

# 课程大作业5报告

---

胡康喆 519021910824

## 一、介绍

---

在这次大作业中，我选择了DPDK performance test。我在KVM上创建了两个虚拟机，并在创建的虚拟机上安装编译了DPDK环境。然后我在VM2上运行了l2fwd，在VM1上运行了pktgen-dpdk，并评估了DPDK的二层转发的性能。

## 二、背景

---

### 2.1 DPDK介绍

DPDK全称Intel Data Plane Development Kit，是intel提供的数据平面开发工具集，为Intel architecture (IA) 处理器架构下用户空间高效的数据包处理提供库函数和驱动的支持。通俗地说，就是一个用来进行包数据处理加速的软件库。

相对传统的基于内核的网络数据处理，DPDK 对从内核层到用户层的网络数据流程进行了重大突破：

- UIO（用户空间的 I/O 技术）的加持。DPDK 能够绕过内核协议栈，本质上是得益于 UIO 技术，通过 UIO 能够拦截中断，并重设中断回调行为，从而绕过内核协议栈后续的处理流程。
- 内存池技术。DPDK 在用户空间实现了一套精巧的内存池技术，内核空间和用户空间的内存交互不进行拷贝，只做控制权转移。这样，当收发数据包时，就减少了内存拷贝的开销。
- 大页内存管理。DPDK 实现了一组大页内存分配、使用和释放的 API，上层应用可以很方便使用 API 申请使用大页内存，同时也兼容普通的内存申请。
- 无锁环形队列。DPDK 基于 Linux 内核的无锁环形缓冲 kfifo 实现了自己的一套无锁机制。支持单生产者入列/单消费者出列和多生产者入列/多消费者出列操作，在数据传输的时候，降低性能的同时还能保证数据的同步。
- poll-mode 网卡驱动。DPDK 网卡驱动完全抛弃中断模式，基于轮询方式收包，避免了中断开销。
- NUMA。DPDK 内存分配上通过 proc 提供的内存信息，使 CPU 核心尽量使用靠近其所在节点的内存，避免了跨 NUMA 节点远程访问内存的性能问题。
- CPU 亲和性。DPDK 利用 CPU 的亲和性将一个线程或多个线程绑定到一个或多个 CPU 上，这样在线程执行过程中，就不会被随意调度，一方面减少了线程间

的频繁切换带来的开销，另一方面避免了 CPU 缓存的局部失效性，增加了 CPU 缓存的命中率。

- 多核调度框架。DPDK 基于多核架构，一般会有主从核之分，主核负责完成各个模块的初始化，从核负责具体的业务处理。

除了上述之外，DPDK 还有很多的技术突破，可以用下面这张图来概之。

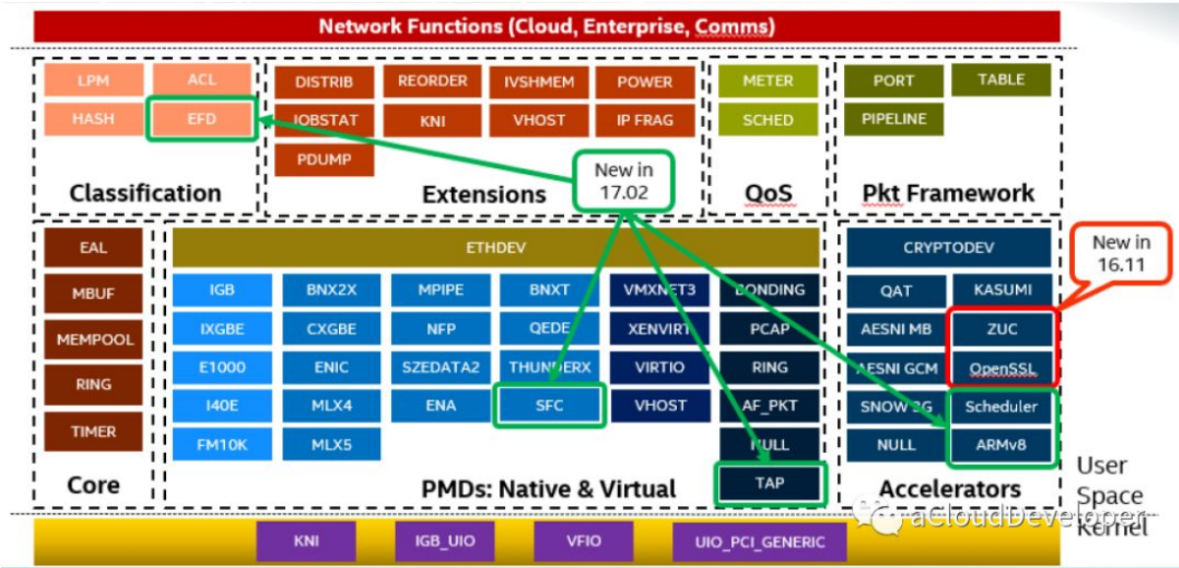


图1 DPDK的技术突破

## 2.2 DPDK架构与核心组件

DPDK的总体结构如下：

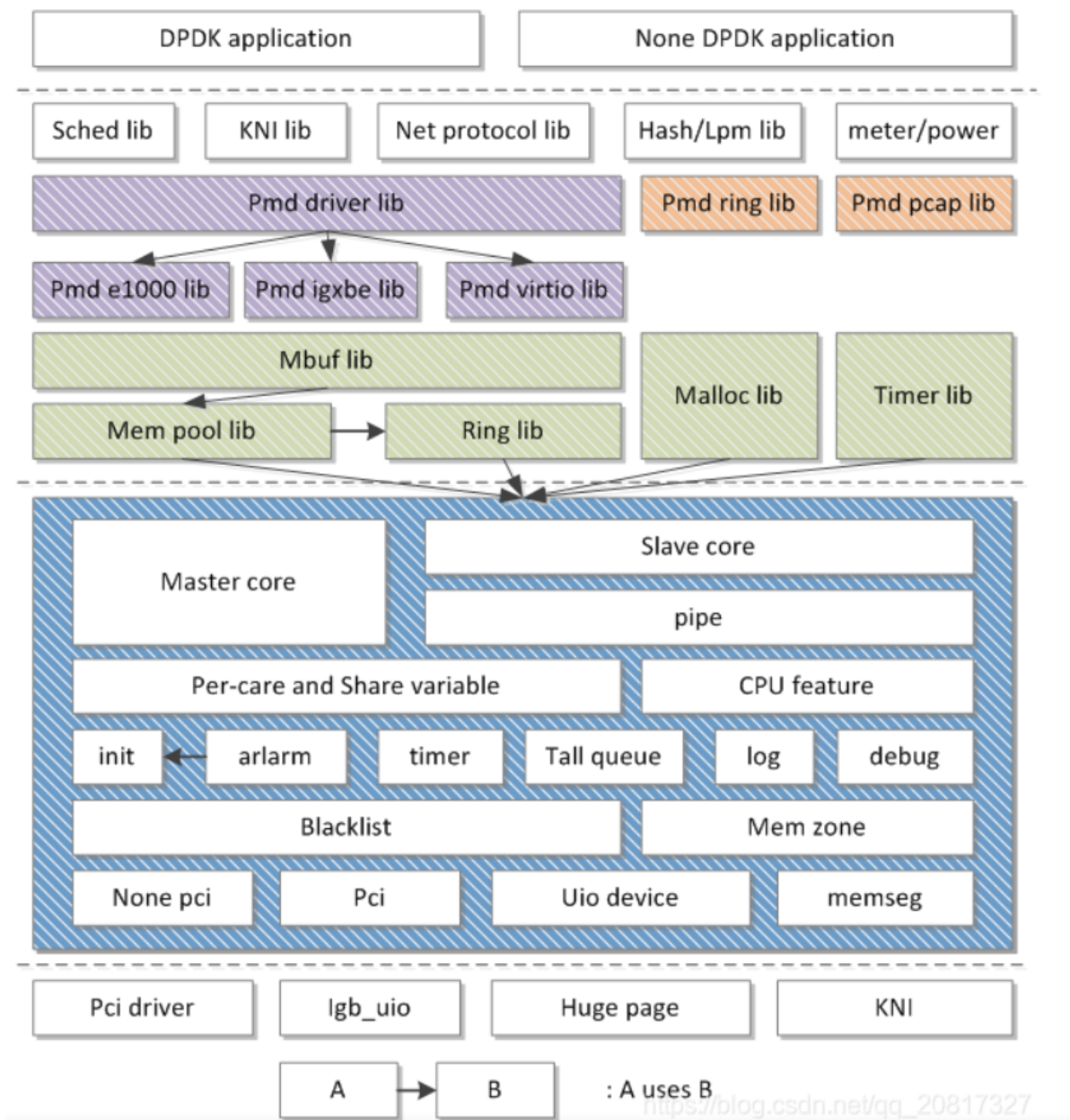


图2 DPDK总体架构

DPDK主要有六个核心组件.

1. 环境抽象层 (EAL) : 为DPDK其他组件和应用程序提供一个屏蔽具体平台特性的统一接口, 环境抽象层提供的功能主要有: DPDK加载和启动; 支持多核和多线程执行类型; CPU核亲和性处理; 原子操作和锁操作接口; 时钟参考; PCI总线访问接口; 跟踪和调试接口; CPU特性采集接口; 中断和告警接口等。
2. 堆内存管理组件 (Malloc lib) : 堆内存管理组件为应用程序提供从大页内存分配对内存的接口。当需要分配大量内存小块时, 使用这些接口可以减少TLB缺页。
3. 环缓冲区管理组件 (Ring lib) : 环缓冲区管理组件为应用程序和其他组件提供一个无锁的多生产者多消费者FIFO队列API: Ring。Ring是借鉴了Linux内核kfifo无锁队列, 可以无锁出入对, 支持多消费/生产者同时出入队。
4. 内存池管理组件 (Mem pool lib) : 为应用程序和其他组件提供分配内存池的接口, 内存池是一个由固定大小的多个内存块组成的内存容器, 可用于存储相同对象实体, 如报文缓存块等。内存池由内存池的名称来唯一标识, 它由一个环缓冲

区和一组核本地缓存队列组成，每个核从自己的缓存队列分配内存块，当本地缓存队列减少到一定程度时，从内存缓冲区中申请内存块来补充本地队列。

5. 网络报文缓存块管理组件 (Mbuf lib)：提供应用程序创建和释放用于存储报文信息的缓存块的接口，这些MBUF存储在内存池中。提供两种类型的MBUF，一种用于存储一般信息，一种用于存储报文信息。
6. 定时器组件 (Timer lib)：提供一些异步周期执行的接口（也可以只执行一次），可以指定某个函数在规定的时间内异步地执行，就像LIBC中的timer定时器，但是这里的定时器需要应用程序在主循环中周期调用rte\_timer\_manage来使定时器得到执行。定时器组件的时间参考来自EAL层提供的时间接口。

除了以上六个核心组件外，DPDK还提供以下功能：

- 以太网轮询模式驱动 (PMD) 架构：把以太网驱动从内核移到应用层，采用同步轮询机制而不是内核态的异步中断机制来提高报文的接收和发送效率。
- 报文转发算法支持：Hash 库和LPM库为报文转发算法提供支持。
- 网络协议定义和相关宏定义：基于FreeBSD IP协议栈的相关定义如：TCP、UDP、SCTP等协议头定义。
- 报文QOS调度库：支持随机早检测、流量整形、严格优先级和加权随机循环优先级调度等相关QOS 功能。
- 内核网络接口库 (KNI)：提供一种DPDK应用程序与内核协议栈的通信的方法、类似普通Linux的TUN/TAP接口，但比TUN/TAP接口效率高。每个物理网口可以虚拟出多个KNI接口。

## 三、安装DPDK

### 3.1 获取源码

在命令行中输入以下命令获取源码。

```
$ git clone git://dpdk.org/dpdk
$ git clone http://dpdk.org/git/dpdk-kmods
$ cp -r ./dpdk-kmods/linux/igb_uio ./dpdk/kernel/linux/
```

### 3.2 更改配置文件

需更改以下两个文件。

```
$ sudo vim dpdk/kernel/linux/meson.build
subdirs = ['kni', 'igb_uio']
```

```
$ sudo vim dpdk/kernel/linux/igb_uio/meson.build
-----
# 加入如下的内容:
-----
# SPDX-License-Identifier: BSD-3-Clause
# Copyright(c) 2017 Intel Corporation
mkfile = custom_target('igb_uio_makefile',
output: 'Makefile',
command: ['touch', '@OUTPUT@'])
custom_target('igb_uio',
input: ['igb_uio.c', 'kbuild'],
output: 'igb_uio.ko',
command: ['make', '-C', kernel_dir + '/build',
'M=' + meson.current_build_dir(),
'src=' + meson.current_source_dir(),
'EXTRA_CFLAGS=-I' + meson.current_source_dir() +
'../../../../../lib/librte_eal/include',
'modules'],
depends: mkfile,
install: true,
install_dir: kernel_dir + '/extra/dpdk',
build_by_default: get_option('enable_kmods'))
```

### 3.3 配置环境

在命令行中输入以下命令完成安装与环境配置。

```
$ sudo apt-get install python3 python3-pip
$ sudo pip3 install meson ninja pwntools
$ cd dpdk
$ sudo meson -D examples=all build
$ cd build
$ sudo ninja install
```

### 3.4 使用大页

在命令行中输入以下命令，开启大页使用。

```
$ sudo mkdir -p /dev/hugepages
$ sudo mountpoint -q /dev/hugepages || mount -t hugetlbfs nodev
/dev/hugepages
$ sudo chmod 777
/sys/devices/system/node/node0/hugepages/hugepages-
2048kB/nr_hugepages
$ sudo echo 512 >
/sys/devices/system/node/node0/hugepages/hugepages-
2048kB/nr_hugepages
```

此时，测试运行dpdk-helloworld。

```
hkz-vm1@hkz-virtual-machine-1:~/dpdk/build/examples$ sudo ./dpdk-helloworld
[sudo] hkz-vm1 的密码:
EAL: Detected CPU lcores: 2
EAL: Detected NUMA nodes: 1
EAL: Detected static linkage of DPDK
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'PA'
EAL: No available 1048576 kB hugepages reported
EAL: VFIO support initialized
EAL: Probe PCI driver: net_virtio (1af4:1041) device: 0000:01:00.0 (socket 0)
eth_virtio_pci_init(): Failed to init PCI device
EAL: Requested device 0000:01:00.0 cannot be used
TELEMETRY: No legacy callbacks, legacy socket not created
hello from core 1
hello from core 0
```

图3 测试运行

可以看到，helloworld虽然能运行，但是发现requested device不能使用，即还需要绑定网卡。

## 3.5 绑定网卡

查看网卡状态及信息，输入：

```
$ sudo ./dpdk-devbind.py --status
$ ifconfig
```

```
hkz-vm2@hkz-virtual-machine-2:~/dpdk/usertools$ sudo ./dpdk-devbind.py --status
[sudo] hkz-vm2 的密码:

Network devices using kernel driver
=====
0000:01:00.0 'Virtio network device 1041' if=enp1s0 drv=virtio-pci unused=vfio-pci *Active*
```



```

hkz-vm2@hkz-virtual-machine-2:~/dpdk/build/examples$ ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.64.134 netmask 255.255.255.0 broadcast 192.168.64.255
    inet6 fe80::8b31:4440:1f74:bc06 prefixlen 64 scopeid 0x20<link>
    ether 52:54:00:b7:4b:99 txqueuelen 1000 (以太网)
    RX packets 11761 bytes 69484581 (69.4 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 11451 bytes 663609 (663.6 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (本地环回)
    RX packets 205 bytes 17831 (17.8 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 205 bytes 17831 (17.8 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

图4&5 查看网卡状态及信息

输入以下命令，以完成绑定网卡。

```

$ sudo modprobe uio_pci_generic
$ sudo ifconfig enp1s0 down
$ sudo ./usertools/dpdk-devbind.py --bind=uio_pci_generic enp1s0

```

再次查看状态，可以看到网卡已经绑定成功。

```

hkz-vm2@hkz-virtual-machine-2:~/dpdk/usertools$ sudo ifconfig enp1s0 down
hkz-vm2@hkz-virtual-machine-2:~/dpdk/usertools$ sudo ./dpdk-devbind.py --bind=uio_pci_generic enp1s0
hkz-vm2@hkz-virtual-machine-2:~/dpdk/usertools$ sudo ./dpdk-devbind.py --status

Network devices using DPDK-compatible driver
=====
0000:01:00.0 'Virtio network device 1041' drv=uio_pci_generic unused=vfio-pci

```

图6 网卡绑定成功

## 3.6 测试运行

在/dpdk/build/examples中测试dpdk-helloworld的运行。

```

hkz-vm2@hkz-virtual-machine-2:~/dpdk/build/examples$ sudo ./dpdk-helloworld
[sudo] hkz-vm2 的密码:
EAL: Detected CPU lcores: 2
EAL: Detected NUMA nodes: 1
EAL: Detected static linkage of DPDK
EAL: Multi-process socket /var/run/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'PA'
EAL: No available 1048576 kB hugepages reported
EAL: VFIO support initialized
EAL: Probe PCI driver: net_virtio (1af4:1041) device: 0000:01:00.0 (socket 0)
TELEMETRY: No legacy callbacks, legacy socket not created
hello from core 1
hello from core 0

```

图7 测试helloworld运行

每个核发送一个hello，运行成功。至此两个vm上的dpdk都已经安装成功。

## 四、 VM2上编译运行l2fwd

### 4.1 l2fwd原理

l2fwd应用程序为RX\_PORT上接收的每个数据包执行二层转发。目标端口是启用的端口掩码的相邻端口，即，如果启用前四个端口（端口掩码0xf，每个端口用一个比特位表示，启动4个就是4个比特位置1），端口1和2相互转发，端口3和4相互转发。此外，如果启用了MAC地址更新，则MAC地址按照如下方式更新：

数据包的源mac会变成发送端口的mac地址。

数据包的目的mac会变成02:00:00:00:00:发送端口的ID。

也就是说在不开启mac地址更新的情况下，仅仅对数据包进行紧邻端口的转发不对数据包的mac地址进行修改，如果开启的话，会对需要转发的数据包的mac地址按照如上的规则进行更新。

转发的模式如下：

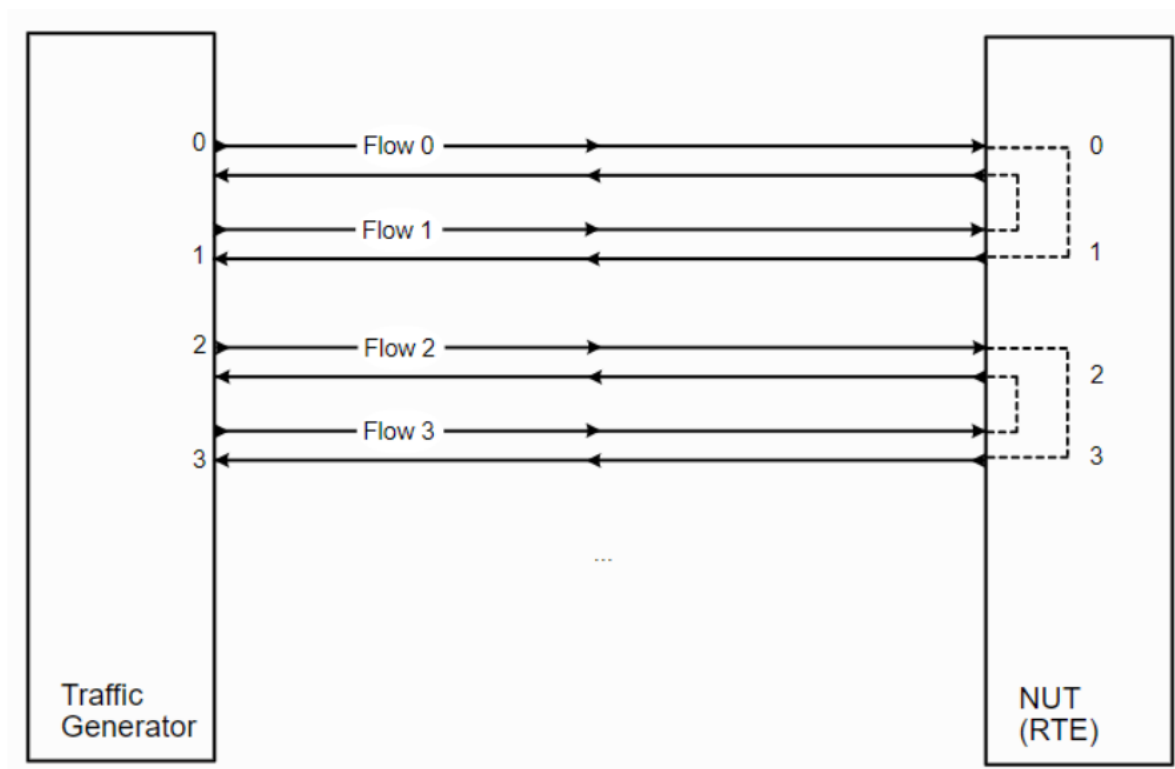


图8 l2fwd转发模式



也就是说端口0和端口1是相邻的两个端口，2和3是相邻的两个端口，0口进的数据包会从1口发出，1口进的数据包会从0口发出，同理2和3两个端口的转发逻辑也是一样的。

## 4.2 编译运行l2fwd

在/dpdk/build/examples中编译运行dpdk-l2fwd。

输入如下命令：

```
$ sudo ./dpdk-l2fwd -c 1 -n 2 -- -p 1 -q 1 -T 15
```

l2fwd的参数解析：

EAL(DPDK 环境抽象层参数)形式必须为-c COREMASK -n NUM；

COREMASK 是一个十六进制的掩码表示分配的逻辑内核数量；

NUM 表示一个十进制整数表示每个逻辑内核的内存通道数量。

L2fwd 的应用程序参数：

-p PORTMASK:一个十六进制位掩码表示分配的端口的数量，例如 0x3 表示分配端口 0 和 1；

-q NQ NQ 表示分配给每个逻辑内核的收发队列数量；

-T t t 表示统计数据打印到屏幕上的时间间隔。

测试指令含义即为分配 1 个逻辑内核，每个逻辑内核2个内存通道，1 个收发队列，分配一个端口(端口 0)，每隔15秒将数据打印到屏幕上。

出现如下运行结果，表示l2fwd运行成功。

```
Checking link statusdone
Port 0 Link up at Unknown FDX Autoneg
L2FWD: entering main loop on lcore 0
L2FWD: -- lcoreid=0 portid=0

Port statistics =====
Statistics for port 0 -----
Packets sent:                                0
Packets received:                            0
Packets dropped:                             0
Aggregate statistics =====
Total packets sent:                          0
Total packets received:                      0
Total packets dropped:                       0
=====
```

图9 l2fwd运行结果

## 五、vm1上编译运行pktgen-dpdk

### 5.1 安装编译pktgen-dpdk

pktgen-dpdk是用于对DPDK进行高速数据包测试的工具。我发现，在我安装的dpdk中没有提供pktgen-dpdk，故还需另外安装编译pktgen-dpdk。

输入以下命令以完成安装编译。

```
$ git clone git://dpdk.org/apps/pktgen-dpdk
$ cd pktgen-dpdk
$ sudo git checkout master
$ sudo meson build
$ cd build
$ sudo ninja
```

### 5.2 运行pktgen-dpdk

在/pktgen-dpdk/build/app中运行pktgen。

输入以下命令：

```
$ sudo ./pktgen -l 0-1 -n 2 --file-prefix pg -- -P -m"[1].0"
```

参数解析：

EAL 命令行选项：

-l 0-1 指定使用核 0 和核 1(pktgen 要求必须要使用两种核)；

-n 2 意味着每个处理器使用两个内存通道；

--file-prefix 为 DPDK 进程设置一个不同的共享文件前缀，命名为 pg，此处只有一个文件前缀名，因此仅仅设置了一个独立的 DPDK 进程。

pktgen 命令行选项：

-P 开始混杂模式；

-m DPDK 网卡和逻辑核之间的矩阵映射，[1].0 表示逻辑核 1 处理端口 0 的收发包 (rx/tx)。

Pktgen 的命令行含义：

start portnum 端口 portnum 开始发包，默认情况下一直进行收包工作；

stop portnum 端口 portnum 停止发包；

quit 退出 pktgen 命令行界面；

set portnum src|dst mac 设定端口 portnum 发送的源/目的 MAC 地址。

出现如下运行结果，表示pktgen运行成功。

```

*** Copyright(c) <2010-2021>, Intel Corporation. All rights reserved.
|** Pktgen  created by: Keith Wiles -- >>> Powered by DPDK <<<
P-----Sngl      :0
Port: Name      I  <UP-4294967295-FD>UMA  PCI
   0: net_virtio      0      00
                        0      0
Initialize Port 0 --      0/0      0/0
Src MAC 52:54:00:13:1      4      4
<Promiscuous mode En      0      0
                        0
RX/TX processing lc      01
                        3
- Ports 0-0 of 1 <M      1c) <2010-2021>, Intel Corporation
  Flags:Port      :      0
-link State      :      0      ---Total Rate---
Pkts/s Rx      :      0      0
  Tx      :      0/0      0
Mbits/s Rx/Tx      :      0/0      0
Pkts/s Rx Max      :      0/0      0
  Tx Max      :      4      0/0
Broadcast      :      0      4
Multicast      :      0/0      0
Sizes 64      :      0
      65-127      :      0
      128-255      :      0
      256-511      :      3
      512-1023      :      1
      1024-1518      :      0
Runts/Jumbos      :      0
ARP/ICMP Pkts      :      0
Errors Rx/Tx      :      0/0
Total Rx Pkts      :      0/0
  Tx Pkts      :      0/0
  Rx/Tx MBs      :      4
TCP Flags      :      0
-- Pktgen 21.11.0 (D:      0/0red by DPDK (pid:4076) -----

```

图10 pktgen运行结果

## 六、l2fwd与pktgen同时运行

我尝试着让VM2运行l2fwd，同时VM1运行pktgen。

在pktgen出现的命令行中输入：

```

>set 0 dst mac 52:54:00:b7:4b:99
>start 0

```

运行结果如下：

```
hkz-vm1@hkz-virtual-machine-1: ~/pktgen-dpdk/build/app
\Port: Name      IfIndex Alias      NUMA  PCI
      0: net_virtio  P-----Sngl      :0 0  1af4:1041/01:00.0
      <UP-4294967295-FD>
Initialize Port 0 --      0      0
Src MAC 52:54:00:13:1      769,856      769,856
<Promiscuous mode En      0/517      0/517
      90      90
      1,100,288      1,100,288
RX/TX processing lc      01
      15
- Ports 0-0 of 1 <M      7c) <2010-2021>, Intel Corporation
Flags:Port      :      60
Link State      :      18      ---Total Rate---
-kPorts 0-0 of 1 <M      2c) <2010-2021>, Intel Corporation
/ Ports 0-0 of 1 <M      0c) <2010-2021>, Intel Corporation
Flags:Port      :      0      ---Total Rate---
Link State      :      3/0      ---Total Rate---
Pkts/s Rx      :      9/0      0
Tx      :      0/0      272,832
Mbits/s Rx/Tx      :      90      0/183
Pkts/s Rx Max      :      51,281,664      90
Tx Max      :      0/34,461      1,100,288
Broadcast      :      0
Multicast      :      15
Sizes 64      :      7
      65-127      :      60
      128-255      :      18
      256-511      :      2
      512-1023      :      0
      1024-1518      :      0
Runts/Jumbos      :      3/0
ARP/ICMP Pkts      :      9/0
Errors Rx/Tx      :      0/0
Total Rx Pkts      :      90red by DPDK (pid:2130) -----
Tx Pkts      :      39,375,872 Interface without timers
Rx/Tx MBs      :      0/26,460
TCP Flags      :      .A....
Pkts/s
```

```
Port statistics =====
Statistics for port 0 -----
Packets sent:      8267790
Packets received:      8803096
Packets dropped:      535306
Aggregate statistics =====
Total packets sent:      8267790
Total packets received:      8803096
Total packets dropped:      535306
=====
```

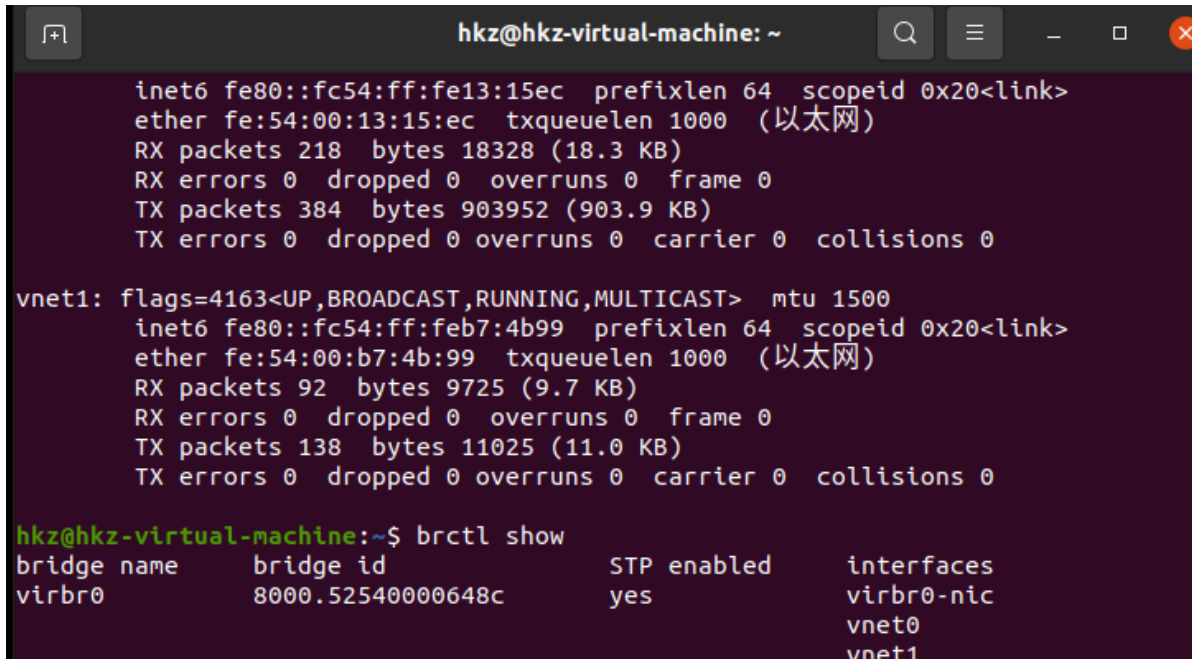
图11&12 l2fwd与pktgen运行结果

可以看到，l2fwd有数据，说明VM2有接收到VM1发来的包并转发回去。但是VM1的pktgen中接收包的数据为0，说明VM1并未接收到VM2转发来的包，这与预期结果不相符。

经过查阅资料后，我发现，实验缺少了关键的一步，即VM1与VM2的网卡间需要建立一个网桥。

回到host主机的客户端，输入：

```
$ ifconfig
$ brctl show
```



```
hkz@hkz-virtual-machine: ~
inet6 fe80::fc54:ff:fe13:15ec prefixlen 64 scopeid 0x20<link>
ether fe:54:00:13:15:ec txqueuelen 1000 (以太网)
RX packets 218 bytes 18328 (18.3 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 384 bytes 903952 (903.9 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vnet1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet6 fe80::fc54:ff:feb7:4b99 prefixlen 64 scopeid 0x20<link>
ether fe:54:00:b7:4b:99 txqueuelen 1000 (以太网)
RX packets 92 bytes 9725 (9.7 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 138 bytes 11025 (11.0 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

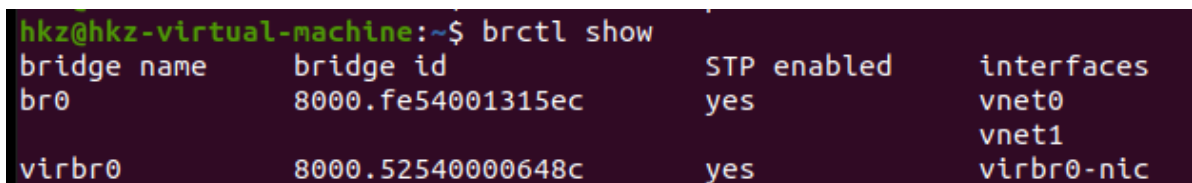
hkz@hkz-virtual-machine:~$ brctl show
bridge name      bridge id                STP enabled  interfaces
virbr0           8000.52540000648c       yes          virbr0-nic
                                                         vnet0
                                                         vnet1
```

图13 查看网卡、网桥信息

需要新建一个网桥，将vnet0和vnet1（VM1与VM2的网卡）连到网桥的两端。  
输入以下命令：

```
$ sudo brctl addbr br0
$ sudo brctl delif virbr0 vnet0
$ sudo brctl delif virbr0 vnet1
$ sudo brctl addif br0 vnet0
$ sudo brctl addif br0 vnet1
```

结果如下：



```
hkz@hkz-virtual-machine:~$ brctl show
bridge name      bridge id                STP enabled  interfaces
br0              8000.fe54001315ec       yes          vnet0
                                                         vnet1
virbr0           8000.52540000648c       yes          virbr0-nic
```

图14 网桥建立

可以看到，此时网桥已经建立完成。由于网桥不会形成环，可以将STP enabled 设为no。此外，还需启动该网桥。输入以下命令：

```
$ sudo brctl stp br0 off
$ sudo ifconfig br0 up
```

此时，回到VM1与VM2内，重新搭好环境（使用大页、绑定网卡等），再次让VM2运行l2fwd，同时VM1运行pktgen。

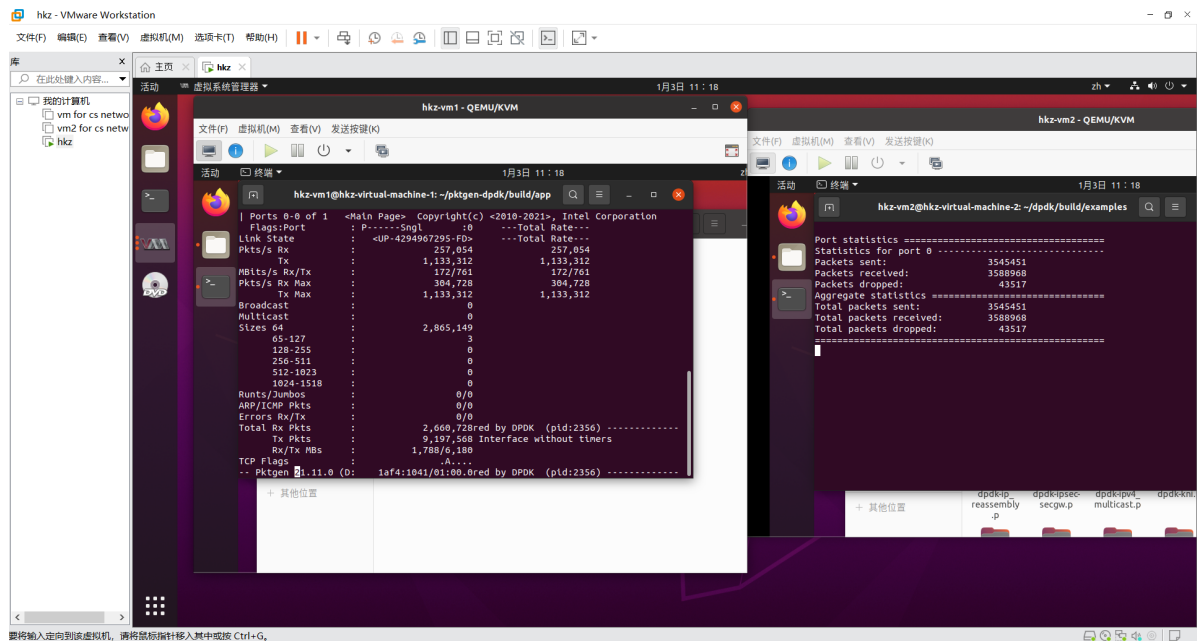


图15 运行结果

可以看到，此时运行结果与预期结果相符。VM1向VM2发包，VM2转发回VM1，VM1接受。实现了二层转发。

## 七、评估DPDK二层转发的性能

### 7.1 l2fwd性能

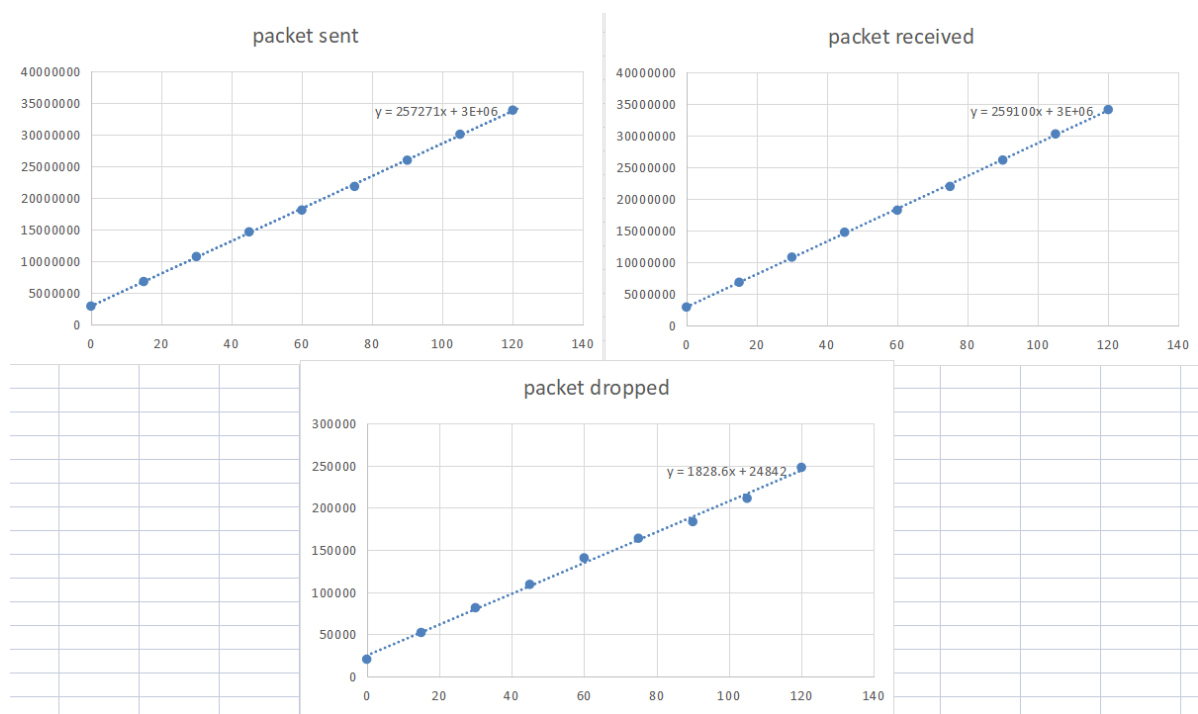
当l2fwd开始大量出现数据时，每隔15秒记录数据。

时间	packet sent	packet received	packet dropped
0	2860544	2881064	20520
15	6747927	6800140	52213
30	10692083	10773671	81588
45	14601254	14710504	109250
60	18054131	18194776	140645
75	21789941	21953872	163931
90	25936596	26120216	183620
105	30041872	30253160	211288
120	33856725	34104686	247961

图16 l2fwd数据记录

绘制折线图如下：





l2fwd数据折线图

由上图可以发现，l2fwd 每 15s 能够接收大约 259100 个数据包，其中成功转发的大概有 257271 个，被丢弃的大概有 1829 个，数据包丢弃率为 0.70%左右。

## 7.2 pktgen性能

当pktgen开始大量出现数据时，每隔15秒进行一次截图，并将数据记录下来。

时间	Pkts/s Rx	Pkts/s Tx	MBits/s Rx/Tx	Pkts/s Rx Max	Pkts/s Tx Max	时间	Total Rx pkts	时间	Total Tx pkts
0	168760	376000	113/252	168760	376000	0	661804	0	1465280
15	165592	338752	111/227	198176	438144	15	3444604	15	7397568
30	159056	336832	106/226	205369	462208	30	5918988	30	12813184
45	159157	351360	106/236	205360	462208	45	8343588	45	17909824
60	132476	270848	89/182	222520	491392	60	10406564	60	22503808
75	156912	353280	105/237	222520	491392	75	12330892	75	26702080
90	180912	413760	121/278	222520	491392	90	14890108	90	32223360
105	173246	389888	116/262	222520	491392	105	17469642	105	37882880
120	173392	369408	116/248	222520	491392	120	21698180	120	47273856

图19 pktgen数据记录

将收发包数据绘制成折线图如下：

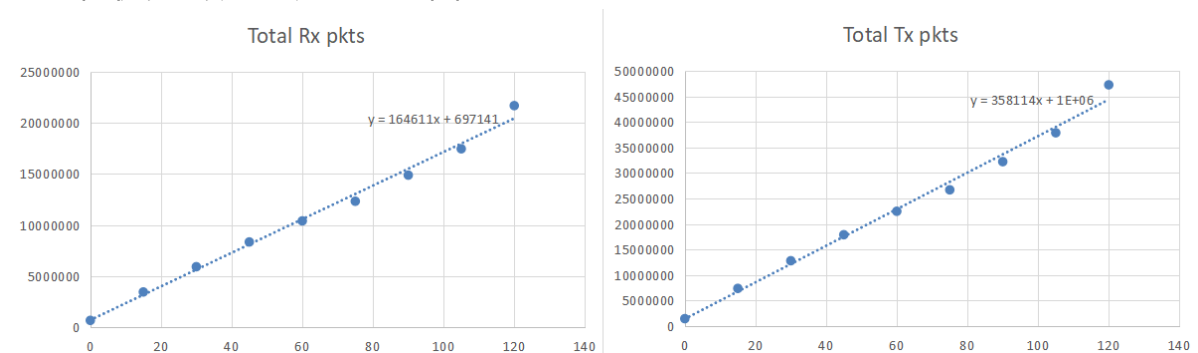


图20 pktgen数据折线图

由上图可以发现，pktgen 每 15s 能够接发送大约 358114 个数据包，成功接收到的有大约164611个数据包。

## 7.3 二层转发性能评估

综合7.1和7.2，我们可以得到，每15秒，VM1的pktgen发出的包约为358114，VM2的l2fwd接收到的包约为259100，丢包率为27.65%。VM2的l2fwd转发的包约为257211，VM1的pktgen接收到的包约为164611，丢包率约为36.00%。二层转发的总丢包率为54.03%。**丢包率如此高，可能是因为发送出去的数据包触发了局域网的拥塞控制或出现了传输差错而被丢弃。网卡接收数据包时，需要维护一个缓冲区队列，来缓存可能存在的突发数据，为了实现数据链路层的拥塞控制，收发队列缓冲区往往有一个接收上限，超过上限时后续进入的数据包将被丢弃。由于本次实验只用了一个端口共一个收发队列，因此丢包率较高。**

同时，我们从图15中可以看到，pktgen发出的包绝大部分大小均为64MB。由此可以推测出 pktgen 更倾向发送偏小的数据包，这可能是由于数据包长度小的帧在传输 中出错的概率更低，在数据链路层二层回转中，数据包长度越小越容易存活转发效率越高。

## 八、总结

---

在这次大作业中，我选择了DPDK performance test。我在KVM上创建了两个虚拟机，并在创建的虚拟机上安装编译了DPDK环境。然后我在VM2上运行了l2fwd，在VM1上运行了pktgen-dpdk，并评估了DPDK的二层转发的性能。通过学习，我对二层转发有了更深刻的理解，也懂得了如何将理论知识与实际相结合。

在过程中我遇到了许多困难，如配环境出bug，pktgen无法收到转发的包等等。但在查阅资料、询问助教老师后，我都一一解决，也让我感到了满满的成就感。

这次大作业是本学期的最后一次作业。回顾这一学期，通过5次作业，我已经熟练地掌握了qemu-kvm虚拟化的常见操作与适用技术，对云计算、虚拟化都有了更深刻的理解。非常感谢老师能循循善诱地带领我打开虚拟化这扇大门，非常感谢助教老师能耐心解答我的疑问。总的来说，这一学期的学习让我收获颇丰。

## 九、参考

---

[https://blog.csdn.net/weixin\\_44602409/article/details/114894700?utm\\_medium=distribute.pc\\_aggpage\\_search\\_result.none-task-blog-2~aggregatepage~first\\_rank\\_ecpm\\_v1~rank\\_v31\\_ecpm-2-114894700.pc\\_agg\\_new\\_rank&utm\\_term=dpdk%E5%AE%89%E8%A3%85+ubuntu&spm=1000.2123.3001.4430](https://blog.csdn.net/weixin_44602409/article/details/114894700?utm_medium=distribute.pc_aggpage_search_result.none-task-blog-2~aggregatepage~first_rank_ecpm_v1~rank_v31_ecpm-2-114894700.pc_agg_new_rank&utm_term=dpdk%E5%AE%89%E8%A3%85+ubuntu&spm=1000.2123.3001.4430)

[https://blog.csdn.net/weixin\\_38197902/article/details/121272255](https://blog.csdn.net/weixin_38197902/article/details/121272255)

[https://blog.csdn.net/weixin\\_39286923/article/details/100030426](https://blog.csdn.net/weixin_39286923/article/details/100030426)

<https://blog.csdn.net/choumin/article/details/120884509>