哈尔滨工业大学

<<计算机网络>>

实验报告

(2018 年度春季学期**)**

姓名:	
学号:	
学院:	计算机科学与技术学院
教师:	

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。

二、实验内容

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。

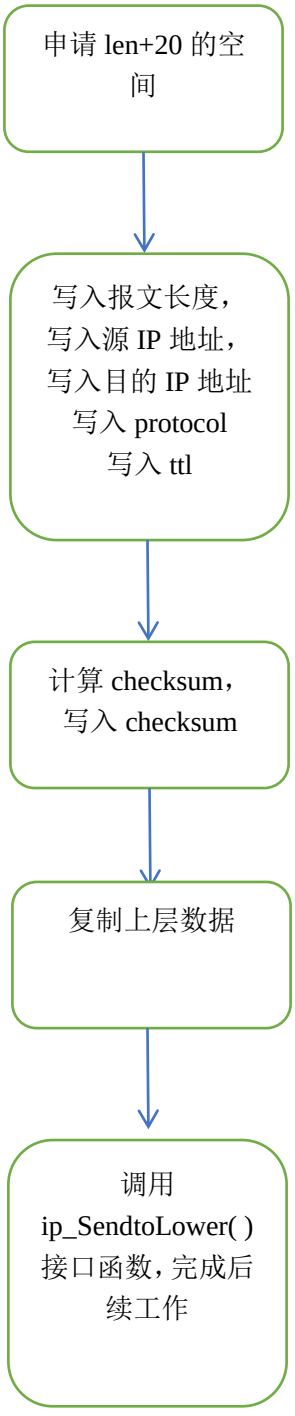
实现 IPv4 分组的封装发送

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。

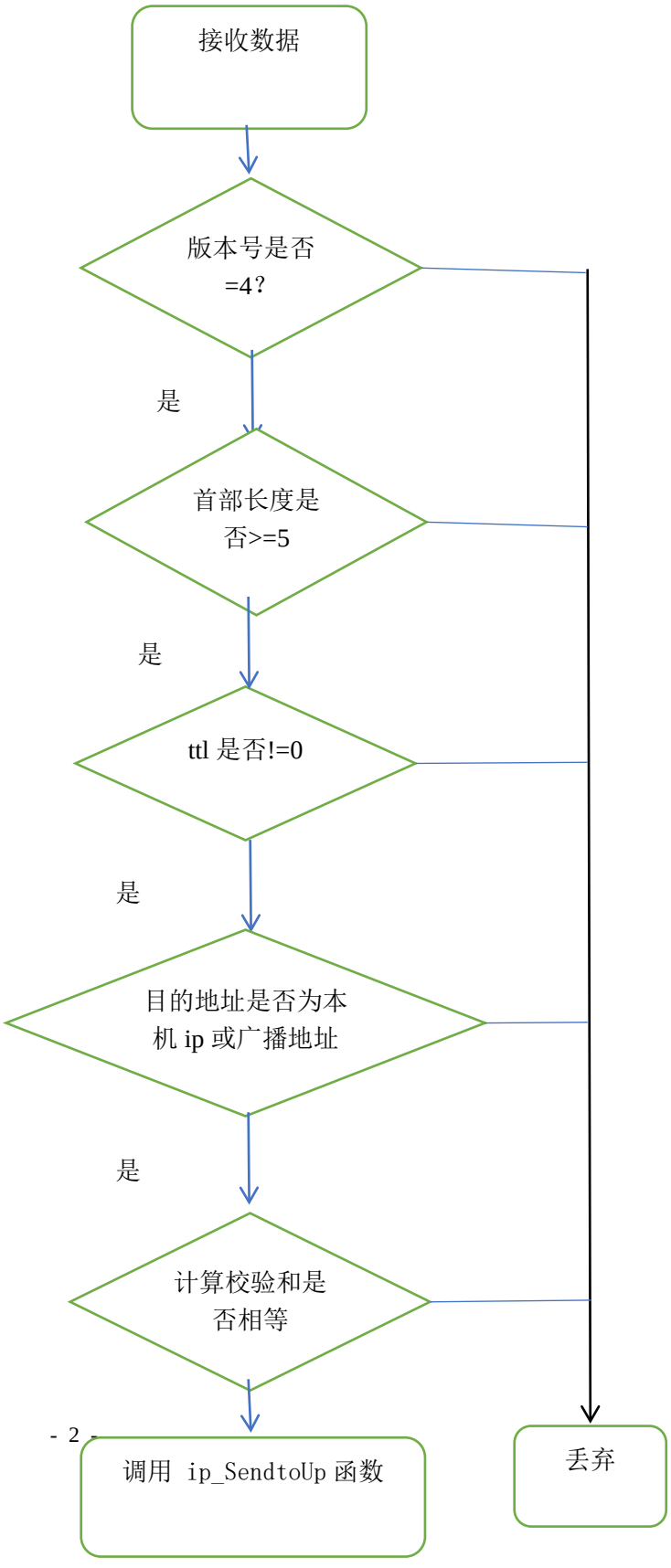
三、实验过程及结果

1. 发送和接收函数流程图

发送函数流程图：



接收函数流程图



2. 具体流程

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

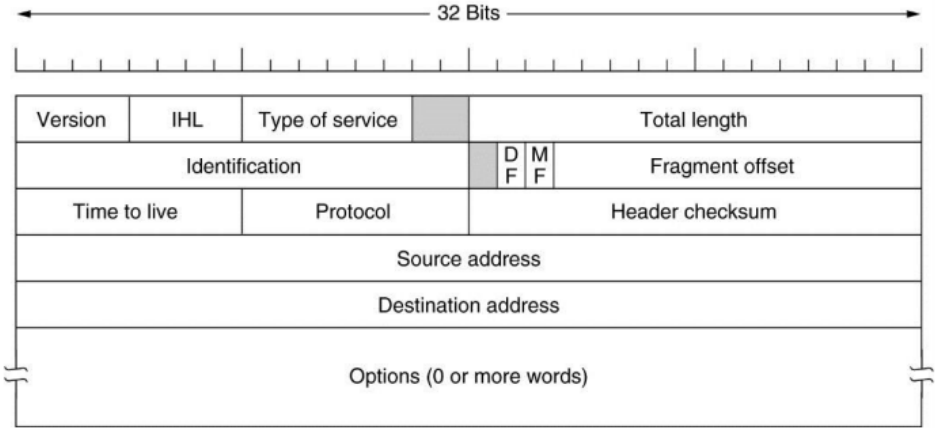
① 根据所传参数（如数据大小），来确定分配的存储空间的大小并 申请分组的存储空间。

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

3. 自定义数据结构

IP 报文头



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
}
```

4.检错原理

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;
  }
}
```

四、实验心得

通过本次实验，我对 IP 协议有了更加深刻的理解，对 IP 数据报头部信息各项字段也更加清楚其含义与功能。对整个接收过程有了切实的体验，也越来越体会到网络协议的必要与其所发挥的极大的作用。更细的一点，理清并实现了 checksum 的过程。越来越爱上网络，感受到其趣味十足。

五、源代码

```
#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

struct Ipv4
{
    char version_ihl;
    char type_of_service;
    short total_length;
    short identification;
    short fragment_offset;
    char time_to_live;
    char protocol;
    short header_checksum;
    unsigned int source_address;
    unsigned int destination_address;
    Ipv4() {
        memset(this,0,sizeof(Ipv4));
    }
    Ipv4(unsigned int len,unsigned int srcAddr,unsigned int dstAddr,
```

```
        byte _protocol,byte ttl) {
        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));
        int sum = 0;
        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) {
           sum = (sum & 0xffff) + ((sum >> 16) & 0xffff);
        }
        unsigned short int ssum = sum;
        header_checksum = htons(~ssum);
    }
};

int stud_ip_recv(char *pBuffer,unsigned short length)
{
    Ipv4 *ipv4 = new Ipv4();
    *ipv4 = *(Ipv4*)pBuffer;
```

```
int version = 0xf & ((ipv4->version_ihl)>> 4);
if(version != 4)    {
   ip_DiscardPkt(pBuffer,STUD_IP_TEST_VERSION_ERROR);
   return 1;
}
int ihl = 0xf & ipv4->version_ihl;
if(ihl < 5) {
   ip_DiscardPkt(pBuffer,STUD_IP_TEST_HEADLEN_ERROR);
   return 1;
}
int ttl = (int)ipv4->time_to_live;
if(ttl == 0) {
   ip_DiscardPkt(pBuffer,STUD_IP_TEST_TTL_ERROR);
   return 1;
}
int destination_address = ntohl(ipv4->destination_address);
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++) {
   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;
    }
    ip_SendtoUp(pBuffer,length);
    return 0;
}

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];
    memset(pack_to_sent,0,len+20);
    *((Ipv4*)pack_to_sent) = Ipv4(len,srcAddr,dstAddr,protocol,ttl);
    memcpy(pack_to_sent+20,pBuffer,len);
    ip_SendtoLower(pack_to_sent,len+20);
    delete[] pack_to_sent;

    return 0;
}
```