

EI313 Project 5

游灏溢 (519030910193, yuri-you@sjtu.edu.cn)

工程实践与科技创新 III – D
计算机科学与技术系，上海交通大学

摘要 本次项目中，本人创建两个虚拟机，分别在两个虚拟机中安装 **dppdk**，在虚拟机之间用 **dppdk** 的应用程序之一 **l2fwd** 进行传包。在网络传递过程中，使用了 **pktgen-dppdk** 来测试 **dppdk** 网络传输的表现和性能。

1 相关技术介绍

1.1 DPDK

DPDK 是 INTEL 公司开发的一款高性能的网络驱动组件，旨在为数据面应用程序提供一个简单方便的，完整的，快速的数据包处理解决方案，主要技术有用户空间的 I/O 技术 (UIO)，用户空间轮询模式 (PMD) 以及大页内存等等。

用户空间的 I/O 技术 (UIO)

提供应用空间下驱动程序的支持，也就是说网卡驱动是运行在用户空间的，减下了报文在用户空间和应用空间的多次拷贝。DPDK 绕过了 Linux 内核的网络驱动模块，直接从网络硬件到达用户空间，不需要进行频繁的内存拷贝和系统调用 (如图 1)。根据官方给出的数据，DPDK 裸包反弹每个包需要 80 个时钟周期，而传统 Linux 内核协议栈每包需要 2k 4k 个时钟周期。DPDK 能显著提升虚拟化网络设备的数据采集效率。

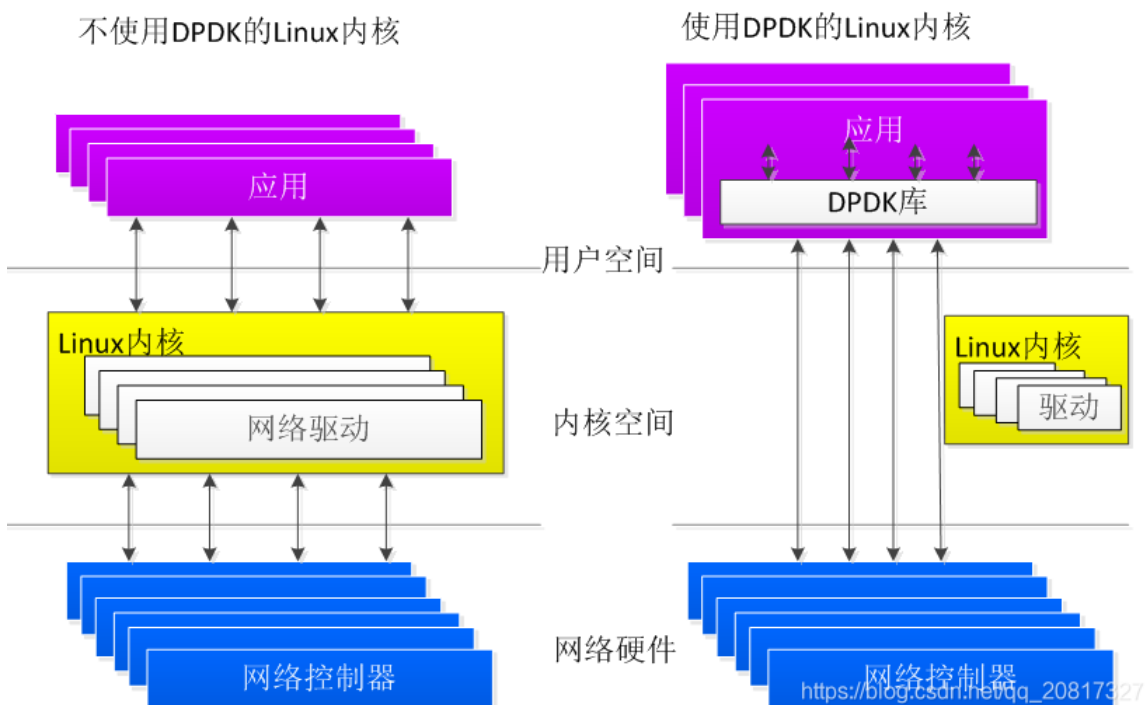


图1. UIO 技术介绍

用户空间轮询模式 (PMD)

传统 Linux 系统中，当网络设备检测到数据帧过来的时候，会使用 DMA（直接内存访问）将帧发送到预先分配好的内核缓冲区里面，然后更新相应的接收描述符环，之后产生中断通知有数据帧过来。Linux 系统会进行相应的响应，然后更新相应的描述符环，再将接收到的数据帧交给内核中的网络堆栈进行处理，网络堆栈处理完之后会将相应的数据拷贝到相应的套接字，从而数据就被复制到了用户空间，应用程序就可以使用这些数据了。

使用 PMD 后，用户空间驱动使得应用程序不需要经过 linux 内核就可以访问网络设备卡。网卡设备可以通过 DMA 方式将数据包传输到事先分配好的缓冲区，这个缓冲区位于用户空间，应用程序通过不断轮询的方式可以读取数据包并在原地址上直接处理，不需要中断，而且也省去了内核到应用层的数据包拷贝过程，(如图 2)。

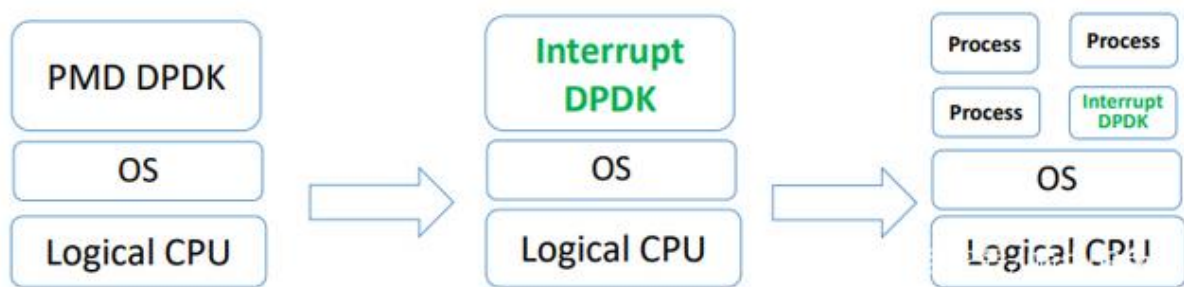


图 2. PMD 技术介绍

大页内存

Linux 操作系统通过查找 TLB 来实现快速的虚拟地址到物理地址的转化。由于 TLB 是一块高速缓冲 cache，容量比较小，容易发生没有命中。当没有命中的时候，会触发一个中断，然后会访问内存来刷新页表，这样会造成比较大的时延，降低性能。Linux 操作系统的页大小只有 4K，所以当应用程序占用的内存比较大的时候，会需要较多的页表，开销比较大，而且容易造成未命中。相比于 linux 系统的 4KB 页，Intel DPDK 缓冲区管理库提供了 Hugepage 大页内存，大小有 2MB 和 1GB 页面两种，可以得到明显性能的提升，因为采用大页内存的话，可以需要更少的页，从而需要更少的 TLB，这样就减少了虚拟页地址到物理页地址的转换时间。

1.2 l2fwd

l2fwd 为 DPDK 的应用程序之一，其为 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，仅仅对数据包进行相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。

参照官网给出的图，转发的模式如图 3

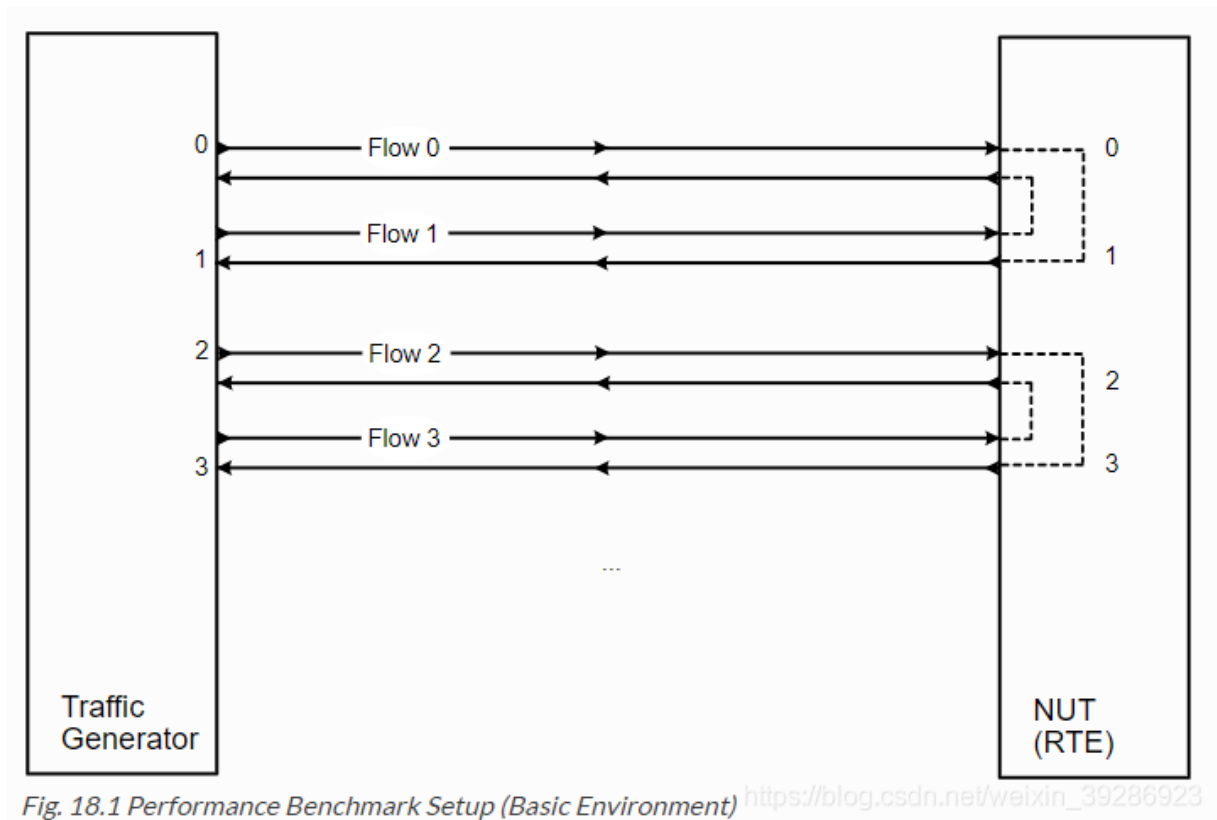


图3. l2fwd 技术介绍

1.3 pktgen-dpdk

pktgen-dpdk 是用于对 DPDK 进行高速数据包测试的工具，可以用于测试其数据包接受转发的性能。

2 环境配置安装过程

2.1 实验环境

本机是在 Ubuntu 20.04 LTS 系统的物理机中，使用 qemu 安装了 2 个虚拟机，并且在虚拟机中安装 Ubuntu 18.04 系统。并且在虚拟机物理机中均使用了 KVM 模块并且打开 2M 的巨页。相关 qemu 虚拟机安装，巨页、KVM 配置过程参见 project 1-4。

2.2 DPDK 的安装和使用

在安装了 2 台虚拟机的基础上，我们需要在每台虚拟机上分别安装 DPDK 软件，并且在其他一台虚拟中安装 pktgen-dpdk 的测试软件，安装步骤如下。

获取 DPDK 源码

DPDK 源码很多地方都可以获得，可以从[官网下载源码](#)，也可以通过第三方软件获得。此处使用 github 中源码，在终端输入

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

获得相关源代码后,需要对其配置进行一些修改,以方便之后对项目的构建。在 dpdk/kernel/linux/meson.build 文件中在

```
subdirs = ['kni']
```

后加入'igb_uio'一项,并且添加 dpdk/kernel/linux/igb_uio/meson.build 文件,加入如下内容

```
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'))
```

编译安装

编译 DPDK 需要用到 ninja 和 meson 等构建编译工具以及 python 相关库,首先需配置环境

```
$ sudo apt install python3 python3-pip
$ sudo pip3 install meson ninja pwntools
```

然后进入 dpdk 文件夹内使用 meson 进行构建

```
$ sudo meson -D examples=all build
```

其中 examples=all build 说明需要构建所有的 examples

构建之后会生成 build 文件夹,进入后使用 ninja 对项目进行编译安装

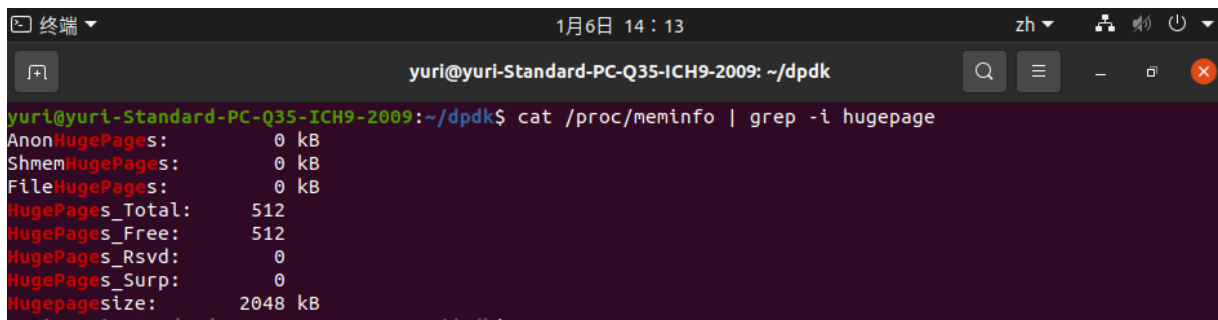
```
$ sudo ninja install
```

开启巨页

使用 dpdk 需要用到其巨页功能，需在虚拟机中开启 2M 巨页，在终端输入以下命令

```
$ 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
```

使用指令”\$ cat /proc/meminfo |grep -i hugepages” 可以查看巨页情况，如图所示，虚拟机的巨页已经打开。



```
yuri@yuri-Standard-PC-Q35-ICH9-2009: ~/dpdk$ cat /proc/meminfo | grep -i hugepage
AnonHugePages:        0 kB
ShmemHugePages:       0 kB
FileHugePages:        0 kB
HugePages_Total:      512
HugePages_Free:       512
HugePages_Rsvd:       0
HugePages_Surp:       0
Hugepagesize:        2048 kB
```

图 4. 打开巨页

绑定网卡

在 dpdk/usertools 文件夹下输入指令

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

结果如 5 所示，可见目前网络数据包收发还是通过 linux 内核驱动，需要将其改为从 DPDK 驱动。



```
yuri@yuri-Standard-PC-Q35-ICH9-2009: ~/dpdk/usertools$ sudo ./dpdk-devbind.py --status
[sudo] yuri 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*

No 'Baseband' devices detected
=====

No 'Crypto' devices detected
=====

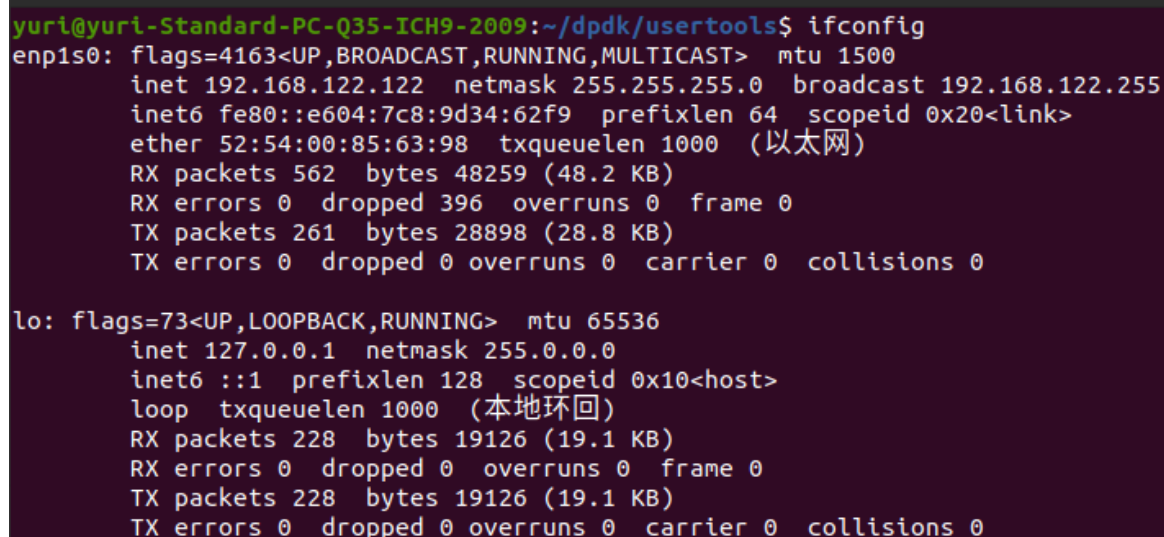
No 'DMA' devices detected
=====

No 'Eventdev' devices detected
=====
```

图 5. Linux Kernel Driver

首先通过 net-tools 查询机器的网卡信息

```
$ sudo apt install net-tools
$ ifconfig
```



```
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/dpdk/usertools$ ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.122.122 netmask 255.255.255.0 broadcast 192.168.122.255
    inet6 fe80::e604:7c8:9d34:62f9 prefixlen 64 scopeid 0x20<link>
    ether 52:54:00:85:63:98 txqueuelen 1000 (以太网)
    RX packets 562 bytes 48259 (48.2 KB)
    RX errors 0 dropped 396 overruns 0 frame 0
    TX packets 261 bytes 28898 (28.8 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 228 bytes 19126 (19.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 228 bytes 19126 (19.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

图6. 网卡信息

可知网卡名称为 enp1s0, mac 地址是 52: 54: 00: 85: 63: 98。现在需要将 dpdk 绑定到该网卡上。首先先断开网卡, 使其不在激活状态

```
$ ifconfig enp1s0 down
```

再将其与 dpdk 进行绑定, 此处使用 dpdk/usertools 中 dpdk-devbind.py 脚本

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

最后再查询 dpdk 状态

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

由图 7可知, 已经成功将网卡 enp1s0 绑定至 dpdk 上, 现在网络设备使用的是 DPDK-compatible 的驱动



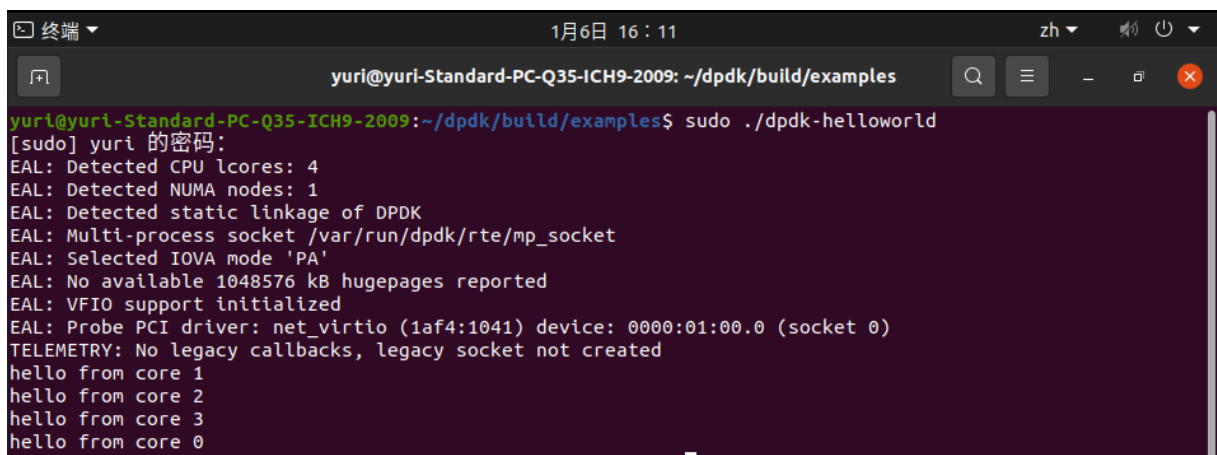
```
yuri@yuri-Standard-PC-Q35-ICH9-2009: ~/dpdk/usertools
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/dpdk/usertools$ sudo ifconfig enp1s0 down
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/dpdk/usertools$ sudo dpdk-devbind.py --bind=
uio_pci_generic enp1s0
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/dpdk/usertools$ sudo dpdk-devbind.py --sta
tus

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

图 7. 绑定网卡

最后运行 dpdk-helloworld 程序来验证一切正常, 如图 ??

```
$ sudo ./dpdk-helloworld
```



```
终端 1月6日 16:11 zh
yuri@yuri-Standard-PC-Q35-ICH9-2009: ~/dpdk/build/examples
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/dpdk/build/examples$ sudo ./dpdk-helloworld
[sudo] yuri 的密码:
EAL: Detected CPU lcores: 4
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 2
hello from core 3
hello from core 0
```

图 8. 运行验证程序

2.3 pktgen-dpdk 的安装和使用

pktgen-dpdk 是用于对 DPDK 进行高速数据包测试的工具。但是它不是随着 dpdk 默认同时安装的, 故还需另外安装编译 pktgen-dpdk。

pktgen-dpdk 的源码下载

此次实验中 pktgen-dpdk 的源码来自于 GitHub, 在终端输入命令

```
$ git clone git://dpdk.org/apps/pktgen-dpdk
```

pktgen-dpdk 的编译安装

进入文件夹 pktgen-dpdk, 使用 meson 进行构建


```
$ sudo meson build
```

会发现进行报错，如图 9所示。

```
Library librte_net_bond found: YES
Program python3 found: YES (/usr/bin/python3)
Library rte_net_i40e found: YES
Library rte_net_ixgbe found: YES
Library rte_net_ice found: YES
Library rte_bus_vdev found: YES
Run-time dependency threads found: YES

app/meson.build:36:0: ERROR: C library 'numa' not found

A full log can be found at /home/yuri/pktgen-dpdk/build/meson-logs/meson-log.txt
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/pktgen-dpdk$
```

图9. 缺少 numa 库

这是因为 numa 的动态链接库不是 C 自带，需要额外安装进行安装，在终端中输入

```
$ sudo apt install libnuma-dev
```

同样的还会出现缺少 pcap 库，同样使用 apt 进行安装

```
$ sudo apt install libpcap-dev
```

之后再次对 pktgen-dpdk 进行构建，出现图 10结果说明构建成功。

```
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/pktgen-dpdk$ sudo meson build
[sudo] yuri 的密码:
The Meson build system
Version: 0.53.2
Source dir: /home/yuri/pktgen-dpdk
Build dir: /home/yuri/pktgen-dpdk/build
Build type: native build
Program cat found: YES (/usr/bin/cat)
Project name: pktgen
Project version: 21.11.0
C compiler for the host machine: cc (gcc 9.3.0 "cc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0")
C linker for the host machine: cc ld.bfd 2.34
Host machine cpu family: x86_64
Host machine cpu: x86_64
Compiler for C supports arguments -mavx: YES
Compiler for C supports arguments -mavx2: YES
Compiler for C supports arguments -Wno-pedantic -Wpedantic: YES
Compiler for C supports arguments -Wno-format-truncation -Wformat-truncation: YES
Found pkg-config: /usr/bin/pkg-config (0.29.1)
Run-time dependency libdpdk found: YES 22.03.0-rc0
Library librte_net_bond found: YES
Program python3 found: YES (/usr/bin/python3)
Library rte_net_i40e found: YES
Library rte_net_ixgbe found: YES
Library rte_net_ice found: YES
Library rte_bus_vdev found: YES
Run-time dependency threads found: YES
Library numa found: YES
Library pcap found: YES
Library dl found: YES
Library m found: YES
Program sphinx-build found: NO
Build targets in project: 9

Found ninja-1.10.2.git.kitware.jobserver-1 at /usr/local/bin/ninja
```

图10. 构建 pktgen-dpdk

pktgen-dpdk 的使用

在/pktgen-dpdk/build/app 中的终端中输入命令

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

其中参数用处如下

1. **-l 0-1**: 指定使用核 0 和核 1(pktgen 要求必须要使用两种核)。
2. **-n 2**: 每个处理器使用两个内存通道。
3. **--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。
4. **-P**: 开始混杂模式, 即为接受所有端口发送的包。
5. **-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)。

运行后如下图 ??所示即说明 pktgen-dpdk 运行成功。

```
yuri@yuri-Standard-PC-Q35-ICH9-2009: ~/pktgen-dpdk/build/app
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/pktgen-dpdk/build/app$ sudo ./pktgen -l 0-1 -n 2 --file-prefix pg -- -P -m "[1].0"

Copyright(c) <2010-2021>, Intel Corporation. All rights reserved. Powered by DPDK
EAL: Detected CPU lcores: 4
EAL: Detected NUMA nodes: 1
EAL: Detected shared linkage of DPDK
EAL: Multi-process socket /var/run/dpdk/pg/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

*** Copyright(c) <2010-2021>, Intel Corporation. All rights reserved.
*** Pktgen created by: Keith Wiles -- >>> Powered by DPDK <<<

Port: Name      IfIndex Alias      NUMA PCI
   0: net_virtio      0              0 1af4:1041/01:00.0

Initialize Port 0 -- TxQ 1, RxQ 1
Src MAC 52:54:00:85:63:98
<Promiscuous mode Enabled>

RX/TX processing lcore:  1 rx:  1 tx:  1

- Ports 0-0 of 1 <Main Page> Copyright(c) <2010-2021>, Intel Corporation
  Flags:Port      : P-----Sngl      :0
-link State      : <UP-4294967295-FD>    ---Total Rate---
Pkts/s Rx        : P-----Sngl      :0
Tx               : <UP-4294967295-FD>    0
Mbits/s Rx/Tx    :                   0
Pkts/s Rx Max    :                   0
Tx Max           :                   0/0
Broadcast        :                   1
```

图 11. 运行 pktgen-dpdk

2.4 l2fwd 的使用

l2fwd 无需额外安装, 是 dpdk 自带的应用, 使用方式为在/dpdk.build/examples 文件夹下的终端输入命令

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

其中各个参数作用为

1. **-c**: 分配的逻辑内核数量，必须为一个十六进制数的形式。
2. **-n**: 分配给每个逻辑内核的内存通道数量，必须为一个十进制数的形式。
3. **-**分割 dpdk-l2fwd 两层。其中之前为 DPDK 环境抽象层参数，之后 l2fwd 的应用程序层参数。
4. **-p**: 分配的端口的数量，必须为一个十六进制数的形式。
5. **-q**: 分配给每个逻辑内核的收发队列数量，必须为一个十进制数的形式。
6. **-T**: 表示统计数据打印到屏幕上的时间间隔，单位为秒。

以上代码仅为测试使用示例，在真实实验中会修改参数。运行命令后出现如图 12 结果，表示 dpdk-l2fwd 运行成功。

```
yuri@yuri-Standard-PC-Q35-ICH9-2009: ~/dpdk/build/examples
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/dpdk/build/examples$ sudo ./dpdk-l2fwd -c 1 -n 2 -- -p 1 -q 1 -T 15
EAL: Detected CPU lcores: 4
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
MAC updating enabled
Notice: odd number of ports in portmask.
Lcore 0: RX port 0 TX port 0
Initializing port 0... done:
Port 0, MAC address: 52:54:00:85:63:98

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 12. l2fwd 运行示例

注意事项说明

在 l2fwd 中默认开启混杂模式，这样会接受流过其所有包。当在公开网络下（如 sjtu）运行该测试时候，需要将 l2fwd 的源码仅需修改，即注释以下代码

```
rte_eth_promiscuous_enable(portid)
```

由于本人在局域网下测试，故此造成的误差可以忽略。

2.5 连接两个虚拟机

根据以上结果，使用虚拟机 1 (VM1) 运行 pktgen-dpdk, 虚拟机 2 (VM2) 运行 l2fwd, 并且使用网桥将其连接。

先新建一个网桥，再将两个虚拟机的网卡接到该网桥上，这一些步骤需再 host 主机上进行，首先再 host 上使用 ifconfig 查询网卡状态，找到两个虚拟网卡 vnet0, vnet1，如图 13 所示。

```
(base) yuri@yuri-Inspiron-7590:~/EI313/my_project$ ifconfig
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 1123 bytes 98467 (98.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1123 bytes 98467 (98.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

virbr0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.122.1 netmask 255.255.255.0 broadcast 192.168.122.255
    ether 52:54:00:fe:36:c6 txqueuelen 1000 (以太网)
    RX packets 2242 bytes 150073 (150.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3166 bytes 10521626 (10.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vnet0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::fc54:ff:fe85:6398 prefixlen 64 scopeid 0x20<link>
    ether fe:54:00:85:63:98 txqueuelen 1000 (以太网)
    RX packets 2242 bytes 181461 (181.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 5798 bytes 10662413 (10.6 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vnet1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::fc54:ff:fec0:86f2 prefixlen 64 scopeid 0x20<link>
    ether fe:54:00:c0:86:f2 txqueuelen 1000 (以太网)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1 bytes 90 (90.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

图 13. 连接虚拟网卡

之后新建一个新的网桥，并且将这两张网卡接到网桥上

```
$ sudo brctl addbr br0
$ sudo brctl delif virbr0 vnet0
$ sudo brctl delif virbr0 vnet1
$ sudo brctl addif br0 vnet0
$ sudo brctl addif br0 vnet1
$ sudo brctl stp br0 off
$ sudo ifconfig br0 up
```

再查询网桥信息，即可发现网桥已经搭好，如图 14所示。

```
root@yuri-Inspiron-7590:/home/yuri# brctl show
bridge name      bridge id        STP enabled      interfaces
br0               8000.fe5400856398 no                vnet0
                  8000.525400fe36c6 yes               vnet1
virbr0            8000.525400fe36c6 yes               virbr0-nic
```

图 14. 查询网桥信息

之后再在两个虚拟机上分别运行 pktgen-dpdk, l2fwd，即可发现其均能接收包。如图 15所示。

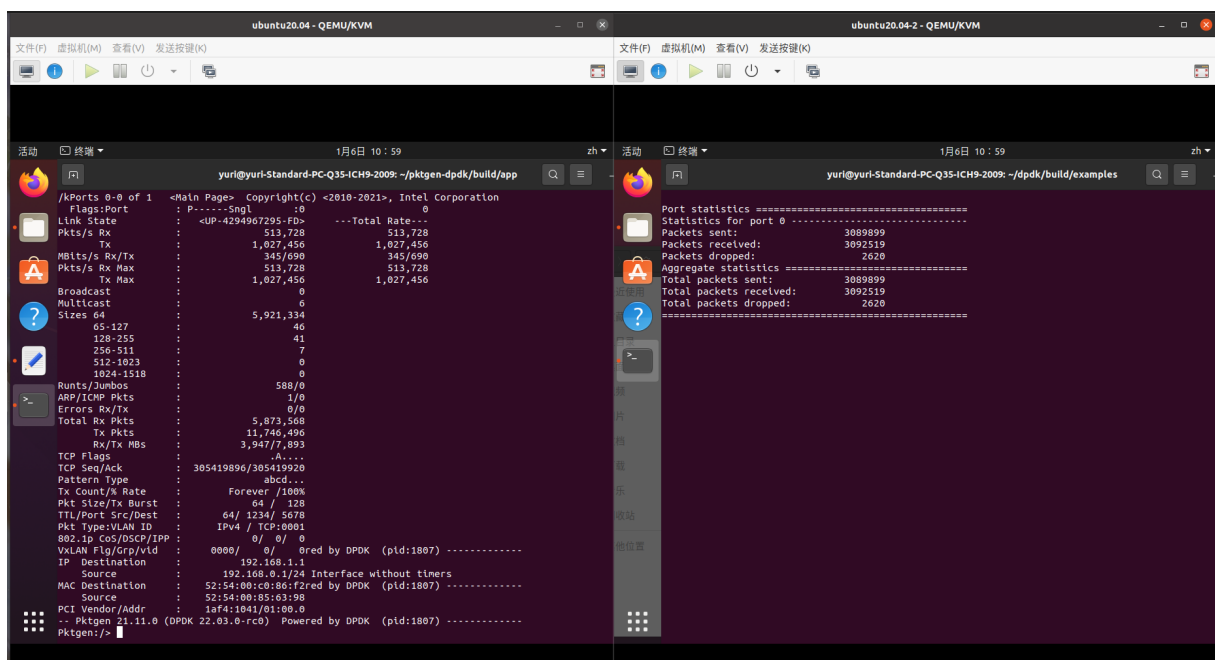


图 15. 同时运行 pktgen-dpdk 与 l2fwd

3 实验结果与分析

由于实验中参数过多，无法将每一个参数变化的所有情况均进行测量。本次实验中采取控制变量法，将一些变量固定，探究其他变量对丢包率的影响。

3.1 数据包大小和数量

在实验中，设定内核数量为 1，队列数量为 2，以测试丢包率与数据包的关系。其中数据包大小从 32bytes 到 1K，数据包数量从 4000-16000，实验结果如下

发包数量	数据包大小 (单位 byte)	丢包率 (%)
4k	32	46.73
	64	34
	128	37.62
	256	39.47
	512	14.34
	1024	5.02
8k	32	56.1
	64	44.48
	128	55.21
	256	39.18
	512	20.22
	1024	4.82
16k	32	56.3
	64	45.81
	128	63.83
	256	39.85
	512	24.11
	1024	5.77

经过可视化后结果如图 16所示。根据数据可以发现以下两个特点

1. 随着数据包大小逐步增大，丢包率会逐步减小
2. 随着数据包数量逐步增大，丢包率会逐步增大。但是这个变化在数据包大小很大的时候不明显，只有在数据包较小时候稍微有一些差距。

3.2 l2fwd 内核数量和队列数量的影响

在实验中，控制数据包的数量为 8k 个，数据包的大小为 64bytes，改变队列和内核数量，实验结果如下

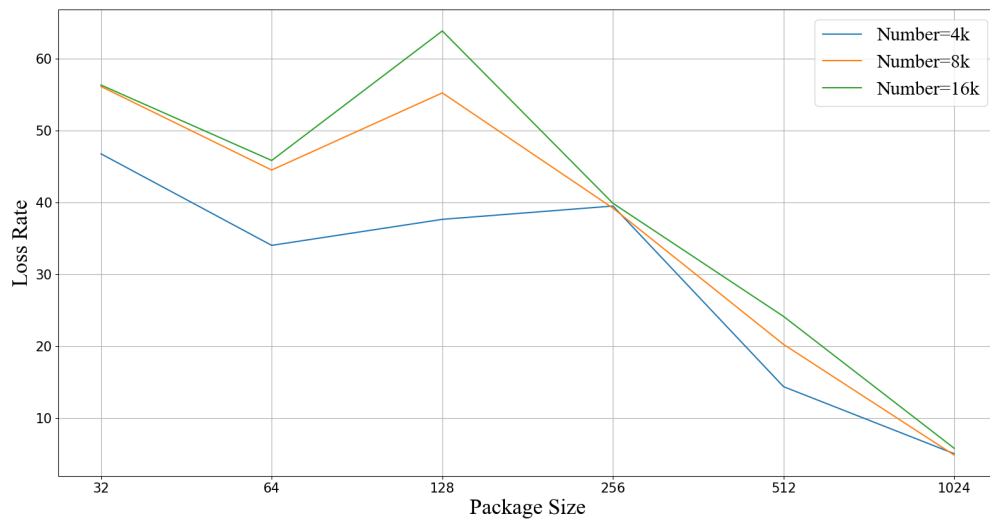


图 16. 丢包率和数据包大小数量关系

内核数量	队列数量	丢包率
1	1	41.99
	2	36.43
	4	25.68
2	1	45.96
	2	40.86
	4	35.8
4	1	47.99
	2	46.88
	4	42.35

经过可视化后结果如图 17 所示。根据数据可以发现以下两个特点

1. 随着内核数量增加，丢包率会逐步增大
2. 随着队列数量增加，丢包率会逐步减小

3.3 结果分析

首先根据测试原理，是由 pktgen 先发包，l2fwd 接受再转发，最后回到 pktgen，由 pktgen 测试丢包率。其中丢包发生在从 pktgen 发包到 l2fwd 中。l2fwd 有个接受的存储队列，队列里面有存包的缓存，而且缓存数量和大小有限，当缓存满时，l2fwd 无法来得及存储到达的数据包，导致丢包的发生。

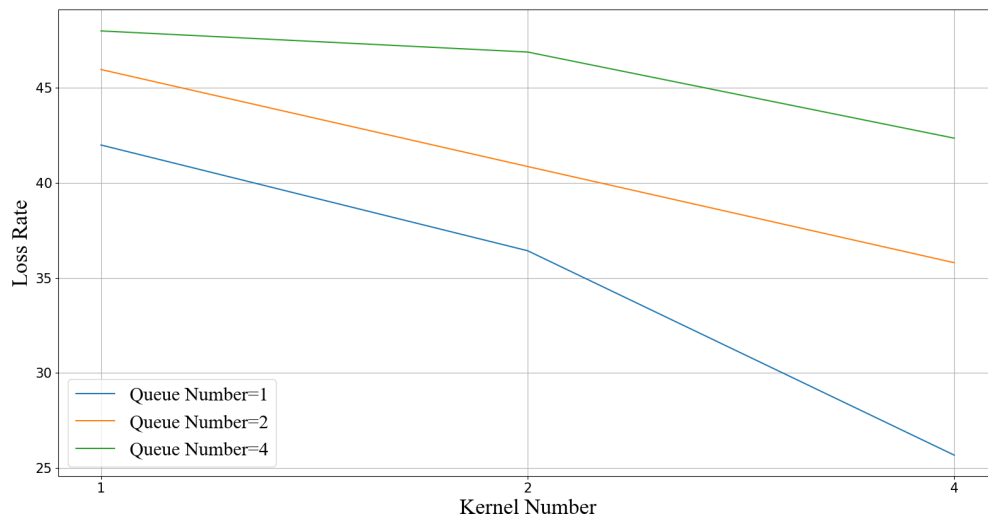


图 17. 丢包率和内核、队列数量关系

根据以上分析可知，可以得到丢包率的一个关系

$$Loss_Rate \sim V_s - V_r - C \quad (1)$$

其中 V_s 为发送包的速率， V_r 为接受包的速率， C 为缓存的大小。反观实验结果，可以发现丢包率与 4 个参数关系为

1. 内核数量越大，会导致发包的处理器数量越多，发包速率越快，从而 V_s 越大，丢包率越大。
2. 队列数量越大，会使得缓存大小越大，而且收包速率越快，从而 V_r, C 越大，丢包率越小。
3. 数据包大小越大，会使得发包处理器处理每个包的时间越多，发包速率越低，从而 V_s 越小，丢包率越大。
4. 数据包数量越多，会使得发包速率越大，收包速率也越大。但数据包很小时，由于处理时间很快，发包的速率有冗余，所以使得发包速率增加更多，从而导致丢包率增大，而在数据包很大时候就不太明显。

4 总结与感想

在此次大作业中，本人选择测试 DPDK performance test 这一项，通过实验深入了解了 DPDK 的工作原理和特性，对计算机网络转发虚拟化这一块有了更深入的了解。

这次项目是本学期最后一个项目，也是融汇贯通的一个项目。回顾之前项目，从为电脑安装 Windows, Ubuntu 双系统，到安装 qemu 虚拟机，到之后虚拟机性能测试，开启巨页，每一次项目都能为本项目作为指导，本人也从完全不了解这一块慢慢变得能熟

练使用 qemu 以及相关工具，在课程中受益匪浅。回顾这个学期工科创 III-D 这门课，本人学到了很多真正有用的知识，也提升了自己动手实践能力，怀着十分感恩的心。

首先感谢管老师以及管海兵，宋涛老师开设这门课，把我带入计算机虚拟化这一块全新的领域，让我对这一块有了很深的感受 and 了解。

再次感谢课程的助教们，每当我在实验过程中遇到麻烦时候，他们给我提供了帮助，使得失去耐心的我重拾对这么课的兴趣，继续向未知的领域探索，从而活动很多有用的知识和经验。

最后感谢帮助我的同学们，和他们的探讨以及帮助极大的助力我学习理解这门课，也减少了我遇到的很多困难。

我相信，云计算与虚拟化在未来必将发展越来越好，而这门课不但带领我走进了这一块领域，而且为我以后在这方面解决问题提供了宝贵的知识和经验。

5 参考内容

- 1、相关具体资料文件参见”<https://github.com/yuri-you/EI313>”
- 2、https://blog.csdn.net/weixin_44602409/article/details/114894700
- 3、https://blog.csdn.net/jhxifeng/article/details/45101805?_t_t_t=0.36389559168839747
- 4、https://blog.csdn.net/weixin_38927796/article/details/72763282