哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

<table>
<tr><td>实验名称</td><td colspan="5">IPV4 收发和转发实验</td></tr>
<tr><td>姓名</td><td colspan="2">梅智敏</td><td>院系</td><td colspan="2">计算机科学与技术</td></tr>
<tr><td>班级</td><td colspan="2">1837101</td><td>学号</td><td colspan="2">1183710118</td></tr>
<tr><td>任课教师</td><td colspan="2">李全龙</td><td>指导教师</td><td colspan="2">李全龙</td></tr>
<tr><td>实验地点</td><td colspan="2">格物 213</td><td>实验时间</td><td colspan="2">2020.11.14</td></tr>
<tr><td rowspan="2">实验课表现</td><td>出勤、表现得分(10)</td><td></td><td rowspan="2">实验报告
得分(40)</td><td rowspan="2"></td><td rowspan="2">实验总分</td><td rowspan="2"></td></tr>
<tr><td>操作结果得分(50)</td><td></td></tr>
<tr><td colspan="6">教师评语</td></tr>
<tr><td colspan="6"></td></tr>
</table>

计算学部

实验目的：

- Part1：IPV4 收发实验

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。

- Part2：IPV4 转发实验

通过前面的实验，我们已经深入了解了IPv4协议的分组接收和发送处理流程。本实验需要将实验模块的角色定位从通信两端的主机转移到作为中间节点的路由器上，在IPv4分组收发处理的基础上，实现分组的路由转发功能。

网络层协议最为关注的是如何将IPv4分组从源主机通过网络送达目的主机，这个任务就是由路由器中的IPv4协议模块所承担。路由器根据自身所获得的路由信息，将收到的IPv4分组转发给正确的下一跳路由器。如此逐跳地对分组进行转发，直至该分组抵达目的主机。IPv4分组转发是路由器最为重要的功能。

本实验设计模拟实现路由器中的 IPv4 协议，可以在原有 IPv4 分组收发实验的基础上，增加 IPv4 分组的转发功能。对网络的观察视角由主机转移到路由器中，了解路由器是如何为分组选择路由，并逐跳地将分组发送到目的主机。本实验中也会初步接触路由表这一重要的数据结构，认识路由器是如何根据路由表对分组进行转发的。

实验内容：

概述本次实验的主要内容，包含的实验项等。

- Part1：IPV4的收发实验

1. 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。

2. 实现 IPv4 分组的封装发送功能

   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。

- Part2：IPV4的转发实验

1. 设计路由表数据结构。

   设计路由表所采用的数据结构。要求能够根据目的IPv4地址来确定分组处理行（转发情况下需获得下一跳的IPv4地址）。路由表的数据结构和查找算法会极大的影响路由器的转发性能，有兴趣的同学可以深入思考和探索。

2. IPv4分组的接收和发送。

   对前面实验（IP实验）中所完成的代码进行修改，在路由器协议栈的IPv4模块中能够正确完成分组的接收和发送处理。具体要求不做改变，参见“IP实验”。

3. IPv4分组的转发。

对于需要转发的分组进行处理，获得下一跳的IP地址，然后调用发送接口函数做进一步处理。

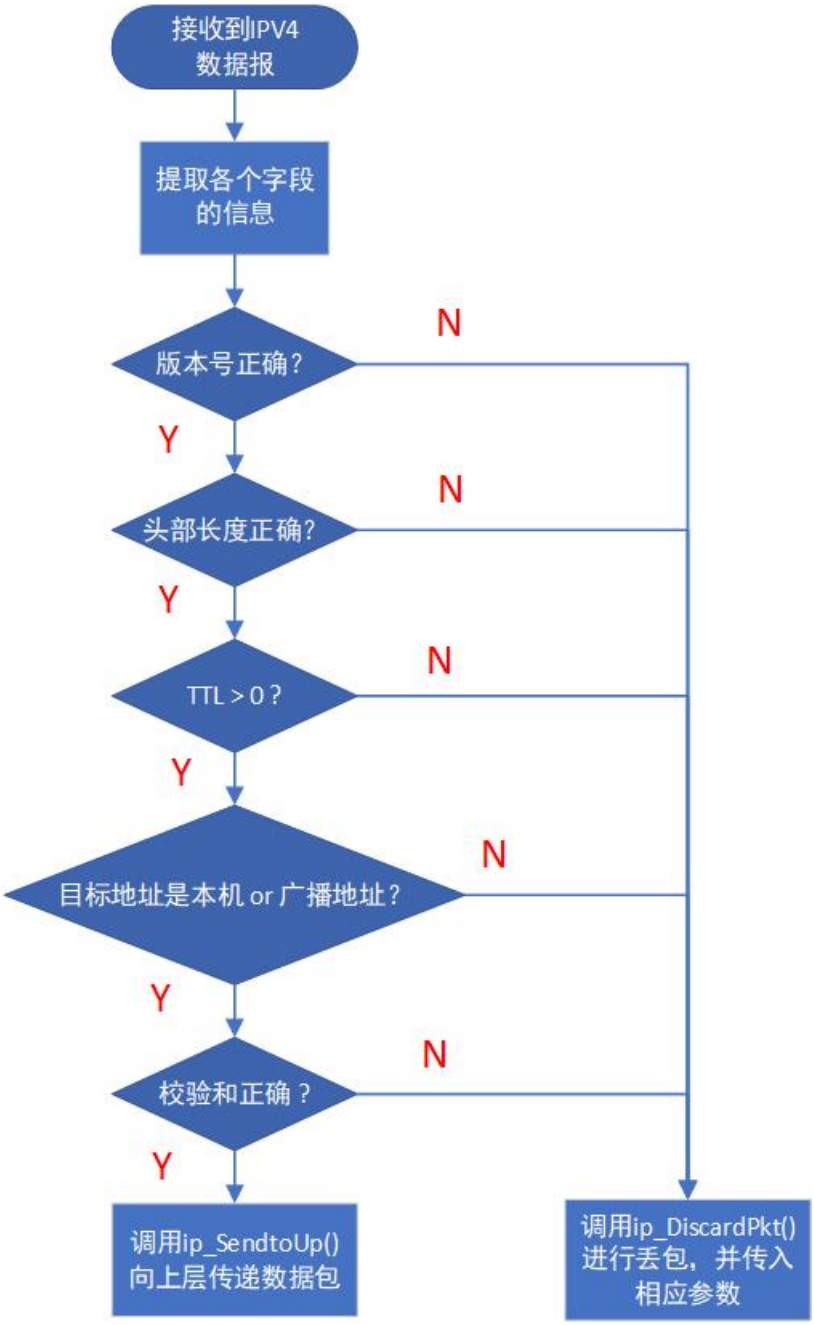
实验过程：

以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

- Part1：IPV4收发实验

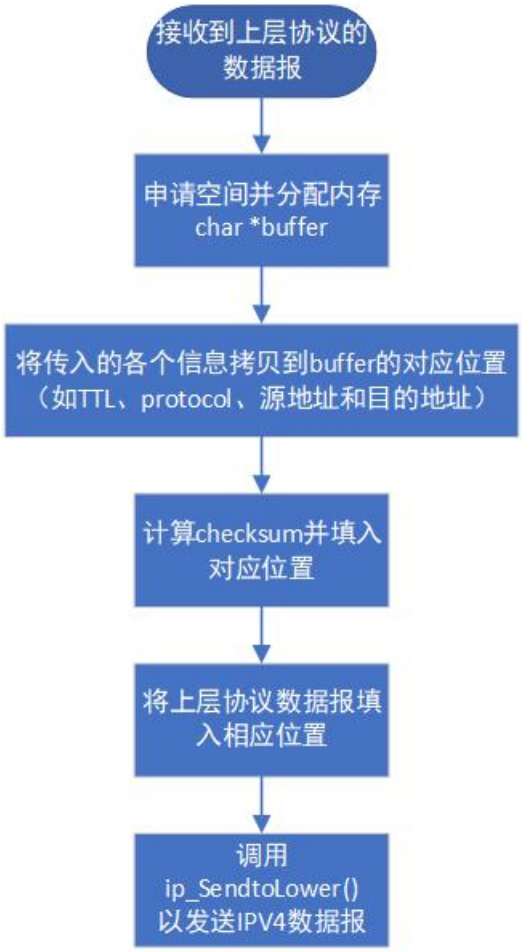
1. 发送和接收函数的实现程序流程图

接收分组函数int stud_ip_recv(char *pBuffer, unsigned short length)的流程图如下：



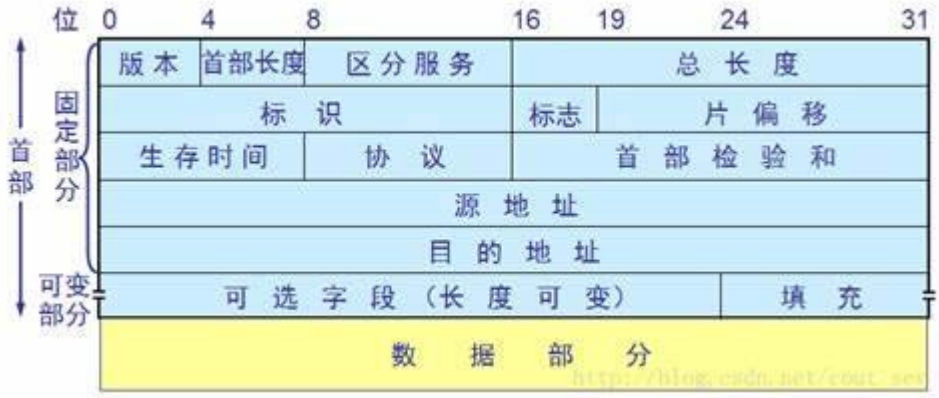
发送分组函数

int stud_ip_Upsend(char* pBuffer, unsigned short len, unsigned int srcAddr, unsigned int dstAddr ,byte protocol, byte ttl)的流程图如下：



2. 字段的错误检测原理

我们需要先了解IPV4数据报文段结构



以及系统提供的接口函数void ip_DiscardPkt(char * pBuffer ,int type)

```
void ip_DiscardPkt(char * pBuffer ,int type)
参数：
    pBuffer：指向被丢弃分组的指针
    type：分组被丢弃的原因，可取以下值：
     STUD_IP_TEST_CHECKSUM_ERROR      //IP 校验和出错
     STUD_IP_TEST_TTL_ERROR           //TTL 值出错
     STUD_IP_TEST_VERSION_ERROR       //IP 版本号错
     STUD_IP_TEST_HEADLEN_ERROR       //头部长度错
     STUD_IP_TEST_DESTINATION_ERROR   //目的地址错
```

a) **检查版本号**：版本号在第0个字节的前4位，因此需要先进行移位

```
// 第0个字节的前4位：ip version
int version = pBuffer[0] >> 4;
```

再将其和4比较即可。若版本号不正确，则丢包，并传递错误原因参数

```
// 检查版本号
if (version != 4)
{
    ip_DiscardPkt(pBuffer, STUD_IP_TEST_VERSION_ERROR);
    return 1;
}
```

b) **检查头部长度**：根据段结构，头部长度信息存储在第0字节的后4位，所以通过“&”运算进行提取低位信息

```
// 第0个字节的后4位：   head length
int head_len = pBuffer[0] & 0xf;
```

头部长度信息是以4字节为单位，根据段结构图：最小的IP数据包的头部信息为20字节，故head_len 必须 >= 5，若不满足则丢包并传递错误原因参数

```
// 检查头部长度
if (head_len < 5)
{
    ip_DiscardPkt(pBuffer, STUD_IP_TEST_HEADLEN_ERROR);
    return 1;
}
```

c) **检查TTL**：TTL存储在第8个字节，直接读取即可

```
// 第8个字节：TTL
short ttl = (unsigned short)pBuffer[8];
```

TTL降为0说明此IP数据报已经失效，故直接丢包即可，并传递错误原因参数

```
// 检查TTL
if (ttl == 0)
{
    ip_DiscardPkt(pBuffer, STUD_IP_TEST_TTL_ERROR);
    return 1;
}
```

d) **检查Checksum**：checksum字段保存在IP数据报的第10个字节，直接读取即可，需要注意的是此处要将网络字节序转换为本地字节序

```
// 第10个字节 : 校验和
short checksum = ntohs(*(unsigned short *)(pBuffer + 10));
```

至于checksum的计算方法，如下

◆◆当发送IP包时，需要计算IP报头的校验和：

1、 把校验和字段置为0；

2、 对IP头部中的每16bit进行二进制求和；

3、 如果和的高16bit不为0，则将和的高16bit和低16bit反复相加，直到和的高16bit为0，从而获得一个16bit的值

4、 将该16bit的值取反，存入校验和字段。

◆◆当接收IP包时，需要对报头进行确认，检查IP头是否有误：

算法同上2、3步(不需要将校验和字段置0)，然后判断取反的结果是否为0，是则正确，否则有错。

发送IP数据包时，具体实现：

```
// 计算校验和
unsigned long sum = 0;
unsigned long temp = 0;
int i;
// 每16bits进行二进制求和
for (i = 0; i < 20; i += 2)
{
    temp += (unsigned char)buffer[i] << 8;
    temp += (unsigned char)buffer[i + 1];
    sum += temp;
    temp = 0;
}
unsigned short l_word = sum & 0xffff;
unsigned short h_word = sum >> 16;
unsigned short checksum = l_word + h_word;
checksum = ~checksum;
// 转换为网络字节序
unsigned short header_checksum = htons(checksum);
// 将校验和拷贝到相应位置
memcpy(buffer + 10, &header_checksum, 2);
```

接收IP数据包时，具体实现：

```
// 检查校验和
unsigned long sum = 0;
unsigned long temp = 0;
int i;
// 每16bits进行二进制求和
for (i = 0; i < head_len * 2; i++)
{
    temp += (unsigned char)pBuffer[i * 2] << 8;
    temp += (unsigned char)pBuffer[i * 2 + 1];
    sum += temp;
    temp = 0;
}
unsigned short low_word = sum & 0xffff;
unsigned short high_word = sum >> 16;
// 校验和有误，则丢包
if (low_word + high_word != 0xffff)
{
    ip_DiscardPkt(pBuffer, STUD_IP_TEST_CHECKSUM_ERROR);
    return 1;
}
```

- Part2：IPV4转发实验

1. 新建的数据结构

a) 路由项routeTableItem结构体，包含转发IP数据报时的各项信息

```
struct routeTableItem
{
    unsigned int destIP;      // 目的IP
    unsigned int mask;        // 掩码
    unsigned int masklen;     // 掩码长度
    unsigned int nexthop;     // 下一跳
};
```

b) Vector<routeTableItem>容器，用来充当路由表的角色

```
vector<routeTableItem> route_table;
```

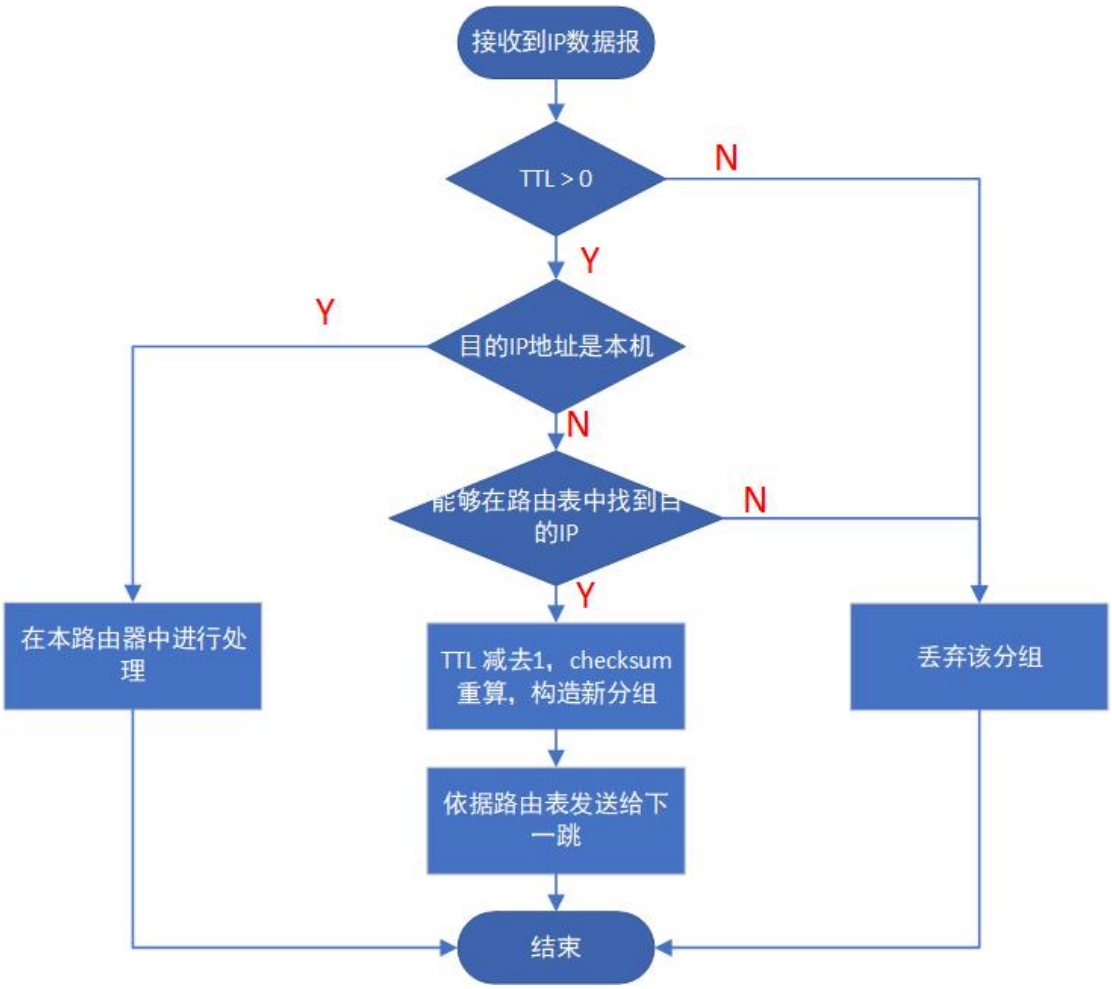
2. 函数流程图

a) 路由表初始化函数void stud_Route_Init()

直接将我们的路由器清空即可

b) 路由表添加函数void stud_route_add(stud_route_msg *proute)
新建一个路由项对象，再给各个域赋值，然后添加到vector容器即可

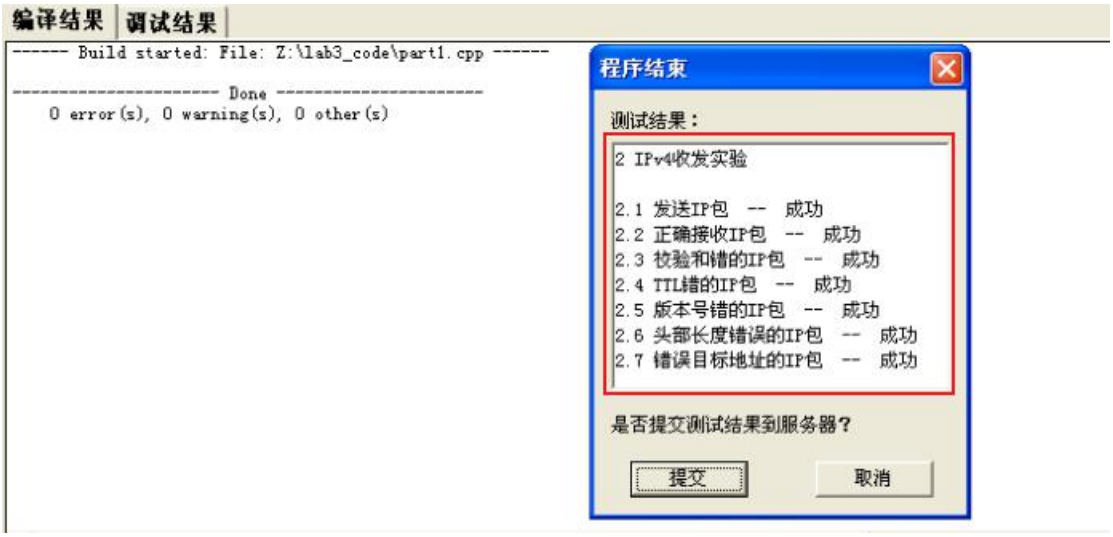
c) 路由表转发函数int stud_fwd_deal(char *pBuffer, int length)



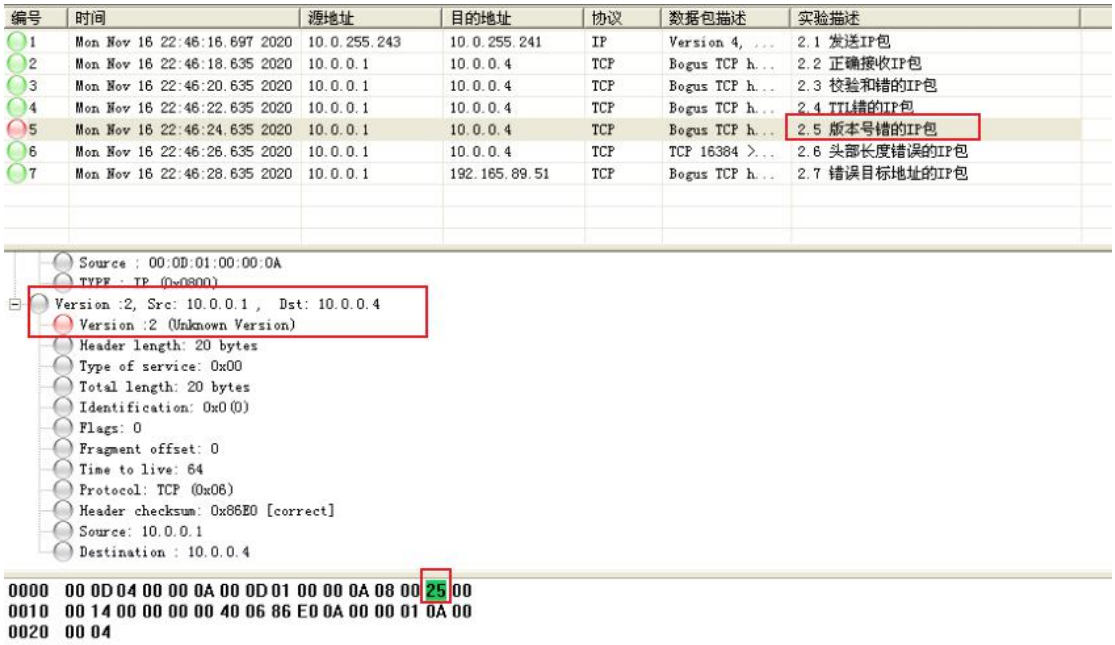
实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

- Part1：IPV4收发实验



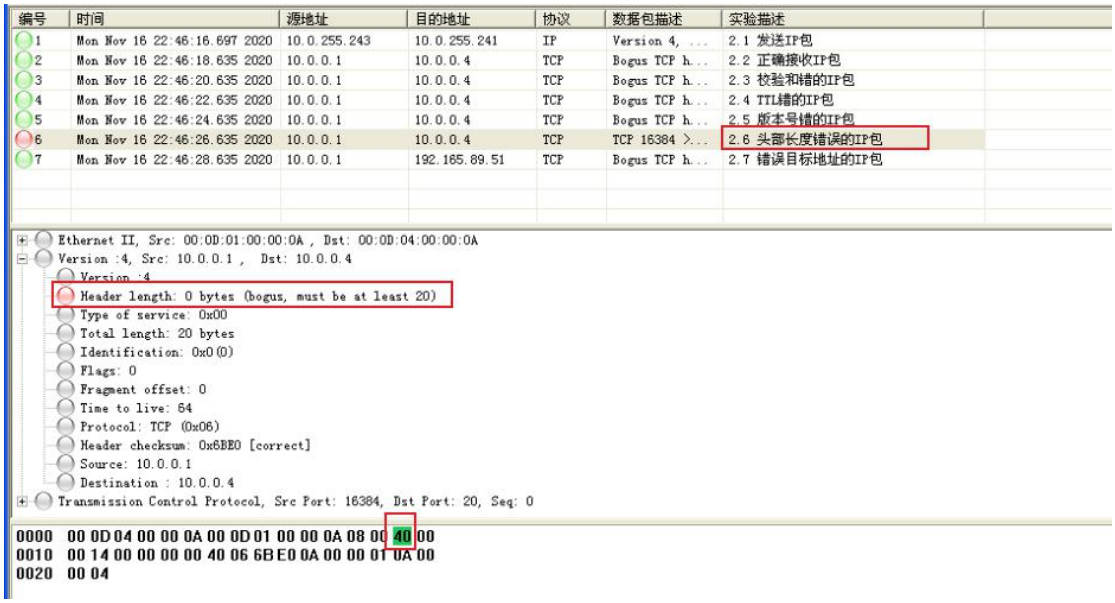
1） 版本号错误的数据报：



编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Mon Nov 16 22:46:16.697 2020	10.0.255.243	10.0.255.241	IP	Version 4, ...	2.1 发送IP包
2	Mon Nov 16 22:46:18.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Mon Nov 16 22:46:20.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Mon Nov 16 22:46:22.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Mon Nov 16 22:46:24.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Mon Nov 16 22:46:26.635 2020	10.0.0.1	10.0.0.4	TCP	TCP 16384 >...	2.6 头部长度错误的IP包
7	Mon Nov 16 22:46:28.635 2020	10.0.0.1	192.165.89.51	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

可见此数据报的verison值为2，与我们IPV4数据报的version值为4不相符

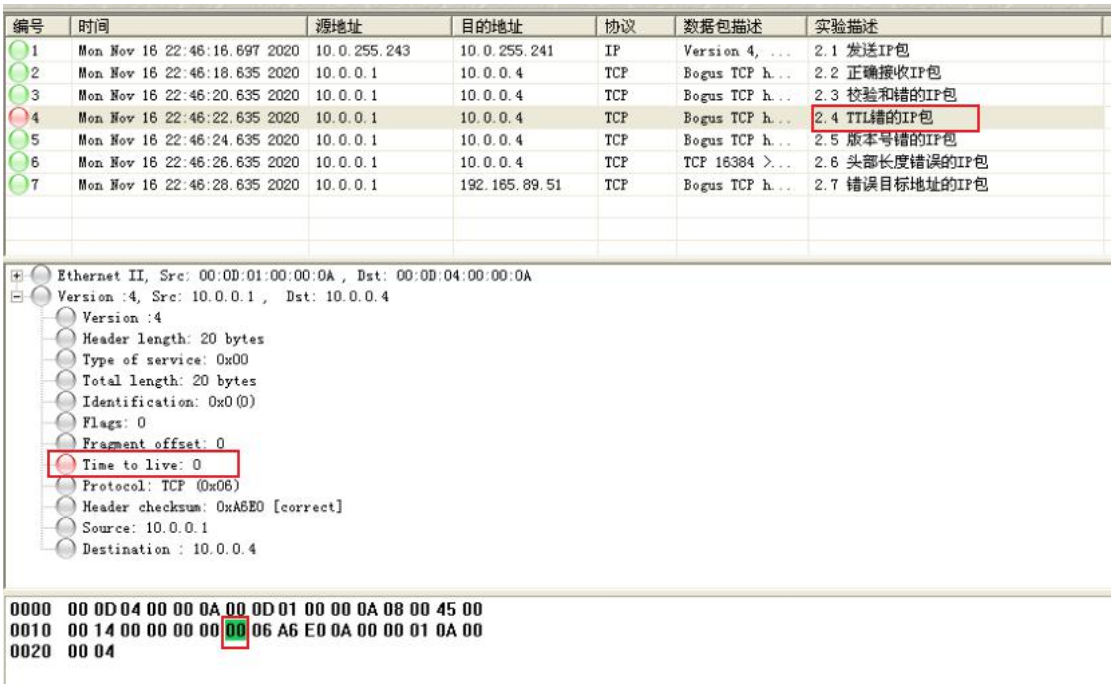
2） 头部长度错误的数据报：



编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Mon Nov 16 22:46:16.697 2020	10.0.255.243	10.0.255.241	IP	Version 4, ...	2.1 发送IP包
2	Mon Nov 16 22:46:18.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Mon Nov 16 22:46:20.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Mon Nov 16 22:46:22.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Mon Nov 16 22:46:24.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Mon Nov 16 22:46:26.635 2020	10.0.0.1	10.0.0.4	TCP	TCP 16384 >...	2.6 头部长度错误的IP包
7	Mon Nov 16 22:46:28.635 2020	10.0.0.1	192.165.89.51	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

可见此数据报的头部长度值为0，与实际的>=5不相符

3） TTL字段错误的数据报：



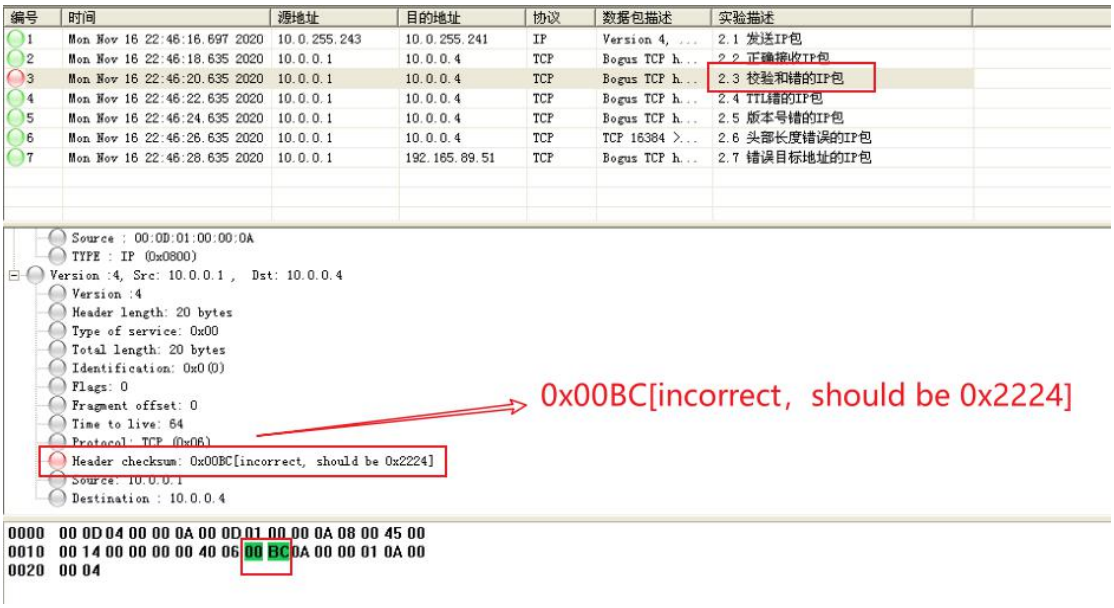
编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Mon Nov 16 22:46:16.697 2020	10.0.255.243	10.0.255.241	IP	Version 4, ...	2.1 发送IP包
2	Mon Nov 16 22:46:18.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Mon Nov 16 22:46:20.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Mon Nov 16 22:46:22.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Mon Nov 16 22:46:24.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Mon Nov 16 22:46:26.635 2020	10.0.0.1	10.0.0.4	TCP	TCP 16384 >...	2.6 头部长度错误的IP包
7	Mon Nov 16 22:46:28.635 2020	10.0.0.1	192.165.89.51	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

```
Ethernet II, Src: 00:0D:01:00:00:0A , Dst: 00:0D:04:00:00:0A
Version :4, Src: 10.0.0.1 ,  Dst: 10.0.0.4
    Version :4
    Header length: 20 bytes
    Type of service: 0x00
    Total length: 20 bytes
    Identification: 0x0 (0)
    Flags: 0
    Fragment offset: 0
    Time to live: 0
    Protocol: TCP (0x06)
    Header checksum: 0xA6E0 [correct]
    Source: 10.0.0.1
    Destination : 10.0.0.4
```

```
0000  00 0D 04 00 00 0A 00 0D 01 00 00 0A 08 00 45 00
0010  00 14 00 00 00 00 00 06 A6 E0 0A 00 00 01 0A 00
0020  00 04
```

可见此数据报的TTL字段值为0，说明此IP报已经失效。

4) Checksum字段错误的数据报：



编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Mon Nov 16 22:46:16.697 2020	10.0.255.243	10.0.255.241	IP	Version 4, ...	2.1 发送IP包
2	Mon Nov 16 22:46:18.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Mon Nov 16 22:46:20.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Mon Nov 16 22:46:22.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Mon Nov 16 22:46:24.635 2020	10.0.0.1	10.0.0.4	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Mon Nov 16 22:46:26.635 2020	10.0.0.1	10.0.0.4	TCP	TCP 16384 >...	2.6 头部长度错误的IP包
7	Mon Nov 16 22:46:28.635 2020	10.0.0.1	192.165.89.51	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

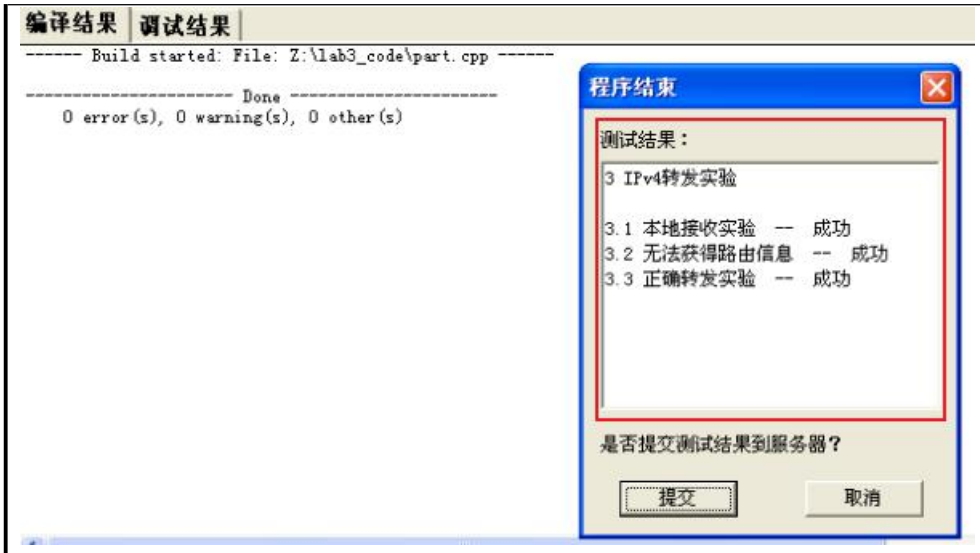
```
    Source : 00:0D:01:00:00:0A
    TYPE : IP (0x0800)
Version :4, Src: 10.0.0.1 ,  Dst: 10.0.0.4
    Version :4
    Header length: 20 bytes
    Type of service: 0x00
    Total length: 20 bytes
    Identification: 0x0 (0)
    Flags: 0
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (0x06)
    Header checksum: 0x00BC[incorrect, should be 0x2224]
    Source: 10.0.0.1
    Destination : 10.0.0.4
```

0x00BC[incorrect, should be 0x2224]

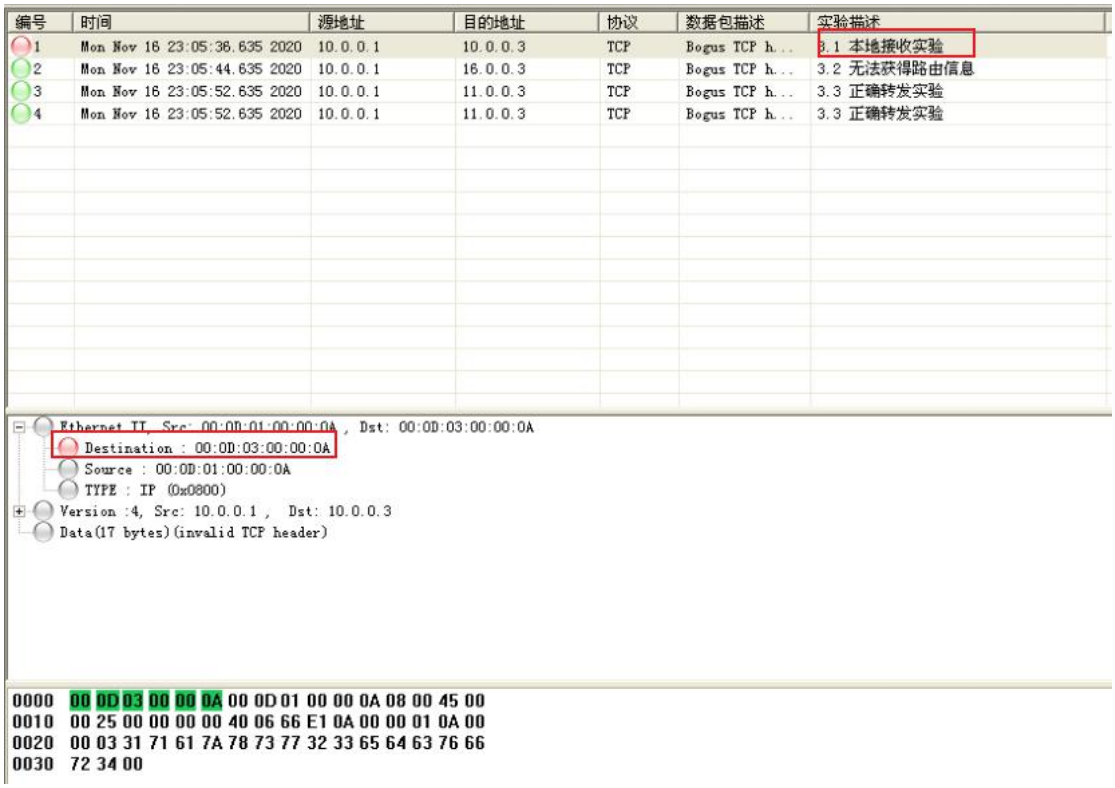
```
0000  00 0D 04 00 00 0A 00 0D 01 00 00 0A 08 00 45 00
0010  00 14 00 00 00 00 40 06 00 BC 0A 00 00 01 0A 00
0020  00 04
```

可见此数据报的checksum字段值不正确

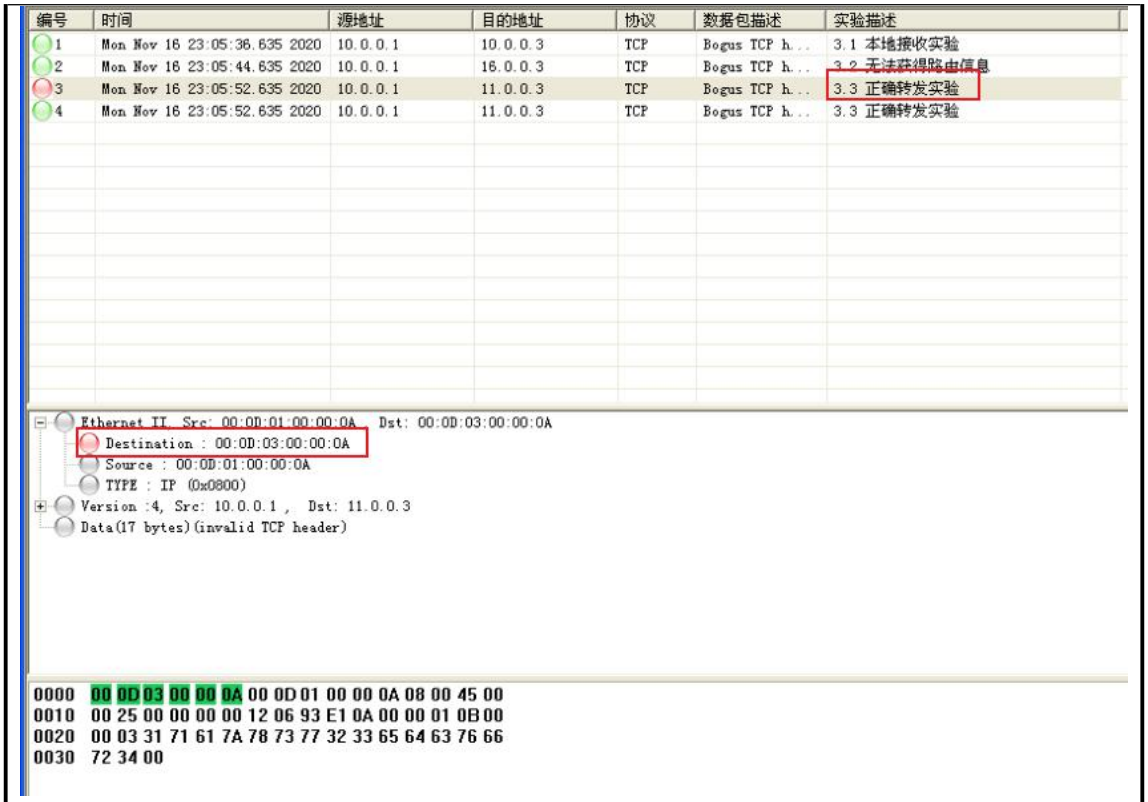
- Part2：IPV4转发实验



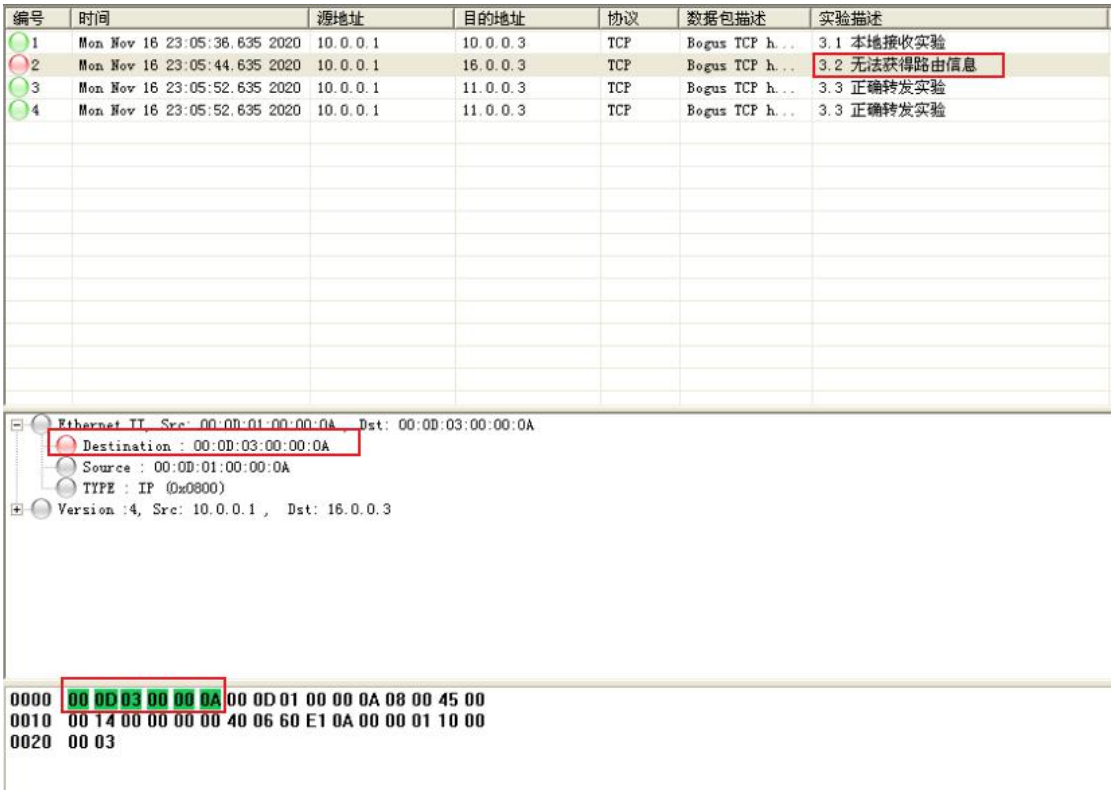
1） 目的地址就是本机，直接接收



2） 路由表中存在目的地址



3) 路由表中不存在目的地址



问题讨论：

问题：在存在大量分组的情况下如何提高转发效率？

答：

1. 可以改进数据结构，利用堆查询的方法进行路由转发查找，提高查找效率，间接提高转发效率。
2. 可以使用多线程进行，多个转发同时进行，可以直接提高转发效率。
3. 可以可以考虑边收边发，以提高整体运转的效率。

心得体会：

1. 通过本次实验，我对于IPV4的收发和转发过程有了更深入的理解，同时更熟悉了IP数据报的段结构
2. 本次的实验平台无法在win10上面运行，为此，我只能安装XP虚拟机来运转此实验平台，虽然比较麻烦，但是也锻炼了我的动手能力