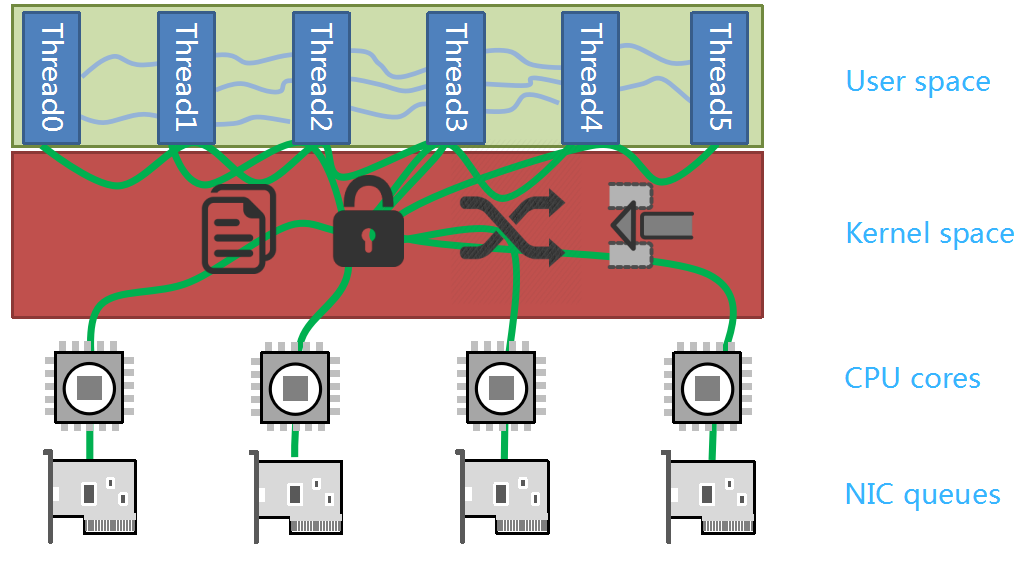
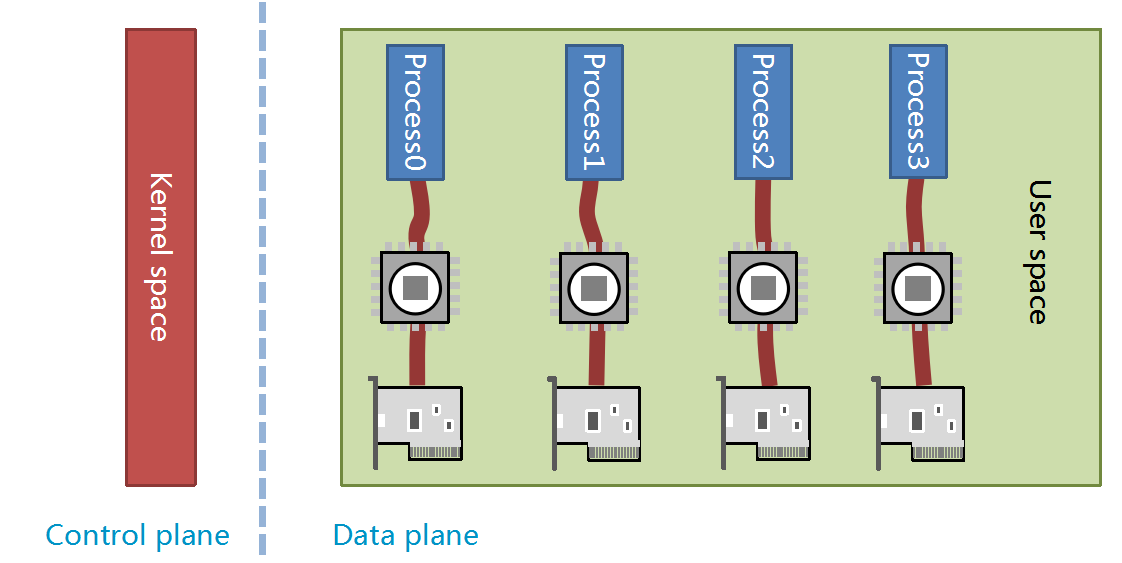
**F-Stack 开发背景**

随着网卡性能的飞速发展，10GE 网卡已经大规模普及，25GE/40GE/100GE 网卡也在逐步推广，linux 内核在网络数据包处理上的瓶颈也越发明显，在传统的内核协议栈中，网卡通过硬件中断通知协议栈有新的数据包到达，内核的网卡驱动程序负责处理这个硬件中断，将数据包从网卡队列拷贝到内核开辟的缓冲区中（DMA），然后数据包经过一系列的协议处理流程，最后送到用户程序指定的缓冲区中。在这个过程中中断处理、内存拷贝、系统调用（锁、软中断、上下文切换）等严重影响了网络数据包的处理能力。操作系统的对应用程序和数据包处理的调度可能跨 CPU 调度，局部性失效进一步影响网络性能。



而互联网的快速发展亟需高性能的网络处理能力，kernel bypass 方案也越来被人所接受，市场上也出现了多种类似技术，如 DPDK、NETMAP、PF\_RING 等，其核心思想就是内核只用来处理控制流，所有数据流相关操作都在用户态进行处理，从而规避内核的包拷贝、线程调度、系统调用、中断等性能瓶颈，并辅以各种性能调优手段，从而达到更高的性能。其中 DPDK 因为更彻底的脱离内核调度以及活跃的社区支持从而得到了更广泛的使用。



**F-Stack 开发框架**

F-Stack 是一款兼顾高性能、易用性和通用性的网络开发框架，传统上 DPDK 大多用于 SDN、NFV、DNS 等简单的应用场景下，对于复杂的 TCP 协议栈上的七层应用很少，市面上已出现了部分用户态协议栈，如 mTCP、Mirage、lwIP、NUSE 等，也有用户态的编程框架，如 SeaStar 等，但统一的特点是应用程序接入门槛较高，不易于使用。

F-Stack 使用纯 C 实现，充当胶水粘合了 DPDK、FreeBSD 用户态协议栈、Posix API、微线程框架和上层应用（Nginx、[Redis](https://cloud.tencent.com/product/crs?from=10680" \t "_blank)），使绝大部分的网络应用可以通过直接修改配置或替换系统的网络接口即可接入 F-Stack，从而获得更高的网络性能。

**发展历程**

* 2012 DNSPod 调研下一代 DNS 方案应对日益严重的 DDoS 攻击
* 2013 上线 DPDK+UDP 版本权威 DNS 解析程序 DKDNS，单机 10GE 网卡性能达到 1100 万 QPS
* 2014 实现用户态简易 TCP/IP 协议栈，支持 TCP DNS，性能达到 60 万+TPS
* 2015 上线 DPDK+TCP/IP+通用网关应用
* 2016 完善协议栈和应用层接口的开发框架 FlashStack，在腾讯云[对象存储](https://cloud.tencent.com/product/cos?from=10680" \t "_blank)，HttpDNS 等业务中上线
* 2017 移植 FreeBSD 协议栈替换自研协议栈的全新 F-Stack 开源。

**F-Stack 架构**



F-Stack 总体架构如上图所示，具有以下特点:

* 使用多进程无共享架构。
* 各进程绑定独立的网卡队列和 CPU，请求通过设置网卡 RSS 散落到各进程进行处理。
* 各进程拥有独立的协议栈、PCB 表等资源。
* 每个 NUMA 节点使用独立的内存池。
* 进程间通信通过无锁环形队列（rte\_ring）进行。
* 使用 DPDK 作为网络 I/O 模块，将数据包从网卡直接接收到用户态。
* 移植 FreeBSD Release 11.0.1 协议栈到用户态并与 DPDK 对接。

**FreeBSD 协议栈**

原本 F-Stack 使用自己开发的 TCP/IP 协议栈，但是随着腾讯云各种复杂业务的接入，原有协议栈已越来越不能满足需求，继续研发并维护一个完整的协议栈将耗费大量的精力，所以决定使用开源的协议栈来代替原有协议栈。在调研了多种协议栈之后，考虑到协议栈的稳定性、完整性、社区活跃性、生态完善程度等因素，最终决定将 FreeBSD 协议栈移植到用户态，提供了完整的协议栈功能并可以快速跟进社区对协议栈的改进。

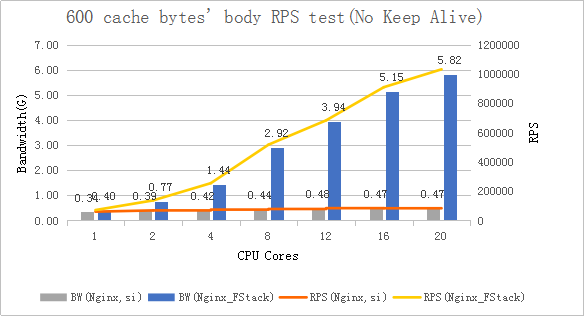
**主要修改：**

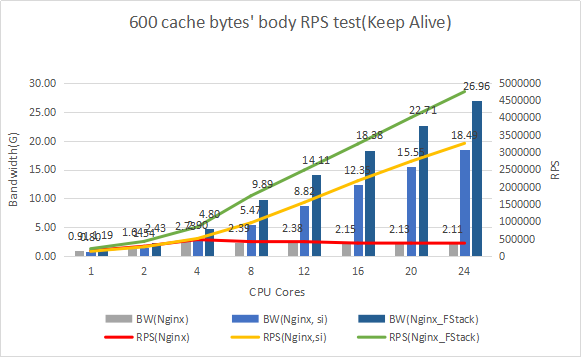
* 调度：对 FreeBSD Network Stack 的内核线程、中断线程、定时器线程、sched、sleep 等进行了去除。
* 锁：对 FreeBSD Network Stack 的锁操作进行了去除，包括 mtx、rw、rm、sx、cond 等。
* 内存相关：phymem、uma\_page\_slab\_hash、uma、kmem\_malloc、malloc 等实现。
* 全局变量：pcpu、curthread、proc0、thread0 等初始化。
* 环境变量：setenv、getenv 实现。
* SYS\_INIT：mi\_startup。
* 时钟：timecounter、ticks、hz、定时器等实现。
* 其他：Linux 和 FreeBSD 的 errno 转换、胶水代码、移除了不需要的功能模块等。

**典型 HTTP 场景性能测试**

以下分别为使用 F-Stack 中的 Nginx 在 40GE 网卡下测试 HTTP 短连接和长连接小包（body 大小为 600 字节）的性能表现，实际带宽性能会随着 body 的增加而增加，并可达到网卡线速。

下图中的 Nginx 和 Nginx-si 都是原生 Nginx，Nginx-si 表示网卡队列中断到不同 CPU，另外的是采用系统默认配置，未将网卡队列中断到不同 CPU。





更多场景下的 HTTP 性能测试数据请关注后续文章。

**F-Stack 交流**

* [GITHUB](https://github.com/f-stack/f-stack), 欢迎业务使用、 或者加入开发、 顺便 star 一下我们的项目，有任何问题可以提 ISSUSES 联系我们。
* [官方网站](http://www.f-stack.org/)，官网资源持续完善中，如果您的项目使用了 F-Stack，也希望通知我们，让我们加入到应用列表中。
* 微信公众号：搜索公众号**FStack**关注，会不定期发布 F-Stack 的技术细节、使用技巧和应用场景等文章，同时欢迎投稿。
* 微信讨论群：我们建立了 F-Stack 讨论群，欢迎联系我们拉您加入。

原创声明，本文系作者授权云+社区发表，未经许可，不得转载。

如有侵权，请联系 yunjia\_community@tencent.com 删除。