Exercise for Lecture Software Defined Networking

Prof. Dr. David Hausheer
Julius Rückert, Leonhard Nobach

TECHNISCHE UNIVERSITÄT DARMSTADT

Winter Term 2015/16
Exercise No. 6
Published at: 12.01.2016
Submission exclusively via Moodle, Deadline: 15.01.2016 (Problem 6.1), 19.01.2016 (Rest)
Contact: [rueckert|lnobach]@ps.tu-darmstadt.de
Web: http://www.ps.tu-darmstadt.de/teaching/ws1516/sdn/
Submission: https://moodle.tu-darmstadt.de/enrol/index.php?id=6349
– Example Solution –

Problem 6.1 - Submit SSH Credentials

In the next exercise (Exercise 7), we will immediately start with a lab exercise in a DPDK virtual machine for which you need an SSH and SCP/SFTP client. For giving you access, we need an SSH public key from everyone of you (Format ssh-rsa, ssh-dsa, or ssh-ed25519).

Most of you will have dealt with SSH before and have some experience with SSH pubkeys. For those of you who don't, please refer to e.g. *https://www.rackspace.com/knowledge_center/article/generating-rsa-keys-with-ssh-puttygen* (Windows) or *https://help.github.com/articles/generating-ssh-keys/* (Mac/Linux).

For Exercise 7 ff., **please form a group of 4 to 6 members** (we do not have more VM capacity). Please submit your **individual** SSH public key individually in a .txt file via Moodle, following the names of your group members:

```
ssh-rsa [a lot of characters] user@host
Other group member 1
Other group member 2
Other group member 3
```

Please pay respect to the format of the key, this is the OpenSSH representation (use Conversions->Export OpenSSH) under PuTTY). Pay attention that no characters are mistakenly added to the key string while copy/pasting. If you just want to work on the computer of another group member, you may omit your own SSH pubkey in your individual submission. Keep your *private* key secret like an individual password, do not share it.

Note: Please provide these credentials until Friday, 15th. This way, we can grant you access to your group's VM on next Tuesday at latest. We will open a special Moodle submission for them. By providing the SSH key to us, you agree that you will use the VM **only for solving the lab tasks**, and for nothing else. Access to the VMs and operations on them are monitored.

Problem 6.2 - 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, 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?

Solution: PMD: Current jobs (i.e. packet modifications) are not interrupted upon a packet reception, and can be completed before the next job (packet) arrives. Interrupts: A job is only - and immediately - started if a packet arrives

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

Solution: PMD: Constant high throughput. Interrupts: Infrequent, few packets with stricter delay requirements

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 original header and the TCP/UDP header, before forwarding the packet. Assume a maximum-sized packet (1500 bytes). **How do you efficiently solve this in DPDK without operating on the entire payload?**

Solution: We refer to Figure 6.2. Here, a headroom can be defined in the mbuf structure. Using the `prepend` function, the original header can be moved (which is much smaller than the payload), so that we can add the VPN header in between.

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.

Solution: While the x86 architecture is little-endian(LE), number representations in packet headers of the Internet protocol suite are big-endian(BE). Therefore, we have to convert the numbers with the given functions here.

Problem 6.3 - 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 (__bswap_32(arpdata->arp_sip)&my_maskv4 != my_ipv4&my_maskv4) {
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 = my_ipv4;
29     arp_hdr->arp_op = 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?

Solution: The code is supposed to check if the ARP request is for us, manipulate the incoming ARP packet so that is becomes a response, and send the response back to the sender.

II Name at least one reason why this script does not exactly do what is expected (not a syntax error or an error caused by undefined variables).

Solution: The arp_sip and arp_op have not been byteswapped, they will have a wrong value in the packet sent out.