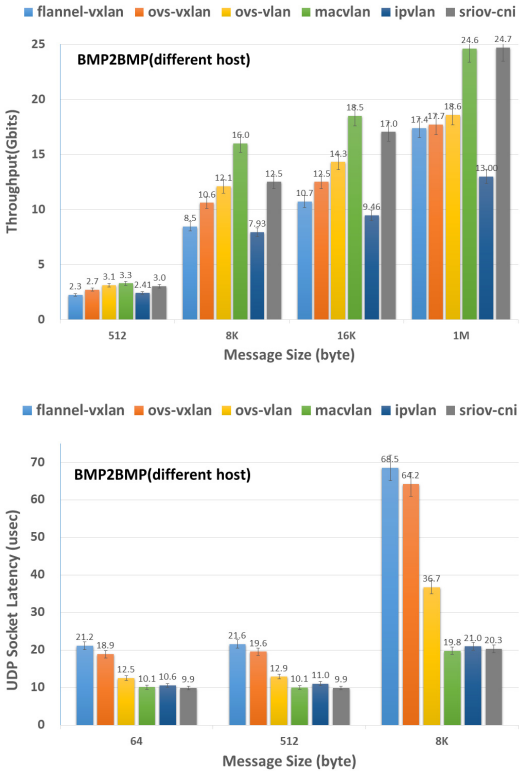


그림 11. 다른 호스트에서 베어메탈기반의 POD간 네트워크 성능 측정
Fig. 11. Measure network performance between BMP2BMP on the different-host

지로 flannel-vxlan 보다 ovs-vxlan 및 ovs-vlan 성능이 더 높게 측정되었으며, MACVLAN 경우 데이터 평면 가속화 네트워크인 SR-IOV 보다 높게 측정 되었다. 이는 Kernel space에서 생성된 SR-IOV의 VF가 User space의 컨테이너 네트워크로 할당되는데, 컨테이너에 적용된 VF의 RDMA(Remote Direct Memory Access) 미지원으로 인해 VF에서 SR-IOV 인터페이스로 직접적으로 패킷 데이터를 Read/Write를 하기 위한 Zero-copy 및 Kernel bypass 기술을 사용할 수 없기 때문에 호스트 커널의 네트워크 스택 사용으로 인한 성능저하가 발생한 것으로 분석되었다. UDP Socket Latency의 MACVLAN 및 SR-IOV 측정결과를 보면, 8K이상의 메시지 사용 시 Overlay 및 veth를 사용하는 ovs-vlan 보다 확연한 성능차이를 보여준다. 특히 IPVLAN의 TCP 성능측정 경우에는 컨테이너에서의 라우팅에 의한 TCP 혼잡제어로 인해 성능 저하가 발생한 반면, UDP의 경우에서는 MACVLAN과 비슷한 결과를 보여주었다.

4.2 베어메탈 및 VM기반 컨테이너 네트워크 성능 비교

Fig. 12은 베어메탈 Pod와 VM기반의 POD간 컨테이너 네트워크 구조이다.

물리호스트에서 VM에 적용한 네트워크는 VLAN 및 SR-IOV를 할당 후 실험을 진행 하였으며, flannel 네트워크의 경우 flannel-vxlan 패킷을 OVS의 VLAN 인터페이스를 통해 VM으로 전송하게 하였다. 또한 Kuryr 네트워크의 VxLAN 및 VLAN 패킷도 동일한 방법으로 처리 하였으며, VM에 전송된 패킷은 OVS를 통해 flannel 네트워크 또는 Kuryr 네트워크에 의해 POD의 컨테이너로 데이터를 전송한다. Fig. 13은 Fig. 12 네트워크 설계에 따른 성능측정 결과이다. 측정 방법은 16KB 메시지 전송에 따른 네트워크 전송량을 측정하였다.

베어메탈기반 POD 및 VM기반 POD에서 VxLAN 네트워크를 사용하고, 베어메탈 호스트와 VM 사이의 네트워크를 VLAN 및 SR-IOV 네트워크로 사용하는 경우 현저한 성능저하를 보였는데, 이는 물리호스트

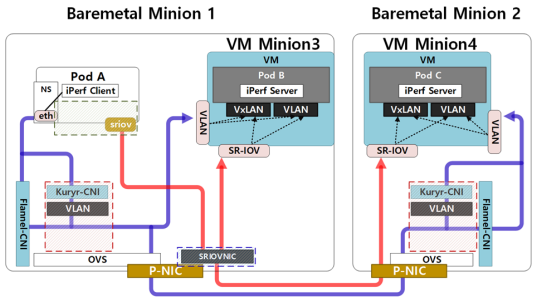


그림 12. 베어메탈기반 POD와 VM기반 POD간 네트워크 성능측정을 위한 구조
Fig. 12. Network structure for BMP2VMP measurement

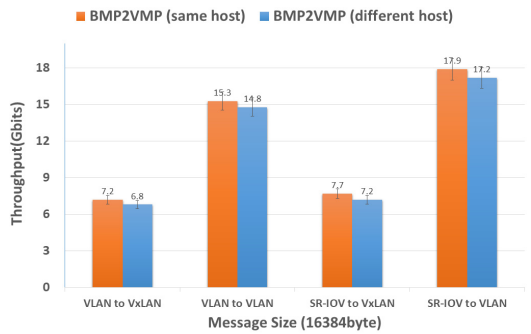


그림 13. 동일 및 다른 호스트에서의 베어메탈기반 POD와 VM기반 POD간 네트워크 성능 측정
Fig. 13. Performance measurement by BMP2VMP on the same and different host

사이를 고속의 네트워크를 사용하더라도 베어메탈 및 VM내에서 VxLAN 패킷의 Enc(Encapsulation) 및 Dec(Decapsulation)에 대한 소프트웨어 처리시 발생하는 오버헤드로 인해 성능저하가 발생 하였다. 그리고 VxLAN 패킷에 사용되는 MTU 사이즈와 물리호스트 사이의 VLAN 또는 FLAT 구조에서의 기본적인 MTU 사이즈의 불일치로 인해 패킷의 단편화에 의한 성능저하가 발생된 것으로 분석된다. VLAN to VLAN과 같이 VM내 컨테이너 네트워크와 VM에 할당된 네트워크를 동일한 서비스 타입의 네트워크로 사용하거나 또는 고속의 네트워크를 사용한 경우 높은 처리량 결과를 확인 할 수 있었다.

4.3 VM기반 컨테이너 네트워크 성능 비교

Fig. 14는 VM기반 POD간의 컨테이너 네트워크 구조이다. VM간 데이터 평면 네트워크를 위한 호스트 네트워크 서비스 타입은 VLAN 및 SR-IOV를 적용하여 실험을 진행 하였다. 베어메탈 POD간 성능평가에서 가장 높게 측정된 MACVLAN의 경우 호스트 네트워크를 MACVLAN 스위치를 사용하고 VM 네트워크 인터페이스로 할당 할 수 없기 때문에 본 실험에서는 제외 하였다. 그리고 VM 네트워크 인터페이스의 SR-IOV 하드웨어 오프로드 기능과 호스트의 VxLAN 네트워크의 하드웨어 오프로드 기능을 추가하여 실험을 진행 하였다.

Fig. 14의 네트워크 구조에 대한 성능측정 결과 BMP2VMP의 동일 호스트 및 다른 호스트의 성능 결과보다 전반적으로 약 20%의 성능 감소를 보였다. VM내 POD 네트워크를 VLAN으로 사용하고, VM에 할당된 호스트 측 네트워크를 SR-IOV Hardware Offload(H/W Off) 기능을 활성화 한 경우, 성능 저하 없이 BMP2BMP의 MACVLAN 성능과 비슷한 결과

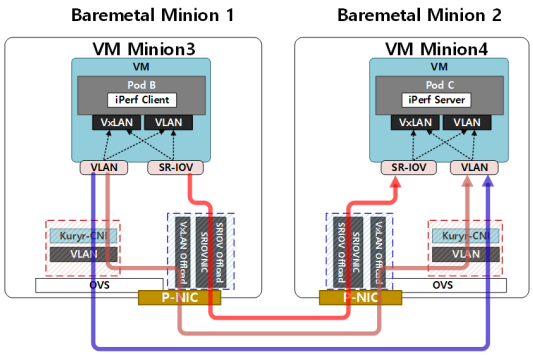


그림 14. VM기반의 POD간 네트워크 구조도
Fig. 14. Network structure for VMP2VMP measurement

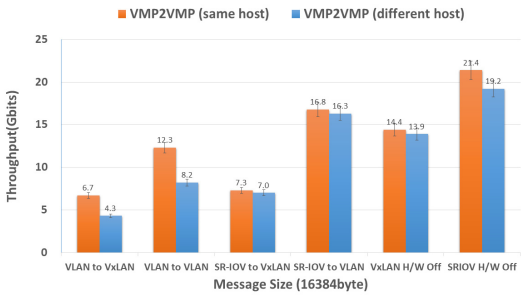


그림 15. 동일 및 다른 호스트에서 VM기반의 POD간 네트워크 성능 측정
Fig. 15. Performance measurement by VMP2VMP on the same and different host

를 얻을 수 있었다. SR-IOV H/W Off는 VM에서 전달된 첫 번째 패킷이 SR-IOV의 스위치로 전달되고, 패킷 데이터의 플로우 결정을 위해 호스트의 네트워크 스택을 이용하여 TC(Traffic Classification) 정보를 SR-IOV 네트워크 카드의 스위치영역에 저장하는 기능을 가지고 있다. 두 번째 패킷 부터는 SR-IOV에 전달된 패킷이, 네트워크 카드에 저장된 TC정보를 기반으로 데이터를 전송하기 때문에 높은 성능을 보여준 원인으로 분석된다.

Table. 2는 다양한 컨테이너 시스템 배치 방법과 이를 위한 네트워크 서비스 타입을 적용하고 성능측정을 바탕으로 전체적으로 요약한 내용이다. Overlay의 경우 모든 컨테이너 시스템 배치에 사용 가능하나 Overlay 패킷 처리시 발생하는 오버헤드에 의해 성능저하가 발생하며, MACVLAN 및 IPVLAN, 네트워크 가속화의 경우 시스템 배치에 따른 제한적인 요소가 존재한다. 베어메탈 기반의 컨테이너 서비스를 구성할 경우 네트워크 서비스 타입을 MACVLAN을 사용하면 높은 네트워크 처리 성능을 얻을 수 있다. 하지만 부모 인터페이스에서 자식 인터페이스 생성시 부모 인터페이스에서 생성 가능한 MAC 주소의 수가 제한되어 있고, MAC 주소 증가에 따른 오버헤드와 베어메탈의 증가시 관리에 대한 복잡성도 고려해야 하는 문제점이 있다. IPVLAN의 경우 L2 모드를 사용할 경우 Ingress/Egress 패킷 포워딩에 대한 오버헤드로 네트워크 처리 성능이 문제가 될 수 있다. SR-IOV와 같은 커널바이패싱 기술을 적용 할 경우 컨테이너에서 SR-IOV에 직접적으로 패킷을 Read/Write 할 수 있도록 적용해야지만 높은 성능을 기대할 수 있으며, 컨테이너를 위한 VF 수가 제한적인 문제점을 가지고 있다. 하지만 VM기반의 컨테이너 서비스를 구성하는 경우 VM의 네트워크 인터페이스를 SR-IOV로 적용

표 2. 컨테이너 시스템 배치 및 네트워크 서비스 타입에 따른 분석요약
Table 2. Summary of performance analysis according to container system placement and network service type

<table>
<tr><th colspan="2">host node / netwok type</th><th colspan="2">BM Pod and BM Pod</th><th colspan="2">BM Pod and VM Pod</th><th colspan="2">VM Pod and VM Pod</th></tr>
<tr><th colspan="2"></th><th>same</th><th>different</th><th>same</th><th>different</th><th>same</th><th>different</th></tr>
<tr><td rowspan="3">Overlay</td><td>flannel</td><td rowspan="3">H/W resource and kernel dependency high</td><td>Performance degradation due to flannel daemon</td><td rowspan="3" colspan="2">Performance degradation due to enc/dec processing even if VMs are same host</td><td rowspan="3" colspan="2">Very low performance due to vxlan packet processing in the VM</td></tr>
<tr><td>os-vxlan</td><td rowspan="2">Performance degradation due to vxlan processing</td></tr>
<tr><td>os-gre</td></tr>
<tr><td rowspan="4">VLAN</td><td>os-vlan</td><td colspan="3">Low performance than macvaln with veth peer network</td><td colspan="2">Depends on VM Host processing performance</td></tr>
<tr><td>MACVLAN</td><td colspan="2">High Performance of L2 between parent and child interface. However, issues when increasing child interface</td><td rowspan="2" colspan="2">Not supported in heterogeneous host network architecture</td><td colspan="2">Depends on VM Host processing performance, but has high throughput</td></tr>
<tr><td>IPVLAN</td><td colspan="2">Performance degradation due to ip routing in POD</td><td colspan="2">Performance degradation due to ip routing in POD in VM</td></tr>
<tr><td rowspan="3">Acceler ation</td><td>SR-IOV</td><td colspan="2">Performance degradation due to non RDMA support between VF and PF</td><td colspan="2">SR-IOV interface in VM is not supported</td><td colspan="2" rowspan="3">High performance when using vlan and macvlan</td></tr>
<tr><td>DPDK</td><td colspan="4" rowspan="2">Need vhost-user for POD network and dpdk app</td></tr>
<tr><td>VPP</td></tr>
<tr><td rowspan="2">H/W Offload</td><td>SR-IOV Offload</td><td colspan="4" rowspan="2">Not supported</td><td colspan="2">Up to 20% better performance than legacy SR-IOV</td></tr>
<tr><td>VXLAN Offload</td><td colspan="2">Similar to legacy SR-IOV</td></tr>
</table>

하거나 또는 H/W Off 기능을 적용하고, VM내 컨테이너 네트워크를 VLAN으로 구성하면, 데이터 평면 네트워크 처리에 대한 높은 성능을 얻을 수 있다.

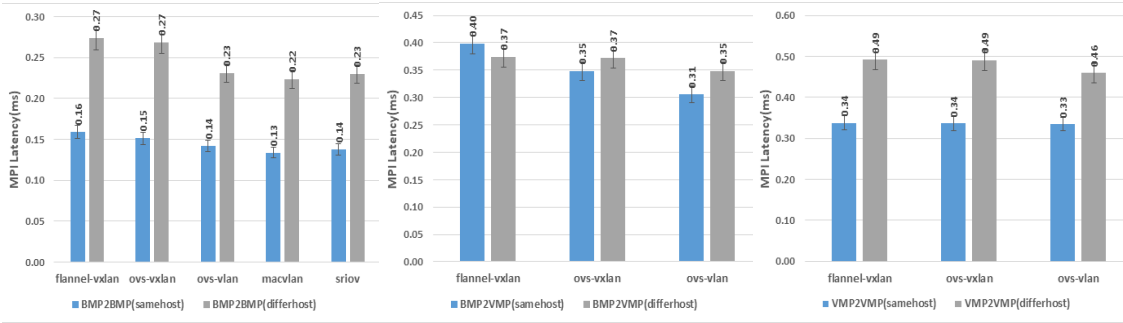
4.4 컨테이너 확장성에 따른 네트워크 성능 비교

Fig. 16는 POD의 증가 및 확장에 따른 성능측정을 위한 구조 이다. 측정방법은 베어메탈 또는 VM에서의 POD 및 데이터 처리를 위한 프로세스 확장에 따른 분산 병렬구조 처리시 네트워크 지연시간을 측정하였다.

실험방법은 MPI(Message Passing Interface)의 집합통신(Collective communication) 중 모든 POD의 테스트 프로세스에서 동일한 크기의 메시지를 전달하는 방법인 MPI_ALLTOALL을 적용하여 성능을 측정하였다[25,26]. MPI소프트웨어는 MVAPICH2.3을 사용하였으며, 컨테이너의 네트워크 통신을 위해 데이터 통신을 위한 네트워크 디바이스 채널은 unix sock을 적용 하였다.

각 Minion 서버에서 하나의 컨테이너를 포함하고 있는 POD를 8개 생성하고, 실험을 위해 각 POD 당 2개의 프로세스를 할당 후 16KB 메시지 사용하여 지연속도를 측정하였다. 베어메탈의 POD에서는 Overlay 네트워크와 VLAN, MACVLAN, 그리고 SR-IOV를 적용하였고, VM기반의 POD는 Overlay 네트워크와 VLAN 만 적용하였다. 이는 베어메탈 기반의 POD에서 SR-IOV 및 MACVLAN을 사용할 경우 VM내에서 생성된 POD와 연결이 안 되기 때문에 실험에서 제외 하였다. Fig. 16의 성능측정 결과를 보면 Overlay 네트워크 보다 VLAN을 사용한 경우 약 10%의 성능차이를 보였다.

Table. 3은 프로세스 수를 4배로 증가 시켜 성능평가를 진행한 결과이다. BMP2BPM 경우 same-host에서는 거의 동일한 성능차를 보였으며, differ-host경우 약 10%의 성능차를 보였다. 그리고 same-host에서 프로세스의 증가에 따라 호스트 리소스 사용량이 증가하기 때문에 differ-host 보다 지연속도가 더 높게 측정되었다. BMP2VPM의 경우에서는 same-host 및 differ-host 실험에서 거의 동일한 성능이 측정되었으며, VMP2VMP의 differ-host 경우 약 20%의 성능차를 보였다. 이는 VM에서 다른 서버에 배치된 VM에 연결되는 네트워크 리소스와 VM 자체의 리소스 소모량이며, 프로세스 증가시 결과 값은 지속적으로 증가될 수 있다.



(a)MPI measurement by BMP2BMP (b)MPI measurement by BMP2VMP (c)MPI measurement by VMP2VMP

그림 16. 동일 및 다른 호스트에서의 MPI-ALLTOALL 집합통신에 의한 성능 측정
Fig. 16. Performance measurement by MPI-ALLTOALL collective communication on the same and different hosts

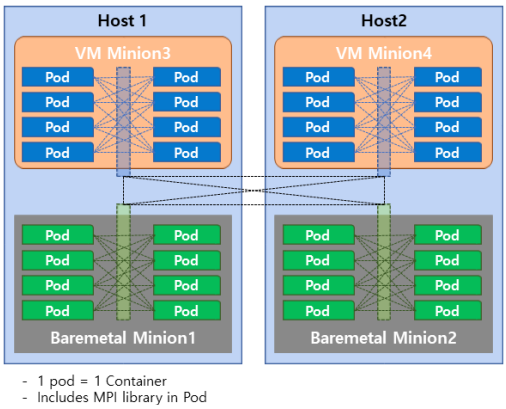


그림. 17. POD 및 POD내 프로세스 증가에 따른 네트워크 성능 측정 구조
Fig. 17. Network performance measurement structure by increasing process in POD and POD

표 3. 각 테스트 시스템에 8개의 POD에서 4개의 프로세스 적용에 따른 성능 측정
Table 3. Performance measurement by allocated four processes on an 8x8 Pod

network / method	flanne vxlan	ovs vxlan	ovs vlan	macvlan	sr-iov
BMP2BMP (same-host)	20.11 (ms)	19.72 (ms)	19.65 (ms)	19.62 (ms)	19.63 (ms)
BMP2BMP (differ-host)	16.11 (ms)	15.84 (ms)	14.81 (ms)	14.52 (ms)	14.46 (ms)
BMP2VMP (same-host)	249.79 (ms)	249.11 (ms)	246.05 (ms)	/	/
BMP2VMP (differ-host)	266.03 (ms)	267.01 (ms)	260.60 (ms)	/	/
VMP2VMP (same-host)	37.48 (ms)	37.18 (ms)	35.35 (ms)	/	/
VMP2VMP (differ-host)	531.39 (ms)	521.39 (ms)	421.83 (ms)	/	/

V. 결 론

본 논문에서는 컨테이너 기반의 네트워크 기술에 대해 연구하고, 이를 기반으로 클라우드 환경 내 CNI 기반 네트워크 적용을 위한 구조를 설계하고 각 네트워크 서비스 타입별 드라이버 및 VM환경을 위한 데이터 평면 가속화 네트워크를 적용하고 실험 하였다.

실험을 위해 베어메탈 기반의 컨테이너 시스템과 VM 기반의 컨테이너 시스템을 구축 하고 실험을 진행 하였다. 한 대의 베어메탈 컨테이너 서비스 구조에서는 물리호스트의 네트워크 스택에서 데이터를 처리하기 때문에 SR-IOV 보다 Overlay 네트워크나 VLAN 환경에서 높은 성능을 보여 주었다. 하지만 하드웨어 리소스 및 커널의 성능에 따라 결과가 다르게 측정될 수 있다. 두 대의 베어메탈 컨테이너 서비스 구조에서는 SR-IOV가 적용된 컨테이너 네트워크의 VF와 물리호스트의 PF간에 RDMA 미지원으로 인해 MACVLAN이 가장 높은 성능을 보여주었다. 베어메탈 기반과 VM 기반 컨테이너 서비스 구조에서는 전체적으로 비슷한 성능을 보였지만 VxLAN의 경우, 패킷처리 오버헤드로 인해 성능이 현저히 낮아지는 것을 확인하였다. 추가적으로, VM내 컨테이너 네트워크 향상을 위해 네트워크 하드웨어 오프로드 기술을 적용한 결과 VM을 위해 VxLAN 네트워크를 사용하더라도 SR-IOV에 근접한 성능을 보여주었으며, SR-IOV를 오프로드한 경우 기존 레거시 모드보다 약 25% 이상의 성능향상 됨을 확인 하였다.

추후 연구에서는 본 논문의 결과를 바탕으로 베어메탈 환경에서의 하드웨어 오프로딩을 적용한 컨테이너 네트워킹 구조 설계 및 구현을 통한 성능검증을 진행하고자 한다.

References

[1] A. Sill, “The Design and Architecture of Microservices,” in *IEEE Cloud Computing*, vol. 3, no. 5, pp. 76-80, Sept. 2016.

[2] D. S. Linthicum, “Practical use of microservices in moving workloads to the cloud,” in *IEEE Cloud Computing*, vol. 3, no. 5, pp. 6-9, Sept. 2016.

[3] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder, “Performance evaluation of microservices architectures using containers,” *2015 IEEE 14th Int. Symp. Netw. Comput. and Appl.*, pp. 27-34, Cambridge, USA, Sept. 2015.

[4] K. Suo, et al., “An analysis and empirical study of container networks,” in *Proc. IEEE INFOCOM*, Honolulu, USA, Apr. 2018.

[5] Docker, *Libnetwork design readme*(2016), Retrieved Jun. 21, 2018, https://github.com/docker/libnetwork/blob/master/docs/design.md.

[6] Containernetworking, *CNI design readme* (2015), Retrieved Jun. 21, 2018, https://github.com/containernetworking/cni.

[7] B. Burns, et al., “Borg, omega, and kubernetes,” *Queue. Containers. ACM*, vol. 14, no. 1, pp. 50-57, Jan. 2016.

[8] B. Hindman, et al., “Mesos: A platform for fine-grained resource sharing in the data center,” in *NSDI*, vol. 11, no. 2011, pp. 22-36, Mar. 2011.

[9] Y. Zhao, et al., “Performance of container networking technologies,” in *Proc. Workshop on Hot Topics in Container Networking and Networked Syst. ACM*, pp. 1-6, Los Angeles, USA, Aug. 2017.

[10] J. Claassen, R. Koning, and P. Grosso, “Linux containers networking: Performance and scalability of kernel modules,” *NOMS 2016 - 2016 IEEE/IFIP Netw. Operations and Management Symp.*, pp. 713-717, Istanbul, Turkey, Jul. 2016.

[11] Y. Park, H. Yang, and Y. Kim, “Application and comparison of data plane acceleration technologies for NFV,” *J. KICS*, vol. 42, no. 8, pp. 1636-1646, Aug. 2017.

[12] P. Sharma, et al., “Containers and virtual machines at scale: A comparative study,” in *Proc. 17th Int. Middleware Conf. ACM*, Trento, Italy, Nov. 2016.

[13] C. Pahl, “Containerization and the PaaS Cloud,” in *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 24-31, May-Jun. 2015.

[14] H.-C. Chang, et al., “Empirical experience and experimental evaluation of Open5GCore over hypervisor and container,” *Wireless Commun. and Mob. Comput.*, vol. 2018, Article ID 6263153, 14 pages, Jan. 2018.

[15] N. Khare, *Docker Cookbook*, Packt Publishing Ltd., 2015.

[16] Multus, *Multi plugin in Kubernetes and provides the multiple network interface support in a pod*(2016), Retrieved Jul. 3, 2018, https://github.com/intel/multus-cni.

[17] Kuryr, *Openstack and container networking integration*(2015), Retrieved Jan. 17, 2018, https://docs.openstack.org/kuryr/latest.

[18] SRIOV-CNI, *SRIOV for Container-networking*(2016), Retrieved Jul. 21, 2018, https://github.com/intel/sriov-cni.

[19] VHOSTUSER-CNI, *Vhost-user for Container-networking*(2017), Retrieved Jul. 21, 2018, https://github.com/intel/vhost-user-net-plugin.

[20] flannel, *Layer 3 network fabric designed for Kubernetes*, Retrieved May 21, 2018, https://coreos.com/flannel/docs/latest.

[21] containernetworking, *CNI Plugin Driver*, Retrieved May 21, 2018, https://github.com/containernetworking/plugins

[22] Netperf, *Network bandwidth testing between two hosts on a network*, Retrieved May 7, 2017, http://www.netperf.org/.

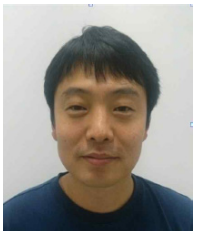
[23] Sockperf, *Network benchmarking utility over socket API-latency and throughput*, Retrieved May 7, 2017, https://github.com/Mellanox/sockperf.

[24] MPI, *MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE benchmarks*, Retrieved Jul. 3, 2018, http://mvapich.cse.ohio-state.edu/benchmarks/.

[25] J. Zhang, X. Lu, and D. K. Panda, "High performance MPI library for container-based HPC cloud on InfiniBand clusters," *IEEE 2016 45th Int. Conf. Parallel Processing (ICPP)*, pp. 268-277, Philadelphia, PA, USA, Sept. 2016.

[26] D. Huang, et al., "Harnessing data movement in virtual clusters for in-situ execution," in *IEEE Trans. Parall. and Distrib. Syst.*, Aug. 2018.

박 영 기 (Young-ki Park)



2013년 8월 : 숭실대학교 정보통신공학 석사

2014년 3월~현재 : 숭실대학교 정보통신공학 박사과정

<관심분야> Cloud, OPNFV, Container, Kubernetes, Data Plane Acceleration

양 현 식 (Hyun-sik Yang)

2013년 2월 : 숭실대학교 정보통신전자공학부 학사

2013년 3월~현재 : 숭실대학교 정보통신소재융합학과 석박통합과정

<관심분야> OPNFV, VNF Monitoring, Blockchain

김 영 한 (Young-han Kim)

1986년 2월 : 한국과학기술원 전기 및 전자 공학과 석사

1990년 2월 : 한국과학기술원 전기 및 전자 공학과 박사

1994년~현재 : 숭실대학교 정보통신전자공학부 교수

<관심분야> OPNFV, SDN/NFV, IoT, CI/CD, Blockchain