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	IPv4 分组收发与转发实验					
姓名	田一间		院系	计算机学院		
班级	1636101		学号	1160300617		
任课教师	李全龙		指导教师	李全龙		
实验地点	格物楼 213		实验时间	2018 年 11 月 10 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

实验目的：

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。初步接触互联网协议栈的结构和计算机网络实验系统。

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。

实验内容：

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

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。

3) 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）。

4) 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。

（1） IPv4分组发送

主要为构造一个分组：申请空间，写入版本号，IHL，总长度，TTL，协议，源地址与目的地址，计算校验和并写入。要注意两个字节及以上的需要转换为网络字节序。

```
00065  int stud_ip_Upsend(char *pBuffer,unsigned short len,unsigned int srcAddr,
00066                             unsigned int dstAddr,byte protocol,byte ttl)
00067  {
00068        char *buffer = (char *)malloc((len + 20) * sizeof(char));    //申请空间
00069        memset(buffer, 0, len+20);  //初始化空间为零
00070        buffer[0] = 0x45;           //版本号以及IHL
00071        unsigned short totalLength = htons(len+20);     //总长度 转为网络字节序
00072        memcpy(buffer+2, &totalLength, sizeof(unsigned short));  //总长度赋值
00073        buffer[8] = ttl;            //TTL
00074        buffer[9] = protocol;       //协议
00075        unsigned int srcAddress = htonl(srcAddr);       //源地址 转为网络字节序
00076        memcpy(buffer+12, &srcAddress, sizeof(unsigned int));  //源地址赋值
00077        unsigned int dstAddress = htonl(dstAddr);       //目的地址 转为网络字节序
00078        memcpy(buffer+16, &dstAddress, sizeof(unsigned int));  //目的地址赋值
00079
00080        unsigned int checkSum = 0;          //校验和计算
00081        for (int i = 0; i < 10; i++)
00082        {
00083              checkSum += (unsigned short)(buffer[i*2]<<8 | buffer[i*2+1]);
00084              checkSum %= 0xFFFF;
00085        }
00086        checkSum = htons(~(unsigned short)checkSum);    //取反 转为网络字节序
00087        memcpy(buffer+10, &checkSum, sizeof(unsigned short));   //校验和赋值
00088        memcpy(buffer + 20, pBuffer, len);              //数据赋值
00089        ip_SendtoLower(buffer,len+20);                  //发送
00090        return 0;
00091  }
```

（2） IPv4分组接收

按照IPv4分组头部的格式，取出需要检查的字段：版本号，IHL，TTL，目的地址，校验和。重新计算校验和并与校验和字段比较。

注意目的地址需要转换为主机字节序。

```
00017  int stud_ip_recv(char *pBuffer,unsigned short length)
00018  {
00019        int version = pBuffer[0]/16;    //IP版本号
00020        if (version != 4)    //检查IP
00021        {
00022              printf("Wrong version is %d\n", version);
00023              ip_DiscardPkt(pBuffer,STUD_IP_TEST_VERSION_ERROR);
00024              return 1;
00025        }
00026        int IHL = pBuffer[0]%16;     //IP Head length
00027        if (IHL< 5)         //检查IHL
00028        {
00029              printf("Wrong IHL is %d\n", IHL);
00030              ip_DiscardPkt(pBuffer,STUD_IP_TEST_HEADLEN_ERROR);
00031              return 1;
00032        }
00033
00034        int ttl = (int)pBuffer[8];                //TTL
00035        if (ttl == 0)       //检查TTL
00036        {
00037              printf("Wrong ttl is %d\n", ttl);
00038              ip_DiscardPkt(pBuffer,STUD_IP_TEST_TTL_ERROR);
00039              return 1;
00040        }
00041        int dstAddr = ntohl(*(unsigned int*)(pBuffer + 16));    //目的地址 转为主机字节序
00042        if (dstAddr != getIpv4Address() && dstAddr != 0xffff)
00043        {
00044              printf("Wrong dstAddr is 0x%x\n", dstAddr);
00045              ip_DiscardPkt(pBuffer,STUD_IP_TEST_DESTINATION_ERROR);
00046              return 1;
00047        }
00048        unsigned int sum = 0;   //计算校验和
00049        for (int i = 0; i < IHL * 2; i++)
00050        {
00051              if (i != 5)       //跳过校验和字段
00052              {
00053                    sum += pBuffer[i*2]<<8 | pBuffer[i*2+1];
00054                    sum %= 0xffff;
00055              }
00056
00057        }
00058        unsigned short calcCheckSum = ~(unsigned short)sum;
00059        unsigned short checkSum =  ntohs(*(unsigned short*)(pBuffer+10));       //取出校验和字段
00060        if (calcCheckSum != checkSum)       //计算的校验和与校验和字段比较
00061        {
00062              printf("Wrong checksum is 0x%x, should be 0x%x\n", checkSum, calcCheckSum);
00063              ip_DiscardPkt(pBuffer,STUD_IP_TEST_CHECKSUM_ERROR);
00064              return 1;
00065        }
00066        ip_SendtoUp(pBuffer,length);  //发送数据包给上层
00067        return 0;
00068  }
```

（3） IPv4分组转发

采用了容器map作为路由表的数据结构，能有效提升路由表的查找性能，进而提升路由器的分组转发性能。

初始化时将map清空即可。

路由表添加函数则将结构体 stud_route_msg 中的信息按照键值对添加进map即可。

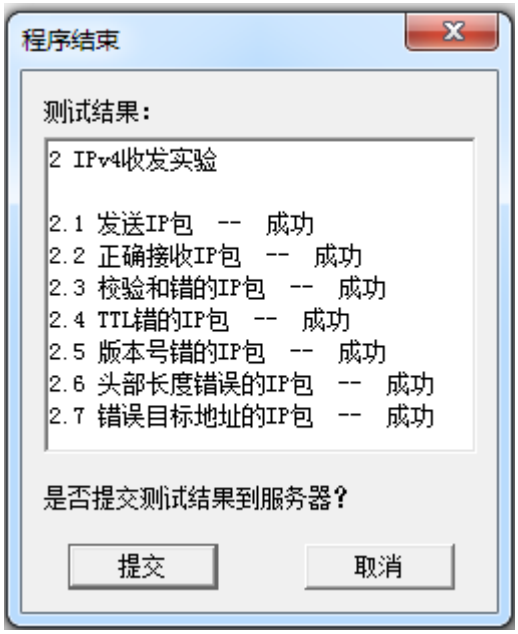
转发处理函数根据目的地址、路由表查询、校验和计算等情况决定将数据包丢弃、接收或者转发。

```
00037  int stud_fwd_deal(char *pBuffer, int length)
00038  {
00039        int ttl = (int)pBuffer[8];                //获得 TTL
00040        int IHL = pBuffer[0]%16;                  //获得 IP Head length
00041        int dstAddr = ntohl(*(unsigned int*)(pBuffer + 16));  //获得目的地址
00042        if (dstAddr == getIpv4Address())    //判断是否为本机接受分组
00043        {
00044              fwd_LocalRcv(pBuffer, length);
00045              return 0;
00046        }
00047        if (ttl <=0 )                        //判断TTL是否错误
00048        {
00049              fwd_DiscardPkt(pBuffer,STUD_FORWARD_TEST_TTLERROR);
00050              return 1;
00051        }
00052        printf("Want to find %d \n", dstAddr);
00053        map<unsigned int, unsigned int>::iterator iter; //查找路由表
00054        iter = routeTable.find(dstAddr);
00055        if (iter != routeTable.end()) //
00056        {
00057              printf("Find dstAddr! %d\n", iter->second);
00058              //TTL减1 重新赋值
00059              //printf("ttl before %d \n", ttl);
00060              pBuffer[8] = (unsigned char)(ttl - 1);
00061              //printf("ttl after %d \n", (int)pBuffer[8]);
00062              //重新计算校验和
00063              unsigned int sum = 0;   //计算校验和
00064              for (int i = 0; i < IHL * 2; i++)
00065              {
00066                    if (i != 5)       //跳过校验和字段
00067                    {
00068                          sum += pBuffer[i*2]<<8 | pBuffer[i*2+1];
00069                          sum %= 0xffff;
00070                    }
00071              }
00072              unsigned short checkSum = htons(~(unsigned short)sum);
00073              memcpy(pBuffer+10, &checkSum, sizeof(unsigned short));   //校验和赋值
00074              fwd_SendtoLower(pBuffer, length, iter->second);
00075              return 0;
00076        }
00077        else        //找不到则丢弃该分组
00078        {
00079              printf("Not find, discard Packet!\n");
00080              fwd_DiscardPkt(pBuffer,STUD_FORWARD_TEST_NOROUTE);
00081              return 1;
00082        }
00083        return 0;
00084  }
```

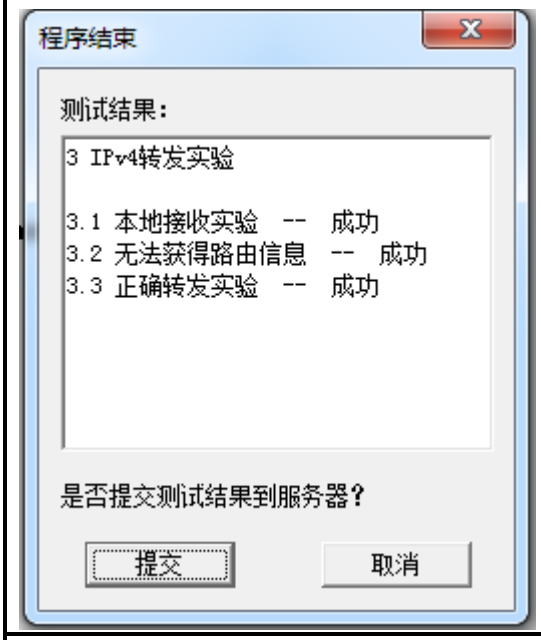
实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

IPv4分组收发实验：



IPv4分组转发实验：

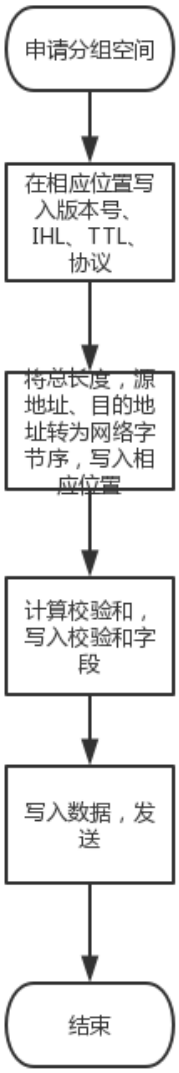


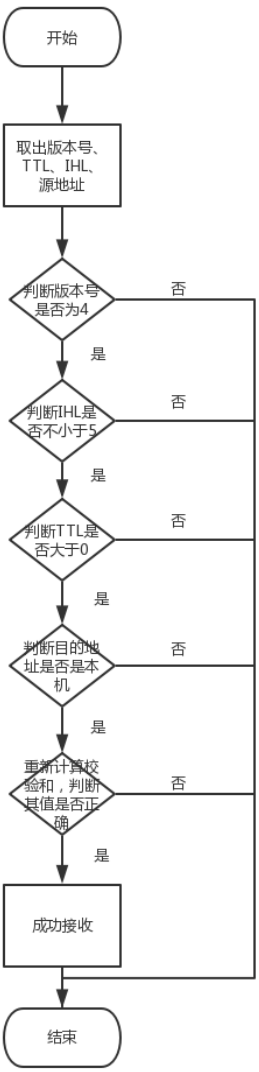
问题讨论：

对实验过程中的思考问题进行讨论或回答。

1) 要求给出发送和接收函数的实现程序流程图
2) 要求给出版本号（Version）、头部长度（IP Head length）、生存时间（Time to live）以及头校验和（Header checksum）字段的错误检测原理，并根据实验具体情况给出错误的具体数据
3) 要求给出路由表初始化、路由增加、路由转发三个函数的实现流程图
4) 要求给出所新建数据结构的说明；
5) 请分析在存在大量分组的情况下如何提高转发效率，如果代码中有相关功能实现，请给出具体原理说明。

1) 发送函数流程图　　接收函数流程图





2) 每次测试时错误值可能不一样，控制台打印错误信息，某一次结果如下：

版本号错误检测原理：IPv4协议数据包中版本号应为 4，错误值：1

头部长度检测原理：IHL不小于5，错误值：3

生存时间检测原理：TTL应该大于0，错误值：0

目的地址检测原理：判断其值是否与getIpv4Address()相等或者是0xffffffff。

```
send a message to main ui, len = 36  type = 2  subtype = 0
Wrong dstAddr is 0xc0a55933, localAddr is 0xa000004
```

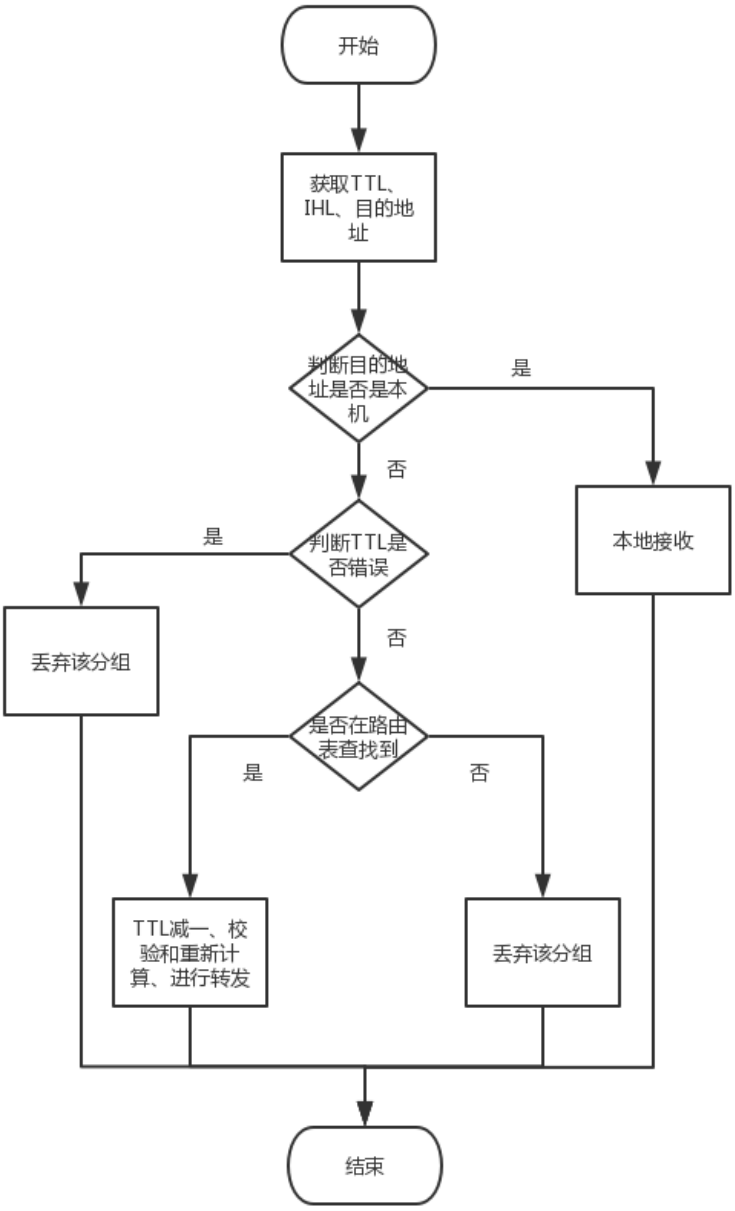
校验和检测原理：

发送方：首部以16位划分，补码求和，最高位进位加到最低位，取反码

接收方：与发送法相同算法计算，得到的值全零则无错。

```
Wrong checksum is 0x29a, should be 0x66e0
```

3) 路由转发函数流程图



4) 数据结构说明

Map 是STL的一个关联容器，它提供一对一的数据处理能力，其内部自建一颗红黑树，具有数据自动排序的功能。

在查找性能上，其根据key值快速查找记录，查找的复杂度基本是Log(N)。

5) 转发效率的提高

路由器转发效率主要受到路由表查找性能的影响，在大量分组的情况下，如何实现快速的查找是效率提高的关键。

本实验采用STL中的容器map，查找复杂度为Log(N)，不算太理想。

再需提高的情况下，可以考虑采用哈希表，哈希查找良好情况下复杂度为O(1)。

心得体会：
结合实验过程和结果给出实验的体会和收获。 通过本次实验，自己对于网络层协议栈的基本原理有了更深的了解，对于IPv4分组的格式有了更为清晰的认识，掌握每个字段的意义与其错误检测方法。 不仅如此，自己也熟悉了分组的接收、发送和转发流程，包括路由表的构建与使用，对于网络层协议有了更为透彻的理解。 实验过程中，遇到的麻烦就是校验和字段检验那里，涉及到检测原理、反码计算以及网络字节序转换等，实现时费了较大的功夫调试。