实验二 利用DPDK构造并发送数据包

1. **实验目的**
2. 学会编写和测试DPDK数据包发送程序。
3. 理解DPDK实现高性能数据包处理的原理。
4. **实验内容**
5. 编写一个DPDK程序，实现如下功能：直接构造一个数据包(内容任意)，要求数据包的协议头符合UDP/IP/Ethernet协议规范，并将其发送出去。该程序基于skeleton程序修改得到，附录中给出一个供参考的编程框架。
6. 使用tcpdump命令，抓取数据包，观察各协议字段值，确认实际输出与程序意图是否一致。

过程举例：

1. //在命令行窗口运行如下命令
2. //抓取经过指定网卡含有特定端口号的数据包
3. sudo tcpdump -vv -A -e -i <网卡名称> port <端口号>
4. //然后在另一窗口运行DPDK程序
6. //注意：tcpdump只能在内核驱动网卡上抓取数据包
7. //tcpdump的更多用法可参阅相关资料

**三、注意事项**

1. 由于构造DPDK UDP包时需要构造以太网首部，IP首部以及UDP首部，可能会比较复杂，要注意各字节对应的协议内容以及主机字节顺序和网络字节顺序的转换。

2. 可以先使用tcpdump命令抓取UDP包，观察收到的UDP包的首部和payload结构，合理分析自己构造的UDP数据包是否符合协议规范（源目的MAC地址，源目的IP地址，源目的端口，以及协议类型字段是否按照要求构造）。

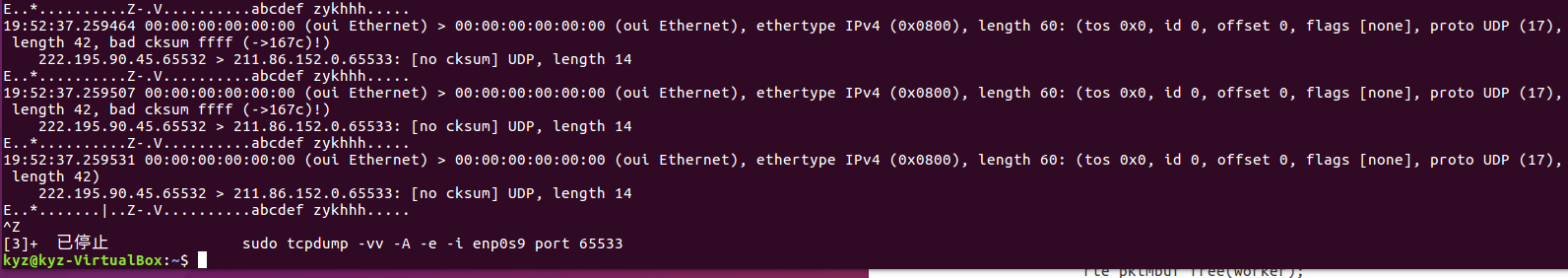
3. 在DPDK程序运行时，使用tcpdump命令过滤端口和协议类型后如果收不到包，可以适当放宽过滤条件，在观察到自己程序中构造的包后再分析是哪个字段的构造出了问题。最后应该能够保证抓到的包的各部分字段与预期一致，且tcpdump命令能够在过滤了端口以及协议类型的情况下接收到包。

4. 由于运行DPDK程序的网卡应该挂载igb\_uio的驱动，不能够使用tcpdump命令抓包。所以推荐的配置为使用一个虚拟机，其中一张网卡配置为NAT模式，用于上网。再新开辟出两张网卡，设置为桥接网卡，并指定为同一个物理网卡。在做实验时，一张网卡挂载DPDK的驱动运行DPDK的程序，另一张网卡挂载内核驱动并开启混杂模式，使用tcpdump命令用于抓包。（进行虚拟机网卡设置时应该将虚拟机关机）。

关于虚拟机网卡的NAT、内部网路、桥接网卡模式：https://www.jb51.net/article/243423.htm

5. 增加新网卡后需要重新安装DPDK，编译新的igb\_uio.ko。总之如果出现莫名其妙的挂载错误，可以尝试重新编译一次DPDK，大概率可以解决问题。

6. 进行UDP包构造时，应该先使用rte\_pktmbuf\_append函数在rte\_mbuf后申请内存空间，然后再进行数据填入，具体使用方法请参考手册。



1.构造数据包发送时，不管目的ip、目的mac地址是多少，挂载内核驱动的那张网卡都能收到消息。但是如果用tcpdump监听时，tcpdump的过滤规则port必须和发送的数据包的udp目的端口号一致，否则监听不到数据包。

2.更奇怪的是，实验室的两台电脑，一个电脑发送，另一个电脑监听，竟然能接收到数据包。

3.而且不知道什么原因，虚拟机上运行dpdk程序发送数据包，运行一段时间宿主机上网络会断掉。

rte\_eth\_tx\_burst(port\_id, queue\_id, struct rte\_mbuf \*\*rx\_pkts, nb\_pkts)这个burst函数是二层转发？为什么能向本机以外的电脑发送？port\_id不是都是自己电脑网卡的portid吗？

1. **回答问题**

官方文档：[DPDK documentation](http://doc.dpdk.org/guides-18.11/), [DPDK API](http://doc.dpdk.org/api-18.11/)

注意：不同版本DPDK对应的文档内容也不完全一样，应对应查看

1. UDP协议头的cksum是必须的吗？不是的话应当如何设置？
2. DPDK为IP协议头的cksum计算提供了专用函数，请在相关文档中找到它，并应用在程序中，调用该函数的注意事项是什么？
3. DPDK实现高性能收发包的原理是什么？
4. **思考题**

参照本实验，一个接收数据包并处理其中数据的DPDK程序应该如何实现？

1. **进展报告**

本实验的完成情况(需要附tcpdump抓到构造的udp包的截图)以及对以上3个问题的回答作为第二阶段的进展报告（进展报告中可以附关键代码），于 10 月 25日23:59之前上传到ftp，ip：202.38.79.85，于10月28日进行课堂讨论。

进展报告中需注明小组成员（组长排在第一个），以及每位成员对该报告的贡献比例。上课前各小组做好PPT，每个小组派一位同学上台主讲，其余同学可以补充。

提交文件命名及格式：进展报告：第二阶段\_组长姓名.pdf，代码：（随意命名）.c，PPT仅做上课展示用，不做提交要求。

实验过程中遇到任何问题，请及时联系助教或老师，助教邮箱：

（blazarx@mail.ustc.edu.cn）

1. **附录**

首先，可以在example目录下新建一个目录用于本实验，可以看到，example/skeleton目录中有Makefile和meson.build这两个文件，各拷贝一份到新建的目录中，将其中的文件名修改为你的主程序文件名。

注意：

* 1. 主机字节顺序和网络字节顺序的转换
  2. 某些字段值可使用宏定义，如IPVERSION
  3. 可以使用的常用函数如inet\_addr()
  4. 各协议头对应的结构体有不同的版本，既可以使用经典版本(如[Ethernut](http://www.ethernut.de/api/annotated.html))，也可以使用DPDK自带的版本([DPDK\_Data\_Structure](http://doc.dpdk.org/api-18.11/annotated.html))，结构体中包含的数据项可以参阅相关文档。

以下程序代码基于skeleton修改得来，主要是增加了一个build\_udp\_packet函数，并需要在其中补充相关代码。

1. /\* SPDX-License-Identifier: BSD-3-Clause
2. \* Copyright(c) 2010-2015 Intel Corporation
3. \*/
5. #include <stdint.h>
6. #include <unistd.h>
7. #include <stdbool.h>
8. #include <inttypes.h>
9. #include <rte\_eal.h>
10. #include <rte\_ethdev.h>
11. #include <rte\_cycles.h>
12. #include <rte\_lcore.h>
13. #include <rte\_mbuf.h>
14. #include <rte\_ip.h>
16. #include <pcap/pcap.h>
17. #include <netinet/ip.h>
18. #include <netinet/in.h>
19. #include <rte\_ether.h>
20. #include <rte\_udp.h>
21. #include <arpa/inet.h>
23. #define RX\_RING\_SIZE 1024
24. #define TX\_RING\_SIZE 1024
26. #define NUM\_MBUFS 8191
27. #define MBUF\_CACHE\_SIZE 250
28. #define BURST\_SIZE 32
30. **struct** rte\_mempool \*mbuf\_pool;
32. **static** **const** **struct** rte\_eth\_conf port\_conf\_default = {
33. .rxmode = {
34. .max\_rx\_pkt\_len = ETHER\_MAX\_LEN,
35. },
36. };
38. **static** **inline** **int**
39. port\_init(uint16\_t port, **struct** rte\_mempool \*mbuf\_pool)
40. {
41. **struct** rte\_eth\_conf port\_conf = port\_conf\_default;
42. **const** uint16\_t rx\_rings = 1, tx\_rings = 1;
43. uint16\_t nb\_rxd = RX\_RING\_SIZE;
44. uint16\_t nb\_txd = TX\_RING\_SIZE;
45. **int** retval;
46. uint16\_t q;
47. **struct** rte\_eth\_dev\_info dev\_info;
48. **struct** rte\_eth\_txconf txconf;
50. **if** (!rte\_eth\_dev\_is\_valid\_port(port))
51. **return** -1;
53. rte\_eth\_dev\_info\_get(port, &dev\_info);
54. **if** (dev\_info.tx\_offload\_capa & DEV\_TX\_OFFLOAD\_MBUF\_FAST\_FREE)
55. port\_conf.txmode.offloads |=
56. DEV\_TX\_OFFLOAD\_MBUF\_FAST\_FREE;
58. /\* Configure the Ethernet device. \*/
59. retval = rte\_eth\_dev\_configure(port, rx\_rings, tx\_rings, &port\_conf);
60. **if** (retval != 0)
61. **return** retval;
63. retval = rte\_eth\_dev\_adjust\_nb\_rx\_tx\_desc(port, &nb\_rxd, &nb\_txd);
64. **if** (retval != 0)
65. **return** retval;
67. /\* Allocate and set up 1 RX queue per Ethernet port. \*/
68. **for** (q = 0; q < rx\_rings; q++) {
69. retval = rte\_eth\_rx\_queue\_setup(port, q, nb\_rxd,
70. rte\_eth\_dev\_socket\_id(port), NULL, mbuf\_pool);
71. **if** (retval < 0)
72. **return** retval;
73. }
75. txconf = dev\_info.default\_txconf;
76. txconf.offloads = port\_conf.txmode.offloads;
77. /\* Allocate and set up 1 TX queue per Ethernet port. \*/
78. **for** (q = 0; q < tx\_rings; q++) {
79. retval = rte\_eth\_tx\_queue\_setup(port, q, nb\_txd,
80. rte\_eth\_dev\_socket\_id(port), &txconf);
81. **if** (retval < 0)
82. **return** retval;
83. }
85. /\* Start the Ethernet port. \*/
86. retval = rte\_eth\_dev\_start(port);
87. **if** (retval < 0)
88. **return** retval;
90. /\* Display the port MAC address. \*/
91. **struct** ether\_addr addr;
92. rte\_eth\_macaddr\_get(port, &addr);
93. printf("Port %u MAC: %02" PRIx8 " %02" PRIx8 " %02" PRIx8
94. " %02" PRIx8 " %02" PRIx8 " %02" PRIx8 "\n",
95. port,
96. addr.addr\_bytes[0], addr.addr\_bytes[1],
97. addr.addr\_bytes[2], addr.addr\_bytes[3],
98. addr.addr\_bytes[4], addr.addr\_bytes[5]);
100. /\* Enable RX in promiscuous mode for the Ethernet device. \*/
101. rte\_eth\_promiscuous\_enable(port);
103. **return** 0;
104. }
106. **static** **void**
107. build\_udp\_packet(**struct** rte\_mbuf\* worker)
108. {
109. //DPDK
110. /\* add your code here \*/
111. **char** \* payload = (**char**\*)rte\_pktmbuf\_append(worker, 14);
113. \*(payload +  0) = 'h';
114. \*(payload +  1) = 'e';
115. \*(payload +  2) = 'l';
116. \*(payload +  3) = 'l';
117. \*(payload +  4) = 'o';
118. \*(payload +  5) = ',';
119. \*(payload +  6) = ' ';
120. \*(payload +  7) = 'w';
121. \*(payload +  8) = 'o';
122. \*(payload +  9) = 'r';
123. \*(payload + 10) = 'l';
124. \*(payload + 11) = 'd';
125. \*(payload + 12) = '.';
126. \*(payload + 13) = '\0';

129. }
131. /\*
132. \* The lcore main. This is the main thread that does the work, construct a
133. \* packet and deliver it.
134. \*/
135. **static** \_\_attribute\_\_((**noreturn**)) **void**
136. lcore\_main(**void**)
137. {
138. uint16\_t port;
140. /\*
141. \* Check that the port is on the same NUMA node as the polling thread
142. \* for best performance.
143. \*/
144. RTE\_ETH\_FOREACH\_DEV(port)
145. **if** (rte\_eth\_dev\_socket\_id(port) > 0 &&
146. rte\_eth\_dev\_socket\_id(port) !=
147. (**int**)rte\_socket\_id())
148. printf("WARNING, port %u is on remote NUMA node to "
149. "polling thread.\n\tPerformance will "
150. "not be optimal.\n", port);
152. printf("\nCore %u is running. [Ctrl+C to quit]\n",
153. rte\_lcore\_id());
155. /\* Run until the application is quit or killed. \*/
156. **for** (;;) {
157. **int** ret;
158. **struct** rte\_mbuf \*worker;
160. **do** {
161. worker = rte\_pktmbuf\_alloc(mbuf\_pool);
162. } **while** (unlikely(worker == NULL));
163. worker->nb\_segs = 1;
164. worker->next = NULL;
165. build\_udp\_packet(worker);
167. ret = rte\_eth\_tx\_burst(0, 0, &worker, 1);
169. /\* Free unsent packet. \*/
170. **if** (unlikely(ret < 1)) {
171. rte\_pktmbuf\_free(worker);
172. }
173. }
174. }
176. /\*
177. \* The main function, which does initialization and calls the per-lcore
178. \* functions.
179. \*/
180. **int**
181. main(**int** argc, **char** \*argv[])
182. {
184. unsigned nb\_ports;
185. uint16\_t portid;
187. /\* Initialize the Environment Abstraction Layer (EAL). \*/
188. **int** ret = rte\_eal\_init(argc, argv);
189. **if** (ret < 0)
190. rte\_exit(EXIT\_FAILURE, "Error with EAL initialization\n");
192. argc -= ret;
193. argv += ret;
195. nb\_ports = rte\_eth\_dev\_count\_avail();
197. /\* Creates a new mempool in memory to hold the mbufs. \*/
198. mbuf\_pool = rte\_pktmbuf\_pool\_create("MBUF\_POOL", NUM\_MBUFS \* nb\_ports,
199. MBUF\_CACHE\_SIZE, 0, RTE\_MBUF\_DEFAULT\_BUF\_SIZE, rte\_socket\_id());
201. **if** (mbuf\_pool == NULL)
202. rte\_exit(EXIT\_FAILURE, "Cannot create mbuf pool\n");
204. /\* Initialize all ports. \*/
205. RTE\_ETH\_FOREACH\_DEV(portid)
206. **if** (port\_init(portid, mbuf\_pool) != 0)
207. rte\_exit(EXIT\_FAILURE, "Cannot init port %"PRIu16 "\n",
208. portid);
210. **if** (rte\_lcore\_count() > 1)
211. printf("\nWARNING: Too many lcores enabled. Only 1 used.\n");
213. /\* Call lcore\_main on the master core only. \*/
214. lcore\_main();
216. **return** 0;
217. }