用TestPMD测试DPDK性能和功能

原创 DPDK开源社区 2017-05-19

作者 Pablo.D

本文介绍了数据平面开发工具包（DPDK）TestPMD应用程序，展示了如何构建和配置TestPMD, 以及如何用它来检查使用DPDK的不同网络设备的性能和功能。

TestPMD是一个使用DPDK软件包分发的参考应用程序。其主要目的是在网络接口的以太网端口之间转发数据包。此外，用户还可以用TestPMD尝试一些不同驱动程序的功能，例如RSS，过滤器和英特尔®以太网流量控制器（Intel® Ethernet Flow Director）。

我们还将研究TestPMD运行时的命令行，命令行可用于配置端口之间的数据包转发和网络接口支持的其他功能。TestPMD应用程序适用于所有版本的DPDK。

TestPMD的配置示例

为了展示如何使用TestPMD，我们会考虑两个典型的硬件设置。

如图1所示，第一个配置，TestPMD应用程序把两个以太网口连接到外部的流量发生器。这样用户可以在不同的网络工作负载下测试吞吐量和功能。

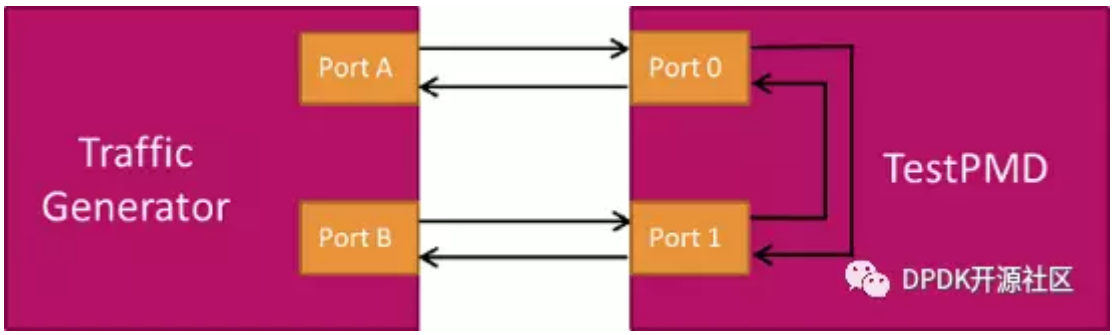

图1. 设置1——使用外部流量发生器

第二个设置，TestPMD应用程序把两个以太网端口连成环回模式。 这样用户可以在没有外部流量发生器的情况下检查网络设备的接收和传输功能。

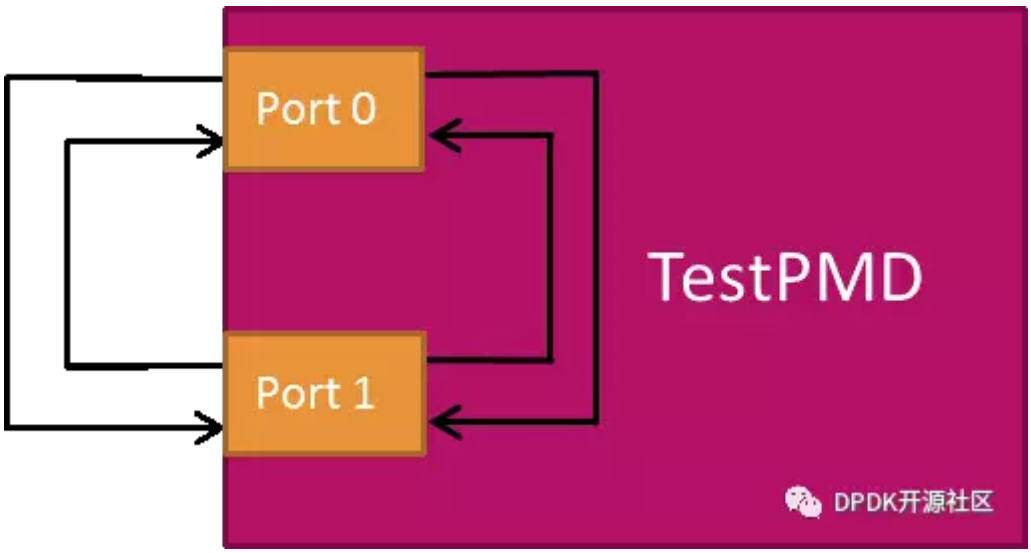

图2.设置2 -环回模式的TestPMD

转发模式

TestPMD可以使用如下几种不同的转发模式。

输入/输出模式（Input/output mode）：此模式通常称为IO模式，是最常用的转发模式，也是TestPMD启动时的默认模式。 在IO模式下，CPU内核从一个端口接收数据包（Rx），并将其发送到另一个端口（Tx）。 如果需要的话，一个端口可同时用于接收和发送。

收包模式（Rx-only mode）：在此模式下，应用程序会轮询Rx端口的数据包，然后直接释放而不发送。 它以这种方式充当数据包接收器。

发包模式（Tx-only mode）：在此模式下，应用程序生成64字节的IP数据包，并从Tx端口发送出去。 它不接收数据包，仅作为数据包源。

后两种模式（收包模式和发包模式）对于单独检查收包或者发包非常有用。

除了这三种模式，TestPMD文档中还介绍了其他一些转发模式。

编译、准备TestPMD

以下步骤用于编译和设置TestPMD应用程序：

1.如下命令从源目录中编译DPDK，默认也编译了TestPMD应用程序：

```
$ make config T = x86_64-native-linuxapp-gcc
```

2.初始化内核模块uio：

```
$ sudo modprobe uio
```

3.加载内核模块igb_uio：

```
$ sudo insmod ./build/kmod/igb_uio.ko
```

4.预留大页内存以供DPDK TestPMD 应用程序使用，最简单的方法是通过使用DPDK附带的dpdk-setup.sh脚本工具（更多信息请参阅DPDK Getting Started Guide）：

```
$ sudo ./usertools/dpdk-setup.sh
```

5.将网络接口端口绑定到igb_uio。举例来说，我们假设使用的端口PCI地址为0000：83：00.1和0000：87：00.1：(可左右滑动↓)

```
$ sudo ./usertools/dpdk-devbind.py -b igb_uio 0000：83：00.1 0000：87：00.1
```

运行TestPMD

TestPMD可以使用一系列命令行参数在非交互模式下运行，也可以使用-i选项运行在交互模式，来实时接收命令行。实时命令行可以动态配置TestPMD：(可左右滑动↓)

```
$ sudo ./build/app/testpmd –l 12,13,14 –n 4 -- -i
```

在此例中， –l选项指定了逻辑核。核12用于管理命令行，核13和14将用于转发数据包。 -n选项用于指定系统的内存通道数。--（破折号）分开了EAL参数和应用程序参数。程序运行时可以看到如下所示的输出：(可左右滑动↓)

```
$ sudo ./build/app/testpmd –l 12,13,14 –n 4 -- -i

EAL: Detected 40 lcore(s)
EAL: Probing VFIO support...
EAL: PCI device 0000:83:00.0 on NUMA socket 1
EAL:   probe driver: 8086:10fb net_ixgbe
EAL: PCI device 0000:83:00.1 on NUMA socket 1
EAL:   probe driver: 8086:10fb net_ixgbe
EAL: PCI device 0000:87:00.0 on NUMA socket 1
EAL:   probe driver: 8086:10fb net_ixgbe
EAL: PCI device 0000:87:00.1 on NUMA socket 1
EAL:   probe driver: 8086:10fb net_ixgbe
Interactive-mode selected
USER1: create a new mbuf pool <mbuf_pool_socket_0>:
      n=163456, size=2176, socket=0
Configuring Port 0 (socket 0)
Port 0: 00:1B:21:B3:44:51
Configuring Port 1 (socket 0)
Port 1: 00:1B:21:57:EE:71
Checking link statuses...
Port 0 Link Up - speed 10000 Mbps - full-duplex
Port 1 Link Up - speed 10000 Mbps - full-duplex
Done
testpmd>
```

testpmd>提示符允许用户输入命令，这被称为实时命令行。例如，我们可以用它来输入命令，来检查转发配置：(可左右滑动↓)

```
testpmd> show config fwd
io packet forwarding - ports=2 - cores=1 - streams=2
  - NUMA support disabled, MP over anonymous pages disabled
Logical Core 13 (socket 1) forwards packets on 2 streams:
  RX P=0/Q=0 (socket 0) -> TX P=1/Q=0 (socket 0) peer=02:00:00:00:00:01
  RX P=1/Q=0 (socket 0) -> TX P=0/Q=0 (socket 0) peer=02:00:00:00:00:00
```

这表明TestPMD正使用前面介绍过的默认io转发模式，同时也表明核13（第二个启用的内核）将轮询端口0上的数据包，然后转发到端口1，反之亦然。 核12，也就是命令行中第一个核正用于处理运实时命令行本身。

要开始转发，只需键入命令’ start’ ：

```
testpmd> start
```

然后，要检查端口之间是否有包正在转发，执行以下命令来显示应用程序正在使用的所有端口的统计信息：(可左右滑动↓)

```
testpmd> show port stats all
###################### NIC statistics for port 0 ######################
RX-packets: 8480274     RX-missed: 0          RX-bytes: 508816632
RX-errors: 0
RX-nombuf: 0
TX-packets: 5763344     TX-errors: 0          TX-bytes: 345800320
Throughput (since last show)
Rx-pps: 1488117
Tx-pps: 1488116
############################################################################
###################### NIC statistics for port 1 ######################
RX-packets: 5763454     RX-missed: 0          RX-bytes: 345807432
RX-errors: 0
RX-nombuf: 0
TX-packets: 8480551     TX-errors: 0          TX-bytes: 508832612
Throughput (since last show)
Rx-pps: 1488085
Tx-pps: 1488084
############################################################################
```

此输出显示了应用程序开始转发后的所有数据包总数，包含有这两个端口接收和发送的数据包数。吞吐率是以数据包每秒来显示的。在这个例子中，所有端口上接收到的包都以理论线速14.88Mpps往外转

发。线速是指给定数据包大小和网络接口的最大速度。若要停止转发，只需输入 stop，这会停止转发并显示两个端口的累计统计数字以及一个概要。(可左右滑动↓)

```
testpmd> stop
Telling cores to stop...
Waiting for lcores to finish...
---------------------- Forward statistics for port 0 ----------------------
RX-packets: 136718750      RX-dropped: 0         RX-total: 136718750
TX-packets: 136718750      TX-dropped: 0         TX-total: 136718750
----------------------------------------------------------------------------
---------------------- Forward statistics for port 1 ----------------------
RX-packets: 136718750      RX-dropped: 0         RX-total: 136718750
TX-packets: 136718750      TX-dropped: 0         TX-total: 136718750
----------------------------------------------------------------------------
++++++++++Accumulated forward statistics for all ports +++++++++
RX-packets: 273437500      RX-dropped: 0         RX-total: 273437500
TX-packets: 273437500      TX-dropped: 0         TX-total: 273437500
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

使用多核

对于一个核不足以转发所有收到的包的情况，多核可以用于处理来自不同端口的数据包。在前面的例子中，核 13和14可用于转发数据包，但只有核 13被用到了。要启用另一个核，我们可以使用以下命令：(可左右滑动↓)

```
testpmd> set nbcore 2

testpmd> show config fwd

io packet forwarding - ports=2 - cores=2 - streams=2
  - NUMA support disabled, MP over anonymous pages disabled
Logical Core 13 (socket 1) forwards packets on 1 streams:
  RX P=0/Q=0 (socket 0) -> TX P=1/Q=0 (socket 0) peer=02:00:00:00:00:01
Logical Core 14 (socket 1) forwards packets on 1 streams:
  RX P=1/Q=0 (socket 0) -> TX P=0/Q=0 (socket 0) peer=02:00:00:00:00:00
```

这样核13将从端口0接收数据包，并从端口1发送数据包，而core14将接收来自端口1的数据包，并从端口0上发送。

改变转发模式

如上所述，TestPMD具有不同的转发模式。若要将转发模式更改为收包模式，我们可以使用set fwd命令：

```
testpmd> set fwd rxonly
 testpmd> start
```

现在，如果我们看端口统计数据，可以看到只有接收到的数据包才会显示。由于没有发送的数据包，Tx数据仍然为0：(可左右滑动↓)

```
testpmd> show port stats all
######################## NIC statistics for port 0  ############################
RX-packets: 524182888  RX-missed: 0        RX-bytes:  31450974816
RX-errors:  0
RX-nombuf:  0
TX-packets: 0          TX-errors: 0        TX-bytes:  0

Throughput (since last show)
Rx-pps:     14880770
Tx-pps:           0
############################################################################
######################## NIC statistics for port 1  ############################
RX-packets: 486924876  RX-missed: 0        RX-bytes:  29215494352
RX-errors:  0
RX-nombuf:  0
TX-packets: 0          TX-errors: 0        TX-bytes:  0

Throughput (since last show)
Rx-pps:     14880788
Tx-pps:           0
############################################################################
```

在TestPMD获得帮助

TestPMD为运行时可用的命令提供在线帮助。这些帮助分为几个部分，可以通过帮助命令获取。

```
testpmd> help
```

以下部分可用“帮助”：(可左右滑动↓)

```
help control    : Start and stop forwarding.
help display    : Displaying port, stats and config information.
help config     : Configuration information.
```

```
help ports      : Configuring ports.
help registers  : Reading and setting port registers.
help filters    : Filters configuration help.
help all        : All of the above sections.
```

例如，要获取有关显示数据和其他信息的命令的帮助：(可左右滑动↓)

```
testpmd> help display
Display:
--------

show port (info|stats|xstats|fdir|stat_qmap|dcb_tc|cap) (port_id|all)
    Display information for port_id, or all.

show port X rss reta (size) (mask0,mask1,...)
    Display the rss redirection table entry indicated by masks on port X. size is used to indicat

show port rss-hash ipv4|ipv4-frag|ipv4-tcp|ipv4-udp|ipv4-sctp|ipv4-other|ipv6|ipv6-frag|ipv6
    Display the RSS hash functions and RSS hash key of port X

clear port (info|stats|xstats|fdir|stat_qmap) (port_id|all)
    Clear information for port_id, or all.

show (rxq|txq) info (port_id) (queue_id)
    Display information for configured RX/TX queue.

show config (rxtx|cores|fwd|txpkts)
    Display the given configuration.

read rxd (port_id) (queue_id) (rxd_id)
    Display an RX descriptor of a port RX queue.

read txd (port_id) (queue_id) (txd_id)
    Display a TX descriptor of a port TX queue.
```

结论

本文研究了如何编译、设置和运行TestPMD以及如何通过实时命令行进行配置。

（* 本文翻译自Intel Developer Zone，可拉至文末“阅读原文”浏览原文 ）

作者简介

Pablo de Lara Guarch是英特尔网络软件工程师，主要关注DPDK的数据平面函数和库的开发，其贡献包括哈希算法增强和新的加解密设备驱动。此外，他还维护了DPDK crypto subtree。

DPDK开源社区往期精选文章

- FD.IO/VPP和DPDK Cryptodev，会产生什么样的化学反应？
- VFD大揭秘，一定有你想知道的！
- cryptodev 概述、状态及未来的工作——DPDK用户空间大会
- 从PCRE到Hyperscan
- Generic Flow API简介
- 关于DPDK Cryptodev，你不得不明白的几点！
- 玩物志 | 什么！DPDK在盒子里？
- 无锁队列详细分解 — 顶层设计
- VMware Player 搭建DPDK实验平台
- Qemu/Kvm 搭建DPDK实验平台
- 技术贴：利用DPDK加速容器网络
- DPDK IP分片与重组设计实现
- DPDK流分类优化，不得不知道的事~

— 长按扫描二维码关注我们 —