2015中国DPDK开发者大会
China DPDK Summit 2015

Presented By:

中国移动 China Mobile　中国电信 CHINA TELECOM　HUAWEI　intel　ZTE

电信业务场景下的数据平面转变

刘珺

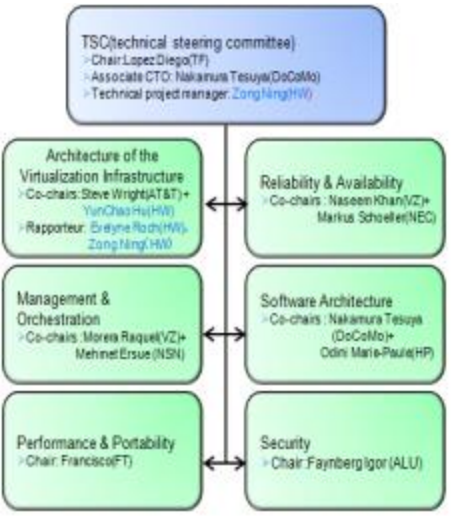
2015.04.21

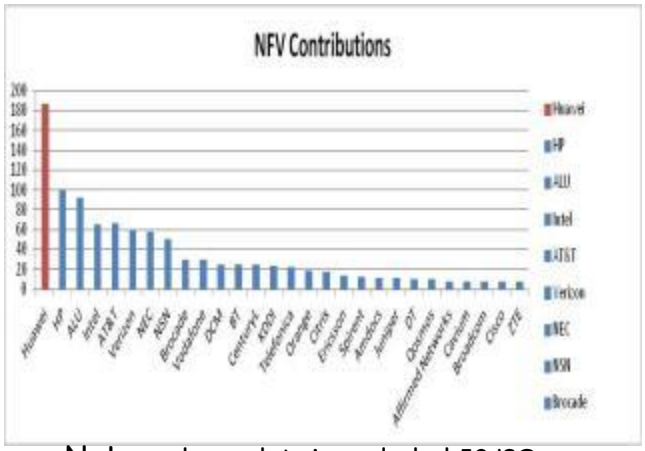
目录

- NFV数据面软件系统的发展趋势
- 从DPDK应用实践看数据面挑战
- DPDK架构的分析及PMD分层解耦设想
- DPDK的未来工作

华为在NFV中的贡献

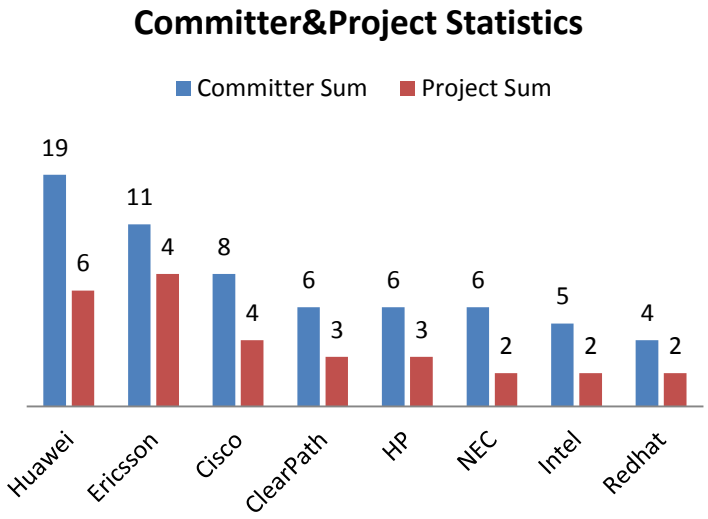
ETSI ISG NFV





Notes: above date is excluded 59 ISG contributions

OPNFV



- 1 工作组主席
- 1 技术项目负责人
- 2 汇报人和数名key editors
- 6 全职NFV标准主管
- 15 全职ISG NFV参会者
- 15+ 贡献者
- 236 贡献 (No.1)

HUAWEI

- 华为是OPNFV Funding成员之一，也是白金会员
- 华为在OPNFV项目中有董事会投票权和技术表决权
- 华为在19个正式项目中发起了6 个，在所有参与公司中排名**第一**
- 华为在OPNFV中获得19位committer职位，在所有参与公司中排名**第一**

NFV的愿景

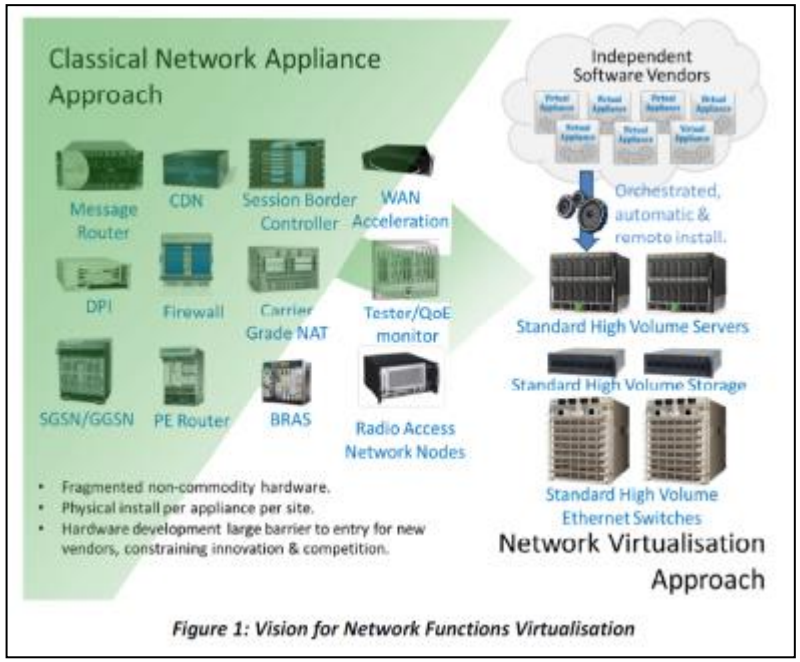


Figure 1: Vision for Network Functions Virtualisation

摘自NFV白皮书

●NFV目标：采用IT技术，用标准servers、switches、storage实现网络功能

●NFV价值：减少CAPEX、OPEX、设备占用空间、能耗、提升运营商业务和网络灵活性。

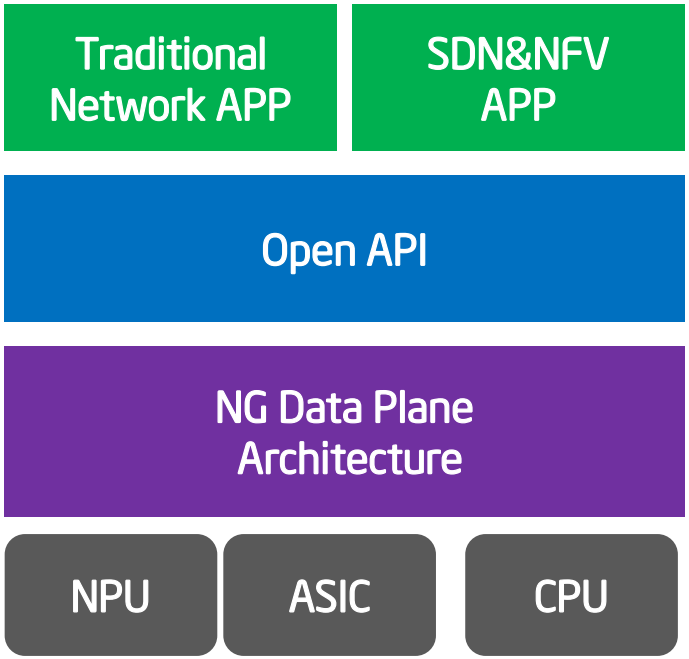
NFV固网和无线网关功能列表

应用领域	主要网络功能			
交换功能	BNG	CG-NAT	Router	
移动网功能		RNC	SGSN/MME	GGSN/S-GW/P-GW
隧道网关功能	IPSec/SSL	VPN GW		
业务分析功能	DPI	QoE		
NGN功能	SBC	IMS		
网络公共功能	Policy Control	AAA	Charging	
应用优化功能	Load Balancer	CDN		Cache Servers
安全功能	Virus Scanners	Firewalls		
其他功能	Service Assurance	SLA monitoring		

NFV对网络数据面提出了新课题，通用CPU及服务器技术、虚拟化技术、通用编程技术为网关的数据面实现提供了新的途径，如x86 CPU及其虚拟化技术、DPDK开发包

电信设备数据面的发展趋势

•未来数据平面的架构及其趋势



传统物理网络与SDN/NFV平滑兼容，并存、互通

开放，灵活编程

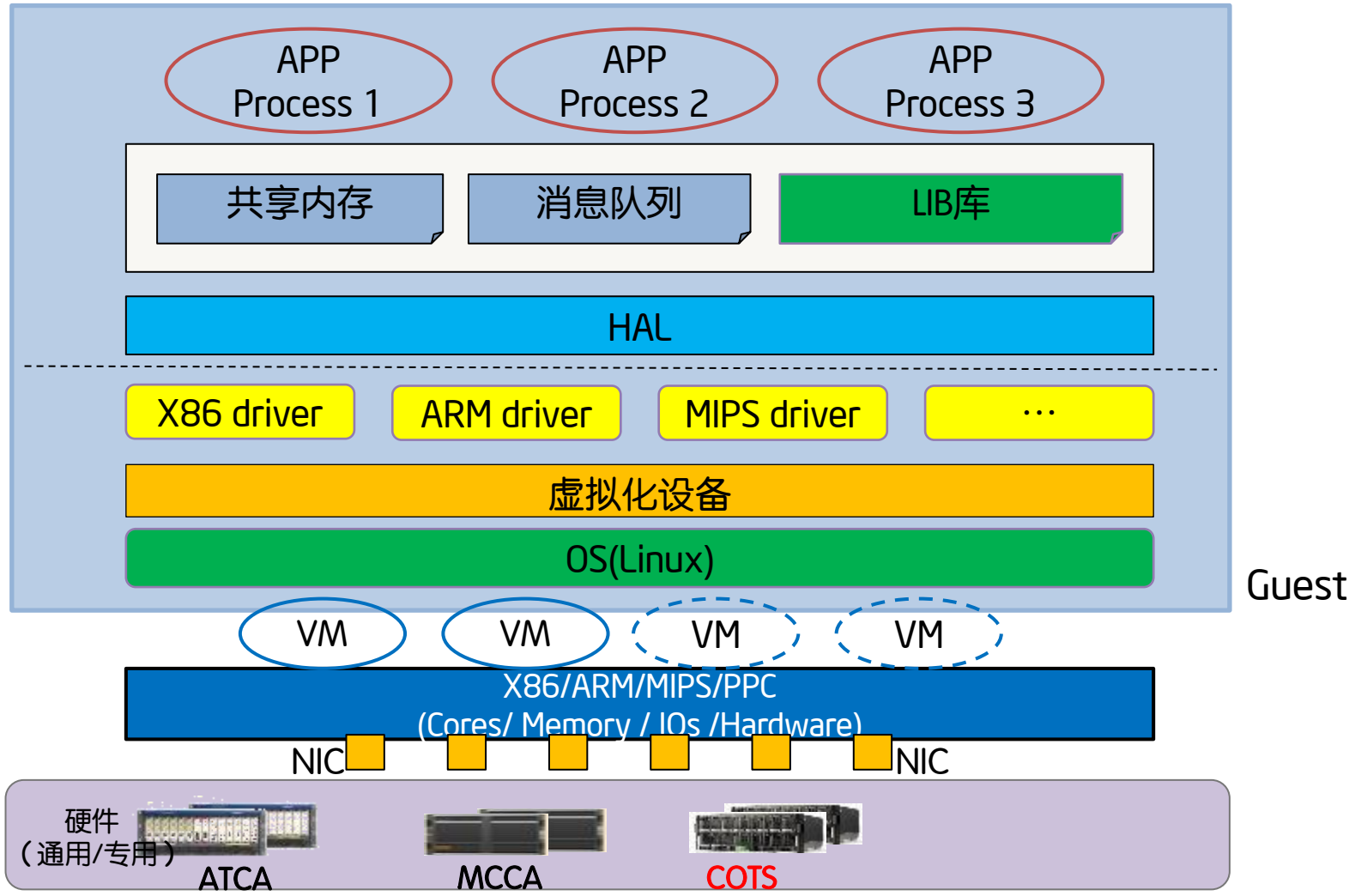
统一架构，高性能，大容量，低时延，低功耗

支持异构芯片及硬件

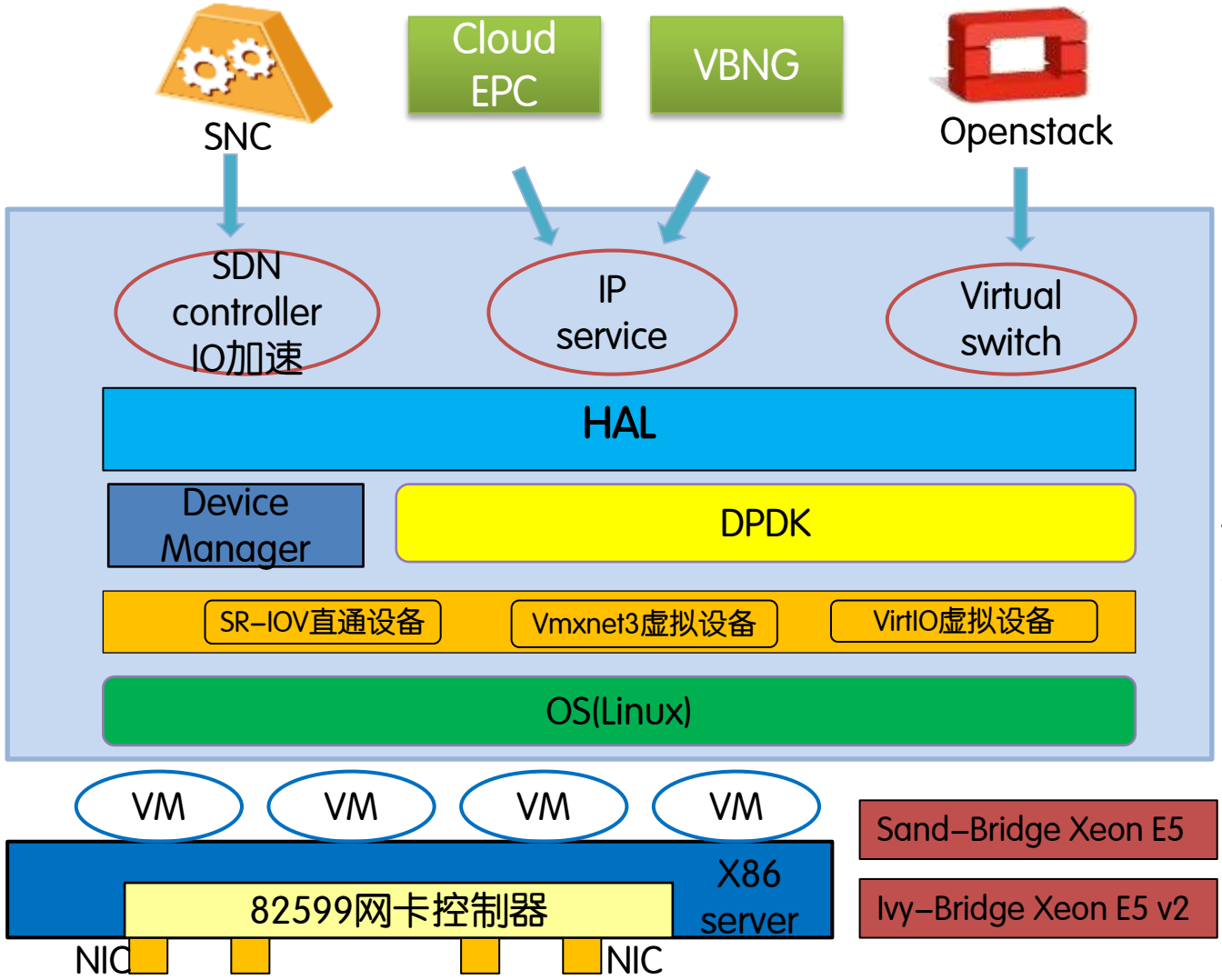
•未来数据平面的特征

•软、硬转发异构结合的SDN弹性架构，使系统达到最佳能效比
•硬转发应对高性能，软转发应对业务灵活性，软、硬转发在各自适用的场景均衡发展
•NPU/ASIC构建的硬转发电信设备经过长期在网运行的考验，仍将长期发展
•基于通用CPU构建NFV网关的软转发是未来的重要方向

NFV软转发的系统架构



DPDK在NFV数据面中的应用实践

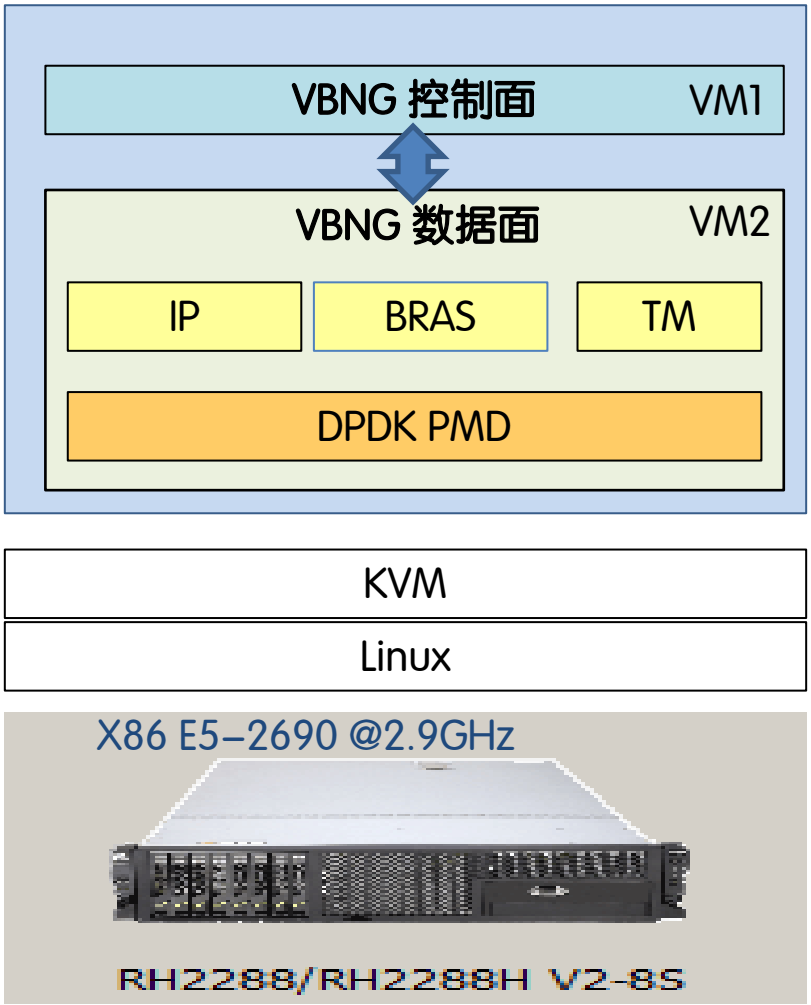


DPDK为SDN控制器、IP业务、虚拟交换提供数据面IO、x86虚拟化及驱动编程功能

HUAWEI

从实践看NFV通用CPU转发的挑战1–性能与时延

测试环境



vBNG PPPOE用户侧到网络侧性能测试数据

1、IP转发性能(1.5M PPS/Thread)（ 包长64Bytes ）
2、IP转发 + 一次CAR 约下降 90KPPS （1.45M PPS/Thread)
3、IP转发 + 一次CAR + 一次 ACL下降约 290KPPS (1.2M PPS/Thread)

VM软转发的时延和抖动都比硬件路由器大

Avg Latency (us)	Min Latency (us)	Max Latency (us)
83.74	36.47	352.41
105.88	52.86	3,609.97
99.61	43.87	4,220
119.38	48.25	5,698.20
177.05	37.2	128,594.56
364.46	62.29	152,987.52

注：硬件路由器单设备平均时延通常<30us)

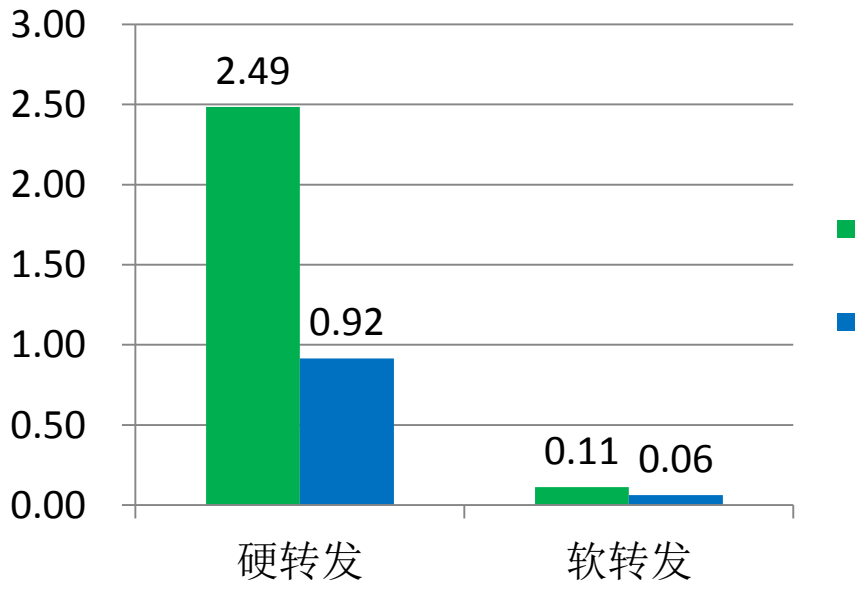
转发时延、性能以RFC 2544规定的 0丢包作为测量标准的，而现在有些发布的性能数据并未遵守这个标准，比如有的是在万分之一丢包时的数据

NFV通用CPU转发的挑战2–能效比

- LPUF120单板典型功耗：195W
- 58XX NP：功耗为~70W
  - IP　性能为 178.5MPPS（120G线速64Bytes小包性能）
  - IPOE性能为120　MPPS

LPUF120单板

- RH2288 Server典型功耗：489W
- X86 E5–2690 @2.9GHz 功耗：135W
  - IP 性能为 1.9*16=30.4MPPS（ RH2288 Server，2 CPU，16 core）

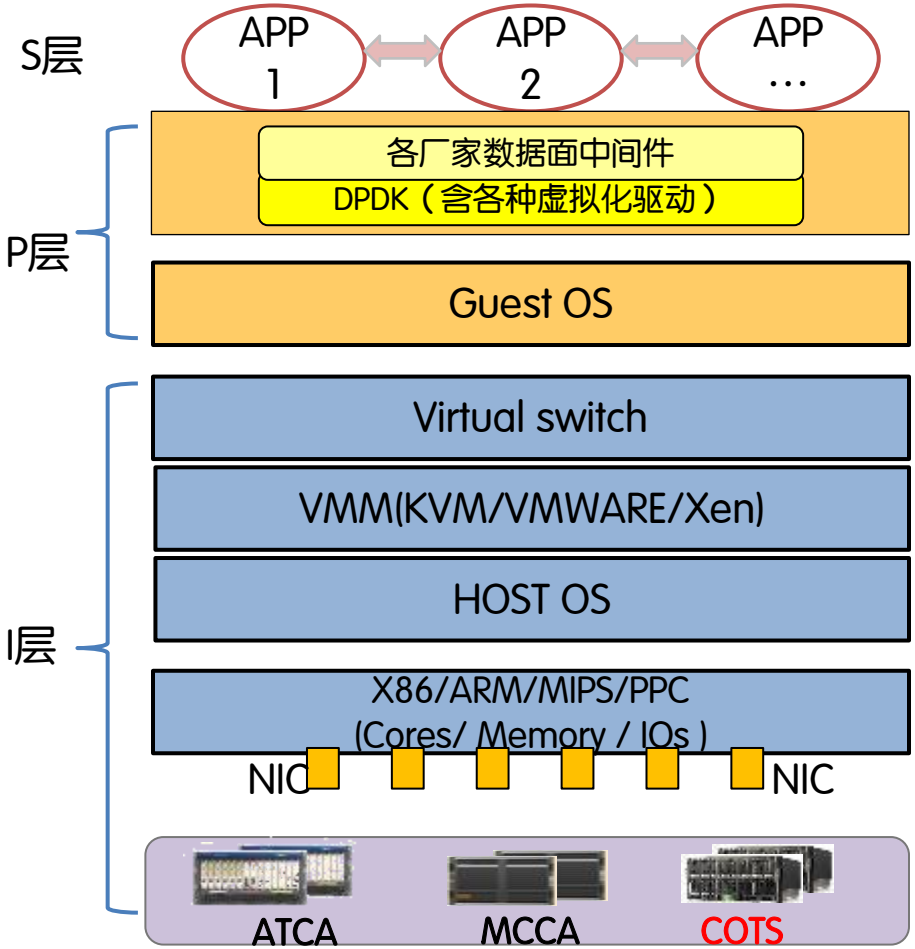


以当前的测试数据来比较：

- NP 58XX vs x86 E5的能效比>22
- LPUF120 vs RH 2288的能效比>15

NFV通用CPU转发的挑战3–集成环境

•NFV水平分层的集成环境对电信设备而言变得更加复杂，需要建立一个更加开放、可信、可靠的集成环境，DPDK的可靠性、可维护性功能非常重要



•各厂家、开源APP的互联互通缺乏标准

•内存共享机制、VM之间高速通信与数据包传递、安全隔离等优化技术复杂

•虚拟交换版本多样化，功能、性能存在差异

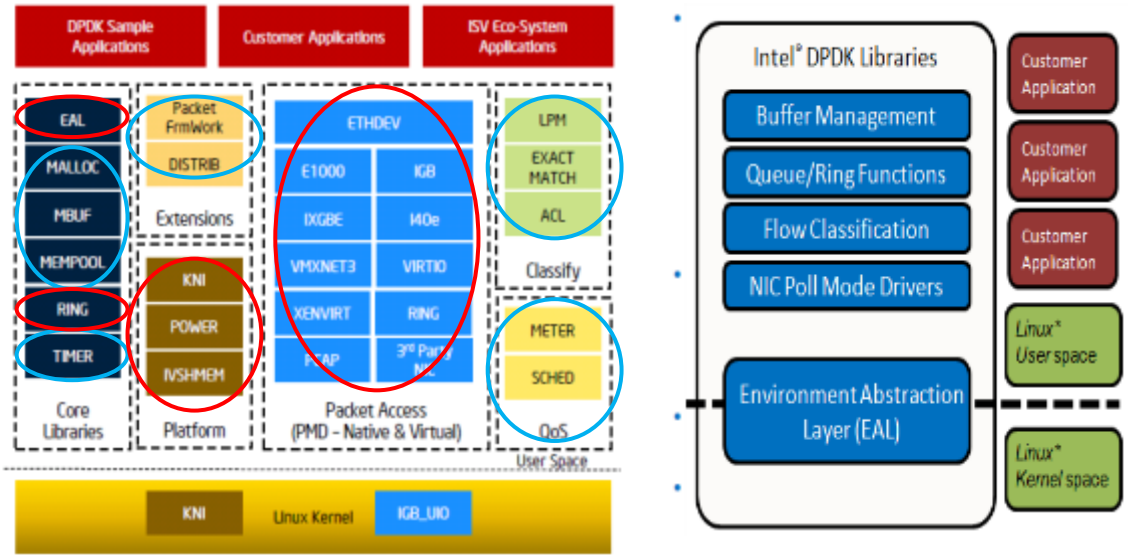
•虚拟化环境多样化，性能存在差异

•CPU及其驱动有待在电信环境下进一步检验，其异常影响上层APP的稳定性还难以预测

•网卡种类繁多，网卡驱动的可靠性及API稳定性很关键

•硬件通用化，服务器品牌多样化，通用服务器对于电信设备还需要进一步标准化

DPDK的应用–概述

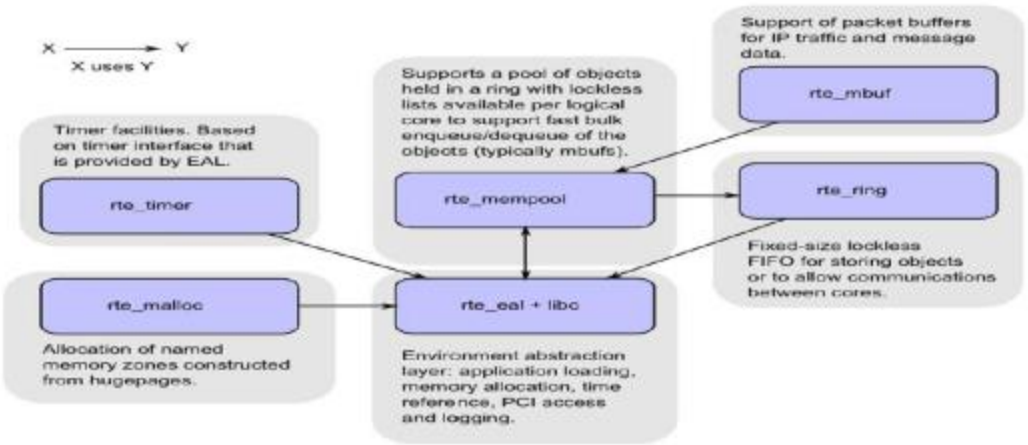


图中：

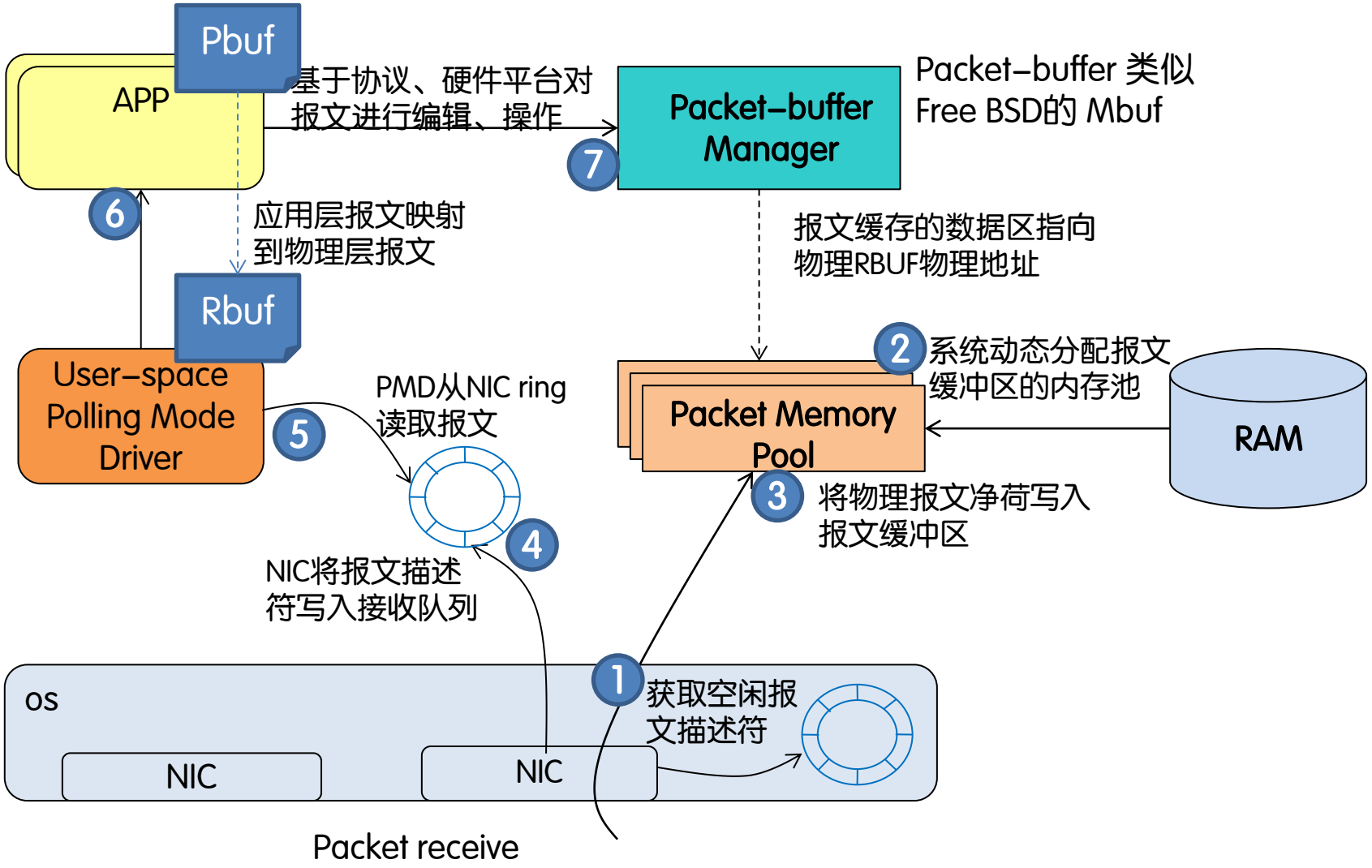
- 红色圈圈模块与硬件相关
- 蓝色圈圈模块为纯软件功能

几个核心模块的依赖关系

Figure 1. Core Components Architecture



应用DPDK进行报文处理的流程



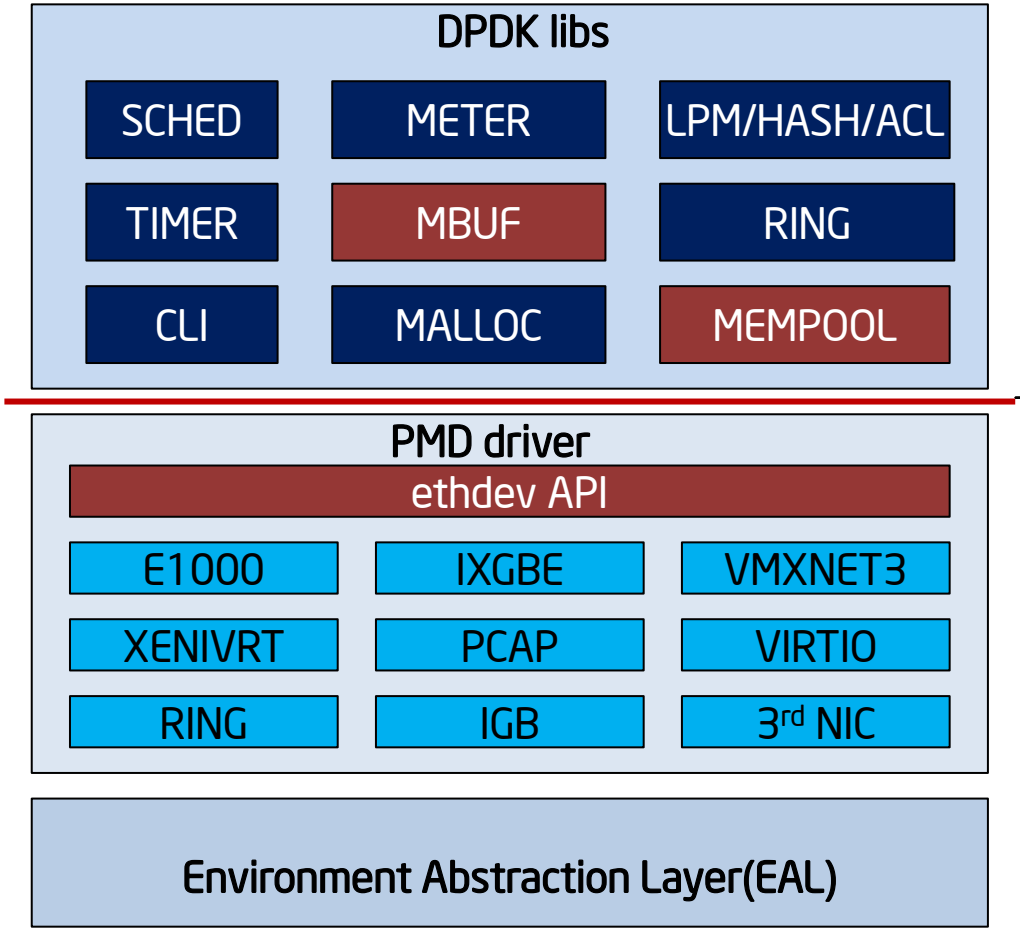
DPDK的性能和多芯片适应性的若干问题

- **报文零拷贝**
  - Packet buffer管理要能适应各种CPU、硬件加速芯片和业务特征
  - Packet buffer 数据结构应更简单、短小
- **报文缓冲区内存池动态收缩**
  - 报文缓冲区内存需要由系统统一分配，数据面采用与整个软件系统相同的内存操作API

从问题看几个PMD可改进的方向

- PMD增加raw packet send and receive API, 与DPDK core LIBs解耦，不依赖于 DPDK Rte_Mbuf 和 MemPool;
- PMD 可以独立发布LIB,与DPDK core libraries解耦;
- PMD增强故障诊断和维护功能，提供较好的故障定界能力;
- Ethdrv 提供一个统一稳定的PMD API ,屏蔽所有网卡对外 接口的差异，PMD升级不导致上层应用软件联动升级

增强DPDK PMD多芯片适应性的设想–分层解耦



✓驱动层与框架层解耦
✓驱动层对外呈现出裸报文接口
✓驱动层内部所需Buf接口抽象为钩子函数

DPDK耦合现状分析

```
uint16_t
rte_eth_rx_burst(uint8_t port_id, uint16_t queue_id,
        struct rte_mbuf **rx_pkts, uint16_t nb_pkts)
```

```
uint16_t
rte_eth_tx_burst(uint8_t port_id, uint16_t queue_id,
        struct rte_mbuf **tx_pkts, uint16_t nb_pkts)
```

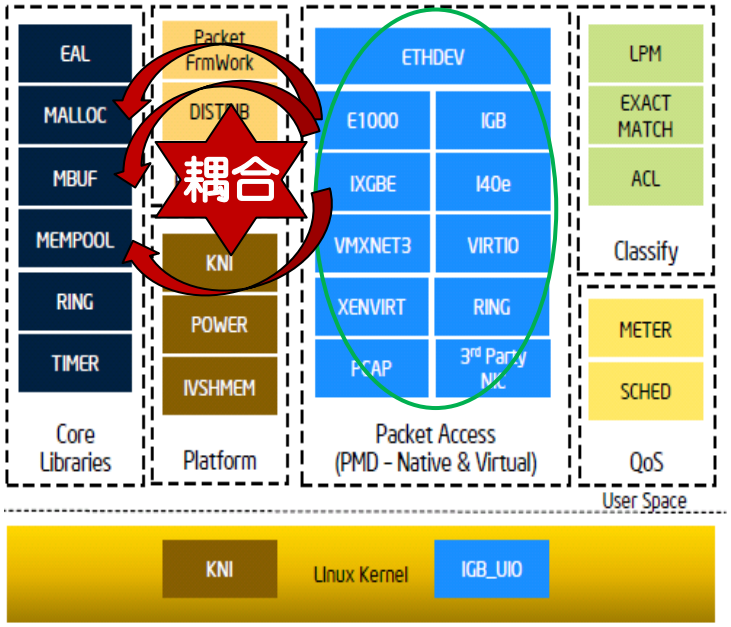
```
uint16_t
ixgbe_recv_pkts(void *rx_queue, struct rte_mbuf **rx_pkts,
        uint16_t nb_pkts)
{
    struct igb_rx_queue *rxq;
    volatile union ixgbe_adv_rx_desc *rx_ring;
    volatile union ixgbe_adv_rx_desc *rxdp;
    struct igb_rx_entry *sw_ring;
    struct igb_rx_entry *rxe;
    struct rte_mbuf *rxm;
    struct rte_mbuf *nmb;
```

```
/* free buffers one at a time */
if ((txq->txq_flags & (uint32_t)ETH_TXQ_FLAGS_NOREFCOUNT) != 0) {
    for (i = 0; i < txq->tx_rs_thresh; ++i, ++txep) {
        rte_mempool_put(txep->mbuf->pool, txep->mbuf);
        txep->mbuf = NULL;
    }
} else {
    for (i = 0; i < txq->tx_rs_thresh; ++i, ++txep) {
        rte_pktmbuf_free_seg(txep->mbuf);
        txep->mbuf = NULL;
    }
}
```

```
static inline int
ixgbe_rx_alloc_bufs(struct igb_rx_queue *rxq)
{
    volatile union ixgbe_adv_rx_desc *rxdp;
    struct igb_rx_entry *rxep;
    struct rte_mbuf *mb;
    uint16_t alloc_idx;
    uint64_t dma_addr;
    int diag, i;

    /* allocate buffers in bulk directly into the S/W ring */
    alloc_idx = (uint16_t)(rxq->rx_free_trigger -
                (rxq->rx_free_thresh - 1));
    rxep = &rxq->sw_ring[alloc_idx];
    diag = rte_mempool_get_bulk(rxq->mb_pool, (void *)rxep,
                rxq->rx_free_thresh);
```

```
static inline struct rte_mbuf *
rte_rxmbuf_alloc(struct rte_mempool *mp)
{
    struct rte_mbuf *m;

    m = __rte_mbuf_raw_alloc(mp);
    __rte_mbuf_sanity_check_raw(m, RTE_MBUF_PKT, 0);
    return (m);
}
```

```
int
ixgbe_dev_rx_queue_setup(struct rte_eth_dev *dev,
        uint16_t queue_idx,
        uint16_t nb_desc,
        unsigned int socket_id,
        const struct rte_eth_rxconf *rx_conf,
        struct rte_mempool *mp)
```



PMD改进1–提供裸报文API

- **解耦：**不依赖于rte_mbuf; eth_tx()和eth_rx()对外仅呈现基本的packet信息(addr, length) 。
- **解耦及增加可靠性：**不依赖与mempool；各PMD driver内部的mempool_alloc()则采用钩子函数，由初始化时候注册。eth_dev_init_t()函数增加钩子函数的注册参数。

```
struct raw_buf
{
      void*                 data;
      phys_addr_t        phys_addr;
      uint32_t sop       : 1;
       uint32_t eop       : 1;
      uint32_t resv14    : 14;
      uint32_t data_len:16; /* length of segment buffer */
};

uint16_t rte_eth_tx_burst(uint8_t port_id, uint16_t queue_id, struct raw_buf **tx_pkts, uint16_t nb_pkts);
uint16_t rte_eth_rx_burst(uint8_t port_id, uint16_t queue_id, struct raw_buf **rx_pkts, uint16_t nb_pkts);

/* <obj_table> include <data: virtual addr> and <phys_addr> */
typedef int (*raw_buf_bulk_get_hook)(void* mempool, void **obj_table, unsigned n);
typedef int (*raw_buf_bulk_free_hook)(void* memaddr);

typedef int (*eth_dev_init_t)(struct eth_driver   *eth_drv,  struct rte_eth_dev *eth_dev,
                                                raw_buf_bulk_get_hook *raw_buf_get,  raw_buf_bulk_free_hook
*raw_buf_free);
```

PMD改进2–网卡功能卸载 API探讨

•网卡卸载功能如何解耦及定义：

➢大部分场景下会使用标准能力，或者不会用到卸载能力。如何精简rte_mbuf?

➢如何定义统一标准化的raw_buf结构，既能满足driver API的解耦又能完成软硬件间的metadata传递?

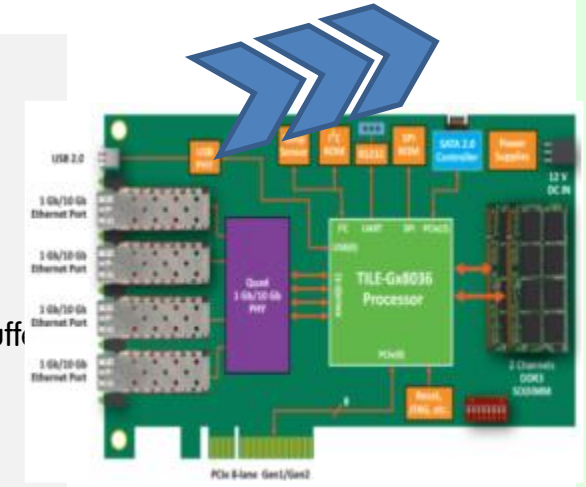
•解决方案探讨：

✓raw_buf中预留rx,tx硬件卸载特性offload_data，各厂家针对自身特性做私有定义(谁用谁定义)

```
uint32_t rx_offload_data[xxx];
uint32_t tx_offload_data[yyy];
```

随着网卡卸载特性的增加，软硬件间的metadata如何传递？并且不损失运行时刻的分层解耦?

```
struct raw_buf
{
    void*          data;
    phys_addr_t    phys_addr;
    uint32_t sop   : 1;
    uint32_t eop   : 1;
    uint32_t resv14  : 14;
    uint32_t data_len:16; /* length of segment buff
*/
   uint32_t rx_offload_data[xxx];
   uint32_t tx_offload_data[yyy];
};
```



Ref: rte_mbuf结构采自dpdk1.8.0

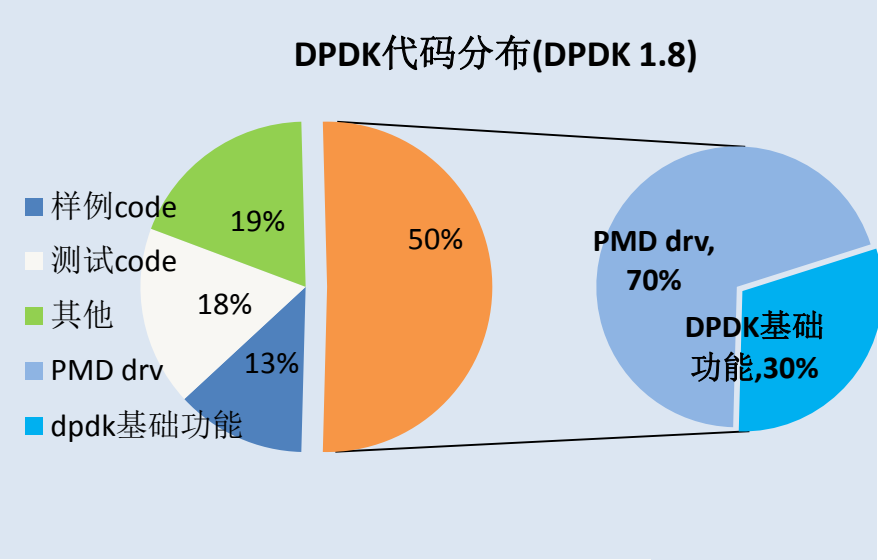
```
/**
 * The generic rte_mbuf, cont
 */
struct rte_mbuf {
    MARKER cacheline0;

    void *buf_addr;
    phys_addr_t buf_physaddr;

    uint16_t buf_len;

    /* next 6 bytes are initi
    MARKER8 rearm_data;
    uint16_t data_off;

    uint16_t packet_type;
    uint16_t data_len;          /**< Amount of data i
    uint32_t pkt_len;           /**< Total pkt len: s
    uint16_t vlan_tci;          /**< VLAN Tag Control
    uint16_t reserved;
    union {
        uint32_t rss;     /**< RSS hash result if R
        struct {
            union {
                struct {
                    uint16_t hash;
                    uint16_t id;
                };
                uint32_t hi;
                /**< First 4 flexible bytes or FD ID,
                    PKT_RX_FDIR_* flag in o_fl
            } fdir;           /**< Filter identi
            uint32_t sched;   /**< Hierarchical
            uint32_t usr;     /**< User defined
        } hash;               /**< hash info
    union {
        void *userdata;   /**< Can be us
        uint64_t udata64; /**< Allow 8-b
    };
    struct rte_mempool *pool; /**< Pool
    struct rte_mbuf *next;    /**< Next

    /* fields to support TX offloa
    union {
        uint64_t tx_offload;       /**< combin
        struct {
            uint64_t l2_len:7; /**< L2 (MAC) H
            uint64_t l3_len:9; /**< L3 (IP) He
            uint64_t l4_len:8; /**< L4 (TCP/UD
            uint64_t tso_segsz:16; /**< TCP TS
            /* fields for TX offloading of tun
            uint64_t outer_l3_len:9;  /**< Oute
            uint64_t outer_l2_len:7;  /**< Oute

            /* uint64_t unused:8; */
        };
    };
```

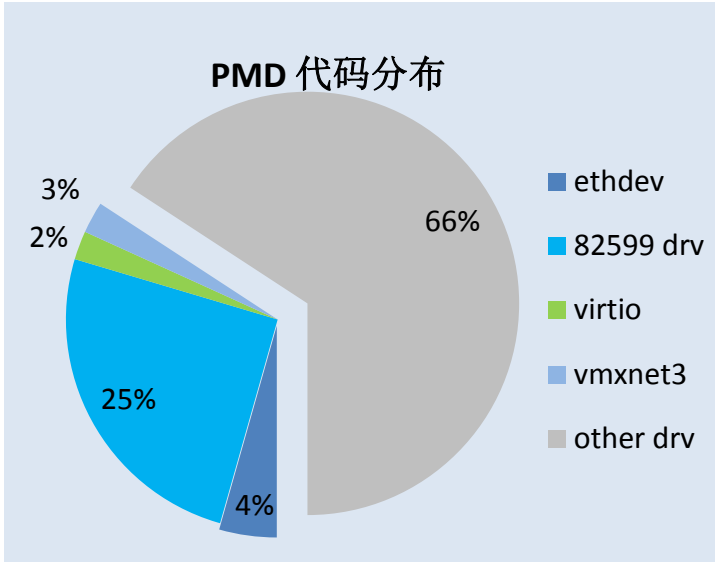
RX卸载特性

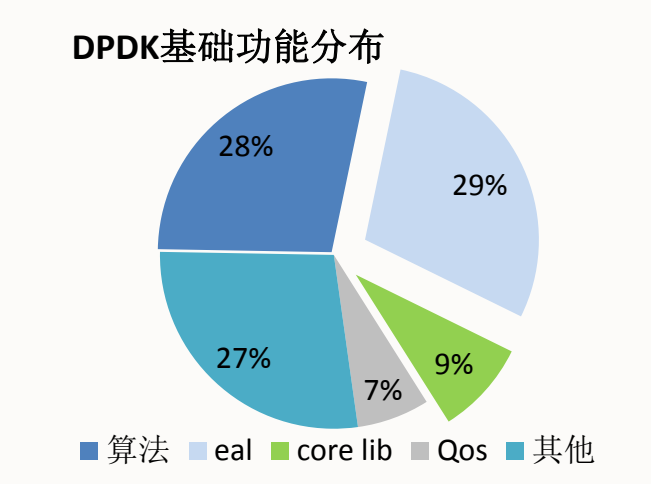
TX卸载特性

定义过于X86定制化，不利于兼容所有厂商网卡

PMD改进3–增强可靠性和可维护性







- PMD驱动部分占DPDK核心代码比重约70%
- PMD驱动中，常用的几种vNic驱动仅占34%

PMD是DPDK中最关键的模块，PMD的代码质量与API的稳定性决定了DPDK在数据面的可用性。

DPDK的未来工作

- **提升转发性能效率**
  - 转发性能线性增长能力：性能随CPU核资源scale up，线性增长；性能随VM scale out，线性增长
  - 需要提升通用CPU转发的能效比，用“转发性能效率”来比较通用服务器实现电信网关设备的可用性，而非简单的“转发性能”
- **支持网卡即插即用**
  - 支持各种芯片驱动及各种网卡，网卡驱动升级不影响上层应用软件
- **增加可靠性功能**
  - 提供线程、内存、NIC队列等异常检测、告警、恢复
  - 支持NIC/VNIC热插拔
- **增强可维护功能**
  - 提供调测统计、故障定位等可维护性功能

Thanks