哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	实验 3：IPv4 分组收发和转发实验				
姓名	张志路		院系	计算机学院	
班级	1603106		学号	1160300909	
任课教师	聂兰顺		指导教师	聂兰顺	
实验地点	格物 207		实验时间	2018 年 11 月 14 日	
实验课表现	出勤、表现得分(10)		实验报告得分(40)		实验总分	
	操作结果得分(50)				
教师评语					

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

目 录

实验 3-1：IPv4 分组收发实验

1 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。

2 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 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，要求能够检测出接收到的 IP 分组是否存在校验和错、TTL 错、版本号错、头部长度错、错误目标地址等不合法内容。

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

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。

注：不要求实现 IPv4 协议中的选项和分片处理功能。

3 实验环境

① 工具：接入到 Netriver 网络实验系统服务器的主机

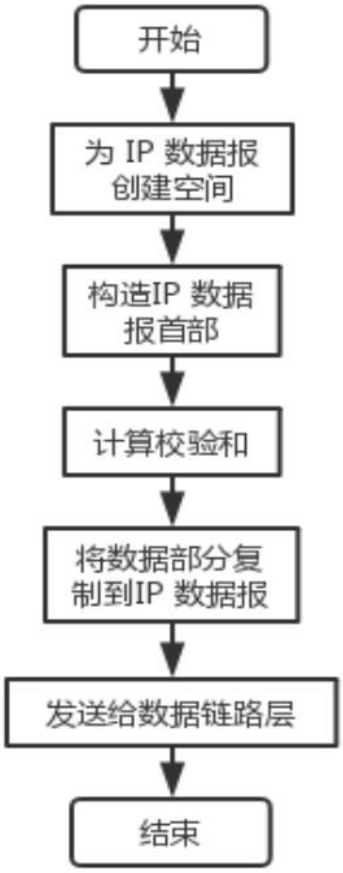
② 操作系统：WindowsXP 虚拟机

③ 开发语言：C 语言

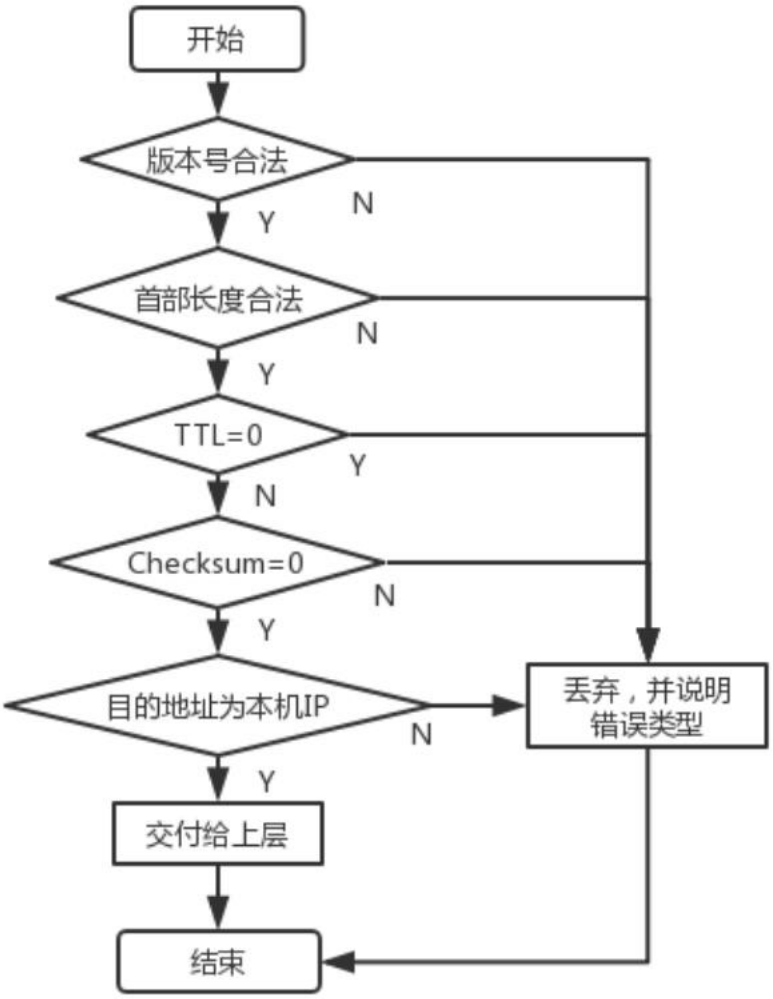
4 实验过程

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。

4.1 发送函数的程序流程图



4.2 接收函数的程序流程图



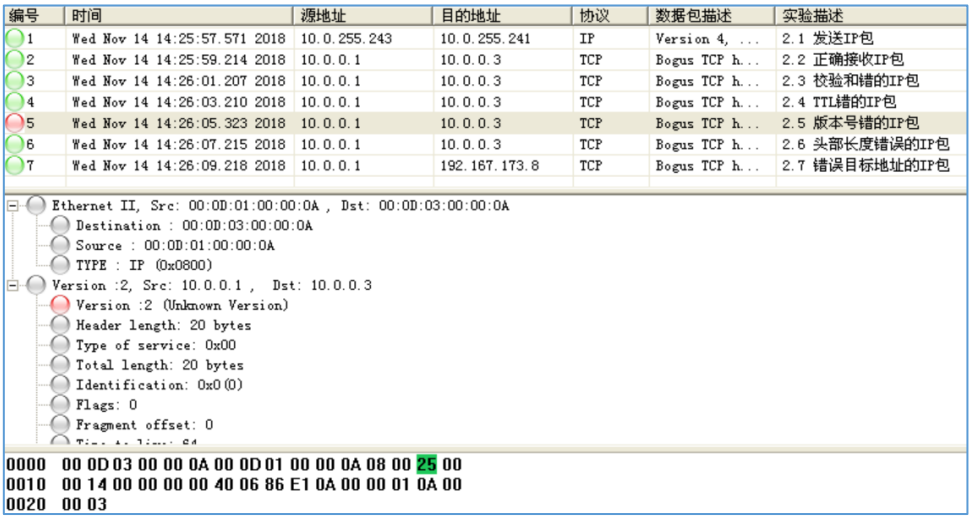
4.3 各字段的错误检测原理和具体数据

(1) 版本号

① 检测原理

```
1.  if((pBuffer[0] & 0xf0) != 0x40) //check Version
2.  {
3.      ip_DiscardPkt(pBuffer,STUD_IP_TEST_VERSION_ERROR);
4.      return 1;
5.  }
```

② 具体数据



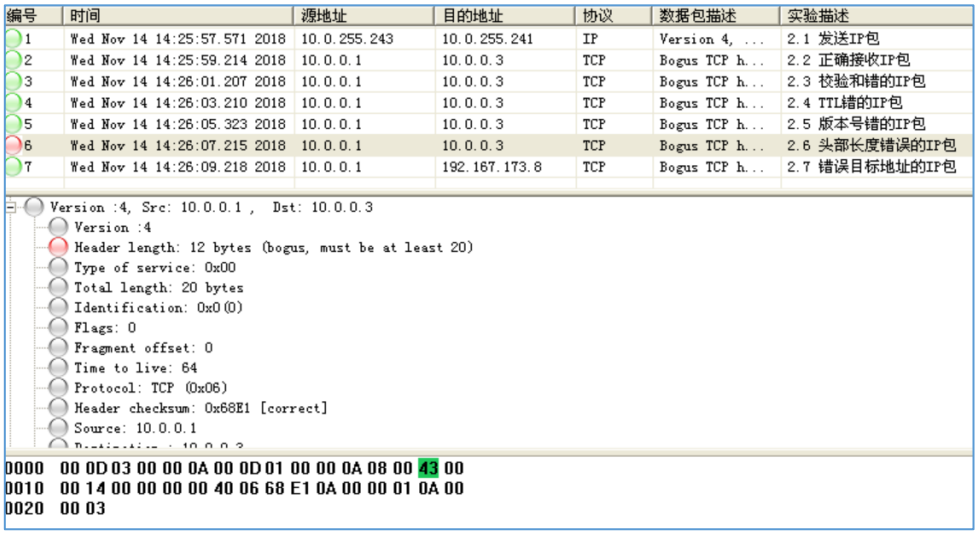
如上图，错误版本号为 2。

(2) 头部长度

① 检测原理

```
1.  if((pBuffer[0] & 0x0f) != 0x05) //check IP Head length (4 Bytes/unit)
2.  {
3.      ip_DiscardPkt(pBuffer,STUD_IP_TEST_HEADLEN_ERROR);
4.      return 1;
5.  }
```

② 具体数据



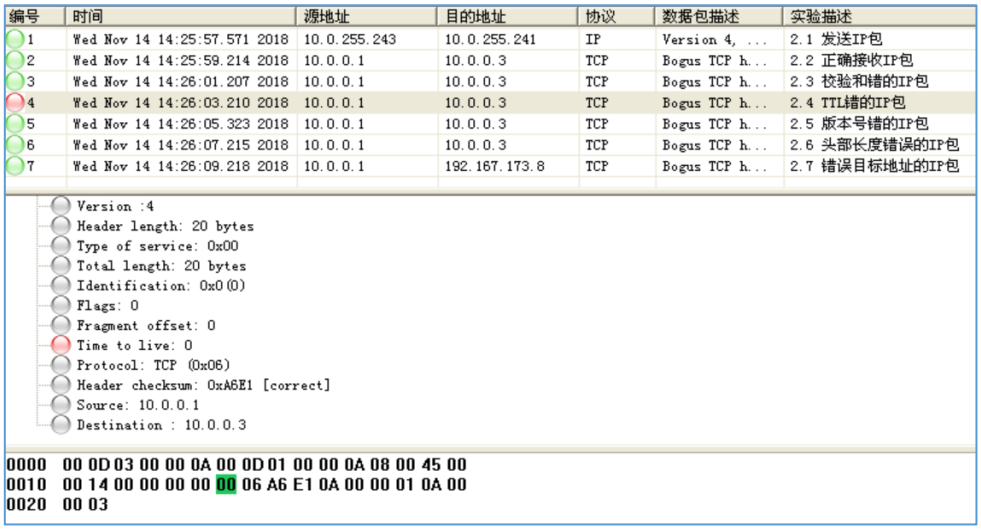
如上图，错误头部长度为 12 字节。

(3) 生存时间 TTL

① 检测原理

```
1.  if(pBuffer[8] == 0x00)  //check TTL
2.  {
3.      ip_DiscardPkt(pBuffer,STUD_IP_TEST_TTL_ERROR);
4.      return 1;
5.  }
```

② 具体数据



如上图，错误 TTL 为 0。

(4) 头部校验和

① 检测原理

a. 校验和计算

```
1.  /**
2.  Calculate the sent IPv4 packet checksum
3.
4.  buffer: Pointer to the receive buffer, pointing to the IPv4 packet header
5.  Length: IPv4 header length
6.  **/
7.  short checksum0(unsigned short *buffer,int length)
8.  {
9.      unsigned long checksum = 0;
10.     while(length > 1)
11.     {
12.         checksum += *buffer++;
13.         length -= sizeof(unsigned short);
14.     }
15.     if(length)
```

```
16.     {
17.         checksum += *(unsigned char *)buffer;
18.     }
19.     checksum = (checksum>>16) + (checksum & 0xffff);
20.     checksum += (checksum>>16);
21.     return (unsigned short)(~checksum);
22. }
```

b. 校验和检验

```
1.  unsigned short checksum = checksum0((unsigned short *)pBuffer,20);
2.  if(checksum != 0)  //check Header checksum
3.  {
4.      ip_DiscardPkt(pBuffer,STUD_IP_TEST_CHECKSUM_ERROR);
5.      return 1;
6.  }
```

② 具体数据

编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Wed Nov 14 14:25:57.571 2018	10.0.255.243	10.0.255.241	IP	Version 4, ...	2.1 发送IP包
2	Wed Nov 14 14:25:59.214 2018	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Wed Nov 14 14:26:01.207 2018	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Wed Nov 14 14:26:03.210 2018	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Wed Nov 14 14:26:05.323 2018	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Wed Nov 14 14:26:07.215 2018	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.6 头部长度错误的IP包
7	Wed Nov 14 14:26:09.218 2018	10.0.0.1	192.167.173.8	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

```
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 0x2225]
Source: 10.0.0.1
Destination : 10.0.0.3

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
```

如上图，错误首部校验和为 0x00BC。

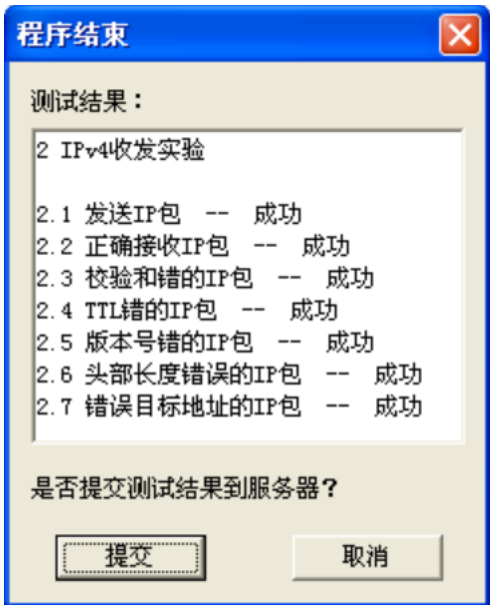
5 实验结果

a. 测试样例如下图：

```
begin test!, testItem = 1  testcase = 0
accept len = 32 packet
accept len = 166 packet
send a message to main ui, len = 53  type = 2  subtype = 1
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
begin test!, testItem = 1  testcase = 1
accept len = 32 packet
accept len = 166 packet
accept len = 38 packet
send a message to main ui, len = 36  type = 2  subtype = 0
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
begin test!, testItem = 1  testcase = 2
accept len = 32 packet
accept len = 166 packet
accept len = 38 packet
send a message to main ui, len = 36  type = 2  subtype = 0
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
begin test!, testItem = 1  testcase = 3
accept len = 32 packet
accept len = 166 packet
accept len = 38 packet
send a message to main ui, len = 36  type = 2  subtype = 0
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
```

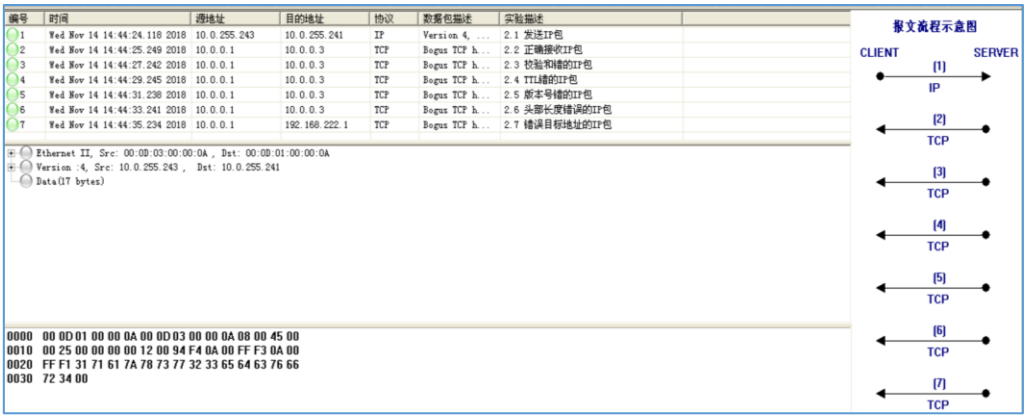
```
begin test!, testItem = 1  testcase = 4
accept len = 32 packet
accept len = 166 packet
accept len = 38 packet
send a message to main ui, len = 36  type = 2  subtype = 0
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
begin test!, testItem = 1  testcase = 5
accept len = 32 packet
accept len = 166 packet
accept len = 38 packet
send a message to main ui, len = 36  type = 2  subtype = 0
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
begin test!, testItem = 1  testcase = 6
accept len = 32 packet
accept len = 166 packet
accept len = 38 packet
send a message to main ui, len = 36  type = 2  subtype = 0
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
Test over!
```

b. 测试结果如下图：



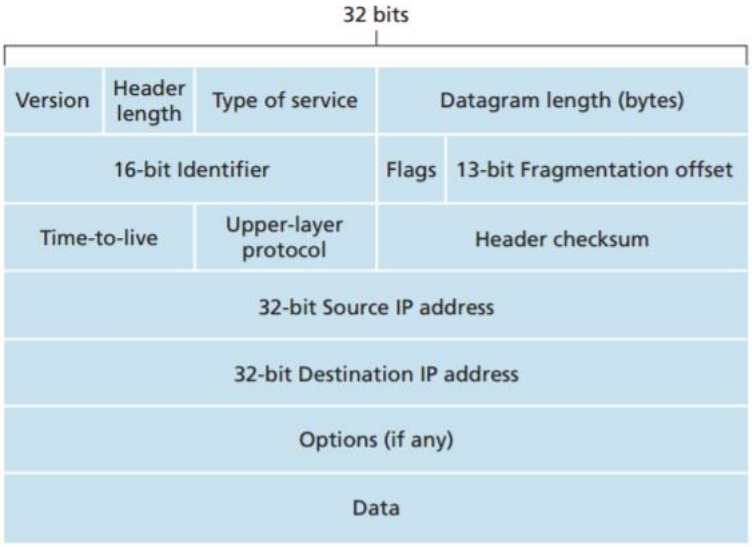
学期	序号	学号	姓名	院系	班级	实验名称	实验日期	实验结果	总成绩	程序	报告
2018年秋季	1	1160300909	张志路	计算机科学与技术	1603106	IPv4转发实验	2018-11-08		10		
2018年秋季	2	1160300909	张志路	计算机科学与技术	1603106	IPv4收发实验	2018-11-08		10		

c. 报文分析界面如下图：



6 问题讨论

6.1 IP 头部分组格式



7 心得体会

通过本次实验，我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，我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 3-2：IPv4 分组转发实验

1 实验目的

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。

2 实验内容

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为（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。。

3 实验环境

① 工具：接入到 Netriver 网络实验系统服务器的主机

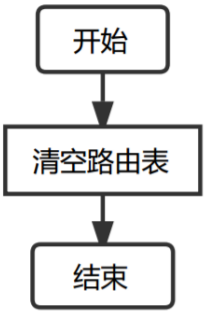
② 操作系统：WindowsXP 虚拟机

③ 开发语言：C 语言

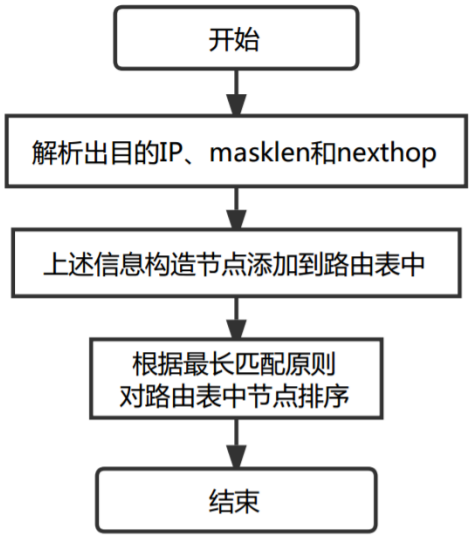
4 实验过程

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。

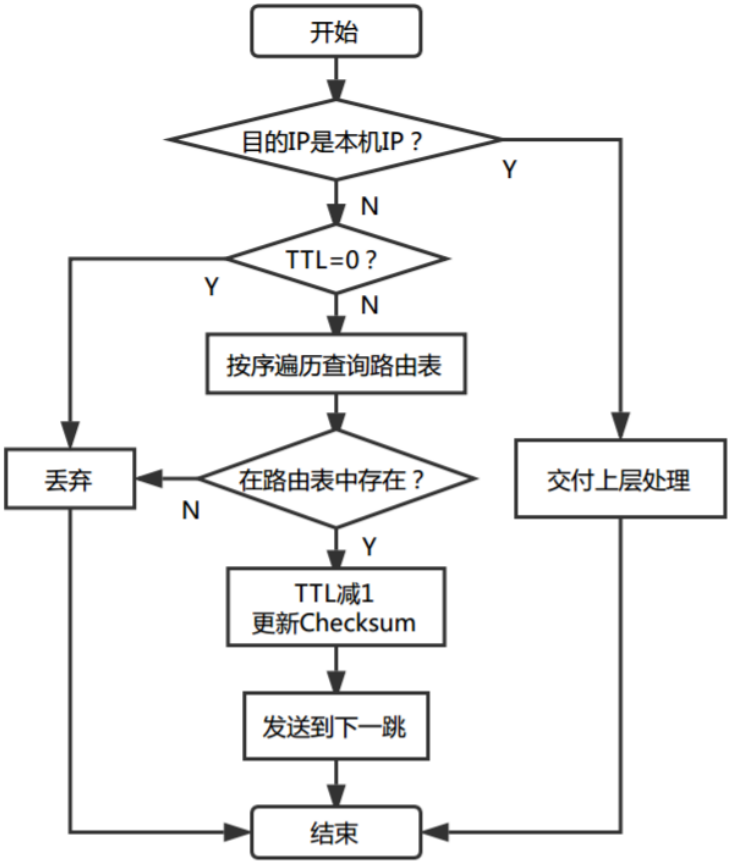
4.1 初始化函数的程序流程图



4.2 路由增加函数的程序流程图



4.3 路由转发函数的程序流程图



4.4 数据结构的说明

新建一个结构体 RNode 作为路由表节点，其中包括目的 IP 地址、掩码长度和下一跳。

利用 C++中的数据结构 vector 存储路由表。

新建数据结构如下。

```
/**set Route Node **/
struct RNode
{
    int dest;    //Destination network address
    int masklen; //Mask length
    int nexthop; //Next hop
    RNode(int d=0, int m=0, int n=0):
        dest(d), masklen(m), nexthop(n){}
};

/** vector routeTable **/
vector<RNode> routeTable;
```

4.5 提高转发效率

查路由表是路由转发的瓶颈，也就是说，查表的速度决定着路由转发的效率。

下面讨论提高转发效率的几种方法。

① 树形结构

路由表数据结构的设计是非常重要的，会极大地影响路由表的查找速度，进而影响路由器的分组转发性能。

使用树状结构存储路由表，如红黑树，可以将查询时间从 O(n)降低到 O(logn)。链表结构是最简单的，但效率比较低；树型结构的查找效率会提高很多，但组织和维护有些复杂。

② 路由聚合

路由聚合是让路由选择协议能够用一个地址通告众多网络，旨在缩小路由器中路由选择表的规模，以节省内存，并缩短 IP 对路由选择表进行分析以找出前往远程网络的路径所需的时间。

路由聚合技术可以大大的缩短路由表项，从而缩短寻找下一跳的时间。

③ 多线程

当分组到达路由器时，将其分配到不同线程，各线程同时查找路由表，以减少延迟。

④ Cache 方法

借鉴 Cache 思想，设置路由表缓存，利用 LRU 算法，缓存最近使用的路由表项，从而提高查表速度，进而提高转发效率。

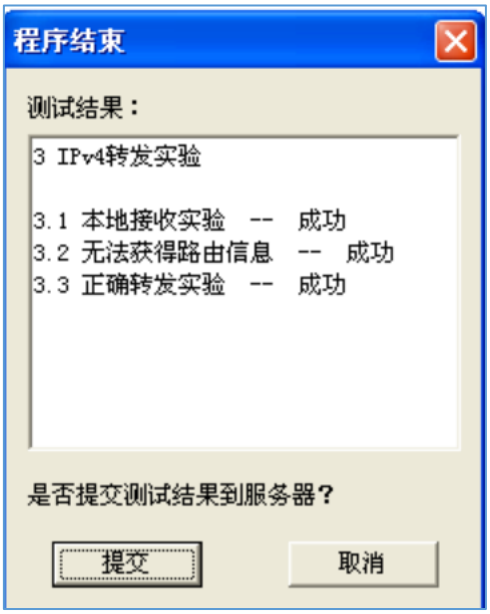
5 实验结果

a. 测试样例如下图：

```
begin test!, testItem = 2  testcase = 0
accept len = 32 packet
accept len = 244 packet
accept len = 41 packet
accept len = 55 packet
send a message to main ui, len = 53  type = 2  subtype = 0
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
begin test!, testItem = 2  testcase = 1
accept len = 32 packet
accept len = 244 packet
accept len = 41 packet
accept len = 38 packet
send a message to main ui, len = 36  type = 2  subtype = 0
```

```
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
begin test!, testItem = 2  testcase = 2
accept len = 32 packet
accept len = 244 packet
accept len = 41 packet
accept len = 55 packet
send a message to main ui, len = 53  type = 2  subtype = 0
send a message to main ui, len = 53  type = 2  subtype = 1
accept len = 6 packet
result = 0
send a message to main ui, len = 6  type = 1  subtype = 7
Test over!
```

b. 测试结果如下图：



学期	序号	学号	姓名	院系	班级	实验名称	实验日期	实验结果	总成绩	程序	报告
2018年秋季	1	1160300909	张志路	计算机科学与技术	1603106	IPv4转发实验	2018-11-08		10		
2018年秋季	2	1160300909	张志路	计算机科学与技术	1603106	IPv4收发实验	2018-11-08		10		

c. 报文分析界面如下图：

6 问题讨论

6.1 分组转发算法

简单总结一下 IP 路由分组转发算法：

(1) 从数据报的首部提取目的主机的 IP 地址 D，得出目的网络地址为 N。

(2) 若 N 就是与此路由器直接相连的某个网络地址，则进行直接交付，不需要再经过其他的路由器，直接把数据报交付给目的主机（这里包括把目的主机地址 D 转换为具体的硬件地址，把数据报封装为 MAC 帧，再发送此帧）；否则就要执行(3)进行间接交付。

(3) 若路由表中有目的地址为 D 的特定主机路由，则把数据报传送给路由表中所指明的下一跳路由器，否则执行(4)。

(4) 若路由表中有到达网络 N 的路由，则把数据报传送给路由表中所指明的下一跳路由器，否则执行(5)。

(5) 若路由表中有一个默认路由，则把数据报传送给路由表中所指明的下一跳路由器，否则执行(6)。

(6) 报告转发分组出错。

7 心得体会

通过本次实验，我设计模拟了实现路由器中的 IPv4 协议，在原有 IPv4 分组收发实验的基础上，实现了 IPv4 分组的转发功能。同时也了解了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的。

本实验中我也初步接触了路由表这一重要的数据结构，认识到路由器是如何根据路由表对分组进行转发的。

纸上得来终觉浅，绝知此事要躬行。通过实验也使得我对该部分的理论内容加深了理解，一些原来理解有偏差的点得到纠正，真正起到了巩固与提升的作用，真正做到了学以致用。

附录一：IP 分组收发实验代码

```
/*
* 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

/**
Calculate the sent IPv4 packet checksum

buffer: Pointer to the receive buffer, pointing to the IPv4 packet header
Length: IPv4 header length
**/
short checksum0(unsigned short *buffer,int length)
{
    unsigned long checksum = 0;
    while(length > 1)
    {
        checksum += *buffer++;
        length -= sizeof(unsigned short);
    }
    if(length)
    {
        checksum += *(unsigned char *)buffer;
    }
    checksum = (checksum>>16) + (checksum & 0xffff);
    checksum += (checksum>>16);
    return (unsigned short)(~checksum);
}

/**
receive packet from MAC
```

```
42.
43. pBuffer: pointer to IPv4 packet group header (char 8 bit/1 Byte)
44. length: IPv4 group length
45. **/
46. int stud_ip_recv(char *pBuffer,unsigned short length)
47. {
48.     if((pBuffer[0] & 0xf0) != 0x40) //check Version
49.     {
50.         ip_DiscardPkt(pBuffer,STUD_IP_TEST_VERSION_ERROR);
51.         return 1;
52.     }
53.     if((pBuffer[0] & 0x0f) != 0x05) //check IP Head length (4 Bytes/unit)
54.     {
55.         ip_DiscardPkt(pBuffer,STUD_IP_TEST_HEADLEN_ERROR);
56.         return 1;
57.     }
58.     if(pBuffer[8] == 0x00)  //check TTL
59.     {
60.         ip_DiscardPkt(pBuffer,STUD_IP_TEST_TTL_ERROR);
61.         return 1;
62.     }
63.     unsigned short checksum = checksum0((unsigned short *)pBuffer,20);
64.     if(checksum != 0)  //check Header checksum
65.     {
66.         ip_DiscardPkt(pBuffer,STUD_IP_TEST_CHECKSUM_ERROR);
67.         return 1;
68.     }
69.     //check destination ip
70.     unsigned int address = getIpv4Address(); //get local host ip
71.     char *tempAddress = pBuffer+16; //dest ip address in ipv4
72.     unsigned int *intAddress = (unsigned int *)tempAddress;
73.     if(address != ntohl(*intAddress)) //ntohl:network to host long
74.     {
75.         ip_DiscardPkt(pBuffer,STUD_IP_TEST_DESTINATION_ERROR);
76.         return 1;
77.     }
78.     ip_SendtoUp(pBuffer,length); //pay for up level
79.     return 0;
80. }
81.
82. /**
83. send packet to lower level
84.
```

```
pBuffer: Pointer to the send buffer, pointing to the IPv4 upper layer protocol data header
Len: IPv4 upper layer protocol data length
srcAddr: source IPv4 address
dstAddr: destination IPv4 address
Protocol: IPv4 upper layer protocol number
Ttl: Time To Live
**/
int stud_ip_Upsend(char *pBuffer,unsigned short len,unsigned int srcAddr,
                   unsigned int dstAddr,byte protocol,byte ttl)
{
    byte *datagram = new byte[20+len]; //acllocate memory (Byte/unit)

    datagram[0] = 0x45; //version=ipv4 IHL=5 (4Byte/unit)
    datagram[1] = 0x00; //Type of service = 0x00

     byte *total_length = datagram+2; //Total Length
     unsigned short int *length = (unsigned short int *)total_length;
     *length = htons(20+len); //host to network(short)

     datagram[4] = 0x00; //identification random num
     datagram[5] = 0x00; //identification random num

     datagram[6] = 0x00; //flag(3bits) and offset(13 offsets)
     datagram[7] = 0x00; //flag(3bits) and offset(13 offsets)

     datagram[8] = ttl; //ttl

     datagram[9] = protocol;  //up level protocol

     datagram[10] = 0x00; //checksum set 0x0000 initial
     datagram[11] = 0x00; //checksum set 0x0000 initial

     byte *datagram_srcAddr = datagram+12;  //src ip address
     unsigned int *srcAddrTemp = (unsigned int *)datagram_srcAddr;
     *srcAddrTemp = ntohl(srcAddr);

     byte *datagram_dstAddr = datagram+16; //dest ip address
     unsigned int *dstAddrTemp = (unsigned int *)datagram_dstAddr;
     *dstAddrTemp = ntohl(dstAddr);

     byte *datagram_cksum = datagram+10; //checksum
     short int *headerChecksum = (short int *)datagram_cksum;
     *headerChecksum = checksum0((unsigned short *)datagram,20);
```

```
128.
129.     for(int i=0;i<len;i++)  //add header
130.     {
131.         datagram[i+20] = pBuffer[i];
132.     }
133.
134.     ip_SendtoLower(datagram,20+len); //send to lower level
135.     return 0;
136. }
```

附录二：IP 分组转发实验代码

```
1.  /*
2.  * THIS FILE IS FOR IP FORWARD TEST
3.  */
4.  #include "sysInclude.h"
5.  #include <vector>
6.  #include <algorithm>
7.  using std::vector;
8.
9.  // system support
10. extern void fwd_LocalRcv(char *pBuffer, int length);
11.
12. extern void fwd_SendtoLower(char *pBuffer, int length, unsigned int nexthop)
    ;
13.
14. extern void fwd_DiscardPkt(char *pBuffer, int type);
15.
16. extern unsigned int getIpv4Address( );
17.
18. // implemented by students
19.
20. /**set Route Node **/
21. struct RNode
22. {
23.     int dest;    //Destination network address
24.     int masklen; //Mask length
25.     int nexthop; //Next hop
26.     RNode(int d=0, int m=0, int n=0):
27.         dest(d), masklen(m), nexthop(n){}
28. };
29.
30. /** vector routeTable **/
```

```
vector<RNode> routeTable;

/** Initialize the routing table **/
void stud_Route_Init()
{
    routeTable.clear();
    return;
}

/** sort by hostip or mask length **/
bool cmp(const RNode & a, const RNode & b)
{
    if(htonl(a.dest) > htonl(b.dest))
    {
        return true;
    }
    else if(htonl(a.dest) == htonl(b.dest)) //According to the longest match

    {
        return htonl(a.masklen) > htonl(b.masklen);
    }
    else
    {
        return false;
    }
}

/** Add routing information to the routing table **/
void stud_route_add(stud_route_msg *proute)
{
    int dest;
    routeTable.push_back(RNode(ntohl(proute->dest), ntohl(proute->masklen),
ntohl(proute->nexthop)));
    sort(routeTable.begin(), routeTable.end(), cmp);
    return;
}

/** Calculate the sent IPv4 packet checksum **/
short checksum0(unsigned short *buffer,int length)
{
    unsigned long checksum = 0;
    while(length > 1)
    {
        checksum += *buffer++;
```

```
73.         length -= sizeof(unsigned short);
74.     }
75.     if(length)
76.     {
77.         checksum += *(unsigned char *)buffer;
78.     }
79.     checksum = (checksum>>16) + (checksum & 0xffff);
80.     checksum += (checksum>>16);
81.     return (unsigned short)(~checksum);
82. }
83.
84. /** Handling received IP packets **/
85. int stud_fwd_deal(char *pBuffer, int length)
86. {
87.     int version = pBuffer[0] >> 4; //get version
88.     int ihl = pBuffer[0] & 0xf; //get IP packet header length
89.     int ttl = (int)pBuffer[8]; //get TTL
90.
91.     int dstIP = ntohl(*(unsigned int*)(pBuffer + 16)); //get Destination IP
    address
92.     if(dstIP == getIpv4Address()) //IP packet received by this machine
93.     {
94.         fwd_LocalRcv(pBuffer, length);
95.         return 0;
96.     }
97.
98.     if(ttl <= 0)
99.     {
100.         fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR);
101.         return 1;
102.     }
103.
104.     //Look up the routing table to get the next hop and calculate checksum

105.     for(vector<RNode>::iterator ii = routeTable.begin(); ii != routeTable.e
    nd(); ii++)
106.     {
107.         if(ii->dest == dstIP)
108.         {
109.             char *buffer = new char[length];
110.             memcpy(buffer, pBuffer, length);
111.             buffer[8]--; //TTL--
112.             buffer[10] = 0;
113.             buffer[11] = 0; //checksum=0
```

```
114.             unsigned short int localCheckSum = checksum0((unsigned short *)
    buffer,20);
115.             memcpy(buffer+10, &localCheckSum, sizeof(short unsigned int));

116.             fwd_SendtoLower(buffer, length, ii->nexthop);
117.             return 0;
118.         }
119.     }
120.     fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE); //Look for failure

121.     return 1;
122. }
```