# DPDK IP分片与重组设计实现

DPDK开源社区 2016-11-02

作者 杨浩鹏


## 前言

IP的分片和重组会影响效率，应该尽量避免分片包的形成。在TCP中，已经自己限定了MSS，默认是536字节，不会形成IP分片。所以值得注意的就是UDP包和ICMP，他们并未对此作出限定。因此，在发送数据包时需要注意包的大小。在这篇文章中，我们主要分析在DPDK中，是怎么设计分片和重组的。

**设计时要考虑的问题：**

分片：

1.如何确定要进行分片？

2.分片都要做哪些事情？

重组：

1.如何确定是分片包？

2.如何确定分片包已经到齐了？

3.如何存储这些分片包？如何查找？

4.重组过程中都需要做些什么事？

接下来我们将依次对以上的问题进行解答。

## 1 分片

分片需要解决的问题相对较少。主要有两个：第一，如何判断是否需要分片（若报文的长度大于1500字节且在分片标志上又允许分片，则需要分片）。第二，在分片时都需要做些什么事？如果不允许分片，那么IP层就直接把数据包丢弃，同时，发送一个ICMP的错误回应报文给源端。

在分片时，需要把原来的数据报的IP头都给新的分片一个拷贝，要注意如果是UDP报文，并不会把UDP报文头也拷贝给每一个分片，为什么不拷贝UDP头呢？因为在IP层的任务是传送数据包到正确的主机，只需要IP头就可以做到，并不需要额外添加负载。当然，如果一开始实现时拷贝了完整的UDP报文头，多拷贝20字节，则现在就可能不会遇到NAT穿越中的问题等，但是在IP分片实施时，也未必会料到会有NAT出现，这都是题外话了。在拷贝完IP头部后，需要重新设置报文的长度和校验和。

接下来看一下在DPDK中是怎么组织实现的（以2.1版本为例）。

分片的起点是在rte_ipv4_fragment_packet()函数中开始的，在了解实现之前，不妨看一下传递的参数，主要是要分片的报文，分片后存储的报文，mtu的大小，direct pool和indirect pool。DPDK为了提高报文处理速度，使用了零拷贝技术，关键就是使用了这两个pool。

**主要步骤如下：**

1.判断报文的有效长度是不是8的倍数。
2.取出分片标志和偏移字段。
3.如果设置了禁止分片，那么就返回错误，没有进一步返回ICMP错误通知源端。
4.检查分片是否能承载原报文长度，不能就返回错误。
5.先在direct pool里分配一个out_pkt，填充报文长度为IP头长。
6.循环在indirect pool里分配一个out_seg,挂到第一个out_pkt后面。
7.把新分配的out_seg attach到in_seg，调整out_seg的长度等。
8.检查out_pkt是否已经够mtu大小了，如果够了，就下一步处理。
9.填充out_pkt头，设置校验和标志等，设置长度，然后放到pkts_out中返回发送。

## 2 重组

重组涉及到多个点项，判断分片包是根据头的标志和偏移来做的，存储与查找是由hash表来实现的，此外还有表项的老化机制。确定包是否到齐了可以通过接收的片的长度跟第一片中的报文长度比较。

重组是从rte_ipv4_frag_reassemble_packet()中开始的，同样先看一下参数：首先是tbl表，用以存储查找条目，然后是dr表，用于存储超时的分片，然后是进来重组的报文，时间戳，以及IP头指针。

**主要的框架流程如下：**

1.根据进来的报文填充key值，以便后面查找使用。
2.ip_frag_find()，查找，如果找到条目，下一步就是重组。
3.ip_frag_process()，这一步是重组的重点，重组成功就会返回重组后的报文。

整理框架是很清晰，定义了多个结构体，这些结构体体现了设计思路。在详细看流程实现之前，可以简单来分析一下这些结构体的用意：

这几个结构体定义在rte_ip_frag.h中，

```
struct ip_frag {
uint16_t ofs;          /**< offset into the packet */
uint16_t len;          /**< length of fragment */
struct rte_mbuf *mb;   /**< fragment mbuf */
};先封装了分片报文，其中的ofs会有什么用呢？
struct ip_frag_key {
uint64_t src_dst[4];      /**< src address, first 8 bytes used for IPv4 */
uint32_t id;           /**< dst address */
uint32_t key_len;      /**< src/dst key length */
```

};定义了hash的key结构，从定义和文档中知道，DPDK的IP重组的HASH表的KEY是以{src_ip,dst_ip,packet_id}确定的。使用的Jhash算法。

```
struct ip_frag_pkt {
TAILQ_ENTRY(ip_frag_pkt) lru;   /**< LRU list */
struct ip_frag_key key;          /**< fragmentation key */
uint64_t           start;       /**< creation timestamp */
uint32_t           total_size;  /**< expected reassembled size */
uint32_t           frag_size;   /**< size of fragments received */
uint32_t           last_idx;    /**< index of next entry to fill */
struct ip_frag       frags[IP_MAX_FRAG_NUM]; /**< fragments */
} __rte_cache_aligned;
```

这是个重要的结构，代表了一个hash的entry，在这个entry中存储的是这个包的分片。其中有时间戳，刚说到会有老化，有total_size，期望接收的重组长度，用以判断分片是否接收完成。

```
struct rte_ip_frag_death_row {
uint32_t cnt;          /**< number of mbufs currently on death row */
struct rte_mbuf *row[IP_FRAG_DEATH_ROW_LEN * (IP_MAX_FRAG_NUM +
1)];
/**< mbufs to be freed */
};
```

老化的表项。

```
 struct rte_ip_frag_tbl {
uint64_t           max_cycles;      /**< ttl for table entries. */
uint32_t           entry_mask;      /**< hash value mask. */
uint32_t           max_entries;     /**< max entries allowed. */
uint32_t           use_entries;     /**< entries in use. */
uint32_t           bucket_entries;  /**< hash assocaitivity. */
uint32_t           nb_entries;      /**< total size of the table. */
uint32_t           nb_buckets;      /**< num of associativity lines. */
struct ip_frag_pkt *last;         /**< last used entry. */
struct ip_pkt_list lru;           /**< LRU list for table entries. */
struct ip_frag_tbl_stat stat;     /**< statistics counters. */
struct ip_frag_pkt pkt[0];        /**< hash table. */
};
```

分片tbl表，定义了整个hash表。如使用的条目数量，上次使用的条目（先查这个，能提高速度）。

**接下来，我们梳理一下重组的主要操作。**

1.填充hash的key结构体，这个比较简单。

2.查找或者添加条目。

step 1:用<src_ip,dst_ip,pkt_id>生成的key在fragment table中查找。

step 2:如果条目查找到，检查条目是否超时。如果超时，那么释放所以之前接收的分片，并移除他们在条目中的相关信息。
setp 3:如果使用那个key在表中没有找到对应的条目，那么会尝试使用两种方法来创建一个新的：
a)使用一个空的条目
b)删除一个已经超时的条目，释放相关的Mbuf,重新存储一个新的key在里面。
step 4:更新分片的信息，检查数据包是否能被重组（entry中已经包含了所有的分片。）
a)如果检查分片都已经到齐了，则进行重组，把entry标记为空，同时，返回重组后的mbuf指针给调用者。
b)如果没有到齐，则重组结果就是NULL。（以上步骤来自于DPDK官网，如有需要，请点击阅读全文。）

## 总结

从整个的设计来看，分片设计好坏的差别在于效率，尽量避免拷贝，在DPDK中使用indirect mbuf的方法实现零拷贝。而重组设计最重要的部分当属分片包的存储查找。在DPDK和Linux的设计中，都使用了hash表来完成存储查找，因为条目规模不是很大，这也是一个hash表设计使用的范例，值得借鉴。
作者简介：
杨浩鹏 主要研究领域：LTE EPC网络方面。