**异构网络编程接口**

**函数接口及作用**

1. **发送端发送数据包相关函数**



**1.1封装函数**

**Socket结构体：**



**Socket 结构体定义**

typedef struct \_socket\_addr{

PMAC\_HEADER machdr; //MAC头结构体指针

PMAC\_TAIL mactail; //MAC尾结构体指针

PIP\_HEADER iphdr; //IP头结构体指针

PUDP\_HEADER udphdr; //UDP头结构体指针

}SOCKET\_ADDR,\*PSOCKET\_ADDR;

**1.1.1添加UDP头函数**

**int udp\_packaging(struct rte\_mbuf \*m,****PSOCKET\_ADDR sock);**

**接口**：输入：要添加UDP头部的mbuf和填充的socket结构体，

输出：0：UDP头部添加成功，-1：UDP头部添加失败

**作用：**在传入的mbuf的data部分前开辟空间用于添加socket结构体中UDP头部

**UDP头部结构体定义**

typedef struct \_udphdr{

unsigned short SourcePort; //2字节，16位源端口号

unsigned short DestPort; //2字节，16位目的端口号

unsigned short Length; //2字节，16位数据包长度

unsigned short Checksum； //2字节，16位校验和

}UDP\_HEADER,\*PUDP\_HEADER;

**1.1.2添加IP头函数**

**int ip\_packaging(struct rte\_mbuf \*m, PSOCKET\_ADDR sock);**

**接口**： 输入：要添加IP头部的mbuf和填充的socket结构体，

输出：0：IP头部添加成功，-1：IP头部添加失败

**作用：**在传入的mbuf的data部分前开辟空间用于添加socket结构体中IP头部

**IP头结构体定义**

typedef struct \_iphdr{

unsigned char Lenver; //1字节，4位首部长度+4位IP版本号

unsigned char Tos; //1字节，8位服务类型TOS

unsigned short Total\_Len; //2字节，16位总长度

unsigned short Ident； //2字节，16位标识

unsigned short Frag\_And\_Flags; //2字节，3位标志位+13位分片偏移

unsigned char TTL; //1字节，8位生存期

unsigned char Proto; //1字节，8位协议（TCP，UDP或其他）

unsigned short Checksum； //2字节，16位头部校验和

unsigned int SourceIP； //4字节，32位源IP地址

unsigned int DestIP； //4字节，32位目的IP地址

}IP\_HEADER,\*PIP\_HEADER;

**1.1.3添加MAC帧函数**

**int mac\_packaging(struct rte\_mbuf \*m, PSOCKET\_ADDR sock);**

**接口**：输入：要添加MAC头部和MAC尾部的mbuf和填充的socket结构体

输出：0：MAC帧添加成功，-1：MAC帧添加失败

**作用：**在传入的mbuf的data部分前后分别开辟空间用于添加socket中的MAC帧头和帧尾

**MAC帧头结构体**

typedef struct \_mac\_hdr{

unsigned char SourceMacAddr[6]; //6字节，源MAC地址

unsigned char DestMacAddr [6]; //6字节，目的MAC地址

unsigned short Type； //2字节，上一层的协议类型

}MAC\_HEADER,\*PMAC\_HEADER;

typedef struct \_mac\_tail{

unsigned int Checksum; //4字节，数据帧尾校验和

}MAC\_TAIL,\*PMAC\_TAIL;

**1.2构造数据帧函数**

**数据帧结构体**



**数据帧头结构体定义**

typedef struct data\_frame{

MAC\_HEADER machdr; //14字节，MAC帧头部

IP\_HEADER iphdr; //20字节，IP帧头部

UDP\_HEADER udphdr; //8字节，UDP帧头部

LLC\_HEADER llchdr; //4字节，LLC帧头部

Struct rte\_mbuf data; //若干字节，业务数据

MAC\_TAIL mactail; //4字节，MAC帧尾部

}DATA\_FRAME;

**1.2.1构造业务数据包函数**



**int business\_data\_packaing(struct rte\_mbuf \*m,DATA\_PACK data\_pack);**

**接口：**输入：要加载数据帧的mbuf和构造的数据帧的结构体

输出：0:数据包构造成功，-1：数据包构造失败

**作用：**将已经构造好的业务数据结构体添加到mbuf上，构造出业务数据包

**业务数据包结构体定义**

typedef struct data\_pack{

unsigned char Type; //1字节，数据类

unsigned short DataSeqNum; //2字节，数据包序号

unsigned char FragNum； //1字节，数据包分片号

struct rte\_mbuf data; //若干字节，IP数据包

}DATA\_PACK;

**1.2.2构造业务数据和ACK数据包函数**



**int business\_and\_ack\_packaing(struct rte\_mbuf \*m,DATA\_AND\_ACK data\_and\_ack);**

**接口：**输入：要加载数据包的mbuf和构造的数据和ACK合包的结构体

输出：0:数据包构造成功，-1：数据包构造失败

**作用：**将已经构造好的业务数据和ACK结构体添加到mbuf上，构造业务数据和ACK包

**业务数据包与ACK包结构体定义**

typedef struct \_data\_and\_ack{

unsigned char Type; //1字节，数据类型

unsigned short DataSeqNum; //2字节，数据包序号

unsigned char FragNum； //1字节，数据包分片号

unsigned short AckSeqNum; //2字节，ACK确认序号

struct rte\_mbuf data; //若干字节，IP数据包

}DATA\_AND\_ACK;

**1.2.3构造ACK包函数**



**int ack\_packaing(struct rte\_mbuf \*m,ACK\_PACK ack\_pack);**

**接口：**输入：要加载ACK包的mbuf和构造ACK包的结构体

输出：0:ACK包构造成功，-1：ACK包构造失败

**作用：**将已经构造好的ACK结构体添加到mbuf上，构造出ACK包

**ACK包结构体定义**

typedef struct \_ack\_pack{

unsigned char Type; //1字节，数据类型

unsigned short SeqNum; //2字节，ACK确认序号

}ACK\_PACK；

**1.2.4构造复位包函数**



**int reset\_packaing(struct rte\_mbuf \*m,RESET\_PACK reset\_pack);**

**接口：**输入：要加载复位包的mbuf和构造好的复位包的结构体

输出：0:复位包构造成功，-1：复位包构造失败

**作用：**将已经构造好的复位结构体添加到mbuf上，构造出复位包

**复位包结构体定义**

typedef struct \_reset\_pack{

unsigned char Type; //1字节，数据类型

unsigned char Parameter[3]; //3字节，重置的参数类型

}RESET\_PACK;

**1.3构造链路控制帧函数**



**控制帧结构体定义**

typedef struct link\_control\_frame{

MAC\_HEADER machdr; //14字节，MAC帧头部

IP\_HEADER iphdr; //20字节，IP帧头部

UDP\_HEADER udphdr; //8字节，UDP帧头部

Struct rte\_mbuf data; //若干字节，链路控制包

MAC\_TAIL mactail; //4字节，MAC帧尾部

}LINK\_CONTROL\_FRAME;

**1.3.1构造链路探测包函数**



**int reply\_packaing(struct rte\_mbuf \*m,REPLY\_PACK reply\_pack);**

**接口：**输入：要加载链路探测包的mbuf和构造好的链路探测包的结构体

输出：0:链路探测包构造成功，-1：链路探测包构造失败

**作用：**将已经构造好的链路探测包结构体添加到mbuf上，构造出链路探测包

**链路探测包结构体定义**

typedef struct \_seek\_pack{

unsigned char Type; //1字节，数据类型

unsigned int Length; //4字节，各部分长度

unsigned short Value; //2字节，链路探测包序号

}SEEK\_PACK;

**1.3.2构造链路响应包函数**



**int reply\_packaing(struct rte\_mbuf \*m,REPLY\_PACK reply\_pack);**

**接口：**输入：要加载链路响应包的mbuf和构造好的链路响应包的结构体

输出：0:链路响应包构造成功，-1：链路响应包构造失败

**作用：**将已经构造好的链路响应包结构体添加到mbuf上，构造出链路响应包

**链路探测响应包结构体定义**

typedef struct \_reply\_pack{

unsigned char Type; //1字节，数据类型

unsigned int Length; //4字节，各部分长度

unsigned short Value; //2字节，链路探测响应包序号

}REPLY\_PACK;

**1.3.3构造ARP请求帧函数**



**int arp\_request\_packaing(struct rte\_mbuf \*m,ARP\_PACK arp\_pack);**

**接口：**输入：要加载ARP请求包的mbuf和构造好的ARP请求包的结构体

输出：0:ARP请求帧构造成功，-1：ARP请求帧构造失败

**作用：**将已经构造好的ARP请求包结构体添加到mbuf上，构造出ARP请求包

**ARP包结构体定义**

typedef struct \_arp\_pack{

unsigned short Hardware\_Type; //2字节，硬件类型

unsigned short Proto\_Type; //2字节，协议类型

unsigned char Hardware\_Length; //1字节，硬件地址长度

unsigned char Proto\_Length; //1字节，协议地址长度

unsigned short Type; //2字节，ARP数据包类型

unsigned char SourceMacAddr[6]; //6字节，发送端MAC地址

unsigned int SourceIPAddr; //4字节，发送端IP地址

unsigned char DestMacAddr[6]; //6字节，目的端MAC地址

unsigned int DsetIPAddr; //4字节，目的端IP地址

}ARP\_PACK;

**1.3.4构造ARP响应帧函数**

**int arp\_reply\_packaing(struct rte\_mbuf \*m,ARP\_PACK arp\_pack);**

**接口：**输入：要加载ARP响应帧的mbuf和构造好的ARP响应帧的结构体

输出：0:ARP响应帧构造成功，-1：ARP响应帧构造失败

**作用：**将已经构造好的ARP响应包结构体添加到mbuf上，构造出ARP响应包

**1.4发包函数**

**int pack\_send(struct rte\_mbuf \*\*m,int port,int flag)**

**接口：**输入：构造好的数据包mbuf和要发送数据帧的端口以及控制发包模式的flag

输出：0：包发送成功，-1：包发送失败

**作用：**将已经封装好的mbuf通过传入的端口号调用dpdk发包函数发送出去，同时通过设置flag位设置发包模式。

**1.5 ARP查询函数**

**int ARP\_query(PSOCKET\_ADDR sock)**

**接口：**输入：包含要查询MAC地址的IP信息的socket结构体指针

输出：要发送到对应MAC地址的端口号

**作用：**传入包含要查询MAC地址的IP信息的socket结构体指针，通过查询ARP缓存表信息查询到对应MAC地址信息，返回发送到对应MAC地址的端口号

**1.6网络策略选择处理函数**

**int transport\_selection(void)**

**接口：**

输出：返回根据策略选择选出的最优传输线路编号

**作用：**函数先判断各线路的通断信息，再在状态为通路的线路中选择出传输时延最小的线路为最优传输线路，返回其线路编号

1. **接收端处理包相关函数**



**2.1收包函数**

**int pack\_receive(struct rte\_mbuf \*m,int portid,int flag);**

**接口：**输入：接收数据的mbuf，接收数据的端口号和用于标识接收模式的标识位

输出：0：包接收成功，-1：包接收失败

**作用：**接收端轮训网卡从对应的端口号调用dpdk收包函数接收数据将其添加到传入的mbuf上，同时通过设置flag位设置收包模式。

**2.2解封装函数**

**2.2.1除去MAC帧函数**

**int mac\_unpackaing(struct rte\_mbuf \*m, PSOCKET\_ADDR sock);**

**接口：**输入：要去除MAC帧的mbuf和用于保存除去的MAC帧头帧尾信息socket结构体指针

输出：0:MAC帧去除成功，-1：MAC帧去除失败

**作用：**去除收到的mbuf的MAC帧头，同时传入socket结构体保存去除的MAC帧头帧尾的信息。

**2.2.2除去IP头函数**

**int ip\_unpackaing(struct rte\_mbuf \*m, PSOCKET\_ADDR sock);**

**接口：**输入：要去除IP头的mbuf和用于保存除去的IP帧头信息socket结构体指针

输出：0:IP头去除成功，-1：IP头去除失败

**作用：**去除收到的mbuf的IP帧头，同时传入socket结构体保存去除的信息。

**2.2.3除去UDP头函数**

**int udp\_unpackaing(struct rte\_mbuf \*m, PSOCKET\_ADDR sock);**

**接口：**输入：要去除UDP头的mbuf和用于保存除去的UDP帧头信息结构体指针

输出：0:UDP头去除成功，-1：UDP头去除失败

**作用：**去除收到的mbuf的UDP头，同时传入socket结构体保存去除的UDP头信息。

**2.3帧分类函数**

**int frame\_classfied(struct rte\_mbuf \*m);**

**接口：**输入：要分类的mbuf

输出：0:数据帧，1：链路控制帧 -1：分类失败

**作用：**传入解封装后的帧mbuf，根据判断帧是否存在LLC模块区分其是数据帧或链路控制帧。

**2.4接收数据帧处理函数**

**2.4.1数据帧分类处理函数**

**int pack\_classfied(struct rte\_mbuf \*m)**

**接口:**输入：要分类的数据帧mbuf

输出 ：0：处理成功，-1：处理失败

**作用：**传入数据帧mbuf，根据其帧头的TYPE对其进行分类。

**2.4.2接收业务数据包处理函数**

**int business\_pack\_process(struct rte\_mbuf \*m);**

**接口：**输入：要处理的含有业务数据包的mbuf

输出：0:业务数据包处理成功，-1：业务数据包处理失败

**2.4.3接收ACK包处理函数**

**int ack\_pack\_process(struct rte\_mbuf \*m);**

**接口：**输入：要处理的含有ACK包的mbuf

输出：0:ACK包处理成功，-1：ACK包处理失

**2.4.4接收业务数据ACK包处理函数**

**int business\_and\_ack\_process(struct rte\_mbuf \*m);**

**接口：**输入：要处理的含有业务数据和ACK合包的mbuf

输出：0:包处理成功，-1：包处理失败

**2.4.5接收复位包处理函数**

**int reset\_pack\_process(struct rte\_mbuf \*m);**

**接口：**输入：要处理的含有复位包的mbuf

输出：0:包处理成功，-1：包处理失败

**2.5接收链路控制帧处理函数**

**2.5.1数据帧分类处理函数**

**int control\_pack\_classfied(struct rte\_mbuf \*m)**

**接口:**输入：要分类的链路控制帧mbuf

输出 ：0：处理成功，-1：处理失败

**作用：**传入链路控制帧mbuf，根据帧的模块对其进行分类。

**2.5.2接收链路探测包处理函数**

**int seek\_pack\_process(struct rte\_mbuf \*m);**

**接口：**输入：要处理的含有链路探测包的mbuf

输出：0:链路探测包处理成功，-1：链路探测包处理失败

**2.5.3接收链路响应包处理函数**

**int reply\_pack\_process(struct rte\_mbuf \*m);**

**接口：**输入：要处理的含有链路响应包的mbuf

输出：0:链路响应包处理成功，-1：链路响应包处理失败

**2.5.4接收ARP请求包处理函数**

**int arp\_acque\_process(struct rte\_mbuf \*m);**

**接口：**输入：要处理的含有ARP请求包的mbuf

输出：0:ARP请求包处理成功，-1：ARP请求包处理失败

**2.5.5接收ARP响应包处理函数**

**int arp\_reply\_process(struct rte\_mbuf \*m);**

**接口：**输入：要处理的含有ARP响应包的mbuf

输出：0:ARP响应包处理成功，-1：ARP响应包处理失败

**2.6线路通断和往返时延函数**

**static void state\_and\_delay(TRANS\_STATUS trans\_status)**

**接口：**输入：用于保存各线路的通断和时延信息的结构体数组

输出：空

**作用：**在收到链路响应包后调用线路通断和往返时延函数，传入维护各线路通断和时延信息的结构体数组，根据收到的链路响应包信息对其状态和时延数据进行更新。

**线路状态和时延结构体定义**

typedef struct \_transline\_status{

bool state[NET\_ID\_MAX]; //判断通路状态

unsigned int delay[NET\_ID\_MAX]; //通路延时

bool seek\_pack\_state[NET\_ID\_MAX][SEEK\_MAX\_NUM]; //发送的链路探测帧状态

bool reply\_pack\_state[NET\_ID\_MAX][SEEK\_MAX\_NUM]; //对应的链路响应包状态

unsigned int seek\_pack\_time[NET\_ID\_MAX][SEEK\_MAX\_NUM]; //记录每个序号的链路探测帧发送时间

unsigned int reply\_pack\_time[NET\_ID\_MAX][SEEK\_MAX\_NUM]; //记录收到每个序号的链路响应包时间

}TRAN\_STATUS;