# User Manual for DPDK based network I/O for application layer VNFs

This manual provides setup and usage of our implementation of packet distribution mechanism based on DPDK for application layer VNFs. These instructions work for **LINUX based Operating Systems** only. Please read **developer_manual.pdf** for design and implementation details of packet distribution mechanism based on DPDK.

# Chapter 1

# Introduction

We are doing a single host inter VM communication with mTCP over DPDK as our stack inside the VM. The VMs communicate with each other over BESS switch. This manual will show how to do the setup and run client and server programs over this stack. We would need to do the BESS setup in the host and DPDK setup in the VMS as shown in this manual.

# Chapter 2

# Host-Setup

The host OS is Ubuntu 16.04. The hypervisor used was qemu-kvm (qemu version 2.5.0). The software swotch used is BESS which runs over DPDK. The installation of BESS is as shown below.

## 2.1 BESS installation

```
- clone BESS from git as follows:
  $ git clone
   ↪  https://github.com/NetSys/bess/tree/345a9fad7147cf075c683ba6997db8cc1a9f358b
$ cd bess/
$ sudo apt-get install -y software-properties-common
$ sudo apt-add-repository -y ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install -y ansible
$ ansible-playbook -K -i localhost, -c local env/bess.yml
  [Note: Here if installing docker is giving trouble then open env/bess.yml
   ↪  file and comment docker.yml line using a '#' and run this command again]
$ sudo reboot
$ cd bess
$ ./build.py
```

Refer https://github.com/NetSys/bess/wiki/Build-and-Install-BESS for more information.

## 2.2 BESS Forwarding Script

The forwarding script needed to connect the VMs is provided in the folder with name forward_script.bess. To run this script execute the following commands:

```
- allocate huge page
  $ echo 16384 | sudo tee
   ↪  /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
- In case the bess.yml did not create the huge page directory create it
 ↪  manually using the following command:
  $ sudo mkdir /mnt/huge
  $ sudo mount -t hugetlbfs nodev /mnt/huge
- add these lines in the file /etc/fstab:
    nodev /mnt/huge hugetlbfs defaults 0 0
- copy this bess script into samples folder:
  $ cp forward_script.bess bess/bessctl/conf/samples
$ cd bess/bessctl
$ ./bessctl
$ <disconnected> daemon start
$ localhost:10514$ run samples/forward_script
- To check if the script executed correctly:
$ localhost:10514$ show pipeline \\
- Result should be similar to this
```

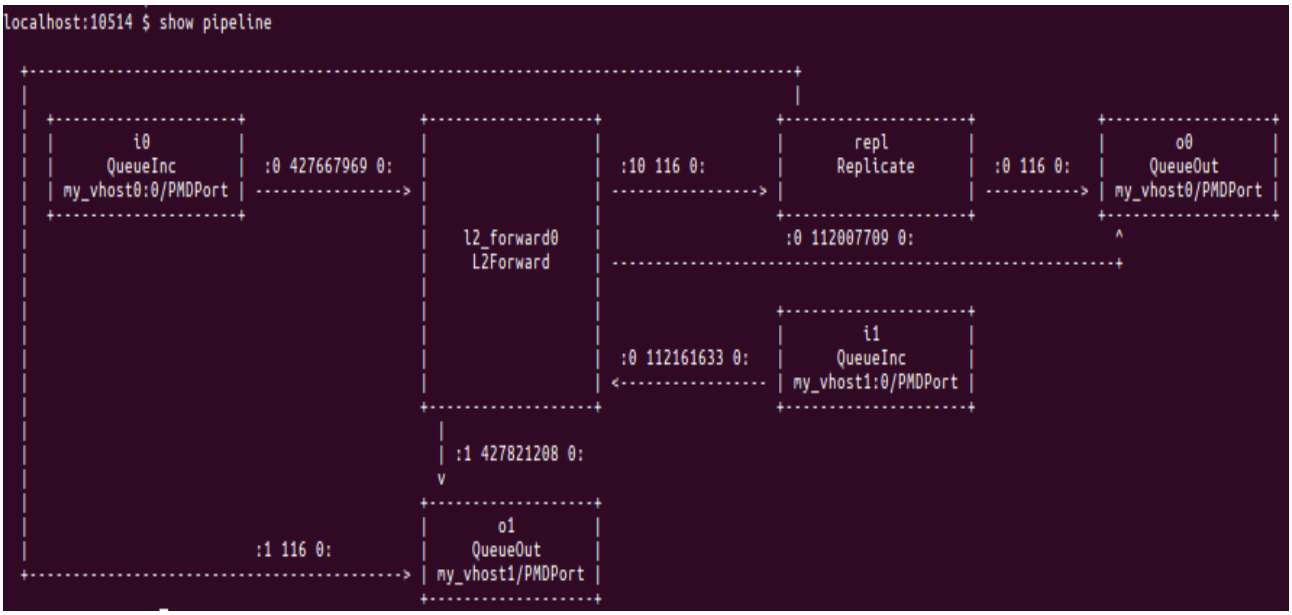


Figure 2.1: Show pipeline output [in case of single queue set up]

```
- On a different terminal:
  $ cd /tmp/
  $ chmod 777 my\_host*
- To stop BESS daemon on the previous terminal:
  $ localhost:10514 \$ daemon stop \\
  $ <disconnected>$ quit
```

## 2.3 KVM setup

To enable high performance networking in kvm do the following setup is needed (to be executed as super user):

- edit /etc/default/qemu-kvm and set VHOST_NET_ENABLED=1
- add the following line in /etc/libvirt/qemu.conf: user="root"

# Chapter 3

# VM Setup

Create two VMs using virt-manager with Ubuntu 16.04. One will be our server VM and other will be the client VM. After creating the VM to add BESS ports as vNIC to the VM add the following lines in the VM XML as follows as super user (sudo su):

```
$ export EDITOR=/usr/bin/gedit
$ virsh edit VM\_NAME
```

- Inside the VM xml edit the following :
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>

- Add the following code snippet before the </domain> line:

For single queue set up:

```
- For server VM:
    <qemu:commandline>
    <qemu:arg value='-chardev'/>
    <qemu:arg value='socket,id=mychr0,path=/tmp/my_vhost0.sock'/>
    <qemu:arg value='-netdev'/>
    <qemu:arg
     ↪  value='vhost-user,id=mydev0,chardev=mychr0,vhostforce=on,queues=1'/>
    <qemu:arg value='-device'/>
    <qemu:arg value='virtio-net-pci,mac=02:00:00:00:00:01,netdev=mydev0'/>
    <qemu:arg value='-m'/>
    <qemu:arg value='16384'/>
    <qemu:arg value='-object'/>
    <qemu:arg
     ↪  value='memory-backend-file,id=mem0,size=16384M,mem-path=/mnt/huge/,
     ↪  share=on'/>
    <qemu:arg value='-mem-prealloc'/>
    <qemu:arg value='-numa'/>
    <qemu:arg value='node,memdev=mem0'/>
  </qemu:commandline>
```

```
- For client VM:
    <qemu:commandline>
     <qemu:arg value='-chardev'/>
     <qemu:arg value='socket,id=mychr1,path=/tmp/my_vhost1.sock'/>
     <qemu:arg value='-netdev'/>
     <qemu:arg
      ↪  value='vhost-user,id=mydev1,chardev=mychr1,vhostforce=on,queues=1'/>
     <qemu:arg value='-device'/>
     <qemu:arg value='virtio-net-pci,mac=02:00:00:00:00:02,netdev=mydev1'/>
     <qemu:arg value='-m'/>
     <qemu:arg value='16384'/>
     <qemu:arg value='-object'/>
     <qemu:arg
      ↪  value='memory-backend-file,id=mem1,size=16384M,mem-path=/mnt/huge/,
      ↪  share=on'/>
     <qemu:arg value='-mem-prealloc'/>
     <qemu:arg value='-numa'/>
     <qemu:arg value='node,memdev=mem1'/>
   </qemu:commandline>
```

For multi- queue set up:

```
 - For server VM:
    <qemu:commandline>
     <qemu:arg value='-chardev'/>
     <qemu:arg value='socket,id=mychr0,path=/tmp/my_vhost0.sock'/>
     <qemu:arg value='-netdev'/>
     <qemu:arg
      ↪  value='vhost-user,id=mydev0,chardev=mychr0,vhostforce=on,queues=4'/>
     <qemu:arg value='-device'/>
     <qemu:arg value='virtio-net-pci,mac=02:00:00:00:00:01,netdev=mydev0,mq=on,
      ↪  vectors=10'/>
     <qemu:arg value='-m'/>
     <qemu:arg value='16384'/>
     <qemu:arg value='-object'/>
     <qemu:arg
      ↪  value='memory-backend-file,id=mem0,size=16384M,mem-path=/mnt/huge/,
      ↪  share=on'/>
     <qemu:arg value='-mem-prealloc'/>
     <qemu:arg value='-numa'/>
     <qemu:arg value='node,memdev=mem0'/>
   </qemu:commandline>

- For client VM:
    <qemu:commandline>
     <qemu:arg value='-chardev'/>
     <qemu:arg value='socket,id=mychr1,path=/tmp/my_vhost1.sock'/>
     <qemu:arg value='-netdev'/>
     <qemu:arg
      ↪  value='vhost-user,id=mydev1,chardev=mychr1,vhostforce=on,queues=4'/>
```

```
    <qemu:arg value='-device'/>
    <qemu:arg value='virtio-net-pci,mac=02:00:00:00:00:02,netdev=mydev1,mq=on,
     ↪  vectors=10'/>
    <qemu:arg value='-m'/>
    <qemu:arg value='16384'/>
    <qemu:arg value='-object'/>
    <qemu:arg
     ↪  value='memory-backend-file,id=mem1,size=16384M,mem-path=/mnt/huge/,
     ↪  share=on'/>
    <qemu:arg value='-mem-prealloc'/>
    <qemu:arg value='-numa'/>
    <qemu:arg value='node,memdev=mem1'/>
  </qemu:commandline>
```

In the above snippets /tmp/my_vhost0.sock and /tmp/my_vhost1.sock are the ports created using the BESS script. 16384M is the RAM size of the VM. Change this value (used twice in this xml) according to your VM RAM.

Before we start the VMs we need to set CPU flags. See figure 3.1, open VM setting then go to processor and open "configuration" option. After that click on "Copy host CPU configuration". It will simply copy all host configuration to VM and all required flags will be set.

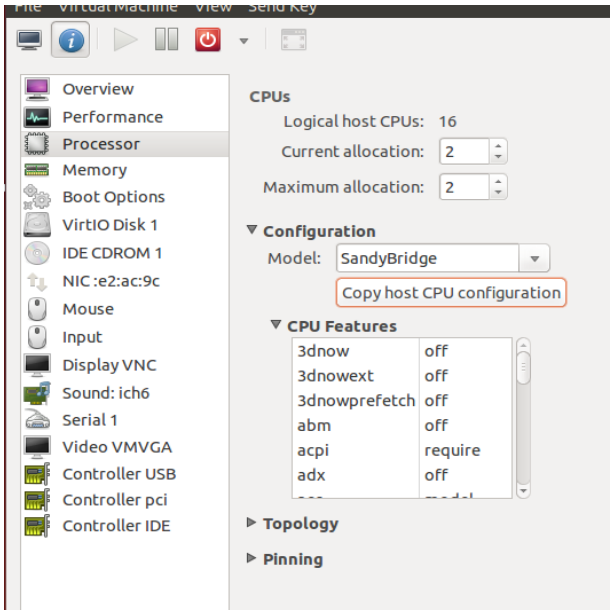


Figure 3.1: Setting CPU flags for VM

Once this setup is done swicth on the VMs through virt-manager. If you face an error (huge page access denied) execute the following commands:

```
$ sudo apt-get install apparmor-utils
$ sudo aa-complain /etc/apparmor.d/libvirt/libvirt-<uuid>
  [Note: (uuid can be found in the VM XML file)]
```

## 3.1 Setup stack inside VM

Install DPDK on each VM.

```
- System Requirement:
    - Hugepage support:
      - Reserve huge page memory:
        $ mkdir /mnt/huge
        $ mount -t hugetlbfs nodev /mnt/huge
      - Mount point can be made permanent across reboots, by opening
       ↪  /etc/fstab and add this line at the end:
        $ nodev /mnt/huge hugetlbfs defaults 0 0
      - Change number of huge pages (need to be executed each time whenever
       ↪  system starts):
        $ echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
- Download current version DPDK source code from "http://dpdk.org/download"
- Set up the environmental variables required by DPDK:
    $ export RTE_SDK=<DPDKInstallDir>
    $ export RTE_TARGET=x86_64-native-linuxapp-gcc
      (For permanent entry add these lines to $HOME/.bashrc)
- Installation:
    $ unzip DPDK-<version>.zip
    $ cd DPDK-<version>
    $ make install T=x86_64-native-linuxapp-gcc
-Steps that need to be executed each time whenever system starts
    - Change number of huge pages:
        $ echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
    - Bind NIC to dpdk driver:
        $ cd $RTE_SDK
        $ sudo modprobe uio_pci_generic
        $ sudo insmod x86_64-native-linuxapp-gcc/kmod/rte_kni.ko
        $ sudo ifconfig <NIC-name e.g ens3> down
        $ sudo ./usertools/dpdk-devbind.py --bind=uio_pci_generic <NIC-name>
    - Run kni in each VM using following commands:
        $ cd $RTE_SDK
        $ cd examples/kni/build/app/
        $ make    #To be done only first time
        $ sudo ./kni -l 0,1 --lcores 0@0,1@0 -- -p 0x1 -P --config="(0,0,1)"
    - Assign ip and mac to dpdk nic:
    Inside VM1:
        $ sudo ifconfig vEth0 169.254.9.8 netmask 255.255.255.0 broadcast
         ↪  169.254.9.255
        $ sudo ifconfig vEth0 hw ether 02:00:00:00:00:01
    Inside VM2:
```

```
        $ sudo ifconfig vEth0 169.254.9.3 netmask 255.255.255.0 broadcast
         ↪  169.254.9.255
        $ sudo ifconfig vEth0 hw ether 02:00:00:00:00:02
- Once these VMs DPDK nic get there ip and mac, ctrl+c to exit from kni on
 ↪  each VM.
```

# Chapter 4

# Code-Execution

## 4.1 Installing C++ compilation dependencies

For compiling C++ code on DPDK the following files need to be changed in the dpdk folder:

- In dpdk_folder/mk/toolchain/gcc: change rte_vars.mk
- In dpdk_folder/mk/internal: change rte.compile-pre.mk

We have provided these files in the deps folder

## 4.2 creating executable folder

We have provided two folders corresponding to mTCP over DPDK and multi-core UDP over DPDK. You can choose any of the folders depending on what code you want to execute i.e single-core mTCP, multi-core mTCP and multi-core UDP. You need to make the following changes according to your environment.

- For server VM rename server.cpp to main.cpp
- For client VM rename client.cpp to main.cpp
- The number of application cores need to be specified in main.conf (num_cores) and main.cpp (MAX_THREADS). (NOTE: For multi-core mTCP or mUDP the number of application core would be 1 less than the total cores of VM in case of 1 core provided to DPDK and, 2 less in case 2 core is provided to DPDK).
- The option of 1 core DPDK or 2 core DPDK can be set by using option OPT_KB in dpdk_api.h.
- Set your VM IP in by setting the MY_IP_(1,2,3,4) variables in dpdk_api.h

## 4.3 code execution

```
-Inside the folder created above on both VMs:
    $ make clean
    $ make
- Check to see if inside the folder a build/app/ folder is created with dpdkapi
 ↪  executable
- For executing the code
    $ sudo ./build/app/dpdkapi -c NO_OF_CORES -- -p NO_OF_PORTS
- The number of cores need to be provided using core mask that is if the VM has 3
 ↪  cores the mask would be 3 for (111 for all cores). If 4 cores the mask would
 ↪  be f (1111 for all cores) and 1f for 5 cores.
- The NO_OF_PORTS would 1
```