1. **发送端相关函数**

**1.1封装函数**



**1.1.1添加UDP头函数**

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

**1.1.2添加IP头函数**

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

**1.1.3添加MAC帧函数**

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

**封装函数思路**：传入要添加对应头部的mbuf和已经填充好对应头部的socket结构体，其中mbuf在其Headroom部分已经预留了足够添加UDP+IP+MAC头部的空间，调用DPDK中对mbuf进行操作的API函数：rte\_pktmbuf\_prepend（m,sizeof(n)）将m中的pkt.data指针前移sizeof(n)的空间，将头部结构体插入。

**1.2构造帧函数**



**1.2.1构造业务数据包函数**

**Int business\_data\_packaing(struct rte\_mbuf \*m,DATA\_PACK 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);**

**1.2.3构造ACK包函数**

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

**1.2.4构造复位包函数**

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

**1.2.5构造链路探测包函数**

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

**1.2.6构造链路响应包函数**

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

**1.2.7构造ARP请求帧函数**

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

**1.2.8构造ARP响应帧函数**

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

**构造帧函数思路：**传入要添加对应数据结构体的mbuf和已经构造好的数据结构体，其中mbuf在其data空间已经预留了足够大的空间。调用复制函数

memcpy(m->pkt.data,&data\_pack,sizeof(data\_pack))将data\_pack里的sizeof（data\_pack）字节从data指针开始复制。

**1.3发包函数**

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

**发包函数思路：**传入要发送的mbuf和发送包的端口号，通过调用DPDK里的API函数rte\_eth\_tx\_buffer(port, 0, buffer, m)每次将一个mbuf发送到port端口对应的TX队列中。

**1.4ARP查询函数**

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

**ARP查询函数思路：**传入包含目的IP地址信息的socket结构体，查询静态存储的ARP缓存表找到对应IP地址的MAC信息。然后通过MAC地址和端口号的对应信息返回要传输的端口号。

**1.5网络策略选择函数**

**int transport\_selection(void);**

**网络策略选择函数思路：**先用for循环遍历传输线路状态结构体里的state[NETMAX]数组先判断该传输线路状态是否为通路，然后在用一个mindelay的double型变量记录所有为通路的传输线路中的最短时延。返回这个传输线路的ID。

1. **接收端相关函数**

**2.1解封装函数**



**2.1.1除去MAC帧函数**

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

**2.1.2除去IP头函数**

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

**2.1.3除去UDP头函数**

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

**解封装函数思路：**传入要保存对应MAC，IP，UDP头部信息的socket结构体和要解封装的mbuf，用指针的强制类型转换例如：sock->machdr=(PMAC\_HEADER)m->pkt.data将mbuf中pkt.data指向的MAC头部信息保存到socket结构体里的machdr结构体指针中。然后调用DPDK中的API函数rte\_pktmbuf\_adj(m,sizeof(MAC\_HEADER))将数据指针后移sizeof（MAC\_HEADER）位，这样就实现了解封装操作。

**2.2收包函数**

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

**收包函数思路：**传入用于接收mbuf的mbuf数组，调用DPDK中的API函数

rte\_eth\_rx\_burst(portid,0,m,MAXSIZE)一次从portid的端口的RX队列中接收MAXSIZE个mbuf存放到m的mbuf数组中。

**2.3帧分类函数**

**unsigned char pack\_classfied(struct rte\_mbuf \*m);**

**帧分类函数思路：**传入用于分类的mbuf，此时mbuf已经经过了MAC、IP、UDP头部的解封装，然后创建一个unsigned char类型的Type指针。用强制类型转换获取mbuf中的pkt.data指向的TYPE信息。然后调用rte\_pktmbuf\_adj(m,sizeof(unsigned char))将pkt.data指针后移sizeof（unsigned char）个单位。函数返回Type指针所获得的数据。

**2.4接收数据帧处理函数**

**2.4.1接收业务数据包处理函数**

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

**业务数据包处理函数思路：**此时传入的mbuf已经移除了Type部分。分别创建short型指针变量pack\_num和char型指针变量frag\_num。pack\_num用于保存业务数据包的序号信息，frag\_num用于保存业务数据包的分片信息。随后将业务数据包放入对应序号的接收端接收窗口中，同时用累计确认的方法构造ACK确认包。如果业务数据包存在分片信息需将其同时放到分片重组窗口中。然后将ACK确认包封装UDP、IP、MAC头部信息，通过策略选择函数选择最优传输线路将其通过对于端口发送出去。

**2.4.2接收ACK包处理函数**

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

**接收ACK包处理函数思路：**创建一个short型指针变量ack\_num用于保存ACK包中确认信息，如果ack\_num为null或不位于发送窗口的序号中则判断为错，然后通过ack\_num信息对发送窗口进行调整，用一个count变量记录ack\_num信息确认的收到包的数量，然后将发送窗口向右移count变量，这样实现对ACK包的处理。

**2.4.3接收业务数据ACK包处理函数**

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

**接收业务数据ACK包处理函数思路：**函数中只需要调用业务数据包处理函数：business\_pack\_process（）和ACK包处理函数：ack\_pack\_process()即可完成对业务数据和ACK包的处理。

**2.4.4接收复位包处理函数**

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

**2.5接收链路控制帧处理函数**

**2.5.1接收链路探测包处理函数**

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

**接收链路探测包处理函数思路：**分别创建unsigned int型指针变量Length和unsigned short型的指针变量num获取链路探测包中的长度和序号信息，然后调用rte\_pktmbuf\_adj（）函数将data指针后移。获取序号信息后创建链路响应包，封装UDP、IP、MAC头部后通过传入的portid的端口将其发送出去。

**2.5.2接收链路响应包处理函数**

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

**接收链路响应包处理函数思路：**分别创建unsigned int型指针变量Length和unsigned short型的指针变量num获取链路探测包中的长度和序号信息，然后调用rte\_pktmbuf\_adj（）函数将data指针后移。获取序号信息后将当前系统时间存储在链路状态结构体中的记录每个序号的链路响应包接收时间的数组：reply\_pack\_time[portid][\*num]中，然后调用函数new\_delay\_time（）计算出该条线路的平滑往返时延。同时判断该通路状态state[portid]是否为false，若为false则将其置为true。最后将链路状态结构体中记录该条线路链路响应包接收状态数组中该序号前的包状态全部置为true。

**2.5.3接收ARP请求包处理函数**

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

**接收ARP请求包处理函数思路：**创建一个ARP\_PACK类型的结构体指针保存ARP请求包中的ARP结构体信息，获取主机当前的IP地址，判断ARP请求包中的目的IP地址是否与自身IP地址相同，若相同的话则获取当前MAC地址构造ARP响应包封装MAC头部通过对于portid端口发送出去。同时更新自身的ARP缓存表。

**2.5.4接收ARP响应包处理函数**

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

**接收ARP响应包处理函数思路：**创建一个ARP\_PACK类型的结构体指针保存ARP响应包中的ARP结构体信息，根据其中的源MAC地址和源IP地址更新自身的ARP缓存。

**2.5.5更新链路时延函数**

**double new\_delay\_time(double old\_time,double new\_time,int portid);**

**更新链路时延函数思路：**old\_time为一个序号的链路探测包的发送时间，new\_time为该序号的链路响应包的接收时间。根据公式

计算出该portid链路的平滑往返时延，更新到对应的时延记录数组中去。