1. What is DPDK?

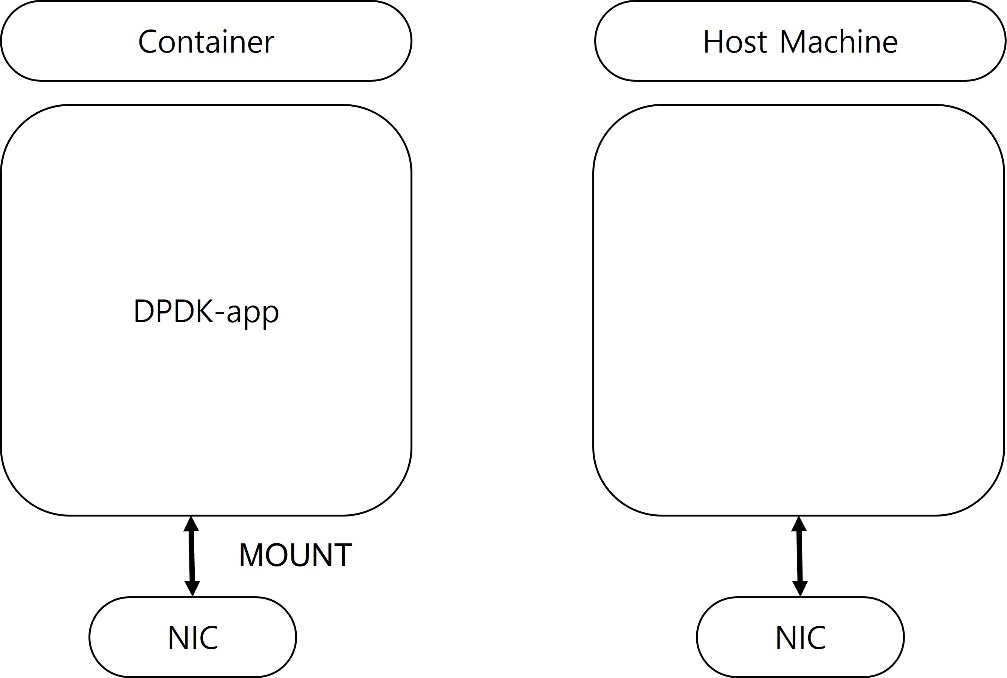
DPDK is an acronym for Data Plane Development Kit. It is developed by Intel. When we process the network, we usually divide the process into two parts: control plane and data plane. Control plane is deciding where should packets go or what path should packets take using the header part of the packet. Data plane is where we actually move the data of packets. Why would this needed?

2. What does DPDK do?

Refer to <A Survey of Trends in Fast Packet Processing> Konstantina Tsiamoura Betreuer: Florian Wohlfart, Daniel G. Raumer Seminar Future Internet SS2014. Way to process high speed network abound. DPDK is one of the solutions to high-speed packet processing. You may ask “our normal networks don’t need this. Our OS does all the work. However, the interrupt based packet processing that now OS uses is very slow solution. As more packets arrive, the more interrupts it may make. Also, if it is some kind of attack scenario, interrupt will block other operations that OS should process either. We can simulate this kind of scenario using packet generator(pktgen) which can send various size of packets until it reaches the maximum packet rate through the interface. If we send 64byte packets in a high rate (through 10Gb/s NIC), the receiver wouldn’t be able to receive all packets through normal OS (like Ubuntu, Windows). However, if we apply dpdk application on receiver side, we can receive all packets. In other words, we can receive packets in 10Gb/s rate with 64byte packet. This is due to the mechanism of DPDK. DPDK uses polling, huge page, batch processing, and memory pool techniques.

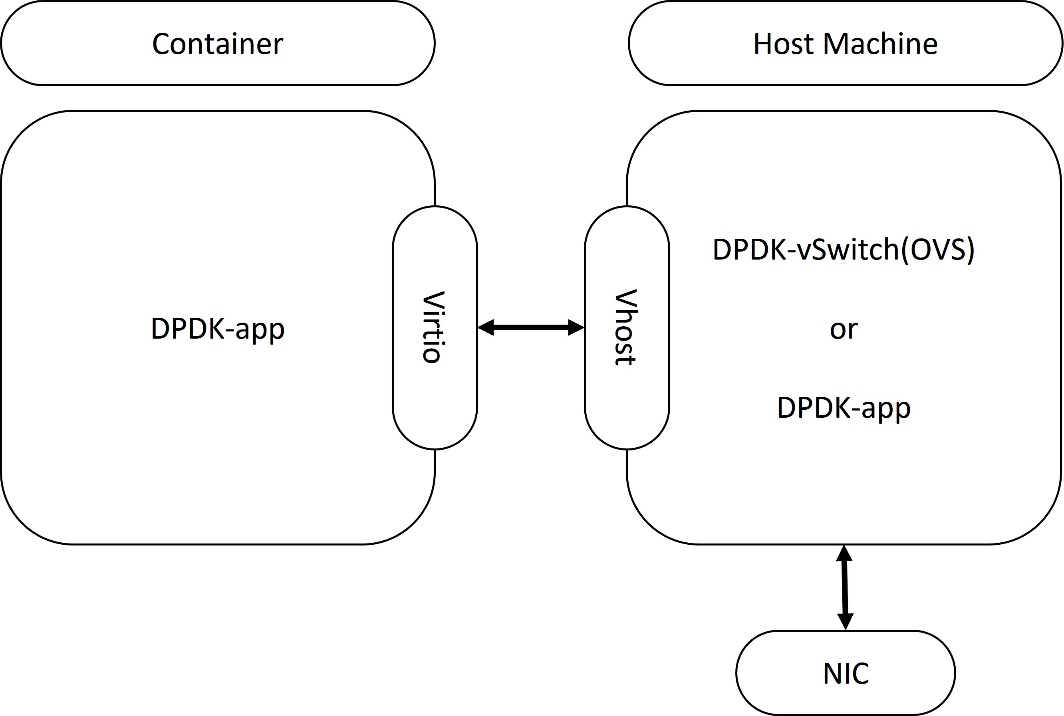
|  |  |
| --- | --- |
| Polling | It fully utilizes cpu(just like while loop waiting packets to come), processing packets at the moment it is received. It is not interrupted by OS to schedule other process. |
| Huge page | In linux or normal types of OS, page size is usually 4KB. In DPDK, we use 2MB or 1GB huge pages inside memory. In this case, we can think of receiving 10KB of data. We would need 3-page translation for receiving in 4KB page size while 1 page translation for 2MB or 1GB huge pages. |
| Batch processing | The speed of CPU is much faster than memory speed. DPDK solves this issue by processing batch of packets at a time. |
| Memory pool | Allocating memory and freeing takes too much time. DPDK allocates memory pool at initialization process. |

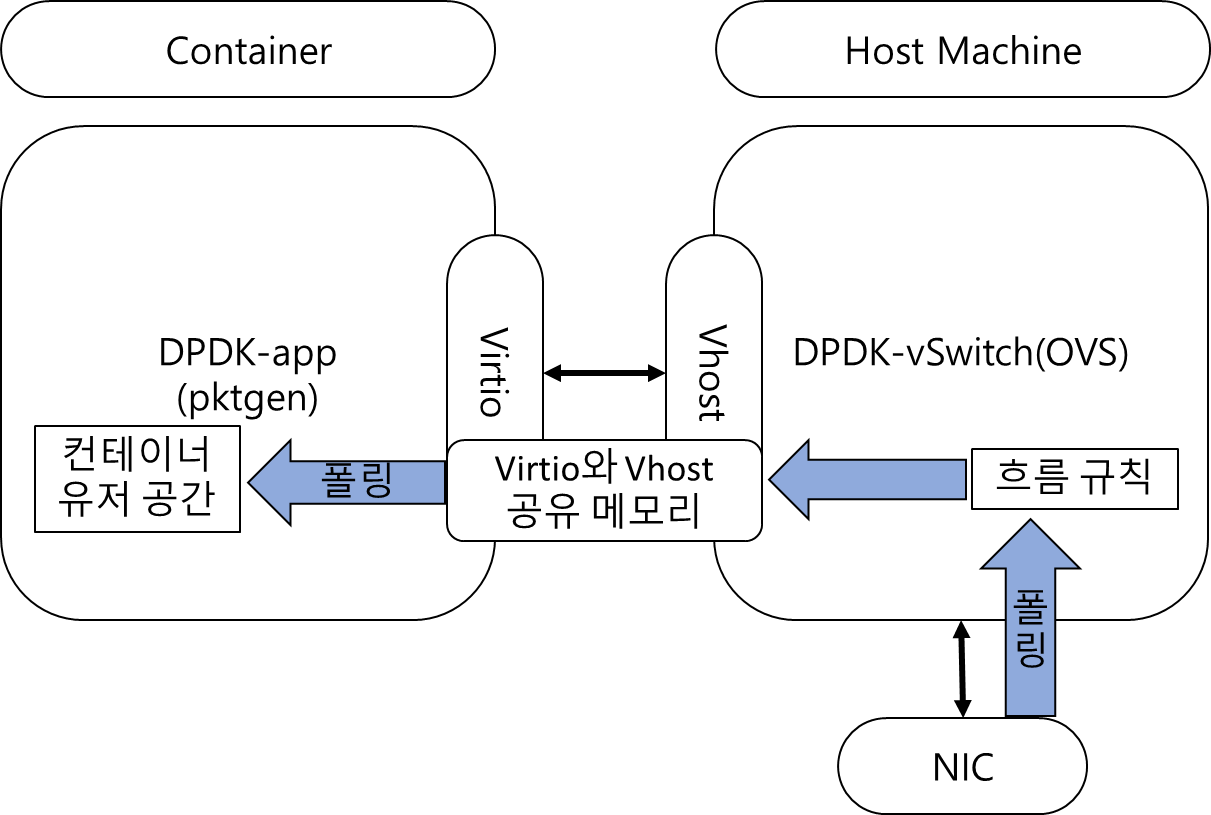
3. What are we going to do?

 We will try to enhance the speed of the packet transmission to container. In ordinary condition, there will be host OS and container on it. Host will receive the packets first and the container can receive the packet. To apply DPDK here, we have two solutions. First, give container our NIC so that container can directly receive it. In this scenario, it is just same as using DPDK at host side. It will be able to receive the packets in a full rate.

Second, use DPDK both in host and container side. You first receive packets in host and then pass it to container. Container can receive the packets using DPDK. In this type of scenario, we need at least two cpu to isolate. It can be quite redundant but cpu is more flexible than NIC in the first solution. Some alleges that we have SR-IOV solution in first case. It is virtualizing the NIC into multiple software NIC which is called as VF (virtual function). However, its number is still bound to NIC specification.

4. How to pass packets into container using second solution?

We found out two ways to put packets into container. First solution is using OVS-DPDK in host, and DPDK inside container. OVS is an acronym for OpenvSwitch which can utilize DPDK. It receives packets using DPDK and follows flow rule that can be set manually by programmers. Flow rule is simple like for src\_ip:10.1.2.3/24 -> out:”vhost-user0”, the packets from 10.1.2.3/24 goes to vhost-user0 port if there is port named “vhost-user0”. Then, we connect this port with container. If container executes DPDK application, they can poll packets. Actually, second solution came up due to slow speed of first solution. After some experiments done in above solution, we found out that performance is really low just like this paper. <Performance of OVS-DPDK in container networks> Quang-Huy Nguyen from ICTC 2020. There, we tried abstraction and we didn’t need OVS for passing packets into container. Second solution is executing DPDK application in host and container. We use vhost-user with virtio to connect host and container.

5. Environment settings

To set environment like above, we first install DPDK and OVS.

<https://docs.openvswitch.org/en/latest/intro/install/dpdk/>

We follow above build. We used dpdk-stable-20.11.2 in our experiment.

$ cd /usr/src/

$ wget https://fast.dpdk.org/rel/dpdk-20.11.1.tar.xz

$ tar xf dpdk-20.11.1.tar.xz

$ export DPDK\_DIR=/usr/src/dpdk-stable-20.11.1

$ cd $DPDK\_DIR

$ export DPDK\_BUILD=$DPDK\_DIR/build

$ meson build

$ ninja -C build

$ sudo ninja -C build install

$ sudo ldconfig

You have to make sure OVS build requirements and dpdk build requirements are done beforehand.

After installing OVS build requirements, use git clone https://github.com/openvswitch/ovs.git

or download tarball from openvswitch.org/download/

Then ./boot.sh inside the root folder of ovs. Then, ./configure --with-dpdk=static

Probably, $DPDK\_DIR will be used for static. If not, they will find dpdk from /usr/src/dpdk- directory.

Since it is the default directory. So, try to give path to dpdk-20.11.2/build.

Then,

$ make

$ make install

to install ovs-dpdk completely.

If you want to test packets, make sure to modprobe vfio-pci.

You can use any of drivers mentioned <https://doc.dpdk.org/guides/linux_gsg/linux_drivers.html> here.

$ export PATH=$PATH:/usr/local/share/openvswitch/scripts

$ export DB\_SOCK=/usr/local/var/run/openvswitch/db.sock

$ /usr/src/dpdk-20.11/usertools/dpdk-devbind.py -b=vfio-pci 0000:03:00.1 (you can check your pci-address using ifconfig. In our case, 10G NIC was 0000:03:00.1)

$ /usr/src/dpdk-20.11/usertools/dpdk-devbind.py -s will make you see the changes.

In black server (if you are the one who is viewing this documentation on our lab), we have set an iommu on it through bios and /etc/grub.d (iommu\_pt= sth).

You also have to set huge pages as mentioned here.

<https://docs.openvswitch.org/en/latest/intro/install/dpdk/> or <https://doc.dpdk.org/guides/linux_gsg/sys_reqs.html>

You can see huge pages in cat /proc/meminfo

We start ovs-dpdk using these steps

$ovs-ctl --no-ovs-vswitchd start

$ovs-vsctl --no-wait set Open\_vSwitch . other\_config:dpdk-init=true

$ovs-ctl --no-ovsdb-server --db-sock="$DB\_SOCK" start

$ovs-vsctl --no-wait set Open\_vSwitch . other\_config:dpdk-init=true other\_config:dpdk-lcore-mask=0x2 other\_config:dpdk-socket-mem="1024"

$ovs-vsctl set Open\_vSwitch . other\_config:pmd-cpu-mask=0x4 // (use cpu==2)

$sudo ovs-vsctl add-br br0 -- set bridge br0 datapath\_type=netdev // set bridge

$sudo ovs-vsctl add-port br0 vhost-user0 -- set Interface vhost-user0 type=dpdkvhostuser //attach port

$ovs-vsctl add-port br0 dpdk-p0 -- set Interface dpdk-p0 type=dpdk options:dpdk-devargs=0000:03:00.1 //attach NIC port

Now, dpdkvhostuser is for container. We receive packets through dpdk-p0, transfer to dpdkvhostuser port which is named vhost-user0.

$ovs-ofctl add-flow br0 dl\_type=0x0800,nw\_src=10.1.2.3/24,actions=output:"vhost-user0"

$ovs-ofctl add-flow br0 in\_port="dpdk\_p0",dl\_type=0x800,action=output:"vhost-user0"

Two command is similar.

If you want to set bridge’s IP to receive packets from network,

$ip addr add 10.0.0.17/24 dev br0

$ip link set br0 up

these commands will help. Then, this bridge is visible from ifconfig.

$ovs-ofctl dump-flows br0

$ovs-ofctl dump-ports br0

$ovs-vsctl show

The dump commands show the information of corresponding commands.

docker.io/bruzn/ubuntunetplus is the container image that I downloaded few necessary packages for compiling DPDK inside container. So, you may pull the image and use it.

$ sudo docker run -it --privileged -v /mnt/huge:/mnt/huge -v /usr/local/var/run/openvswitch:/var/run bruzn/ubuntunetplus:2.0 /bin/bash

by above command, you can run docker. However, if you mind using shared memory with host, you can use --ipc=host option together. We have to mount /usr/local/var/run/openvswitch since that is the position where db.sock, which enables connection with host and container using dpdkvhostuser.

sudo docker run -it --cpus="1.0" will limit usage of cpu inside container.

To open DPDK application inside container, (the host has turned on OVS-DPDK for now.) you have to install any DPDK version. For compiling new version of DPDK,

```

apt update -y

apt-get upgrade -y

apt-get install libelf-dev libnuma-dev pkg-config cmake libbsd-dev libpcap-dev libkmod-dev oprofile -y

sudo apt install linux-headers-$(uname -r) -y

cd /usr/src

wget https://github.com/DPDK/dpdk/archive/refs/tags/v20.08.tar.gz

tar -xvzf v20.08.tar.gz

git clone https://github.com/libbpf/libbpf.git

cd libbpf/src

make && make install

cp /usr/lib64/libbpf\* /usr/lib

cd /usr/lib

rm libbpf.so

rm libbpf.so.0

ln -s libbpf.so.0.5.0 libbpf.so

ln -s libbpf.so.0.5.0 libbpf.so.0

ln -s libbpf.so.0.4.0 libbpf.so

ln -s libbpf.so.0.4.0 libbpf.so.0

cd /usr/src/dpdk-20.08

meson -Dexamples=all build

ninja -C build

ninja -C build install

ldconfig

cd ~

wget https://github.com/pktgen/Pktgen-DPDK/archive/refs/tags/pktgen-20.10.0.tar.gz

tar -xvzf pktgen-20.10.0.tar.gz

cd Pktgen-DPDK-pktgen-20.10.0/

export RTE\_SDK=/usr/src/dpdk-20.08

export RTE\_TARGET=build

make

Here, we can use two types of DPDK application. One is dpdk-testpmd which can be found $DPDK\_DIR/build/app/dpdk-testpmd. This can be executed using

$./dpdk-testpmd -l 5-6 -n 4 -v --file-prefix=vhost --vdev 'eth\_vhost0,iface=vhost-net,queues=1,path=/var/run/vhost-net' -- -i --txd=1024 --rxd=1024 --nb-cores=1

and type start to start receive packet. Default mode is iofwd mode.

The other is pktgen.

You can modify pktgen/cfg/default.cfg and then execute

$pktgen/tools/run.py default

or,

$ sudo -E usr/local/bin/pktgen -l 2,3-4 -n 2 --proc-type auto --log-level 8 --file-prefix pg -w 04:00.1 -- -v -T -P -j -m [3:4].0 -f themes/black-yellow.theme

or,

$export DPDK\_PARAMS="-c 0x19 --master-lcore 3 -n 1 --socket-mem 1024 --file-prefix pktgen --no-pci --vdev=net\_virtio\_user3,mac=00:00:00:00:00:03,path=/var/run/vhost-net"

root@58356f1466a0:~/Pktgen-DPDK-pktgen-20.10.0/Builddir/app# echo $DPDK\_PARAMS

-c 0x19 --master-lcore 3 -n 1 --socket-mem 1024 --file-prefix pktgen --no-pci --vdev=net\_virtio\_user3,mac=00:00:00:00:00:03,path=/var/run/vhost-net

To execute using this parameter,

./pktgen $DPDK\_PARAMS -- -T -P -m “0.0,4.1”

<https://www.slideshare.net/MichelleHolley1/dpdk-in-containers-handson-lab>

This will help you use these commands easily.