北 京 大 学

信息科学技术学院

本科生实验报告

实验课程：计算机网络概论

实验名称：IPv4 协议收发实验

学生姓名：季凯航

学　　号：1400012727

提交时间：2016 年 12 月 5 日

目录

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。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、实验要求

根据计算机网络实验系统所　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；

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

三、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　供的参数，封装 IPv4 分组，调用系统　供的发送接口函数将分组发送出去。

四、实验过程

1. 接收处理

对于接收到的 IPv4 分组，首先完成以下检查：

- 版本号是否正确
- 报头长度是否合法
- 该报文是否超出生存时间
- 是否为发送给本主机的报文
- 报头校验是否正确

如果通过所有检查，则该报文是正确报文，通过 ip_SendtoUp 接口函数将其交给上层；否则调用 ip_DiscardPkt 抛弃报文，并由 type 参数给出抛弃的原因。

2. 封装发送

首先计算出总报文的大小，分配存储空间。再按照 IPv4 协议标准填写头部各字段。其中须填写的字段包括版本号、总长度、最长生存时间、协议、头校验和、源地址和目的地址，分段、偏移等字段都以 0 填充。

完成 IPv4 分组的封装后，调用 ip_SendtoLower 接口函数将分组发送到网络中。

五、附录——代码

```
#include "sysInclude.h"
#include <cstdlib>
#include <cstring>

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();

#ifndef byte
#define byte unsigned char
#endif


/* Handle packets received */
int stud_ip_recv(char* pBuffer, unsigned short length) {
     const unsigned int Version = (unsigned)pBuffer[0] >> 4;
     const unsigned int IHL = (unsigned)pBuffer[0] & 0xF;
     const unsigned int TTL = (unsigned)pBuffer[8];
     const unsigned int DstAddr = ntohl(*(unsigned int *)(&pBuffer[16]));
     unsigned int HeaderCheckSum = 0;
```

```
    if (Version != 4) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_VERSION_ERROR);
        return -1;
    }
    if (IHL < 5) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_HEADLEN_ERROR);
        return -11;
    }
    if (!TTL) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_TTL_ERROR);
        return -1;
    }

    if (DstAddr != getIpv4Address() && DstAddr != 0xFFFFFFFF) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_DESTINATION_ERROR);
        return -1;
    }

    for (int i = 0; i < 20; i += 2) {
        HeaderCheckSum += ((pBuffer[i] & 0xFF) << 8) + (pBuffer[i + 1] & 0xFF);
    }
    HeaderCheckSum += (HeaderCheckSum >> 16);

    if ((unsigned short)(~HeaderCheckSum)) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_CHECKSUM_ERROR);
        return -1;
    }

    ip_SendtoUp(pBuffer, length);

    return 0;
}

/* Send packets */
int stud_ip_Upsend(char* pBuffer, unsigned short len, unsigned int srcAddr,
                   unsigned int dstAddr, byte protocol, byte ttl) {
    const unsigned int Version = 4;
    const unsigned int IHL = 5;
    const unsigned int TotalLen = 4 * IHL + len;
    unsigned int HeaderCheckSum = 0;
    unsigned char* Buffer = (unsigned char*) malloc(TotalLen);

    memset(Buffer, 0, TotalLen);
```

```
    Buffer[0] = Version << 4 | IHL;
    Buffer[2] = TotalLen >> 8;
    Buffer[3] = TotalLen;
    Buffer[8] = ttl;
    Buffer[9] = protocol;
    Buffer[12] = srcAddr >> 24;
    Buffer[13] = srcAddr >> 16;
    Buffer[14] = srcAddr >> 8;
    Buffer[15] = srcAddr;
    Buffer[16] = dstAddr >> 24;
    Buffer[17] = dstAddr >> 16;
    Buffer[18] = dstAddr >> 8;
    Buffer[19] = dstAddr;
    memcpy(Buffer + 20, pBuffer, len);
    for (int i = 0; i < 20; i += 2) {
        HeaderCheckSum += ((Buffer[i] & 0xFF) << 8) + (Buffer[i + 1] & 0xFF);
    }
    HeaderCheckSum += HeaderCheckSum >> 16;
    HeaderCheckSum = ~HeaderCheckSum;
    Buffer[10] = (char)((unsigned short)HeaderCheckSum >> 8);
    Buffer[11] = (char)((unsigned short)HeaderCheckSum & 0xFF);

    ip_SendtoLower((char*)Buffer, TotalLen);

    return 0;
}
```