哈尔滨工业大学

<<计算机网络>>

实验报告

(2017 年度春季学期)

姓名:	张茗帅
学号:	**1140310606**
学院:	计算机科学与技术学院
教师:	聂兰顺

实验三 IPv4 分组收发实验

一、实验目的

Ipv4 协议是互联网的核心协议，它保证了网络节点（包括网络设备 和主机）在网络层能够按照标准协议互相通信。IPv4 地址唯一标识了网络节点和网络的连接关系。在我们日常使用的计算机的主机协议栈中， IPv4 协议必不可少，它能够接收网络中传送给本机的分组，同时也能根 据上层协议的要求将报文封装为 IPv4 分组发送出去。

本实验通过设计实现主机协议栈中的 IPv4 协议，让学生深入了解网 络层协议的基本原理，学习 IPv4 协议基本的分组接收和发送流程。

另外，通过本实验，学生可以初步接触互联网协议栈的结构和计算 机网络实验系统，为后面进行更为深入复杂的实验奠定良好的基础。

实验环境：

接入到Netriver网络实验系统服务器的主机

Windows XP、Windows7/8/10或苹果OS

开发语言：C语言

二、实验内容

根据计算机网络实验系统所提供的上下层接口函数和协议中分组收发的主要流程，独立设计实现一个简单的 IPv4 分组收发模块。要求实现的主要功能包括：

1 IPv4 分组的基本接收处理，能够检测出接收到的 IP 分组是否存 在如下错误：校验和错、TTL 错、版本号错、头部长度错、错误 目标地址；

2 IPv4 分组的封装发送；

注：不要求实现 IPv4 协议中的选项和分片处理功能

实现 IPv4 分组的基本接收处理功能

对于接收到的 IPv4 分组，检查目的地址是否为本地地址，并检查 IPv4 分组头部中其它字段的合法性。提交正确的分组给上层协议继续处理， 丢弃错误的分组并说明错误类型。

实现 IPv4 分组的封装发送

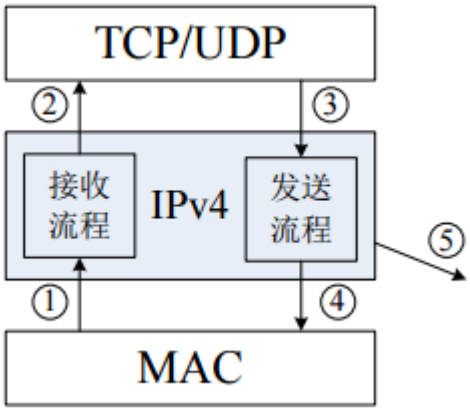
根据上层协议所提供的参数，封装 IPv4 分组，调用系统提供的发送 接口函数将分组发送出去。

三、实验过程及结果

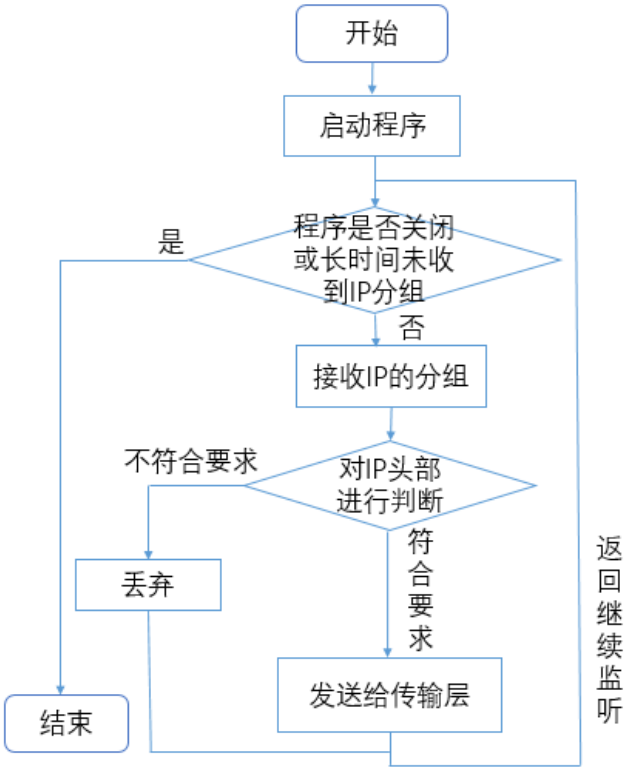
实验原理说明：

1 发送和接收函数的实现程序流程图

概览



接收流程

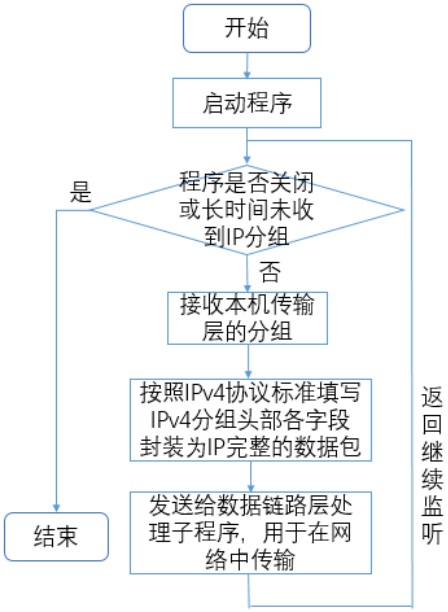


① 检查接收到的 IPv4 分组头部的字段，包括版本号（Version）、头 部长度（IP Head length）、生存时间（Time to live）以及头校验 和（Header checksum）字段。对于出错的分组调用 ip_DiscardPkt( ) 丢弃，说明错误类型。

② 检查 IPv4 分组是否应该由本机接收。如果分组的目的地址是本 机地址或广播地址，则说明此分组是发送给本机的；否则调用 ip_DiscardPkt()丢弃，并说明错误类型。

③ 如果 IPV4 分组应该由本机接收，则提取得到上层协议类型，调用 ip_SendtoUp() 接口函数，交给系统进行后续接收处理。

发送流程



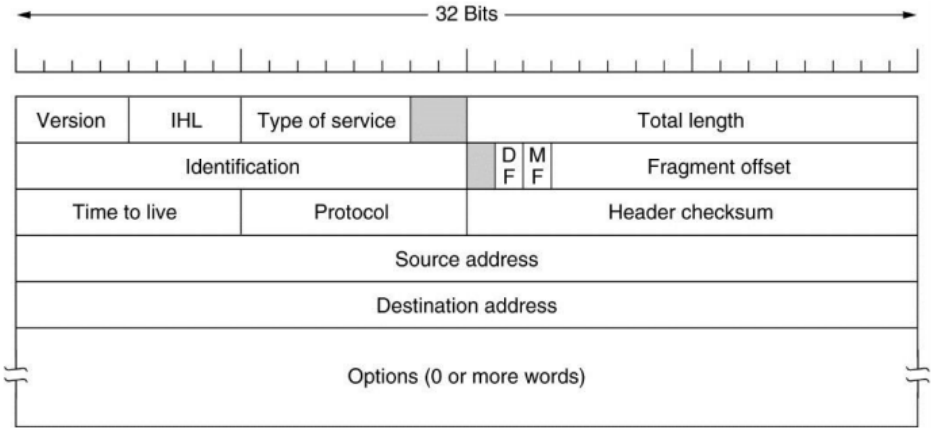
① 根据所传参数（如数据大小），来确定分配的存储空间的大小并 申请分组的存储空间。

② 按照 IPv4 协议标准填写 IPv4 分组头部各字段，标识符 （Identification）字段可以使用一个随机数来填写。（注意：部分 字段内容需要转换成网络字节序）

③ 完成 IPv4 分组的封装后，调用 ip_SendtoLower( )接口函数完成 后续的发送处理工作，最终将分组发送到网络中

2 新建的数据结构

先看一下 IP 头部字段



按照 IPv4 首页的顺序构造结构体，其中 char 是一个字节，即 8Bits，short 是两个字节，即 16 bit，unsigned int 是 4 个字节，即 32 Bits。

结构体的构造完美符合 IPv4 首部的情况

```
struct Ipv4
{
  char version_ihl;                 // 版本号
  char type_of_service;             .// 协议类型
```

```
    short total_length;             // 总长度
    short identification;               // 标志符
    short fragment_offset;          // 偏移量
    char time_to_live;              // TTL
    char protocol;                  // 协议
    short header_checksum;          // 首部校验和
    unsigned int source_address;        // 源地址
    unsigned int destination_address;   // 目标地址
}
```

3 错误检测原理

版本号校验

版本号在第一个字节的前 4 位里面，version_ihl 是结构体的第一个字节，则它的前 4 位代表版本号，右移 4 位，和 0xF 取和运算，如果结果依旧是 4，则代表版本号正确

```
int version = 0xf & ((ipv4->version_ihl)>> 4);
if(version != 4)  {
  ip_DiscardPkt(pBuffer,STUD_IP_TEST_VERSION_ERROR);
  return 1;
}
```

头部长度出错

头部长度在第一个字节的后 4 位里面，则 version_ihl 和 0xF 取和，如果结果为 5，则代表头部长度没有问题

```
int ihl = 0xf & ipv4->version_ihl;
if(ihl < 5) {
  ip_DiscardPkt(pBuffer,STUD_IP_TEST_HEADLEN_ERROR);
  return 1;
}
```

TTL 出错

TTL 存在于第 9 个字节里面，按照数据结构的定义（char + char + short + short + short），存在于 time_to_live 里面，按照规定，如果 ttl 的值是 0，则代表生命周期结束，要抛弃这个包

```
    int ttl = (int)ipv4->time_to_live;
  if(ttl == 0) {
        ip_DiscardPkt(pBuffer,STUD_IP_TEST_TTL_ERROR);
        return 1;
 }
```

目标地址出错

目标地址存在于首部的第 17 -20 个字节中，取出他和本地 Ip 地址做比较，如果不等于本地地址，并且也不等于 0xffffffff，则表示目标地址出错。需要注意字节码序的转变

```
int destination_address = ntohl(ipv4->destination_address);
if(destination_address != getIpv4Address() && destination_address != 0xffffffff) {
   ip_DiscardPkt(pBuffer,STUD_IP_TEST_DESTINATION_ERROR);
```

```
    return 1;
  }
```

校验和出错

校验和存在于 11-12 个字节，校验和检测的规则如下： 16 进制反码求和，也就是说是将所有的字节加起来（校验和部分忽略，即为 0），然后用 ffff 减去

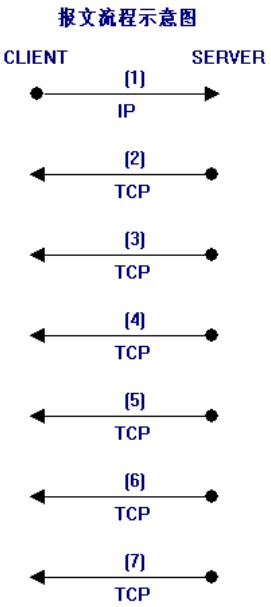
```
int header_checksum = ntohs(ipv4->header_checksum);
int sum = 0;
for(int i = 0; i < ihl*2; i++) {
  if(i!=5)
  {
    sum += (int)((unsigned char)pBuffer[i*2] << 8);
    sum += (int)((unsigned char)pBuffer[i*2+1]);
  }
}

while((sum & 0xffff0000) != 0) {
  sum = (sum & 0xffff) + ((sum >> 16) & 0xffff);
}
unsigned short int ssum = (~sum) & 0xffff;
if(ssum != header_checksum) {
  ip_DiscardPkt(pBuffer,STUD_IP_TEST_CHECKSUM_ERROR);
  return 1;
}
```

4 测试的具体错误数据

一共测试了 7 组数据，第一组是客户端发送给服务器端，接下来六组是服务器端发送给客户端，其中前两组数据均是正确数据，用于验证客户端是否能正确地构造 IP 报文并发送给服务器端以及服务器端发给客户端的 IP 报文能否被正确接收，下面针对其余 5 种错误报文类型给出其错误的原因（具体错在哪）。

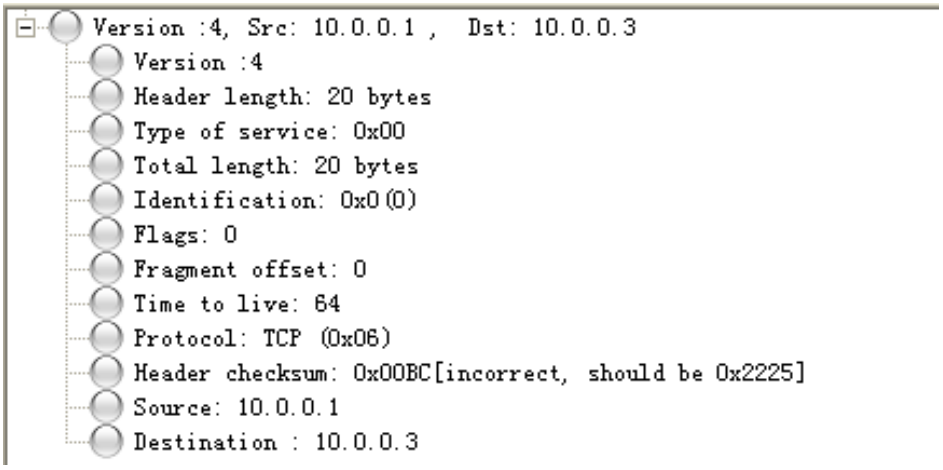
编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Sun May 21 12:17:16.781 2017	10.0.255.243	10.0.255.241	IP	Version 4, ...	2.1 发送IP包
2	Sun May 21 12:17:18.843 2017	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Sun May 21 12:17:20.781 2017	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Sun May 21 12:17:22.828 2017	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Sun May 21 12:17:24.890 2017	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Sun May 21 12:17:26.843 2017	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.6 头部长度错误的IP包
7	Sun May 21 12:17:28.765 2017	10.0.0.1	192.166.77.9	TCP	Bogus TCP h...	2.7 错误目标地址的IP包



（1）编号为 3 的报文 校验和错误

```
0000  00 0D 03 00 00 0A 00 0D 01 00 00 0A 08 00 45 00
0010  00 14 00 00 00 00 40 06 00 BC 0A 00 00 01 0A 00
0020  00 03
```

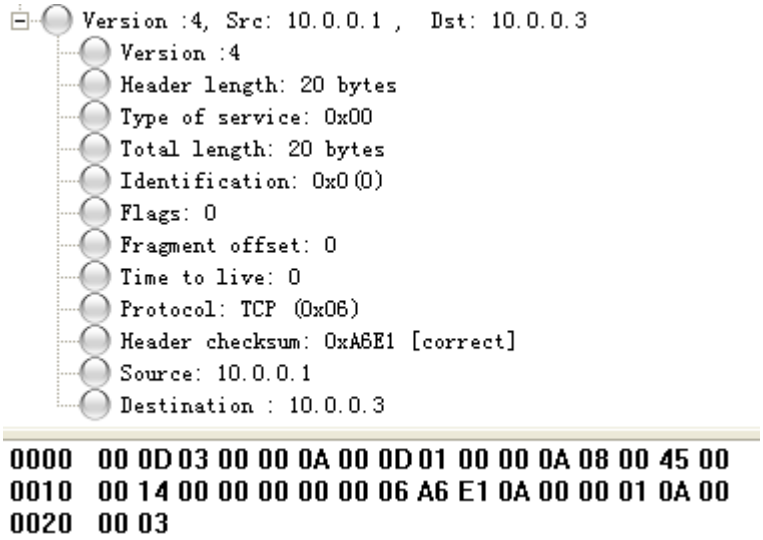
这是该 IP 报文的在网络中以二进制的表示形式，其中我们要从 000E 处开始读，这是 IP 头部字段的第一个字节所在处，也就是在上图用蓝色的框圈的那里，45 正好代表着协议类型为 IPV4 以及首部长度为 5（代着 20 字节），从 000E 开始一直读到 0021 正好就是 IP 头部字段的 20 个字节，我们按照 IP 头部各个字段的含义与其相对应就可以知晓每个字段的值是多少了。下面这张图就是按照这些二进制数字得到的各字段的值。



可以看到，我按照所收到的 IP 头部字段计算其校验和得到 0x2225 与 IP 头部字段所自带的校验和 0x00BC 不相符，所以说该 IP 包不正确，原因为校验和错误。

（2） 编号为 4 的报文 TTL 错误

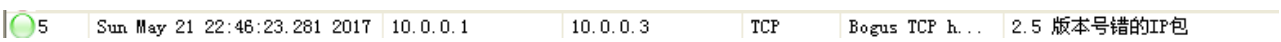
通过其 IP 头部字段的二进制数据分析得到各个字段的值如下

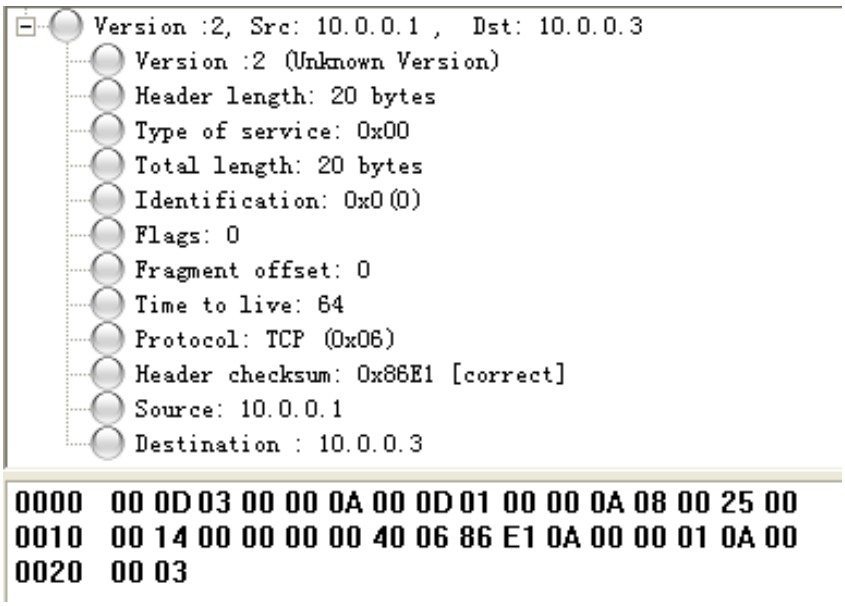


这里面通过二进制数据读取报文依旧是从 000E 出开始读，读到 0021 处，在本实验中每个 IP 数据报的首部字段的起点均为 000E，以后若再次出现，则不加以赘述。

在这里可以发现 TTL 的值为 0，因此这是一个超时的 IP 数据包

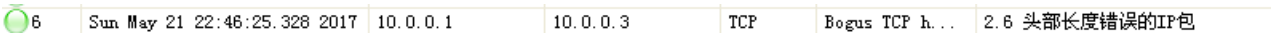
（3）编号为 5 的报文 版本号码错误

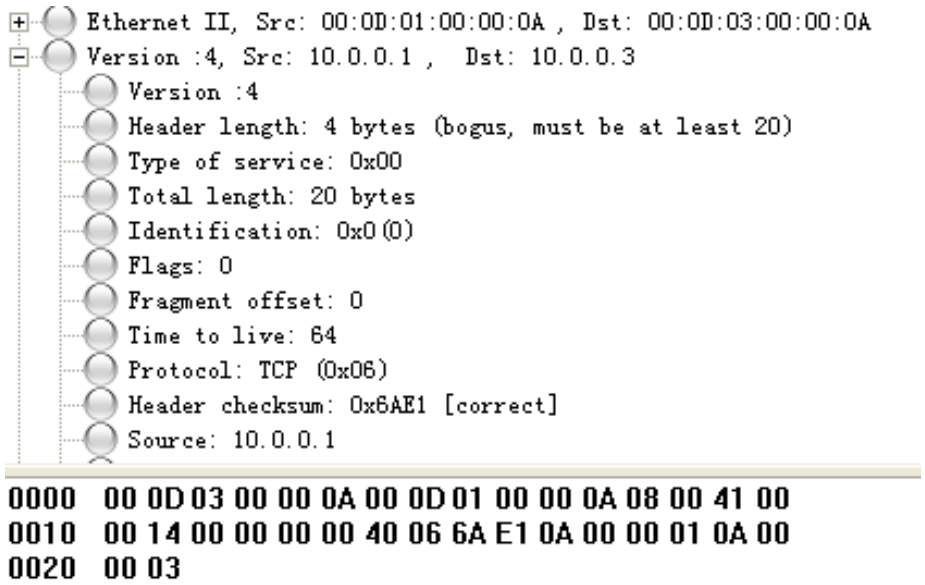




在上面这张图可以看到，得到的版本号码为 2，由于不存在 IPV2，故系统无法识别该 IP 报文。

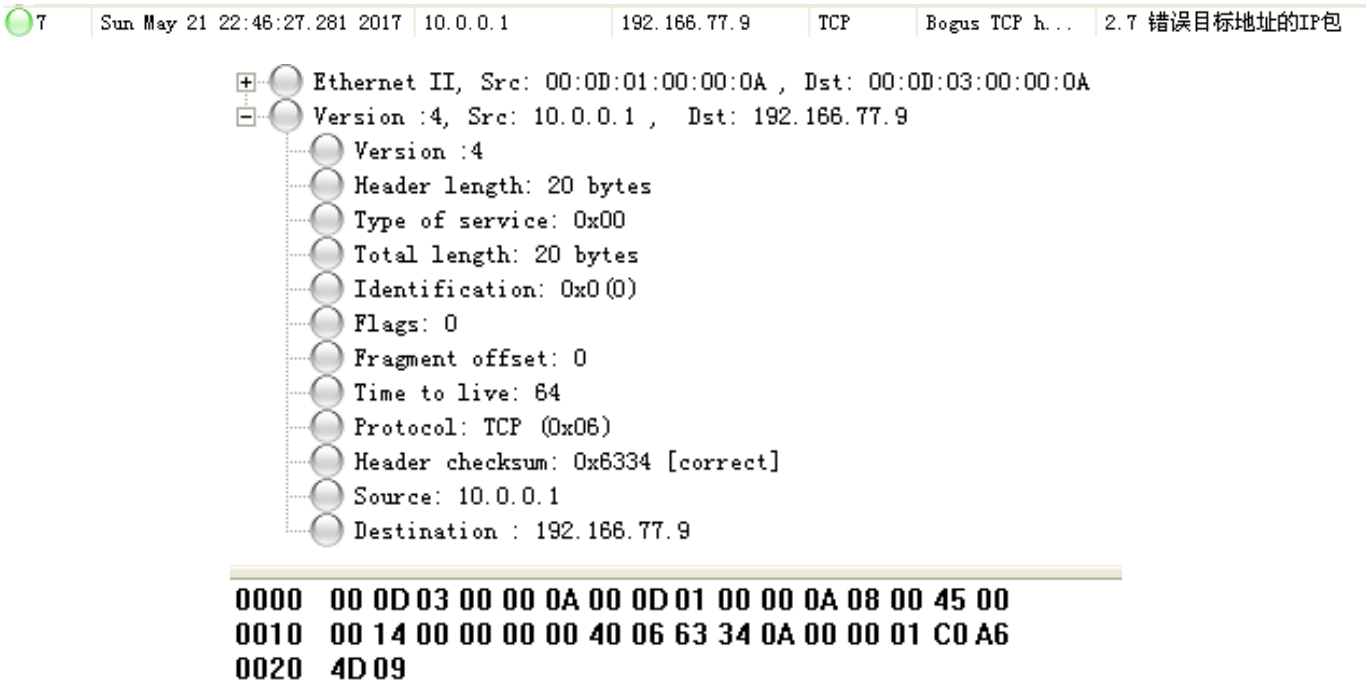
（4）编号为 6 的报文 头部长度错误





我们先在二进制数据的 IP 头部字段前 8 位，这 8 位分别代表着版本号和版本头部字段长度。找到 000E 处，发现这个部分的数据为 41，也就是说版本号吗为 4，代表 IPV4 协议，1 乘以 4 代表 IP 头部字段长度为 4 字节，显然我们知道 IP 头部字段为 20 字节，故该 IP 包不合法，头部字段长度出现问题。

（5）编号为 7 的报文



可以看到，001E 到 0021 这四个字节代表目的地址，也就是 C0 A6 4D 09 翻译成点分十进制也就是 192.166.77.9，这就是 IP 头部字段的目的地址，然而本次实验正确的目的地址应该是 10.0.0.3，因此通过检测可以发现该 IP 报文的目的地址与本机地址不符合，故为非法的 IP 报文。

总体截图：

具体的报文分析请看上面的第四部分，即测试的具体错误数据

源代码（含有详细注释）

```
/*
* THIS FILE IS FOR IP TEST
*/
// system support
#include "sysInclude.h"

extern void ip_DiscardPkt(char* pBuffer,int type);

extern void ip_SendtoLower(char*pBuffer,int length);

extern void ip_SendtoUp(char *pBuffer,int length);

extern unsigned int getIpv4Address();

// implemented by students

//定义 Ipv4 结构体，保存 Ivp4 的头部相关信息
struct Ipv4
{
   char version_ihl;              //版本号&收首部长度
   char type_of_service;          //服务类型
   short total_length;            //总长度
   short identification;          //标识
   short fragment_offset;         //段偏移量
   char time_to_live;             //生存时间
   char protocol;                 //协议类型
   short header_checksum;         //首部校验和
   unsigned int source_address;   //源 IP 地址
   unsigned int destination_address;     //目的 IP 地址
   Ipv4() {                       //初始化归 0
      memset(this,0,sizeof(Ipv4));
   }
   Ipv4(unsigned int len,unsigned int srcAddr,unsigned int dstAddr,
      byte _protocol,byte ttl) {//另外一种初始化方法，传入参数，构造 IP 头部字段
      memset(this,0,sizeof(Ipv4));
      version_ihl = 0x45;
      total_length = htons(len+20);
      time_to_live = ttl;
      protocol = _protocol;
      source_address = htonl(srcAddr);
      destination_address = htonl(dstAddr);

      char *pBuffer;
```

```
        memcpy(pBuffer,this,sizeof(Ipv4));//将头部先输入到接受缓冲区中
        int sum = 0;                        //开始计算头部校验和
        for(int i = 0; i < 10; i++) {
            if(i != 5) {
                sum += (int)((unsigned char)pBuffer[i*2] << 8);
                sum += (int)((unsigned char)pBuffer[i*2+1]);
            }
        }
        while((sum & 0xffff0000) != 0) {//将进位的 1 回加到尾部
            sum = (sum & 0xffff) + ((sum >> 16) & 0xffff);
        }
        unsigned short int ssum = sum;
        header_checksum = htons(~ssum);//取反码作为校验和
    }
};

int stud_ip_recv(char *pBuffer,unsigned short length) //接收接口
{
    Ipv4 *ipv4 = new Ipv4();
    *ipv4 = *(Ipv4*)pBuffer;   //获取接收缓冲区的 IP 头部字段
    int version = 0xf & ((ipv4->version_ihl)>> 4);
    if(version != 4)    {           //判断版本号是否为 4
        ip_DiscardPkt(pBuffer,STUD_IP_TEST_VERSION_ERROR);
        return 1;
    }
    int ihl = 0xf & ipv4->version_ihl;
    if(ihl < 5) {               //判断头部字段是否为 20 字节
        ip_DiscardPkt(pBuffer,STUD_IP_TEST_HEADLEN_ERROR);
        return 1;
    }
    int ttl = (int)ipv4->time_to_live;
    if(ttl == 0) {              //判断 TTL 是否合法
        ip_DiscardPkt(pBuffer,STUD_IP_TEST_TTL_ERROR);
        return 1;
    }
    int destination_address = ntohl(ipv4->destination_address);    //判断目的地址是否为本机地址
    if(destination_address != getIpv4Address() && destination_address != 0xffffffff) {
        ip_DiscardPkt(pBuffer,STUD_IP_TEST_DESTINATION_ERROR);
        return 1;
    }
    int header_checksum = ntohs(ipv4->header_checksum);
    int sum = 0;
    for(int i = 0; i < ihl*2; i++) {     //计算目前接收到的头部的校验和
        if(i!=5)
```

```
        {
            sum += (int)((unsigned char)pBuffer[i*2] << 8);
            sum += (int)((unsigned char)pBuffer[i*2+1]);
        }
    }

    while((sum & 0xffff0000) != 0) {
        sum = (sum & 0xffff) + ((sum >> 16) & 0xffff);
    }
    unsigned short int ssum = (~sum) & 0xffff;
    if(ssum != header_checksum) { //通过和曾经的校验和字段对比判断 IP 头部是否发生了改
变
        ip_DiscardPkt(pBuffer,STUD_IP_TEST_CHECKSUM_ERROR);
        return 1;
    }
    ip_SendtoUp(pBuffer,length);   //一切均合法，发送给上一层
    return 0;
}
                    //发送接口
int stud_ip_Upsend(char *pBuffer,unsigned short len,unsigned int srcAddr,
                   unsigned int dstAddr,byte protocol,byte ttl)
{
    char *pack_to_sent = new char[len+20];   //加上头部字段的长度
    memset(pack_to_sent,0,len+20);              //初始化头部字段
    *((Ipv4*)pack_to_sent) = Ipv4(len,srcAddr,dstAddr,protocol,ttl);//填充头部字段
    memcpy(pack_to_sent+20,pBuffer,len);   //将上层协议数据报文填充形成完整 IP 报文
    ip_SendtoLower(pack_to_sent,len+20);   //发送给下一层
    delete[] pack_to_sent;

    return 0;
}
```

四、实验心得

通过本次实验，我对 Ipv4 协议有了较为深入的理解，对于 IP 头部字段的每一部分的作用和意义认识的很深刻，尤其是校验和字段。之前在 UDP 协议的学习中便学习过校验和计算的相关原理，但是时间久了就忘记了一部分，这几天通过实验很好的捡起来了这部分知识点，而且通过实践对这个问题理解地更加深刻。在 C 语言中熟悉掌握了移位运算的相关操作，提高了自己的编程技巧，收获颇丰。