SOFTWARE DEFINED NETWORKS

LAB 2

AKHILA NAIR

STUDENT ID: 01743358

HAND IN & DUE DATE: 5 March 2018

Estimated Hours of work: 10

1. **Purpose:**
   - Installing and configuring DPDK (Data Plane Development Kit).
   - Testing the SSH connection and controlling the kit remotely.
   - Learning the basic packet transmitting and receiving.
   - Get acquainted with DPDK components – data plane libraries.
   - Understanding the role of DPDK in high speed data packets transfer and processing.

2. **Lab Setup:**
   - Connect the laptop to DPDK box using the Ethernet cable via RJ45 Adapter.
   - Go to Network and sharing centre and change the adapter setting to "Set up IP Manually" and define static IP to 10.0.0.2 and subnet to 255.255.255.0
   - Now SSH into the system to log in the DPDK box using "ssh test@10.0.0.1" the user id as "test" and password as 'tester".
   - Download the file from GitHub and use WinScp to copy it into the DPDK box.
   - Now use putty to change the directory by using "cd /home/test.dpdk/lab2".
   - Choose the file you want to run for Ex: TX, then "cd TX"
   - Enter the "make" command.
   - Now to build the code use " sudo ./build/tx_demo -w 0000:03:00.0 --socket-mem 64 --file-prefix tx -c 0x02".
     This will run the main.c file from the TX folder.

3. **Software Design:**
   Major software and code changes is explained shortly in the section below,
   - **To set up the EAL:**

```
static void app_init_eal(int argc, char **argv)
{
  int status;
  status = rte_eal_init(argc, argv);
  if (status < 0) { rte_panic("EAL init error\n"); }
}
```

   - **struct PacketDetails**:
     It is used to store different types of ether types and count of received packets for each ether type. This structure is defined as shown below and has 2 members.

```
 typedef struct node PacketDetails;
struct node {
int32_t packet_ether_type; // destination host address
 int32_t packet_count; // source host address
};
```

- **To start the link:**

  static void app_init_link()
  rte_eth_dev_configure():
  This function is used to configure the number of queues for a port. For each port there is only one RX queue. The number of TX queues depend on the number of lcores.

```
  ret = rte_eth_dev_configure((uint8_t) portid, 1, 1, &port_conf);
 if (ret < 0)
     rte_exit(EXIT_FAILURE, "Cannot configure device: err=%d, port=%u\n", ret,
                                                                        portid);
```

- **IP address to structure mapping:**

  inet_ntoa: This is used to store IP address and store it inside structure.
  Ntohs: This function is used to store ether_type in the structure.

```
packet_stat[current].packet_ether_type = ntohs(ethernet->ether_type);
```

- **To enable promiscuous mode:**

```
if (link_temp.promisc)
{
rte_eth_promiscuous_enable(pmd_id);
}
```

  This function mean that this port will receive all the packets.

- **sniff_ip:**

```
struct sniff_ethernet
{
      u_char  dmac[6];    // destination host address
      u_char  smac[6];    // source host address
      u_short ether_type; //
};
```

  This function is used to store network layer details in the serialized manner. A pointer is created to the structure to access the packet details from it.

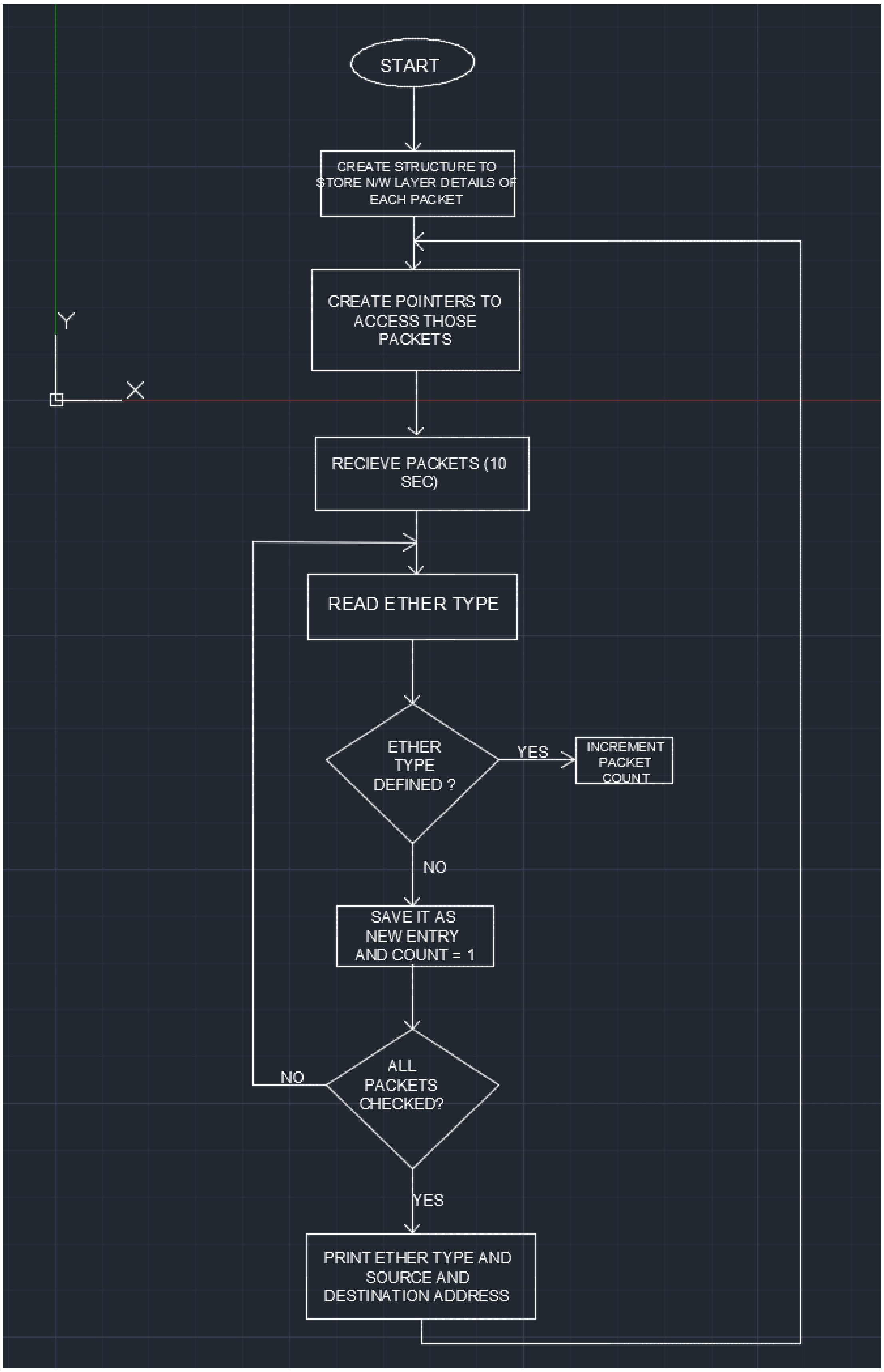
- **To retrieve the data from packet:**

```
  for(i=0; i<n_pkts; i++)
   {
 uint8_t* packet = rte_pktmbuf_mtod(pkts[i], uint8_t*);
 struct sniff_ethernet *ethernet = (struct sniff_ethernet*) packet;
bucket[b_index++] = ntohs(ethernet->ether_type);
if(ntohs(ethernet->ether_type)!=0x0800)
printf("The ether_type of the packet is %x \n", ntohs(ethernet->ether_type))
 };
```

- **Printing the source and destination address and ether type:**

```
 struct ip *ipv4 = (struct ip*) (packet+sizeof(struct sniff_ethernet));
if(ntohs(ethernet->ether_type) == 0x0800)
   {
   printf("The ether_type of the packet is %x \n", ntohs(ethernet->ether_type));
   printf("srcIp- %s",inet_ntoa(ipv4 -> ip_src));
  printf(" & destIp- %s \n",inet_ntoa(ipv4 -> ip_dst));
 }
```

FLOWCHART:



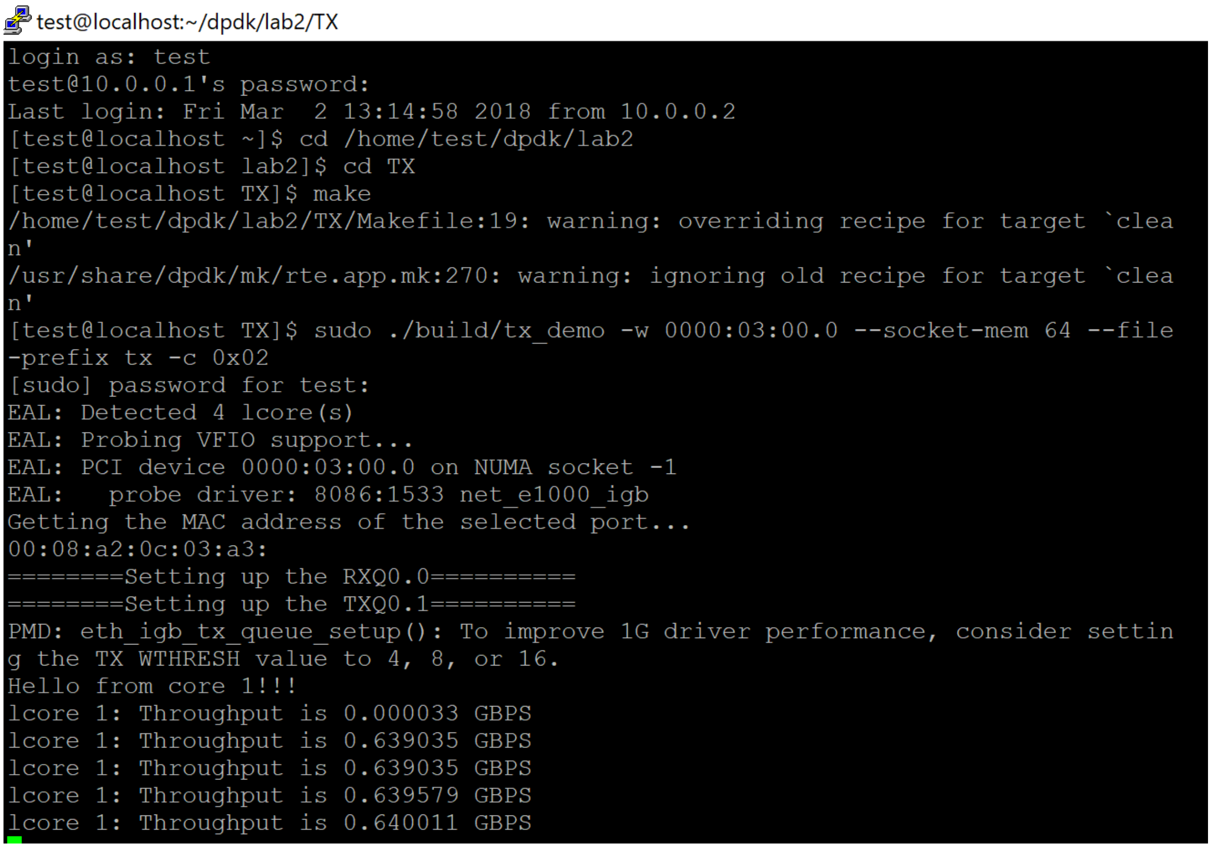
START
CREATE STRUCTURE TO STORE N/W LAYER DETAILS OF EACH PACKET
CREATE POINTERS TO ACCESS THOSE PACKETS
Y
X
RECIEVE PACKETS (10 SEC)
READ ETHER TYPE
ETHER TYPE DEFINED ?
YES
INCREMENT PACKET COUNT
NO
SAVE IT AS NEW ENTRY AND COUNT = 1
NO
ALL PACKETS CHECKED?
YES
PRINT ETHER TYPE AND SOURCE AND DESTINATION ADDRESS

4. Troubleshooting:

- I had difficulty in connecting the putty to DPDK box remotely as I was not able to SSH into it.
  After I switched of the Wi-Fi, I could successfully login to the DPDK box.
- Following the steps given on GitHub,I performed following command which led to killing of client server. Due to that I couldn't perform any other action ahead.
  "sudo systemctl stop dpdk-config"
  "sudo pkill -9 dhclient"
  "sudo dhclient -v"
  This problem got solved by following command,
  "sudo systemctl start dpdk-config".
- I had difficulty fetching IP address from the packets as I was not aware that packet details are stored in serialized manner.
  Therefore, after reading reference materials I understood that IP address of packet is present 14 bytes away from the start of the packet which will require a pointer to pint to that location.

5. Results:

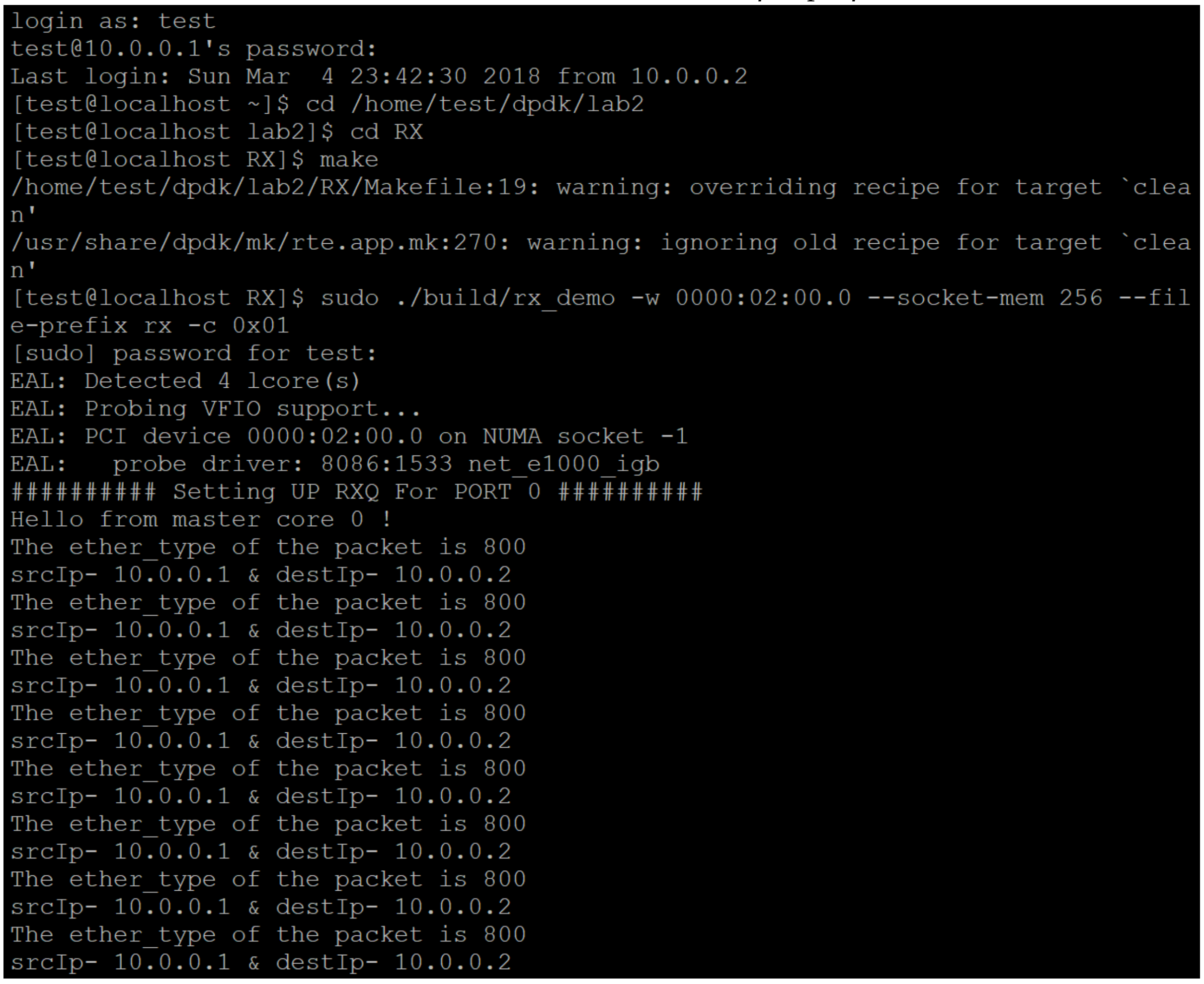
- The following image shows the terminal after running the TX side. The command used to run the TX code is :
  sudo ./build/tx_demo -w 0000:03:00.0 --socket-mem 64 --file-prefix tx -c 0x02



test@localhost:~/dpdk/lab2/TX

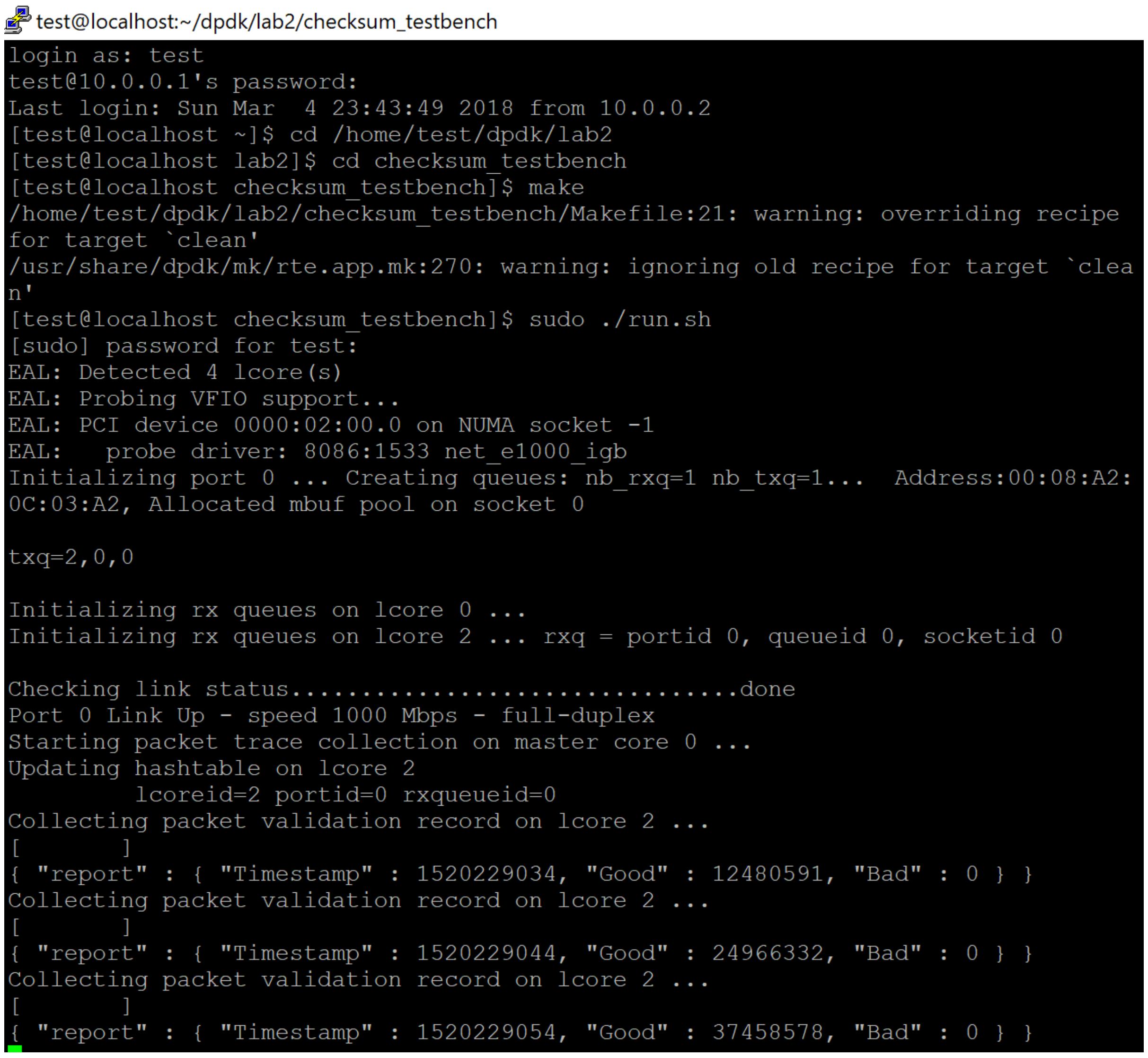
```
login as: test
test@10.0.0.1's password:
Last login: Fri Mar  2 13:14:58 2018 from 10.0.0.2
[test@localhost ~]$ cd /home/test/dpdk/lab2
[test@localhost lab2]$ cd TX
[test@localhost TX]$ make
/home/test/dpdk/lab2/TX/Makefile:19: warning: overriding recipe for target `clea
n'
/usr/share/dpdk/mk/rte.app.mk:270: warning: ignoring old recipe for target `clea
n'
[test@localhost TX]$ sudo ./build/tx_demo -w 0000:03:00.0 --socket-mem 64 --file
-prefix tx -c 0x02
[sudo] password for test:
EAL: Detected 4 lcore(s)
EAL: Probing VFIO support...
EAL: PCI device 0000:03:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1533 net_e1000_igb
Getting the MAC address of the selected port...
00:08:a2:0c:03:a3:
========Setting up the RXQ0.0==========
========Setting up the TXQ0.1==========
PMD: eth_igb_tx_queue_setup(): To improve 1G driver performance, consider settin
g the TX WTHRESH value to 4, 8, or 16.
Hello from core 1!!!
lcore 1: Throughput is 0.000033 GBPS
lcore 1: Throughput is 0.639035 GBPS
lcore 1: Throughput is 0.639035 GBPS
lcore 1: Throughput is 0.639579 GBPS
lcore 1: Throughput is 0.640011 GBPS
```

- The following image shows the terminal of RX side where the source and destination address get printed. The command used to run the code is:
  sudo ./build/rx_demo -w 0000:02:00.0 --socket-mem 256 --file-prefix rx -c 0x01



```
login as: test
test@10.0.0.1's password:
Last login: Sun Mar  4 23:42:30 2018 from 10.0.0.2
[test@localhost ~]$ cd /home/test/dpdk/lab2
[test@localhost lab2]$ cd RX
[test@localhost RX]$ make
/home/test/dpdk/lab2/RX/Makefile:19: warning: overriding recipe for target `clea
n'
/usr/share/dpdk/mk/rte.app.mk:270: warning: ignoring old recipe for target `clea
n'
[test@localhost RX]$ sudo ./build/rx_demo -w 0000:02:00.0 --socket-mem 256 --fil
e-prefix rx -c 0x01
[sudo] password for test:
EAL: Detected 4 lcore(s)
EAL: Probing VFIO support...
EAL: PCI device 0000:02:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1533 net_e1000_igb
########## Setting UP RXQ For PORT 0 ##########
Hello from master core 0 !
The ether_type of the packet is 800
srcIp- 10.0.0.1 & destIp- 10.0.0.2
The ether_type of the packet is 800
srcIp- 10.0.0.1 & destIp- 10.0.0.2
The ether_type of the packet is 800
srcIp- 10.0.0.1 & destIp- 10.0.0.2
The ether_type of the packet is 800
srcIp- 10.0.0.1 & destIp- 10.0.0.2
The ether_type of the packet is 800
srcIp- 10.0.0.1 & destIp- 10.0.0.2
The ether_type of the packet is 800
srcIp- 10.0.0.1 & destIp- 10.0.0.2
The ether_type of the packet is 800
srcIp- 10.0.0.1 & destIp- 10.0.0.2
The ether_type of the packet is 800
srcIp- 10.0.0.1 & destIp- 10.0.0.2
```

- The following image shows that when the test bench code is made to run, the program runs perfectly and we get a good image at the output terminal.
  The code should be first made executable by *chmod +x ./run.sh*
  And then to run the code: *sudo ./run.sh*



test@localhost:~/dpdk/lab2/checksum_testbench

```
login as: test
test@10.0.0.1's password:
Last login: Sun Mar  4 23:43:49 2018 from 10.0.0.2
[test@localhost ~]$ cd /home/test/dpdk/lab2
[test@localhost lab2]$ cd checksum_testbench
[test@localhost checksum_testbench]$ make
/home/test/dpdk/lab2/checksum_testbench/Makefile:21: warning: overriding recipe
for target `clean'
/usr/share/dpdk/mk/rte.app.mk:270: warning: ignoring old recipe for target `clea
n'
[test@localhost checksum_testbench]$ sudo ./run.sh
[sudo] password for test:
EAL: Detected 4 lcore(s)
EAL: Probing VFIO support...
EAL: PCI device 0000:02:00.0 on NUMA socket -1
EAL:   probe driver: 8086:1533 net_e1000_igb
Initializing port 0 ... Creating queues: nb_rxq=1 nb_txq=1...  Address:00:08:A2:
0C:03:A2, Allocated mbuf pool on socket 0

txq=2,0,0

Initializing rx queues on lcore 0 ...
Initializing rx queues on lcore 2 ... rxq = portid 0, queueid 0, socketid 0

Checking link status...............................done
Port 0 Link Up - speed 1000 Mbps - full-duplex
Starting packet trace collection on master core 0 ...
Updating hashtable on lcore 2
         lcoreid=2 portid=0 rxqueueid=0
Collecting packet validation record on lcore 2 ...
[       ]
{ "report" : { "Timestamp" : 1520229034, "Good" : 12480591, "Bad" : 0 } }
Collecting packet validation record on lcore 2 ...
[       ]
{ "report" : { "Timestamp" : 1520229044, "Good" : 24966332, "Bad" : 0 } }
Collecting packet validation record on lcore 2 ...
[       ]
{ "report" : { "Timestamp" : 1520229054, "Good" : 37458578, "Bad" : 0 } }
```

6. **Conclusion:**
   This program prints IP address of only IPv4 packets.
   At the end of every 10 seconds, program prints count of different types of packet received.
   UDP packets were converted to TCP packets.