# Dsitributed System Lab2 Report

516030910293 姚子航

## Part1

Q1: What’s the purpose of using hugepage?

A: Using huge page reduce the number of page table entry because now an entry can map 4MB memory.Then the hit rate of TLB may increase significantly, which accelerates memory access.

Q2: Take examples/helloworld as an example, describe the execution flow of DPDK programs?

1.Initialize basic running environment using function "rte_eal_init"
2.Initialize multi-core running using function "rte_eal_remote_launch"
Then lcore_hello will be called on every slave logical core.
3.Call lcore_hello on master logical core
4.Wait all logical cores to end up executing using function "rte_eal_mp_wait_lcore"

Q3: Read the codes of examples/skeleton, describe DPDK APIs related to sending and receiving packets.

A:
{
Create mempool API:struct rte_mempool *rte_pktmbuf_pool_create(const char *name, unsigned n, unsigned cache_size, uint16_t priv_size, uint16_t data_room_size, int socket_id)

This function create a memory pool for storing rte_mbuf.
}

{
Receive API: static inline uint16_t rte_eth_rx_burst(uint8_t port_id, uint16_t queue_id, struct rte_mbuf **rx_pkts, const uint16_t nb_pkts)

Send API:static inline uint16_t rte_eth_tx_burst(uint8_t port_id, uint16_t queue_id, struct rte_mbuf **tx_pkts, uint16_t nb_pkts)

1st parameter: port
2nd parameter:queue identifier
3rd parameter:packet buffer
4th parameter:the number of packets we hope to send or receive

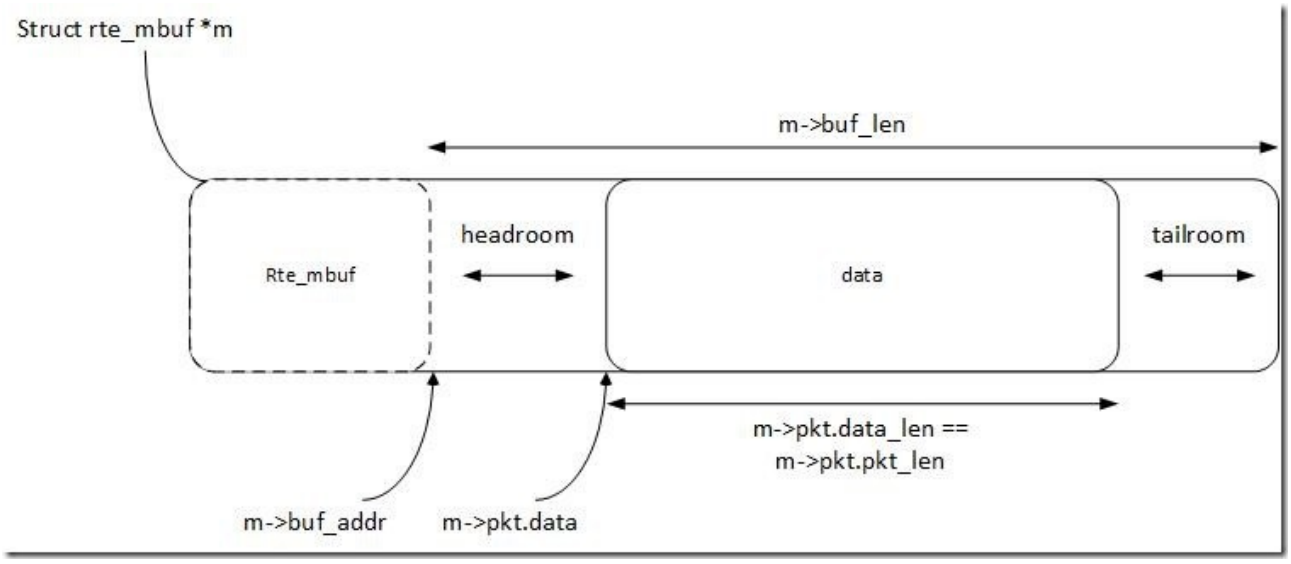
return value: the number packets we actually send or receive.
}

{
Allocate API:struct rte_mbuf *rte_pktmbuf_alloc(struct rte_mempool *mp);

This function apply a rte_mbuf from memory pool.
}

Q4: Describe the data structure of ‘rte_mbuf’.

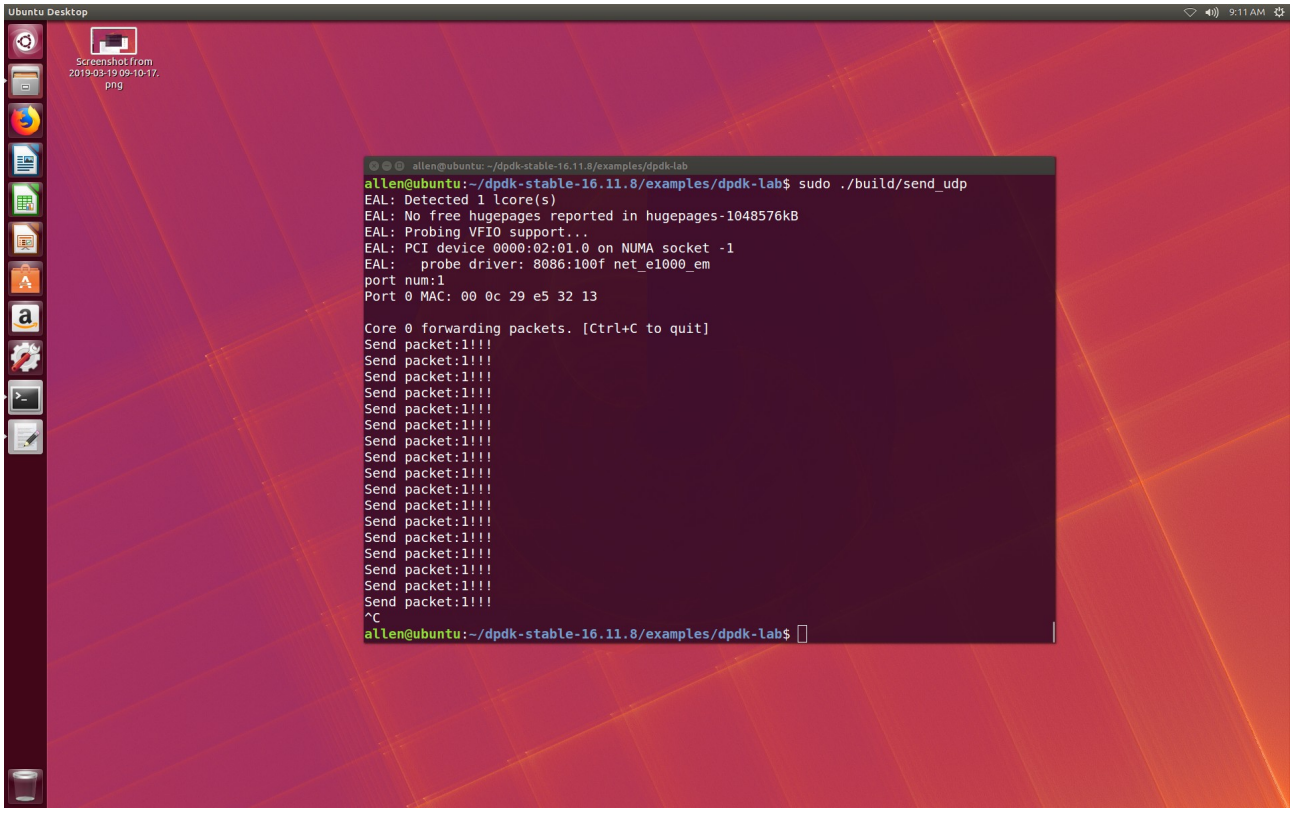
A:"rte_mbuf" structure describes a packet mbuf, which mainly contains a data buffer. When we use rte_pktmbuf_alloc function to allocate a new rte_buf, dpdk will reserve some headroom. If we want to fill data  in it, we could use rte_pktmbuf_prepend or rte_pktmbuf_append to increase the size of headroom or tailroom.The following image gives us a more intuitive description.

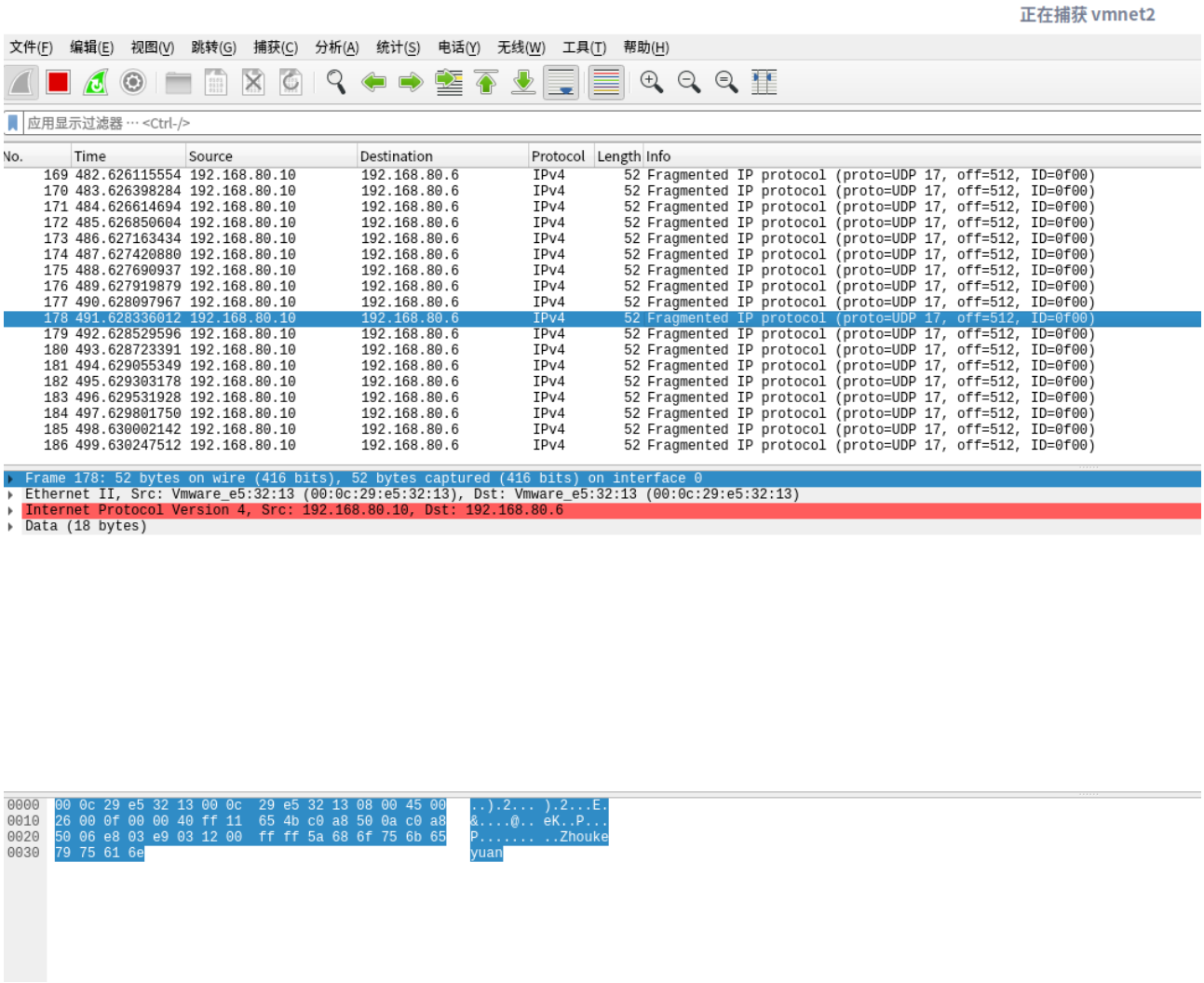



## Part 2

Action:
1.Install wireshark in host OS
2.Make and start program “send_udp”
3.Open wireshark and select to catch packets on vmnet2(the virtual NIC I just add)
4.Detect many packets sent by dpdk.

Screenshot:




正在捕获 vmnet2
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
应用显示过滤器 … <Ctrl-/>
No. Time Source Destination Protocol Length Info
169 482.626115554 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
170 483.626398284 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
171 484.626614694 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
172 485.626850604 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
173 486.627163434 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
174 487.627420880 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
175 488.627690937 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
176 489.627919879 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
177 490.628097967 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
178 491.628336012 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
179 492.628529596 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
180 493.628723391 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
181 494.629055349 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
182 495.629303178 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
183 496.629531928 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
184 497.629801750 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
185 498.630002142 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
186 499.630247512 192.168.80.10 192.168.80.6 IPv4 52 Fragmented IP protocol (proto=UDP 17, off=512, ID=0f00)
Frame 178: 52 bytes on wire (416 bits), 52 bytes captured (416 bits) on interface 0
Ethernet II, Src: Vmware_e5:32:13 (00:0c:29:e5:32:13), Dst: Vmware_e5:32:13 (00:0c:29:e5:32:13)
Internet Protocol Version 4, Src: 192.168.80.10, Dst: 192.168.80.6
Data (18 bytes)
0000 00 0c 29 e5 32 13 00 0c 29 e5 32 13 08 00 45 00 ..).2... ).2...E.
0010 26 00 0f 00 00 40 ff 11 65 4b c0 a8 50 0a c0 a8 &....@.. eK..P...
0020 50 06 e8 03 e9 03 12 00 ff ff 5a 68 6f 75 6b 65 P..... ..Zhouke
0030 79 75 61 6e yuan