DPDK Developer Manual

-

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. In this manual we would describe in detail the BESS forwarding script and the DPDK layer processing of the our user-space stack. Inside the VM, mTCP would handle the TCP and IP processing and DPDK layer would handle the layer 2 processing.

2 BESS Forwarding Script

BESS is a framework to build a software switch. We need to add a script using BESS in built modules to create a datapath pipeline. The pipeline can enable packet forwarding between ports. These ports can be assigned to VMs which would allow communication between the VMs. We have provided the forward_script.bess in the bess folder. In this script we have created 2 PMDPorts (one for each VM) and one input and one output queue per-PMDPort. We have used the L2Forward() module which redirects the incoming packet to output queue. To add the forwarding rules in L2Forward() module, we have added two entries of form ¡mac,output gate¿ where mac is the VM's MAC address and, output gate are mapped to output queues of the VMs. To handle arp broadcast packet, we have added entry of form ¡broadcasting mac,unique output gate¿ where broadcasting mac is ff:ff:ff:ff:ff:ff. Whenever we receive arp broadcast packet, it will be redirected to unique output gate which in turn would be forwarded to the Replicate() module and then we duplicate these arp packet using this module and forward it to all the VMs.

3 DPDK Layer of the user-space network stack

DPDK layer provides ethernet layer processing and handles the packet distribution to the multiple cores of the VMs. The DPDK layer receives packet from the single queue NIC and distributes it to all the core using soft RSS. Also, it gathers packets from all the cores, adds ethernet header and sends them to the NIC. The DPDK layer logic is defined in dpdk_api.cpp and dpdk_api.h.

Depending on the no of cores provided to VMs, we have two designs for the DPDK layer.

Case-I) Single Core VM

The packet flow in case of single core VM with single queue vNIC is as follows:
For reception:

- The packet is read from the vNIC whenever mTCP requests for the packet *get_rx_count()*. In this function, packets are read from the vNIC, filtered based on VM's IP and, the number of packets read is given to the mTCP layer.
- To access these packets, mTCP uses *get_buffer_rx()* and calls this function as many times as the number of packets read.

For transmission:

- mTCP acquires an empty packet buffer from DPDK layer via *get_buffer_tx()* and fills the packet (with IP, TCP headers and payload) into it.
- For transmission to the vNIC, mTCP uses *addBufferToRing()* DPDK API function. This function adds ethernet header to the packets and sends it directly to the vNIC.

Case-II) Multi Core VM

In case of multi-core VM, packets received from the single queue vNIC need to be distributed to all the cores and packets coming from mtCP thread running on multiple core need to be send to vNIC. This distribution and transmission is done in the DPDK layer. In the DPDK layer there can be a single thread handling this distribution on reception and transmission (*1 KB option* KB means Kernel Bypass). There can be another option with two separate threads (*2 KB option*) handling reception (with packet distribution) and transmission. These threads would run on two different cores. In case of 1 KB option, the last core provided to the application would handle the reception and transmission (RxTx_Thread). In case of 2 KB the last core would handle the reception (Rx_Thread) and the last but one core would handle the transmission (Tx_Thread). The packet flow is as follows:
For reception:

- In both the cases i.e 1 KB or 2 KB option, *rx_main_loop()* function will read packets from the vNIC, filter based on VM's IP, then redirect these packets to the per-core rx_queue ring based on the soft RSS.

- Per-core mTCP thread calls *get_rx_count()* function to get the number of unprocessed received packets.
- To access these packets, the per-core mTCP thread calls *get_buffer_rx()* which returns packets from the respective per-core rx_queue ring. This function is called as many times as the number return by the *get_rx_count()* function which was previously called by the mTCP thread.

For Transmission:

- mTCP acquires an empty packet buffer from DPDK layer via *get_buffer_tx()* and fills the packet (with IP, TCP headers and payload) into it.
- To buffer these packet for transmission, mTCP uses *addBufferToRing()* DPDK API function. This function adds ethernet header to the packets and add this packet to its per-core tx_queue ring.
- In case of 1 KB option, *rx_main_loop()* is the only function which handles both Rx as well as Tx functionality. So, this function reads packets from the per-core tx_queue ring in round-robin fashion and sends them to the vNIC. In case of 2 KB option, *tx_main_loop()* function handles this Tx thread functionality on a different core.

dpdk_init is called in the application code for DPDK ports and core initilaization.