



## DPDK 全面分析

本文首发于我的公众号 **Linux云计算网络 (id: cloud\_dev)**，专注于干货分享，号内有 **10T** 书籍和视频资源，后续持续更新，欢迎大家关注，二维码文末可以扫。

### 高性能网络技术

随着云计算产业的异军突起，网络技术的不断创新，越来越多的网络设备基础架构逐步向基于通用处理器平台的架构演进，从扁平化的网络结构到基于 SDN 分层的网络结构，无不体现出这种创新与融合。

这在使得网络变得更加可控制和成本更低的同时，也能够支持大规模用户或应用程序的性能需求，以及海量数据的性能网络编程技术随着网络架构的演进不断突破的一种必然结果。

### C10K 到 C10M 问题的演进

如今，关注的更多是 C10M 问题（即单机 1 千万个并发连接问题）。很多计算机领域的大佬们从硬件上和软件上都提出了多种解决方案。从硬件上，比如说，现在的类似很多 40Gbps、32-cores、256G RAM 这样配置的 X86 服务器完全可以处理 1 千万个以上的并发连接。

但是从硬件上解决问题就没多大意思了，首先它成本高，其次不通用，最后也没什么挑战，无非就是堆砌硬件而已。所以，抛开硬件不谈，我们看看从软件上该如何解决这个世界难题呢？

这里不得不提一个人，就是 Errata Security 公司的 CEO Robert Graham，他在 Shmoocon 2013 大会上很巧妙地解释了这个问题。有兴趣可以查看其 YouTube 的演进视频：[C10M Defending The Internet At Scale](#)。



他提到了 UNIX 的设计初衷其实为电话网络的控制系统而设计的，而不是一般的服务器操作系统，所以，它仅仅是一个数据负责数据传送的系统，没有所谓的控制层面和数据层面的说法，不适合处理大规模的网络数据包。最后他得出的结论是：

OS 的内核不是解决 C10M 问题的办法，恰恰相反 OS 的内核正是导致 C10M 问题的关键所在。

### 为什么这么说？基于 OS 内核的数据传输有什么弊端？

**1、中断处理。**当网络中大量数据包到来时，会产生频繁的设备中断请求，这些设备中断可以打断之前较低优先级的软中断或者系统调用的执行过程，如果这种打断频繁的话，将会产生较高的性能开销。

**2、内存拷贝。**正常情况下，一个网络数据包从网卡到应用程序需要经过如下的过程：数据从网卡通过 DMA 等方式传到内核开辟的缓冲区，然后从内核空间拷贝到用户态空间，在 Linux 内核协议栈中，这个耗时操作甚至占到了数据包整个处理流程的 57.1%。

**3、上下文切换。**频繁到达的设备中断和软中断都可能随时抢占系统调用的运行，这会产生大量的上下文切换开销。另外，在基于多线程的服务

**4、局部性失效。**如今主流的处理器的都是多个核心的，这意味着一个数据包的处理可能跨多个 CPU 核心，比如一个数据包可能中断在 cpu0，内核态处理在 cpu1，用户态处理在 cpu2，这样跨多个核心，容易造成 CPU 缓存失效，造成局部性失效。如果是 NUMA 架构，更会造成跨 NUMA 访问内存，性能受到很大影响。

**5、内存管理。**传统服务器内存页为 4K，为了提高内存的访问速度，避免 cache miss，可以增加 cache 中映射表的条目，但这又会影响 CPU 的检索效率。


综合以上问题，可以看出内核本身就是一个非常大的瓶颈所在。那很明显解决方案就是想办法绕过内核。

### 解决方案探讨

针对以上弊端，分别提出以下技术点进行探讨。

#### CONTENTS

1. 高性能网络技术
2. C10K 到 C10M 问题的演进
3. 为什么这么说？基于 OS 内...
4. 解决方案探讨
5. dpdk 的突破
6. dpdk 的应用
  - 6.1. 数据面（虚拟交换机）：
  - 6.2. 数据面（虚拟路由器）：
  - 6.3. 用户空间协议栈
7. 总结

 **控制层和数据层分离。**将数据包处理、内存管理、处理器调度等任务转移到用户空间去完成，而内核仅仅负责部分控制指令的处理。这样就不存在上述所说的系统中断、上下文切换、系统调用、系统调度等等问题。

- 2、使用多核编程技术代替多线程技术，并设置 CPU 的亲 and 性，将线程和 CPU 核进行一比一绑定，减少彼此之间调度切换。
- 3、针对 NUMA 系统，尽量使 CPU 核使用所在 NUMA 节点的内存，避免跨内存访问。
- 4、使用大页内存 代替普通的内存，减少 cache-miss。
- 5、采用无锁技术 解决资源竞争问题。

经很多前辈先驱的研究，目前业内已经出现了很多优秀的集成了上述技术方案的高性能网络数据处理框架，如 6wired、dpdk 等，其中，Intel 的 dpdk 在众多方案中脱颖而出，一骑绝尘。



CONTENTS

1. 高性能网络技术

2. C10K 到 C10M 问题的演进

3. 为什么这么说？基于 OS 内...

4. 解决方案探讨

5. dpdk 的突破

6. dpdk 的应用

6.1. 数据面（虚拟交换机）：

6.2. 数据面（虚拟路由器）：

6.3. 用户空间协议栈

7. 总结

dpdk 为 Intel 处理器架构下用户空间高效的数据包处理提供了库函数和驱动的支持，它不同于 Linux 系统以通用性设计为目的，而是专注于网络应用中数据包的高性能处理。

也就是 dpdk 绕过了 Linux 内核协议栈对数据包的处理过程，在用户空间实现了一套数据平面来进行数据包的收发与处理。在内核看来，dpdk 就是一个普通的用户态进程，它的编译、连接和加载方式和普通程序没有什么两样。

Linux云计算网络

云计算 | 网络 | Linux | 干货

获取学习大礼包后台  
回复“1024”

加群交流后台回复“加群”

dpdk 的突破

相对传统的基于内核的网络数据处理，dpdk 对从内核层到用户层的网络数据流程进行了重大突破，我们先看看传统的数据流程和 dpdk 中的网络流程有什么不同。

传统 Linux 内核网络数据流程：

硬件中断-->取包分发至内核线程-->软件中断-->内核线程在协议栈中处理包-->处理完毕通知用户层  
用户层收包-->网络层-->逻辑层-->业务层

Copy

dpdk 网络数据流程：

硬件中断-->放弃中断流程  
用户层通过设备映射取包-->进入用户层协议栈-->逻辑层-->业务层

Copy

下面就具体看看 dpdk 做了哪些突破？

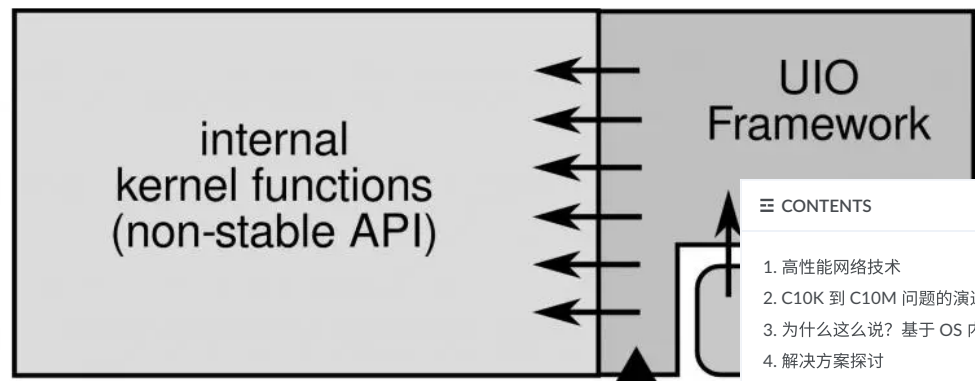
UIO（用户空间的 I/O 技术）的加持。

dpdk 能够绕过内核协议栈，本质上是得益于 UIO 技术，通过 UIO 能够拦截中断，并重设中断回调行为，从而绕过内核协议栈后续的数据处理流程。

UIO 设备的实现机制其实是对用户空间暴露文件接口，比如当注册一个 UIO 设备 uioX，就会出现文件 /dev/uioX，对该文件的读写就是该设备内存的读写。除此之外，对设备的控制还可以通过 /sys/class/uio 下的各个文件的读写来完成。

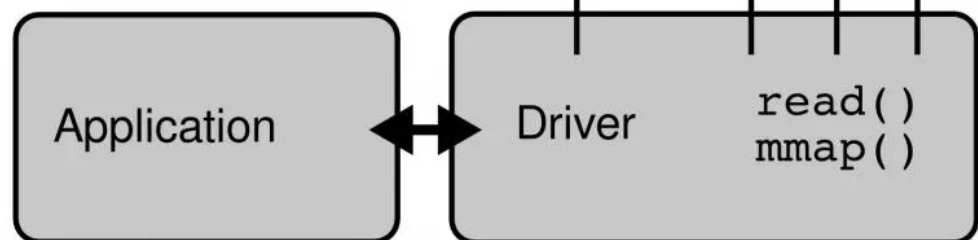


Kernelspace



Interface

Userspace



CONTENTS

1. 高性能网络技术
2. C10K 到 C10M 问题的演进
3. 为什么这么说? 基于 OS 内...
4. 解决方案探讨
5. dpdk 的突破
6. dpdk 的应用
  - 6.1. 数据面 (虚拟交换机):
  - 6.2. 数据面 (虚拟路由器):
  - 6.3. 用户空间协议栈
7. 总结

### 内存池技术

dpdk 在用户空间实现了一套精巧的内存池技术，内核空间和用户空间的内存交互不进行拷贝，只做控制权转移。这样，当收发数据包时，就减少了内存拷贝的开销。

### 大页内存管理

dpdk 实现了一组大页内存分配、使用和释放的 API，上层应用可以很方便使用 API 申请使用大页内存，同时也兼容普通的内存申请。

### 无锁环形队列

dpdk 基于 Linux 内核的无锁环形缓冲 kfifo 实现了自己的一套无锁机制。支持单生产者入列/单消费者出列和多生产者入列/多消费者出列操作，在数据传输的时候，降低性能的同时还能保证数据的同步。

### poll-mode网卡驱动

DPDK网卡驱动完全抛弃中断模式，基于轮询方式收包，避免了中断开销。

**\*\*NUMA \*\***

dpdk 内存分配上通过 proc 提供的内存信息，使 CPU 核心尽量使用靠近其所在节点的内存，避免了跨 NUMA 节点远程访问内存的性能问题。

### CPU 亲和性

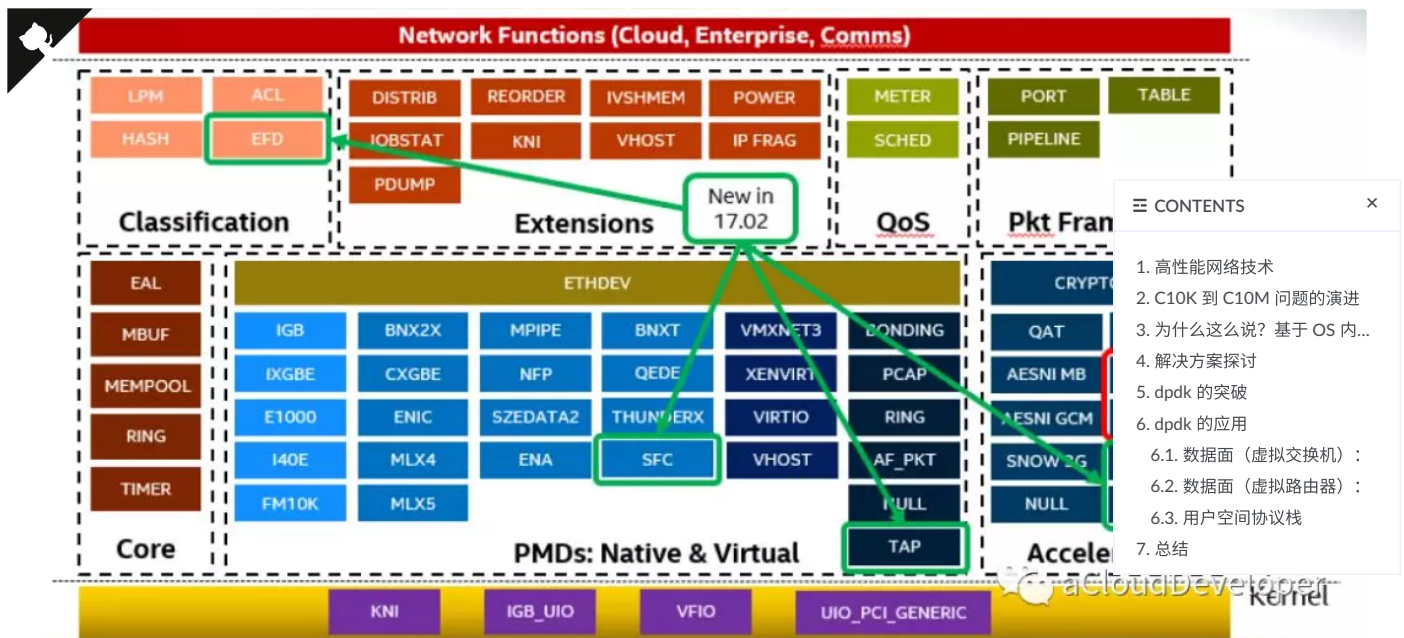
dpdk 利用 CPU 的亲和性将一个线程或多个线程绑定到一个或多个 CPU 上，这样在线程执行过程中，就不会被随意调度，一方面减少了线程间的频繁切换带来的开销，另一方面避免了 CPU 缓存的局部失效性，增加了 CPU 缓存的命中率。

### 多核调度框架

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

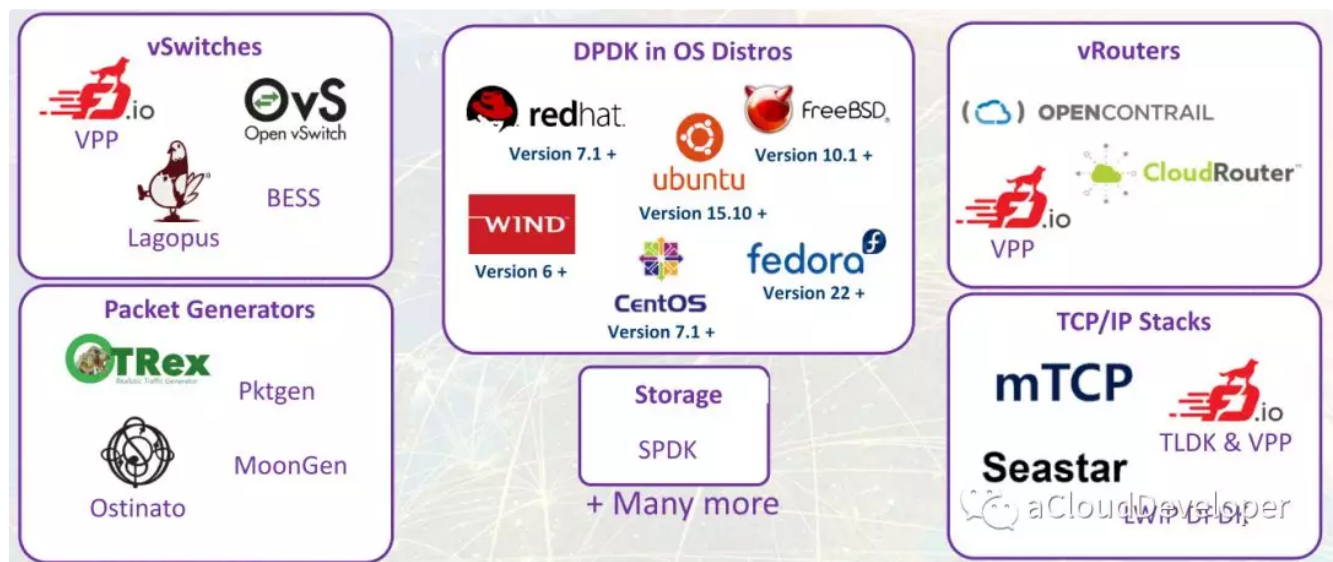
除了上述之外，dpdk 还有很多的技术突破，可以用下面这张图来概之。





## dpdk 的应用

dpdk 作为优秀的用户空间高性能数据包加速套件，现在已经作为一个“胶水”模块被用在多个网络数据处理方案中，用来提高性能。如下是众多的应用。



## 数据面（虚拟交换机）：

### OVS

Open vSwitch 是一个多核虚拟交换机平台，支持标准的管理接口和开放可扩展的可编程接口，支持第三方的控制接入。

<https://github.com/openvswitch/ovs>

### VPP

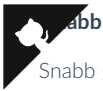
VPP 是 cisco 开源的一个高性能的包处理框架，提供了 交换/路由 功能，在虚拟化环境中，使它可以当做一个虚拟交换机来使用。在一个类 SDN 的处理框架中，它往往充当数据面的角色。经研究表明，VPP 性能要好于 ovs+dpdk 的组合，但它更适用于NFV，适合做特定功能的网络模块。

<https://wiki.fd.io/view/VPP>

### Lagopus

Lagopus 是另一个多核虚拟交换的实现，功能和 OVS 差不多，支持多种网络协议，如 Ethernet, VLAN, QinQ, MAC-in-MAC, MPLS PBB, 以及隧道协议，如 GRE, VxLan 和 GTP。

<https://github.com/lagopus/lagopus/blob/master/QUICKSTART.md>



Snabb 是一个简单且快速的数据包处理工具箱。

<https://github.com/SnabbCo/snabbswitch/blob/master/README.md>

## 数据面（虚拟路由器）：

### OPENCONTRAIL

一个集成了 SDN 控制器的虚拟路由器，现在多用在 OpenStack 中，结合 Neutron 为 OpenStack 提供一站式的网络

<http://www.opencontrail.org/>

### CloudRouter

一个分布式的路由器。

<https://cloudrouter.org/>

## 用户空间协议栈

### mTCP

mTCP 是一个针对多核系统的高可扩展性的用户空间 TCP/IP 协议栈。

<https://github.com/eunyoung14/mtcp/blob/master/README>

### lwIP

lwIP 针对 RAM 平台的精简版的 TCP/IP 协议栈实现。

<http://git.savannah.gnu.org/cgit/lwip.git/tree/README>

### Seastar

Seastar 是一个开源的，基于 C++ 11/14 feature，支持高并发和低延迟的异步编程高性能库。

<http://www.seastar-project.org/>

### f-stack

腾讯开源的用户空间协议栈，移植于 FreeBSD 协议栈，粘合了 POSIX API，上层应用（协程框架，Nginx, Redis），纯 C 编写，易上手。

<https://github.com/f-stack/f-stack>

## 总结

dpdk 绕过了 Linux 内核协议栈，加速数据的处理，用户可以在用户空间定制协议栈，满足自己的应用需求，目前出现了很多基于 dpdk 的高性能网络框架，OVS 和 VPP 是常用的数据面框架，mTCP 和 f-stack 是常用的用户态协议栈。很多大公司都在使用 dpdk 来优化网络性能。

### CONTENTS

1. 高性能网络技术
2. C10K 到 C10M 问题的演进
3. 为什么这么说？基于 OS 内...
4. 解决方案探讨
5. dpdk 的突破
6. dpdk 的应用
  - 6.1. 数据面（虚拟交换机）：
  - 6.2. 数据面（虚拟路由器）：
  - 6.3. 用户空间协议栈
7. 总结

我的公众号「Linux云计算网络」(id: cloud\_dev)，号内有 10T 书籍和视频资源，后台回复「1024」即可领取，分享的内容包括但不限于 Linux、网络、云计算虚拟化、容器 Docker、OpenStack、Kubernetes、工具、SDN、OVS、DPDK、Go、Python、C/C++ 编程技术等内  
容，欢迎大家关注。

## Linux云计算网络

云计算 | 网络 | Linux | 干货

获取学习大礼包后台  
回复“1024”

加群交流后台回复“加群”



作者：公众号「Linux云计算网络」，专注于Linux、云计算、网络领域技术干货分享

出处：<https://www.cnblogs.com/bakari/p/8404650.html>

本站使用「署名 4.0 国际」创作共享协议，转载请在文章明显位置注明作者及出处。



标签: 云计算, 虚拟化, DPDK

推荐 3

赞赏

收藏

反对 0

« 上一篇: vhost: 一种 virtio 高性能的后端驱动实现

» 下一篇: vhost-user 简介

posted @ 2018-02-02 13:01 CloudDeveloper

注册用户登录后才能发表评论, 请 [登录](#) 或 [注册](#), [访问](#) 网站首页。

Copyright © 2020 CloudDeveloper  
Powered by .NET Core on Kubernetes  
Powered By Cnblogs | Theme Silence v2.0.0



#### CONTENTS



1. 高性能网络技术
2. C10K 到 C10M 问题的演进
3. 为什么这么说? 基于 OS 内...
4. 解决方案探讨
5. dpdk 的突破
6. dpdk 的应用
  - 6.1. 数据面 (虚拟交换机):
  - 6.2. 数据面 (虚拟路由器):
  - 6.3. 用户空间协议栈
7. 总结

