HANDS-ON TUTORIAL: OPEN VSWITCH* + DPDK* IN NFV USE CASE

Irene Liew

Intel Corporation

November 7th, 2016

NOTE

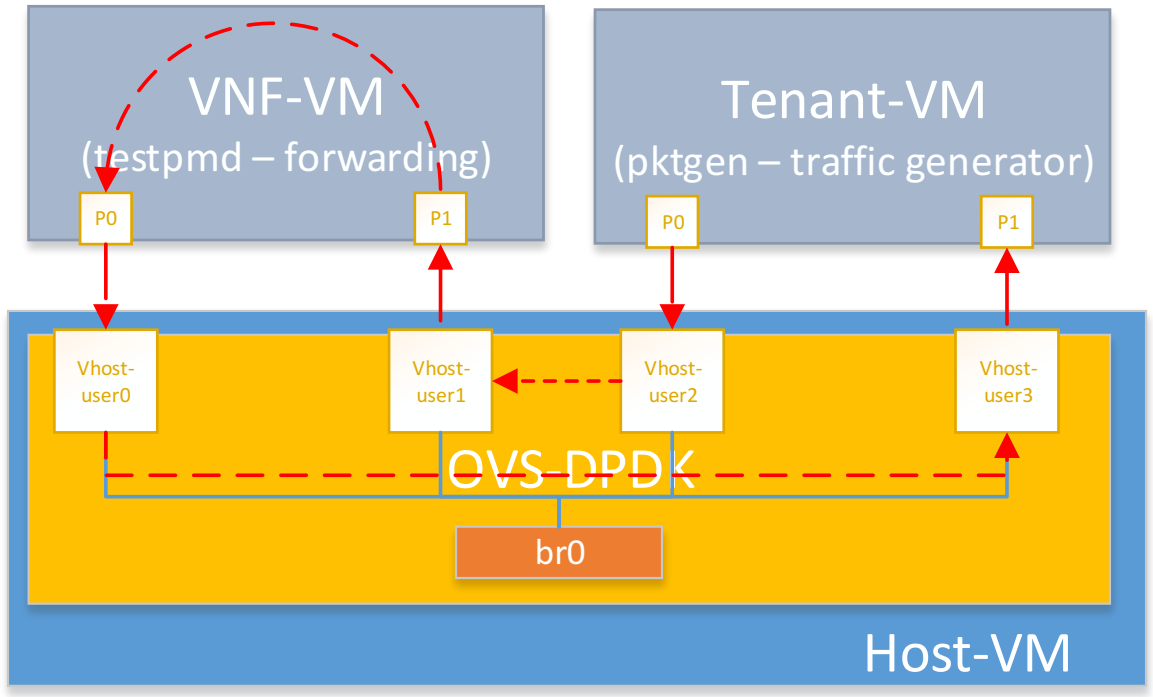
These slides were originally presented as part of a hands-on lab at the IEEE NFV/SDN conference in November 2016. They have been modified to make them more relevant to an audience that does not have access to the resources that were available at the time.

THE NEW CENTER OF POSSIBILITY

GOAL

- Provide hands-on in DPDK* application usage in the NFV use case
  - Focus on the functional setup and usage in NFV use case
  - Setup and hands-on exercise are not optimized for performance

OPEN VSWITCH* (OVS) + DPDK* VNF SETUP



Legend:

Traffic flow

OVS bridge connection

- Host-VM is the host machine that runs the OVS-DPDK and hosts VNF-VM and Tenant-VM:
  a. VNF-VM: VM that would run DPDK testpmd application – packet forwarding
  b. Tenant-VM: VM that would run pktgen application as a traffic generator
- Tenant-VM generates packets to VNF-VM, which would then be forwarded by VNF-VM back to the Tenant-VM.
- Tenant-VM shows throughput in packets/second and Mbps.
- Host-VM creates OVS bridge (br0) and 4 x vhost-user ports with each 2 ports attached to VNF-VM and Tenant-VM
- By default the 1 PMD core thread is created in OVS+DPDK. All 4 vhost-user ports are sharing a core.

LAUNCH VNF-VM & TENANT-VM

1. Login to Host-VM using this credential:
   User: user
   Password: password
2. After login, go to /home/user/training/dpdk-lab directory:
   # cd /home/user/training/dpdk-lab
   # ls
3. Run the following script files in order:
   a. # sudo sh 01_start_ovs.sh //start OVS-DPDK
   b. # sudo sh 02_createports_ovs.sh //create ports in OVS-DPDK
   c. # sudo sh 03_addroutes_vm-vm.sh //define flows
   d. # sudo sh 04_start_VNF-VM.sh //launch VNF-VM
   e. # sudo sh 05_start_TenantVM.sh //launch Tenant-VM

STARTING THE EXERCISE

Requirement: putty session to login to the setup consoles.

1. Login to VNF-VM and Tenant-VM using this credential:
   User: user
   Password: password
2. After login to the VMs, do the following: # sudo su -
3. Tenant-VM shows the Pktgen application is running.
4. Testpmd application is located in VNF-VM. Go to /root/dpdk-16.07/app/testpmd directory:
   # cd /root/dpdk-16.07/app/testpmd
   # ls

```
[root@vnf-vm test-pmd]# ls
build       flowgen.c       parameters.c  testpmd
cmdline.c   flowgen.o       iofwd.o       parameters.o  testpmd.c
cmdline.o   icmpecho.c      macfwd.c      _postbuild    testpmd.h
config.c    icmpecho.o      macfwd.o      _postinstall  testpmd.map
config.o    ieee1588fwd.c   macswap.c     _preinstall   testpmd.o
csumonly.c  _install        macswap.o     rxonly.c      txonly.c
csumonly.o  iofwd.c         Makefile      rxonly.o      txonly.o
```

SIMPLE FORWARDING: IOFWD.C

- The file we will be editing is iofwd.c in function static void pkt_burst_io_forward(struct fwd_stream *fs)

  # vi iofwd.c

- This function forwards packets in I/O mode by forwarding packets “as-is”.
- It does not access to packets data.
- Coding instructions:
  - Implement the code for receiving packets and transmitting packets

Note:

- Receive and transmit in a burst of packets
- Use rte_eth_rx_burst and rte_eth_tx_burst function calls to implement the received and transmit
- rte_eth_* functions are defined in:

  /root/dpdk-16.07/lib/librte_ether/rte_ethdev.h
- rte_mbuf structure is defined in:

  /root/dpdk-16.07/lib/librte_mbuf/rte_mbuf.h

```
static void
pkt_burst_io_forward(struct fwd_stream *fs)
{
 80      struct rte_mbuf *pkts_burst[MAX_PKT_BURST];
 81      uint16_t nb_rx;
 82      uint16_t nb_tx;
 83      uint32_t retry;
    ...

 95      /*
 96       * Receive a burst of packets and forward them.
 97       */
 98      nb_rx = rte_eth_rx_burst(fs->rx_port, fs->rx_queue,
 99                               pkts_burst, nb_pkt_per_burst);
100
101     if (unlikely(nb_rx == 0))
102         return;
103     fs->rx_packets += nb_rx;
   ...
107      /*
108      * Transmit  a burst of packets
109      */
110     nb_tx = rte_eth_tx_burst(fs->tx_port, fs->tx_queue,
111                              pkts_burst, nb_rx);
112     /*
113      * Retry if necessary
114      */
      if (unlikely(nb_tx < nb_rx) && fs->retry_enabled) {
          retry = 0;
          while (nb_tx < nb_rx && retry++ < burst_tx_retry_num) {
              rte_delay_us(burst_tx_delay_time);
              nb_tx += rte_eth_tx_burst(fs->tx_port, fs->tx_queue,
                       &pkts_burst[nb_tx], nb_rx - nb_tx);
          }
      }
      fs->tx_packets += nb_tx;

      if (unlikely(nb_tx < nb_rx)) {
          fs->fwd_dropped += (nb_rx - nb_tx);
          do {
              rte_pktmbuf_free(pkts_burst[nb_tx]);
          } while (++nb_tx < nb_rx);
      }
```

RTE_ETH_RX_BURST

- Retrieve a burst of input packets from a receive queue of an Ethernet device. The retrieved packets are stored in *rte_mbuf* structures whose pointers are supplied in the *rx_pkts* array.

Function:

```
rte_eth_rx_burst(uint8_t port_id, uint16_t queue_id,
                 struct rte_mbuf **rx_pkts, const uint16_t nb_pkts)
```

@param port_id: The port identifier of the Ethernet device.

@param queue_id: The index of the receive queue from which to retrieve input packets. The value must be in the range [0, nb_rx_queue - 1] previously supplied to rte_eth_dev_configure(). Default nb_rx_queue is 1

@param rx_pkts: The address of an array of pointers to *rte_mbuf* structures that must be large enough to store *nb_pkts* pointers in it.

@param nb_pkts: The maximum number of packets to retrieve. User defined value for packet burst size. Default value is 32.

nb_rx = the number of packets actually retrieved, which is the number of *rte_mbuf* data structures effectively supplied into the *rx_pkts* array.

```
static void
pkt_burst_io_forward(struct fwd_stream *fs)
{
 80      struct rte_mbuf *pkts_burst[MAX_PKT_BURST];
 81      uint16_t nb_rx;
 82      uint16_t nb_tx;
 83      uint32_t retry;
    ...

 95      /*
 96       * Receive a burst of packets and forward them.
 97       */
 98      nb_rx = rte_eth_rx_burst(fs->rx_port, fs->rx_queue,
 99                               pkts_burst, nb_pkt_per_burst);
100
101     if (unlikely(nb_rx == 0))
102             return;
103      fs->rx_packets += nb_rx;
    ...
```

RTE_ETH_TX_BURST

- Send a burst of output packets on a transmit queue of an Ethernet device.

Function:

rte_eth_tx_burst(uint8_t port_id, uint16_t queue_id,

struct rte_mbuf **tx_pkts, uint16_t nb_pkts)

@param port_id: The port identifier of the Ethernet device.

@param queue_id: The index of the transmit queue through which output packets must be sent. The value must be in the range [0, nb_tx_queue - 1] previously supplied to rte_eth_dev_configure(). Default nb_tx_queue is 1.

@param tx_pkts: The address of an array of *nb_pkts* pointers to *rte_mbuf* structures which contain the output packets.

@param nb_pkts: The maximum number of packets to transmit.

nb_tx = returns the number of packets it actually sent.

```
static void
pkt_burst_io_forward(struct fwd_stream *fs)
{
 80       struct rte_mbuf *pkts_burst[MAX_PKT_BURST];
 81       uint16_t nb_rx;
 82       uint16_t nb_tx;
 83       uint32_t retry;
      ...

...
 107      /*
 108      * Transmit  a burst of packets
 109      */
 110      nb_tx = rte_eth_tx_burst(fs->tx_port, fs->tx_queue,
 111                                       pkts_burst, nb_rx);
 112      /*
 113       * Retry if necessary
 114       */
       if (unlikely(nb_tx < nb_rx) && fs->retry_enabled) {
            retry = 0;
            while (nb_tx < nb_rx && retry++ < burst_tx_retry_num) {
                  rte_delay_us(burst_tx_delay_time);
                  nb_tx += rte_eth_tx_burst(fs->tx_port, fs->tx_queue,
                              &pkts_burst[nb_tx], nb_rx - nb_tx);
            }
      }
      fs->tx_packets += nb_tx;

       if (unlikely(nb_tx < nb_rx)) {
            fs->fwd_dropped += (nb_rx - nb_tx);
            do {
                  rte_pktmbuf_free(pkts_burst[nb_tx]);
            } while (++nb_tx < nb_rx);
      }
```

IOFWD.C

/root/dpdk-16.07/app/testpmd/testpmd.h:

```
/**
 * The data structure associated with a forwarding stream between a receive
 * port/queue and a transmit port/queue.
 */
struct fwd_stream {
	/* "read-only" data */
	portid_t   rx_port;   /**< port to poll for received packets */
	queueid_t  rx_queue;  /**< RX queue to poll on "rx_port" */
	portid_t   tx_port;   /**< forwarding port of received packets */
	queueid_t  tx_queue;  /**< TX queue to send forwarded packets */
	streamid_t peer_addr; /**< index of peer ethernet address of packets */

	unsigned int retry_enabled;

	/* "read-write" results */
	unsigned int rx_packets;  /**< received packets */
	unsigned int tx_packets;  /**< received packets transmitted */
	unsigned int fwd_dropped; /**< received packets not forwarded */
	unsigned int rx_bad_ip_csum ; /**< received packets has bad ip checksum */
	unsigned int rx_bad_l4_csum ; /**< received packets has bad l4 checksum */
#ifdef RTE_TEST_PMD_RECORD_CORE_CYCLES
	uint64_t     core_cycles; /**< used for RX and TX processing */
#endif
#ifdef RTE_TEST_PMD_RECORD_BURST_STATS
	struct pkt_burst_stats rx_burst_stats;
	struct pkt_burst_stats tx_burst_stats;
#endif
};
```

```
static void
pkt_burst_io_forward(struct fwd_stream *fs)
{
 80       struct rte_mbuf *pkts_burst[MAX_PKT_BURST];
 81       uint16_t nb_rx;
 82       uint16_t nb_tx;
 83       uint32_t retry;
      ...

 95       /*
 96        * Receive a burst of packets and forward them.
 97        */
 98       nb_rx = rte_eth_rx_burst(fs->rx_port, fs->rx_queue,
 99                   pkts_burst, nb_pkt_per_burst);
100
101      if (unlikely(nb_rx == 0))
102           return;
103       fs->rx_packets += nb_rx;
     ...
107       /*
108       * Transmit  a burst of packets
109       */
110       nb_tx = rte_eth_tx_burst(fs->tx_port, fs->tx_queue,
111                   pkts_burst, nb_rx);
112       /*
113        * Retry if necessary
114        */
        if (unlikely(nb_tx < nb_rx) && fs->retry_enabled) {
            retry = 0;
            while (nb_tx < nb_rx && retry++ < burst_tx_retry_num) {
                rte_delay_us(burst_tx_delay_time);
                nb_tx += rte_eth_tx_burst(fs->tx_port, fs->tx_queue,
                            &pkts_burst[nb_tx], nb_rx - nb_tx);
            }
        }
        fs->tx_packets += nb_tx;

        if (unlikely(nb_tx < nb_rx)) {
            fs->fwd_dropped += (nb_rx - nb_tx);
            do {
                rte_pktmbuf_free(pkts_burst[nb_tx]);
            } while (++nb_tx < nb_rx);
        }
```

TIPS IN RUNNING TESTPMD IN VNF-VM

1. Save the iofwd.c file:
   Press “Esc” key
   # :wq!
2. Go to /root/dpdk-16.07/app/testpmd and compile testpmd application:
   # cd /root/dpdk-16.07/app/testpmd
   #export RTE_SDK=/root/dpdk-16.07
   #export RTE_TARGET=x86_64-native-linuxapp-gcc
   #make
2. Start testpmd application by running:
   #sh /root/run_testpmd.sh
3. Once testpmd application is running, do the following:
   testpmd> set fwd io
   testpmd> start

```
PMD: bnxt_rte_pmd_init() called for (null)
EAL: PCI device 0000:00:03.0 on NUMA socket -1
EAL:   probe driver: 1af4:1000 rte_virtio_pmd
EAL: PCI device 0000:00:04.0 on NUMA socket -1
EAL:   probe driver: 1af4:1000 rte_virtio_pmd
EAL: PCI device 0000:00:05.0 on NUMA socket -1
EAL:   probe driver: 1af4:1000 rte_virtio_pmd
Interactive-mode selected
USER1: create a new mbuf pool <mbuf_pool_socket_0>: n=155456, size=2176, socket=
0
Configuring Port 0 (socket 0)
Port 0: 00:00:00:00:00:01
Configuring Port 1 (socket 0)
Port 1: 00:00:00:00:00:02
Checking link statuses...
Port 0 Link Up - speed 10000 Mbps - full-duplex
Port 1 Link Up - speed 10000 Mbps - full-duplex
Done
testpmd> set fwd io
Set io packet forwarding mode
testpmd> start
io packet forwarding - ports=2 - cores=1 - streams=2 - NUMA support disabled, MP
 over anonymous pages disabled
Logical Core 2 (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=64
  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> show port stats all
```

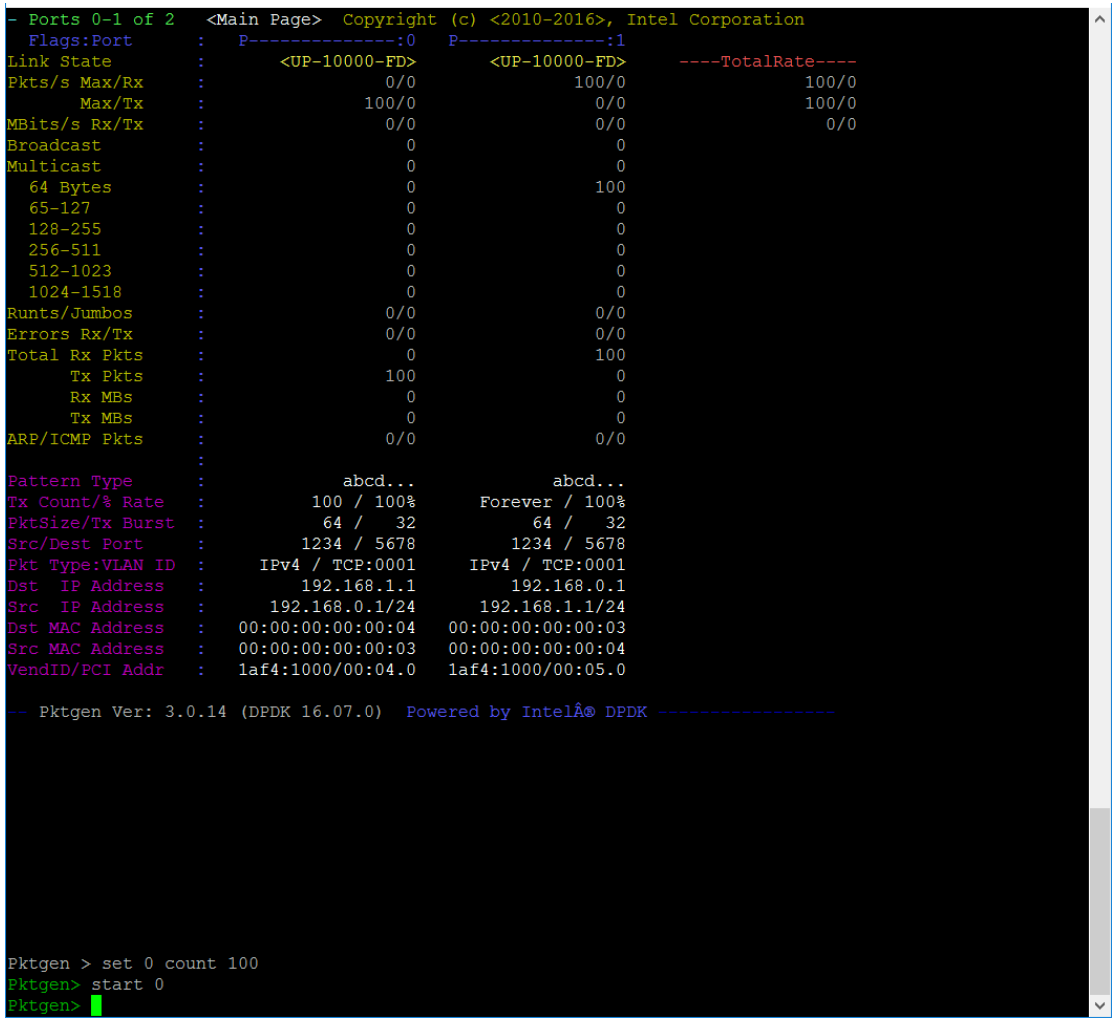
RUNNING PKTGEN IN TENANT-VM

Let's start generating 100 packets from Port 0:

1. Pktgen > set 0 count 100
2. To start traffic on Port0:
   Pktgen > start 0

To start pktgen to generate 10% line rate from Port 0:

1. Pktgen > set 0 rate 10
2. To start traffic on Port0:
   Pktgen > start 0



HEX DUMP OF PACKETS IN PKTGEN

- Default packet pattern type is alphabet string (abcd…)
- To view the hex dump of the first packet to be sent on Port 0, type:

  Pktgen > pdump 0

```
00000000: 00 00 00 00 00 04 00 00 00 00 00 03 08 00 45 00 | ..............E.
00000010: 00 2E 18 37 00 00 04 06 1C 41 C0 A8 00 01 C0 A8 | ...7.....A......
00000020: 01 01 04 D2 16 2E 12 34 56 78 12 34 56 90 50 10 | .......4Vx.4V.P.
00000030: 20 00 FF E6 00 00 77 78 79 7A 30 31 | | | | | .....wxyz01
```

OTHER TIPS IN PKTGEN RUNTIME COMMAND LINE

- To clear the statistics for all ports, type:
  Pktgen > clr
- To reset the configuration for all ports, type:
  Pktgen > rst
- To set DEST MAC address on Port 0, type:
  Pktgen > set mac 0 00:00:00:00:00:02
- Other Pktgen options:
  Pktgen> help

MONITOR TRAFFIC IN VNF-VM

- To monitor the Tx and Rx traffic in the testpmd application:

  testpmd> show port stats all

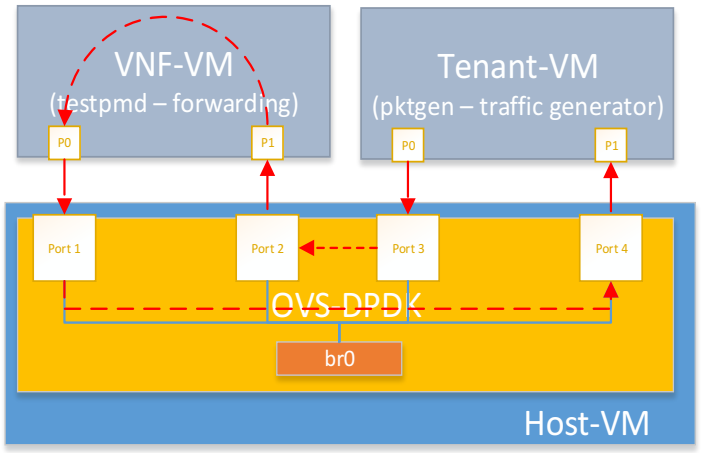
```
testpmd> show port stats all

  ######################## NIC statistics for port 0  ########################
  RX-packets: 0          RX-missed: 0          RX-bytes:  0
  RX-errors: 0
  RX-nombuf:  0
  TX-packets: 100        TX-errors: 0          TX-bytes:  6000

  Throughput (since last show)
  Rx-pps:            0
  Tx-pps:    368695647
  ############################################################################

  ######################## NIC statistics for port 1  ########################
  RX-packets: 100        RX-missed: 0          RX-bytes:  6000
  RX-errors: 0
  RX-nombuf:  0
  TX-packets: 0          TX-errors: 0          TX-bytes:  0

  Throughput (since last show)
  Rx-pps:    368695605
  Tx-pps:            0
  ############################################################################
```

  Note: 100 packets are received at Port 1 and transmit out from Port 0

- To clear the ports statistics in the testpmd application:

  testpmd> clear port stats all

WORKING VNF FORWARDING APPLICATION: FWD IO



Legend:

Traffic flow

OVS bridge connection

Tenant-VM

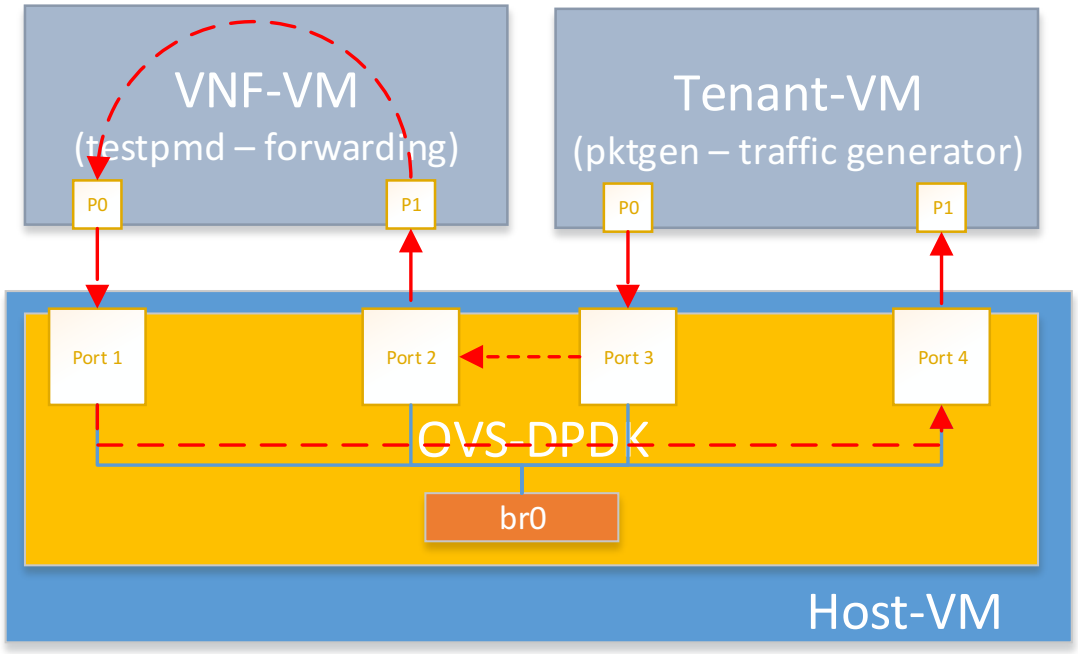
```
| Flags:Port      :   P--------------:0   P--------------:1
Link State        :   P--------------:0   P--------------:1     ----TotalRate---
-                 :        <UP-10000-FD>        <UP-10000-FD>              0/
0                 :          10944/8224           11648/7744          21696/15968
8                 :        726336/629632        710464/641216     1382112/1270848
4                 :                5/423                5/430               10/854
0                 :                    0                    0
  64 Bytes        :                    0                    0
  65-127          :              6328576              6353512
  128-255         :                    0                    0
  256-511         :                    0                    0
  512-1023        :                    0                    0
  1024-1518       :                    0                    0
Runts/Jumbos      :                    0                    0
Errors Rx/Tx      :                  0/0                  0/0
Total Rx Pkts     :                  0/0                  0/0
      Tx Pkts     :              6325248              6350696
      Rx MBs      :            505662272            507402720
      Tx MBs      :                 4250                 4267
ARP/ICMP Pkts     :               339805               340974
                  :                  0/0                  0/0
Pattern Type      :                abcd...              abcd...
Tx Count/% Rate   :       Forever / 100%       Forever / 100%
PktSize/Tx Burst  :            64 /   32            64 /   32
Src/Dest Port     :          1234 / 5678          1234 / 5678
Pkt Type:VLAN ID  :      IPv4 / TCP:0001      IPv4 / TCP:0001
Dst  IP Address   :         192.168.1.1          192.168.0.1
Src  IP Address   :       192.168.0.1/24       192.168.1.1/24
Dst MAC Address   :   00:00:00:00:00:04   00:00:00:00:00:03
Src MAC Address   :   00:00:00:00:00:03   00:00:00:00:00:04
VendID/PCI Addr   :   1af4:1000/00:04.0   1af4:1000/00:05.0

-- Pktgen Ver: 3.0.14 (DPDK 16.07.0)  Powered by IntelÂ® DPDK -----------------
Pktgen > start all
Pktgen>
```

VNF-VM

```
  ######################## NIC statistics for port 1  ########################
  RX-packets: 211507492  RX-missed: 0          RX-bytes:  12690451212
  RX-errors: 0
  RX-nombuf:  0
  TX-packets: 212011174  TX-errors: 0          TX-bytes:  12720671850

  Throughput (since last show)
  Rx-pps:       354098
  Tx-pps:       354942
  ############################################################################
testpmd>
testpmd> show port stats all

  ######################## NIC statistics for port 0  ########################
  RX-packets: 331695972  RX-missed: 0          RX-bytes:  19901759730
  RX-errors: 0
  RX-nombuf:  0
  TX-packets: 330449052  TX-errors: 0          TX-bytes:  19826944812

  Throughput (since last show)
  Rx-pps:       620370
  Tx-pps:       617297
  ############################################################################

  ######################## NIC statistics for port 1  ########################
  RX-packets: 330569764  RX-missed: 0          RX-bytes:  19834187532
  RX-errors: 0
  RX-nombuf:  0
  TX-packets: 331570086  TX-errors: 0          TX-bytes:  19894206570

  Throughput (since last show)
  Rx-pps:       617547
  Tx-pps:       620123
  ############################################################################
testpmd>
```

- Pktgen in Tenant-VM generates packets from Port 0 and transmits them to VNF-VM via vhost-user Port 3 -> Port 2 running in OVS-DPDK (Host-VM).
- A working testpmd application in VNF-VM receives the packets at P1 and transmits them out at P0 to Tenant-VM via vhost-user Port 1 -> Port 4
- Observed transmit (Tx) packets and receive (Rx) packets on the Tenant-VM and VNF-VM ports.

PORT NUMBERING IN OVS AND VM



Legend:

Traffic flow

OVS bridge connection

- Port numbering in the OVS Host-VM is shown via command line:

  #/home/user/ovs/utilities/ovs-ofctl show br0

- Output screen:

```
1(vhost-user0): addr:00:00:00:00:00:00
    config:     PORT_DOWN
    state:      LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
2(vhost-user1): addr:00:00:00:00:00:00
    config:     PORT_DOWN
    state:      LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
3(vhost-user2): addr:00:00:00:00:00:00
    config:     PORT_DOWN
    state:      LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
4(vhost-user3): addr:00:00:00:00:00:00
    config:     PORT_DOWN
    state:      LINK_DOWN
    speed: 0 Mbps now, 0 Mbps max
```

- In each VNF-VM and Tenant-VM view, the ports are shown as Port 0 and Port 1

PACKET MONITORING IN OVS-DPDK : HOST-VM

1. Check the vhost-user ports' stats (e.g. RX, Tx, drop, error, etc) in Host-VM:
   - #/home/user/ovs/utilities/ovs-ofctl dump-ports br0
   - Run the following script to get the real-time stats:

     # sh /home/user/training/dump-ports.sh

```
Every 1.0s: /root/ovs/utilities/ovs-ofctl dump-ports br0

OFPST_PORT reply (xid=0x2): 5 ports
  port  4: rx pkts=38467002, bytes=2308023618, drop=0, errs=0, frame=?, over=?, crc=?
           tx pkts=526754, bytes=31630962, drop=37265166, errs=?, coll=?
  port  2: rx pkts=37775940, bytes=2266589652, drop=0, errs=0, frame=?, over=?, crc=?
           tx pkts=37801705, bytes=2268105744, drop=251041, errs=?, coll=?
  port  1: rx pkts=37792073, bytes=2267553120, drop=0, errs=0, frame=?, over=?, crc=?
           tx pkts=37785865, bytes=2267155062, drop=681120, errs=?, coll=?
  port  3: rx pkts=38052763, bytes=2283169560, drop=0, errs=0, frame=?, over=?, crc=?
           tx pkts=520202, bytes=31242354, drop=37255585, errs=?, coll=?
  port LOCAL: rx pkts=0, bytes=0, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
```

2. Check the traffic flows among the vhost-user ports in Host-VM:
   - #/home/user/ovs/utilities/ovs-ofctl dump-flows br0
   - Run the following script to get the real-time stats:

     # sh /home/user/training/dump-flows.sh

```
Every 1.0s: /root/ovs/utilities/ovs-ofctl dump-flows br0

NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2976.679s, table=0, n_packets=0, n_bytes=0, idle_age=0, ip,in_port=2 actions=output:3
 cookie=0x0, duration=2976.677s, table=0, n_packets=158970348, n_bytes=9538224264, idle_age=0, ip,in_port=3 actions=output:2
 cookie=0x0, duration=2976.676s, table=0, n_packets=156882706, n_bytes=9412988304, idle_age=0, ip,in_port=1 actions=output:4
 cookie=0x0, duration=2976.674s, table=0, n_packets=0, n_bytes=0, idle_age=0, ip,in_port=4 actions=output:1
```

REREFENCES

- DPDK Programmer's Guide: http://dpdk.org/doc/guides/prog_guide/index.html
- Testpmd Application User Guide: http://dpdk.readthedocs.io/en/v16.04/testpmd_app_ug/index.html
- Sample Application User Guide: http://dpdk.readthedocs.io/en/v16.04/sample_app_ug/index.html
- ONP 2.1 Performance Test Report (https://download.01.org/packet-processing/ONPS2.1/Intel_ONP_Release_2.1_Performance_Test_Report_Rev1.0.pdf)
- How to get best performance with NICs on Intel platforms with DPDK: http://dpdk.org/doc/guides-2.2/linux_gsg/nic_perf_intel_platform.html
- Open vSwitch 2.5.0 documentation and installation guide:
  - http://openvswitch.org/support/dist-docs-2.5/
  - https://github.com/openvswitch/ovs/blob/master/INSTALL.DPDK-ADVANCED.md