# 基于DPDK的协议栈设计

## 一、内容简介

传统的TCP/IP网络通信需要大量的拷贝，以及上下文切换，这严重的影响了网络通信效率。为此我们提出了一个DPDK来解决这些瓶颈。传统TCP/IP网络通信在大数据环境的瓶颈主要是CPU数据拷贝和网卡速度不匹配，导致数据丢包，同时还限制了网卡和CPU的性能。解决这些瓶颈，Intel提出了DPDK，该技术使用零拷贝技术、轮询机制、绕过内核等机制，使数据包在处理的时候绕过内核，减少上下文切换和数据包复制的开销，从而更大程度上面释放CPU和网卡的性能。

## 二、理论基础

### 2.1 知识点一 ICMP协议

2.1.1 ICMP协议原理之ping

ICMP时internet控制报文协议。它是TCP/IP协议簇的一个子协议，用于IP主机、路由器之间传递控制消息。控制消息指的时网络不可达、主机不可达等。这些控制信息虽然并不能传输用户数据，但是对于用户数据的传递起着重要的作用。简单来说就是侦测本机到目标主机的网络是否畅通。

ICMP提供一致易懂的出错报告信息。发送的出错报文返回到发送源数据的设备，因为只有发送设备才是出错报文的逻辑接受者。发送设备随着后可根据ICMP报文确定发生错误类型，并确定如何才能更好地重发失败的数据包。但是ICMP唯一的功能时报告问题而不是纠正错误，纠正错误的任务由发送方完成。

我们在网络中经常使用到ICMP协议，比如我们经常使用的用于检查网络通不通的ping命令(Linux和windows中均有)，这个Ping的过程实际上就是ICMP协议工作的过程。

2.1.2 ICMP ping报文格式

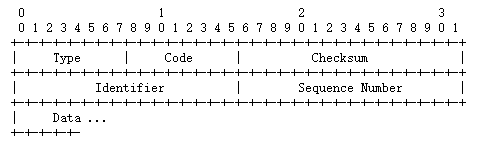


图 1 ICMP应答包结构

Type占据8位，用于标识发包的类型。

Code占据8位，标识ICMP分组类型的某一种具体分组。

Checksum校验和，用于检验数据包是否完整是否被修改。

Identifier标识符，标识本进程。

Sequence Number 序列号，标识本地到目的的数据包序号，一般从序号1开始。

2.1.3 ICMP实现—ping

Ping的发送和接受过程。

例子：

假定主机A的IP地址是192.168.1.1，主机B的IP地址是192.168.1.2，它们都在同一个子网。当我们主机A上运行ping 192.168.1.2后，ping命令执行的时候，源主机会构建一个ICMP请求数据包，ICMP数据包内包含多个字段。最重要的就是两个，第一个是类型字段，对于请求数据包而言字段为8；另外一个是顺序号，主要用于区分连续ping的时候发出的多个数据包。每发出一个请求数据包，顺序号自动加1.为了能够计算往返时间RTT，它会在报文的数据部分插入发送时间。

然后，由于ICMP协议将这个数据包连同地址192.168.1.2，一起交给IP层。IP层将以192.168.1.2作为目标地址，本机IP地址作为源地址，加上一些其他控制信息，构建一个IP数据包。

接下来，再MAC层，如果在本节ARP映射表中查找到IP地址192.168.1.2所对应的MAC地址，则可以直接使用；如果没有，则需要发送ARP协议查询MAC地址，获得MAC地址后，由数据链路层构建一个数据帧，目的地址时IP层传来的MAC地址，源地址则是本机的MAC地址；还要附加上一些控制信息，依据以太网的介质访问规则，将它们传送出去。

主机B收到这个数据帧后，先检查它的目的MAC地址，并和本机的MAC地址对比，如符合，则接受；否则就是丢弃。接收后检查该数据帧，将IP数据包从帧中提取出来，交给本机IP层，同样，IP层检查后，将有用的信息提取后交给ICMP协议。主机B会构建一个ICMP应答包，应答数据包的类型字段为0，顺序号为接收到的请求数据包中到的顺序号，然后再发送出去给主机A。再规定的时间内，源主机如果没有接收到ICMP的应答包，则说明目标主机不可达；如果接收到了ICMP应答包，则说明目标主机可达。并且此时，源主机会检查用当时时刻减去数据包最初从源主机上发出的时刻，就是ICMP数据包的时间延迟。

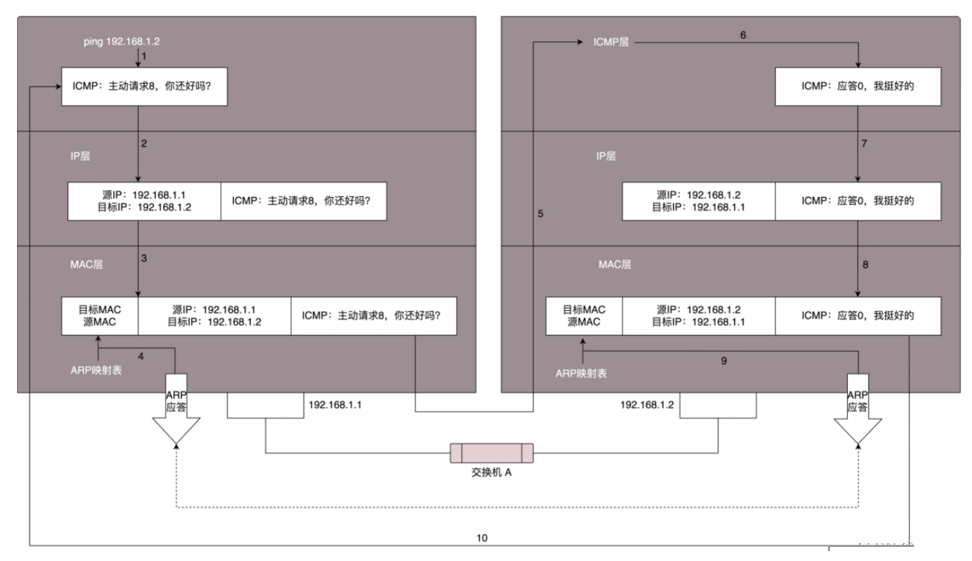


图 2 PING效果

### 2.2 知识点二 UDP协议

2.2.1 UDP协议原理

internet协议簇支持一个无连接的传输协议，称之为UDP。UDP为应用程序提供了一个无需建立连接就可以发送封装的IP数据包的方法。

使用UDP传输数据时，由应用程序根据需要提供报文到达确认、排序、流量控制等功能。通常情况下，UDP采用实时传输机制和时间戳来传输语言和视频数据。

2.2.2 UDP 特点

1、无连接：UDP是无连接的协议，它在进行数据传输之前不需要先建立连接，也没有各种重传机制、拥塞控制和流量控制，所以传输速度很快，消耗很低，延迟很小，数据传输效率高，适合对可靠性要求不高的应用程序，就或者可以保障可靠性的应用程序，如DNS，TFTP，SNMP等。

2、不可靠：只负责数据发送，不关心数据是否送达，没有确认机制，主机收到数据也不会有相应。

3、分组首部开销小：TCP的首部是20个字节，UDP首部是8个字节。

4、面向报文的：TCP（面向连接的传输控制协议）是面向字节的传输，而UDP是面向报文传输，对于应用层交下来的报文段不进行拆分合并，直接保留原有报文段的边界然后添加UDP的首部就交付给网络层。无论报文的长短，UDP都不会进行处理。因此为了避免报文段过短降低传输效率以及报文段过长导致网络层对IP数据进行分片操作，应用层应该选择合适长度的报文交付给传输层的UDP。

2.2.3 UDP数据包格式

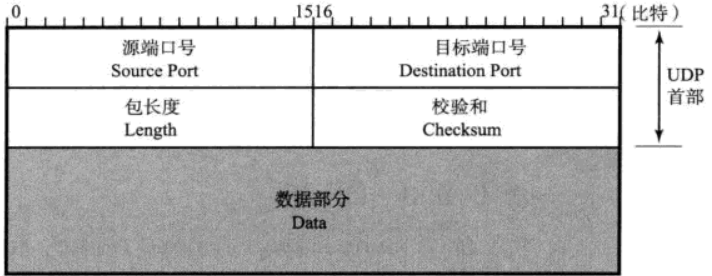


图 3 UDP数据包格式

1. 源端口号（Source Port）

表达发送端口端口号，字段长16位。该字段是可选项，有时可能不会设置源端口号。没有源端口号的时候该字段值设置为0。可用于不需要返回的通信中。

1. 目标端口号（Destination Port）

表示接收端端口，字段长度16位。

1. 包长度（Length）

该字段保持了UDP首部的长度根数据的长度之和。单位为字节

1. 校验和（Checksum）

检查整个UDP数据包是否被修改，如果正确这表明没有被修改；如果不正确则表示修改了，丢弃数据包。

2.2.4 UDP 通信方式

用一段小代码表示一下。

环境：服务器IP：192.168.1.67，端口：8080

发送方：

import socket

1、创建套接字

udpsocket=socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM);

2、准备接受方地址

udpadd=(“192.168.1.67”, 8080);

3、从键盘获取数据

udpData=input(“请输入发送消息：”);

4、发送

udpsocket.sendto(udpData.encode(‘utf-8’), udpadd);

5、关闭

udpsocket.close();

接收方：

import socket

1. 创建套接字

udpsocket=socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

2、接受

while True:

data\_info.ip\_info = udpsocket.recvform(1024);

print(data\_info.decode(‘gbk’))

print(ip\_info)

3、关闭套接字

udpsocket.close();

此时端口号是随机的，我们可以绑定端口比如像110一样有固定的端口

例如：

import socket

1. 创建套接字

udpsocket=socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM);

1. 绑定本地

udpsocket.bind((‘’, 110));

1. 接受

data\_info.ip\_info=udpsocket.recvform(1024);

print(data\_info.decode(‘gbk’));

print(ip\_info);

1. 关闭套接字

udpsocket.close();

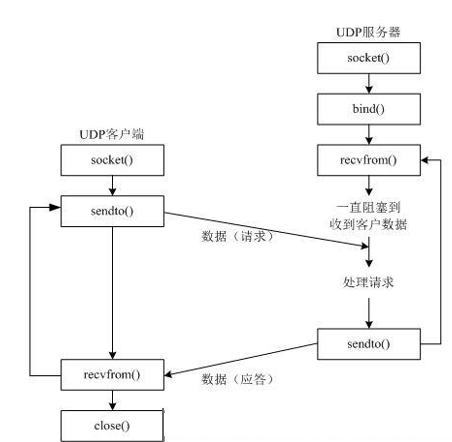


图 4 UDP工作方式

### 2.3 知识点三 ARP协议

2.3.1 ARP协议的原理。

Arp是一个地址解析协议，工作在以太网链路层。具体工作过程如下

主机A的IP地址为192.168.1.1，MAC地址为0A-11-22-33-44-01;

主机B的IP地址为192.168.1.2，MAC地址为0A-11-22-33-44-02;

当主机A要与主机B通信时，地址解析协议可以将主机B的IP地址（192.168.1.2）解析成B的MAC地址，以下为工作流程：

第1步：根据主机A上的路由表内容，IP确定用于访问主机B的转发IP地址是192.168.1.2.然后主机A在自己本地ARP缓存中检查主机B的匹配MAC地址。

第2步：如果主机A在ARP缓存中没有找到映射，它将询问192.168.1.2的硬件地址，从而将ARP请求帧广播到本地网络上的所有主机。源主机A的IP地址和MAC地址都包括在ARP请求中。本地网路上的每台主机都接收ARP请求并且检查是否与自己的IP地址匹配。如果主机发现请求的IP地址于自己的IP地址不匹配，它将丢弃ARP请求。

第3步：主机B确定ARP请求中IP地址与自己的IP地址匹配，则将主机A的IP地址和MAC地址映射添加到本地ARP缓存中。

第4步：主机B将包含其MAC地址的ARP回复消息直接发送回主机A。

第5步：当主机A收到从主机B发来的ARP回复消息时，会用主机B的IP和MAC地址映射更新ARP缓存。本机缓存是有生存周期的，生存周期结束后，将再次重复上面的过程。主机B的MAC地址一旦确定，主机A就能向主机B发送IP通信了。

ARP缓存表

ARP缓存表是一个用来存储IP地址和MAC地址的缓冲区，其本质就是一个IP地址-> MAC地址的对应表，表中每一个条目分别记录了网络上其他主机IP地址和对应的MAC地址。每一个以太网或者令牌环网络适配器都有自己单独的表。当地址解析协议被询问下一个已知IP地址节点的MAC地址时，现在ARP缓存中查看，若存在，就直接返回与之对应的MAC地址，若不存在，才发送ARP请求给局域网查询。

为使广播量最小，ARP维护地址到MAC地址映射的缓存以便将来使用。ARP缓存可以包含动态和静态项目。动态项目随时间推移自动添加和删除。每个动态ARP缓存项的潜在生命周期使10分钟。新添加到缓存中的项目带有时间戳，如果某个项目添加后2分钟内没有在使用，则此项目过去并从ARP表缓存中删除；如果一个项目已在使用，则又收到2分钟的生命周期；如果某个项目始终在使用，则回另外收到2分钟的生命周期，一直到10分钟的最长生命周期。静态项目一直保留在缓存中，直到计算机重新启动。

2.3.2 ARP数据包格式。



从左到右一路分析：ARP报文总长度为28字节，MAC地址长度为6字节，IP地址长度为4字节。

1. 硬件类型：指明了发送方想知道硬件接口类型，以太网值1；
2. 上层协议类型：指明了发送方提供的高层协议类型，IP为0800（16进制）
3. 硬件地址长度和协议长度：分别指出硬件地址和协议长度，以字节为单位。对于以太网IP地址的ARP请求或应答来说，它们的值分别为6和4。
4. 操作类型：用来表示这个报文的类型，ARP请求为1，ARP 响应2。
5. 发送方MAC地址：发送方设备的硬件地址。
6. 发送方IP地址：发放方设备的IP地址。
7. 目标MAC地址：接收方设备的硬件地址。
8. 目标IP地址：接收方设备的IP地址。

### 2.4 知识点四 DPDK

2.4.1 DPDK基本原理

网络设备（路由器、交换机、媒体网关、SBC、PS网关）需要再瞬间进行大量的报文转发，因此在传统的网络设备上，往往能够看到专门的NP（Network Process）处理器，有的用FPGA，有的用ASIC。这些专用器件通过内置的硬件电路（或通过编程形成的硬件电路）高效转发报文，只有需要对报文进行深度处理的时候才需要CPU干预。但在公有云、NFV等应用场景下，基础设施以CPU为运算核心，往往不具备专用的NP处理器，操作系统也以通用的Linux为主，网络数据包的收发处理器路径如下所示：

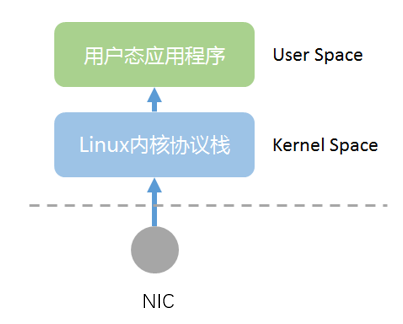


图 5 通用数据包收发路径

在虚拟化环境中，路径则会更长

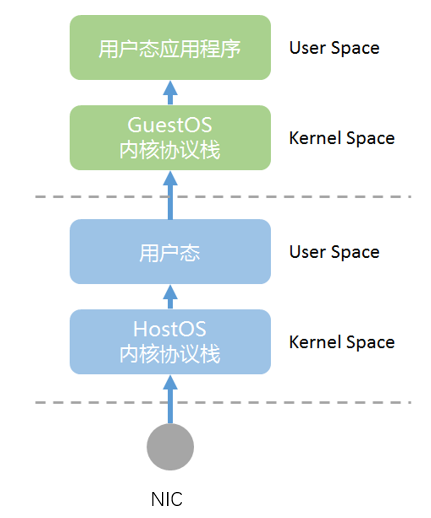


图 6 虚拟化环境数据路径

由于包处理任务存在内核态与用户态的切换，以及多次的内存拷贝，系统消耗变大，以CPU为核心的系统存在很大的处理瓶颈。为了提升在通用服务器（COTS）的数据包处理性能，Intel推出了服务于IA（Intel Architecture）系统的DPDK技术。DPDK是Data Plane Development Kit缩写。简单来说，DPDK应用程序运行在操作系统的User Space，利用自身提供的数据面库进行收发包处理，绕过了Linux内核态协议栈，以提升报文处理效率。

DPDK是一组lib库和工具包的集合。最简单的架构描述如下图所示：

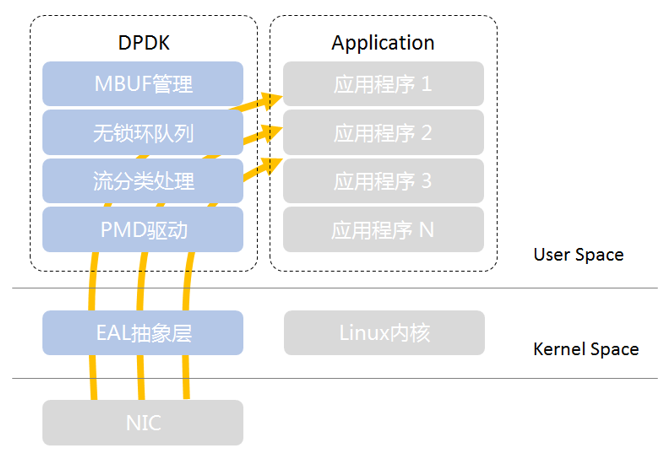


图 7 DPDK基本工作流程

上图蓝色部分是DPDK的主要组件，简单解释一下：

* PMD:Pool Mode Driver, 轮询模式驱动，通过非中断，以及数据帧进行应用缓冲区内存的零拷贝机制，提高发送/接受数据帧的效率。
* 流分类处理：Flow Classification， 为N组元组匹配和LPM（最长前缀匹配）提供优化的查找算法。
* 环队列：Ring Queue，针对内存创建缓冲区，并通过建立MBUF对象，封装实际数据帧，提供给应用程序使用。
* EAL: Environment Abstract Layer, 环境抽象（设配）层，PMD初始化、CPU内核和DPDK线程配置/绑定、设置HugePage大页内存等系统初始化。

这么说可能还是有一点点抽象，在总结一下DPDK的核心思想：

1. 用户态模式的PMD驱动，除去中断，避免内核态和用户态内存拷贝，减少性能开销，从而提升I/O吞吐量能力
2. 用户态有一个好处，一旦程序崩溃，不至于导致内核完蛋，带来了更高级的健壮性。
3. Huge Page，通过更大的内存页（如1G内存），减少TLB（Translation Lookaside Buffer， 即块表）Miss， Miss对报文转发性能影响很大。
4. 多核设备上创建多线程，每个线程绑定到独立的物理核，减少线程调度的开销。同时每个线程对于的独立免锁队列，同样为了降低系统开销。
5. 向量指令集，提升CPU流水线效率，降低内存等待开销。

下面简单描述了DPDK的多队列和多线程机制：

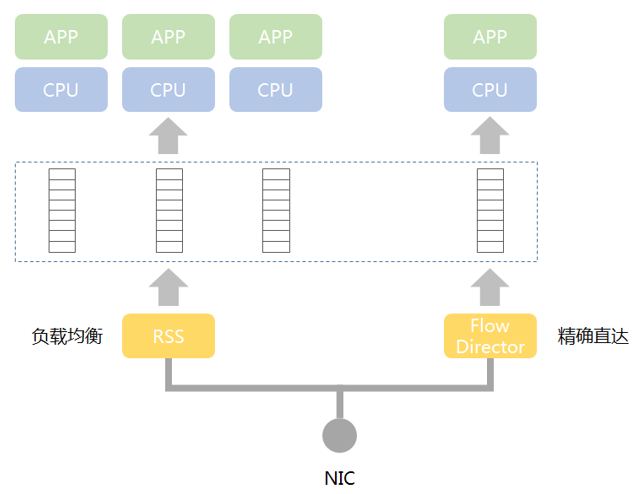


图 8 多队列机制

DPDK将网卡接受队列分配给某个CPU核，该队列收到的报文都交给该核上的DPDK线程处理。存在两种方式将数据包发送到接收队列之上：

* RSS（Receive Side Scaling，接收方扩展）机制：根据关键字，比如根据UDP的四元组<srcIP>、<dstIP>、<srcPort>、<dstPort>进行哈希
* Flow Director机制：可设定根据数据包某些信息进行精准匹配，分配到指定的队列和CPU核。

当网络数据包（帧）被网卡接收后，DPDK网卡驱动将其存储在一个高效的缓冲区中，并在MBUF缓存中创建MBUF对象与实际网络数据包相连，对网络数据包分析和处理都会基于该MBUF，必要的时候才会访问缓冲区中的实际网络包。

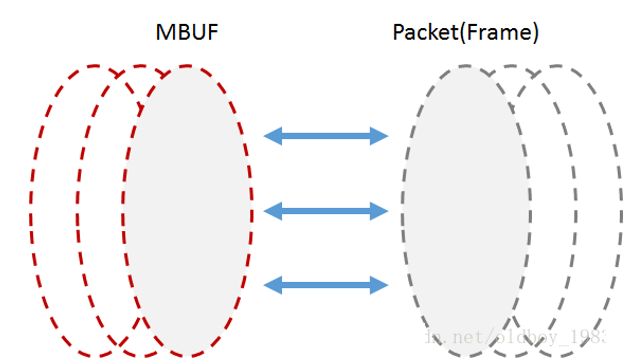


图 9 MBUF和Packet对应关系

### 2.5 知识点五 基于DPDK的icmp实现

2.5.1 ICMP之Ping 概述

ICMP（Internet Control Message Protocol）协议是因特网控制报文协议，ICMP常被认为网络层协议，它的报文存在于IP数据报的部分，如图。



图 10 协议层之ICMP

ICMP是基于IP数据报的，所以跟TCP不同的是，它是不需要指定端口的，更没有建立连接一说。而且，通常来说ICMP协议都是内核帮你实现的，系统自身就支持了，并不像TCP/HTTP等还要自己开几个服务监听对应端口啥的。可能有人会有疑问，既然没有端口来标识了，那我有时候开多个ping进程，这些响应消息是怎么对应到不同的ping进程的？这个就是ICMP报文里面标识符的作用了。标识符会在响应中带回来，这样发送方就能根据标识符将请求和应答匹配了。在ping中，这个标识符就是进程ID。

ICMP报文有多种类型，如地址掩码请求和应答、时间戳请求和应答、请求回显和回显应答等。ICMP报文通用格式如下，不同类型的报文内容有所不同。ICMP协议在ping中，下面分析一下。

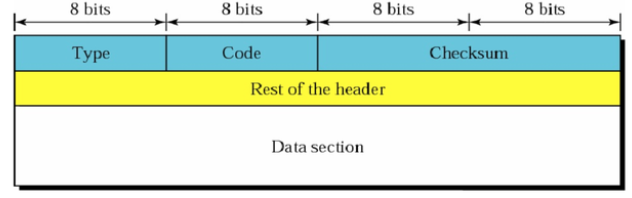


图 11 ICMP通用格式

PING使用的是ICMP的请求回显/回显应答类型的报文，格式如下。 它的内容包括标识符、序列号以及回显数据3部分，报文大小默认为64字节（header的8个字节+body漏洞56字节）。

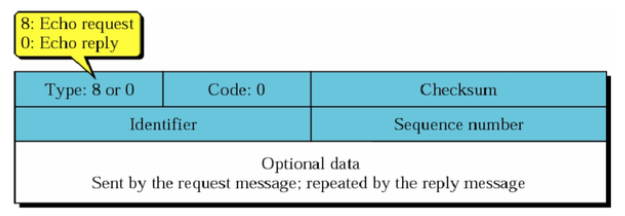


图 12 ping的数据包结构

ICMP回显报文格式

请求回显类型是8，回显应答类型是0，校验和是报内容根据算法生成用于校验数据完整性。标识符在Linux/MacOS中使用进程ID。序列号在Linux/MacOS中是从0递增的，每个进程独立的。回显数据包括发送ping请求的时间戳（在macOS占8个字节，在Linux占16字节），以及一串填充数据，在Linux这串数字默认值是0x10…37，共40字节。在macOS中是0x0809…37,共48字节。填充数据你也可以通过-p pattern指定，比如ping -pff 192.168.33.10，则填充数据全部是ff。默认TTL是64，你可以通过-t ttl指定TTL值。Ping请求时间=接收到回显应答的时间-应答回显数据中的时间。

2.5.2 ICMP之Ping 具体代码实现

我们的代码实现分为三个部分，这是这是叙述核心的代码

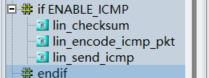


图 13 icmp实现的层次结构

为什么要计算一个checksum，原因是dpdk没有提供icmp校验和函数，所以必须我们自己写，查阅资料，我们发现Window、Linux或者MacOS都是参考设计规范文档里面的算法来实现的,连接[RFC 1071: Computing the Internet checksum (rfc-editor.org)](https://www.rfc-editor.org/rfc/rfc1071.html)

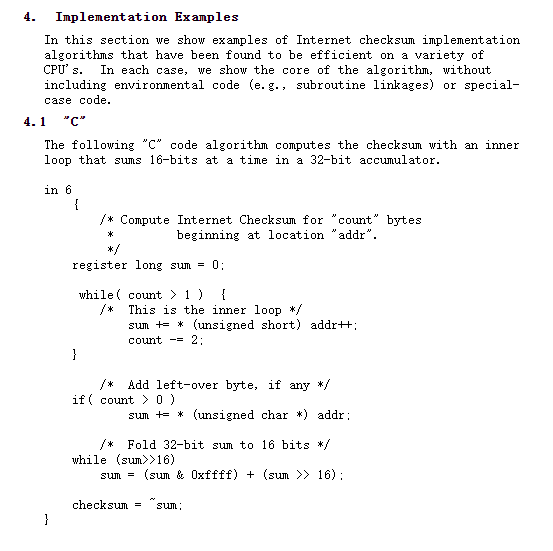


图 14 实现的例子

通过上面的算法，实现了下面这个checksum函数，参数addr是icmp的头部地址，count是icmp的长度。

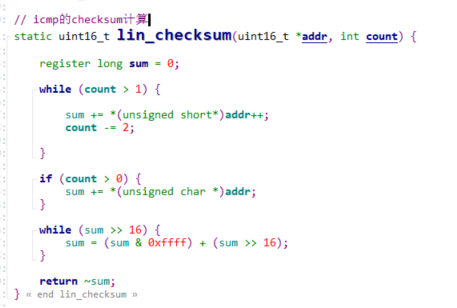


图 15 icmp的checksum的实现

Icmp的包封装过程，参数msg，dst\_mac, sip, dip, id, seqnb,这些参数的意思分别是

1. msg：数据包的基地址。msg是从mbuf\_pool里面申请的一块内存地址，然后把这个内存地址作为基地址，传递到函数里面，进行计算。
2. dst\_mac：目标的mac地址。这个值是，从对方发送过来的数据包里面提出来的源mac地址，主要是我要构造包发送回去。
3. sip：自己的IP地址。
4. dip：对方的IP地址。这个IP值，从对方的发送过来的数据包里面提出来的源ip地址。
5. id：这个是标识符，用于标识不同的进程。正如上面介绍的那样，我们没有端口，所以只能通过不同的标识符来对应不同的进程，阅读相关文档，这个id，其实就是Pid的值。这个是也是从发送过来的数据包里面获取的，主要是返回的数据包里面的id必须和之前的发送过来的数据包里面的id要一致。
6. seqnb: 这个是序列号。简单来说就是包的序号，和tcp里面包序号是一样的。这个也是从发送过来的数据包里面获取，主要是返回数据包里面id必须和之前发送过来的数据包里面sqenb一样，目的用于确认这个数据包对方已经收到。

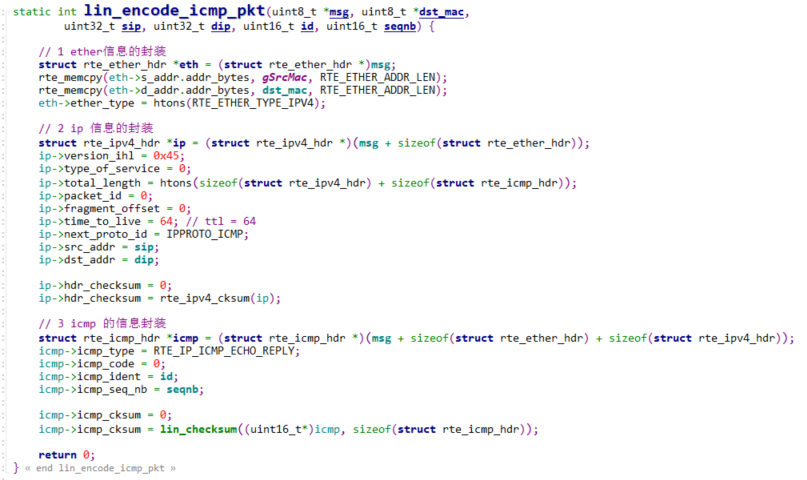


图 16 封装icmp包过程

ICMP 组合封装好的包

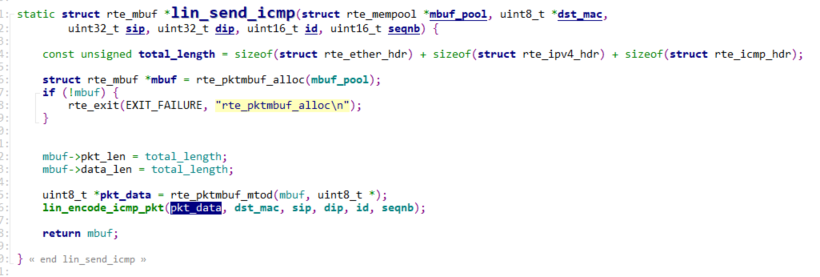


图 17 返回icmp包到主函数

参数：mbuf\_pool, dst\_mac, sip, dip, id, seqnb

这些参数和上面的叙述是一样的，只是mbuf\_pool不一样，mbuf\_pool是一个mbuf内存池，用于分配不同的mbuf来组合数据包。然后发送给lin\_encode\_icmp\_pkt打包icmp的数据包。

### 2.6 知识点六 基于DPDK的udp实现

2.6.1 udp 概述

UDP是User Datagram Protocol的简称，中文名字用户数据报协议，是IOS（Open System Interconnection，开放式系统互联）参考模型中一种无连接的传输层协议，提供面向事务简单不可靠信息传递服务，UDP在IP报文协议是17。于TCP（传输控制协议）一样，UDP协议直接位于IP（网络协议）协议的顶层。根据TCP/IP参考模型，UDP和TCP都属于传输层协议。UDP协议的主要工作是将数据压缩成数据包的形式。一个典型的数据包就是一个二进制的传输单位。每一个数据包的前8个字节用来包含报头信息，剩余字节则用来包含具体的传输数据。

UDP报文格式如下：

源端口（2字节）+目的端口（2字节）+长度（2字节）+校验和（2字节）+数据，用图表的形式展现出来，如下如所示：



图 18 udp数据包格式

2.6.2 udp 代码实现

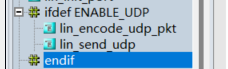


图 19 UDP代码布局

具体实现

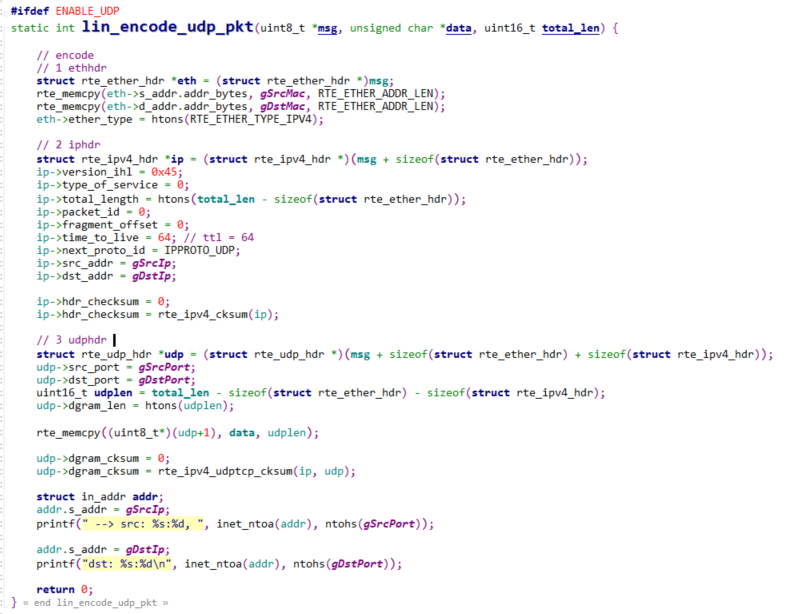


图 20 udp报构造过程

函数参数分析：msg， data， total\_len

msg：数据包的基地址。msg是从mbuf\_pool里面申请的一块内存地址，然后把这个内存地址作为基地址，传递到函数里面，进行计算。

data：要重复发送回去的数据信息。Data里面转载的数据是，对方发过来的数据信息，我们把数据拷贝到data里面，然后通过data填装到发送udp里面负载中，然后发包发送出去。

Length：表示的是整个数据包的长度。由上面的UDP包结构可以看到，我们在组装UDP包的时候，是需要填充length长度的，我们使用是uint16\_t udplen = total\_len - sizeof(struct rte\_ether\_hdr) - sizeof(struct rte\_ipv4\_hdr)来计算出，UDP包的length长度，使用全部长度减去ether头和ip头。

整个UDP组装逻辑是，先从数据链路层开始组装，填写双方的mac地址和上层类型，通知上层使用什么类型解析，我们使用的是IP协议，所以填写的是，RTE\_ETHER\_TYPE\_IPV4;之后就是IP层填写，首先是将数据包的首地址移动到ip层的开始，也就是struct rte\_ipv4\_hdr \*ip = (struct rte\_ipv4\_hdr \*)(msg + sizeof(struct rte\_ether\_hdr));，将数据包的地址加上数据链路层的size，然后移动到IP层的开始处，填写IP层的信息；之后同样的过程，移动到UDP头部开始处，填写的UDP数据包，从而形成一个完整的数据包格式。



图 21 生成UDP函数

参数：mbuf\_pool, data, length

Mbuf\_pool:表示的是mbuf池，所有的数据包内存申请都是从这个mbuf\_pool里面申请的；

Data：表示的是接收到的数据信息；

Length：表示的是接收到UDP的长度信息。

过程：

首先计算出整个数据包的长度。

然后去mbuf\_pool里面申请一个mbuf，用于存储组合的UDP数据包。

然后包里面的数据长度和包长度赋值，赋长度。

然后发给lin\_encode\_udp\_pkt函数打包生成UDP数据包，返回回来。

### 2.7 知识点七 基于DPDK的arp实现

2.7.1 arp概述

ARP协议的全称是Address Resoluction Protocol（地址解析协议），它是一个通过用于实现从IP地址到MAC地址映射，即询问目标IP对应的MAC地址的一种协议。ARP在IPv4中极其重要。（注意：ARP只用于IPv4中，IPv6协议使用的是Neighbor Discovery Prototocl, 译为邻居发现协议，它被纳入ICMPv6中）。

简而言之，ARP就是一种解决地址解析协议，它以IP地址为线索，定位下一个应该接收数据分包的主机MAC地址。如果目标主机不在同一个了链路上，那么会查找下一条路由器的mac地址。

Arp的工作机制，假设A和B位于同一链路，不需要经过路由器转换，主机A向主机B发送一个IP分组，主机A的地址192.168.1.2，主机B的地址是192.168.1.3它们都不知道对方mac地址是啥，主机C和主机D是同一链路的其他主机。

主机A想要获取主机B的MAC地址，通过主机A会通过广播的方式向以太网上的所有主机发送一个ARP请求包，这个ARP请求包包含了主机A想要知道主机B的IP地址MAC地址。主机A发送的ARP请求包会被同一个链路上的所有主机/路由器接收并进行解析。每个主机/路由器都会检查ARP请求包中的信息，如果ARP请求包中目标IP地址和自己的相同，就会将自己的主机MAC地址写入相应包返回主机A。由此，可以通过ARP从IP地址获取MAC地址，实现链路内通信。

我将获取到的mac写入到arp表里，下次发送数据的时候，直接从ARP表里面查找这个IP所对应的mac地址。

Arp数据包格式：



图 22 arp数据包格式

2.7.2 基于dpdk的arp实现

我使用两个文件来书写，用了一个头文件，arp.h

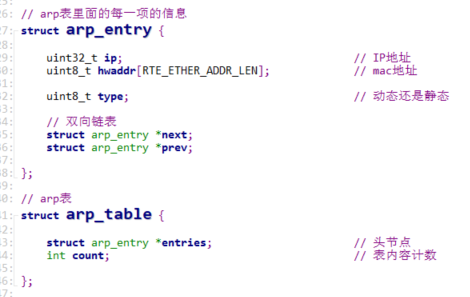


图 23 arp表结构

创建的ARP表是一个全局唯一的，所有我们采用单例模式编写的代码

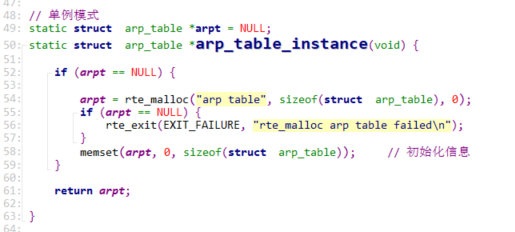


图 24 单例模式arp表

Ip转变成mac地址，通过查询arp表

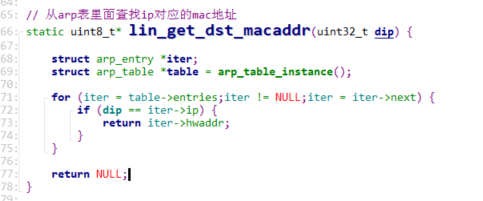


图 25 ip映射mac地址

对arp表增加和删除，写了两个宏，通用

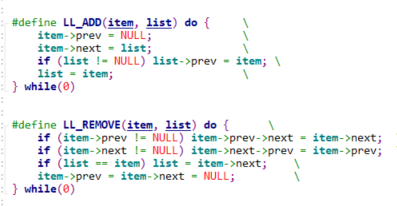


图 26 增加和删除

arp代码结构

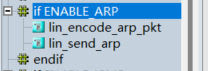


图 27代码结构

同样，lin\_encode\_arp\_pkt函数也是生成arp包，代码如下

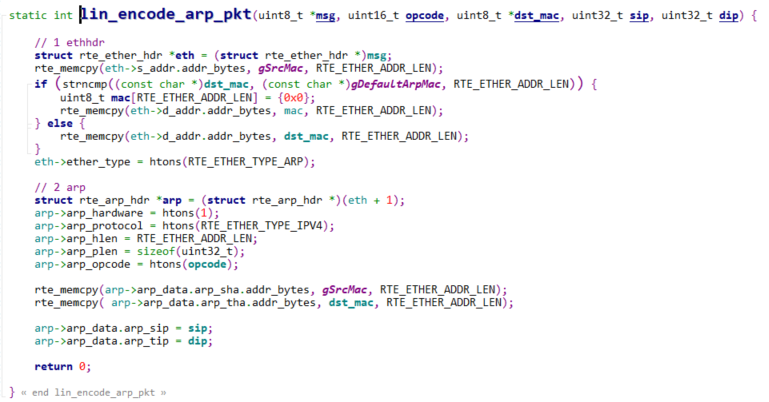


图 28 arp数据包生成代码

参数：msg， opcode，dst\_mac, sip, dip

msg：数据包的基地址。msg是从mbuf\_pool里面申请的一块内存地址，然后把这个内存地址作为基地址，传递到函数里面，进行计算。

opcode:是操作码，用于区别是请求还是回复，同时也需要往数据包里面写，用于回应。

dst\_mac:是目标的mac地址。用于返回数据包填写。

sip:表示生成arp数据包的时候，填写的源IP信息。

dip:表示生成arp数据包的时候，填写的目的IP信息。

过程：

和上述的icmp和udp的过程类似，区别在于存在两个构造以太网层的方式，一个采用广播的方式，对外发送arp请求；另一个方式，正常的回应arp数据包。

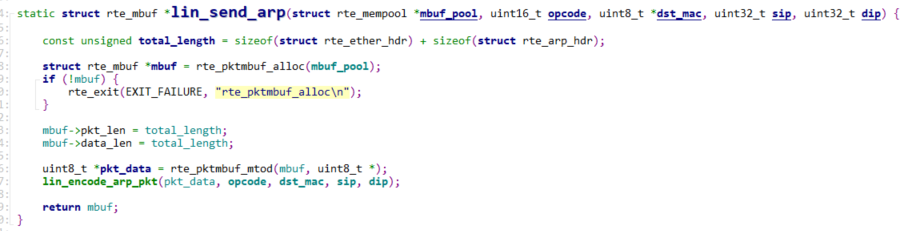


图 29 返回arp包

Mbuf\_pool：内存池，用于分配mbuf。

opcode:是操作码，用于区别是请求还是回复，同时也需要往数据包里面写，用于回应。

dst\_mac:是目标的mac地址。用于返回数据包填写。

sip:表示生成arp数据包的时候，填写的源IP信息。

dip:表示生成arp数据包的时候，填写的目的IP信息。

上面我们只是考虑了arp的工作过程，没有考虑我们定期发送arp请求维护arp表。所以我们写了一个定时器，代码如下：

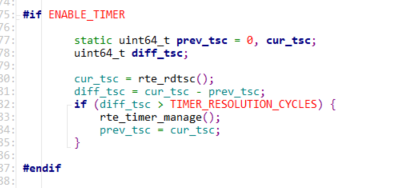


图 30 定时器

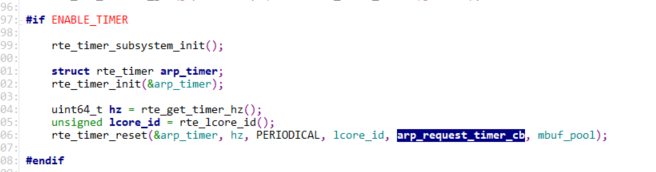


图 31 时间子系统

其中rte\_timer\_manager函数被调用，会执行rte\_timer\_reset()函数，回调arp\_request\_timer\_cb函数，从而定时主动的发送arp请求的数据包。

arp\_request\_timer\_cb函数:

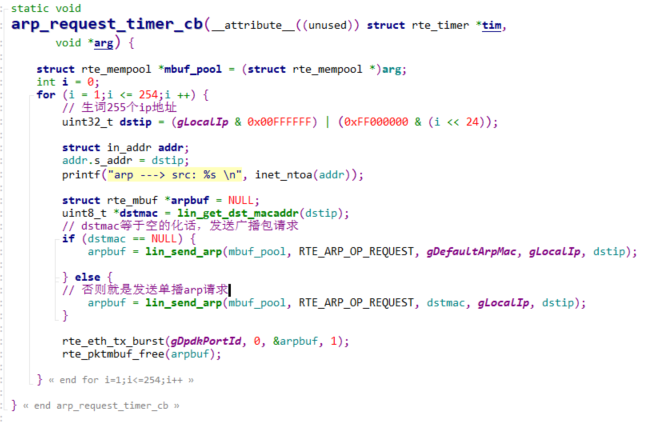


图 32 arp\_request\_timer\_cb函数

该函数里面的for循环就是不断的生成新的ip地址，然后去arp表里面查询，如果没有对应的mac地址，则采用广播的形式发送出去；如果有返回结果，那就使用单播的方式发送一个arp请求，目的就是为了检测对方是否还存活。

## 三、演示验证

### 3.1 运行环境搭建

### 3.2 实验任务一 DPDK安装并且运行helloworld例子

步骤1 去dpdk官网https://www.dpdk.org/或者GitHub上面下载dpdk源代码<https://github.com/DPDK/dpdk（我自己使用的是19>版本）。

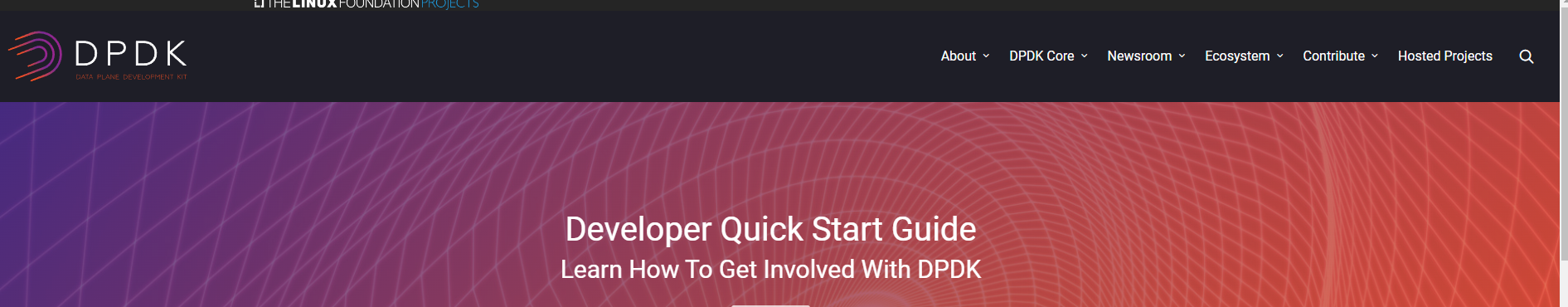


图 33 dpdk官网

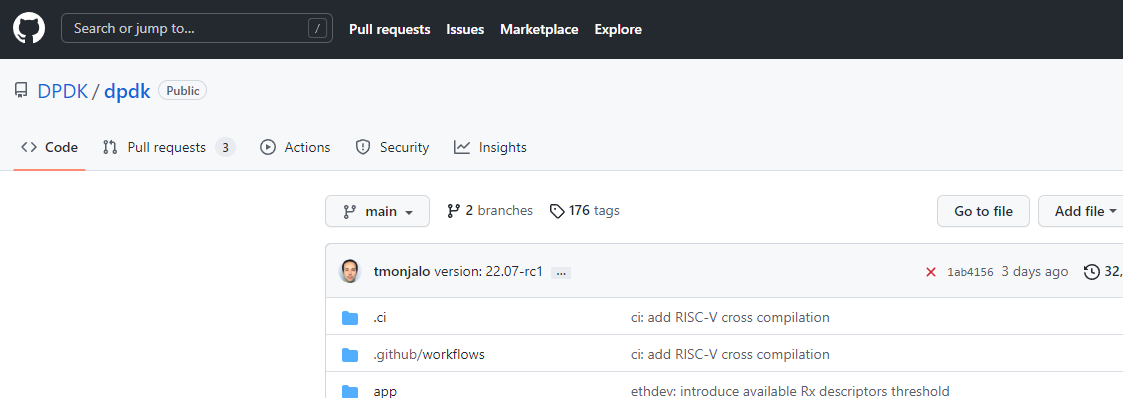
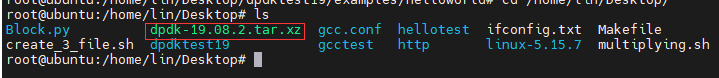


图 34 GitHub上的dpdk

步骤2安装dpdk（步骤标题）



dpdk-19.08.2.tar.xz是源代码文件，由于我们的Ubuntu一开始没有安装过相关的库，所有我们要输入下面两个命令

sudo apt-get install libnuma-dev

sudo apt-get install numactl

步骤1.3 编译dpdk代码，并且运行helloworld代码

首先，我们配置声明

export RTE\_SDK=/home/lin/Desktop/dpdktest19

上面这个代码的意思是，当前这个dpdk代码所处的位置。可以使用pwd来查看完整的路径。

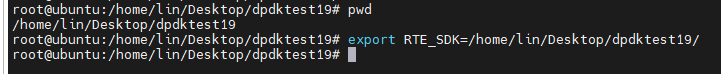


图 35 RTE\_SDK环境变量的设置

接着接入dpdk-setup文件，配置dpkd所需的信息



图 36 进入dpdk的配置阶段

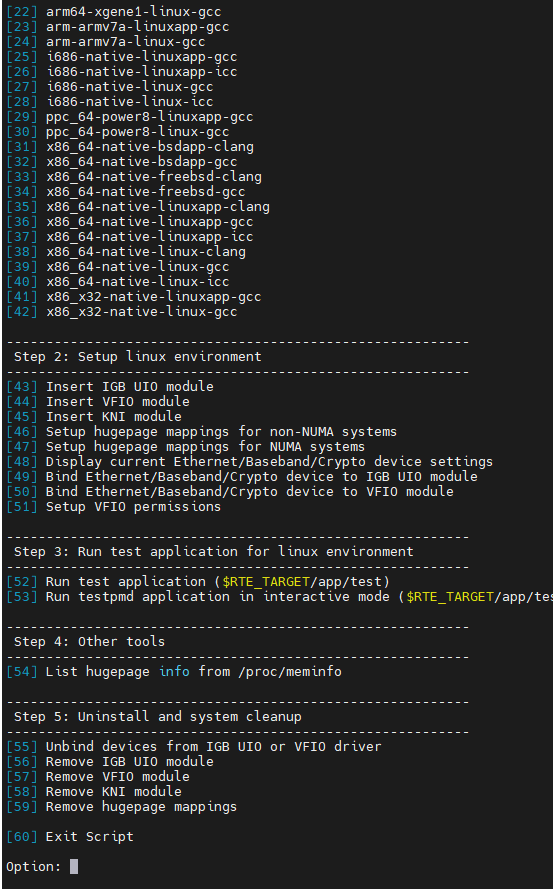


图 37 dpdk-setup

首先选择39对应的是x86\_64-native-linux-gcc，编译完成之后，dpdk代码目录里面会出现这个文件夹，这个文件是我们的运行环境，等会后面运行代码的时候需要这个。

之后我们选择43，44，45，逐个输入。这些是插入对应的驱动模块，以便网卡被用户态接管。

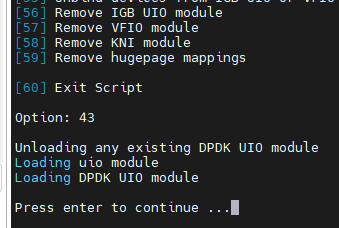


图 38 插入IGB模块

接下来设置non-numa，输入46，并且输入128；设置numa，输入47，并且输入128；这表示是巨页映射的数量。

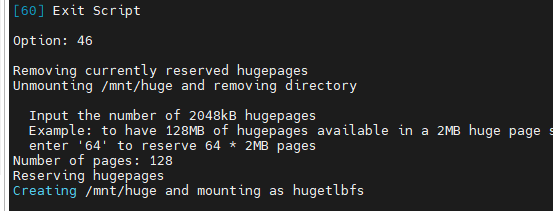


图 39 设置non-numa

最后配置设置网卡，我们先查看当前网卡信息，输入48，这个是查看当前有多少网卡。

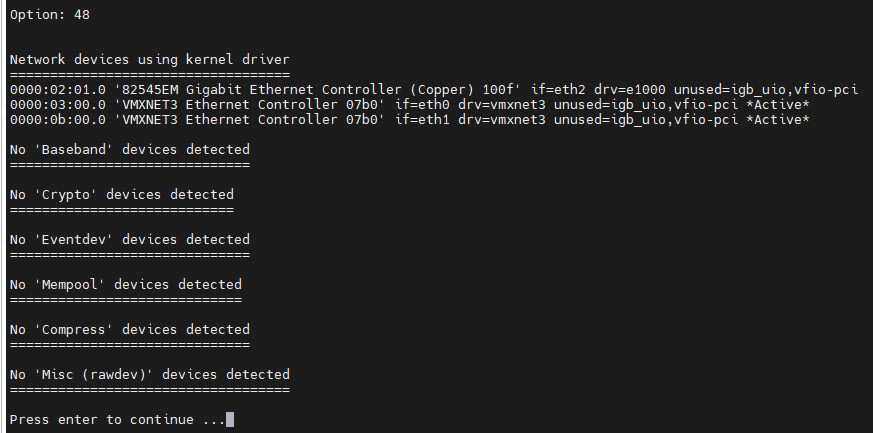


图 40 网卡状态

本实验使用eth0网卡作为dpdk实验的网卡。我们输入60推出去，然后使用sudo ifconfig eth0 down，在输入这个命令之前一定要把这个网卡之前运行的ip地址记录下来，之后再编写代码的时候要使用。在此运行dpdk-setup.sh，输入49，

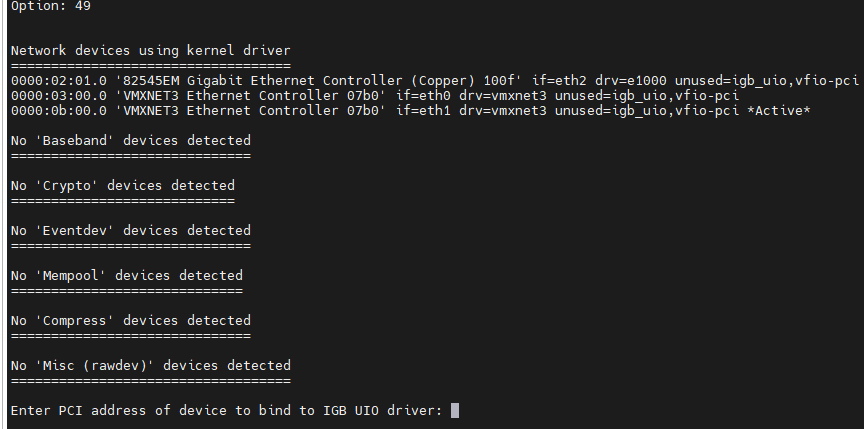


图 41 网卡状态

由上图可知，eth0由之前的active变成了down，接下面我们输入0000:03:00:0，然后就绑定成功了。接着就是退出。

输入：cd /home/lin/Desktop/dpdktest19/examples/helloworld/命令，进入helloworld例子里面，再输入make，然后输入./build/helloworld

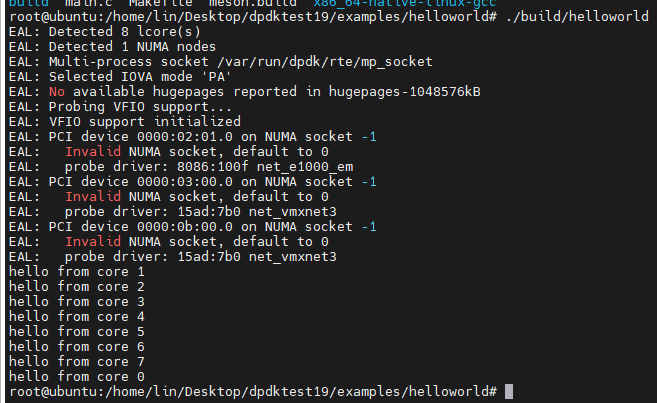


图 42 helloworld例子

### 3.3 实验任务二 UDP协议设计

步骤1 C语言编码

rte\_eal\_init使初始化dpdk环境的函数。rte\_pktmbuf\_pool\_create函数创建一个内存池。

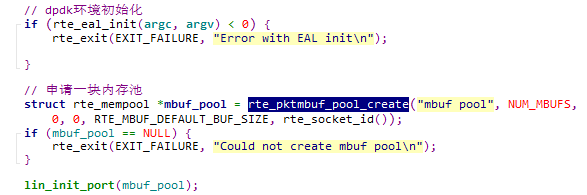


图 43 dpdk环境初始化

lin\_init\_port函数就是初始化网卡端口



图 44对网卡初始化

接着进入主循环，目的使处理所来的数据包。

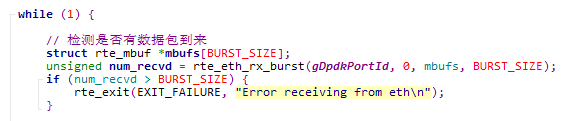


图 45 检查是否有数据过来

UDP数据包处理的过程。

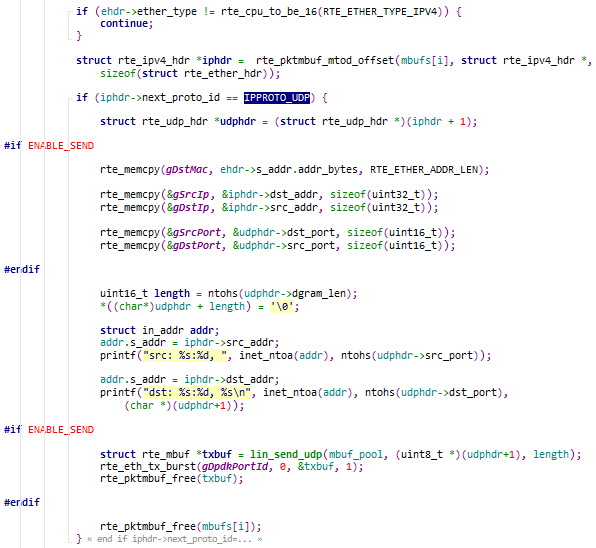


图 46 udp发包的具体流程

其中lin\_send\_udp函数组装udp数据包



图 47 组装数据包

其中lin\_encode\_pkt函数使更为具体的组装过程，涉及到每一层结构的组装。

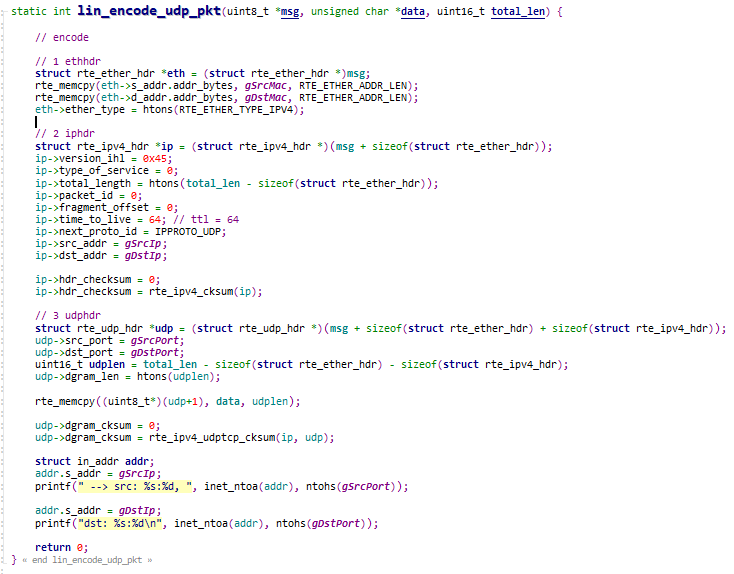


图 48 UDP包生成

步骤2 启动ubuntu+dpdk

上面以及讲述了dpdk环境的启动。

我们进入代码目录，我使放在dpdk的例子目录里面。

/home/lin/Desktop/dpdktest19/examples/lin\_protocol



图 49 代码所在地

进入这个文件夹，然后直接make就行，然后会生成一个build文件，里面有一个lin\_protocol文件，这个就是我们生成的可执行文件。



图 50 lin\_protocol可执行文件

步骤3 打开网络调试助手，并且发送数据包测试



图 51 配置ip地址以及端口信息

实验结果如下，使用网络调试工具发送消息给我们的udp协议，udp协议组装数据包，并且返回给了网络调试工具。



### 3.4 实验任务三 ICMP协议

步骤1 C语言编码ICMP协议

rte\_eal\_init使初始化dpdk环境的函数。rte\_pktmbuf\_pool\_create函数创建一个内存池。

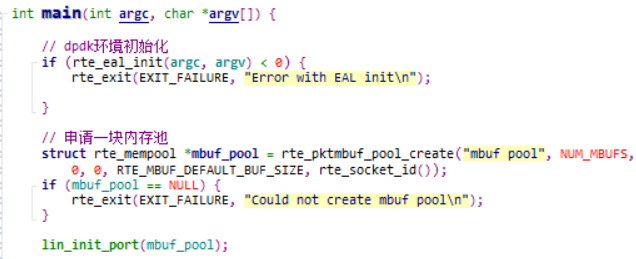


图 52 初始化dpdk环境

lin\_inti\_port(mbuf\_pool)函数用于初始化网卡端口信息。



主循环里面的icmp判断代码如下：



图 53 接受是ICMP数据包，并且处理返回

lin\_send\_icmp函数是用来生成要返回icmp相应的数据包。

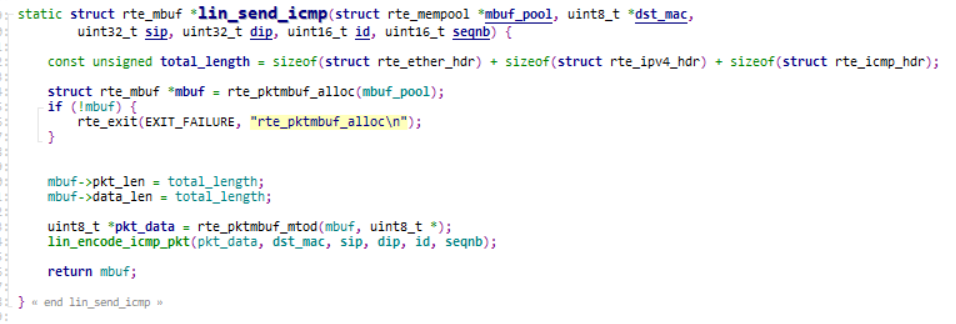


图 54 lin\_send\_icmp代码

生成icmp数据包的代码如下

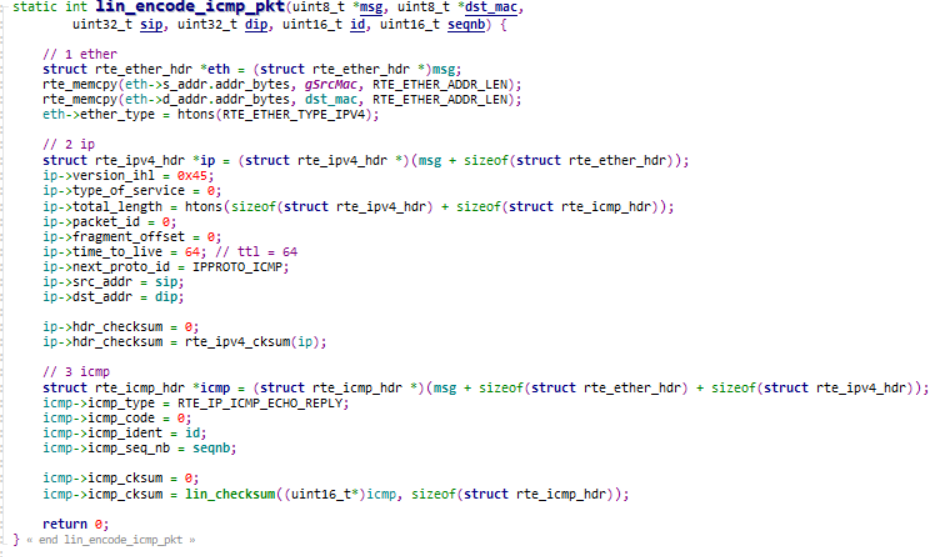


图 55 icmp组装过程

由于dpdk没有提供icmp的checksum函数，所以必须自己写一个，这个是参考RFC 1071-Computing the Internet checksum里面规范

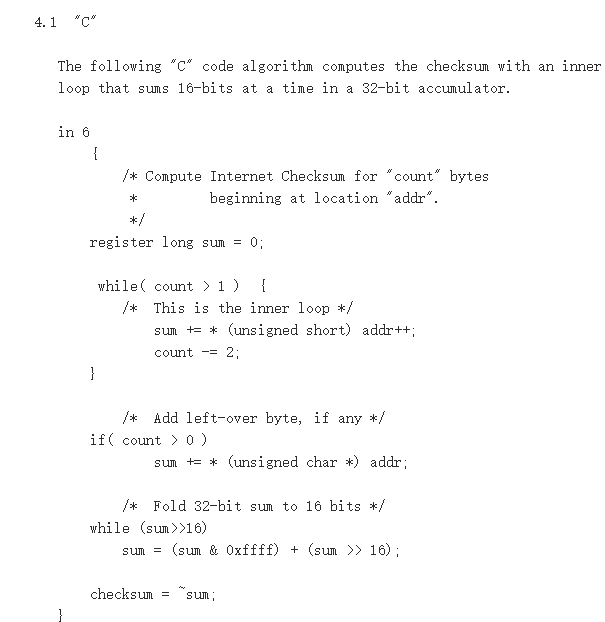


图 56 RFC 1071 规范

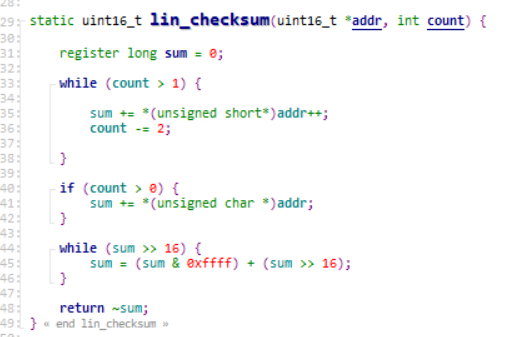


图 57 RFC 1071checksum的实现

步骤2 测试icmp

环境和上面的udp一样，运行lin\_protocol文件。然后再window系统打开cmd，然后ping dpdk绑定的网卡

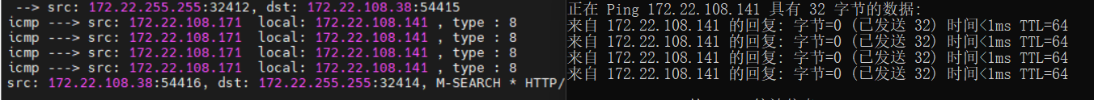


图 58 ping测试成功

### 3.5 实验任务三 ARP协议

步骤1 C语言编程ARP协议

初始化过程和上面的一样，主循环里面写了接受arp数据包的过程。



图 59 接受arp包

lin\_send\_arp和lin\_encode\_arp\_pkt两个函数用于构造arp相应包

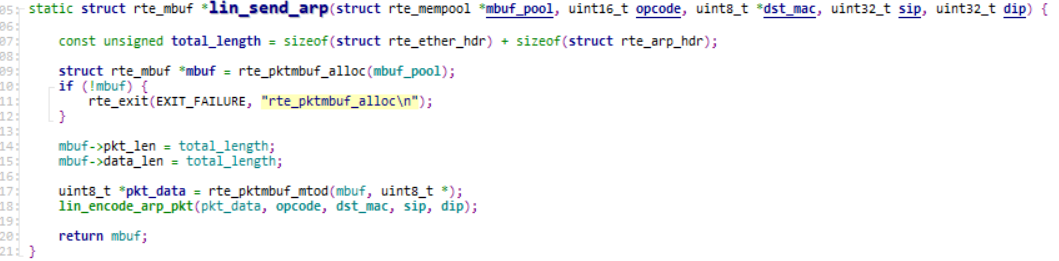


图 60 lin\_send\_arp代码

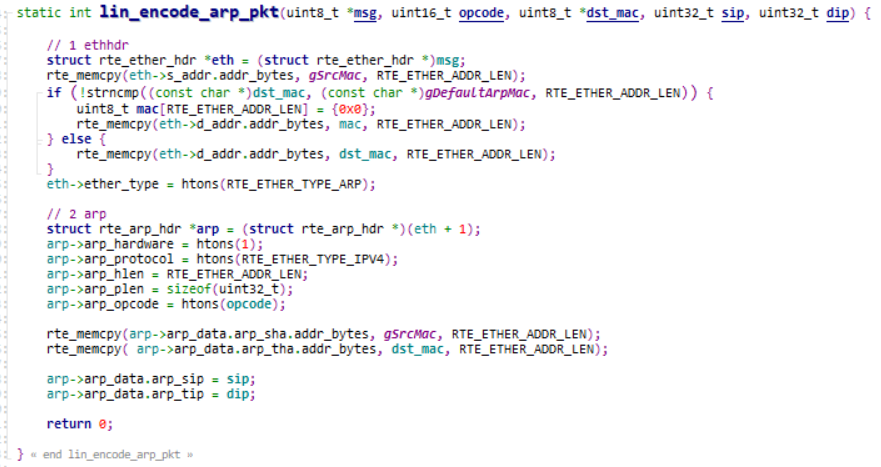


图 61 lin\_encode\_arp\_pkt代码

以上是接受arp数据包的过程，还有返回arp数据包过程。

接下来我们要实现定时发送arp数据包，自己主动维护arptable。

首先我们需要实现一个定时器，首先是回调函数，这个是参考dpdk本身的事件模块的例子。

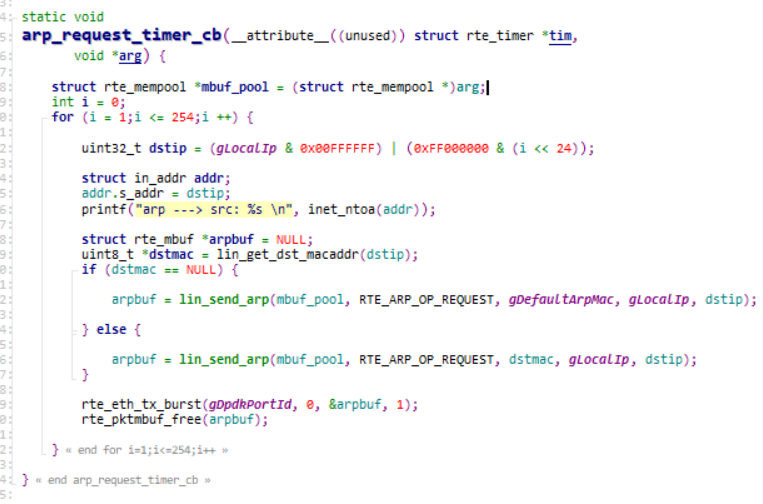


图 62 回调函数

设置计时器

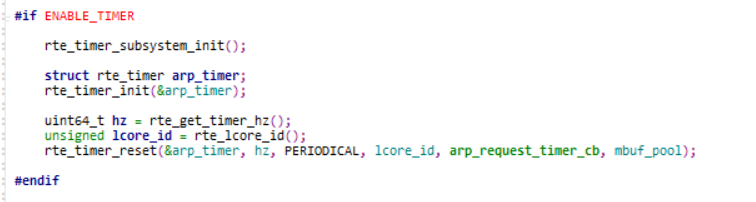


图 63 设置计时器相关信息

监控时间，固定时间回调上面的函数，并且主动发包。

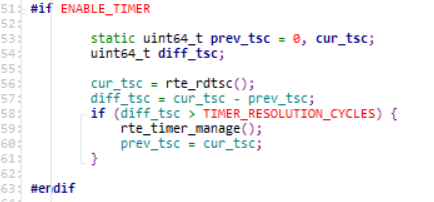


图 64 时间计时器

步骤2 测试arp

启动ubuntu+dpdk还有这个lin\_protocol都在上面讲述了，都是一样流程。首先打开window cmd，输入arp -a

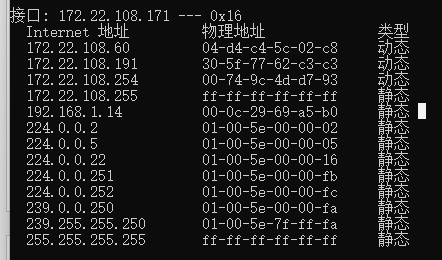


图 65 arp表

表里面没有172.22.108.141信息。现在启动lin\_protocol,在观察arp -a

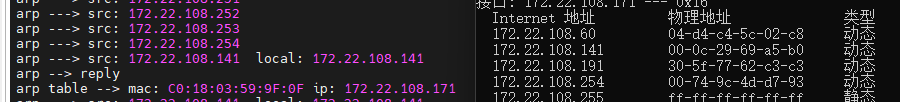


图 66 定时器自动发送rap，维护table

## 四、实验环境

以下仅供参考：

操作系统：centos7、ubuntu 18.10 、window11

平台软件： SourceInsight、编译器gcc

辅助工具：VMware 16pro、Mobaxterm\_personal\_22.0、网络调试助手

网络拓扑：

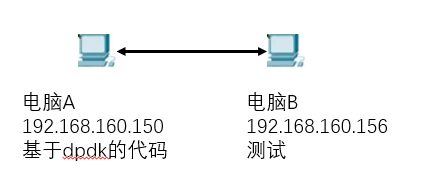


图 67 实验拓扑图

## 五、界面设计

无界面。