Exercise for Lecture
Software Defined Networking

Prof. Dr. David Hausheer, Julius Rückert
Christian Koch, Jeremias Blendin, Leonhard Nobach, Matthias Wichtlhuber

TECHNISCHE UNIVERSITÄT DARMSTADT

Winter Term 2016/17
Exercise No. 6
Published: 10.01.2017
Submission exclusively via Moodle, Deadline: 24.01.2017
Contact: Please use the Moodle forum to post questions and remarks on the exercise.
Web: `http://www.ps.tu-darmstadt.de/teaching/ws1617/sdn/`
Submission: `https://moodle.tu-darmstadt.de/course/view.php?id=8385`

Surname (Nachname):	
First name (Vorname):	
ID# (Matrikelnummer):	

Problem 6.1 - DPDK Basics

For the following tasks, we refer to the DPDK documentation, especially

- https://en.wikipedia.org/wiki/New_API (Not part of DPDK, but of the Linux kernel)
- http://dpdk.org/doc/guides-2.1/prog_guide/poll_mode_drv.html
- http://dpdk.org/doc/guides/prog_guide/mbuf_lib.html
- http://dpdk.org/doc/api/rte__byteorder_8h.html

You can answer in free text to solve this problem, writing code is optional / not necessary.

a) Interrupts vs. Poll-Mode Drivers

The "New API" (not part of DPDK) is a Linux-kernel effort to speed up network processing. It achieves this by finding a trade-off between interrupts and polling.

I What are the advantages of polling, and the advantages of interrupts?

II Define one network load situation in which you would prefer poll-mode-drivers, and one in which you would prefer interrupts.

b) Mempools / MBuf Structures

Incoming packets are stored in `mbuf` structures. You want to implement a (non-encrypting) tunnel endpoint, which must append an additional header between the Ethernet and the IP header before forwarding the packet. Assume a maximum-sized packet (1500 bytes). **How do you efficiently solve this in DPDK without shifting the entire payload?**

c) DPDK and Endianness

Explain why the functions of `rte_byteorder.h` are useful if you want to manipulate or do calculations on IP packet headers on an x86-based system.

d) NFV Performance

Assume a packet of 1500 bytes.

- NF 1 checks if the payload contains a specific signature of at least 16 bytes anywhere in the payload and, in case, drops the packet.
- NF 2 encapsulates a packet in a VLAN if it originates from a list of 20.000 exact, well-known malicious IP addresses.
- NF 3 performs generic routing encapsulation (GRE) of all input traffic.

Estimate the relative performance of the three NFs you can achieve when implementing it in DPDK, based on the expected processing complexity. Order them by your estimation, and provide the reasons for your answer. ***(Example: NFx should achieve the highest performance, because ..., NFy achieves medium performance, because ...)***

Problem 6.2 - DPDK Code Study

```
1
2  \\...packet receive code omitted...
3
4  const struct ether_addr* intAddr2set =
       {.addr_bytes={0x52,0x54,0x00,0x00,0x00,0x01}}
5  uint32_t my_maskv4 = IPv4(255,255,255,0);
6  uint32_t my_ipv4 = IPv4(192,168,0,1);
7
8  struct arp_hdr* arp_hdr = rte_pktmbuf_mtod_offset(packet, struct arp_hdr*,
       sizeof(struct ether_hdr) + vlan_offset);
9
10 if (__bswap_16(arp_hdr->arp_op) == 1) {
11
12   struct arp_ipv4* arpdata = &arp_hdr->arp_data;
13
14   uint32_t swapped_tip = __bswap_32(arpdata->arp_tip);
15
16   if (swapped_tip == my_ipv4) {
17
18     if ( /* TODO */) {
19       RTE_LOG(INFO, USER1, "   Sent from a wrong subnet...\n");
20       return 0;
21     }
22
23     memcpy(&l2hdr->d_addr, &arpdata->arp_sha, sizeof (struct ether_addr));
24     memcpy(&l2hdr->s_addr, my_mac, sizeof(struct ether_addr));
25     memcpy(&arpdata->arp_tha, &arpdata->arp_sha, sizeof(struct
           ether_addr));
26     memcpy(&arpdata->arp_sha, my_mac, sizeof(struct ether_addr));
27     arpdata->arp_tip = arpdata->arp_sip;
28     arpdata->arp_sip = __bswap_32(my_ipv4);
29     arp_hdr->arp_op = __bswap_16(2);
30
31     struct rte_mbuf* pkt2send_buf[1];
32     pkt2send_buf[0] = packet;
33
34     int retrn = rte_eth_tx_burst(IF_IDENTIFIER, 0, pkt2send_buf, 1);
35     if (retrn == 1) {
36       printf("    Successful.\n");
37       return 1;
38     } else {
39       printf("    NOT successful.\n");
40       return -1;
41     }
42   }
43 }
```

I This code works on an incoming ARP packet. What is this code supposed to do with it?

II Add the missing `if` statement checking whether the ARP was sent from a wrong subnet.