Selected Topics on Software Defined Networking

EECE.7290

Instructor: *Prof. Yan Luo*

Lab 4: Programming Data Plane Using DPDK
Hand in Date: 21/04/17
Due Date: 21/04/17

By,
- Naga Ganesh Kurapati
ID:01592079

Table of contents:

a. Purpose

The main purpose this lab is to learn about development of data plane applications using DPDK (Data Plane Development Kit). We need to parse the different fields of Ethernet and IP headers from the received packets.

b. Lab Procedure

The code should be executed on a dedicated server (129.63.205.39). It has 100Gbps NIC and receives continuous network data flows from another server generating these packets.

Initially do ssh from the local machine and supply password

```
ssh YOUR_ACCOUNT@129.63.205.39
```

clone the git repository as shown

```
git clone https://github.com/ACANETS/sdn-course
```

Compile and execute the program as shown

cd sdn-course/dpdk-lab // Enter to the sub-folder

```
Follow the steps in README as below:
1. sudo su
2. export RTE_SDK=/home/acanets/Downloads/dpdk-16.07
3. make
4. ./build/rx_demo -c 0x01
```

With the sample code the output is printed as below:

```
nagak@acanets-PowerEdge-R730:~/sdn-course$ cd dpdk-lab/
nagak@acanets-PowerEdge-R730:~/sdn-course/dpdk-lab$ ls
main.c  Makefile  README
nagak@acanets-PowerEdge-R730:~/sdn-course/dpdk-lab$ sudo su
[sudo] password for nagak:
root@acanets-PowerEdge-R730:/home/nagak/sdn-course/dpdk-lab# export RTE_SDK=/home/acanets/Downloads/dpdk-16.07
root@acanets-PowerEdge-R730:/home/nagak/sdn-course/dpdk-lab# make
/home/nagak/sdn-course/dpdk-lab/Makefile:19: warning: overriding commands for target `clean'
/home/acanets/Downloads/dpdk-16.07/mk/rte.app.mk:266: warning: ignoring old commands for target `clean'
  CC main.o
  LD rx_demo
  INSTALL-APP rx_demo
  INSTALL-MAP rx_demo.map
root@acanets-PowerEdge-R730:/home/nagak/sdn-course/dpdk-lab# ./build/rx_demo -c 0x01
EAL: Detected 12 lcore(s)
EAL: No free hugepages reported in hugepages-1048576kB
EAL: Probing VFIO support...
PMD: bnxt_rte_pmd_init() called for (null)
EAL: PCI device 0000:83:00.0 on NUMA socket 1
EAL:   probe driver: 15b3:1013 librte_pmd_mlx5
PMD: librte_pmd_mlx5: PCI information matches, using device "mlx5_0" (SR-IOV: false, MPS: false)
PMD: librte_pmd_mlx5: 1 port(s) detected
PMD: librte_pmd_mlx5: port 1 MAC address is 24:8a:07:29:c7:dc
PMD: librte_pmd_mlx5: 0x82bb00: RX queues number update: 0 -> 1
########## Setting UP RXQ For PORT 0 ##########
Hello from master core 0 !
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
```

Ehter types are printed as follows

```
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
lcore 0, received 4 packets in 10 seconds.
The ether_type of the packet is 86dd
lcore 0, received 4 packets in 10 seconds.
The ether_type of the packet is 86dd
lcore 0, received 3 packets in 10 seconds.
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
```

Now, we modify the sample program to print the source and destination IP addresses of the IPV4 packet. It can be seen below.

```
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
lcore 0, received 25 packets in 10 seconds.
The ether_type of the packet is 86dd
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
The ether_type of the packet is 86dd
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
lcore 0, received 6 packets in 10 seconds.
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
lcore 0, received 1 packets in 10 seconds.
The ether_type of the packet is 86dd
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
```

Now, when we try to print the number of unique ether types of the received packets. It can be seen below.

```
srcIp- 0.0.0.0 & destIp- 255.255.255.255
number of unique ether types: 2
lcore 0, received 3 packets in 10 seconds.
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
number of unique ether types: 2
lcore 0, received 3 packets in 10 seconds.
The ether_type of the packet is 86dd
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
The ether_type of the packet is 86dd
number of unique ether types: 2
lcore 0, received 18 packets in 10 seconds.
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
The ether_type of the packet is 800
srcIp- 0.0.0.0 & destIp- 255.255.255.255
The ether_type of the packet is 86dd
The ether_type of the packet is 86dd
```

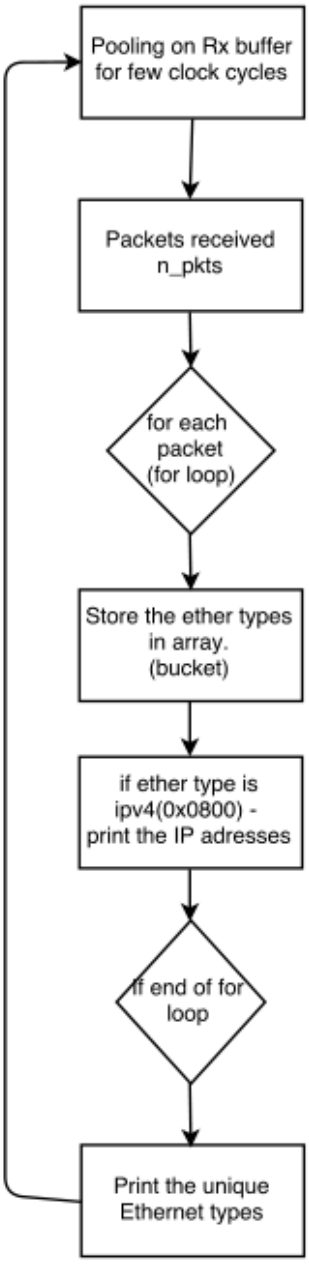
c. Software

Now let us see the software design:

From the flowchart we can see that, by polling IO method packets are received from the RX. The received packets are aprsed one by one. First ether types are stored in a array called bucket. And the ether type is ipv4 then IP address of source and destination is printed. Later, Unique ether types are printed.

To print the ip adresses

```
//check for ivp4 type
if(ntohs(ethernet->ether_type) == 0x0800)
          {
            //Print the ether type
            printf("The ether_type of the packet is %x \n",
                    ntohs(ethernet->ether_type));
            //Print source ip address
            printf("srcIp- %s",inet_ntoa(ipv4 -> ip_src));
            //Print destination ip address
            printf(" & destIp- %s \n",inet_ntoa(ipv4 -> ip_dst));
          }
```



Now to find unique ether types:
//To store ether types

```
bucket[b_index++] = ntohs(ethernet->ether_type);
```

//ntohs to convert network bytes to short
// inet_ntoa to convert network bytes to dot notion

```
//To print number of unique ether types
for(i=0; i<=b_index; i++)
{
 uint32_t j;
 for (j=0; j<i; j++)
      {
           if (bucket[i] == bucket[j])
                break;
       }
      if (i == j)
      {
        if(bucket[i]!=0)
       unique_ethpkt_no++;
      }
}
printf("number of unique ether types: %d\n",unique_ethpkt_no);
```

// Structure to parse ip packets

```
struct ip {
u_char ip_vhl; /* version *//* header length */
u_char ip_tos; /* type of service */
short ip_len; /* total length */
u_short ip_id; /* identification */
short ip_off; /* fragment offset field */
#define IP_DF 0x4000 /* dont fragment flag */
#define IP_MF 0x2000 /* more fragments flag */
u_char ip_ttl; /* time to live */
u_char ip_p; /* protocol */
u_short ip_sum; /* checksum */
struct in_addr ip_src,ip_dst;    /* source and dest address */
};
```

d. Trouble shooting:

We observe that the packets ether types are printed to seen which packets we received. It clearly prints 0x86dd(ipv6) and 0x0800 (ipv4). And, we have only two unique ether type packets.

e. Results:

We can see from the screen-shots that Source IP: 0.0.0.0 and Destination IP 255.255.255.255 is printed. And also unique number ether types are printed each time.

f. Appendix:

Source code for the project can be found at
https://github.com/ganeshkurapati/sdn-course/tree/master/dpdk-lab