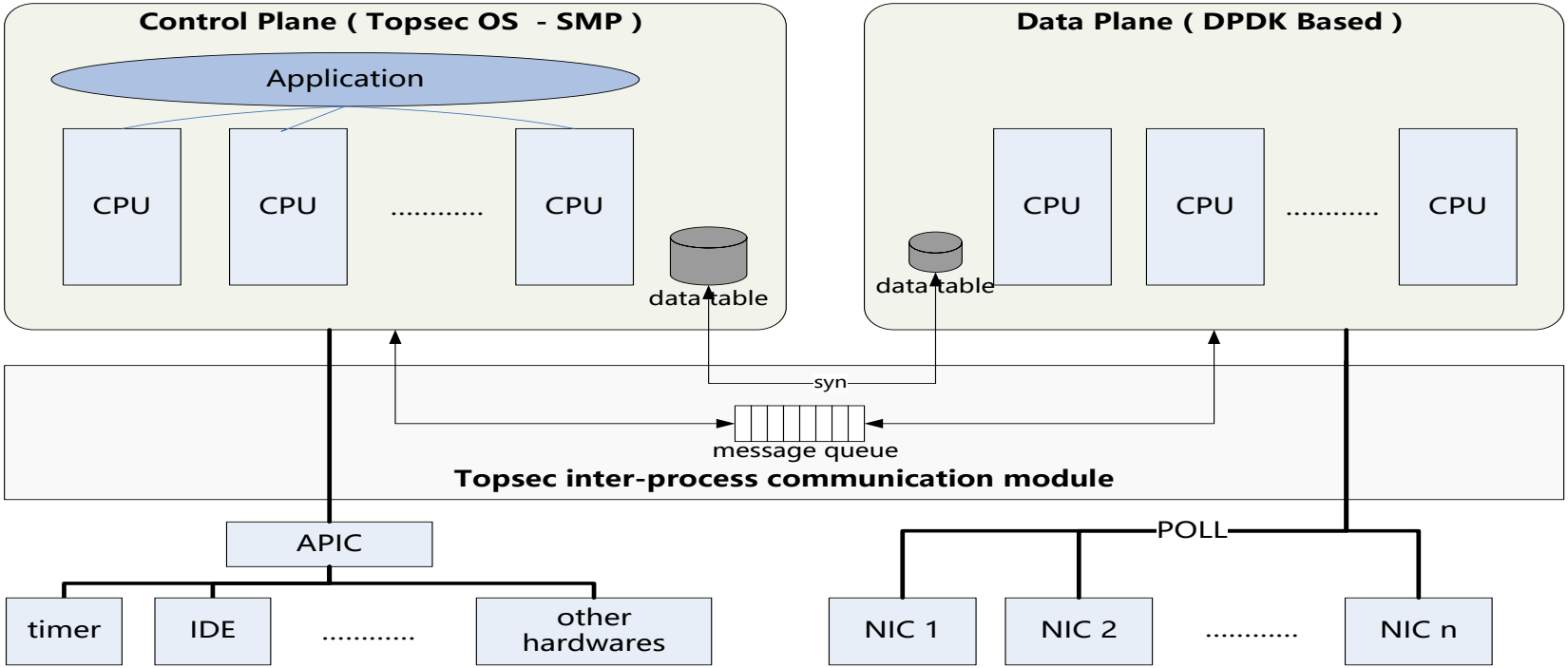
DPDK: A journey of migration to Linux kernel
将DPDK移植到内核态的旅行

北京天融信
娄扬

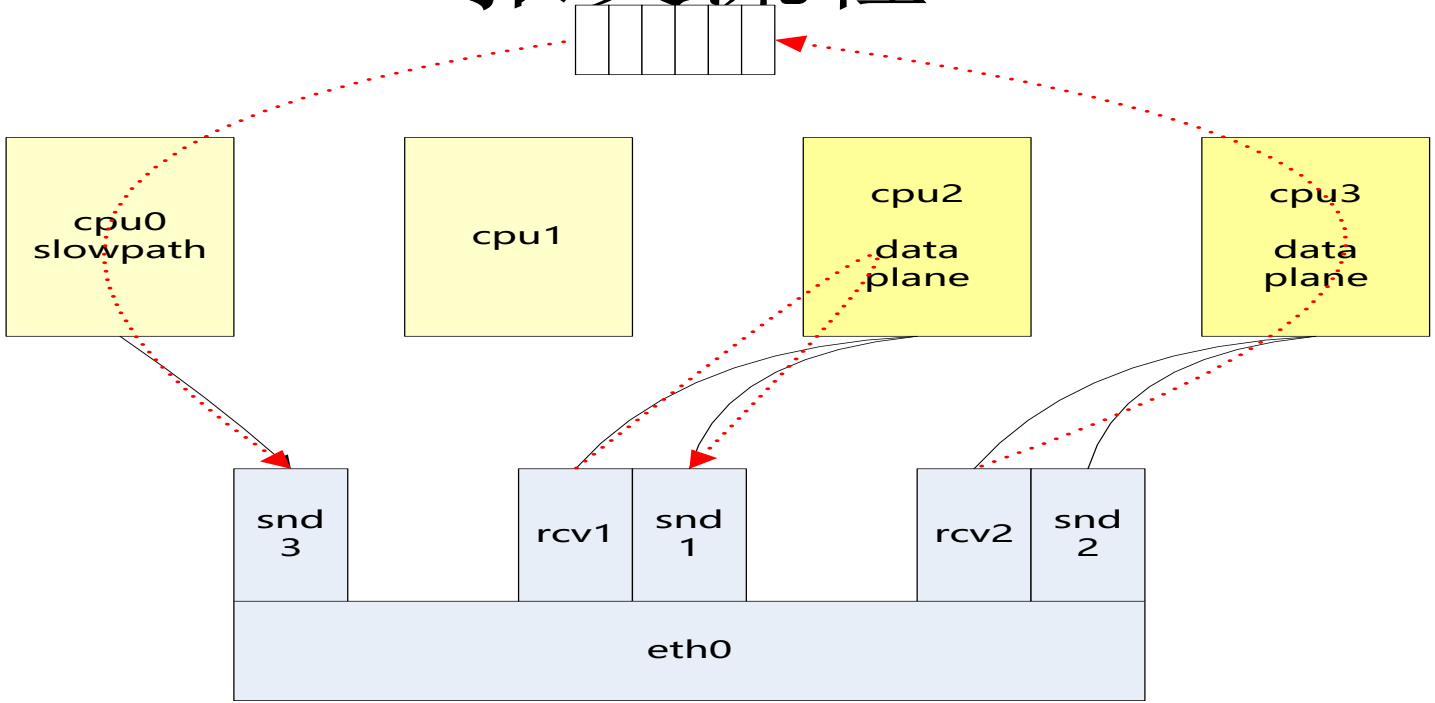
项目背景

- 现有应用程序的复杂性使一些应用难以不加改动的使用dpdk
  - linux内核态的应用
  - 使用特殊操作系统的应用
  - 尚未被支持的网卡类型
- 项目背景
  - 现有应用在linux内核态运行，严重依赖内核数据结构，移植到应用层需要较大的工作量
  - 有些硬件型号的网卡不被dpdk支持
  - 有些硬件平台是单核cpu
- 结合需求，理解dpdk提高网络应用性能的原理
  - →改动应用以适配dpdk
  - →改动dpdk以适配应用

工作模式


Control Plane (Topsec OS - SMP)
Application
CPU
CPU
.....
CPU
data table
Data Plane (DPDK Based)
CPU
CPU
.....
CPU
data table
syn
message queue
Topsec inter-process communication module
APIC
POLL
timer
IDE
.....
other hardwares
NIC 1
NIC 2
.....
NIC n

报文流程


cpu0
slowpath
cpu1
cpu2
data
plane
cpu3
data
plane
snd
3
rcv1
snd
1
rcv2
snd
2
eth0

DPDK内存管理的优势

- 本地缓存
  - 在mempool中，每个cpu core有自己的缓存，申请和释放时无需竞争操作。
- 无锁的内存块管理
  - mempool采用无锁ring管理内存块资源
- NUMA的支持
  - 优先使用相同socket的内存
- 巨页
  - 降低TLB miss

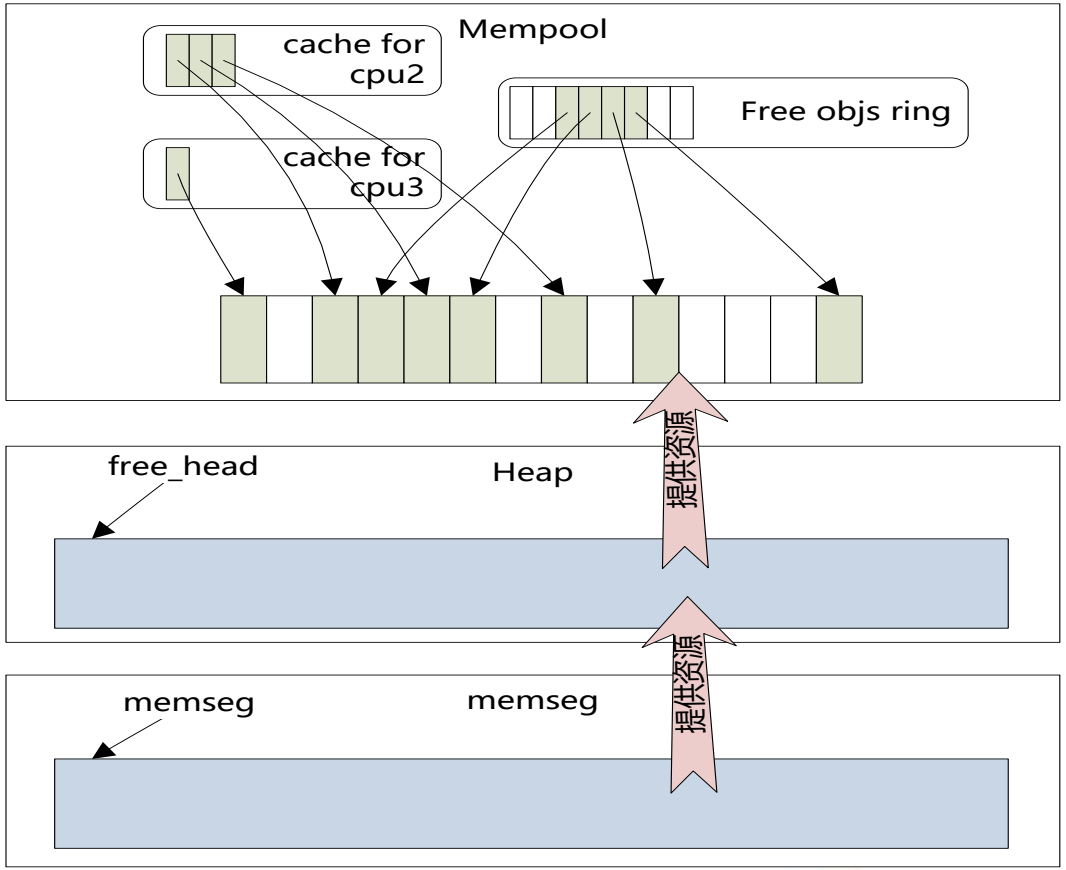
- linux的消息管理
  - 由于linux是通用型操作系统，所以并未对消息管理做更多的优化，消息管理使用的是linux内核中通用的内存管理方法
  - sk_buff数据结构使用kmem_cache管理，kmem_cache管理相同大小的内存块，但是申请释放时仍然会有自旋锁的操作
  - 报文内容是kmalloc分配的，增加了额外的内存分配操作

进行巨页的映射

通过重新映射得到连续的物理内存块

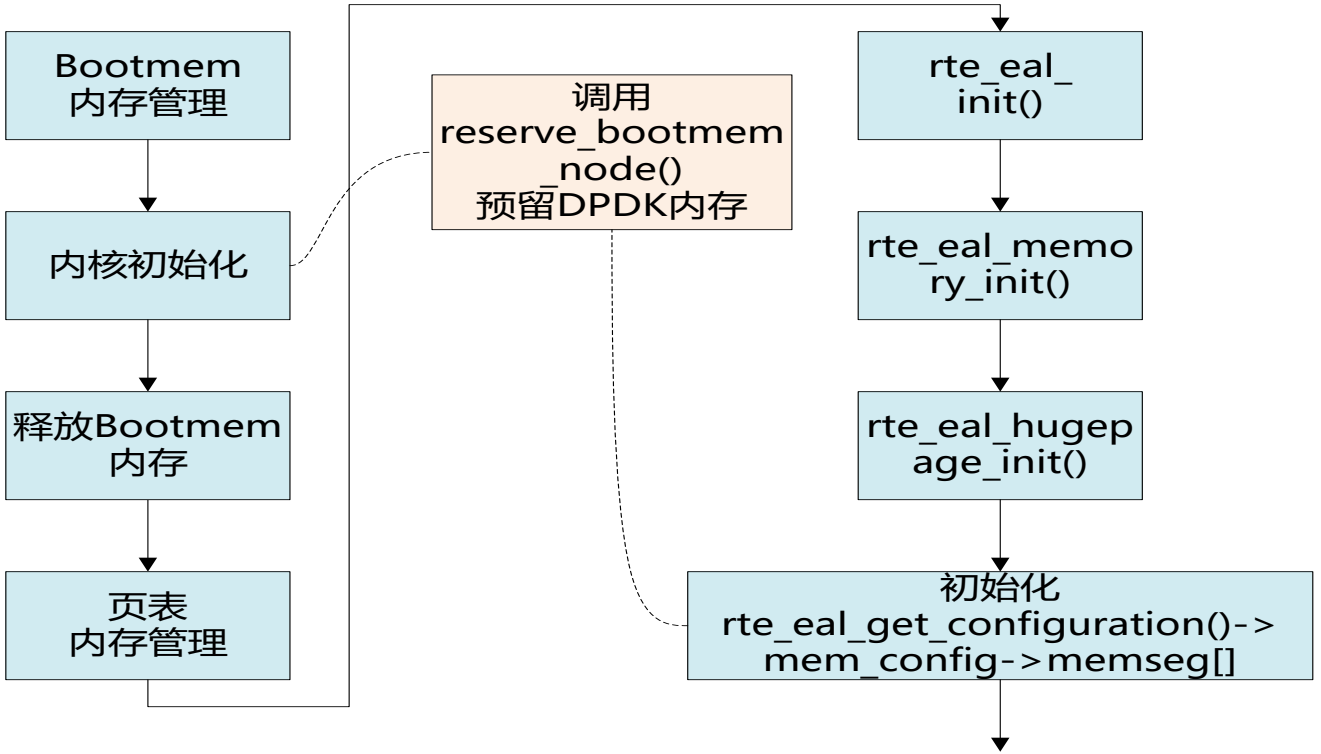
memseg记录这些连续的物理内存块

初始化heap来管理所有的物理内存



内核态的dpdk内存管理

- 内核原有的内存管理不做改动
- 内核划分一块足够的物理内存供dpdk相关的模块使用
- 预留的物理内存完全采用dpdk的管理方式
- 常用的内核分配内存方法，如kmalloc、__get_free_page只能分配有限的连续物理内存
- reserve_bootmem_node：在系统初始化的过程中，预留一块专有内存，使这块内存不再被linux的内存管理使用


Bootmem
内存管理
内核初始化
释放Bootmem
内存
页表
内存管理
调用
reserve_bootmem
_node()
预留DPDK内存
rte_eal_
init()
rte_eal_memo
ry_init()
rte_eal_hugep
age_init()
初始化
rte_eal_get_configuration()->
mem_config->memseg[]

消息内容的修改

```
struct rte_mbuf {
    COMM_SKB_ATTR_PART;
    COMM_SKB_DATA_PART;
    FP_SKB_PART
};
```

```
struct sk_buff {
    ......
    COMM_SKB_ATTR_PART;
    COMM_SKB_DATA_PART;
    ......
};
```

- COMM_SKB_ATTR_PART和COMM_SKB_DATA_PART记录数据面消息和管理面消息的共有成员，如标记、各层头指针、长度、设备指针等等。
- 由于数据面无法转发的报文需要进入linux内核，所以采用了同样的内存分布并使用cache line对齐，可以更快的把rte_mbuf拷贝给sk_buff

数据平面

- 标准的dpdk中
  - 数据平面（lcore）以linux用户态线程形式运行
  - 通过pthread_setaffinity_np实现cpu绑定
- linux内核中
  - 数据平面可以以linux内核线程的方式存在
  - 通过set_cpus_allowed实现数据面的cpu绑定
- 类裸核方式
  - 数据面脱离linux的控制，不受linux的调度
  - 无中断处理，不接受cpu的中断

如何降低数据平面抖动

- 数据面的抖动 → 降低了数据面的稳定性 → 少量丢包的现象
- 抖动的来源
  - 进程调度
  - 中断的干扰
- 理想的数据平面
  - 无中断干扰
  - 绑定专属的线程/进程
  - 尽量轻巧的定时任务
  - 读取相同node的资源

进程调度的优化

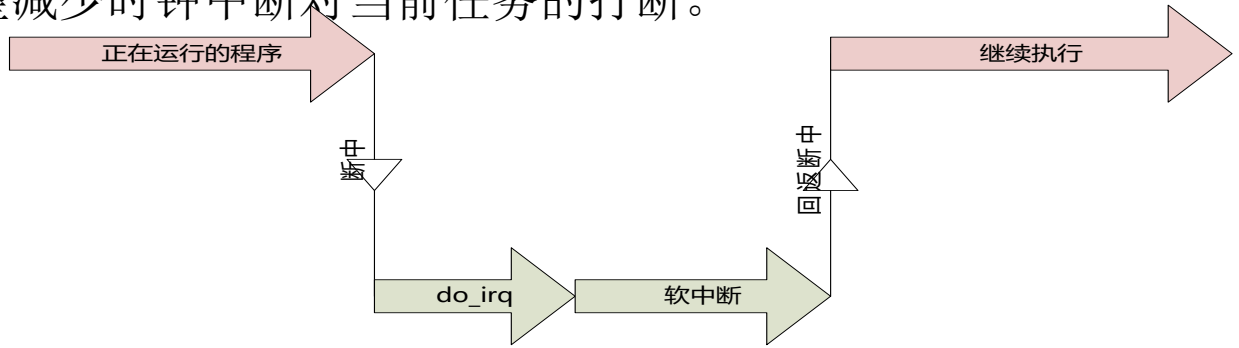
- 调整linux的进程调度的目标：
  - 所有的进程缺省调度到管理面cpu core上
  - 数据面线程和数据面cpu core一一对应绑定
- 手工亲和绑定方法
  - set_cpus_allowed（内核态）
  - sched_setaffinity、pthread_setaffinity_np（用户态）
- 子进程会继承父进程的cpu亲和属性，亲和到cpu0上运行的进程产生的子进程也会被亲和到cpu0
- isolcpus内核启动参数
  - isolcpus功能用于在SMP均衡调度算法中将一个或多个CPU孤立出来。同时可通过亲和性设置将进程置于“孤立CPU”运行
  - isolcpus与手动设置每个任务的亲和性相比，提高了调度器的性能
  - 比如孤立cpu5~8核（cpu id对应4~7），添加isolcpus=4,5,6,7至内核命令行

进程调度的验证

- ps命令
  - ps -eLo pid,lwp,args:50,psr
  - psr列可以显示进程/线程当前被分配的cpu id
- taskset命令
  - taskset -p pid 显示某进程的cpu绑定
  - taskset -p mask pid
    - -sh-3.2# taskset -p 795
    - pid 795's current affinity mask: f
    - -sh-3.2# taskset -p 1 795
    - pid 795's current affinity mask: f
    - pid 795's new affinity mask: 1
    - -sh-3.2# taskset -p 795
    - pid 795's current affinity mask: 1

linux的定时中断处理

- 定时中断处理每秒执行100至1000次，主要的目的是计算进程的时间片，进行进程切换等，这对于通用操作系统有重要意义。
- 缺省情况下，linux会尽可能把产生的中断在cpu间做均衡
- 随着linux应用范围的扩大，一方面功耗敏感的嵌入式系统希望在cpu空闲的时候停止时钟中断以降低功耗；另一方面，实时或高性能计算系统也希望减少时钟中断对当前任务的打断。



如何减少数据面中断干扰

- 关闭irqbalance服务
  - /etc/init.d/irqbalance stop
- 设置特定中断的cpu亲和
  - /proc/irq/**/smp_affinity
- dynticks-idle模式
  - 完全避免了时钟中断的干扰，提高了实时性
  - 选中内核编译选项CONFIG_NO_HZ_FULL
  - 提供内核启动参数"nohz_full="，指定adaptive-ticks CPUs（不能指定所有的cpu，至少要保留boot CPU，否则像gettimeofday()之类的系统调用将无法返回准确的结果）
  - 如果cpu core上挂载了RCU回调，它无法进入dynticks-idle模式
- 更彻底的方式：使数据平面cpu完全脱离linux的管理

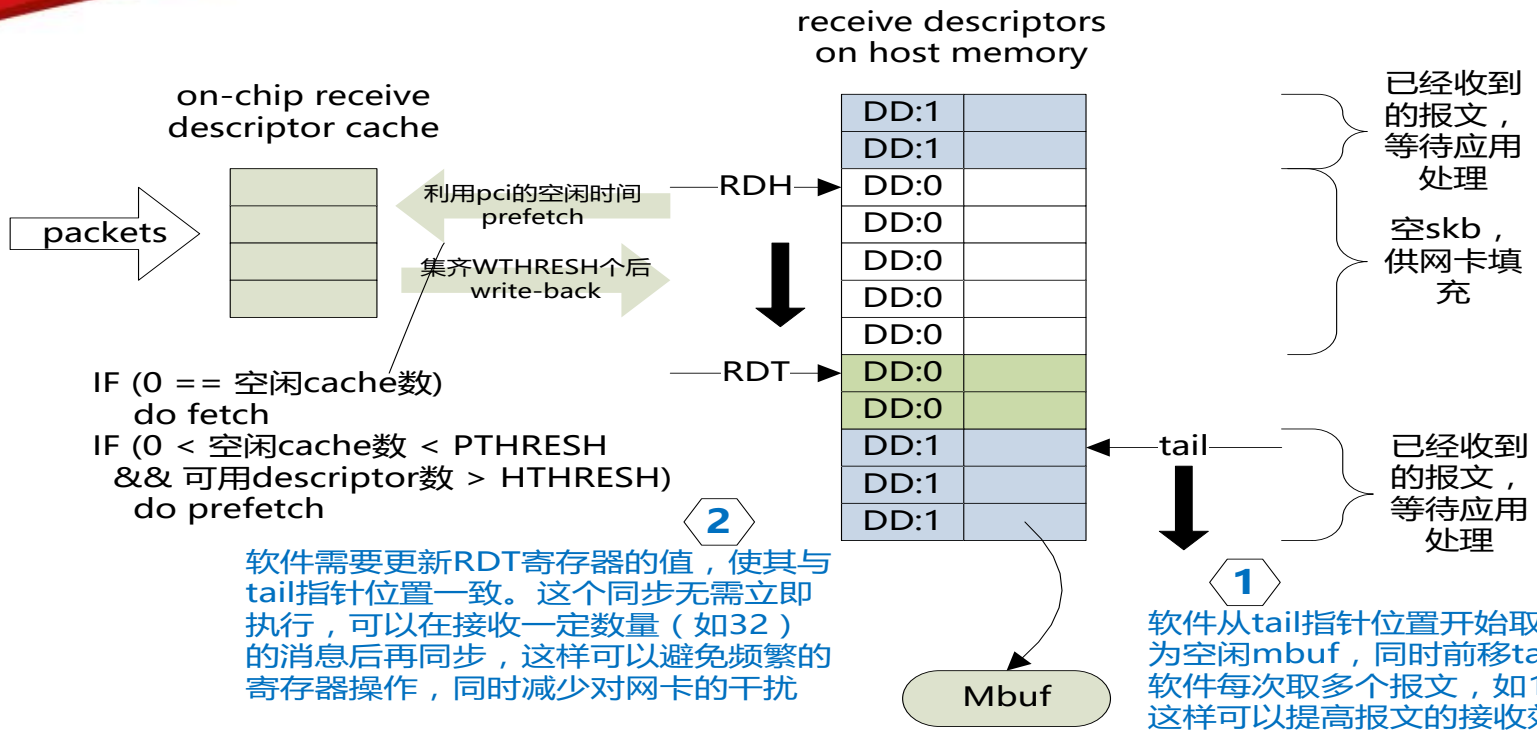
网卡驱动

- dpdk的网卡驱动的关键点：
  - 去掉了收发包相关的中断
  - 每个logic core在每个网卡上有自己的收发包队列，避免了并发竞争
  - 每次收发包处理更多的报文
  - 控制修改网卡寄存器的频率和时机
  - 提高网卡的收发队列cache的效率

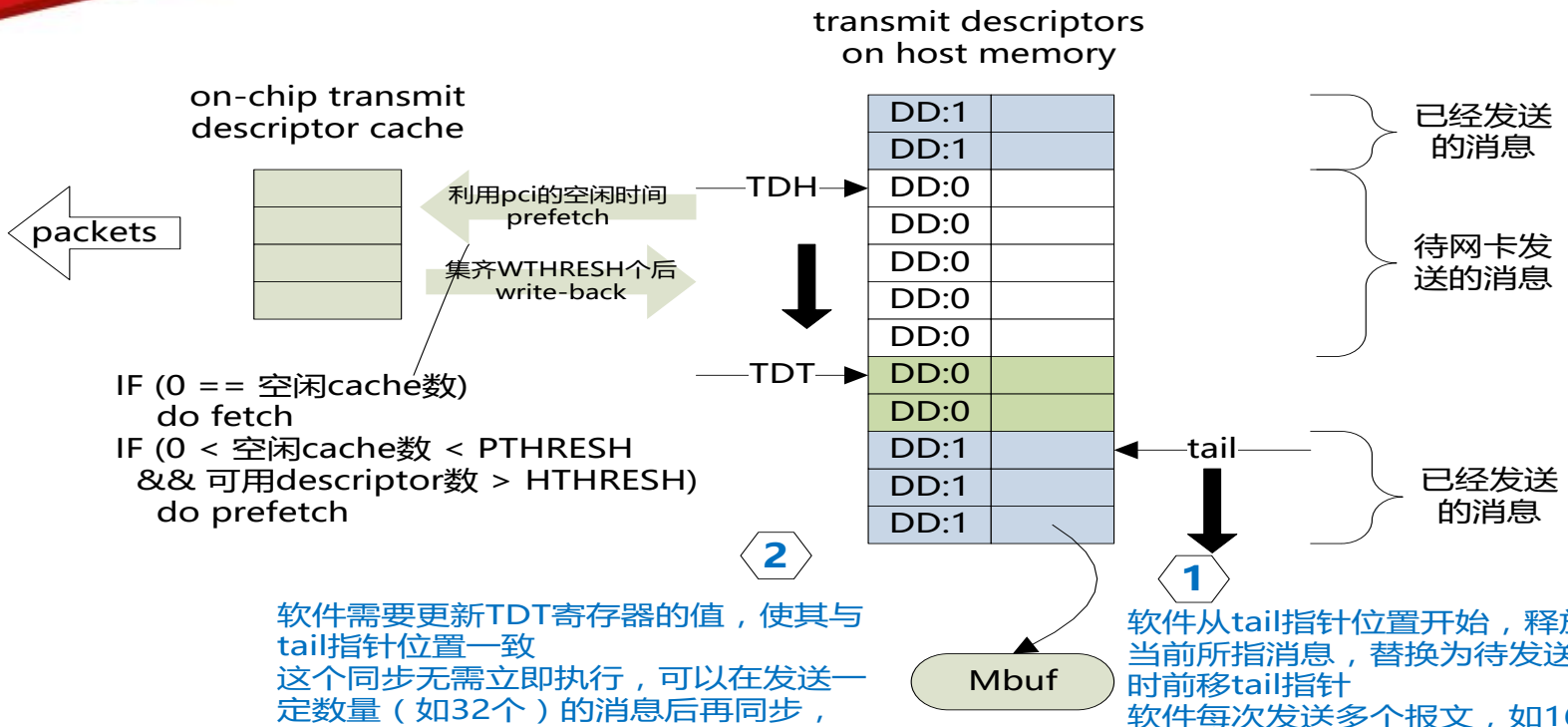
内核态的poll驱动

- 几乎所有的网卡都有官方稳定的linux内核驱动版本，所以在其基础上修改是最好的方式。
- 网卡的管理、配置操作等仍然保持不变，保证系统的稳定性。
- 只需要做和性能优化相关的改动，每个网卡特有的相关代码不需要关心。
  - 网卡的链路状态变化相关的通知中断仍然由linux内核管理。
  - 收发包相关的中断取消。
  - 网卡驱动初始化时，需要将网卡的收发包队列个数设置为数据面的个数（管理面需要一个额外的发送队列）
  - 去掉对NAPI的操作：netif_napi_del、netif_napi_add、napi_enable
  - net_device结构提供新的函数指针：tx_burst，rx_burst，供数据平面调用
  - 记录所有RUNNING状态的设备，以便轮询方式收发报文

以82599为例



以82599为例

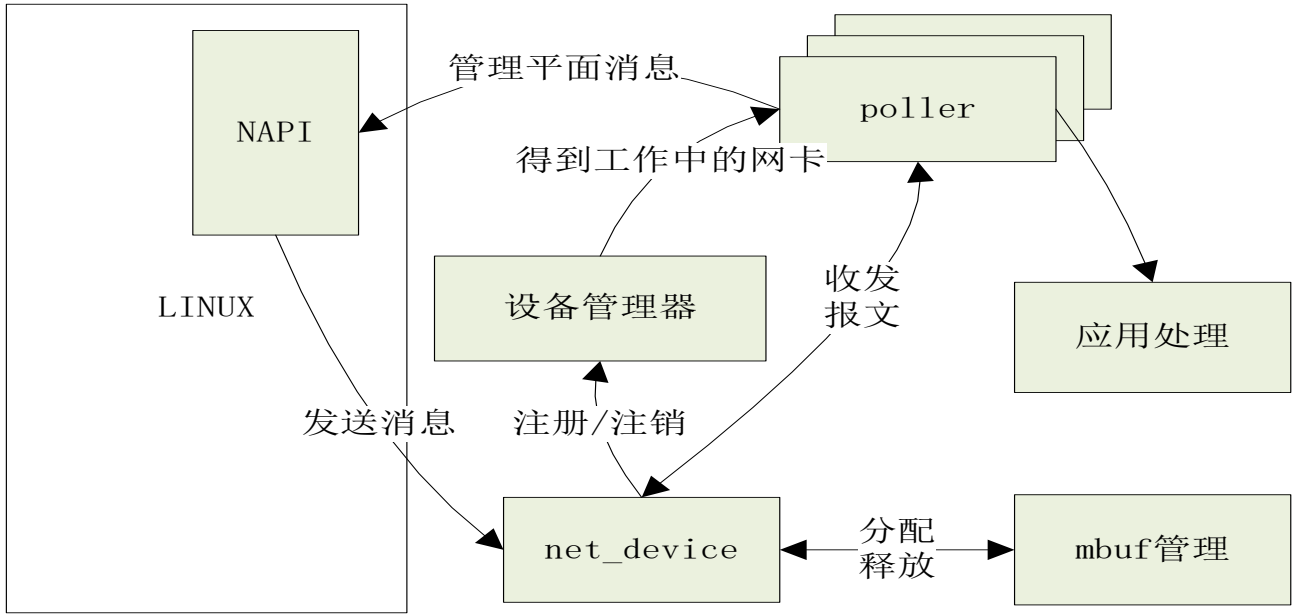


软件需要更新TDT寄存器的值，使其与tail指针位置一致
这个同步无需立即执行，可以在发送一定数量（如32个）的消息后再同步，这样可以避免频繁的寄存器操作，同时减少对网卡的干扰

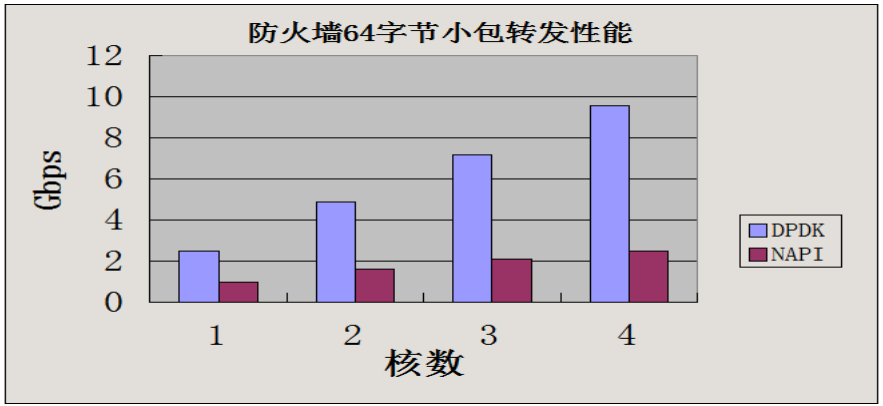
软件从tail指针位置开始，释放描述符当前所指消息，替换为待发送消息，同时前移tail指针
软件每次发送多个报文，如16或32个。这样可以提高报文的发送效率，可以更好的控制cpu的预取操作并减少cache失效

其他的改动

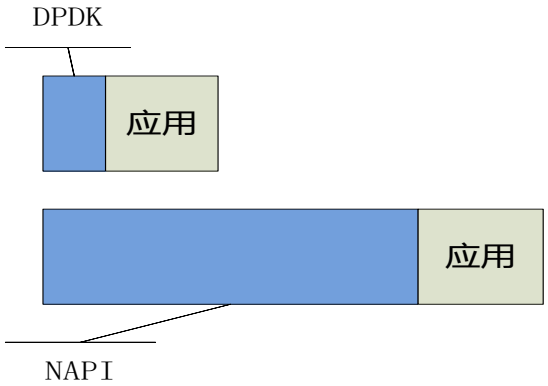
- 数据平面到管理平面的单向通道
  - 利用dpdk的ring作为消息队列
  - 管理平面采用收包软中断来接收消息
  - 数据平面把消息加入队列后触发软中断
- 网卡的管理面发送方法
  - 管理平面仍然使用net_device结构的hard_start_xmit函数指针发送消息


NAPI
管理平面消息
poller
得到工作中的网卡
LINUX
设备管理器
收发
报文
应用处理
发送消息
注册/注销
net_device
分配
释放
mbuf管理

性能分析



报文转发耗时分析:



总结

- 通过给DPDK预分配一大块连续物理内存，并使用DPDK的内存管理机制
- 通过隔离进程调度和中断的cpu得到一个理想的数据平面
- 通过把标准驱动修改为高效的轮询模式
- 最终我们得到一个性能和标准dpdk近似的内核版dpdk


DPDK
DATA PLANE DEVELOPMENT KIT