기술문서

개방형가상스위치(OVS) 기반의 패킷가속화기술
적용을 통한 SDN/NFV 성능 측정 및 분석

Date : 2017. 02.
Version : 1.0

■ 문서의 연혁

버전	날짜	작성자	비고
초안 - 0.1	2017.01.06	김기현	5장 추가
0.7	2017.01.13	김기현	교정
0.8	2017.01.16	김기현	교정
0.9	2017.01.18	김기현	교정
1.0	2017.01.23	김기현	교정

작성자 : 김기현
Reviewer : 김동균
공동참여자 : 김용환, 김주범

목차

그림 목차

표 목차

1. 서론

최근 빅 데이터에 대한 많은 연구들이 활발히 이루어지고 있다. BDT Insights의 조사에 의하면 2013년 6월 세계 720명의 가트너 멤버를 상대로 조사한 결과, 64%의 기업이 빅 데이터에 투자하고 있거나 투자할 계획이 있으며, 또한 15%의 기업은 2년 이내로 투자할 계획이 있다고 밝혔다[1]. 이와 같이 빅 데이터를 이용하고자 하는 기업들이 늘어남에 따라 빅 데이터 시스템을 구축하기 위해 필요한 사항들 중 중요한 사항이 바로 방대한 데이터의 전송이다[2]. 데이터 전송 속도 향상 및 처리 속도를 향상시키는 직접적인 방안은 물리적인 네트워크 장비 및 케이블의 증설을 통한 하드웨어적 솔루션이 있지만 이는 상당한 CAPEX 및 OPEX의 증가를 초래한다. 이러한 비용 문제를 해결하기 위해 나타난 기술이 네트워크 가상화 기술[3]이다. 하지만 네트워크 가상화 기술은 네트워크의 속도를 보장할 수 없다는 문제점이 있다. 네트워크의 속도를 보장하지 못하는 이유는 서버의 운영체제 커널 단에서 패킷을 처리하기 때문이다. 서버의 운영체제 커널은 네트워크 패킷을 처리하는 과정에서 성능을 저하시키는 요소들이 다수 존재하기 때문에 네트워크 속도를 보장하지 못한다. 성능을 저하시키는 요소로는 버퍼 메모리 할당/해제 문제, 대용량 패킷을 처리할 경우 많은 인터럽트 횟수로 인해 발생하는 성능 저하 문제가 존재한다[4]. 이러한 문제점을 해결하기 위해 나타난 기술이 패킷가속화기술(Data Plane Development Kit, DPDK)[1]이다. DPDK는 인텔 아키텍쳐 기반의 고성능 패킷 처리에 최적화된 프레임을 지원하고, 고가의 장비를 사용하지 않아도 빠른 패킷 처리 기술을 사용할 수 있는 시스템 소프트웨어이다. DPDK는 오픈소스로써, 프로그래밍 프레임워크를 지원하기 때문에 고속의 패킷 처리를 위한 어플리케이션 개발이 가능하다.

DPDK는 크게 데이터 플레인 라이브러리들과 패킷 처리 최적화를 위한 네트워크 인터페이스 카드 드라이버의 집합으로 구성되어 있다. 그림 1은 DPDK의 라이브러리 구조를 나타낸다. 그림 1과 같이 DPDK 라이브러리의 구성요소는 메모리 관리자(Memory Manager), 버퍼 관리자(Buffer Manager), 큐 관리자(Queue Manager), 패킷 플로우 분류(Packet Flow Classification), Poll Mode Driver (PMD)로 구성되어 있다. 메모리 관리자는 메모리 객체들의 풀을 할당하는 기능을 수행하며, Huge Page 메모리[2]에 생성된다. 버퍼 관리자는 운영체제가 버퍼를 할당, 해제하는 시간을 줄이기

1) DPDK (Data Plane Development Kit) : 인텔 아키텍쳐 기반의 고성능 패킷처리 최적화를 위한 시스템 소프트웨어이다.
2) Huge Page 메모리 : 일반적으로 컴퓨팅 시스템에서 메모리 크기를 극복하기 위해 가상메모리 기법을 사용하며, 대용량 메모리를 장착한 시스템의 효율적인 Page Table 관리를 위해 도입된 기술을 의미한다.

위해 메모리 풀에 저장되는 버퍼들을 사전에 할당하여 관리한다. 큐 관리자는 기존에 사용하던 SpinLock Queue[3]를 사용하는 대신 서로 다른 소프트웨어 요소들이 네트워크 패킷을 병렬 처리할 때 불필요한 대기시간을 발생시키기 않도록 Lockless Queue[4]를 구현하여 사용한다. 패킷 플로우 분류는 네트워크 패킷 헤더에 해쉬(Hash) 정보를 효과적으로 생성하여, 네트워크 패킷들이 동일한 플로우에 할당되어 고속 처리를 수행한다. PMD는 기존의 비동기식 인터럽트 기반의 네트워크 패킷 처리 방식 대신에 동기식 Poll Mode 인터럽트 처리 루틴을 사용하여 패킷 파이프라인 속도를 높인다. 따라서 본 보고서에서는 고속의 패킷처리 기술 DPDK를 가상스위치(Virtual Switch, vSwitch)인 개방형가상스위치(Open Virtual Switch, OVS)[5][5]와 연계하여 성능의 보장이 가능한지 실험을 통하여 확인하고자 한다. 실험은 2가지의 시나리오를 바탕으로 실험을 수행하고, 그 결과를 바탕으로 DPDK에 대한 우수성을 검증하고자 한다.

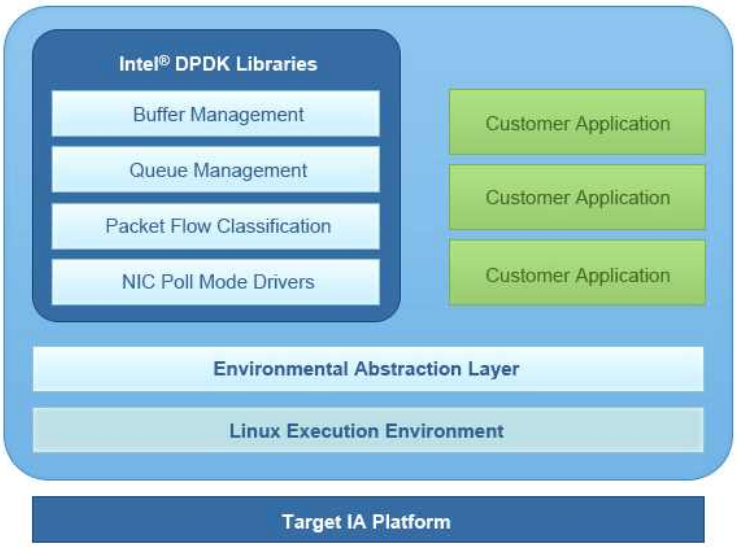

그림 1. DPDK의 구조

3) Spin Lock Queue : 임계 영역에 접근이 불가능한 상황에서 접근이 가능할 때까지 루프를 돌면서 재시도하는 방식으로 구현된 큐를 가리킨다.
4) Lockless Queue : 임계 영역에 대해 바로 접근할 수 있는 경우 접근하고, 바로 접근하기 힘든 경우 에러를 반환하며, 에러를 받은 경우 데이터를 정상적으로 받을 때까지 계속 접근 요청을 다시 보내는 방식으로 구현된 큐를 가리킨다.
5) OVS (Open Virtual Switch) : Apache 2.0 라이센스 기반의 오픈소스 가상 스위치로 멀티레이어 네트워크 스위치의 기능을 하는 소프트웨어이다.

2. 기술 적용 환경 및 성능 측정 시나리오

본 실험은 크게 2가지의 시나리오로 나누어 실험을 수행하였다. 실험 환경을 2가지의 시나리오로 나누어 구성한 이유는 다음과 같다. DPDK가 나타나면서 각광받는 이유는 크게 두 가지로 나누어 이야기 할 수 있다. 첫 번째는 가상화 환경에서도 네트워크 성능을 보장할 수 있는 패킷 가속화 기술과 두 번째는 소프트웨어 정의 네트워크(Software Defined Network, SDN)[6]/네트워크 기능 가상화(Network Function Virtualization, NFV)[7] [6, 7] 기술에서 많이 사용되는 기술로써, 특정 서비스를 위해 필요한 서비스 기능들과 이들 간의 적용 순서를 추상화시킨 기술인 서비스 체이닝(Service Function Chain, SFC)[8] [8] 기술을 구성할 수 있다는 것이다. 따라서 첫 번째 시나리오는 현재 오픈소스로 사용할 수 있는 vSwitch들 중 우수한 vSwitch를 확인해 보기 위함이다. 두 번째 시나리오는 DPDK를 이용하여 SFC 네트워크 환경을 구성하고, 구성된 환경을 바탕으로 네트워크의 성능을 보장할 수 있는 SFC 네트워크 환경을 구성할 수 있는지 확인하기 위함이다.

실험을 위해 총 세 대의 서버를 사용하였으며, 세 대의 서버 사양은 아래의 서버 실험 환경과 같다. 세 대의 서버 중 두 대의 서버는 네트워크 트래픽을 발생시켜 성능을 측정하기 위한 서버와 클라이언트로 사용하고, 나머지 한대의 서버는 vSwitch를 설치하기 위해 사용한다. 첫 번째 시나리오는 오픈소스로 사용할 수 있는 vSwitch인 Linux Bridge, OVS, OVS & DPDK를 서버 한대에 설치하고 세 가지 vSwitch의 성능을 비교하는 실험을 수행했다. 두 번째 시나리오는 DPDK를 이용하여 SFC 네트워크 환경을 구성하고, SFC 기술에서 사용되는 가상서버(Virtual Machine, VM)의 수를 증가시킴에 따라 성능의 변화를 확인하기 위한 실험을 수행하였다.

<u>서버 실험 환경</u>

- Intel Xeon E5-2695 v3 2.3GHz, 35M Cache, 14코어
- Intel X520 10Gb DA/SFP+
- Memory DDR4 128GB
- 400GB SSD SAS Write/12Gbps

---

6) SDN (Software Defined Network) : 네트워크 장비의 제어부분을 데이터 전송 부분과 분리하고, 네트워크 장비의 기능을 정의할 수 있는 API를 외부에 제공하여 이를 통해 프로그램된 소프트웨어로 다양한 네트워크 경로 설정 및 제어 등을 할 수 있도록 하는 기술이다.
7) NFV (Network Function Virtualization) : 네트워크에 필요로 하는 L4-L7 관련 서비스 기능들을 고가의 전용 하드웨어 장비 대신 소프트웨어 기반 고성능의 범용 서버에 가상화시키는 기술이다.
8) SFC (Service Function Chain) : 가상화된 네트워크 기능 및 서비스를 하나의 연결로 순서화한 체인을 따라 네트워크 트래픽을 처리 및 전달하는 기술이다.

<u>OpenSource 버전 환경</u>

- DPDK Version - 16.11
- OpenvSwitch Version - 2.6.09
- QEMU Version - 2.7.91

2-1. 시나리오 1 : 가상스위치 별(vSwitch) 성능 비교 실험

2-1-1. 물리서버 간 성능(throughput) 시험 (PM-vSwitch-PM)

먼저 물리 서버(Physical Machine, PM) 환경에서 세 가지 vSwitch의 성능 측정 시험을 수행하였다. vSwitch로 사용할 서버에 첫 번째로 Linux Bridge, 두 번째로 OVS, 세 번째로 OVS & DPDK 환경을 설치한다. OVS를 설치한 경우 플로우 룰(Flow Rule)을 설정해 준다. OVS에서 사용한 플로우 룰은 port1으로 들어오는 플로우에 대해서는 port2로 나가도록 설정하고, port2로 들어오는 플로우에 대해서는 port1으로 나가도록 설정한다. 플로우 룰에 대한 자세한 명령어는 제3장 개방형가상스위치(OVS)와 패킷가속화기술(DPDK) 시스템 구성 순서에서 자세하게 서술한다. 성능 측정을 수행한 정보를 바탕으로 세 가지 vSwitch에 대한 성능을 비교한다. vSwitch 성능 측정을 위한 PM 환경의 시스템 구성도는 그림 2와 같다.

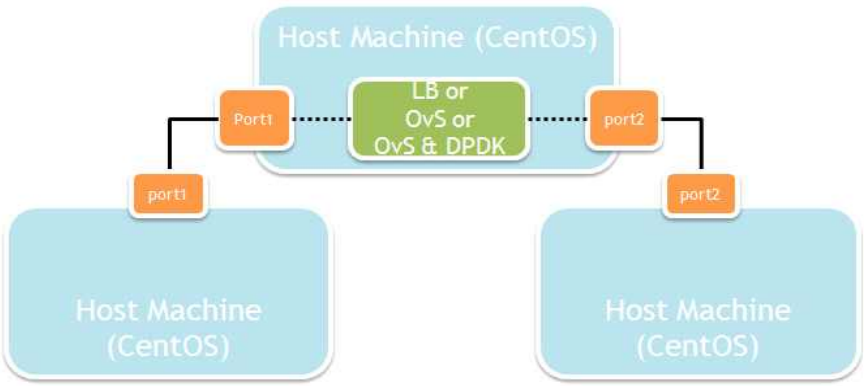

그림 2. vSwitch 성능 측정을 위한 물리적 시스템 구성도

2-1-2. 가상서버 간 성능 시험 (VM-vSwitch-VM)

VM 환경의 시스템 구성도는 그림 3과 같다. VM 환경은 PM 환경과 동일한 실험환경에서 실험을 수행하였다. 단지 다른 실험환경은 그림 3을 보면 서버와 클라이언트로 사용하는 서버의 운영체제로 VMware를 설치하고, 서버에 VM을 하나씩 설치하여 VM 간의 vSwitch 성능 측정 시험을 수행하였다.

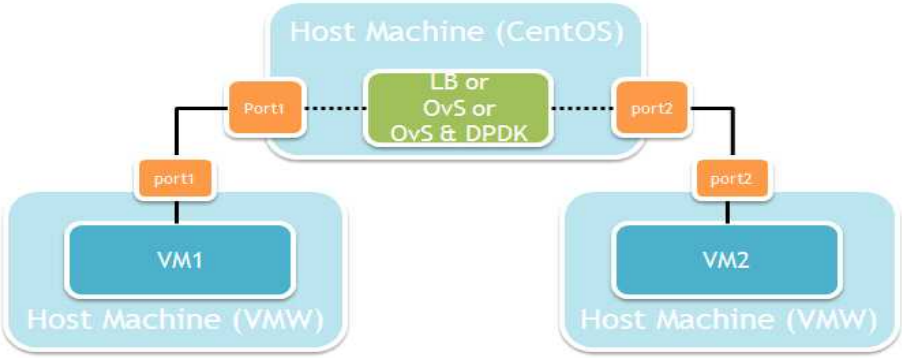

그림 3. vSwitch 성능 측정을 위한 가상 시스템 구성도

2-2. 시나리오 2 : 패킷가속화기술을 이용한 서비스 체이닝 기술 적용 실험

두 번째 시나리오의 실험을 수행하기 위해 두 대의 서버는 트래픽을 발생시킬 클라이언트와 서버로 사용하고, 나머지 한대의 서버는 OVS & DPDK로 사용한다. vSwitch로 사용할 서버에는 OVS, DPDK, Quick Emulator (QEMU)[9]를 설치한다. QEMU는 서버위에 VM을 설치하고 VM과 호스트 머신 간에 DPDK 네트워크를 연결하기 위한 하이퍼바이저로 사용한다. 생성된 각 VM에 할당된 자원은 CPU 4core, Memory 4096MB, HDD 30GB를 할당하며, 실험에서 DPDK를 설치한 VM의 개수를 1개에서 4개까지 늘려가면서 성능 측정을 수행한다. OVS의 플로우 룰은 VM이 생성될 때 마다 모든 VM을 거친 후 빠져나가도록 설정해 주었다. 플로우 룰에 대한 자세한 명령어는 제3장 개방형가상스위치(OVS)와 패킷가속화기술(DPDK) 시스템 구성 순서에서 자세하게 서술한다. OVS & DPDK를 이용하여 SFC 기술을 적용한 시스템 구성도는 그림 4와 같다. 그림 4와 같은 시스템 구성을 통해 VM의 개수에 따른 OVS & DPDK의 성능의 변화를 측정하고자 한다.

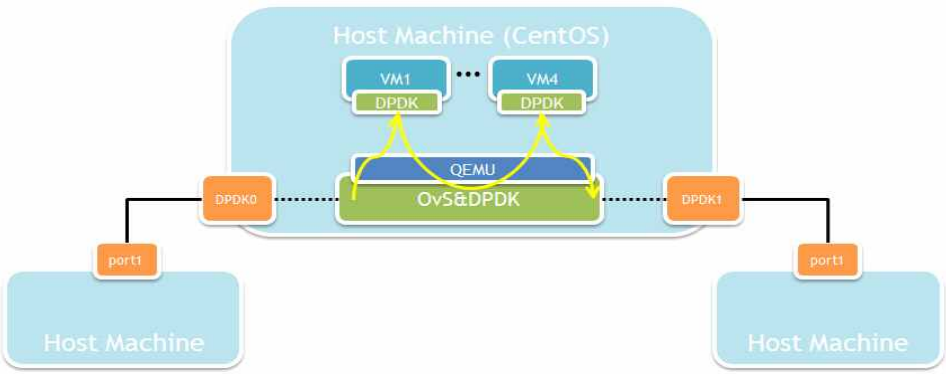

그림 4. Ovs & DPDK 성능 측정을 위한 SFC 시스템 구성도

9) QEMU (Quick Emulator) : x86 이외의 기종을 위해 만들어진 소프트웨어 스택 전체를 가상머신 위에서 실행할 수 있는 가상화 소프트웨어이다.

3. 개방형가상스위치(OVS)와 패킷가속화기술(DPDK) 시스템 구성 순서

3-1. OVS와 DPDK 설치

```
▶ DPDK 설치
# cd /usr/src/
# git clone git://dpdk.org/dpdk
# cd dpdk
# vi /usr/src/dpdk/config/common_base
  -- CONFIG_RTE_BUILD_SHARED_LIB=n
  ++ CONFIG_RTE_BUILD_SHARED_LIB=y
# make install T=x86_64-native-linuxapp-gcc DESTDIR=install

▶ OVS 설치
# cd /usr/src/
# git clone https:://github.com/openvswitch/ovs.git
# cd ovs
# ./boot.sh
# ./configure —with-dpdk=/usr/src/dpdk/x86_64-native-linuxapp-gcc
# make
```

3-2. OVS & DPDK 연계 및 환경설정

```
▶ OVS 연결을 위한 환경설정
# vi /etc/default/grub
  GRUB_CMDLINE_LINUX="rd.lvm.lv=fedora-server/root rd.lvm.lv=fedora-server/swap
  default_hugepagesz=1G hugepagesz=1G hugepages=16 hugepagesz=2M
  hugepages=2048 iommu=pt intel_iommu=on isolcpus=1-13,15-27 rhgb quiet"
# grub2-mkconfig –o /boot/grub2/grub.cfg
# reboot
# mkdir –p /mnt/huge
# mkdir –p /mnt/huge_2mb
```

 # mount –t hugetlbfs nodev /mnt/huge

 # mount –t hugetlbfs nodev /mnt/huge_2mb –o pagesize=2MB

 # mount

▶ mount 결과 화면

```
nodev on /mnt/huge type hugetlbfs (rw,relatime,seclabel)
nodev on /mnt/huge_2mb type hugetlbfs (rw,relatime,seclabel,pagesize=2MB)
```

▶ DPDK 네트워크 인터페이스 바인딩

 # modprobe uio

 # insmod /usr/src/dpdk/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko

 # cd /usr/src/dpdk/tools/

 # ./dpdk-devbind.py --bind=igb_uio 03:00.1

 # ./dpdk-devbind.py --bind=igb_uio 03:00.0

 # ./dpdk-devbind.py —status

▶ DPDK 네트워크 인터페이스 바인딩 상태 결과 화면

```
[root@localhost tools]# ./dpdk-devbind.py --status

Network devices using DPDK-compatible driver
============================================
0000:03:00.1 'Ethernet Controller 10-Gigabit X540-AT2' drv=vfio-pci unused=

Network devices using kernel driver
===================================
0000:03:00.0 'Ethernet Controller 10-Gigabit X540-AT2' if=eno1 drv=ixgbe unused=
vfio-pci *Active*
```

▶ OVS & DPDK 연결 및 가상 네트워크 구성

 # mkdir -p /usr/local/etc/openvswitch

 # mkdir -p /usr/local/var/run/openvswitch

 # rm /usr/local/etc/openvswitch/conf.db

 # ovsdb-tool create /usr/local/etc/openvswitch/conf.db ₩

/usr/local/share/openvswitch/vswitch.ovsschema

 # ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock ₩

--remote=db:Open_vSwitch,Open_vSwitch,manager_options ₩

--pidfile —detach

 # ovs-vsctl —no-wait init

 # export DB_SOCK=/usr/local/var/run/openvswitch/db.sock

```
# ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-init=true
# ovs-vswitchd unix:$DB_SOCK —pidfile --detach
# ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk-socket-mem="1024,0"
# ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=6
# ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
# ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk
# ovs-vsctl add-port br0 dpdk1 -- set Interface dpdk1 type=dpdk
```

▶ OVS & DPDK 가상 네트워크 구성 결과 화면

```
# ovs-vsctl show
 Bridge "br0"
     Port "br0"
         Interface "br0"
             type: internal
     Port "dpdk1"
         Interface "dpdk1"
             type: dpdk
     Port "dpdk0"
         Interface "dpdk0"
             type: dpdk
```

※ 만약 아래와 같은 에러가 나타났을 경우

```
Libraries have been installed in:
   /usr/local/lib
If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the `-LLIBDIR'
flag during linking and do at least one of the following:
   - add LIBDIR to the `LD_LIBRARY_PATH' environment variable
     during execution
   - add LIBDIR to the `LD_RUN_PATH' environment variable
     during linking
   - use the `-Wl,-rpath -Wl,LIBDIR' linker flag
   - have your system administrator add LIBDIR to `/etc/ld.so.conf'
See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
```

※ 해결방법

```
# export LD_LIBRARY_PATH=$DPDK_DIR/x86_64-native-linuxapp-gcc/lib

▶ 시나리오1 플로우 룰 설정 방법
# ovs-ofctl del-flows br0
# ovs-ofctl add-flow br0 in_port=1,action=output:2
# ovs-ofctl add-flow br0 in_port=2,action=output:1

▶ 시나리오2 플로우 룰 설정 방법
– 1VM
# ovs-vsctl add-port br0 dpdkvhostuser0 -- set Interface dpdkvhostuser0 ₩
 type=dpdkvhostuser
# ovs-vsctl add-port br0 dpdkvhostuser1 -- set Interface dpdkvhostuser1 ₩
 type=dpdkvhostuser
# ovs-ofctl del-flows br0
# ovs-ofctl add-flow br0 in_port=1,action=output:3
# ovs-ofctl add-flow br0 in_port=3,action=output:1
# ovs-ofctl add-flow br0 in_port=4,action=output:2
# ovs-ofctl add-flow br0 in_port=2,action=output:4
– 2VM
# ovs-vsctl add-port br0 dpdkvhostuser2 -- set Interface dpdkvhostuser0 ₩
 type=dpdkvhostuser
# ovs-vsctl add-port br0 dpdkvhostuser3 -- set Interface dpdkvhostuser1 ₩
 type=dpdkvhostuser
# ovs-ofctl del-flows br0
# ovs-ofctl add-flow br0 in_port=1,action=output:3
# ovs-ofctl add-flow br0 in_port=3,action=output:1
# ovs-ofctl add-flow br0 in_port=4,action=output:5
# ovs-ofctl add-flow br0 in_port=5,action=output:4
# ovs-ofctl add-flow br0 in_port=2,action=output:6
# ovs-ofctl add-flow br0 in_port=6,action=output:2
– 3VM
# ovs-vsctl add-port br0 dpdkvhostuser4 -- set Interface dpdkvhostuser0 ₩
 type=dpdkvhostuser
# ovs-vsctl add-port br0 dpdkvhostuser5 -- set Interface dpdkvhostuser1 ₩
 type=dpdkvhostuser
```

```
# ovs-ofctl del-flows br0
# ovs-ofctl add-flow br0 in_port=1,action=output:3
# ovs-ofctl add-flow br0 in_port=3,action=output:1
# ovs-ofctl add-flow br0 in_port=4,action=output:5
# ovs-ofctl add-flow br0 in_port=5,action=output:4
# ovs-ofctl add-flow br0 in_port=6,action=output:7
# ovs-ofctl add-flow br0 in_port=7,action=output:6
# ovs-ofctl add-flow br0 in_port=8,action=output:2
# ovs-ofctl add-flow br0 in_port=2,action=output:8
– 4VM
# ovs-vsctl add-port br0 dpdkvhostuser6 -- set Interface dpdkvhostuser0 ₩
 type=dpdkvhostuser
# ovs-vsctl add-port br0 dpdkvhostuser7 -- set Interface dpdkvhostuser1 ₩
 type=dpdkvhostuser
# ovs-ofctl del-flows br0
# ovs-ofctl add-flow br0 in_port=1,action=output:3
# ovs-ofctl add-flow br0 in_port=3,action=output:1
# ovs-ofctl add-flow br0 in_port=4,action=output:5
# ovs-ofctl add-flow br0 in_port=5,action=output:4
# ovs-ofctl add-flow br0 in_port=6,action=output:7
# ovs-ofctl add-flow br0 in_port=7,action=output:6
# ovs-ofctl add-flow br0 in_port=8,action=output:9
# ovs-ofctl add-flow br0 in_port=9,action=output:8
# ovs-ofctl add-flow br0 in_port=2,action=output:10
# ovs-ofctl add-flow br0 in_port=10,action=output:2
```

3-3. QEMU 설치 및 VM과 DPDK 네트워크 연계

```
▶ QEMU 설치
# cd /usr/src
# git clone git://git.qemu.org/qemu.git
# cd qemu
# ./configure --target-list=x86_64-softmmu
# make
```

▶ **VM 설치 및 DPDK 네트워크 연계**

```
# ./qemu-system-x86_64 -name dpdk1 -cpu host -enable-kvm –m 4096M ₩
 –hda create address –cdrom image address --nographic -snapshot –vnc :11 ₩
 -numa node,memdev=mem -mem-prealloc -smp sockets=1,cores=2 –object ₩
 memory-backend-file,id=mem,size=4096M,mem-path=/dev/hugepages,share=on ₩
–chardev socket,id=char0,path=/usr/local/var/run/openvswitch/dpdkvhostuser0 ₩
–netdev type=vhost-user,id=mynet1,chardev=char0,vhostforce ₩
-device virtio-net-pci,mac=00:00:00:00:00:01,netdev=mynet1,mrg_rxbuf=off ₩
-chardev socket,id=char1,path=/usr/local/var/run/openvswitch/dpdkvhostuser1 ₩
-netdev type=vhost-user,id=mynet2,chardev=char1,vhostforce ₩
–device virtio-net-pci,mac=00:00:00:00:00:02,netdev=mynet2,mrg_rxbuf=off

# cd /usr/src/dpdk/
# git clone git://dpdk.org/dpdk
# cd dpdk
# make install T=x86_64-native-linuxapp-gcc
# cd app/test-pmd
# vi testpmd.c
  ++ define RTE_TEST_RX_DESC_DEFAULT 2048
  ++ define RTE_TEST_TX_DESC_DEFAULT 2048
# export RTE_SDK=/root/dpdk
# export RTE_TARGET=x86_64-native-linuxapp-gcc
# make
# vi /etc/default/grub
  GRUB_CMDLINE_LINUX="rd.lvm.lv=fedora-server/root rd.lvm.lv=fedora-server/swap
  default_hugepagesz=1G hugepagesz=1G hugepages=1 hugepagesz=2M
  hugepages=1024 isolcpus=1,2 rhgb quiet"

# grub2-mkconfig –o /boot/grub2/grub.cfg
# reboot
# mount -t hugetlbfs hugetlbfs /mnt/huge
# mount -t hugetlbfs none /mnt/huge_2mb –o pagesize=2MB
# modprobe uio
# insmod /usr/src/dpdk/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko
# /usr/src/dpdk/tools/dpdk-devbind.py --status
```

```
# /usr/src/dpdk/tools/dpdk-devbind.py -b igb_uio 00:03.0
# /usr/src/dpdk/tools/dpdk-devbind.py -b igb_uio 00:04.0
# ./testpmd -c 0x3 -n 4 --socket-mem 1024 -- ₩
   --burst=512 -i --txqflags=0xf00 --disable-hw-vlan
     > set nbcore 1
     > show config fwd
     > set fwd mac retry
     > start
```

▶ **DPDK 어플리케이션 testpmd 실행 결과**

```
testpmd> start
io packet forwarding - ports=2 - cores=1 - streams=2 - NUMA support disabled, MP over anonymous pages disabled
Logical Core 1 (socket 0) forwards packets on 2 streams:
  RX P=0/Q=0 (socket 0) -> TX P=1/Q=0 (socket 0) peer=02:00:00:00:00:01
  RX P=1/Q=0 (socket 0) -> TX P=0/Q=0 (socket 0) peer=02:00:00:00:00:00

  io packet forwarding - CRC stripping disabled - packets/burst=512
  nb forwarding cores=1 - nb forwarding ports=2
  RX queues=1 - RX desc=2048 - RX free threshold=0
  RX threshold registers: pthresh=0 hthresh=0 wthresh=0
  TX queues=1 - TX desc=2048 - TX free threshold=0
  TX threshold registers: pthresh=0 hthresh=0 wthresh=0
  TX RS bit threshold=0 - TXQ flags=0xf00
testpmd> clear port stats all

  NIC statistics for port 0 cleared

  NIC statistics for port 1 cleared
```

4. 실험 및 분석결과

4-1. 실험 데이터 수집 방법

실험 데이터를 수집하기 위해 두 대의 서버에 네트워크 성능(throughput) 측정 프로그램 iperf3를 이용하여 성능 데이터를 수집한다. iperf3를 이용한 성능(throughput) 측정 간격은 1초로 총 시간은 60초간 10번의 명령어를 수행하여 데이터를 텍스트 형식으로 추출한 후 저장한다. 이와 같은 방법으로 스트림의 수를 1 부터 6 까지 변화를 주어 데이터를 구성하였다. 구성된 데이터를 바탕으로 분석을 수행한다.

4-2. 시나리오 1의 실험 및 분석결과

첫 번째 시나리오의 실험은 PM 환경과 VM 환경에서 Linux Bridge, OVS, OVS & DPDK의 성능 비교를 통한 vSwitch 비교 실험을 수행하였다. 첫 번째 시나리오의 PM 환경에서 싱글 스트림으로 데이터를 보냈을 때, 구성된 데이터의 실험 횟수에 따른 성능의 변화를 보기 위한 그래프는 그림 5이다. 그림 5는 1초 간격으로 60초간 추출한 성능의 평균을 계산하여 각 횟수에 따른 성능을 나타낸 그래프이다. 그림 5를 보면 Linux Bridge가 갑자기 5.5Gbits/sec로 떨어지는 현상이 발생한다. 이러한 현상은 네트워크의 불안정을 나타내며, Linux Bridge는 다른 vSwitch에 비해 네트워크가 불안하다는 것을 알 수 있다. 이러한 현상은 운영체제의 커널 단에서 패킷을 처리해 주기 때문에 생기는 현상들이다. 반면에, OVS와 OVS & DPDK의 경우 안정된 그래프를 보여준다.

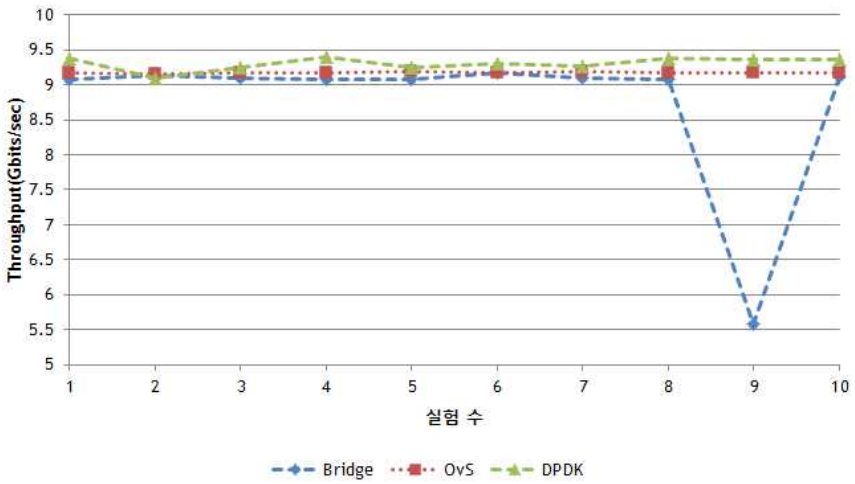

그림 5. PM-vSwitch-PM 환경에서 실험 횟수에 따른 성능 비교 그래프

그림 6은 첫 번째 시나리오의 VM 환경에서 싱글 스트림으로 데이터를 보냈을 때, 구성된 데이터의 실험 횟수에 따른 성능의 변화를 보기 위한 그래프이다. 그래프를 보면 VM 환경에서는 Linux Bridge, OVS의 경우 PM 환경에서 보다 성능의 변동이 크게 나타나는 것을 볼 수 있다. 그에 반면 OVS & DPDK의 경우 성능의 변동이 작게 나타난다. 이와 같은 현상으로 보았을 때 Linux Bridge나 OVS에 비해 OVS & DPDK가 안정성 면에서 더 우수하다고 볼 수 있다. (이하 성능 -> 성능으로 대체)

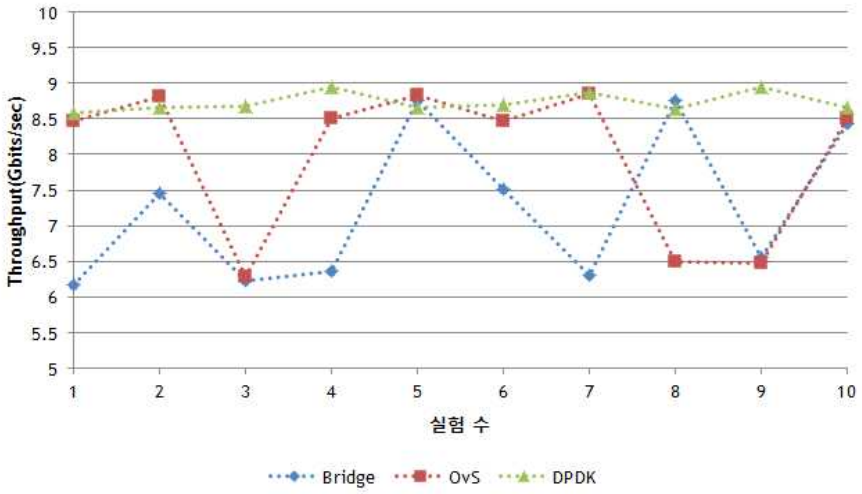

그림 6. VM-vSwitch-VM 환경에서 실험 횟수에 따른 성능 비교 그래프

그림 7은 첫 번째 시나리오의 PM 환경에서 싱글스트림부터 6스트림까지 각 스트림에 대한 성능의 변화를 보기 위한 그래프이다. 99% 신뢰구간 공식을 사용하여 신뢰구간 값을 구한 후 최대, 최소, 평균을 이용하여 그래프를 그린다. 사용된 99% 신뢰구간 공식은 식 (1)과 같다.

$$\left[X-2.53\frac{\sigma}{\sqrt{n}},\ X+2.53\frac{\sigma}{\sqrt{n}}\right]\quad n:\text{표본크기},\ X:\text{표본평균},\ \frac{\sigma}{\sqrt{n}}:\text{표준편차}\quad\cdots\cdots\ (1)$$

PM 환경의 신뢰구간 그래프에서 Linux Bridge, OVS, OVS & DPDK를 비교해 보면, 싱글 스트림과 2스트림에서 Linux Bridge의 경우 성능이 낮고, 성능의 편차도 크게 나타난다. 신뢰구간 그래프를 통해 PM 환경에서 OVS와 OVS & DPDK는 안정성 면에서 Linux Bridge보다 우수하다는 것을 알 수 있다. OVS, OVS & DPDK는 Linux Bridge보다 모든 스트림 수에서 9Gbits/sec이상의 높은 성능을 보장한다는 것을 알 수 있다.

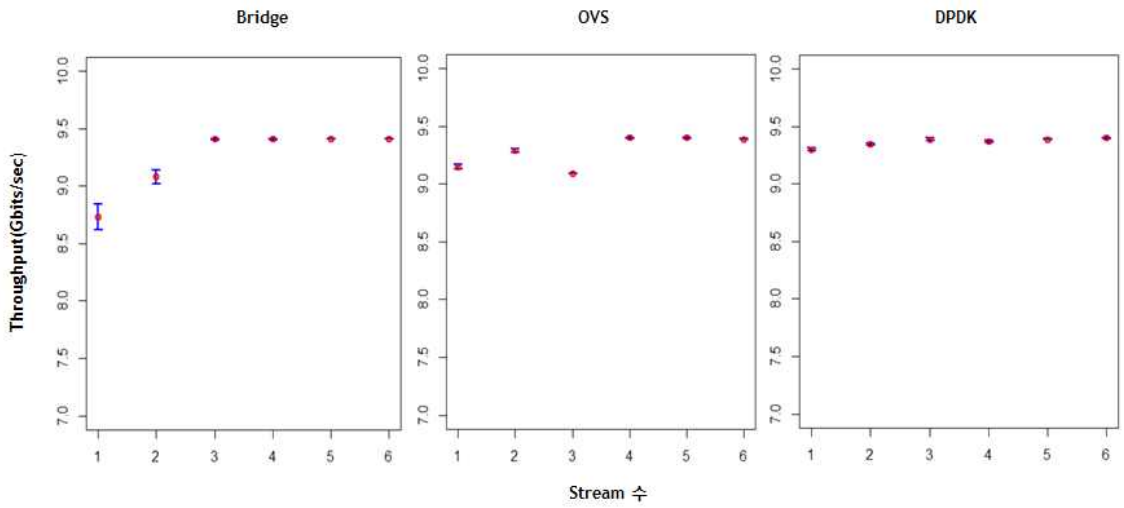

그림 7. PM-vSwitch-PM 환경에서 신뢰도구간 그래프

그림 8은 첫 번째 시나리오의 VM 환경에서 싱글 스트림부터 6스트림까지 각 스트림에 대한 성능의 변화를 보기 위한 신뢰구간 그래프이다. VM 환경에서 싱글 스트림부터 6스트림까지의 성능의 변화를 보면 많은 성능의 편차를 보인 vSwitch는 OVS이다. OVS의 경우 싱글 스트림부터 3스트림까지 성능의 편차가 크고 성능 또한 낮다. Linux Bridge의 경우 싱글 스트림에서 성능의 편차도 크지만 성능 또한 낮게 나타난다. 그에 반면 DPDK의 경우 안정성 면에서 우수하며, 성능 또한 높게 나타난다.

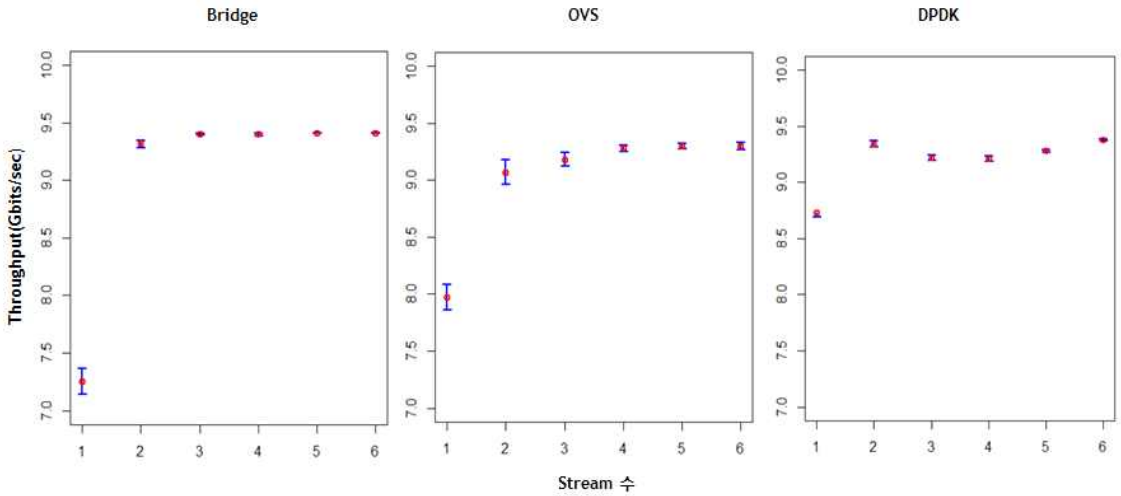

그림 8. VM-vSwitch-VM 환경에서 신뢰도구간 그래프

4-3. 시나리오 2의 실험 및 분석결과

두 번째 시나리오의 실험은 SFC 기술을 구현하였을 때 VM의 수가 늘어남에 따라 성능의 변화에 대한 실험을 수행하였다. 그림 9는 VM의 수가 늘어남에 따라 싱글 스트림

으로 데이터를 보냈을 때, 구성된 데이터의 실험 횟수에 따른 성능의 변화를 보기 위한 그래프이다. 그래프를 보면 1VM과 2VM에서는 성능의 편차가 심하지 않고, 9Gbits/sec에 근접한 성능을 제공한다. 하지만 3VM와 4VM에서 성능이 낮아지기 시작한다. 이에 따라 본 실험 환경에서는 SFC를 수행하기 위한 최적의 VM 수는 2개이다. 2개 이상일 경우 높은 성능을 보장해 주지 못한다는 것을 알 수 있다.

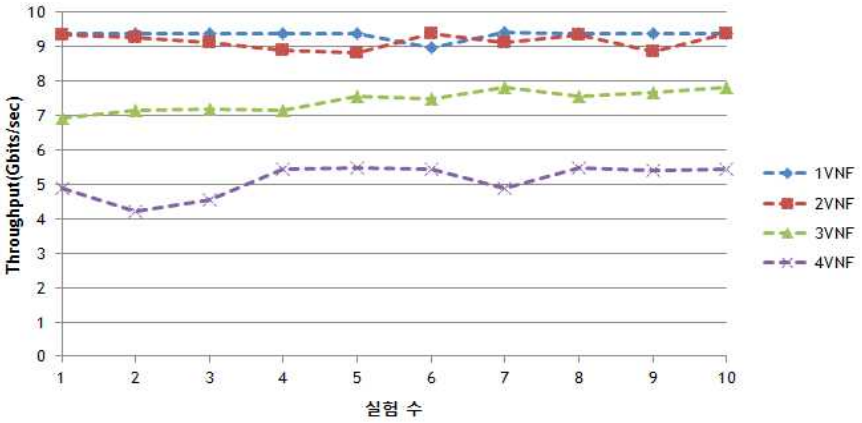

그림 9. SFC 환경에서 VM의 개수에 따른 성능의 변화 그래프

두 번째 시나리오에서 VM의 수가 늘어남에 따라 싱글 스트림에서 6 스트림까지 데이터를 보냈을 때, 구성된 데이터의 스트림 수에 따른 성능의 변화를 보기 위한 신뢰구간 그래프는 그림 10이다. 그래프를 보면 1VM과 2VM에서는 성능의 편차가 심하지 않고, 9Gbits/sec에 근접한 성능을 제공한다. 하지만 3VM와 4VM에서 성능이 낮아지기 시작하는 것이 확연히 눈에 보인다. 이에 따라, 본 실험 환경에서는 SFC를 수행하기 위한 최적의 VM 수는 2개라고 판단된다.

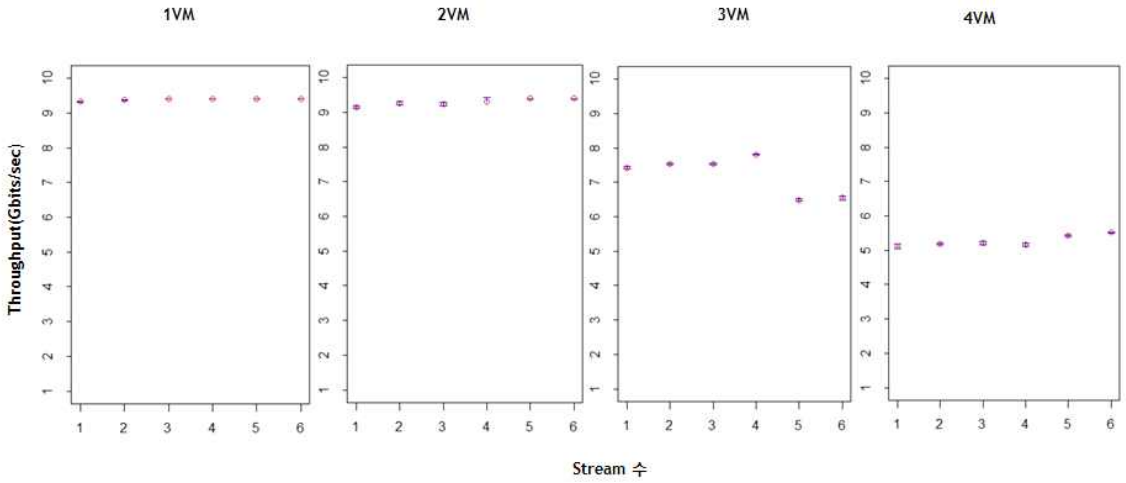

그림 10. SFC 환경에서 VM의 개수에 따른 성능의 변화 신뢰도구간 그래프

5. 결론

본 보고서에서는 운영체제의 네트워크 문제점을 보완한 인텔 아키텍쳐 기반의 고성능 패킷 처리 기술 DPDK에 대해 알아보았다. 또한 세 대의 서버를 이용하여 시나리오 1과 시나리오 2에 대한 실험을 수행하였다. 시나리오 1에서는 PM 환경과 VM 환경에서 Linux Bridge, OVS, OVS & DPDK 세 가지의 vSwitch를 설치하여 네트워크 성능 비교 실험을 수행하였다. 표 1은 시나리오 1의 VM 환경에서 싱글 스트림으로 iperf 명령어를 사용했을 때, 명령어 실행 횟수에 따른 성능의 변화를 나타낸 결과이다. 표 1을 보면 Linux Bridge의 경우 6Gbits/sec 부터 9Gbits/sec 까지 성능의 변화가 약 3Gbits/sec 정도의 편차를 보인다. 또한 OVS의 경우도 Linux Bridge와 마찬가지로 6Gbits/sec 부터 9Gbits/sec 까지 성능의 변화를 보이며 약 3Gbits/sec 정도의 편차를 보인다. 이는 네트워크 성능의 보장이 어렵다고 볼 수 있다. OVS & DPDK의 경우 성능의 편차가 거의 없으며, 8.6Gbits/sec 정도의 성능을 유지하는 것을 볼 수 있다. 이에 따라 OVS & DPDK를 사용하면 PM 환경뿐만 아니라 VM 환경에서도 성능의 우수성과 안전성을 검증하였다.

표 1. 시나리오 1의 VM 환경에서 vSwitch 비교

	1	2	3	4	5	6	7	8	9	10
Bridge	6.15	7.44	6.21	6.35	8.76	7.50	6.28	8.76	6.54	8.43
OVS	8.48	8.8	6.27	8.51	8.82	8.46	8.86	6.47	6.47	8.50
OVS&DPDK	8.61	8.67	8.67	8.94	8.66	8.69	8.81	8.64	8.94	8.66

시나리오 2에서는 DPDK를 이용하여 SFC 환경을 구성한 후, VM의 수를 1개에서 4개까지 늘려가며 성능의 변화에 대한 실험을 수행하였다. 표 2는 시나리오 2의 실험 환경에서 싱글 스트림으로 iperf를 사용하였을 때 1VM부터 4VM까지의 성능의 변화를 나타낸 표이다. 표 2를 보면 1VM부터 4VM까지 성능의 편차는 크지 않지만, VM의 개수가 3개 이상으로 늘어남에 따라 성능이 8Gbits/sec 보다 낮아지는 현상을 볼 수 있다. 따라서 본 실험 환경에서 이상적인 VM의 수는 2개로 판단된다. 하지만 서버의 성능을 향상시키거나 네트워크 성능을 높였을 경우 3개의 VM까지 성능을 보장할 수 있다고 사료된다.

표 2. 시나리오 2의 DPDK를 이용한 SFC 성능 시험

	1	2	3	4	5	6	7	8	9	10
1VM	9.38	9.39	9.38	9.37	9.37	8.98	9.39	9.39	9.37	9.38
2VM	9.35	9.28	9.12	8.89	8.84	9.37	9.13	9.35	8.87	9.36
3VM	6.91	7.13	7.19	7.16	7.57	7.47	7.80	7.55	7.67	7.81
4VM	4.90	4.23	4.55	5.44	5.47	5.44	4.87	5.47	5.41	5.42

6. 향후 연구 및 실험 계획

본 실험을 수행하면서 OVS & DPDK를 이용하여 네트워크 성능의 보장을 검증하였다. 하지만 네트워크의 속도가 변하거나 거리가 멀어짐에 따라 성능을 보장할 수 있을지 의문이다. 또한 OVS & DPDK를 구성하기 위해서는 제 3장에서 설명한 시스템 구성 순서에 따라 명령어를 입력해야 한다. 하지만 이와 같이 명령어를 입력하기 위해서는 많은 시간적 비용이 투자된다. 이를 바탕으로 본 보고서에서는 아래와 같은 향후 연구 및 실험을 계획할 예정이다.

6-1. 네트워크 성능 및 광역망(WAN) 적용에 따른 추가 실험

서버의 네트워크의 속도를 10G에서 40GB로 향상시키고, 서울 데이터센터와 대전 데이터센터에 OVS & DPDK를 설치한 서버를 SDN 광역망(SD-WAN)인 소프트웨어 융합형 첨단연구망(KREONET-S)[10]과 연결하여 네트워크의 연동 거리가 증가함에 따라 성능의 결과를 도출하고자 한다. 그림 11은 OVS & DPDK를 KREONET-S[11]망과 연동하여 실험을 수행할 구조도를 나타낸다. 대전과 서울에 KREONET-S망과 OVS & DPDK 서버를 연결하고 성능 측정 실험을 수행한다. 제2장에서 설명한 실험 시나리오 및 환경과 동일한 실험방법을 이용하여 실험을 수행할 예정이다.

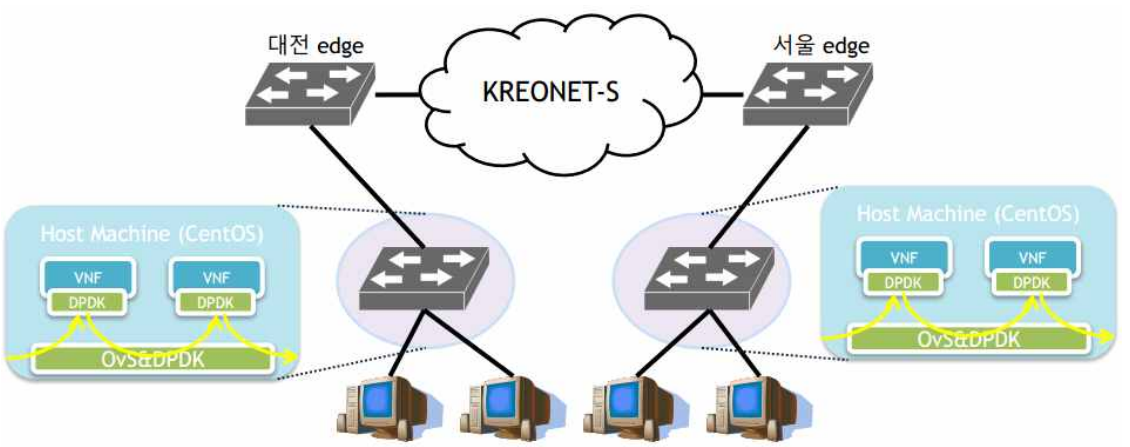

그림 11. KREONET-S 망과 DPDK를 연동한 실험 구조도

6-2. OpenStack과 ONOS의 연계를 통한 클라우드 네트워크

기존의 KREONET-S 망에 서버 세 대를 스위치를 이용하여 연결시키고, 각 서버에 OpenStack[10], OVS & DPDK, ONOS (Open Network Operating System)[11]를 설치한다. 그 후 오픈소스 환경의 연계를 통하여 네트워크 설정을 마우스 클릭으로 손쉽

게 변경하고 관리할 수 있는 클라우드 네트워크 툴을 구성할 예정이다.

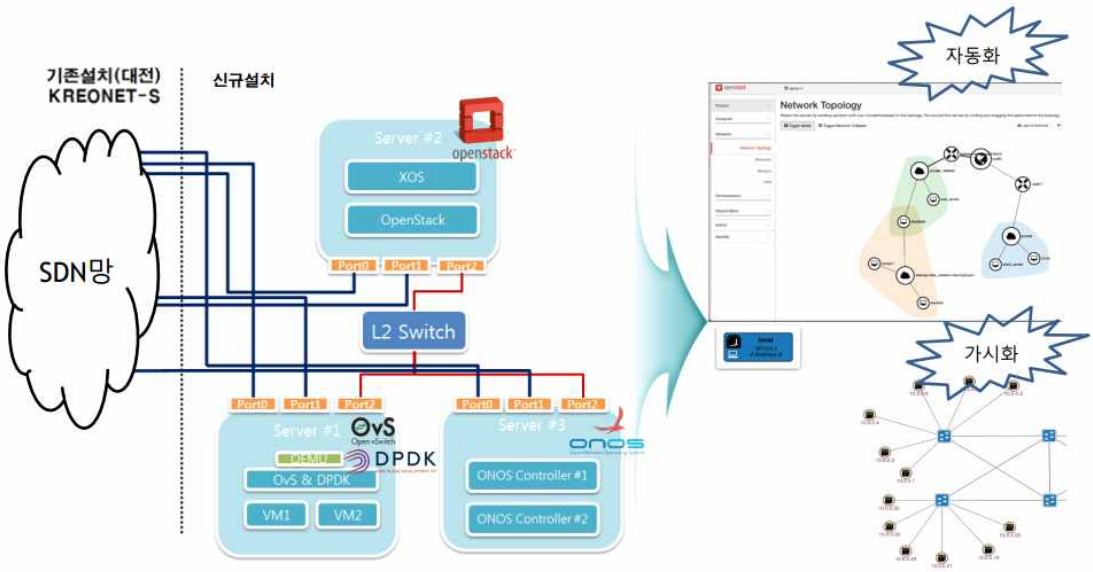

그림 12. OpenStack, ONOS, OVS & DPDK 연계 구조도

10) OpenStack : 프로세싱, 저장공간, 네트워킹의 가용자원을 제어하는 목적의 IaaS 형태의 클라우드 컴퓨팅 오픈소스 프로젝트이다.
11) ONOS (Open Network Operating System) : 미국의 비영리 연구소인 ON.Lab과 선도적인 글로벌 통신사들이 주도하고 있는 최초의 분산형 오픈소스 SDN 제어 플랫폼이다.

Reference

[1] BigDataTranser insights, “세계 64%의 기업, ‘빅데이터에 투자하겠다.’”, http://www.bdtinsights.com
[2] BigDataTranser insights, “연구소 빅데이터, ‘연구 자료의 전송’”, http://www.bdtinsights.com
[3] CUI, Laizhong, YU, F. Richard, YAN, Qiao, When big data meets software-defined networking: SDN for big data and big data for SDN. IEEE Network, 2016, 30.1: 58-65.
[4] 김석구, 정규식, 자료 구조 재사용을 이용한 리눅스 TCP 네트워킹 성능 개선. 한국통신학회 종합 학술 발표회 논문집 (추계) 2013, 2013, 251-252.
[5] SDxCentral, “What is Open vSwitch (OVS)?”, https://www.sdxcentral.com
[6] Open Networking Foundation , “Software-Defined Networking (SDN) Definition, https://www.opennetworking.org
[7] SDxCentral, “What is NFV - Network Functions Virtualization - Definition?”, https://www.sdxcentral.com/
[8] 한국정보통신기술협회, “SFC(Service Function Chaining) 기술 소개 및 표준화 동향”, http://www.tta.or.kr/
[9] INTEL, “Intel Open Network Platform Release 2.1 Performance Test Report”,2016.03.
[10] 김용환; 김동균. SD-WAN 기반의 사용자 중심 가상 전용 네트워크 시스템 설계 및 구현. 한국통신학회논문지, 2016, 41.9: 1081-1094.
[11] KREONET-S, http://www.kreonet-s.net