DPDK 技术白皮书

V1.0

中国电信股份有限公司广州研究院

2015 年 10 月

目 录

1 概述

1.1 问题背景

ETSI NFV 技术通过运行在通用 x86 架构硬件上的虚拟化网络功能，为电信运营商和互联网服务商提供了一种灵活的业务部署手段和高效的组网方案，可以支持固移网络和 IDC 中 NAT、DPI、EPC、FW 等各类业务功能的广域灵活部署与网元弹性扩展。

不同于典型数据中心业务和企业网业务，电信广域网业务要求网元（如 BNG、DPI 等）具有高吞吐、低时延、海量流表支持、用户级 QoS 控制的特点。大量实践表明，通用 x86 服务器作为 NFV 基础设施用于高转发业务时，面临着严重的转发性能瓶颈，需要有针对性地从硬件架构、系统 I/O、操作系统、虚拟化层、组网与流量调度、VNF 功能等层面进行性能优化，才能达到各类 NFV 网络业务的高性能转发要求。

根据 ETSI 的 NFV 参考架构，现实中的 NFV 应用系统一般由 NFV 基础设施和 VNF 两类系统服务商提供。因此，相应的 NFV 端到端性能测试，也应划分为底层的 NFV 基础设施性能与上层的 VNF 性能两类，以明确各自的性能瓶颈，并避免性能调优工作相互干扰。

在各类 NFV 基础设施性能优化技术方案中，DPDK（Data Plane Development Kit）类软件加速方案已成为一种普遍采用的基本方法，它以用户数据 I/O 通道优化为基础，结合了 Intel VT 技术、操作系统、虚拟化层与 vSwitch 等多种优化方案，已经形成了完善的性能加速整体架构，并提供了用户态 API 供高速转发类应用访问。

在 DPDK 技术的应用中，大多数 NFV 应用与开发单位一般关注如下几类问题：

1） DPDK 应用于不同硬件、软件环境下的基本性能

2） DPDK 转发性能调优方法、性能影响因素

3） DPDK 调优参考配置

4） DPDK 应用开发的稳定性、可移植性

针对这些问题，本文基于 DPDK 的提供 NFV 转发性能调优的最佳实践和测试结果，以有助于业界同仁开展 NFV 系统的研发与性能评估工作。

1.2 范围

本白皮书主要集中分析基于DPDK的单节点物理主机与虚拟机环境的I/O性能优化方法，以及相关性能指标与测试方法。有关多节点NFV服务器、端到端NFV基础设施、VNF相关性能的集成测试方法，请参见ETS NFV GS-PER-001。

另外，我们使用DPDK网站提供的L2/L3转发样例程序作为性能测试的环回工具，具体的VNF性能调优方法不在本文讨论范围。

本文将提供如下内容：

- DPDK技术的原理、应用场景概述
- 通过介绍部分实验性结果，分析DPDK转发技术的性能影响因素
- DPDK测试方法、测试基准、不同软硬件配置下的DPDK性能测试结果
- 根据我们的最佳实践经验，在附录中提供不同用例的硬件、OS、虚拟化层、DPDK配置模板

本文后续将根据应用要求不断完善，持续更新。后续可能的工作包括：

- 海量流表相关的DPDK转发性能测试
- 高负载压力测试
- 基于OVS的VM内部环回性能测试
- 部分VNF性能测试，以及跨HOST端到端性能测试

2 DPDK 技术简介

DPDK 技术框架可以划分为 DPDK 基本技术与 DPDK 优化技术两部分，前者指标准的 DPDK 数据平面开发包和 I/O 转发实现技术，本章将概述该部分的主要原理；后者是在 DPDK 应用过程中，为进一步提高各类用户应用程序的转发性能，所采取的性能调优方法和关键配置，这部分将在第 3 章介绍。

2.1 技术原理与架构

由于采用软件转发和软件交换技术，单服务器内部的转发能力是 NFV 系统的主要性能瓶颈。在各类高速转发的 NFV 应用中，数据报文从网卡中接收，再传送到虚拟化的用户态应用程序（VNF）处理，整个过程要经历 CPU 中断处理、虚拟化 I/O 与地址映射转换、虚拟交换层、网络协议栈、内核上下文切换、内存拷贝等多个费时的 CPU 操作和 I/O 处理环节。

业内通常采用消除海量中断、旁路内核协议栈、减少内存拷贝、CPU 多核任务分担、Intel VT 等技术来综合提升服务器数据平面的报文处理性能，普通用户较难掌握。业界迫切需要一种综合的性能优化方案，同时提供良好的用户开发和商业集成环境，DPDK 加速技术方案成为其中的典型代表。

DPDK 是一个开源的数据平面开发工具集，提供了一个用户空间下的高效数据包处理库函数，它通过环境抽象层旁路内核协议栈、轮询模式的报文无中断收发、优化内存/缓冲区/队列管理、基于网卡多队列和流识别的负载均衡等多项技术，实现了在 x86 处理器架构下的高性能报文转发能力，用户可以在 Linux 用户态空间开发各类高速转发应用，也适合与各类商业化的数据平面加速解决方案进行集成。

英特尔在 2010 年启动了对 DPDK 技术的开源化进程，于当年 9 月通过 BSD 开源许可协议正式发布源代码软件包，并于 2014 年 4 月在 www.dpdk.org 上正式成立了独立的开源社区平台，为开发者提供支持。开源社区的参与者们大幅推进了 DPDK 的技术创新和快速演进，而今它已发展成为 SDN 和 NFV 的一项关键技术。

2.2 软件架构

DPDK 的组成架构如图 2-1 所示，相关技术原理概述如下：

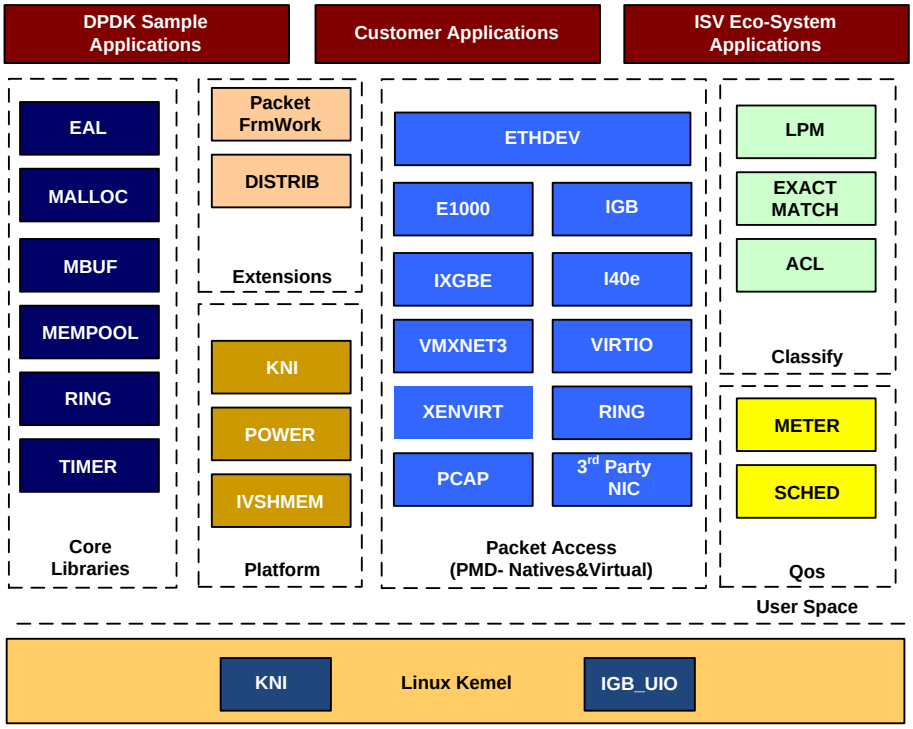

图 2－1 DPDK 组成结构

在图 2-1 中，在最底部的内核态（Linux Kernel）DPDK 有两个模块：KNI 与 IGB_UIO。其中，KNI 提供给用户一个使用 Linux 内核态的协议栈，以及传统的 Linux 网络工具（如 ethtool, ifconfig）。IGB_UIO（igb_uio.ko 和 kni.ko. IGB_UIO）则借助了 UIO 技术，在初始化过程中将网卡硬件寄存器映射到用户态。

如图 2-1，DPDK 的上层用户态由很多库组成，主要包括核心部件库（Core Libraries）、平台相关模块(Platform)、网卡轮询模式驱动模块（PMD-Natives&Virtual）、QoS 库、报文转发分类算法（Classify）等几大类，用户应用程序可以使用这些库进行二次开发，下面分别简要介绍。

核心部件库：该模块构成的运行环境是建立在 Linux 上，通过环境抽象层(EAL)的运行环境进行初始化，包括：HugePage 内存分配、内存/缓冲区/队列分配与无锁操作、CPU 亲和性绑定等；其次，EAL 实现了对操作系统内核与底层网卡 I/O 操作的屏蔽（I/O 旁路了内核及其协议栈），为 DPDK 应用程序提供了一组调用接口，通过 UIO 或 VFIO 技术将 PCI 设备地址映射到用户空间，方便了应用程序调用，避免了网络协议栈和内核切换造成的处理延迟。另外，核心部件还包括创建适合报文处理的内存池、缓冲区分配管理、内存拷贝、以及定时器、环形缓冲区管理等。

平台相关模块：其内部模块主要包括 KNI、能耗管理以及 IVSHMEM 接口。其中，KNI 模块主要通过 kni.ko 模块将数据报文从用户态传递给内核态协议栈处理，以便用户进程使用

传统的 socket 接口对相关报文进行处理；能耗管理则提供了一些 API，应用程序可以根据收包速率动态调整处理器频率或进入处理器的不同休眠状态；另外，IVSHMEM 模块提供了虚拟机与虚拟机之间，或者虚拟机与主机之间的零拷贝共享内存机制，当 DPDK 程序运行时，IVSHMEM 模块会调用核心部件库 API，把几个 HugePage 映射为一个 IVSHMEM 设备池，并通过参数传递给 QEMU，这样，就实现了虚拟机之间的零拷贝内存共享。

轮询模式驱动模块：PMD 相关 API 实现了在轮询方式下进行网卡报文收发，避免了常规报文处理方法中因采用中断方式造成的响应延迟，极大提升了网卡收发性能。此外，该模块还同时支持物理和虚拟化两种网络接口，从仅仅支持 Intel 网卡，发展到支持 Cisco、Broadcom、Mellanox、Chelsio 等整个行业生态系统,以及基于 KVM、VMWARE、 XEN 等虚拟化网络接口的支持。

DPDK 还定义了大量 API 来抽象数据平面的转发应用，如 ACL、QoS、流分类和负载均衡等。并且，除以太网接口外，DPDK 还在定义用于加解密的软硬件加速接口（Extensions）。

总体而言，DPDK 技术具有如下特征：

- 采用 BSD License，保证了可合法用于商业产品
- 支持 RedHat、CentOS、Fedora、Ubuntu 等大多数 Linux 系统，已开始进入主流 Linux 发布版本
- DPDK 支持 Run to Completion 和 Pipeline 两种报文处理模式，用户可以依据需求灵活选择，或者混合使用。Run to Completion 是一种水平调度方式，利用网卡的多队列，将报文分发给多个 CPU 核处理，每个核均独立处理到达该队列的报文，资源分配相对固定，减少了报文在核间的传递开销，可以随着核的数目灵活扩展处理能力；Pipeline 模式则通过共享环在核间传递数据报文或消息，将系统处理任务分解到不同的 CPU 核上处理，通过任务分发来减少处理等待时延。
- DPDK 的库函数和样例程序十分丰富，包括 L2/L3 转发、Hash、 ACL、QoS、环形队列等大量示例供用户参考，具体可访问：

  http://dpdk.org/doc/guides/sample_app_ug/index.html

2.3 大页技术

处理器的内存管理包含两个概念：物理内存和虚拟内存。Linux 操作系统里面整个物理内存按帧（frames）来进行管理，虚拟内存按照页（page）来进行管理。内存管理单元（MMU）

完成从虚拟内存地址到物理内存地址的转换。内存管理单元进行地址转换需要的信息保存在一个叫页表（page table）的数据结构里面，页表查找是一种极其耗时的操作。为了减少页表的查找过程，Intel 处理器实现了一块缓存来保存查找结果，这块缓存被称为 TLB（Translation Lookaside Buffer），它保存了虚拟地址到物理地址的映射关系。所有虚拟地址在转换为物理地址以前，处理器会首先在 TLB 中查找是否已经存在有效的映射关系，如果没有发现有效的映射，也就是 TLS miss，处理器再进行页表的查找。页表的查找过程对性能影响极大，因此需要尽量减少 TLB miss 的发生。

x86 处理器硬件在缺省配置下，页的大小是 4K，但也可以支持更大的页表尺寸，例如 2M 或 1G 的页表。使用了大页表功能后，一个 TLB 表项可以指向更大的内存区域，这样可以大幅减少 TLB miss 的发生。早期的 Linux 并没有利用 x86 硬件提供的大页表功能，仅在 Linux 内核 2.6.33 以后的版本，应用软件才可以使用大页表功能，具体的介绍可以参见 Linux 的大页表文件系统（hugetlbfs）特性。

DPDK 则利用大页技术，所有的内存都是从 HugePage 里分配，实现对内存池(mempool)的管理，并预先分配好同样大小的 mbuf，供每一个数据包使用。

2.4 轮询技术

传统网卡的报文接收/发送过程中，网卡硬件收到网络报文，或发送完网络报文后，需要发送中断到 CPU，通知应用软件有网络报文需要处理。在 x86 处理器上，一次中断处理需要将处理器的状态寄存器保存到堆栈，并运行中断服务程序，最后再将保存的状态寄存器信息从堆栈中恢复。整个过程需要至少 300 个处理器时钟周期。对于高性能网络处理应用，频繁的中断处理开销极大降低了网络应用程序的性能。

为了减少中断处理开销，DPDK 使用了轮询技术来处理网络报文。网卡收到报文后，直接将报文保存到处理器 cache 中（有 DDIO（Direct Data I/O）技术的情况下），或者保存到内存中（没有 DDIO 技术的情况下），并设置报文到达的标志位。应用软件则周期性地轮询报文到达的标志位，检测是否有新报文需要处理。整个过程中完全没有中断处理过程，因此应用程序的网络报文处理能力得以极大提升。

2.5 CPU 亲和技术

现代操作系统都是基于分时调用方式来实现任务调度，多个进程或线程在多核处理器的

某一个核上不断地交替执行。每次切换过程，都需要将处理器的状态寄存器保存在堆栈中，并恢复当前进程的状态信息，这对系统其实是一种处理开销。将一个线程固定一个核上运行，可以消除切换带来的额外开销。另外将进程或者线程迁移到多核处理器的其它核上进行运行时，处理器缓存中的数据也需要进行清除，导致处理器缓存的利用效果降低。

CPU 亲和技术，就是将某个进程或者线程绑定到特定的一个或者多个核上执行，而不被迁移到其它核上运行，这样就保证了专用程序的性能。

DPDK 使用了 Linux pthread 库，在系统中把相应的线程和 CPU 进行亲和性绑定，然后相应的线程尽可能使用独立的资源进行相关的数据处理。

3 DPDK 性能影响因素

本章介绍基于 DPDK 进行应用开发和环境配置时，应用程序性能的影响因素以及相应的优化调整方法。这些因素并非必然劣化性能，可能因硬件能力、OS 版本、各类软硬环境参数配置等的差异产生较大波动，或者存在较大的不稳定性，相关的调优方法需要用户结合自身的 VNF 应用部署在实践中不断完善。

3.1 硬件结构的影响

DPDK 具有广泛的平台适应性，可以运行在整个 x86 平台，从主流服务器平台（从高性能或者高能效产品系列），到桌面或者嵌入式平台，也可以运行于基于 Power 或者其他架构的运算平台。图 3-1 展示了一个通用双路服务器的内部架构，它包含了 2 个中央处理器，2 个分离的内存控制单元来连接系统内存，芯片组会扩展出大量高速的 PCIe 2.0/3.0 接口，用于连接外设，如 10Gbps 或者 25Gbps 网卡外设。

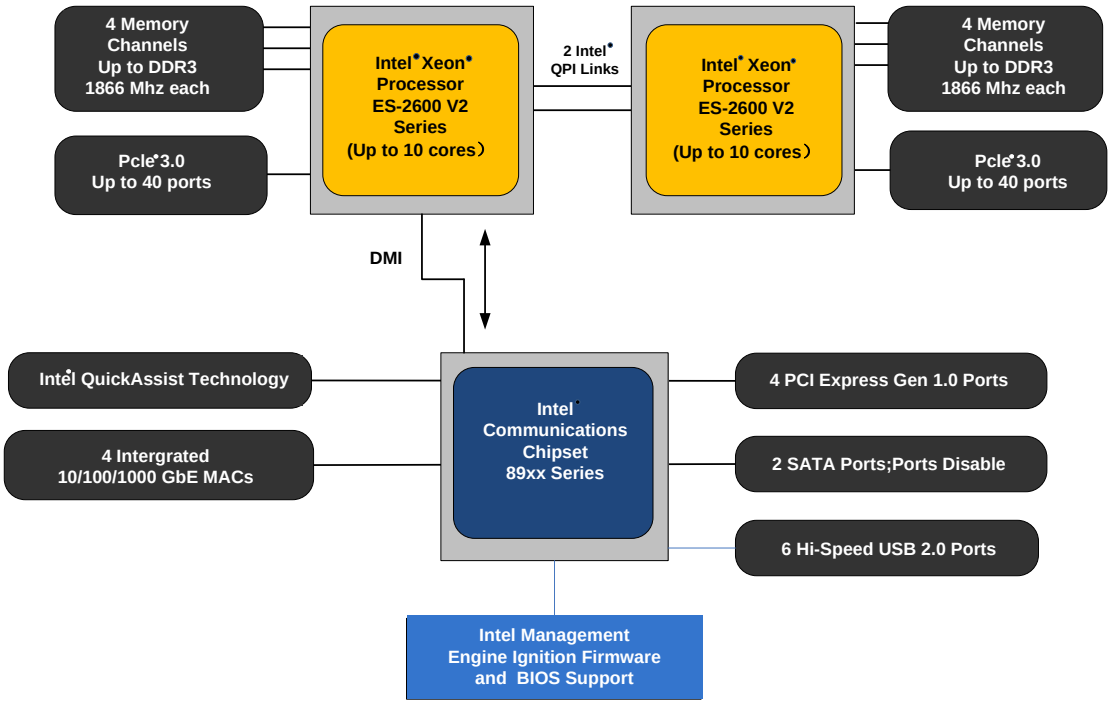

图 3－1 通用双路服务器内部结构

硬件规格对 DPDK 性能的影响体现在几个方面：

- CPU 频率：CPU 频率越高，DPDK 性能越高
- LLC(last leve cache)大小：缓存越大，DPDK 性能越高

- PCIe Lane 的数目：PCIe 链路可以支持 1、2、4、8、12、16 和 32 个 Lane，即×1、×2、×4、×8、×12、×16 和×32 宽度的 PCIe 链路。需要确保其带宽可以满足所插网卡的带宽。
- NUMA：网络数据报文的处理由网卡所在的 NUMA 节点处理，将会比远端 NUMA 节点处理的性能更高。

3.2 OS 版本及其内核的影响

不同的 Linux OS 发行版使用的 Linux 内核版本不一样，配置的 Linux OS 服务也不一样。这些差异都会导致应用程序在网络报文处理能力上有所差别。

由于 Linux 内核还在不断演进，Linux 的发行版也数量众多，本文无法提供最佳 Linux 内核版本和配置，而只能给出部分参考建议，如：关闭部分 OS 服务。在后续章节，我们选取了目前比较流行的 Linux 发行版进行了测试，并给出测试结果。从测试结果中显示，同样的硬件配置环境下，不同的 Linux 发行版在小包的处理能力上存在差异。

3.2.1 关闭 OS 部分服务

在 10G 端口上 64Byte 报文的线速到达率为 14.88Mpps，如果实现零丢包的线速处理，每个报文的平均处理速度应该小于 1/14.48Mpps=67us，DPDK 应用的网卡收发包队列长度缺省配置为 128，网卡队列填充满的时间是 128*67=8579us。也就是说，DPDK 的业务进程，如果被操作系统中断超过 8579us，就会导致网卡收发报的队列被充满，无法接收新的报文，从而导致丢包。而操作系统的中断服务程序、后台服务程序、以及任务调度程序都会导致 DPDK 的应用程序被打断。

在 NFV 应用场景下，Host 机器上和 Guest 机器上都运行着操作系统，由此带来了两级的操作任务调度。DPDK 应用程序（NFV 中的 VNF）均运行在虚拟机上，操作系统带来的影响会更加明显。

为了实现 DPDK 应用程序的零丢包，需要对操作系统进行适当的配置，减少操作系统的对 DPDK 应用程序的干扰。操作系统配置的总体思路是：

- 将运行 DPDK 应用运行的处理器核进行隔离，减少干扰；
- 停用不需要的后台服务程序。将不需要的中断，转移到其它处理器核上处理；
- 对于不能转移的中断，减少中断的次数。

3.2.2 OS 调整示例

下面以 CentOS 7 为例，说明具体的调整方法。绝大部分的操作需要同时在 Host 操作系统和 Guest 操作系统同时应用。

1) 对 DPDK 应用使用处理器核进行隔离

修改 Linux 的 OS 的 GRUB 参数，

设置 isolCPUs=16-23,40-47

2) 打开运行有 DPDK 进程的处理器核上的 nohz_full。nohz_full 可以减少内核的周期性时钟中断的次数。

修改 Linux 的 OS 的 GRUB 参数，

设置 nohz_full=16-23,40-47

3) 关闭 NMI 监控功能，减少 NMI 中断对 DPDK 任务的干扰。

修改 Linux 的 OS 的 GRUB 参数，

设置 nmi_watchdog=0

4) 关闭 SELinux 功能

修改 Linux 的 OS 的 GRUB 参数，

设置 selinux=0

5) 关闭处理器的 P 状态调整和 softlockup 功能。将处理器锁定在固定的频率运行，减少处理器在不同的 P 状态切换带来的处理时延。

修改 Linux 的 OS 的 GRUB 参数，

设置 intel_pstate=disable nosoftlockup

6) 关闭图形显示，减少 GUI 应用的干扰

调用命令 systemctl set-default multi-user.target

7) 关闭操作系统的中断调度程序

调用命令 systemctl disable irqbalance.service

8) 关闭操作系统的审计服务

调用命令 systemctl disable auditd.service

9) 关闭蓝牙服务

调用命令 systemctl disable bluetooth.service

10) 关闭 KVM 的内存页合并服务

调用命令 systemctl disable ksm.service

调用命令 systemctl disable ksmtuned.service

11) 对于 KVM 虚拟的 vCPU 和物理 CPU 进行绑定

使用 QEMU monitor 获取 vCPU 对应的线程号，使用 taskset 命令进行绑定

```
[root@mayancity_centos realtime-virtual-host]# telnet 127.0.0.1 4444
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
QEMU 2.3.0 monitor - type 'help' for more information
 (qemu) info CPUs
```

```
* CPU #0: pc=0xffffffff812eaf26 thread_id=24622
CPU #1: pc=0xffffffff81052dd6 (halted) thread_id=24623
CPU #2: pc=0xffffffff81052dd6 (halted) thread_id=24624
CPU #3: pc=0xffffffff81052dd6 (halted) thread_id=24625
[root@mayancity_centos realtime-virtual-host]# taskset -pc 16 24622
[root@mayancity_centos realtime-virtual-host]# taskset -pc 17 24623
[root@mayancity_centos realtime-virtual-host]# taskset -pc 18 24624
[root@mayancity_centos realtime-virtual-host]# taskset -pc 19 24625
```

12) 在丢包率要求较高的场景下，增加 DPDK 应用程序的 RX/TX 队列的长度。建议将队列长度调整为 2048

```
#define RTE_TEST_RX_DESC_DEFAULT 2048
#define RTE_TEST_TX_DESC_DEFAULT 2048
```

3.3 OVS 性能问题

OVS 作为 NFV 的一个重要组成模块，会运行在绝大多数的服务器节点上，提供虚拟机和虚拟机之间，以及虚拟网络和物理网络间的互连接口，其性能至关重要。OVS 2.4 开始正式支持 DPDK 加速，相比传统基于 Linux 内核的 OVS 版本，转发性能提高了数倍，为 VNF 在通用 x86 服务器上部署提供了有力支持。

OVS 缺省会为每一个 NUMA 节点创建一个 pmd 线程，该 pmd 线程以轮询方式处理属于其 NUMA 节点上的所有 DPDK 接口。为了高性能，需要利用前面提到的 CPU 亲和技术，把 pmd 线程绑定在一个固定的 CPU core 上处理。此外，为了增加扩展性，OVS 2.4 也支持网卡多队列以及多 pmd 线程数，这些参数均可动态配置，但具体配置应根据具体需求来决定。

3.4 内存管理

如前所述，DPDK 考虑了 NUMA 以及多内存通道的访问效率，会在系统运行前要求配置 Linux 的 HugePage，初始化时申请其内存池，用于 DPDK 运行的主要内存资源。Linux 大页机制利用了处理器核上的的 TLB 的 HugePage 表项，这可以减少内存地址转换的开销。

3.4.1 内存多通道的使用

现代的内存控制器都支持内存多通道，比如 Intel 的 E5-2600V3 系列处理器，能支持 4 个通道，可以同时读和取数据。依赖于内存控制器及其配置，内存分布在这些通道上。每一

个通道都有一个带宽上限，如果所有的内存访问都只发生在第一个通道上，这将成为一个潜在的性能瓶颈。

因此，DPDK 的 mempool 库缺省是把所有的对象分配在不同的内存通道上，保证了在系统极端情况下需要大量内存访问时，尽可能地将内存访问任务均匀平滑。

3.4.2 内存拷贝

很多 libc 的 API 都没有考虑性能，因此，不要在高性能数据平面上用 libc 提供的 API，比如，memcpy()或 strcpy()。虽然 DPDK 也用了很多 libc 的 API，但均只是在软件配置方面用于方便程序移植和开发。

DPDK 提供了一个优化版本的 rte_memcpy() API，它充分利用了 Intel 的 SIMD 指令集，也考虑了数据的对齐特性和 cache 操作友好性。

3.4.3 内存分配

在某些情况下，应用程序使用 libc 提供的动态内存分配机制是必要的，如 malloc()函数，它是一种灵活的内存分配和释放方式。但是，因为管理零散的堆内存代价昂贵，并且这种内存分配器对于并行的请求分配性能不佳，所以不建议在数据平面处理上使用类似 malloc()的函数进行内存分配。

在有动态分配的需求下，建议使用 DPDK 提供的 rte_malloc() API，该 API 可以在后台保证从本 NUMA 节点内存的 HugePage 里分配内存，并实现 cache line 对齐以及无锁方式访问对象等功能。

3.4.4 NUMA 考虑

NUMA(Non Uniform Memory Access Architecture)与 SMP(Symmetric Multi Processing)是两种典型的处理器对内存的访问架构。随着处理器进入多核时代，对于内存吞吐量和延迟性能有了更高的要求，NUMA 架构已广泛用于最新的英特尔处理器中，为每个处理器提供分离的内存和内存控制器，以避免 SMP 架构中多个处理器同时访问同一个存储器产生的性能损失。

在双路服务器平台上，NUMA 架构存在本地内存与远端内存的差异。本地和远端是个相对概念，是指内存相对于具体运行程序的处理器而言，如图 3-2 所示。

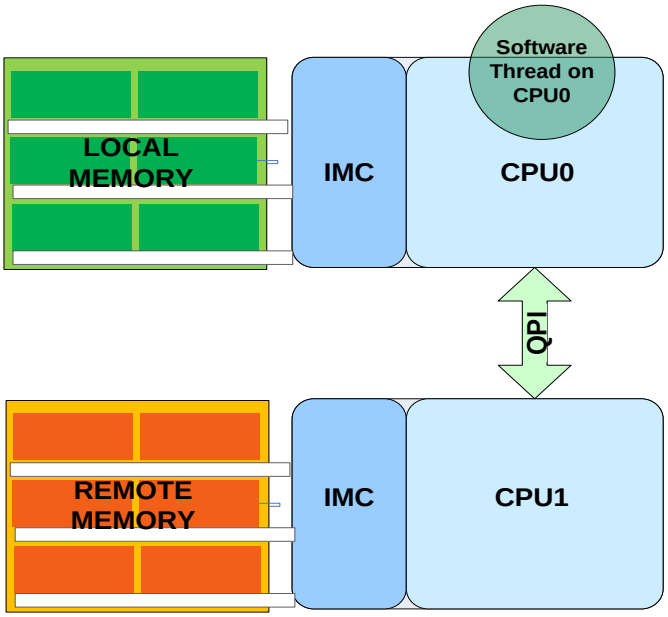

图 3－2 NUMA 本地/远程内存示意

在 NUMA 体系架构中，CPU 进行内存访问时，本地内存的要比访问远端内存更快。因为访问远端内存时，需要跨越 QPI 总线，在应用软件设计中应该尽量避免。在高速报文处理中，这个访问延迟会大幅降低系统性能。尤其是传统嵌入式软件向服务器平台迁移时，需要特别关注。

DPDK 提供了一套在指定 NUMA 节点上创建 memzone、ring, rte_malloc 以及 mempool 的 API，可以避免远端内存访问这类问题。在一个 NUMA 节点端，对于内存变量进行读取不会存在性能问题，因为该变量会在 CPU cache 里。但对于跨 NUMA 架构的内存变量读取，会存在性能问题，可以采取复制一份该变量的副本到本地节点（内存）的方法来提高性能。

3.5 CPU 核间无锁通信

如果想建立一个基于消息传递的核间通信机制，可以使用 DPDK ring API，它是一个无锁的 ring 实现。该 ring 机制支持批量和突发访问，即使同时读取几个对象，也只需要一个昂贵的原子操作，批量访问可以极大地改善性能。

下面是一段伪代码：

```
#define MAX_BULK 32

while (1) {
        /* Process as many elements as can be dequeued. */
```

```
count = rte_ring_dequeue_burst(ring, obj_table, MAX_BULK);
if (unlikely(count == 0))
continue;

my_process_bulk(obj_table, count);
}
```

3.6 设置正确的目标 CPU 类型

DPDK 支持 CPU 微架构级别的优化，可以通过修改 DPDK 配置文件中的 CONFIG_RTE_MACHINE 参数来定义。优化的程度根据随编译器的能力而不同，通常建议采用最新的编译器进行编译。如果编译器的版本不支持该款 CPU 的特性，比如 Intel AVX 指令，那么它在编译时只会选用自己支持的指令集，这可能导致编译 后生成的 DPDK 应用的性能下降。

4 DPDK 在 NFV 中的应用

DPDK 技术在 NFV 环境中主要用于对 x86 设备的数据面转发性能进行优化，以提升 VNF 或特定网络业务处理功能的整体吞吐量。因此，针对 VNF 的部署位置（Host 或 VM），DPDK 主要应用于物理机 I/O 性能优化、基于 OVS 的 VM 内部 I/O 性能优化、基于 SR-IOV 的 VM 内部 I/O 性能优化三种场景。

4.1 VNF 在物理机上应用

运营商现有网络大部分设备都是专用网络设备，物理设备与应用软件紧耦合，设备升级成本高、功能扩展困难。将这些专用的网络功能设备，以软件化的 VNF 形式直接运行在物理服务器上，可以实现网络设备形态的通用化，方便设备功能灵活扩展。

4.1.1 应用场景

该方案将一部分原来由硬件实现的网络功能，以 VNF 软件的形式直接运行在 x86 服务器 OS 上，同时在物理服务器上加载 DPDK 组件。此时，DPDK 接管了物理网卡的 I/O 驱动，VNF 也不再使用传统 Linux 内核网络协议栈，而是通过调用 DPDK 的用户态 API 进行快速转发。同时，DPDK 进程将使用少量的处理器核（如 2 个核）与内存以满足高速转发处理，VNF 可以直接使用剩余的全部硬件资源，适用于 DPI 和 FW 等资源利用率长期较高的网络业务。尽管该方案下整台服务器仅支持单一 VNF 应用，但因设备形态统一，软件功能部署灵活，仍具有较高应用价值。具体实现架构如图 4-1 所示。

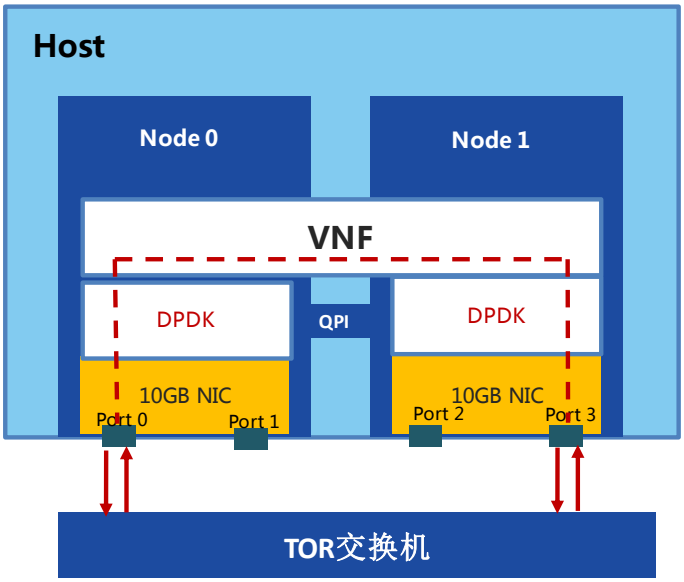

图 4－1 DPDK 在 NFV 物理服务器中的应用示意

4.1.2 讨论

该场景因将物理服务器作为资源池最小的资源分配单位，应用中容易出现以下问题：

（1）单个 VNF 占用的资源粒度过大，系统空闲时设备资源浪费严重。因此，该场景适用于 DPI 等长期资源饱和型网络业务。

（2）在多路 CPU 的服务器上，由于单一 VNF 占用全部的网卡和处理器核，容易导致跨 NUMA 节点的报文处理，引起转发性能下降。采用高性价比的多核单 CPU 服务器可以避免跨路带来的性能影响。

（3）在用户态的 VNF 中，原有基于传统网络协议栈的数据平面处理操作，需要移植为调用 DPDK API，并配合大量多核分布式编程方法与 DPDK 提供的资源使用方法，以满足高性能转发要求。对于必须依赖系统协议栈工作的应用，需要增加额外的用户态协议栈支持，否则不能采用 DPDK 进行优化。

4.2 VNF+OVS 应用

当多个 VNF 分别运行在一台服务器的多个 VM 中时，为满足 VNF 之间可能的流量交换或者共享物理网卡的需要，可以在 Host OS 上安装 OVS 类虚拟交换机，用于连接各个 VNF（VM）和服务器的物理网卡端口。此时，这多个 VNF 一般是不同类型的 VNF 应用，VNF 之间可能产生交互流量或者业务链处理流量，而同类型多个 VNF 或者纯粹共享网卡的多个 VNF，一般采

用 4.3 节的“VNF + SR-IOV”方案。

4.2.1 应用场景

在“VNF+OVS”方案中，因主要的性能瓶颈存在于 VNF 的虚拟 I/O 通道和 OVS 交换机，DPDK 需要分别安装在运行 VNF 的 VM 镜像内部和运行 OVS 的物理服务器 OS 上：前者用于优化 VM 内部的 VNF 数据平面转发性能，包括提供虚拟化网卡驱动、提供用户态转发 API 等，DPDK 的各种配置方法与 VNF 运行在物理机中的机制类似，VNF 并不能感知是运行在 VM 环境；后者用于优化 OVS 交换机性能，连接 VM 与各 NUMA 节点上的 DPDK 端口。该场景可以细分为两种：

（1）VM 到 OVS 仅使用单一连接。

（2）VM 到 OVS 采用一进一出的双向连接，如各类业务链中的 VNF 应用，见图 4-2。

该方案可以实现 VNF 的灵活扩容/缩容，以及在资源池中按需迁移，方案中的第三方 VNF 厂商可以屏蔽物理设备差异，提供各自的高性能业务产品。同时，使用经过 DPDK 优化后的 OVS 交换机，可以灵活实现 VNF 间流量的灵活转发与互联互通，节省硬件交换机。

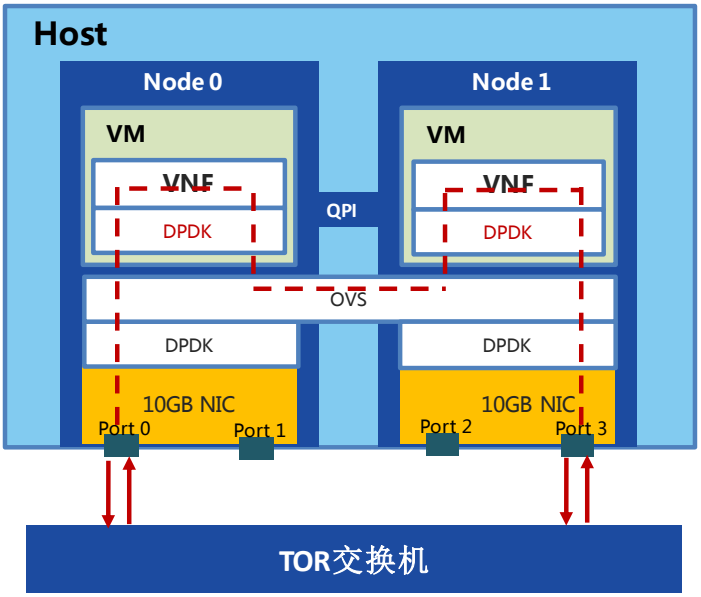

图 4－2 DPDK 在 VNF+OVS 环境下的应用示意

4.2.2 讨论

由于 VNF 本身的多样化和 VNF 产品存在多类提供商，以及 VNF 和 OVS 在基于 DPDK 进行

高速转发时需要消耗 CPU 和内存资源，该方案在实际应用中可能存在如下问题：

（1）不同厂商的 VNF 使用 DPDK 的配置参数可能存在较大差异，如 Hugpe 尺寸、缓冲区大小、CPU 亲和性设置等，应尽量规范或一定程度上限制对 CPU、内存和虚拟网络资源的随意使用，避免某个 VNF 在单台服务器内部占用过多硬件资源。

（2）VNF 内部的 DPDK 配置和 OVS 下的 DPDK 运行环境需要消耗一定数量的 CPU 核和内存，从目前经验看，10G 线速转发情况下，每个 VNF 的 DPDK 进程至少将占用 1 个逻辑核，而安装 OVS 的情况下则需要 2-4 个核，加之普通 VNF 本身至少使用 2－4 核，因此，单服务器内部的 VNF 数量受到较大限制。实际应用中，应统一调配和规范 DPDK 对资源的占用和对系统运行环境的配置，提供参考配置。

（3）该场景下加载 DPDK 的配置过程较为复杂，各类物理/虚拟资源的管理和监控困难。实际应用中，需要考虑自动化配置方案，并且将各类 VNF 以及 OVS 对资源的占用和端口设置状况，向 VIM 层和 NFVO 层进行汇总。

根据上述问题分析，对物理服务器的 CPU 数量、内存数量、VNF 镜像的建议如下：

（1）CPU 数量要求：至少 2 路 CPU；每 CPU 6 核以上

（2）内存数量要求：根据应用级别不同，可分为 n*8G 字节（其中 n=8, 9……K）

（3）VNF 镜像要求：如果涉及到虚拟机部署，则此参数要求为：vCPU 至少 4 核，内存根据应用级别分为 n*8G 字节（其中 n=1, 2, 3……K）。

4.3 VNF + SR-IOV

当多个 VNF 运行在 VM 中时，各 VM 的虚拟网卡可以直接连至 HOST 上支持 SR-IOV 功能的物理网卡（VF）进行数据收发，如同独占物理网卡一样。该方案适用于 VNF 间无需流量交互的场景，或者是基于硬件交换机进行 VNF 互连和流量控制的场景。由于旁路了 HOST 的虚拟化层实现直接转发，可以达到近似物理转发的性能，被业界普遍用于消除 Hypervisor 带来的数据转发性能影响。

4.3.1 应用场景

尽管“VNF+SR-IOV”方案消除了从物理网卡到 VM 虚拟网卡的性能瓶颈，但 VM 内部仍然需要通过加载 DPDK 以进一步优化各 VNF（VM 内部）的转发性能。

此时，DPDK 可以采用与 4.1 和 4.2 小节类似的方法进行加载，同时占用 VM 内部一定 CPU

核和内存资源。

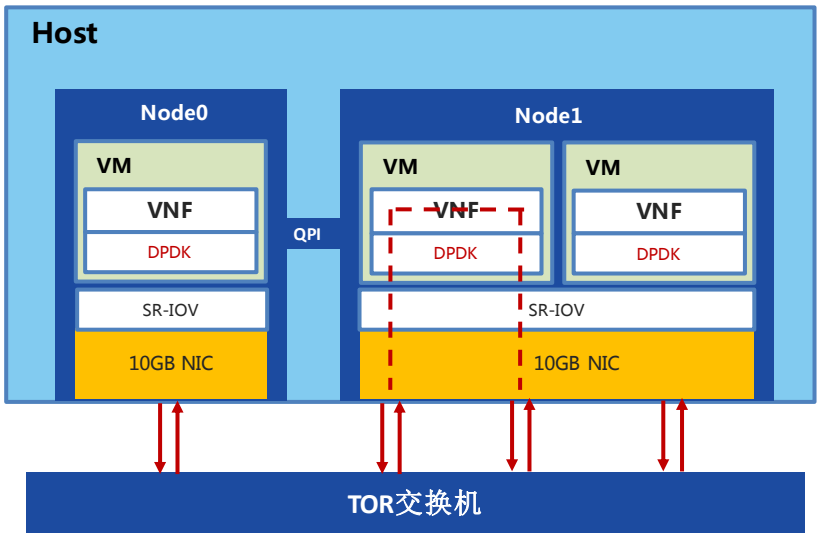

图 4－3 DPDK 在 VNF+SR-IOV 环境下的应用示意

4.3.2 讨论

该方案以 VM 为粒度进行 VNF 资源调度，系统扩容缩容方便灵活，但也存在如下问题：

（1）同 4.2 小节的方案一样，由于 VM 内部的 DPDK 将独占一定的硬件资源以满足线速转发要求，VNF 数量较多时，物理服务器会有较多 CPU 和内存被用于 DPDK 运行环境，成为 VNF 的一种隐性成本开销。

（2）方案不支持在 HOST 内部进行 VNF 流量交互，如果需要支持业务链等业务场景，需要提供基于网卡或者硬件交换机的 VNF 互连技术。

（3）需要规范各类 VNF 对 DPDK 资源的使用和监控，提供 VF 优先级和流量负载均衡方案，同样需要 VIM 层和 NFVO 层参与全局资源的优化调度。

5 DPDK 性能专项测试

5.1 测试网络拓扑

DPDK 测试主要分为纯物理机转发、SR-IOV 与 OVS 三类场景进行测试，详细测试架构见图 5-1。每个场景又分为同路、CPU 跨路和网卡跨路三种情况，三种情况的端口连接方法见图 5-2。三种情况下的流量流向描述如下：

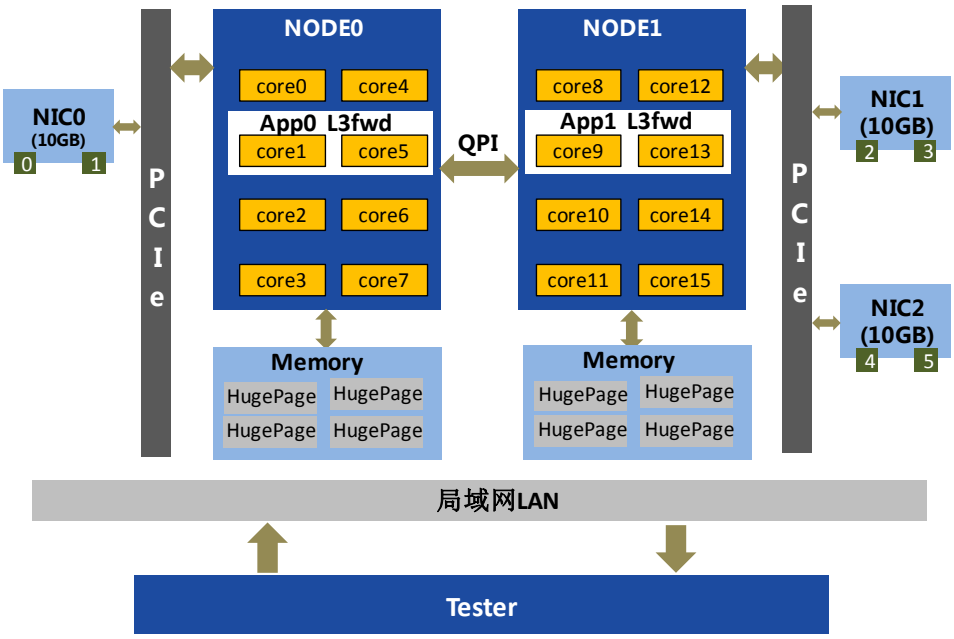

图 5－1 测试架构

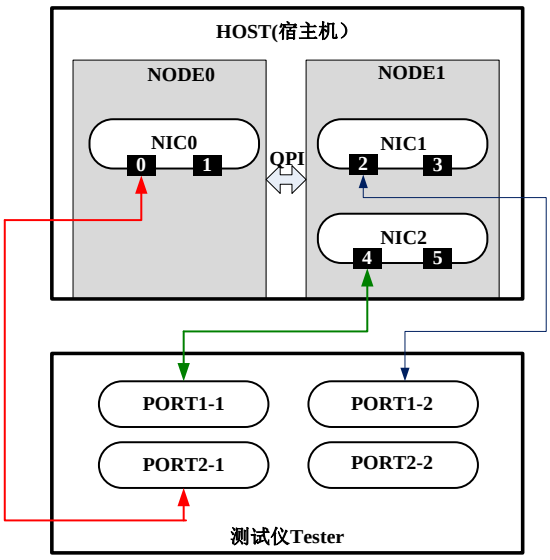

图 5－2 端口连接方法

1) 同路测试情况：数据流从测试仪表 port1-2，通过 NIC1 的端口 2，进入 NUMA NODE1，经过转发 App1，再从 NIC2 的端口 4，回到测试仪表 port1-1；
2) CPU 跨路情况：数据流从测试仪表 port1-2，通过 NIC1 的端口 2，先进入 NUMA NODE1，经过 QPI 进入运行在 NUMA NODE0 上的转发 App0（同时绑定 NIC1 的端口 2 和 NIC2 的端口 4），再经过 QPI 回到 NUMA NODE1，最后从 NIC2 的端口 4，回到测试仪表 port1-1；
3) 网卡跨路情况：数据流从测试仪表 port1-2，通过 NIC1 的端口 2，进入运行在 NUMA NODE1 上的转发 App1（同时绑定 NIC1 的端口 2 和 NIC0 的端口 0），经过 QPI 进入 NUMA NODE0，最后从 NIC0 的端口 0，回到测试仪表 port2-1。

5.2 测试标准

5.2.1 性能指标

主要对以下几个网络性能指标进行测试：

1) 负载（OfferedLoad）：网卡流量负荷百分比，当前流量占端口速率的比例（百分比）
2) 丢包率（Percent Loss）：数据包的丢失比例（百分比）
3) 包速（Frame Per-second）：数据包每秒的收/发个数
4) 最大时延（Max Latency）：数据包传输的最大延迟（微秒，us）
5) 最小时延（Min Latency）：数据包传输的最小延迟（微秒，us）
6) 平均时延（Average Latency）：数据包传输的平均延迟（微秒，us）

5.2.2 测试方法

主要采用 RFC2544 测试方法：

1) 测试时长为 60s
2) 测试包长分别为：66、128、256、512 和 1024 字节

注：使用 Spirent 测试仪时，测试仪需要在数据包中加入 20 字节的数据报文标记。若测试时使用 UDP 报文，在没有负载时报文长度为 46 字节，加上 Spirent 的发包标记后以太帧包长为 66 字节。

3) 测试仪表双向各配置 20000 条流，在各测试例中需要修改各转发 App（如 L3fwd）的转发表或 OVS 配置流表，以保证对上述流量的双向转发
4) 测试丢包率为 0 的情况下最大负载、包速及相应时延

5.3 测试平台说明

5.3.1 硬件平台说明

5.3.1.1被测设备

表 5－1 被测设备硬件表

被测设备	DUT-1	DUT-2	DUT-3	DUT-4	DUT-5	DUT-6	DUT-7	DUT-8	DUT-9	DUT-10	DUT-11
CPU 型号	E5-2630v3	E5-2690v3	E5-2630v3	E5-2690v3	E5-2630v3	E5-2630v3	E5-2690v3	E5-2630v3	E5-2690v3	E5-2650v2	E5-2699v3
内存容量	64G DDR4	64G DDR4	64G DDR4	64G DDR4	64G DDR4	64G DDR4	64G DDR4	64G DDR4	64G DDR4	64G DDR4	64G DDR4
硬盘容量	1T	1T	1T	1T	1T	1T	1T	1T	1T	1T	1T
网卡型号	X520-SR2	X520-SR2	X520-SR2	X520-SR2	X520-SR2	X520-SR2	X520-SR2	X520-SR2	X520-SR2	X520-SR2	X520-SR2
网卡数量	2	2	2	2	2	2	2	2	2	2	2

注：X520-SR2 网卡为 2*10G 光口

5.3.1.2测试仪表

表 5－2 测试仪说明表

仪表型号	Spirent TestCenter SPT-2U
软件版本	Spirent TestCenter Application 4.15

5.3.1.3BIOS 配置

表 5－3 各厂家服务器 BIOS 配置信息表

参数名	参数值	DUT -1	DUT -2	DUT -3	DUT -4	DUT -5	DUT -6	DUT -7	DUT -8	DUT -9	DUT -10	DUT -11
电源管理选项												
CPU Power and Performance Policy	Traditional		✓	✓	✓		✓	✓	✓	✓		
Intel Turbo boost	Disable	✓	✓	✓	✓		✓	✓	✓		✓	✓
Processor C3	Disable	✓									✓	✓
Processor C6	Disable	✓									✓	✓
处理器配置												
Intel Hyper-Threading Technology (HTT)	Disable	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
MLC Streamer（ Hardware Prefetcher）	Enable	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
MLC Spatial Prefetcher （ Adjacent Cache Prefetch）	Enable	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
DCU Data Prefetcher（ DCU Streamer Prefetcher）	Enable	✓	✓	✓	✓		✓	✓	✓	✓		✓
DCU nstructionrefetcher（DCU IP Prefetcher）	Enable	✓	✓	✓	✓		✓	✓	✓	✓		✓
虚拟化选项												
Intelirtualization Technology for Directed I/O (VT-d)	Enable	✓	✓	✓	✓			✓	✓		✓	✓
内存配置选项												
Memory RAS and Performance Configuration –> NUMA Optimized	Auto											✓
I/O 配置												
DDIO (Direct Cache Access)	Auto											✓

5.3.2 软件平台说明

5.3.2.1软件版本

表 5－4 测试用例软件版本表

序号	软件	功能	版本/内核
1	CentOS 7.0 X86_64	主机操作系统(Host OS)	3.10.0-123.el7.x86_64
2	CentOS 7.1 X86_64	主机操作系统(Host OS)	3.10.0-229.el7.x86_64
3	CentOS 7.0 X86_64	虚拟机(VM)	3.10.0-123.el7.x86_64
4	CentOS 7.1 X86_64	虚拟机(VM)	3.10.0-229.el7.x86_64
5	Fedora 20	主机操作系统(Host OS)	3.11.10-301.fc20.x86_64
6	Fedora 21	主机操作系统(Host OS)	3.17.4-301.fc21.x86_64
7	Fedora 22	主机操作系统(Host OS)	4.0.4-301.fc22.x86_64
8	Qemu-kvm	KVM管理工具	V2.20
9	Data Plane Development Kit(DPDK)	DPDK套件	DPDK1.8.0 http://www.dpdk.org
10	Open vSwitch(OVS)	软件交换机	V2.3.9 https://github.com/openvswitch/ovs.git(15c69d2d8f52669a57dd49827fe4e585ebdda105)

5.3.2.2操作系统软件配置

表 5－5 操作系统软件配置表

序号	参数 （Parameter）	使能/非使能 （Enable/Disable）	备注 （Explanation）
1	Firewall	Disable	关闭防火墙
2	irqbalance	Disable	关闭中断平衡
3	ssh	Enable	开启ssh，便于远程连接控制
4	auditd	Disable	关闭审计服务
5	kdump	Disable	关闭kdump服务
6	bluetooth	Disable	关闭蓝牙服务
7	NetworkManager	Disable	关闭网络管理服务
8	ksm	Disable	关闭KVM的内存扫描服务
9	ksmtuned	Disable	关闭KVM的内存扫描服务

详细命令参见附录二。

5.4 测试用例介绍

表 5－6 测试例及配置说明表

测试类别	测试用例	测试目的	相关配置
物理机三层转发	测试例-1	不同 Hugepage 数目对物理机三层转发性能影响	Hugepage 设置为 1G、2G、4G 和 6G，CPU Xeon E5-2690 v3,网卡、CPU 与网卡同 socket，操作系统 cento7.0
	测试例-2	CPU 核数对物理机三层转发性能影响	CPU 核数设置为 1 核、2 核，CPU Xeon E5-2690 v3，网卡、CPU 与网卡同 socket，操作系统 cento7.0
	测试例-3	NUMA 对物理机三层转发性能影响	CPU Xeon E5-2630 v3，转发核 2 个，4G Hugepage，操作系统 cento7.0
	测试例-4	不同被测物理机三层转发性能差异	CPU Xeon E5-2630 v3，转发核 2 个，4G Hugepage，网卡、CPU 与网卡同 socket，操作系统 cento7.0
	测试例-5	相同配置下不同 CPU 型号对物理机三层转发性能影响	转发核 2 个，4G Hugepage，CPU 与网卡同 socket，操作系统 cento7.0
	测试例-6	相同配置下不同 Linux 版本对物理机三层转发性能影响	CPU Xeon E5-2690 v3，转发核 2 个，4G Hugepage，CPU 与网卡同 socket
SR-IOV 测试	测试例-7	不同被测物理机 SR-IOV 转发性能差异	CPU Xeon E5-2630 v3、转发核 2 个、4G Hugepage、操作系统 centos7.0、CPU 与网卡同 socket、虚拟机 4 核，6G 内存
	测试例-8	相同配置下不同 CPU 型号对 SR-IOV 转发性能影响	转发核 2 个、4G Hugepage、CPU 与网卡同 socket、虚拟机 4 核，6G 内存、操作系统 centos7.0
	测试例-9	相同配置下不同 Linux kernel 版本对 SR-IOV 转发性能影响	CPU Xeon E5-2690 v3、转发核 2 个、4G Hugepage、CPU 与网卡同 socket、虚拟机 4 核，6G 内存
	测试例-10	相同配置下 NUMA 对 SR-IOV 转发性能影响	CPU Xeon E5-2630 v3、转发核 2 个、4G Hugepage、操作系统 centos7.0、虚拟机 4 核，6G 内存
OVS 测试	测试例-11	不同被测物理机 OVS 转发性能差异	CPU Xeon E5-2630 v3、转发核 2 个、4G Hugepage、CPU 与网卡同 socket，OVS（V2.3.9）
	测试例-12	相同配置下不同 CPU 型号对 OVS 转发性能影响	转发核 2 个、4G Hugepage、CPU 与网卡同 socket、操作系统 centos7.0，OVS（V2.3.9）
	测试例-13	相同配置下不同 Linux 版本对 OVS 转发性能影响	CPU Xeon E5-2630 v3、转发核 2 个、4G Hugepage、CPU 与网卡同 socket，OVS（V2.3.9）
	测试例-14	相同配置下 NUMA 对 OVS 转发性能影响	CPU XEON E5-2630 V3、转发核 2 个、4G Hugepage、操作系统 centos7.0，OVS（V2.3.9）

5.5 专项测试

5.5.1 主要测试流程

所有测试例的核心测试流程描述如下：

1) 按照每个测试例中的“测试场景”和“测试环境配置”内容要求，准备测试环境

2) 连通测试仪表与服务器网络

3) 启动 DPDK 转发

- 对于 DPDK 物理机三层转发，启动方法见《附录 DPDK 启动配置》
- 对于 SR-IOV 测试，启动方法见《SR-IOV 测试配置》
- 对于 OVS 测试，启动方法见《OVS 测试配置》

4) 分配 DPDK 线程 CPU 转发核数，设置 CPU 与网卡跨 NUMA 情况，设置转发内存大小

5) 配置测试仪表，模拟双向各 20000 条流，分别设置包大小为 66、128、256、512、1024 字节

6) 启动测试仪表进行测试，收集测试结果

5.5.2 DPDK 物理机三层转发

5.5.2.1不同 Hugepage 数目对物理机三层转发性能影响测试

测试场景

表 5－7 测试例-1 测试场景说明

测试编号	测试例-1
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2690 v3
HOST 操作系统	centos7.0
DPDK 转发核数	1 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	1G、2G、4G、6G

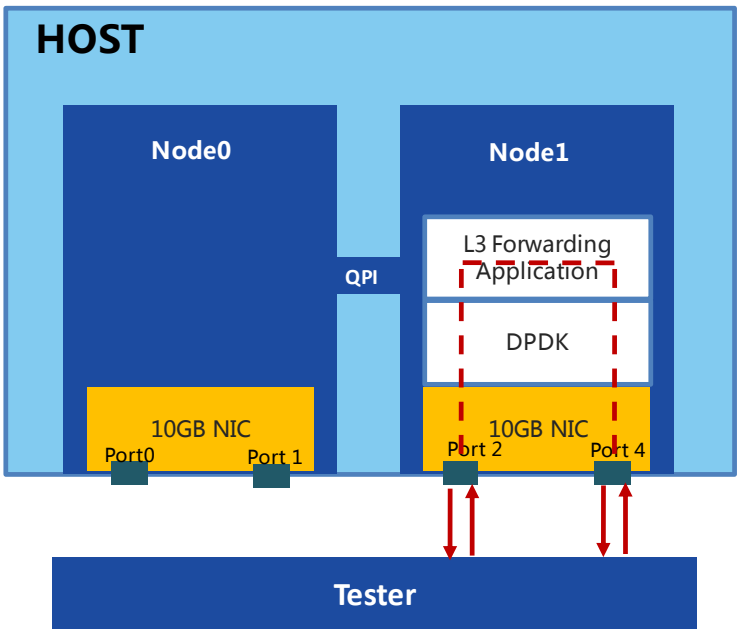

图 5－3 测试例-1 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－8 测试例-1 测试环境配置

服务器配置	DUT-9
服务器 bios 配置	见表 5-5“DUT-9”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 1”
DPDK 编译	见附录四《DPDK 编译》
DPDK 转发程序编译	见附录六《L3fwd 程序编译》
DPDK 软件启动配置	见附录五《DPDK 软件启动配置》

测试结果

表 5－9 测试例-1 吞吐量性能数据

包长	DUT-9（1 核 1G） 负载（%）	DUT-9（1 核 2G） 负载（%）	DUT-9（1 核 4G） 负载（%）	DUT-9（1 核 6G） 负载（%）
66	74.138	74.138	74.138	74.138
128	99.601	99.601	99.601	99.601
256	99.601	99.601	99.601	99.601
512	99.601	99.601	99.601	99.601
1024	99.618	99.618	99.618	99.618

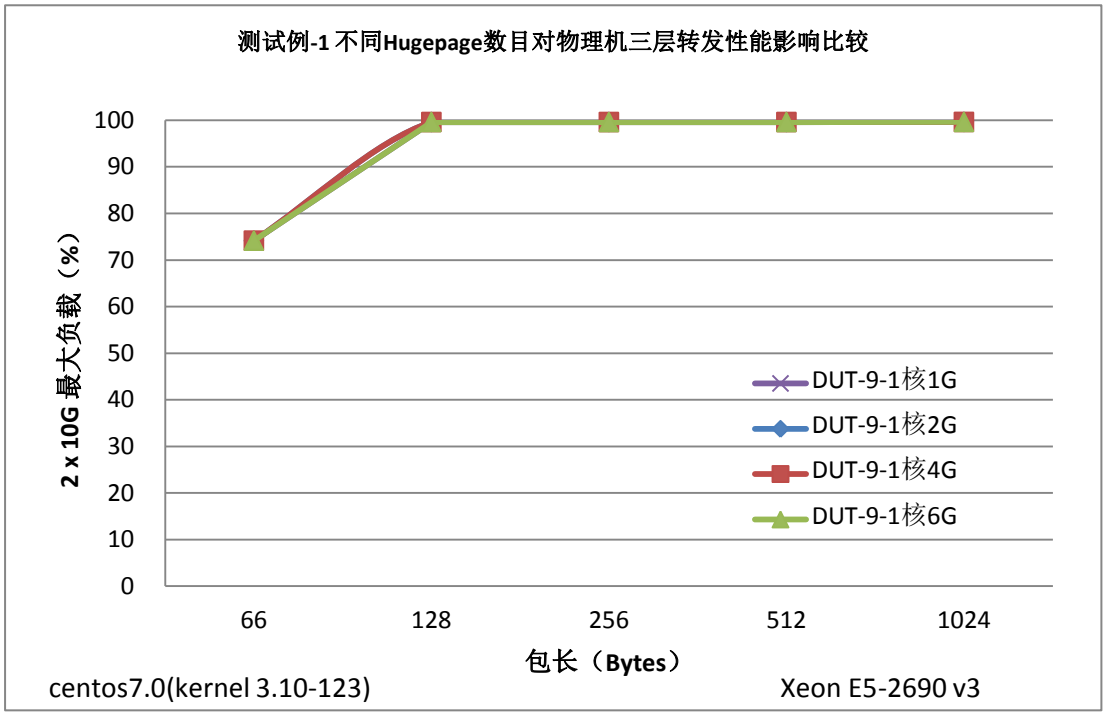

图 5－4 测试例-1 吞吐量性能比较

小结：该测试例的测试结果表明，DPDK 应用转发程序（如 L3fwd）使用的 Hugepage 容量变化并不影响转发性能，使用过多的 Hugepage 并不能进一步提升 DPDK 的转发性能。本白皮书后续测试中将使用 4G Hugepage，作为物理机和虚拟机中 L3fwd 转发程序的运行内存配置。不同 DPDK 应用转发程序的内存大小可由 DPDK 应用程序开发者提供。

5.5.2.2CPU 核数对物理机三层转发性能影响测试

测试场景

表 5－10 测试例-2 测试场景说明

测试编号	测试例-2
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2690 v3
HOST 操作系统	centos7.0
DPDK 转发核数	1 核、2 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	4G

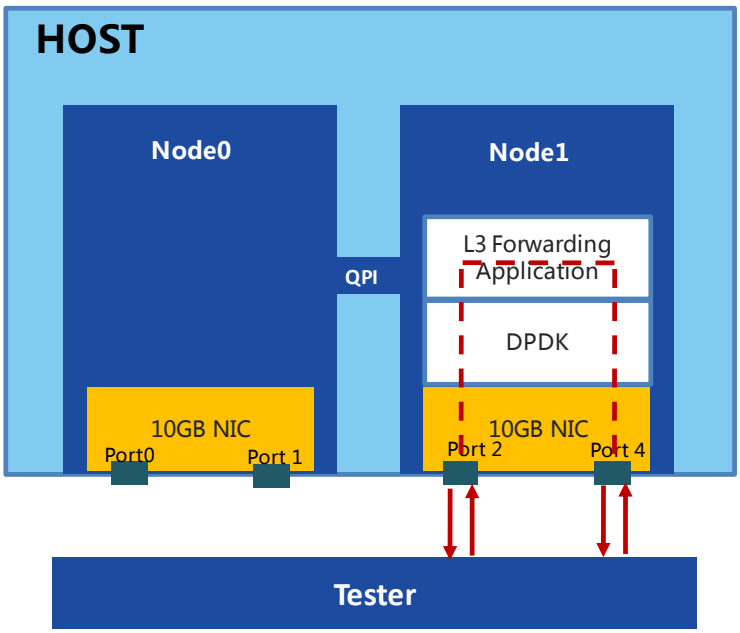

图 5－5 测试例-2 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－11 测试例-2 测试环境配置

服务器配置	DUT-9
服务器 bios 配置	见表 5-5“DUT-9”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 1”
Dpdk 编译	见附录四《DPDK 编译》
DPDK 转发程序编译	见附录六《L3fwd 程序编译》
DPDK 软件配置	见附录五《DPDK 软件启动配置》

测试结果

表 5－12 测试例-2 吞吐量性能数据

包长	DUT-9-1 核 4G 负载（%）	DUT-9-2 核 4G 负载（%）
66	74.138	93.478
128	99.601	99.601
256	99.601	99.601
512	99.601	99.601
1024	99.618	99.618

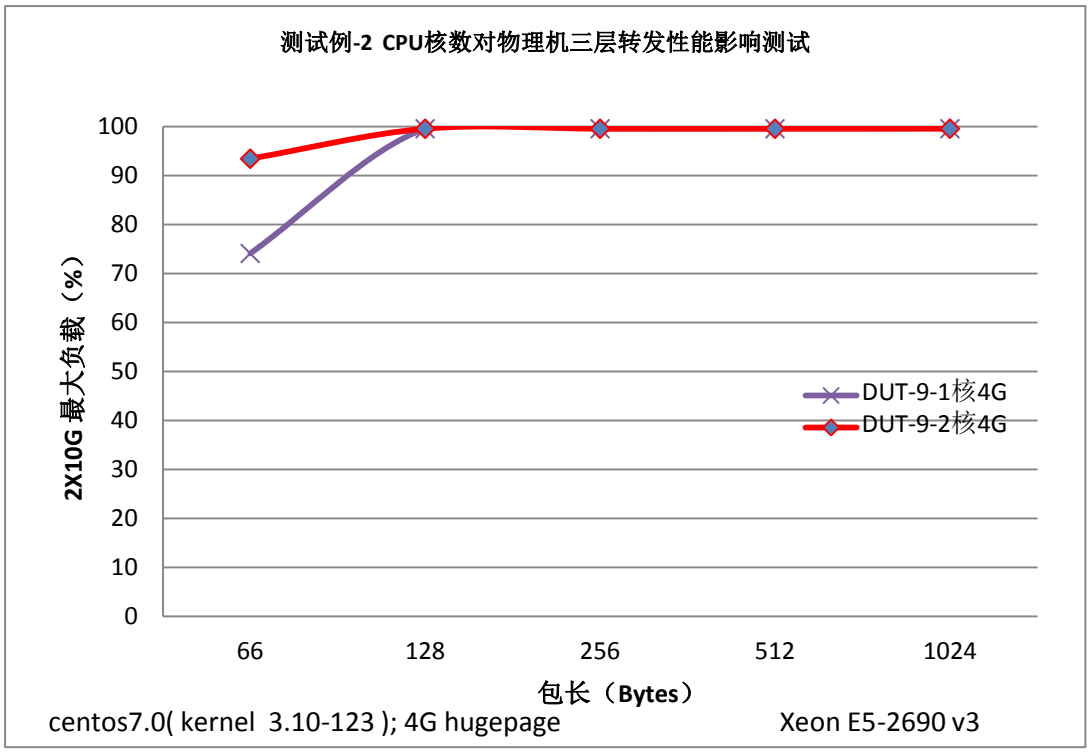

图 5－6 测试例-2 吞吐量性能比较

小结：该测试例表明增加核数可以改善小包的转发性能，但是否增加核数要根据应用场景进行调整。为保证最佳性能，本白皮书的后续测试中将在物理机和虚拟机的 L3fwd 转发程序中使用 2 个转发核。不同 DPDK 应用转发程序使用的 CPU 核数，应该由 DPDK 应用程序开发者决定。

5.5.2.3NUMA 对物理机三层转发性能影响测试

测试场景

表 5－13 测试例-3 测试场景说明

测试编号	测试例-3		
CPU 型号	Xeon E5-2630 v3		
HOST 操作系统	centos7.0		
DPDK 转发核数	2 核		
Hugepage 设置	4G		
跨路情况	CPU 与网卡同 SOCKET	CPU 跨 SOCKET	网卡跨 SOCKET
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口	Port0⇔ L3fwd ⇔Port2，2 x 10G 端口
转发与连接配置	图 5-7	图 5-8	图 5-9

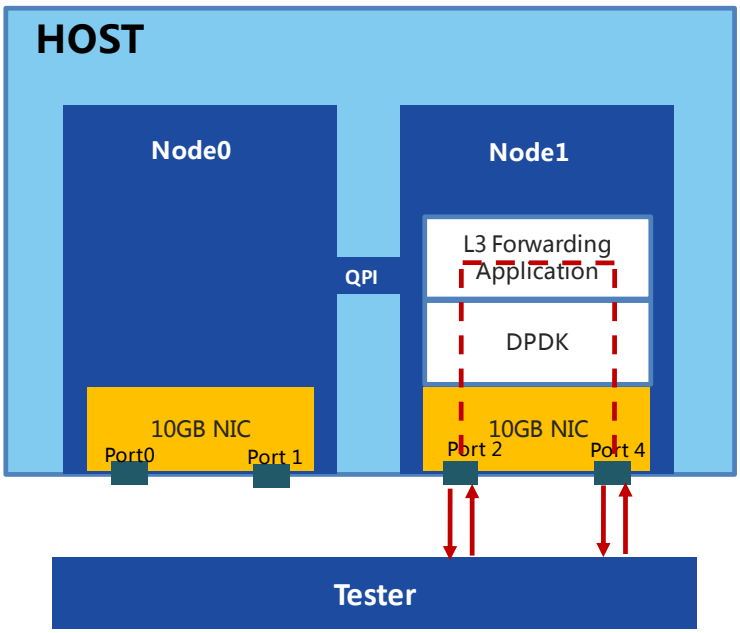

图 5－7 测试例-3 CPU 与网卡同 SOCKET 测试转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

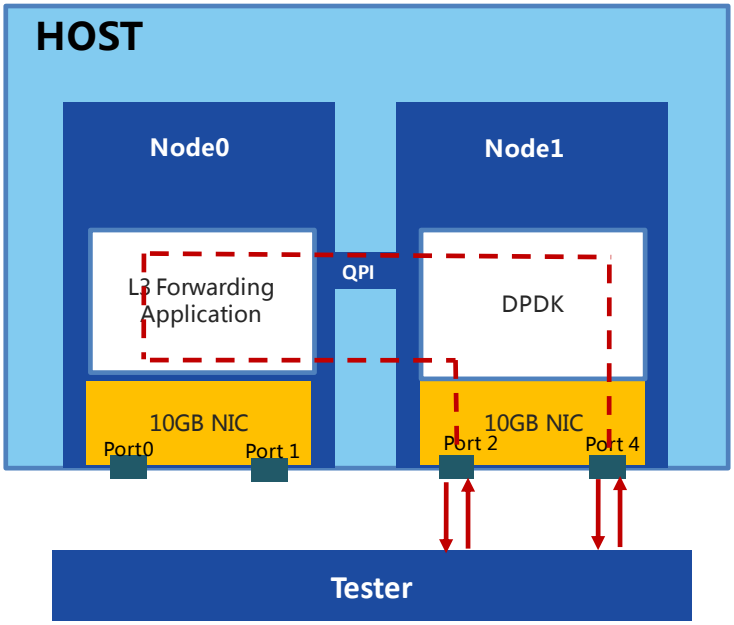

图 5－8 测试例-3 CPU 跨 SOCKET 测试转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

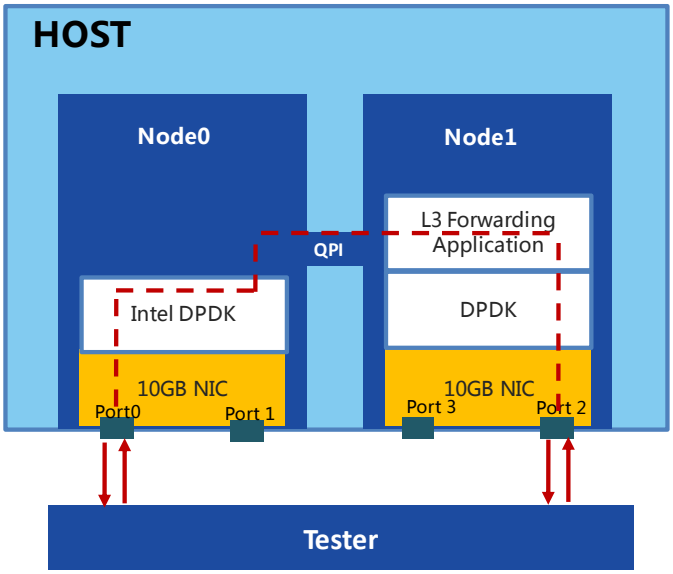

图 5－9 测试例-3 网卡跨 SOCKET 测试转发与连接配置

测试环境配置

表 5－14 测试例-3 测试环境配置

服务器配置	DUT-8
服务器 bios 配置	见表 5-5“DUT-8”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 2”
DPDK 编译	见附录四《DPDK 编译》
DPDK 转发程序编译	见附录六《L3fwd 程序编译》
DPDK 软件配置	见附录五《DPDK 软件启动配置》注意 6、7 项的启动参数，同路与跨路情况绑定不同参数。

测试结果

表 5－15 测试例-3 吞吐量性能数据

包长	DUT-8-同 SOCKET 负载（%）	DUT-8-CPU 跨 SOCKET 负载（%）	DUT-8-网卡跨 SOCKET 负载（%）
66	93.478	65.152	76.786
128	99.601	99.601	99.601
256	99.601	99.601	99.601
512	99.601	99.601	99.601
1024	99.618	99.618	99.618

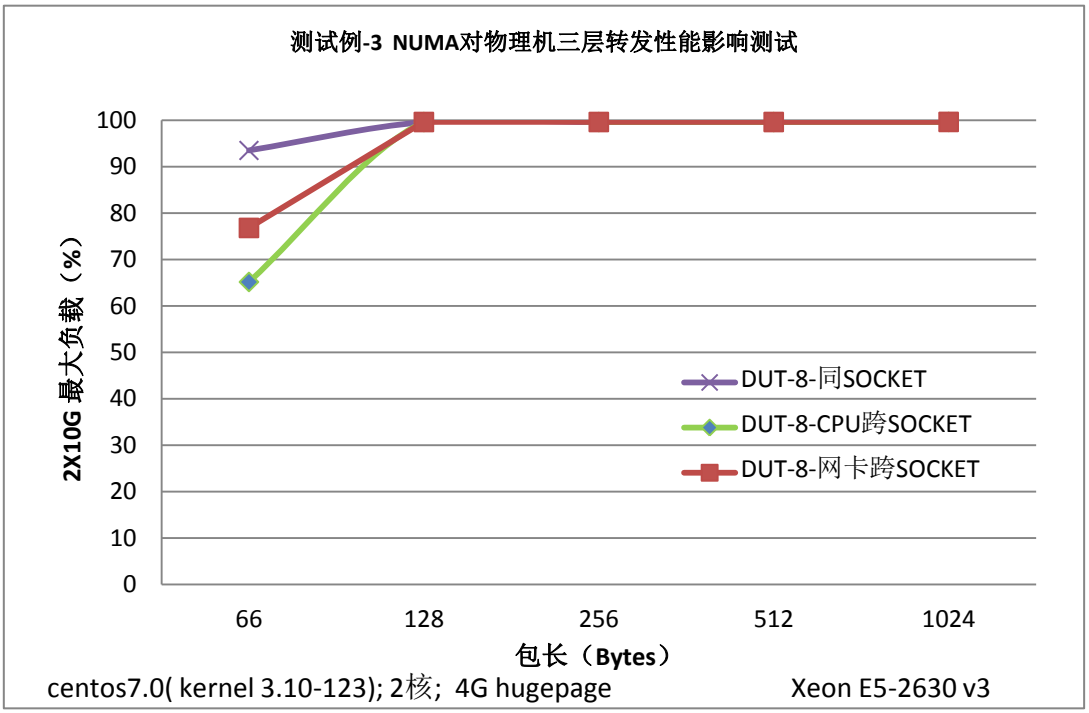

图 5－10 测试例-3 吞吐量性能比较

小结：该测试例有三种 NUMA 跨接方法：

“CPU 与网卡同 SOCKET”，如图 5-7 所示，此时 DPDK 转发性能最佳；

“网卡跨 SOCKET”，如图 5-9 所示，此时 DPDK 转发性能适中；

“CPU 跨 SOCKET”，如图 5-8 所示，此时 DPDK 转发性能劣化明显；

上述测试结果表明，在 20G 吞吐量情况下，NUMA 结构跨接方法对于中长包没有影响，但是对于小包（低于 128 字节）转发有明显影响。大部分被测设备均有类似结果。

为保证最佳性能，应采用“CPU 与网卡同 SOCKET”的方法进行应用程序开发。

5.5.2.4不同被测物理机三层转发性能差异测试

测试场景

表 5－16 测试例-4 测试场景说明

测试编号	测试例-4
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2690 v3
HOST 操作系统	centos7.0
DPDK 转发核数	2 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	4G

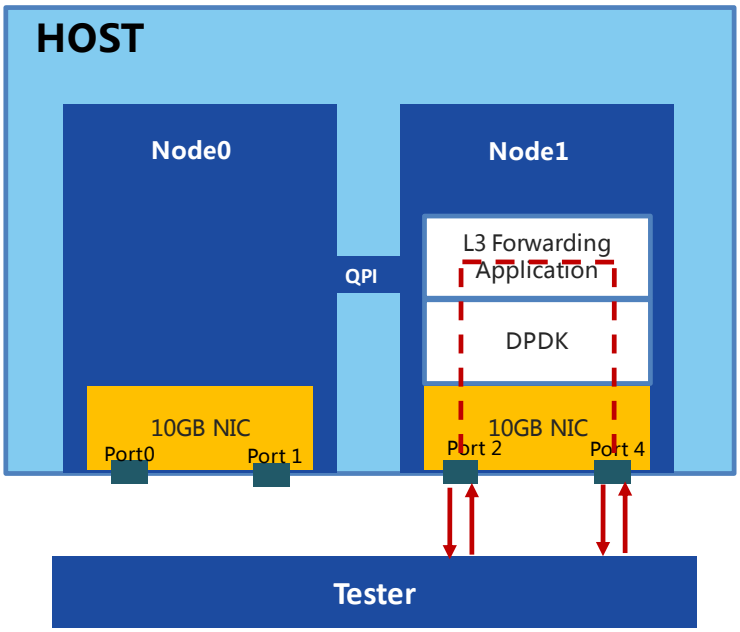

图 5－11 测试例-4 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－17 测试例-4 测试环境配置

服务器配置	DUT-2、DUT-4、DUT-7、DUT-9
服务器 bios 配置	见表 5-5“DUT-2”、“DUT-4”、“DUT-7”、“DUT-9”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 1”
Dpdk 编译	见附录四《DPDK 编译》
DPDK 转发程序编译	见附录六《L3fwd 程序编译》
DPDK 软件配置	见附录五《DPDK 软件启动配置》

测试结果

表 5－18 测试例-4 吞吐量性能数据

包长	DUT-4 负载（%）	DUT-7 负载（%）	DUT-2 负载（%）	DUT-9 负载（%）
66	91.117	93.478	93.478	93.478
128	97.368	99.601	99.601	99.601
256	99.601	99.601	99.601	99.601
512	99.601	99.601	99.601	99.601
1024	99.618	99.618	99.618	99.618

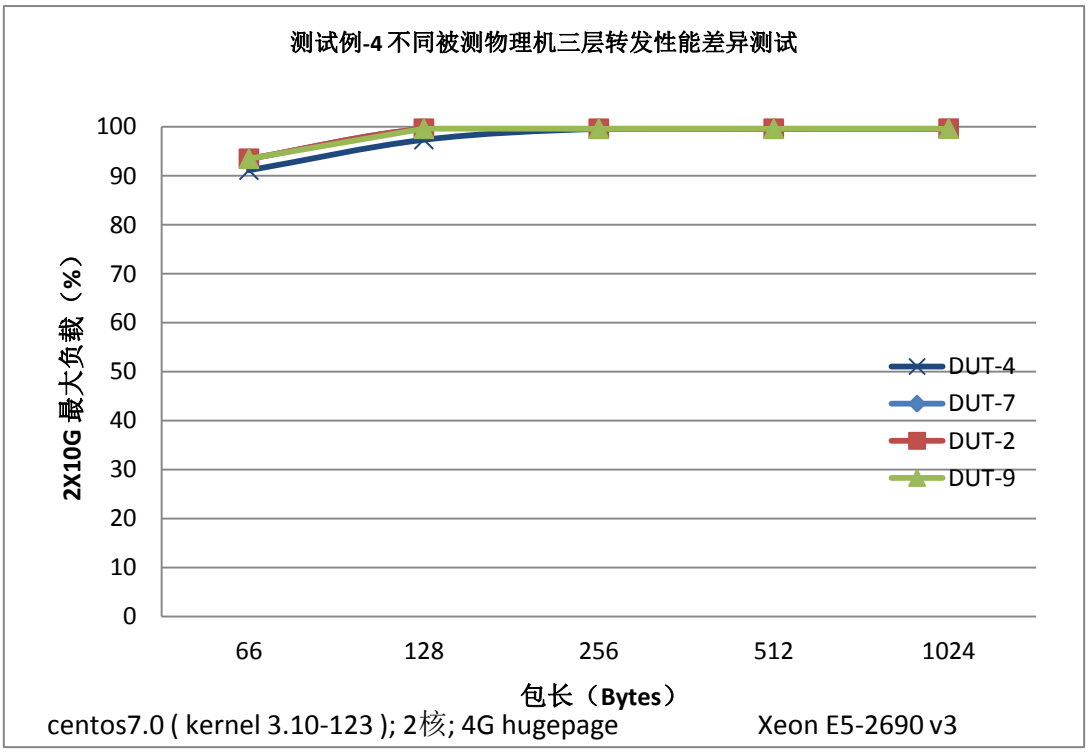

图 5－12 测试例-4 吞吐量性能比较

小结：本测试例主要考察不同厂家在相同基本配置情况下的物理机三层转发性能，采用

“CPU 与网卡同 SOCKET”跨接方法进行测试，结果表明：不同被测设备（每台设备均分属不同厂家）情况下，DPDK 在物理机上的三层转发性能无显著差异。

5.5.2.5不同 CPU 型号对物理机三层转发性能影响测试

测试场景

表 5－19 测试例-5 测试场景说明

测试编号	测试例-5
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2630 v3、Xeon E5-2690 v3
HOST 操作系统	centos7.0
DPDK 转发核数	2 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	4G

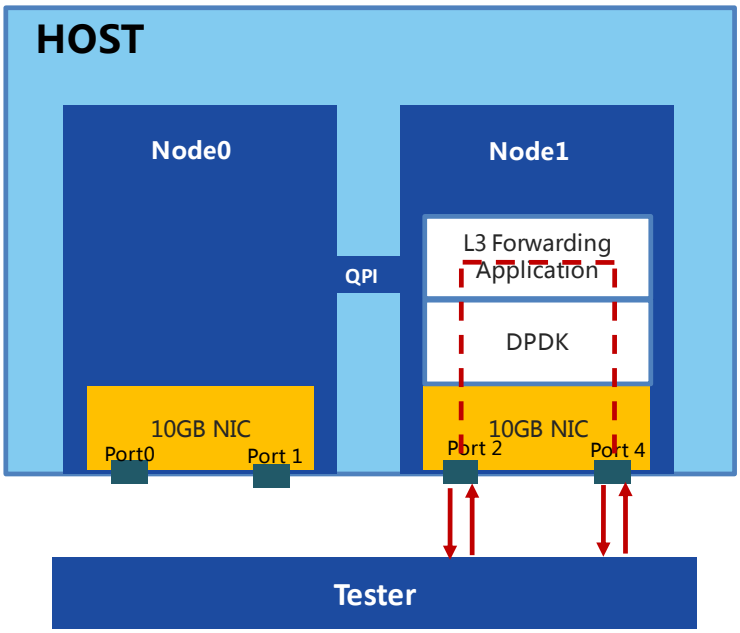

图 5－13 测试例-5 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－20 测试例-5 测试环境配置

服务器配置	DUT-2、 DUT-3
服务器 bios 配置	见表 5-5“DUT-2”、“DUT-3”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 1”与“配置 2”
DPDK 编译	见附录四《DPDK 编译》
DPDK 转发程序编译	见附录六《L3fwd 程序编译》
DPDK 软件配置	见附录五《DPDK 软件启动配置》

测试结果

表 5－21 测试例-5 吞吐量性能数据

包长	DUT-3 负载（%）	DUT-2 负载（%）
66	91.117	93.478
128	99.601	99.601
256	99.601	99.601
512	99.601	99.601
1024	99.618	99.618

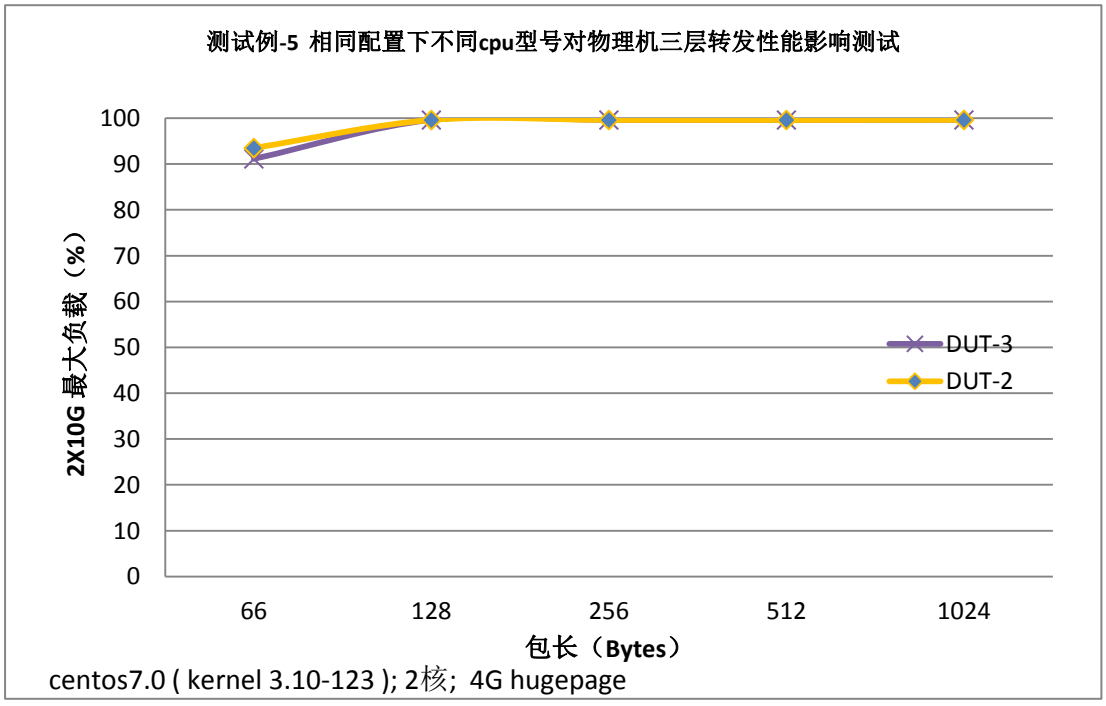

图 5－14 测试例-5 吞吐量性能比较

小结：该测试例的测试结果表明，在 20G 吞吐量情况下，因未达到 CPU 处理瓶颈，不同规格的 CPU 型号（Xeon E5-2690 V3 和 Xeon E5-2630 V3）对 DPDK 物理机三层转发性能无显著影响，需要进一步增大设备的负载流量，或者结合 App 对 CPU 资源的大量消耗，才能检测出 CPU 性能差异对转发性能的影响。大部分被测设备均有类似结果。

5.5.2.6不同 Linux 版本对物理机三层转发性能影响测试

场景描述

表 5－22 测试例-6 测试场景说明

测试编号	测试例-6
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2630 v3、Xeon E5-2690 v3
HOST 操作系统	centos7.0、centos7.1
DPDK 转发核数	2 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	4G

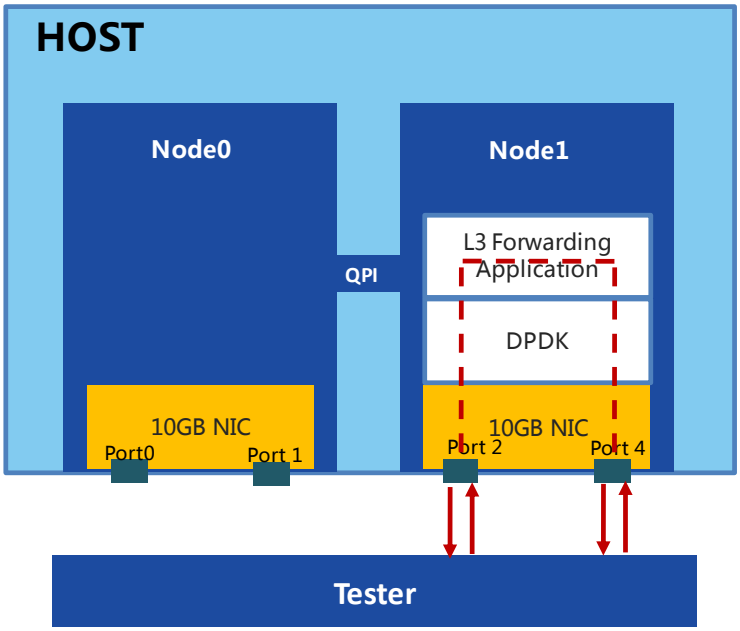

图 5－15 测试例-6 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－23 测试例-6 测试环境配置

服务器配置	DUT-3
服务器 bios 配置	见表 5-5“DUT-3”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64”和“CentOS 7.1x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 2”
DPDK 编译	见附录四《DPDK 编译》
DPDK 转发程序编译	见附录六《L3fwd 程序编译》
DPDK 软件配置	见附录五《DPDK 软件启动配置》

测试结果

表 5－24 测试例-6 吞吐量性能数据

包长	DUT-3-7.0 负载（%）	DUT-3-7.1 负载（%）
66	91.117	93.478
128	99.601	99.601
256	99.601	99.601
512	99.601	99.601
1024	99.618	99.618

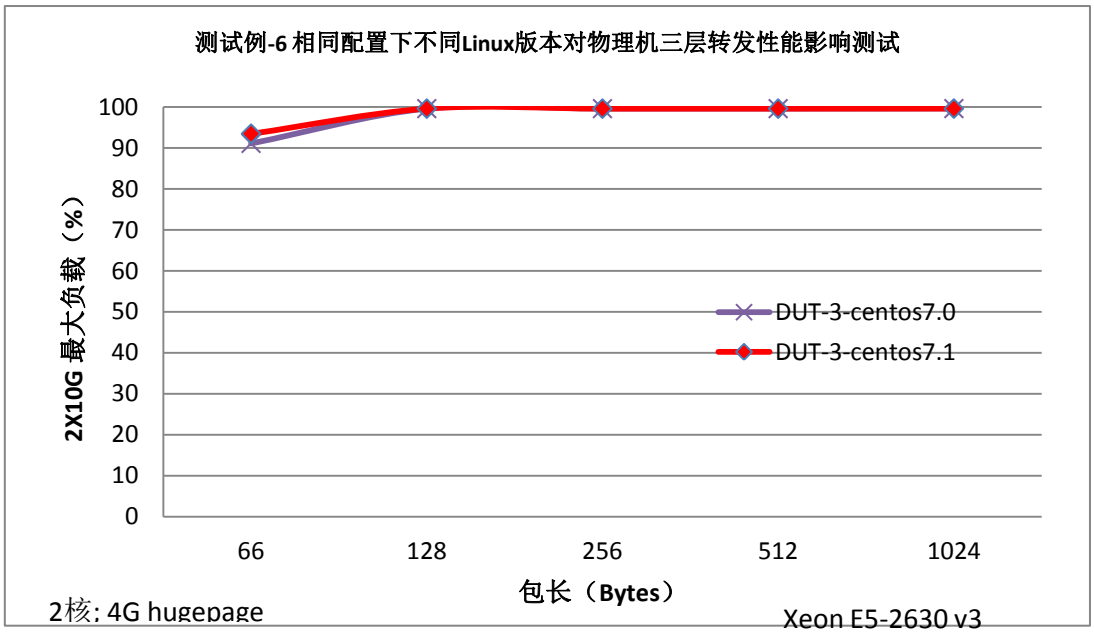

图 5－16 测试例-6 吞吐量性能比较

小结：该测试例的测试结果表明，在 20G 吞吐量情况下，同一被测设备中不同的 Linux 操作系统版本对 DPDK 转发程序的转发性能无显著影响。在 VNF 直接运行物理机的实现方案中，可选用的 Linux 操作系统版本范围较广，可以不用过于考虑性能差异。

5.5.3 SR-IOV 测试

5.5.3.1不同被测物理机 SR-IOV 转发性能差异

测试场景

表 5－25 测试例-7 测试场景说明

测试编号	测试例-7
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2630 v3
HOST 操作系统	centos7.0
VM 操作系统	centos7.0
VM 配置	4 核、6G 内存，40G 硬盘
DPDK 转发核数	2 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	4G

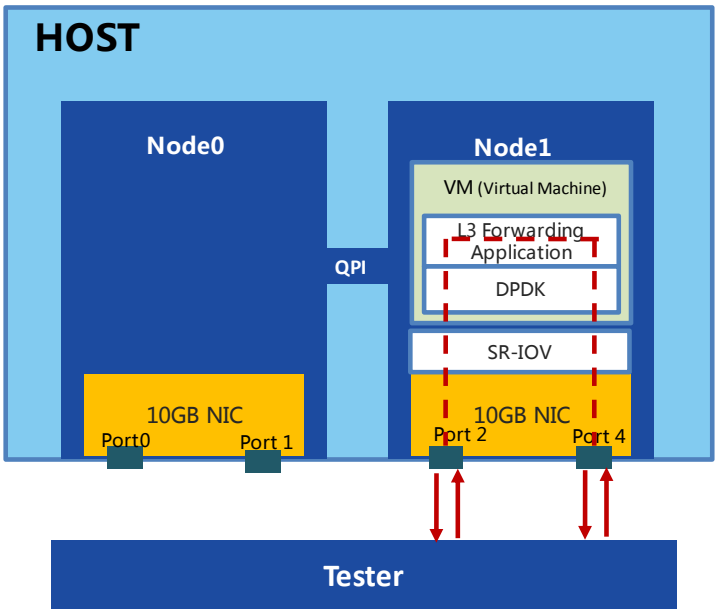

图 5－17 测试例-7 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－26 测试例-7 测试环境配置

服务器配置	DUT-1、DUT-3、DUT-5、DUT-8
服务器 bios 配置	见表 5-5“DUT-2”、“DUT-4”、“DUT-7”、“DUT-9”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 2”
虚拟机中 DPDK 转发程序编译	见附录六《L3fwd 程序编译》
Dpdk 编译	见附录四《DPDK 编译》
虚拟机中 DPDK 软件配置	见附录五《DPDK 软件启动配置》
虚拟机中操作系统配置	见附录三《操作系统启动参数》中“配置 5”
虚拟机配置启动配置	见附录七《SR-IOV 测试配置》

测试结果

表 5－27 测试例-7 吞吐量性能数据

包长	DUT-5 负载（%）	DUT-8 负载（%）	DUT3 负载（%）	DUT1 负载（%）	DUT-6 负载（%）
66	65.152	88.716	88.716	93.478	86.01
128	82.561	99.601	99.601	99.601	99.601
256	96.116	99.601	99.601	99.601	99.601
512	99.601	99.601	99.601	99.601	99.601
1024	99.618	99.618	99.618	99.618	99.618

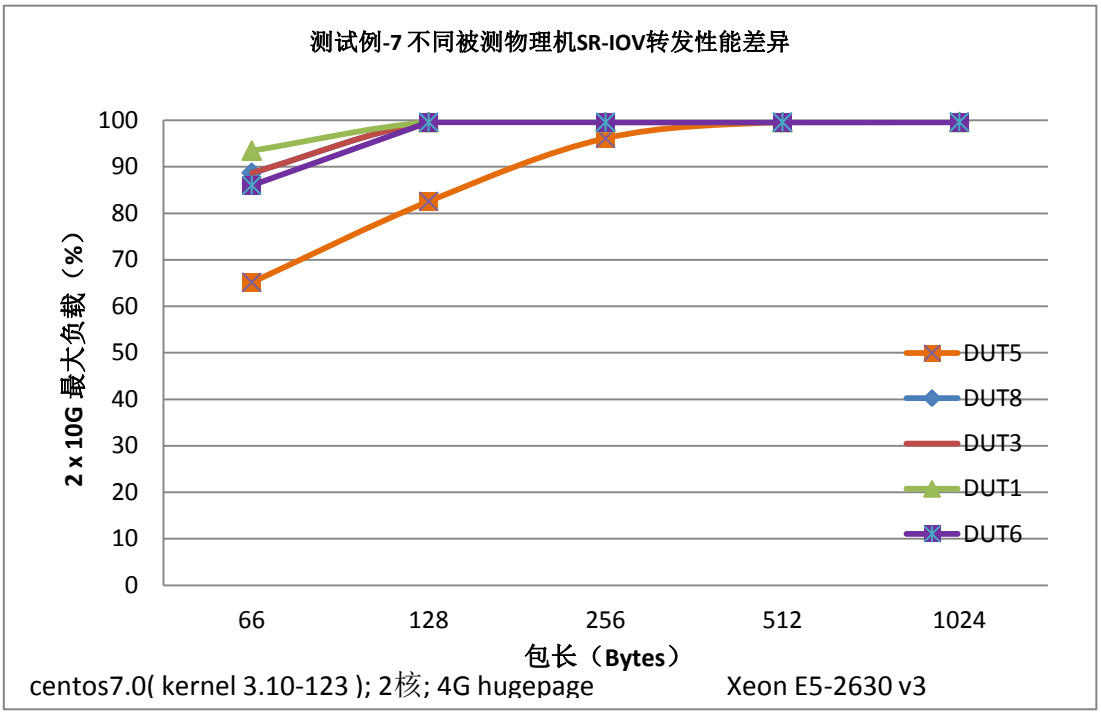

图 5－18 测试例-7 吞吐量性能比较

小结：本测试例主要考察不同厂家的 DUT 在相同基本配置情况下，SR-IOV 结合 VM 内部的 DPDK 三层转发性能。测试结果表明：在中小包情况下，各厂家被测设备 SR-IOV 转发性能差异明显，相差幅度在 10%-25%之间，增加负载流量有可能进一步拉大差距。因为各厂家无法提供统一的 BIOS 配置接口，造成结果差异的原因可能是 BIOS 设置或者硬件结构的不同。

5.5.3.2不同 CPU 型号对 SR-IOV 转发性能影响测试

测试场景

表 5－28 测试例-8 测试场景说明

测试编号	测试例-8
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2630 v3、Xeon E5-2690 v3
HOST 操作系统	centos7.0
VM 操作系统	centos7.0
VM 配置	4 核、6G 内存，40G 硬盘
DPDK 转发核数	2 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	4G

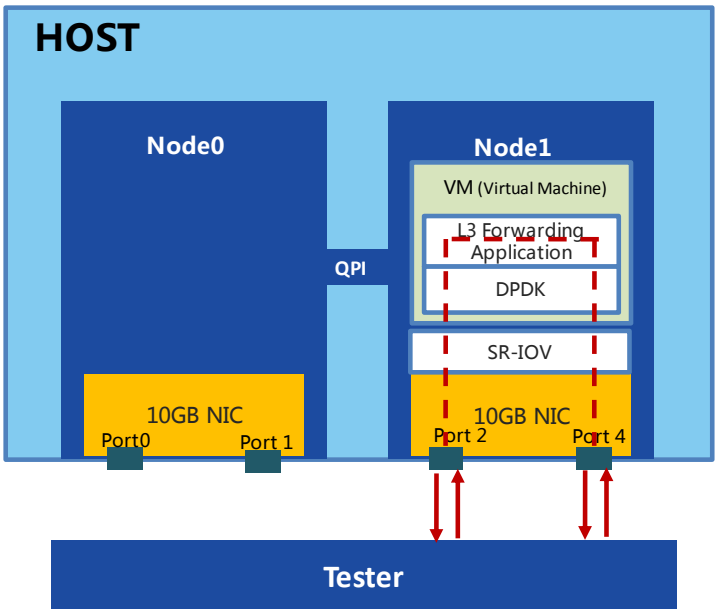

图 5－19 测试例-8 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－29 测试例-8 测试环境配置

服务器配置	DUT-7、 DUT-8
服务器 bios 配置	见表 5-5“DUT-7”、“DUT-8”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 1”与“配置 2”
虚拟机中 DPDK 转发程序编译	见附录六《L3fwd 程序编译》
DPDK 编译	见附录四《DPDK 编译》
虚拟机中 DPDK 软件配置	见附录五《DPDK 软件启动配置》
虚拟机中操作系统配置	见附录三《操作系统启动参数》中“配置 5”
虚拟机配置启动配置	见附录七《SR-IOV 测试配置》

测试结果

表 5－30 测试例-8 吞吐量性能数据

包长	DUT-8 负载（%）	DUT-7 负载（%）
66	88.716	93.478
128	99.601	99.601
256	99.601	99.601
512	99.601	99.601
1024	99.618	99.618

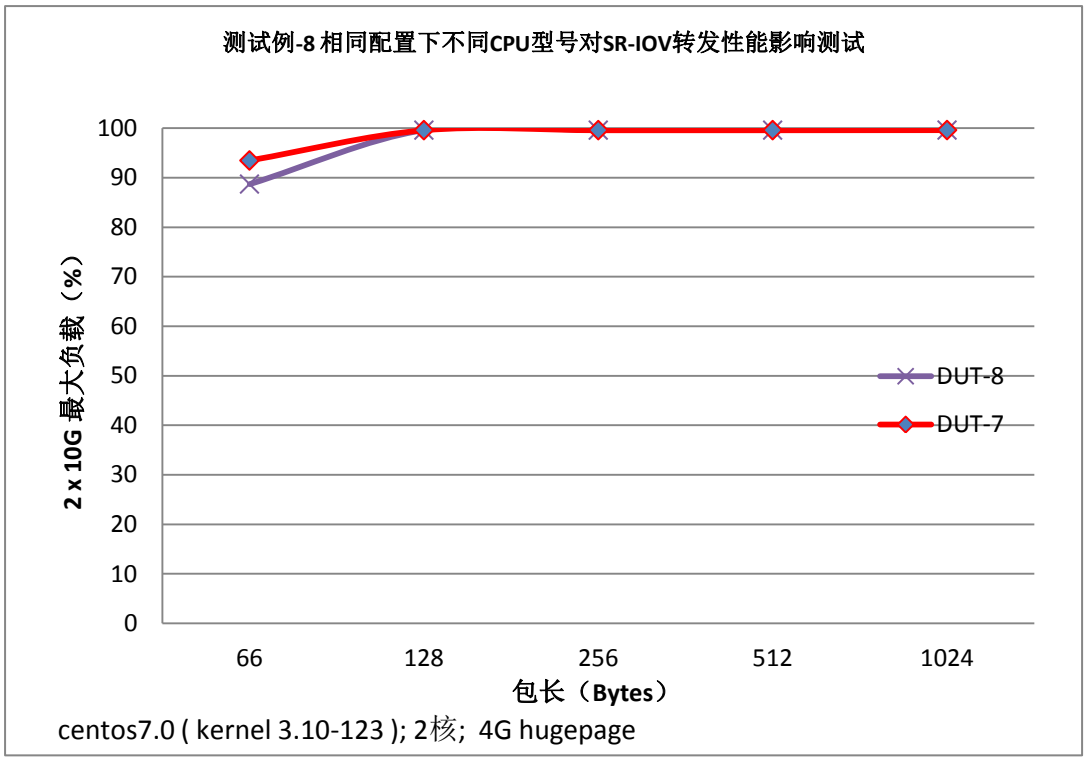

图 5－20 测试例-8 吞吐量性能比较

小结：该测试例采用同厂家设备置换 CPU 方式进行测试，以避免 BIOS 及硬件结构差异。结果表明：在 20G 吞吐量情况下，不同规格的 CPU 型号（Xeon E5-2690 V3 和 Xeon E5-2630 V3），SR-IOV 结合 VM 内部的 DPDK 三层转发性能无显著差异（约 5%）。大部分被测设备均有类似结果。

后续将进一步增大设备的负载流量，并结合 VNF 对 CPU 资源的消耗状况，检测 CPU 性能差异对转发性能的影响。

5.5.3.3不同 Linux kernel 版本对 SR-IOV 转发性能影响测试

测试场景

表 5－31 测试例-9 测试场景说明

测试编号	测试例-9		
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口		
HOST 操作系统	centos7.0、centos7.1	centos7.0（kernel3.10.0-123 与 kernel3.10.0-229）	fedora20、21 与 22
VM 操作系统	centos7.0	centos7.0	centos7.0
CPU 型号	Xeon E5-2630 v3	Xeon E5-2699 v3	Xeon E5-2699 v3
VM 配置	4 核、6G 内存，40G 硬盘		
DPDK 转发核数	2 核		
跨路情况	CPU 与网卡同 SOCKET		
Hugepage 设置	4G		

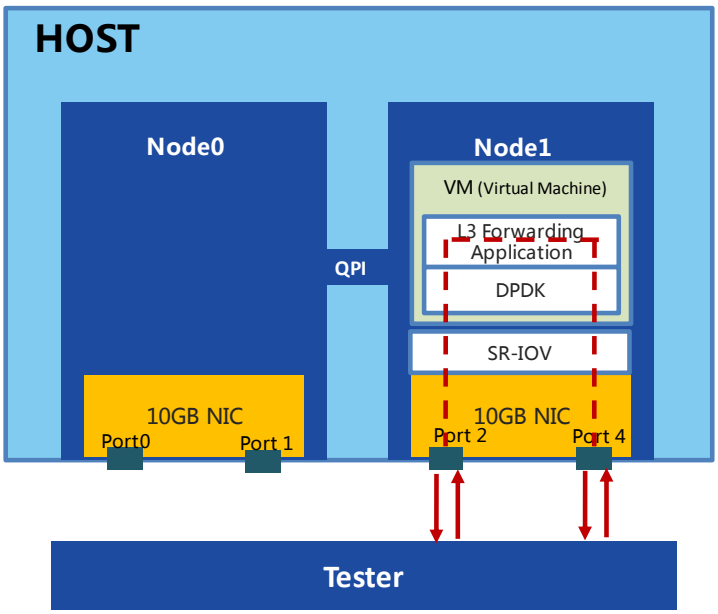

图 5－21 测试例-9 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－32 测试例-9 测试环境配置

	Centos7.0 与 7.1 对比	Kernel 对比	fedora20、21 与 22 对比
服务器配置	DUT-3	DUT-11	DUT-11
服务器 bios 配置	见表 5-5“DUT-3”设置	见表 5-5“DUT-11”设置	见表 5-5“DUT-11”设置
操作系统配置	见表 5-6“CentOS 7.0 x86_64”与“CentOS 7.1x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 2”	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 3”	见表 5-6，Fedora 20、Fedora 21 与 Fedora 20 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 3”
虚拟机中 DPDK 转发程序编译	见附录六《L3fwd 程序编译》		
DPDK 编译	见附录四《DPDK 编译》		
虚拟机中 DPDK 软件配置	见附录五《DPDK 软件启动配置》		
虚拟机中操作系统配置	见附录三《操作系统启动参数》中“配置 5”		
虚拟机配置启动配置	见附录七《SR-IOV 测试配置》		

测试结果

1) centos7.0 与 centos7.1 对比

表 5－33 测试例-9 centos7.0 与 centos7.1 对比吞吐量性能数据

包长	DUT-3-centos7.0 负载（%）	DUT-3-centos7.1 负载（%）
66	88.716	19.027
128	99.601	27.407
256	99.601	57.5
512	99.601	97.79
1024	99.618	99.618

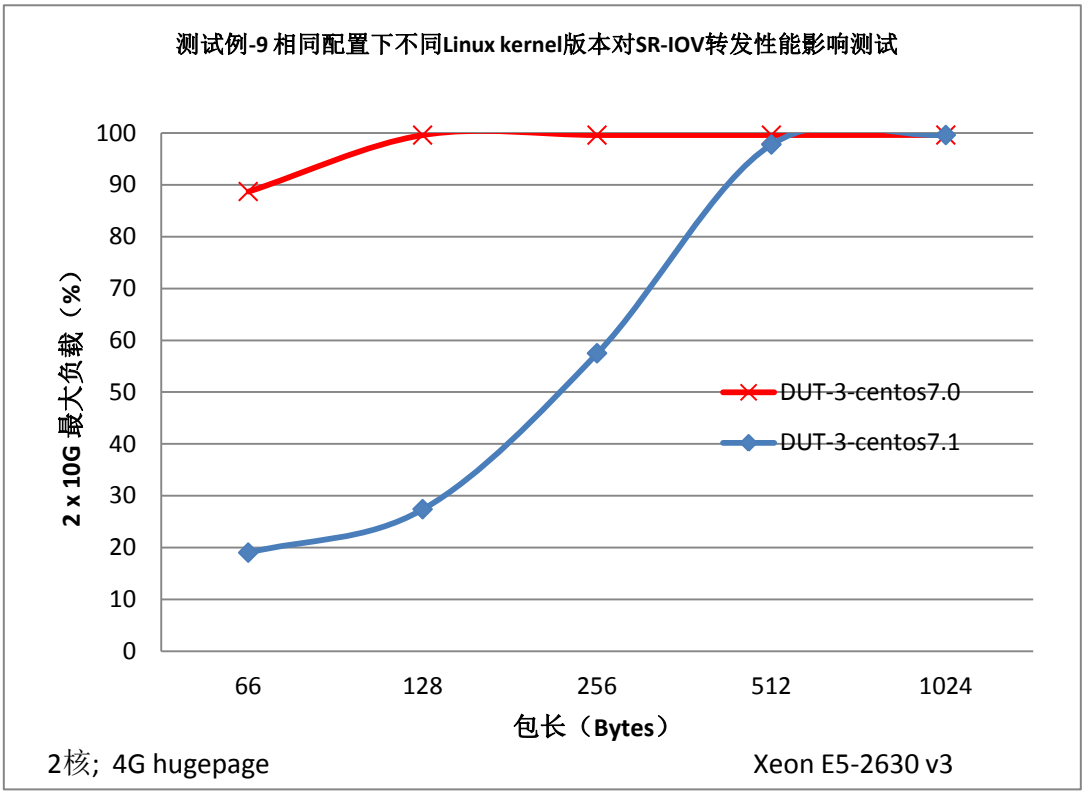

图 5－22 测试例-9 centos7.0 与 centos7.1 对比吞吐量性能比较

小结：该测试用于对比在相同被测设备中，HOST 上操作系统的版本差异，对 SR-IOV 结合 VM 内部的 DPDK 三层转发性能的影响。 centos7.0 与 centos7.1 的对比测试结果表明：HOST 上安装 centos7.0 时，SR-IOV 结合 VM 内部的 DPDK 三层转发性能明显优于安装 centos7.1 的情形。目前在 CentOS7.1 上经过一系列系统调优，性能可进一步提升，我们将在后续工作中进一步完善相关的优化方法。

2) fedora20、21 与 22 版本对比测试

表 5－34 测试例-9 fedora20、21 与 22 版本对比吞吐量性能数据

包长	DUT-11-fedora20 负载（%）	DUT-11-fedora21 负载（%）	DUT-11-fedora22 负载（%）
66	88.715	23.889	21.09
128	99.601	40.659	35.922
256	99.601	72.142	70.381
512	99.601	99.601	99.601
1024	99.618	99.618	99.618

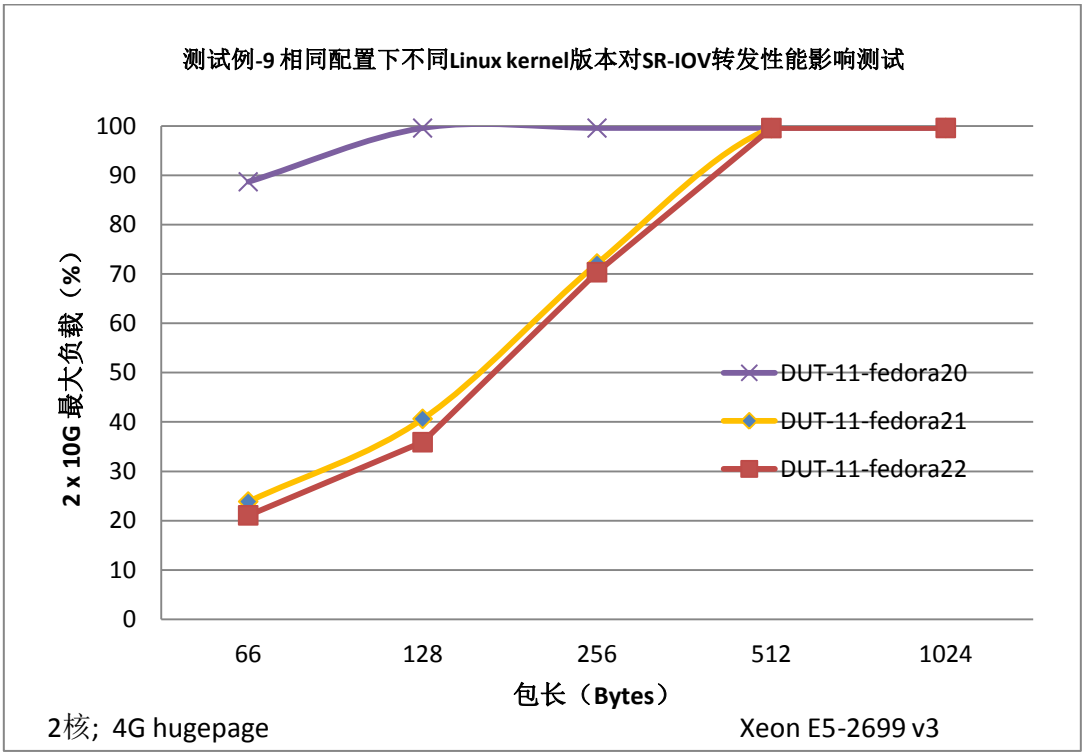

图 5－23 测试例-9 fedora20、21 与 22 版本对比吞吐量性能比较

小结：该测试用于对比在相同被测设备中，在 HOST 上的操作系统版本差异对 SR-IOV 结合 VM 内部 DPDK 三层转发性能的影响。Fedora20、Fedora21 和 Fedora22 的对比测试结果表明：HOST 上安装 Fedora20 时，SR-IOV 结合 VM 内部 DPDK 三层转发性能明显优于安装 Fedora21 和 Fedora22 的情形。

3) Kernel 测试

表 5－35 测试例-9 Kernel 对比测试吞吐量性能数据

包长	kernel 3.10.0-229 负载（%）	kernel 3.10.0-123 负载（%）
66	6.287	74.138
128	10	99.601
256	20.533	99.601
512	39.466	99.601
1024	75.872	99.618

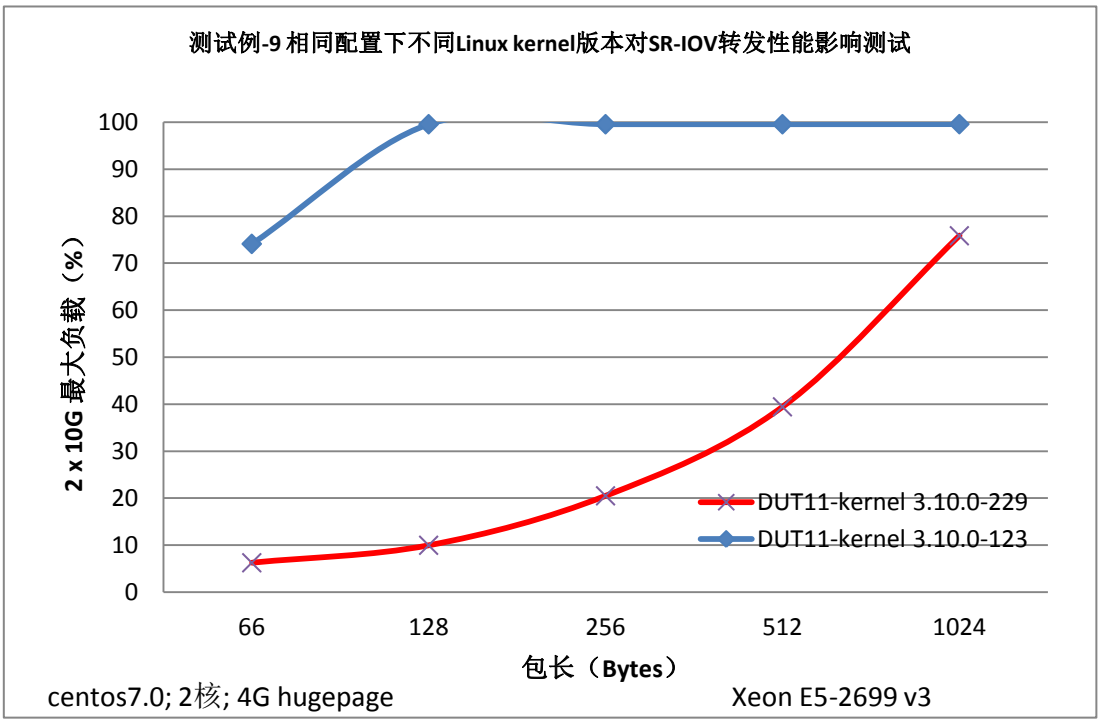

图 5－24 测试例-9 Kernel 对比测试吞吐量性能比较

小结：该测试用于对比在相同被测设备中，HOST 上操作系统中的 kernel 版本差异，对 SR-IOV 结合 VM 内部的 DPDK 三层转发性能的影响。在 centos7.0 操作系统中，3.10.0-123.el7.x86_64 内核和 3.10.0-229.el7.x86_64 内核的对比测试结果表明：HOST 上安装 3.10.0-123.el7.x86_64 内核时，SR-IOV 结合 VM 内部的 DPDK 三层转发性能明显优于安装 3.10.0-229.el7.x86_64 内核的情形。目前原因未明，后续工作中将会继续研究。

综合小结

在 VNF+SR-IOV 应用场景下，操作系统对服务器整体转发性能的影响分为两种情况：

1） HOST 操作系统版本差异的影响

kernel 版本差异是主要原因，部分版本间性能相差 60%，目前测得 3.10.0-123.el7.x86_64 内核性能较佳。实际应用中需要仔细筛选 kernel 版本。

2） Guest（VM 内部）操作系统版本差异的影响

大量测试结果表明 Guest 操作系统对 SR-IOV 场景整体转发性能无显著影响，本文不再提供详细数据。

5.5.3.4NUMA 对 SR-IOV 转发性能影响测试

测试场景

表 5－36 测试例-10 测试场景说明

测试编号	测试例-10		
CPU 型号	Xeon E5-2630 v3		
HOST 操作系统	centos7.0		
DPDK 转发核数	2 核		
VM 操作系统	centos7.0		
Hugepage 设置	4G		
VM 配置	4 核、6G 内存，40G 硬盘		
跨路情况	CPU 与网卡同 SOCKET	CPU 跨 SOCKET	网卡跨 SOCKET
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口	Port0⇔ L3fwd ⇔Port2，2 x 10G 端口
转发与连接配置	图 5-25	图 5-26	图 5-27

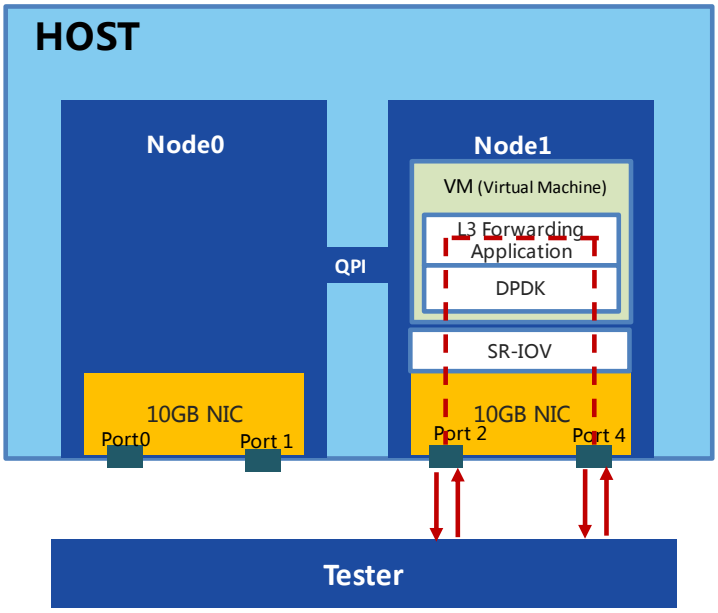

图 5－25 测试例-10 CPU 与网卡同 SOCKET 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

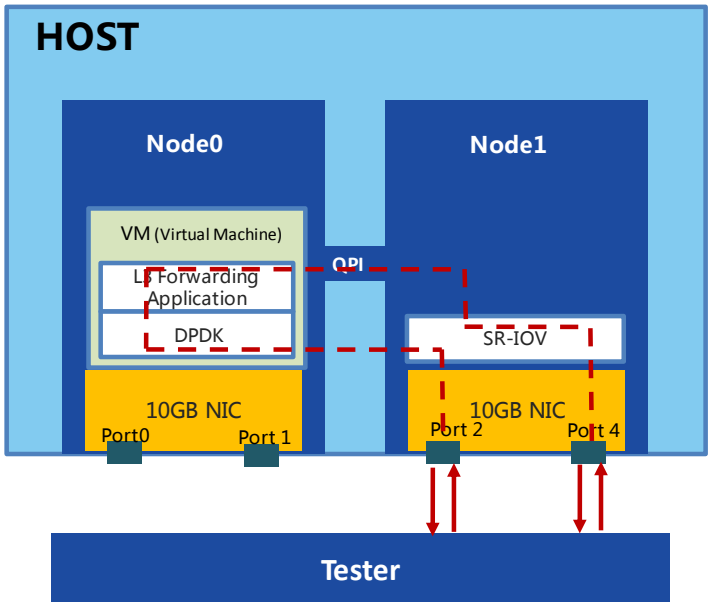

图 5－26 测试例-10 CPU 跨 SOCKET 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

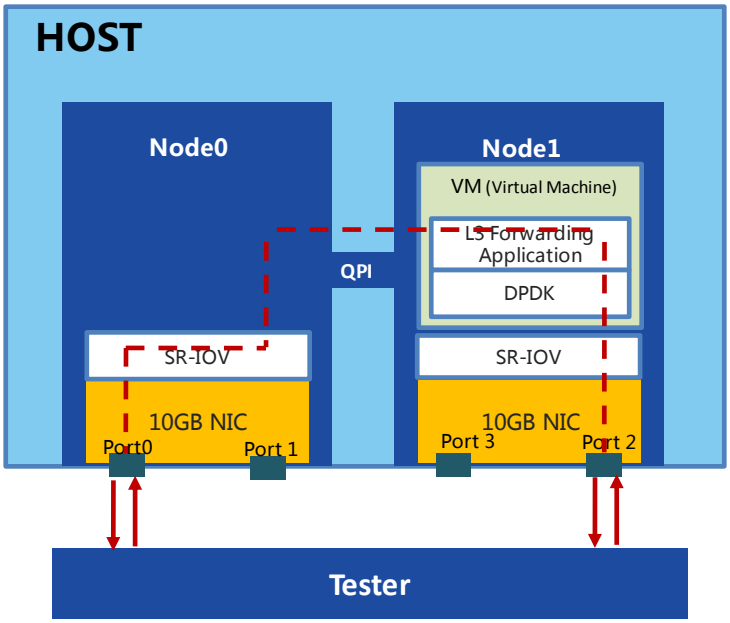

图 5－27 测试例-10 网卡跨 SOCKET 转发与连接配置

测试环境配置

表 5 - 37 测试例-10 测试环境配置

服务器配置	DUT-3
服务器 bios 配置	见表 5-5“DUT-3”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 2”
虚拟机中 DPDK 转发程序编译	见附录六《L3fwd 程序编译》
虚拟机中 DPDK 软件配置	见附录五《DPDK 软件启动配置》
DPDK 编译	见附录四《DPDK 编译》
虚拟机中操作系统配置	见附录三《操作系统启动参数》中“配置 5”
虚拟机配置启动配置	见附录七《SR-IOV 测试配置》

测试结果

表 5 - 38 测试例-10 吞吐量性能数据

包长	DUT-3-同 SOCKET 负载（%）	DUT-3-CPU 跨 SOCKET 负载（%）	DUT-3-网卡跨 SOCKET 负载（%）
66	88.716	73.247	54.422
128	99.601	99.601	96.267
256	99.601	99.601	99.601
512	99.601	99.601	99.601
1024	99.618	99.618	99.618

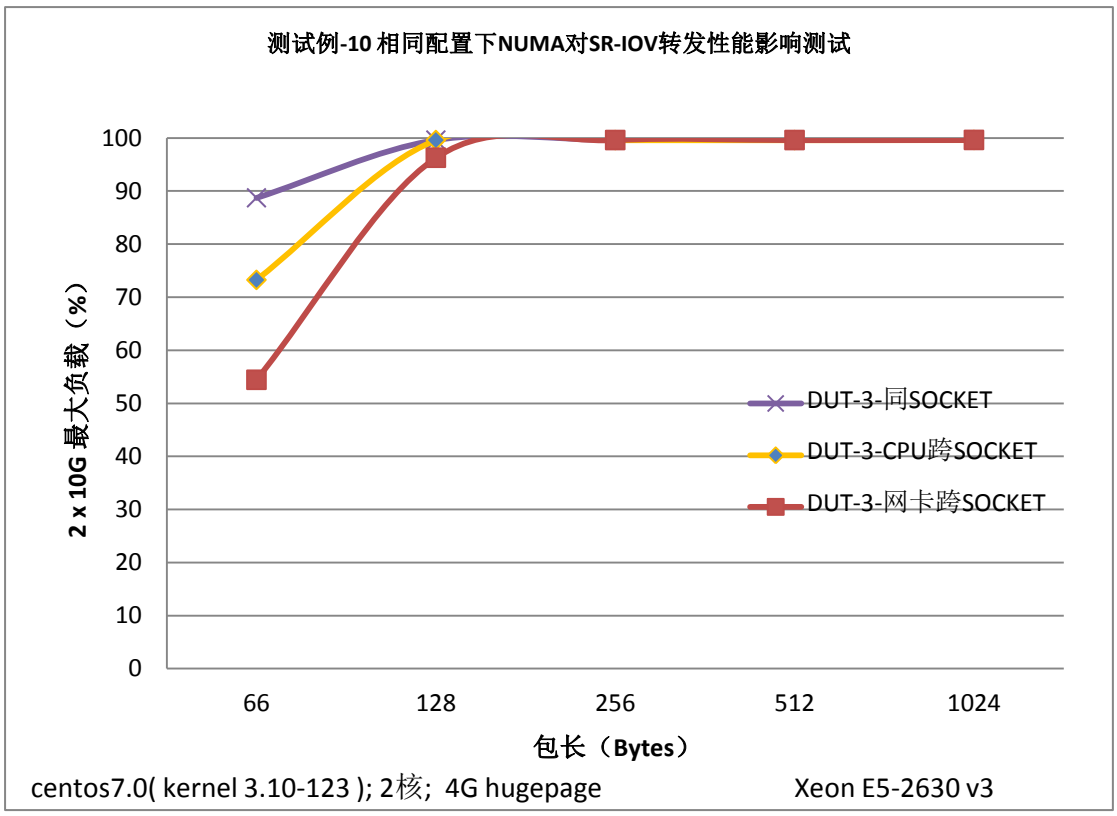

图 5－28 测试例-10 吞吐量性能比较

小结：该测试例有三种 NUMA 跨接方法：

“CPU 与网卡同 SOCKET”，如图 5-26 所示，此时 SR-IOV 转发性能最佳；

“CPU 跨 SOCKET”，如图 5-27 所示，此时 SR-IOV 转发性能适中；

“网卡跨 SOCKET”，如图 5-28 所示，此时 SR-IOV 转发性能劣化明显；

在 20G 吞吐量情况下，本测试例与物理机三层转发性能（测试例-3）的测试结果近似，说明 NUMA 结构跨接方法对于中长包没有影响，但是对于小包（低于 128 字节）转发有明显影响。说明 SR-IOV 技术可以达到与物理机转发近似的性能（相差 10%以内）。

为保证最佳性能，VM（VNF）所使用的逻辑 CPU 应与物理网卡（启用 SR-IOV 功能）在同一 SOCKET 上。本测试例选用 DUT-3，其他 DUT 设备均呈现近似性能测试结果。

5.5.4 OVS 测试

本节所述测试例主要用于说明使用基于 DPDK 的 OVS（OVS with DPDK-netdev）交换机转发性能以及性能影响因素，测试流量仅从物理网口进入 OVS，并由 OVS 进行查表和转发至另一物理网口，并不涉及 VM 间的流量转发问题。

5.5.4.1不同被测物理机 OVS 转发性能差异测试

测试场景

表 5－39 测试例-11 测试场景说明

测试编号	测试例-11
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2630 v3、XEON E5-2690 V3
HOST 操作系统	centos7.0
DPDK 转发核数	2 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	4G

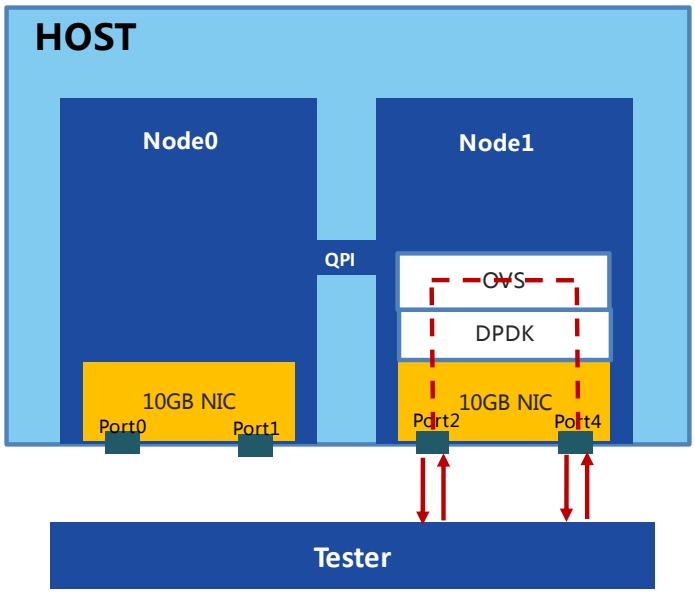

图 5－29 测试例-11 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－40 测试例-11 测试环境配置

服务器配置	DUT-1、DUT-3、DUT-5、DUT-8
服务器 bios 配置	见表 5-5“DUT-1”、“DUT-3”、“DUT-5”、“DUT-8”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 2”
DPDK 编译	见附录四《DPDK 编译》
OVS 安装	见附录八《OVS 安装》
OVS 配置	见附录九《OVS 测试配置》

测试结果

表 5－41 测试例-11 吞吐量性能数据

包长	DUT-5 负载（%）	DUT-8 负载（%）	DUT-3 负载（%）	DUT-1 负载（%）
66	65.152	43.225	53.75	43.225
128	82.561	74	90.244	74.747
256	96.116	96.814	99.601	98.571
512	99.601	99.601	99.601	99.601
1024	99.618	99.618	99.618	99.618

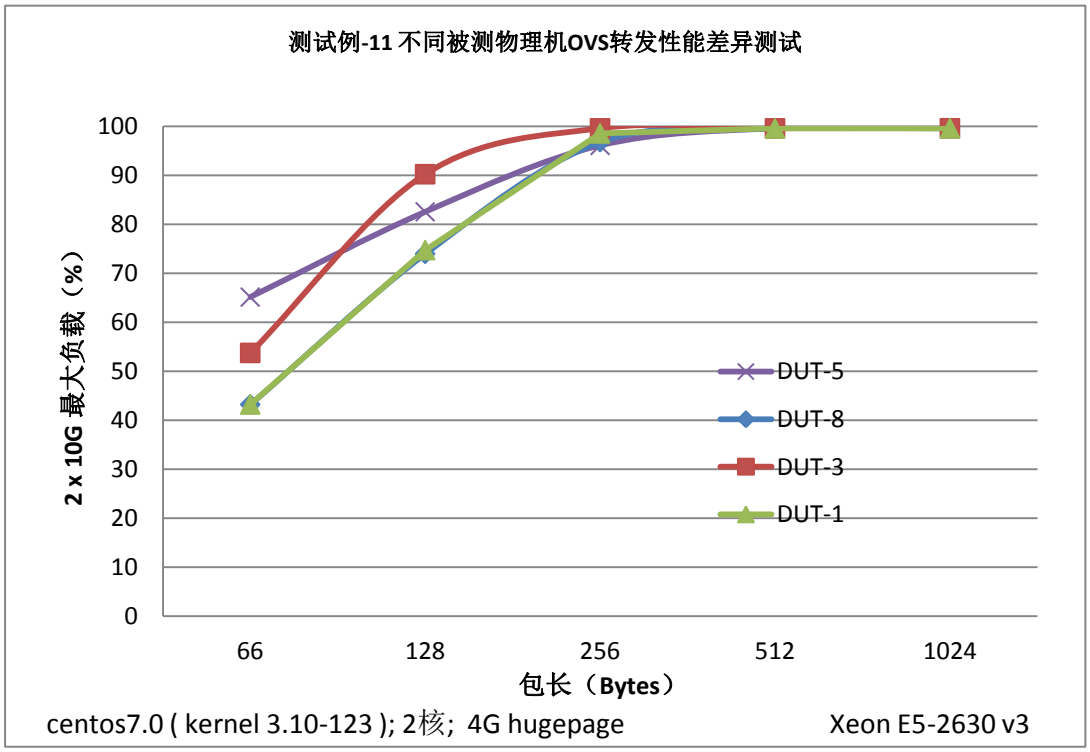

图 5－30 测试例-11 吞吐量性能比较

小结：本测试例主要考察不同厂家 DUT 在相同基本配置情况下，基于 DPDK 驱动的 OVS 转发性能，测试结果表明：中小包情况下，相比 SR-IOV 应用场景，各厂家被测设备的 OVS 转发性能下降 20%-30%。同时，被测设备间的 OVS 转发性能差异明显，相差幅度在 10%-25%之间，增加负载流量有可能进一步拉大差距。因为各厂家无法提供统一的 BIOS 配置接口，造成结果差异的原因可能是 BIOS 设置或者硬件结构的不同。后续将进一步测试增加 OVS 处理核、OVS 海量流表情况下的转发性能。

5.5.4.2不同 CPU 型号对 OVS 转发性能影响测试

测试场景

表 5－42 测试例-12 测试场景说明

测试编号	测试例-12
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2630 v3
HOST 操作系统	centos7.0
DPDK 转发核数	2 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	4G

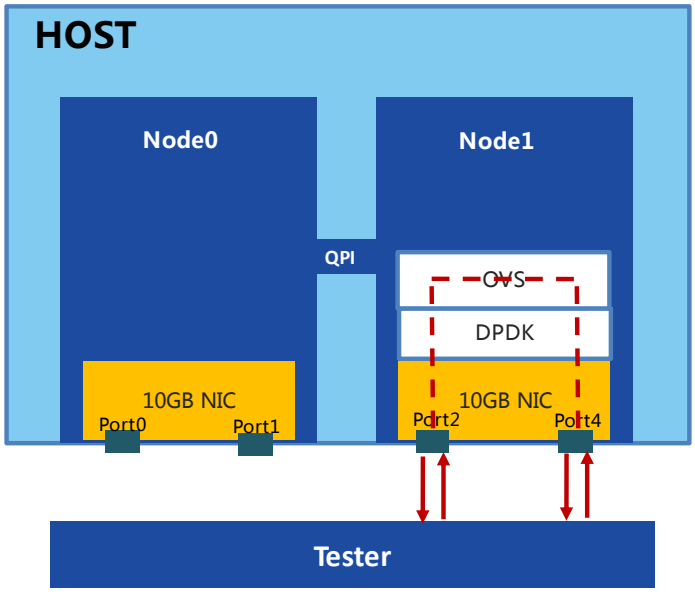

图 5－31 测试例-12 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－43 测试例-12 测试环境配置

服务器配置	DUT-2、DUT-3
服务器 bios 配置	见表 5-5“DUT-2”、“DUT-3”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 1”与“配置 2”
DPDK 编译	见附录四《DPDK 编译》
OVS 安装	见附录八《OVS 安装》
OVS 配置	见附录九《OVS 测试配置》

测试结果

表 5－44 测试例-12 吞吐量性能数据

包长	DUT-3 负载（%）	DUT-2 负载（%）
66	53.75	59.722
128	90.244	96.267
256	99.601	99.601
512	99.601	98.426
1024	99.618	99.618

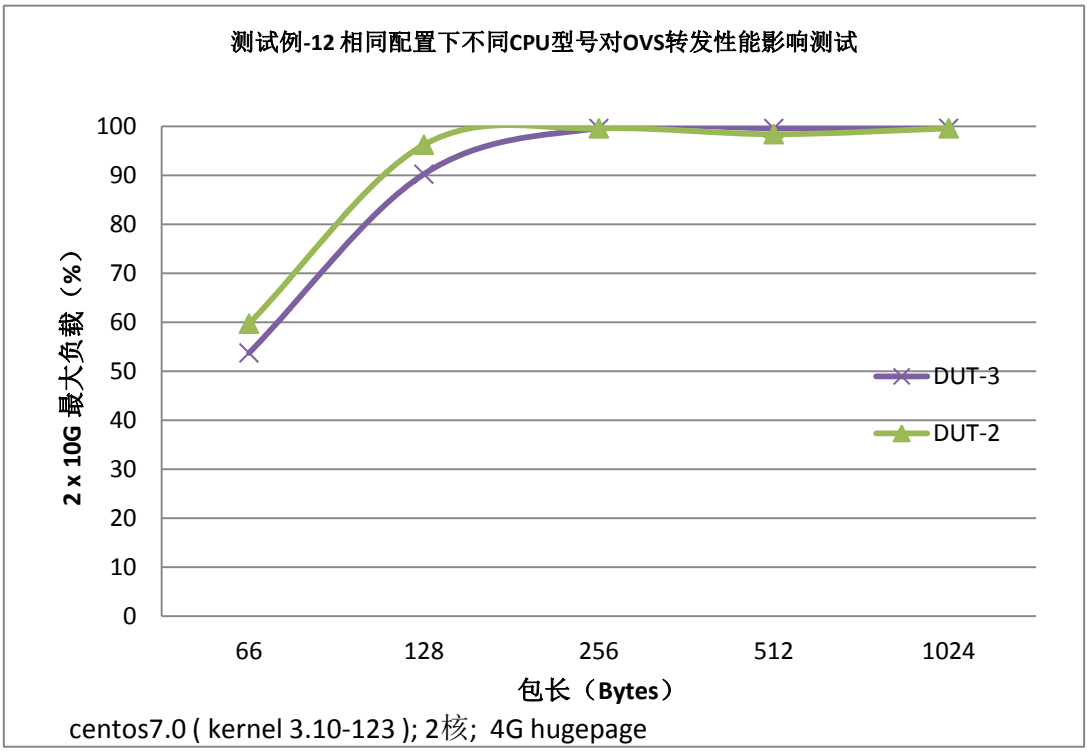

图 5－32 测试例-12 吞吐量性能比较

小结：该测试例采用同一厂家设备置换 CPU 方式进行测试，以避免 BIOS 及硬件结构差异的影响。结果表明：在 20G 吞吐量和小包情况下，不同规格的 CPU 型号（Xeon E5-2690 V3 和 Xeon E5-2630 V3）的 OVS 转发性能差异低于 6%，说明 OVS 此时出现性能瓶颈，仅通过提升 CPU 处理能力并不能显著改善 OVS 转发性能。后续需要在 OVS with DPDK-netdev 的基础上，进一步对 OVS 的转发性能进行优化工作。

其他厂家被测设备也有类似结果。

5.5.4.3不同 Linux 版本对 OVS 转发性能影响测试

测试场景

表 5－45 测试例-13 测试场景说明

测试编号	测试例-13
流量流向	Port2⇔ L3fwd ⇔Port4，2 x 10G 端口
CPU 型号	Xeon E5-2630 v3
HOST 操作系统	centos7.0、centos7.1
DPDK 转发核数	2 核
跨路情况	CPU 与网卡同 SOCKET
Hugepage 设置	4G

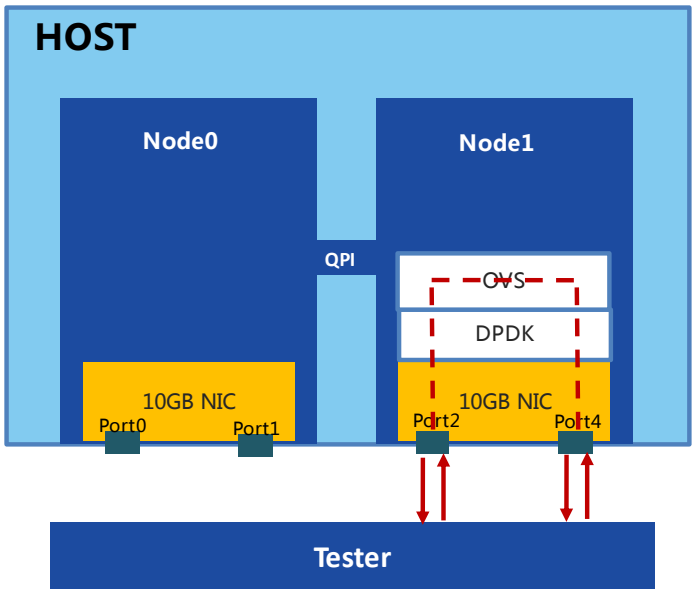

图 5－33 测试例-13 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－46 测试例-13 测试环境配置

服务器配置	DUT-3
服务器 bios 配置	见表 5-5“DUT-3”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64”与“CentOS 7.1x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 2”
DPDK 编译	见附录四《DPDK 编译》
OVS 安装	见附录八《OVS 安装》
OVS 配置	见附录九《OVS 测试配置》

测试结果

表 5－47 测试例-13 吞吐量性能数据

包长	DUT-3-centos7.0 负载（%）	DUT-3-centos7.1 负载（%）
66	53.75	56.738
128	90.244	87.557
256	99.601	98.268
512	99.601	99.601
1024	99.618	99.618

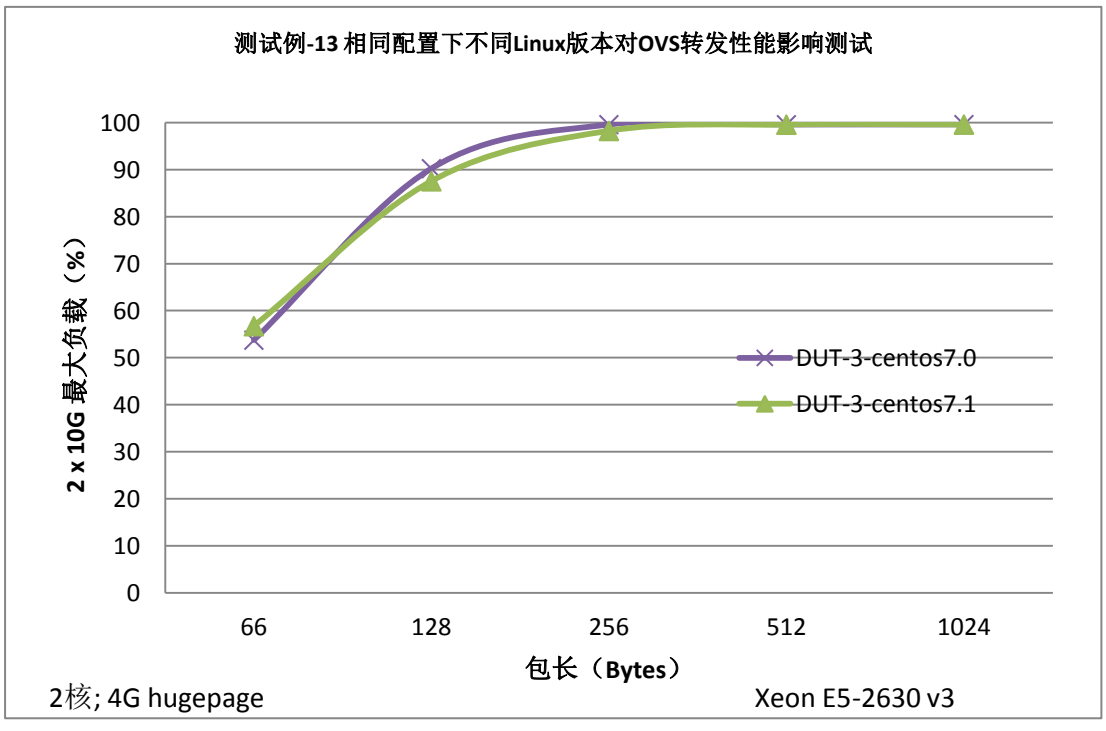

图 5－34 测试例-13 吞吐量性能比较

小结：该测试例的测试结果表明，在 20G 吞吐量情况下，同一被测设备 HOST 上安装不同的 Linux 操作系统版本，对基于 DPDK 的 OVS 转发性能无显著影响。在使用 OVS 的实现方案中，可以结合 SR-IOV 场景需求，尽量部署统一的 Linux 操作系统版本。

5.5.4.4 NUMA 对 OVS 转发性能影响测试

测试场景

表 5－48 测试例-14 测试场景说明

测试编号	测试例-14	
CPU 型号	Xeon E5-2630 v3	
HOST 操作系统	centos7.0	
DPDK 转发核数	2 核	
Hugepage 设置	4G	
跨路情况	CPU 与网卡同 SOCKET	网卡跨 SOCKET
流量流向	Port2 ⇔L3fwd⇔Port4，2 x 10G 端口	Port0 ⇔ L3fwd⇔ Port2，2 x 10G 端口
转发与连接配置图	图 5-35	图 5-36

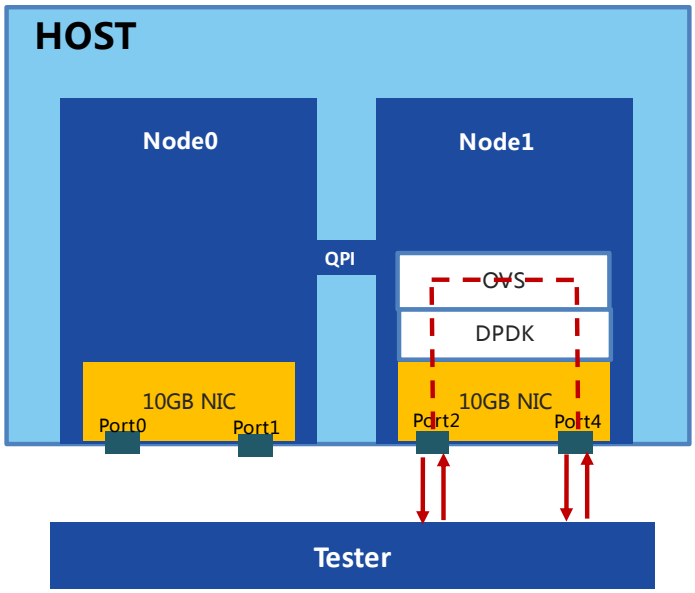

图 5－35 测试例-14 CPU 与网卡同 SOCKET 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

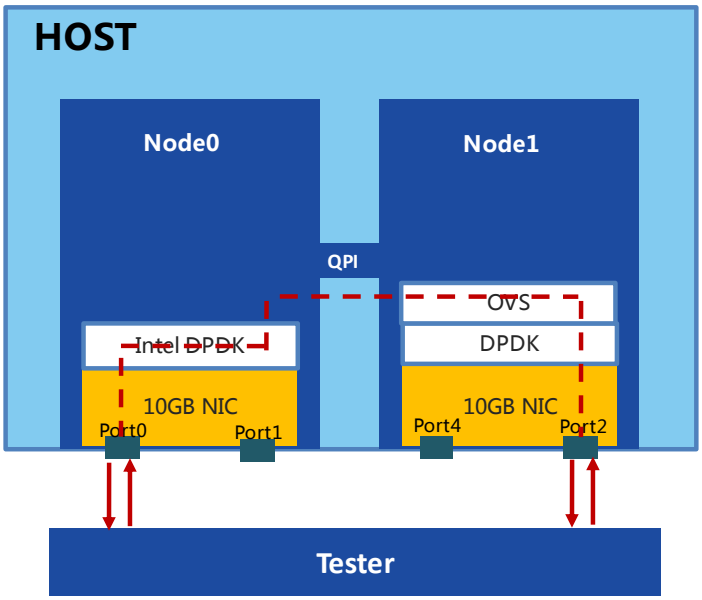

图 5－36 测试例-14 网卡跨 SOCKET 转发与连接配置

（注：Port2 与 Port4 分属不同的网卡）

测试环境配置

表 5－49 测试例-14 测试环境配置

服务器配置	DUT-8
服务器 bios 配置	见表 5-5“DUT-8”设置
操作系统安装	见附录一《操作系统安装》
操作系统配置	见表 5-6“CentOS 7.0 x86_64” 见附录二《操作系统服务关闭说明》 见附录三《操作系统启动参数》中“配置 2”
DPDK 编译	见附录四《DPDK 编译》
OVS 安装	见附录八《OVS 安装》
OVS 配置	见附录九《OVS 测试配置》

测试结果

表 5－50 测试例-14 吞吐量性能数据

包长	DUT-8-同 SOCKET 负载（%）	DUT-8-网卡跨 SOCKET 负载（%）
66	43.225	5.047
128	74	8.132
256	96.814	6.284
512	99.601	0.719
1024	99.618	5.049

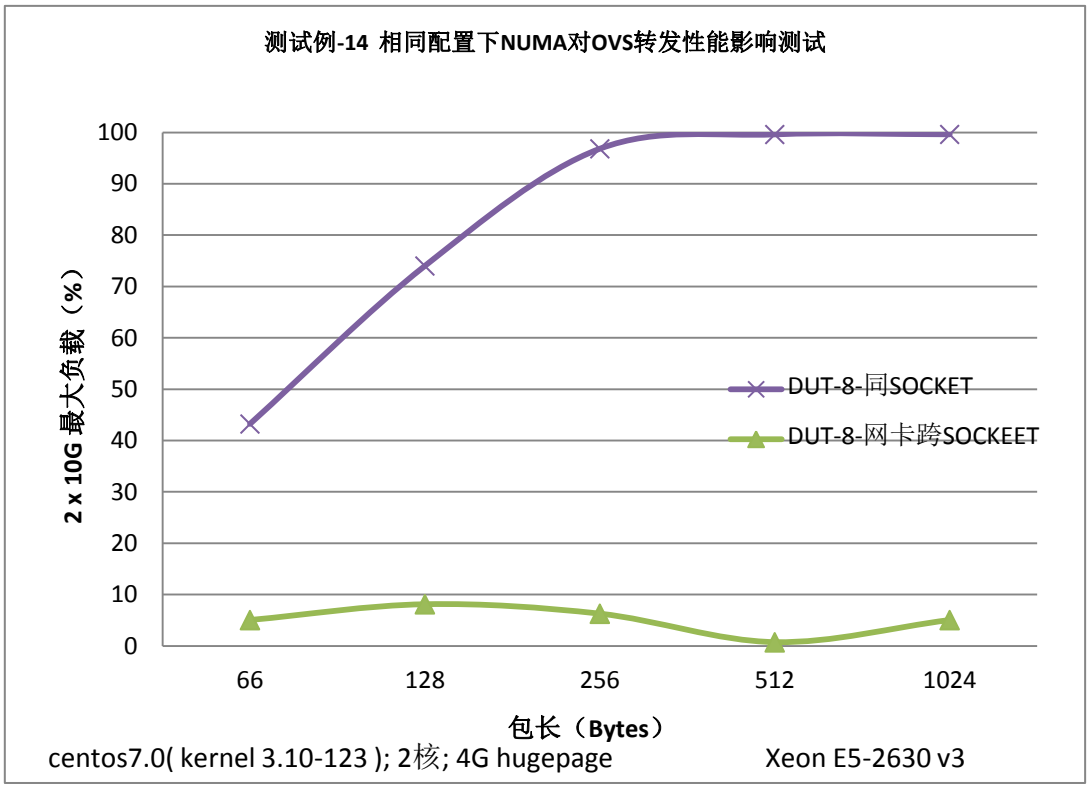

图 5－37 测试例-14 吞吐量性能比较

小结：该测试例有两种 NUMA 跨接方法：

“CPU 与网卡同 SOCKET”，如图 5-35 所示，此时 OVS 对中小包的转发性能较差；

“网卡跨 SOCKET”，如图 5-36 所示，此时 OVS 转发性能较差，在各包长情况下均出现严重丢包；

上述测试结果表明，在 20G 吞吐量情况下，NUMA 结构跨接方法对于 OVS 转发性能影响显著。实际使用中，应尽量避免使用“网卡跨 SOCKET”NUMA 跨接方法，而应采用“CPU 与网卡同 SOCKET” NUMA 跨接方法，并且双向吞吐量控制在 10G 以内。

6 参考配置

6.1 通用硬件参考配置

表 6－1 通用硬件服务器参考配置

CPU	双路 Xeon V3 CPU，单路 CPU 至少 8 核以上
内存	64G 以上，DDR4
硬盘	1T 以上
网卡	82599 兼容万兆网卡

6.2 推荐硬件 BIOS 设置

推荐服务器建议使用以下 BIOS 设置

表 6－2 BIOS 设置推荐

参数名	参数值
电源管理选项	
CPU Power and Performance Policy	Traditional
Intel Turbo boost	Disable
Processor C3	Disable
Processor C6	Disable
处理器配置	
Intel Hyper-Threading Technology (HTT)	Enable
MLC Streamer (Hardware Prefetcher)	Enable
MLC Spatial Prefetcher (Adjacent Cache Prefetch)	Enable
DCU Data Prefetcher (DCU Streamer Prefetcher)	Enable
DCU nstructionrefetcher (DCU IP Prefetcher)	Enable
虚拟化选项	
Intelirtualization Technology for Directed I/O (VT-d)	Enable
内存配置选项	
Memory RAS and Performance Configuration -> NUMA Optimized	Auto
I/O 配置	
DDIO (Direct Cache Access)	Auto

6.3 推荐 OS 及相关设置

目前建议 HOST OS 和 VM 的 GUEST OS 使用 centos7.0 系统，centos7.0 的 kernel 版本建议使用 3.10.0-123 版本。操作系统推荐开启或者关闭的服务如下表：

表 6－3 操作系统推荐开/关服务列表

序号	参数 （Parameter）	使能/非使能 (Enable/Disable)	备注 （Explanation）
1	Firewall	Disable	关闭防火墙
2	irqbalance	Disable	关闭中断平衡
3	ssh	Enable	开启ssh，便于远程连接控制
4	auditd	Disable	关闭审计服务
5	kdump	Disable	关闭kdump服务
6	bluetooth	Disable	关闭蓝牙服务
7	NetworkManager	Disable	关闭网络管理服务
8	ksm	Disable	关闭KVM的内存扫描服务
9	ksmtuned	Disable	关闭KVM的内存扫描服务

6.4 不同应用场景参考配置

6.4.1 物理机三层转发参考配置

表 6－4 物理机三层转发参考配置

跨路情况	CPU 与网卡同 SOCKET
HOST 操作系统	Centos7.0 启动时隔离 CPU，用于 L3fwd 的 CPU 亲和性设置，启动后参照附录二关闭相应服务
L3fwd 程序转发核数	2 核
L3fwd 程序使用 Hugepage	4G

详细设置命令如下：

```
#!/bin/sh
```

1) 中断迁移

```
#move IRQ
killall irqbalance
```

```
irqs=`ls /proc/irq`
for i in ${irqs} ; do
if [ -f /proc/irq/$i/smp_affinity ]; then
echo 2 > /proc/irq/$i/smp_affinity
echo "echo 2 > /proc/irq/$i/smp_affinity"
fi
done
echo "modify irq $i to CPU0 only successfully"
```

2) 使用内存分配

```
#assign hugepage

echo 0 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
echo 0 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages

echo2048> /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
echo2048> /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
```

3) 挂载巨页

```
#mount hugepage
umount /mnt/huge
rm -R /mnt/huge
mkdir -p /mnt/huge
mount -t hugetlbfs nodev /mnt/huge
```

4) 卸载 IGB_UIO 驱动

```
#unloads igb_uio.ko
sudo /sbin/rmmod igb_uio
```

5) 加载 IGB_UIO 驱动

```
#load igb_uio
/sbin/modprobe uio
/sbin/insmod /root/dpdk-1.8.0/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko
```

6) 为相应网卡端口绑定驱动

```
#bind_nics_to_igb_uio
/root/dpdk-1.8.0/tools/dpdk_nic_bind.py -b igb_uio 80:00.0 81:00.0
```

7) 运行 l3fwd 程序

```
#run_l3fwd
/root/dpdk-1.8.0/examples/l3fwd/build/l3fwd -c 0x1000 -n 4 -- -P -p 0x3
--config="(0,0,12),(1,0,12)" --hash-entry-num=0x13c68
```

6.4.2 SR-IOV 推荐设置

表 6－5 SR-IOV 推荐设置

跨路情况	CPU 与网卡同 SOCKET
HOST 操作系统	Centos7.0 启动时隔离 CPU，用于 VM 的 CPU 亲和性设置，启动后参照附录二关闭相应服务
VM 配置要求	CPU4 核以上、内存 6G 以上、硬盘大小 20G 以上
VM GUEST 操作系统	Centos7.0 启动时隔离 CPU，用于 App 的 CPU 亲和性设置，启动后参照附录二关闭相应服务
L3fwd 程序（VM）转发核数	2 核
L3fwd 程序（VM）使用 Hugepage	4G

详细设置命令如下：

```
#!/bin/sh
```

1) 设置网卡端口的虚拟端口数

```
echo 1 > /sys/bus/pci/devices/0000\:81\:00.0/sriov_numvfs
echo 1 > /sys/bus/pci/devices/0000\:85\:00.0/sriov_numvfs
```

2) 设置虚拟网卡 MAC 地址

```
ip link set dev p1p1 vf 0 mac 00:00:00:00:00:01
ip link set dev p2p1 vf 0 mac 00:00:00:00:00:02
```

3) 加载 vfio 及 vfio-pci 驱动

```
modprobe vfio
modprobe vfio-pci
sleep 1;
```

4) 解绑 VF 上的 IXGBEVF 驱动

```
echo 0000:81:10.0 > /sys/bus/pci/devices/0000\:81\:10.0/driver/unbind
echo 0000:85:10.0 > /sys/bus/pci/devices/0000\:85\:10.0/driver/unbind
echo "8086 10ed" > /sys/bus/pci/drivers/vfio-pci/new_id
```

5) 挂载巨页

```
mkdir -p /mnt/huge1g
mount -t hugetlbfs nodev /mnt/huge1g
```

6) 启动虚拟机

```
screen -S centos    numactl    --membind=1 --CPUnodebind=1    qemu-system-x86_64 -hda
/home/IMG/centos7.0.img -m 6144 -smp 4    -CPU host -enable-kvm -mem-path /mnt/huge1g
-vnc :2 -monitor stdio \
    -device vfio-pci,host=0000:81:10.0 \
    -device vfio-pci,host=0000:85:10.0 \
    -name centos7.0-test \
```

7) 虚拟机成功启动后使用 taskset 命令将虚拟机的 vCPU 线程绑定至相应逻辑核

查看虚拟机 vCPU 线程号的示例如下：

```
~
"/home/vm_scripts/centos7.1_start_lp.sh" 10L, 762C written
[root@localhost ~]# ./SRIOV_start_lp.sh
device br0 already exists; can't create bridge with the same name
QEMU 1.6.2 monitor - type 'help' for more information
(qemu) info cpus
* CPU #0: pc=0xffffffff81052dd6 (halted) thread_id=6610
  CPU #1: pc=0xffffffff81052dd6 (halted) thread_id=6611
  CPU #2: pc=0xffffffff81052dd6 (halted) thread_id=6612
  CPU #3: pc=0xffffffff81052dd6 (halted) thread_id=6613
(qemu)
```

使用 taskset –cp CPU 编号线程号绑定对应核，命令格式为：taskset–pc 20 6610

```
[root@localhost ~]# taskset -pc 20 6610
pid 6610's current affinity list: 6-11,18-23
pid 6610's new affinity list: 20
[root@localhost ~]#
```

8) 进入虚拟机运行 l3fwd 程序

详见 6.4.1 命令配置。

6.4.3 OVS 推荐设置

表 6－6 OVS 推荐设置

跨路情况	CPU 与网卡同 SOCKET
HOST 操作系统	Centos7.0 启动时隔离 CPU，用于 OVS 的 CPU 亲和性设置，启动后参照附录二关闭相应服务
OVS 转发核数	2 核
OVS 使用 Hugepage	4G

详细命令如下：

```
#!/bin/sh
```

1) 中断迁移

```
#remove IRQ
killall irqbalance
irqs=`ls /proc/irq`
for i in ${irqs} ; do
if [ -f /proc/irq/$i/smp_affinity ]; then
echo 2 > /proc/irq/$i/smp_affinity
echo "echo 2 > /proc/irq/$i/smp_affinity"
fi
done
echo "modify irq $i to CPU0 only successfully"
```

2) 删除 OVS 旧配置

```
#remove old ovs configuration
if [[ "1" == "1"    ]];then
        rm -rf /usr/local/var/run/openvswitch/
        rm -rf /usr/local/etc/openvswitch/
```

```
        rm -f /tmp/conf.db
        mkdir -p /usr/local/etc/openvswitch
        mkdir -p /usr/local/var/run/openvswitch/
fi
```

3) 加载驱动

```
#init kernel
modprobe cuse
modprobe fuse
modprobe vxlan;
modprobe gre;
modprobe openvswitch;
```

4) 创建 OVS 相关启动文件

#create ovsdb file

```
/opt/ovs/ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db
/opt/ovs/vswitchd/vswitch.ovsschema
```

5) 启动 OVSDB

#start ovsdb-server

```
/opt/ovs/ovsdb/ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock
--remote=db:Open_vSwitch,Open_vSwitch,manager_options
--private-key=db:Open_vSwitch,SSL,private_key --certificate=db:Open_vSwitch,SSL,certificate
--bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert    --detach

#the ovsdb logs in /usr/local/var/log/openvswitch/ovsdb-server.log
```

6) 加载 IGB_UIO 驱动

```
#load ivshmem igb_uio drive
modprobe uio
insmod /root/dpdk-1.8.0/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko
modprobe cuse
modprobe fuse
rm -rf /dev/vhost-net
```

7) 绑定 IGB_UIO 驱动

#bind nic to igb_uio

```
/root/dpdk-1.8.0/tools/dpdk_nic_bind.py --bind=igb_uio 81:00.0
/root/dpdk-1.8.0/tools/dpdk_nic_bind.py --bind=igb_uio 85:00.0
```

8) 巨页挂载

```
#mount hugepage
umount /dev/hugepages
rm -R /dev/hugepages
mkdir /dev/hugepages
```

```
mount -t hugetlbfs nodev /dev/hugepages

umount /dev/hugepages_2mb
rm -R /dev/hugepages_2mb
mkdir /dev/hugepages_2mb
mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
```

9) 启动 OVS 主进程

```
#start vswitchd
/opt/ovs/utilities/ovs-vsctl --no-wait init
/opt/ovs/vswitchd/ovs-vswitchd --dpdk -c 0xc000 -n 4 --socket-mem 4096,4096    --
unix:/usr/local/var/run/openvswitch/db.sock    --detach
```

10) 设置 OVS 转发线程的亲核性

#vswitchd affinity to core

```
/opt/ovs/utilities/ovs-vsctl set Open_vSwitch . other_config:pmd-CPU-mask=c000
```

11) 虚拟桥创建

```
#add br
/opt/ovs/utilities/ovs-vsctl del-br br0
/opt/ovs/utilities/ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
/opt/ovs/utilities/ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk
/opt/ovs/utilities/ovs-vsctl add-port br0 dpdk1 -- set Interface dpdk1 type=dpdk
```

12) 流表创建

```
#add flow
/opt/ovs/utilities/ovs-ofctl del-flows br0
/opt/ovs/utilities/ovs-ofctl add-flow br0 in_port=1,action=2
/opt/ovs/utilities/ovs-ofctl add-flow br0 in_port=2,action=1
```

7 缩略语

缩写	英文全称	中文全称
BNG	Broadband Network Gateway	宽带网络网关控制设备
DPI	deep packet inspection	深度包检测
EAL	Environment Abstraction Layer	环境抽象层
EPC	Evolved Packet Core	演进的分组核心网
ETSI	European Telecommunications Standards Institute	欧洲电信标准化协会
Intel AVX	Intel Advanced Vector Extensions	英特尔高级矢量扩展指令
Intel DPDK	Intel Data Plane Development Kit	英特尔数据平面开发套件
Intel SSE/AVX	Intel Streaming SIMD Extensions/Advanced Vector Extensions	英特尔SIMD流指令扩展/高级矢量扩展指令
Intel VT	Intel Virtualization Technology	英特尔虚拟化技术
MMIO	Memory mapping I/O	内存映射 I/O
MMU	Memory Management Unit	内存管理单元
NAT	Network Address Translation	网络地址转换
NFV	Network Function Virtualization	网络功能虚拟化
NIC	Network Interface Card	网络适配器
NUMA	Non Uniform Memory Access Architecture	非统一内存访问架构
OVS	OpenvSwitch	OVS 软件交换机
PF	Physical Function	物理功能
PMD	Polling Mode Driver	轮询模式驱动
QPI	Quick Path Interconnect	快速通道互联
SR-IOV	Single-Root I/O Virtualization	单根 I/O 虚拟化
TLB	Translation Lookaside Buffer	地址转译缓冲器
UIO	Userspace I/O	运行在用户空间的I/O技术
VF	Virtual Function	虚拟化功能
VM	Virtual Machine	虚拟主机
VNF	Virtual Network Function	虚拟网络功能

8 参考文献

[1] Best Practices for Performance Tuning of Telco and NFV Workloads in vSphere, https://www.vmware.com/resources/techresources/10479

[2] Intel® Open Network Platform Server (Release 1.2) Benchmark Performance Test Report, http://www.dpdk.org

[3] Intel® Open Network Platform Server (Release 1.3.1) Benchmark Performance Test Report,http:// www.dpdk.org

[4] ETSI GS NFV-PER 001 V1.1.1 (2014-06) Network Functions Virtualisation (NFV); NFV Performance & Portability Best Practises, http://www.etsi.org/deliver/etsi_gs/NFV-PER/001_099/001/01.01.01_60/gs_NFV-PER001v010101p.pdf

[5] RFC2544, Benchmarking Methodology for Network Interconnect Devices, http://tools.ietf.org/html/rfc2544

9 鸣谢

本文撰写和相关测试工作中，Intel 公司、武汉绿网公司、思博伦公司提供了技术指导和测试建议，戴尔、华为、惠普、联想、中兴等提供了相关测试设备和技术支持，对这些单位的合作与支持表示衷心感谢！

附录

附录一：操作系统安装

使用相应操作系统镜像安装操作系统，选择如下软件包：
C Development Tools and Libraries
Development Tools
RPM Development Tools
System Tools

附录二：操作系统服务关闭说明

在虚拟机及物理机的操作系统安装后，需要关闭一下服务以提升转发性能：
1、关闭中断服务调度程序：
systemctl disable irqbalance.service
2、关闭防火墙服务：
systemctl disable firewalld.service
3、关闭审计服务：
systemctl disable auditd.service
4、关闭 kdump：
systemctl disable kdump.service
5、关闭蓝牙服务：
systemctl disable bluetooth.service
6、关闭 KVM 的内存扫描服务
systemctl disable ksm.service
systemctl disable ksmtuned.service
7.关闭网络管理服务
systemctl disable NetworkManager.service

附录三：操作系统启动参数

序号	配置名	操作系统启动参数配置	备注
1	配置 1	default_hugepagesz=1Ghugepagesz=1G hugepages=20 hugepagesz=2M hugepages=4096 iommu=pt intel_iommu=on isolCPUs=8,9,10,11,20,21,22,23	使用于 CPU 为 E5-2690v3 服务器
2	配置 2	default_hugepagesz=1Ghugepagesz=1G hugepages=20 hugepagesz=2M hugepages=4096 iommu=pt intel_iommu=on isolCPUs=4,5,6,7,12,13,14,15	使用于 CPU 为 E5-2630v3 服务器
3	配置 3	default_hugepagesz=1Ghugepagesz=1G hugepages=20 hugepagesz=2M hugepages=4096 iommu=pt intel_iommu=on isolCPUs=12,13,14,15,16,17,30,31,32,33,34,35	使用于 CPU 为 E5-2699v3 服务器
4	配置 4	default_hugepagesz=1Ghugepagesz=1G hugepages=20 hugepagesz=2M hugepages=4096 iommu=pt intel_iommu=on isolCPUs=12,13,14,15,16,17,30,31,32,33,34,35	使用于 CPU 为 E5-2650v2 服务器
5	配置 5	hugepagesz=2M hugepages=2048 isolCPUs=1,2,3	使用于虚拟机

注：

1. 需要将以上配置添加至 grub 的启动参数配置文件（/etc/default/grub）中的 GRUB_CMDLINE_LINUX 参数后，如

```
GRUB_CMDLINE_LINUX="vconsole.keymap=us crashkernel=auto
vconsole.font=latarcyrheb-sun16 rhgb quiet default_hugepagesz=1G hugepagesz=1G
hugepages=20 hugepagesz=2M hugepages=2048 iommu=pt intel_iommu=on
isolCPUs=8,9,10,11,20,21,22,23"
```

2.重新生成启动文件

```
grub2-mkconfig –o /boot/grub2/grub.cfg
```

3.重启系统后查看/proc/cmdline 文件，以确定以上配置是否生效，如

```
cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-3.10.0-123.el7.x86_64
root=UUID=a7dcd7eb-fa1d-485b-9df4-9018dc7c5628 ro vconsole.keymap=us crashkernel=auto
vconsole.font=latarcyrheb-sun16 rhgb quiet default_hugepagesz=1G hugepagesz=1G
hugepages=20 hugepagesz=2M hugepages=2048 iommu=pt intel_iommu=on
isolCPUs=8,9,10,11,20,21,22,23
```

附录四：DPDK 编译

1. 下载 DPDK1.8 版本

```
cd /root/
```

wgethttp://dpdk.org/browse/dpdk/snapshot/dpdk-1.8.0.tar.gz

2. 解压 DPDK 源码

```
tar xf dpdk-1.8.0.tar.gz
```

3. 编译 DPDK：

```
cd /root/dpdk-1.8.0
make config T=x86_64-native-linuxapp-gcc
make
```

附录五：DPDK 软件启动配置

启动 DPDK 软件使用以下脚本文件

```
#!/bin/sh
```

1.中断迁移

```
#move IRQ
killall irqbalance
irqs=`ls /proc/irq`
for i in ${irqs} ; do
if [ -f /proc/irq/$i/smp_affinity ]; then
echo 2 > /proc/irq/$i/smp_affinity
echo "echo 2 > /proc/irq/$i/smp_affinity"
fi
done
echo "modify irq $i to CPU0 only successfully"
```

2.使用内存分配

```
#assign hugepage

echo 0 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
echo 0 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages

echo %M1%> /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
echo %M2%> /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
```

注：如上命令为每个 numanode 分配 1024*2m 内存，按实际测试用例修改内存分配。

```
echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
echo 1024 > /sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
```

3.挂载巨页

```
#mount hugepage
umount /mnt/huge
rm -R /mnt/huge
mkdir -p /mnt/huge
mount -t hugetlbfs nodev /mnt/huge
```

4.卸载 IGB_UIO 驱动

```
#unloads igb_uio.ko
sudo /sbin/rmmod igb_uio
```

5.加载 IGB_UIO 驱动

```
#load igb_uio
/sbin/modprobe uio
/sbin/insmod /root/dpdk-1.8.0/x86_64-native-linuxapp-gcc/kmod/igb_uio.ko
```

6.为相应网卡端口绑定驱动

```
#bind_nics_to_igb_uio
```

/root/dpdk-1.8.0/tools/dpdk_nic_bind.py -b igb_uio %P1%%P2%

注：如上命令为网卡端口绑定 UIO 驱动，按测试时实际总线号替换 P1 及 P2 参数以绑定驱动，如

```
/root/dpdk-1.8.0/tools/dpdk_nic_bind.py -b igb_uio 80:00.0 81:00.0
```

7.运行 l3fwd 程序

#run_l3fwd

```
/root/dpdk-1.8.0/examples/l3fwd/build/l3fwd -c %C1% -n 4 -- -P -p 0x3
--config="(0,0,%C2%),(1,0,%C3%)" --hash-entry-num=%F1%
```

注：1.”-c”参数指定具体使用的转发核，按测试用例修改参数 C1

C1 为 CPU 编号掩码，例如 0X100 表示使用第 9 个逻辑核

2.”-config”参数指定网卡端口使用的转发核，按测试用例修改参数 C2,C3。

C2 用于指定 0 号端口使用的转发核

C3 用于指定 1 号端口使用的转发核

3.“--hash-entry-num”参数配置测试使用的流表数 F1

```
/root/dpdk-1.8.0/examples/l3fwd/build/l3fwd -c 0x1000 -n 4 -- -P -p 0x3
--config="(0,0,12),(1,0,12)" --hash-entry-num=0x13c68
```

附录六：L3fwd 程序编译

1.修改 L3fwd 转发程序默认流表数

```
cd /root/dpdk-1.8/example/l3fwd
```

修改代码：

```
staticstruct ipv4_l3fwd_route ipv4_l3fwd_route_array[] = {
        {{IPv4(101,0,0,0), IPv4(100,10,0,1),    101, 11, IPPROTO_TCP}, 0},
        {{IPv4(201,0,0,0), IPv4(200,20,0,1),    102, 12, IPPROTO_TCP}, 1},
        {{IPv4(111,0,0,0), IPv4(100,30,0,1),    101, 11, IPPROTO_TCP}, 2},
        {{IPv4(211,0,0,0), IPv4(200,40,0,1),    102, 12, IPPROTO_TCP}, 3},
};
```

为：

```
staticstruct ipv4_l3fwd_route ipv4_l3fwd_route_array[] = {
        {{IPv4(101,0,0,0), IPv4(100,10,0,1),    101, 11, IPPROTO_UDP}, 0},
        {{IPv4(201,0,0,0), IPv4(200,20,0,1),    101, 11, IPPROTO_UDP}, 1},
        {{IPv4(111,0,0,0), IPv4(100,30,0,1),    101, 11, IPPROTO_UDP}, 2},
        {{IPv4(211,0,0,0), IPv4(200,40,0,1),    101, 11, IPPROTO_UDP}, 3},
};
```

2.设置 DPDK 环境变量

```
export RTE_SDK=/root/dpdk-1.8
```

3.编译 L3fwd

```
make
```

附录七：SR-IOV 测试配置

```
#!/bin/sh
```

1.设置网卡端口的虚拟端口数

```
echo 1 > /sys/bus/pci/devices/0000\:%P1%/sriov_numvfs
echo 1 > /sys/bus/pci/devices/0000\:%P2%/sriov_numvfs
```

注：修改网卡端口总线号参数 P1 和 P2 为测试时使用的网卡端口总线号，如

```
echo 1 > /sys/bus/pci/devices/0000\:81\:00.0/sriov_numvfs
echo 1 > /sys/bus/pci/devices/0000\:85\:00.0/sriov_numvfs
```

2.设置虚拟网卡 MAC 地址

```
ip link set dev %N1% vf 0 mac %M1%
ip link set dev %N2% vf 0 mac %M2%
```

注：修改网卡名参数 N1 和 N2(如，p1p1)及相应 MAC 地址参数 M1 和 M2，如

```
ip link set dev p1p1 vf 0 mac 00:00:00:00:00:01
ip link set dev p2p1 vf 0 mac 00:00:00:00:00:02
```

3.加载 vfio 及 vfio-pci 驱动

```
modprobe vfio
modprobe vfio-pci

sleep 1;
```

4.解绑 VF 上的 IXGBEVF 驱动

```
echo 0000:%P3%> /sys/bus/pci/devices/0000\:%P4%/driver/unbind
echo 0000:%P5%> /sys/bus/pci/devices/0000\:%P6%/driver/unbind
echo "8086 10ed" > /sys/bus/pci/drivers/vfio-pci/new_id
```

注：将 VF 总线号参数 P3,P4,P5,P6 修改为相应测试环境总线号，如

```
echo 0000:81:10.0 > /sys/bus/pci/devices/0000\:81\:10.0/driver/unbind
echo 0000:85:10.0 > /sys/bus/pci/devices/0000\:85\:10.0/driver/unbind
```

4.挂载巨页

```
mkdir -p /mnt/huge1g
mount -t hugetlbfs nodev /mnt/huge1g
```

5.启动虚拟机

```
screen -S centos    numactl    --membind=%NM1% --CPUnodebind=%NC1%
qemu-system-x86_64 -hda %P7% -m %M3% -smp %C1%    -CPU host -enable-kvm -mem-path
/mnt/huge1g    -vnc :2 -monitor stdio \
-device vfio-pci,host=0000:%P3% \
    -device vfio-pci,host=0000:%P5% \
    -name centos7.0-test \
```

注：1.参数 NM1 指定虚拟机内存所在的 numanode

2.参数 NC1 指定虚拟机 vCPU 线程所在的 numanode

3.参数 P7 虚拟机镜像路径

4.参数 M3 虚拟机内存大小

5.参数 C1 虚拟机 CPU 个数

6.参数 P3 和 P5 为虚拟机使用的 VF 总线号

如

```
screen -S centos    numactl    --membind=1 --CPUnodebind=1    qemu-system-x86_64 -hda
/home/IMG/centos7.0.img -m 6144 -smp 4    -CPU host -enable-kvm -mem-path /mnt/huge1g
-vnc :2 -monitor stdio \
    -device vfio-pci,host=0000:81:10.0 \
    -device vfio-pci,host=0000:85:10.0 \
    -name centos7.0-test \
```

6.虚拟机成功启动后使用 taskset 命令将虚拟机的 vCPU 线程绑定至相应逻辑核

查看虚拟机 vCPU 线程号的示例如下：

```
~
"/home/vm_scripts/centos7.1_start_lp.sh" 10L, 762C written
[root@localhost ~]# ./SRIOV_start_lp.sh
device br0 already exists; can't create bridge with the same name
QEMU 1.6.2 monitor - type 'help' for more information
(qemu) info cpus
* CPU #0: pc=0xffffffff81052dd6 (halted) thread_id=6610
  CPU #1: pc=0xffffffff81052dd6 (halted) thread_id=6611
  CPU #2: pc=0xffffffff81052dd6 (halted) thread_id=6612
  CPU #3: pc=0xffffffff81052dd6 (halted) thread_id=6613
(qemu)
```

使用 taskset –cp CPU 编号线程号绑定对应核，命令格式为：taskset–pc 20 6610

```
[root@localhost ~]# taskset -pc 20 6610
pid 6610's current affinity list: 6-11,18-23
pid 6610's new affinity list: 20
[root@localhost ~]#
```

7.进入虚拟机运行 l3fwd 程序

附录八：OVS 安装

1.下载 OVS 2.3.9 版本

```
cd /opt
git clone https://github.com/openvswitch/ovs.git
cd ovs
git checkout 15c69d2d8f52669a57dd49827fe4e585ebdda105
```

2.重编译 DPDK

设置 DPDK 目录绝对路径环境变量

```
export DPDK_DIR=/root/dpdk-1.8
cd $DPDK_DIR
```

修改 DPDK 编译配置参数文件 `config/common_linuxapp` 中参数

CONFIG_RTE_BUILD_COMBINE_LIBS=y

编译 DPDK

```
make install T=x86_64-ivshmem-linuxapp-gcc
```

3.编译 OVS

设置编译后 DPDK 目录绝对路径环境变量

```
export DPDK_BUILD=$DPDK_DIR/x86_64-ivshmem-linuxapp-gcc/
```

设置 OVS 目录绝对路径环境变量

```
export OVS_DIR=/opt/ovs
```

编译 OVS

```
cd $(OVS_DIR)
```

./boot.sh

```
./configure --with-dpdk=$DPDK_BUILD [CFLAGS="-g -O2 -Wno-cast-align"]
make
```

附录九：OVS 测试配置

```
#!/bin/sh
```

1.中断迁移

```
#remove IRQ
killall irqbalance
irqs=`ls /proc/irq`
for i in ${irqs} ; do
if [ -f /proc/irq/$i/smp_affinity ]; then
echo 2 > /proc/irq/$i/smp_affinity
echo "echo 2 > /proc/irq/$i/smp_affinity"
fi
done
echo "modify irq $i to CPU0 only successfully"
```

2.删除 OVS 旧配置

```
#remove old ovs configuration
if [[ "1" == "1"    ]];then
        rm -rf /usr/local/var/run/openvswitch/
        rm -rf /usr/local/etc/openvswitch/
        rm -f /tmp/conf.db
        mkdir -p /usr/local/etc/openvswitch
        mkdir -p /usr/local/var/run/openvswitch/
fi
```

3.加载驱动

```
#init kernel
modprobe cuse
modprobe fuse
modprobe vxlan;
modprobe gre;
modprobe openvswitch;
```

4.创建 OVS 相关启动文件

#create ovsdb file

```
/opt/ovs/ovsdb/ovsdb-tool create /usr/local/etc/openvswitch/conf.db
/opt/ovs/vswitchd/vswitch.ovsschema
```

5.启动 OVSDB

#start ovsdb-server

```
/opt/ovs/ovsdb/ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock
--remote=db:Open_vSwitch,Open_vSwitch,manager_options
```

```
--private-key=db:Open_vSwitch,SSL,private_key --certificate=db:Open_vSwitch,SSL,certificate
--bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert    --detach

#the ovsdb logs in /usr/local/var/log/openvswitch/ovsdb-server.log
```

6.加载 IGB_UIO 驱动

```
#load ivshmem igb_uio drive
modprobe uio
insmod /root/dpdk-1.8.0/x86_64-ivshmem-linuxapp-gcc/kmod/igb_uio.ko
modprobe cuse
modprobe fuse
rm -rf /dev/vhost-net
```

7.绑定 IGB_UIO 驱动

#bind nic to igb_uio

```
/root/dpdk-1.8.0/tools/dpdk_nic_bind.py --bind=igb_uio %P1%
```

/root/dpdk-1.8.0/tools/dpdk_nic_bind.py --bind=igb_uio %P2%

注：将网口总线号修改为实际测试环境所使用网口总线号

/root/dpdk-1.8.0/tools/dpdk_nic_bind.py --bind=igb_uio 81:00.0

```
/root/dpdk-1.8.0/tools/dpdk_nic_bind.py --bind=igb_uio 85:00.0
```

8.巨页挂载

```
#mount hugepage
umount /dev/hugepages
rm -R /dev/hugepages
mkdir /dev/hugepages
mount -t hugetlbfs nodev /dev/hugepages

umount /dev/hugepages_2mb
rm -R /dev/hugepages_2mb
mkdir /dev/hugepages_2mb
mount -t hugetlbfs nodev /dev/hugepages_2mb -o pagesize=2MB
```

9.启动 OVS 主进程

```
#start vswitchd
/opt/ovs/utilities/ovs-vsctl --no-wait init
/opt/ovs/vswitchd/ovs-vswitchd --dpdk -c %C1% -n 4 --socket-mem %M1%,%M2%    --
unix:/usr/local/var/run/openvswitch/db.sock    --detach
```

注：1.参数 C1 是 OVS 转发线程使用的逻辑核掩码
2.参数 M1 为 numanode0 分配的内存
3.参数 M2 为 numanode1 分配的内存

```
/opt/ovs/vswitchd/ovs-vswitchd --dpdk -c 0xc000 -n 4 --socket-mem 4096,4096    --
unix:/usr/local/var/run/openvswitch/db.sock    --detach
```

10.设置 OVS 转发线程的亲核性

#vswitchd affinity to core

```
/opt/ovs/utilities/ovs-vsctl set Open_vSwitch . other_config:pmd-CPU-mask=%C2%
```

注：参数 C2 为 OVS 转发线程所使用的逻辑核掩码，例如 0XC000 表示使用第 11 和 12 个逻辑核，如

```
/opt/ovs/utilities/ovs-vsctl set Open_vSwitch . other_config:pmd-CPU-mask=c000
```

11.虚拟桥创建

```
#add br
/opt/ovs/utilities/ovs-vsctl del-br br0
/opt/ovs/utilities/ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
/opt/ovs/utilities/ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk
/opt/ovs/utilities/ovs-vsctl add-port br0 dpdk1 -- set Interface dpdk1 type=dpdk
```

12 流表创建

```
#add flow
/opt/ovs/utilities/ovs-ofctl del-flows br0
/opt/ovs/utilities/ovs-ofctl add-flow br0 in_port=1,action=2
/opt/ovs/utilities/ovs-ofctl add-flow br0 in_port=2,action=1
```