<https://blog.csdn.net/ygm_linux/article/details/81589536>

# ****——lvyilong316****

f-stack是腾讯基于dpdk开发的一套用户态协议栈，目前已经开源，相关介绍可以看这里：<https://cloud.tencent.com/developer/article/1005179>。 开源代码在这里：<https://github.com/F-Stack/f-stack.git> 。

## 项目背景

F-Stack 这个项目起始于DNSPod的授权DNS项目，当时是12年，DPDK还未开源的时候，腾讯就基于DPDK做了授权DNS，做完的时候正好DPDK也开源了，正式上线后10GE单网卡性能达到1100万qps，后面又实现了一个简易的TCP协议栈用于支持TCP DNS。后来DNSPod合并进入腾讯云，腾讯云有大量业务需要高性能的接入服务，所以就在原来授权DNS的TCP协议栈基础上实现了较为完整的TCP/IP协议栈(参考了mTCP,SeaStar,lwIP等)，加上了协程框架，并且上层兼容了腾讯SNG的服务端框架SPP，使得腾讯云原有的业务可以无缝迁入，但是经过近一年的线上使用运行，腾讯发现，线上的网络环境各式各样，这个TCP/IP协议栈并不能满足全部需求(各种tunnel等)，而自己去实现各种协议既费力又不讨好，所以经过一段时间的调研，决定移植FreeBSD的协议栈。

## 整体介绍

这里不再对这个协议栈的特点和介绍展开，网上都有，其实就是dpdk的一些特点，另一点就是对现有posix编程接口比较友好吧，提供了线程和epoll等接口，方便应用的移植。这里主要是对f-stack的实现框架转发路径做一个简单分析。

f-stack没有使用自己开发的协议栈，据说早期使用的是基于mTCP的自研协议栈，但后来因为兼容性，在公有云环境使用会有诸多问题。如果比较笼统的说，f-stack可以等价于dpdk+freebsd协议栈。F-stack自己的关键工作就是将两者结合起来，入方向将流量从dpdk导入freebsd协议栈，出方向由freebsd协议栈导入dpdk。当然freebsd协议栈也是做了少量改动的，比如：

1) 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。

2) 锁：对 FreeBSD Network Stack 的锁操作进行了去除，包括 mtx、rw、rm、sx、cond 等。

3) 内存相关：phymem、uma\_page\_slab\_hash、uma、kmem\_malloc、malloc 等实现。

4) 全局变量：pcpu、curthread、proc0、thread0 等初始化。

5) 环境变量：setenv、getenv 实现。

6) SYS\_INIT：mi\_startup。

7) 时钟：timecounter、ticks、hz、定时器等实现。

8) 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。

但这些改动涉及的代码非常少，据说只有几百行，所以非常方便将最新的freebsd代码更新过来，便于维护。这样的设计的优势就是一方面可以高效的利用dpdk框架的高性能转发，另一方面不至于陷入协议栈的复杂开发当中。

下面简单的看一下f-stack的代码结构：

## Structure of F-Stack code

    ├── app  -- Nginx(1.11.10)/Redis(3.2.8)/Microthread framework  (适配f-stack的Nginx和Redis)

    ├── config.ini  (f-stack的配置文件)

    ├── doc      (f-stack的一些说明文档)

    ├── dpdk -- Intel DPDK(16.07) directory   (放置dpdk代码)

    ├── example – DEMO                 (使用f-stack的例子)

    ├── freebsd -- FreeBSD(11.0) Network Stack directory  (freebsd协议栈代码)

    ├── lib -- F-Stack lib directory        (f-stack自身的代码)

    ├── mk  
   └── start.sh

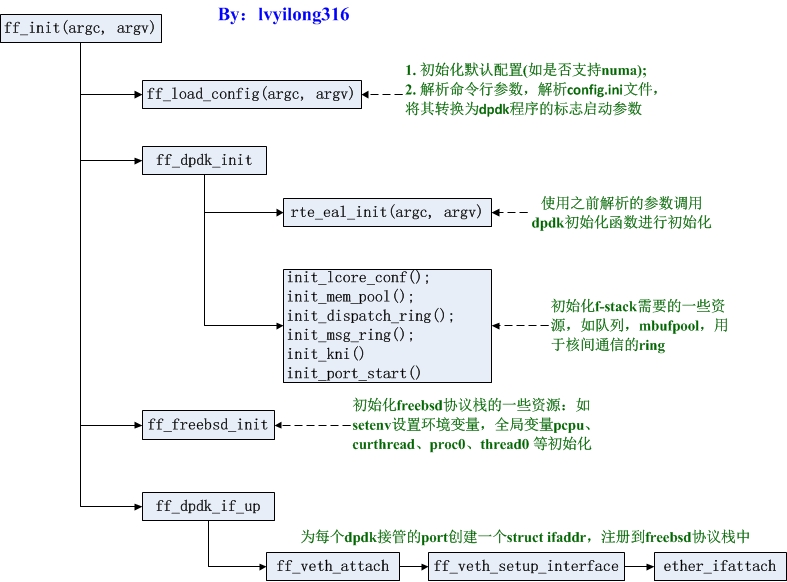
## 初始化分析

我们直接先看一下f-stack提供的demo的main函数，有一个宏观认识。



代码非常简单，和正常的socket编程一样调用socket，bind，listen等接口，可以看到f-stack其提供的ff\_socket，ff\_bind，ff\_listen和posix接口是完全兼容的。其他关键的区别就是最开始的ff\_init(argc, argv)调用，这个主意是为了f-stack框架的启动初始化，下面会分析，另一个就是ff\_run(loop, NULL)，熟悉dpdk的人应该能猜到，这就是启动主循环开始网卡的收发逻辑。

下面一张图看一下ff\_init的主要初始化逻辑。



其中最关键的就是ff\_dpdk\_if\_up，这个函数为每个dpdk接管的port创建一个struct ifnet结构，并将这个结构注册到freebsd中，其中调用的ether\_ifattach就是freebsd的接口，这个结构可以理解为网卡对应的虚拟设备，这个设备也是dpdk和freebsd关联的核心。Ifnet有很多handle函数，会被做如下初始化：

    ifp->if\_start = ff\_veth\_start;

    ifp->if\_transmit = ff\_veth\_transmit;  
   ifp->if\_qflush = ff\_veth\_qflush;  
   ifp->if\_output = ether\_output;  
   ifp->if\_input = ether\_input;

……

其中核心是if\_input，if\_output和if\_transmit三个函数，if\_input就是ifnet的收包函数，可以理解为协议栈的入口函数，有些类似于kernel协议栈的netif\_receive\_skb函数；但是把if\_output比喻成协议栈的出口函数就有点不恰当了，因为启动还有bridge的转发逻辑，其中（ether\_output）调用的ifp->if\_transmit可以理解为真正的协议栈出口函数，类似于kernel的dev\_queue\_xmit。

## 转发流程分析

    转发流程以ff\_run开始的main\_loop开始，我们以tcp收到syn+ack及回复流程为例，分析这个转发过程(syn+ack处理不会上送app，由协议栈完成)，直接上图，流程如下：

