SOFTWARE DEFINED NETWORKS

LAB 5

AKHILA NAIR
STUDENT ID – 01743358

HAND IN AND DUE DATE: 9th APRIL 2018

ESTIMATED HOURS OF WORKING: 10

1. **Purpose:**
   - To understand that single RX queue is not sufficient in high volume network.
   - How to use Receive Side Scaling to increase the stability of DPDK.
   - To understand the l3fwd example and ip-pipeline code and then inturn understand how to implement multiple core.
   - To control each lcore to work on its own Rx queue.

2. **Lab Setup:**
   - The setup remains the same as in other labs.
   - We have to integrate the lab 2 source code to Implement receive side scaling and initiate multiple cores.

3. **Software Design:** Following are the changes made to the receiver.
   - **Changing the Configuration of port:**
     Here, to enable RSS, multimode is selected, and and Ethernet RSS IPV4 is enabled.

```
.rxmode = {
                .mq_mode = ETH_MQ_RX_RSS,
                .header_split   = 0, // Header split
                .hw_ip_checksum  =  0,  //  IP  checksum
offload
                .hw_vlan_filter = 0, // VLAN filtering
                .hw_vlan_strip  = 0, // VLAN strip
                .hw_vlan_extend = 0, // Extended VLAN
                .jumbo_frame   = 0, // Jumbo frame support
                .hw_strip_crc   = 0, // CRC strip by HW
                .enable_scatter = 0, // Scattered packets
RX handler

                .max_rx_pkt_len = 9000, // Jumbo frame max
packet len
                .split_hdr_size  =  0,  //  Header  split
buffer size
        },
        .rx_adv_conf = {
                .rss_conf = {
                        .rss_key = NULL,
                        .rss_key_len = 40,
                        .rss_hf = ETH_RSS_IPV4,
                },
        }
```

- **Changing the number of RX queue:**
  Since we need three Rx queue we have to set the number of rx queue which is a constant value to 3. This is done by using the API rte_eth_dev_configure(pmd_id, No._of_rx_queue,0,&conf). pmd_id will always be zero.

```
status=rte_eth_dev_configure(pmd_id,3,0,&link_temp.conf);
```

- **Setting up the Mempool of each RX queue:**
  Firstly, the mempool is created for each port,

```
charname1[128];sprintf(name1,"MEMPOOL1%u", pmd_id);
charname2[128];sprintf(name2,"MEMPOOL2%u", pmd_id);
charname3[128];sprintf(name3,"MEMPOOL3%u", pmd_id);
```

  Then the pointer is assigned to each structure to point it out to the correct Queue.

```
struct rte_mempool * mp1;
struct rte_mempool * mp2;
struct rte_mempool * mp3;
```

  The mempool name is changed to the current new name and other parameters remain same which is then mapped to mp1,mp2 and mp3 respectively.

```
mp1 = rte_mempool_create(
      name1,//mempoolname
      mempool_params_default.pool_size,
      mempool_params_default.buffer_size,
      mempool_params_default.cache_size,
     sizeof(struct rte_pktmbuf_pool_private),
     rte_pktmbuf_pool_init, NULL,
     rte_pktmbuf_init, NULL,
     socket_id,
       0}                                    );
```

```
if(mp1 == NULL) { printf("Error, can not create mempool for dev %u \n", pmd_id); exit(1); }
```

```
mp2 = rte_mempool_create(
       name2,//mempool name
      mempool_params_default.pool_size,
      mempool_params_default.buffer_size,
      mempool_params_default.cache_size,
     sizeof(struct rte_pktmbuf_pool_private),
     rte_pktmbuf_pool_init, NULL,
     rte_pktmbuf_init, NULL,
     socket_id,
            0);

if(mp2 == NULL) { printf("Error, can not create mempool for dev %u
\n", pmd_id); exit(1); }

mp3 = rte_mempool_create(
       name3,//mempool name
   mempool_params_default.pool_size,
   mempool_params_default.buffer_size,
   mempool_params_default.cache_size,
   sizeof(struct rte_pktmbuf_pool_private),
   rte_pktmbuf_pool_init, NULL,
   rte_pktmbuf_init, NULL,
   socket_id,
          0);
if(mp3 == NULL) { printf("Error, can not create mempool for dev %u
\n", pmd_id); exit(1); }
```

- **Configuring the RX queue:**

```
status = rte_eth_rx_queue_setup(
                                  pmd_id,
                                 0,
            default_hwq_in_params.size,
            socket_id,
            &default_hwq_in_params.conf,
            mp1 //memory pool address
                                );
if (status < 0) { printf("Error, can not set up queue for dev %u \n",
pmd_id);  exit(1); }
```

```
status = rte_eth_rx_queue_setup(
                        pmd_id,
                        1,
     default_hwq_in_params.size,
     socket_id,
    &default_hwq_in_params.conf,
    mp2 //memory pool address
                        );
if (status < 0) { printf("Error, can not set up queue for dev %u \n", pmd_id);  exit(1); }

status = rte_eth_rx_queue_setup(
                                pmd_id,
                                     2,
     default_hwq_in_params.size,
     socket_id,
     &default_hwq_in_params.conf,
     mp3 //memory pool address
                        );
if (status < 0) { printf("Error, can not set up queue for dev %u \n", pmd_id);  exit(1); }
```

- **Setting up each core to work on a different RX queue:**
  For each lcore_id in the burst function queue ID will be changed respectively.

```
   if(lcore_id == 0)
   {
        printf("Hello from master core 0 !\n", lcore_id);

        while(1)
        {
uint32_t total_time_in_sec = 10;
uint64_t p_ticks = total_time_in_sec * rte_get_tsc_hz();
uint64_t p_start = rte_get_tsc_cycles();
uint32_t total_pkts = 0; //for statistics
while(rte_get_tsc_cycles() - p_start < p_ticks)
             {
n_pkts=rte_eth_rx_burst(0,0,pkts,RTE_PORT_IN_BURST_SIZE_MAX); //trying to receive packts
if(unlikely(n_pkts == 0)) {continue;} //if no packet received, then start the next try
```

```
   total_pkts += n_pkts;
   //retrieving the data from each packet
    }
   else if(lcore_id == 1)
   {
        printf("Hello from master core 1 !\n",
lcore_id);

        while(1)
        {
uint32_t total_time_in_sec = 10;
uint64_t p_ticks = total_time_in_sec * rte_get_tsc_hz();
uint64_t p_start = rte_get_tsc_cycles();
uint32_t total_pkts = 0; //for statistics
while(rte_get_tsc_cycles() - p_start < p_ticks)
            {
n_pkts=rte_eth_rx_burst(0,1,pkts,RTE_PORT_IN_BURST_SIZ
E_MAX); //trying to receive packts
if(unlikely(n_pkts == 0)) {continue;} //if no packet
received, then start the next try
total_pkts += n_pkts;


else if(lcore_id == 2)
   {
        printf("Hello from master core 2 !\n",
lcore_id);

   while(1)
        {
   uint32_t total_time_in_sec = 10;
   uint64_tp_ticks=total_time_in_sec* rte_get_tsc_hz();
   uint64_t p_start = rte_get_tsc_cycles();
   uint32_t total_pkts = 0; //for statistics
   hile(rte_get_tsc_cycles() - p_start < p_ticks)
            {
n_pkts=rte_eth_rx_burst(0,2,pkts,RTE_PORT_IN_BURST_SIZ
E_MAX);
if(unlikely(n_pkts == 0)) {continue;}
total_pkts += n_pkts;
//retrieving the data from each packet
```

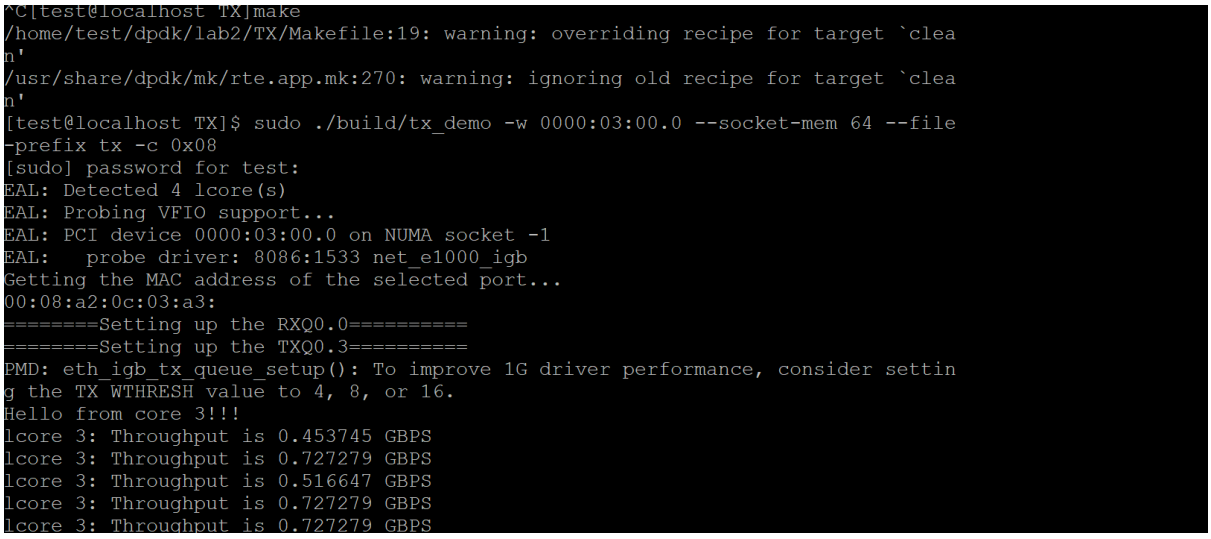
4. **Troubleshooting:**
    - At a point of time, wrong build file was getting created.
      I discussed it with TA and did the necessary changes in the make file and the issue was solved.

    - Other issue was that whenever the lcore calls the rte_eth_rx_burst(), each lcore was passing the same queue ID and hence received packets only in first core.
      After thoroughly understanding the function of rte_eth_rx_burst(), and finding out the parameters used here, different queue ID was passed so that each lcore will handle its own RX queue.

    - I got "expected identifier or '(' before '}' token" error since the main function was not closed correctly. This issue was resolved by carefully going through the code and doing the necessary changes.

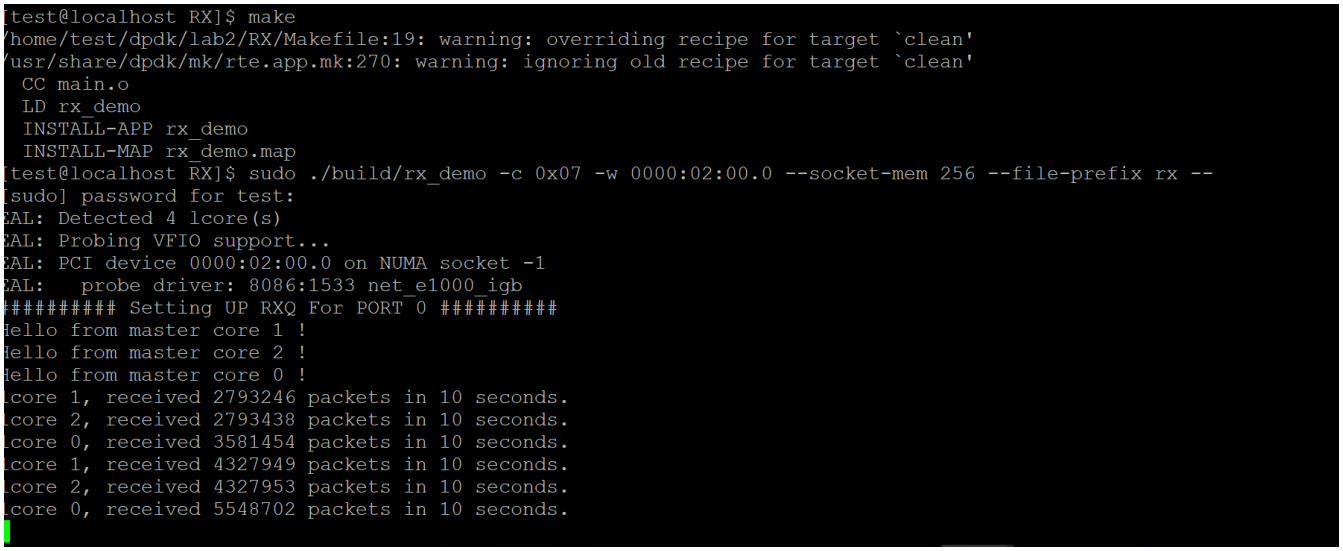
5. **Results:**
    - The following image shows the working of TX side and the command used to run the program is

      sudo ./build/tx_demo -w 0000:03:00.0 --socket-mem 64 --file-prefix tx -c 0x08



- The following image shows the working of RX side and the command used to run the program is

  sudo ./build/rx_demo -c 0x07 -w 0000:02:00.0 --socket-mem 256 --file-prefix rx –



6. Conclusion:

Hence multiple lcore and thus queue was enabled for high volume network. Used Receive Side Scaling efficiently to increase the stability of DPDK.