# NetRiver 2–IPv4 协议收发实验

张煌昭，1400017707，元培学院

**摘要—本次作业对 IPv4 协议分组的发送和接收进行了实现，在接收时检查分组是否正确，并于 NetRiver 平台进行提交和测试。本次报告使用 Overleaf $L^AT_EX$ 在线平台编写[1]，作业源码附于报告之后。**

## I. 介绍

本次作业，要求使用 C/C++ 语言，在 NetRiver 的 IPv4 层虚拟环境中，实现发送 IPv4 分组和接收并检查 IPv4 分组的功能。要求可以将上层内容封装为 IPv4 分组并发送，接收 IPv4 分组并检查其正确性，若正确则将头部剥去，内容交给上层，否则丢弃并报错。

IPv4 分组的结构详情见第 II节，IPv4 分组封装和发送的实现详情见第**??**节，选择重传协议的具体要求及实现详情见第**??**节。

## II. IPv4 分组结构

IPv4 分组的结构如图 1所示，分组由分组头部和可选内容两段组成。内容长度可变，最小长度为 0，具体的长度视上层内容而定。

IPv4 协议下，分组头部长度固定为 160 位/20 字节/5 个 32 位字，其中与本次作业相关的各个字段包括：1) 版本号（Version），确定分组遵循的协议，IPv4 协议下，该字段固定为 4；2) 报头长度（IHL），确定分组头部和内容的分界，IPv4 协议下，该字段固定为 5；3) 分组长度（Total length），确定整个分组长度；4) 生存时间（Time to Live，TTL），限制 IP 分组在网络中的传播时间，超时则死亡；5) 头校验和，用于校验分组头部是否正确；6) 发送方和接收方 IP 地址（Source/Destination address）。

## III. IPv4 分组封装发送

### A. 要求

发送 IPv4 分组时，需要进行如下操作：1) 申请内存用于 IPv4 分组，其大小通过上层内容确定；2) 按照 IPv4 协议标准格式，填写分组头部和内容各个字段，标识符（Identification）字段可以用一个随机数填写；3) 完成分组封装后，使用接口函数完成发送工作，最终将其发送到网络中。

本次作业下，IPv4 分组的封装发送通过以下函数实现。

```
int stud_ip_Upsend(char* pBuffer,
                   unsigned short len,
                   unsigned int srcAddr,
                   unsigned int dstAddr,
                   byte protocol,
                   byte ttl)
```

其中 pBuffer 指向 IPv4 上层协议数据内容，len 为上层写一数据长度，srcAddr 和 dstAddr 为源和目的 IPv4 地址，protocol 为 IPv4 上层协议的协议号，ttl 为生存时间。当封装成功并发送时，函数返回值为 0，否则出错返回 1。

### B. 需要使用的系统接口

申请连续内存的接口如下，length 为申请的内存字节长度，若申请成功则返回分配的内存区域的指针，否则返回 NULL。

```
char* malloc(int length);
```

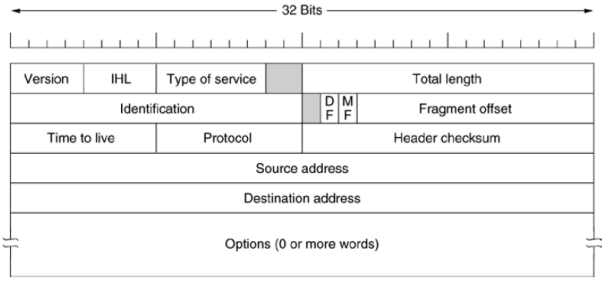


图 1. IPv4 分组结构。以字 (32 比特) 为单位展示，包括 20 字节的头部和可变长（至少 0 字节）的内容。其中头部具体由版本号（Version，IPv4 下固定为 4），报头长度（IHL，IPv4 下固定为 5 个 32 位字），类型（Type of Service），分组长度（Total Length），生存时间（Time to Live，TTL），协议号（Protocol），头校验和（Header Checksum），发送方 IP 地址（Source Address），接收方 IP 地址（Destination Addess）等字段组成。

[1]本报告源码可通过以下 git 命令获得，
git clone https://git.overleaf.com/15853721gmnmbcjkdywj

向下层发送 IPv4 分组的接口如下，pBuffer 为指向待发送的 IPv4 分组头部的指针，length 为待发送的 IPv4 分组的长度。

```
void ip_SendtoLower(char *Buffer, int length);
```

### C. 实现

IPv4 分组结构同第 II节。

首先需要分配 IPv4 分组所需的内存，其长度 = 上层协议数据长度 +IPv4 分组头长度，使用 malloc 接口进行分配。

封装分组时需要填写分组头部，其中必须要填写的字段有：Version 和 IHL 字段，分组长度字段，TTL 字段，Protocol 字段，源 IP 地址和目的 IP 地址字段，校验和字段。分组头填写完成后，将上层数据复制于头部之后，完成封装，之后使用 ip_SendtoLower 接口发送。

Version 字段固定填写 4，IHL 字段固定填写 5，分组长度字段填写申请的内存长度；Protocol 和两个地址字段按照参数进行填写，需要注意的是，由于大端小端区别，填写时需要进行大小端转换；校验和最后填写，按照如下伪代码表示的算法产生。

```
unsigned int GenerateCheckSum(char* Buffer)
{
    unsigned int HeaderCheckSum = 0;
    for (half-word i in Header)
        HeaderCheckSum += i;
    HeaderCheckSum += HeaderCheckSum >> 16;
    HeaderCheckSum = ~HeaderCheckSum;
    return HeaderCheckSum;
}
```

程序的 C/C++ 风格的伪代码如下所示。

```
int stud_ip_Upsend(char* pBuffer,
                   unsigned short len,
                   unsigned int srcAddr,
                   unsigned int dstAddr,
                   byte protocol,
                   byte ttl)
{
    // Calculate length of the packet
    // Malloc and inialize the packet as Buffer

    setVersionIHL(Buffer);
    setLength(Buffer, TotalLen);
    setTTL(Buffer, ttl);
    setProtocol(Buffer, protocol);
    setSrcAddr(Buffer, srcAddr);
    setDstAddr(Buffer, dstAddr);
    setCheckSum(Buffer);
    setContent(Buffer, pBuffer, len);

    // Send the IPv4 packet
    ip_SendtoLower((char*)Buffer, TotalLen);
    return 0;
}
```

## IV. IPv4 分组接收检查

### A. 要求

接收 IPv4 分组后，需要进行如下操作：1) 检查 Version，IHL，TTL 字段是否正确，如果出错则丢弃并指明错误类型；2) 检查目的 IP 地址是否是本机 IP 或者广播地址，若不是则丢弃并指明错误类型；3) 检查校验和，若出错则丢弃并指明错误类型；4) 将头部意外的内容交由上层协议进行后续处理。

本次作业下，IPv4 分组的接收检查通过以下函数实现。

```
int stud_ip_recv(char* pBuffer,
                 unsigned short length)
```

其中 pBuffer 指向接收到的 IPv4 分组内容，length 为分组长度。当检查通过并交付上层后，函数返回值为 0，否则出错返回 1。

### B. 需要使用的系统接口

获取本机 IPv4 地址的接口如下，没有参数，返回值为本机的 IPv4 地址。

```
unsigned int getIpv4Address();
```

丢弃分组并报错的接口如下，pBuffer 为需要丢弃的分组，type 为错误类型编号，共有五种错误类型如下，分别为：校验和出错，TTL 出错，IP 版本号出错，IHL 出错，目的地址出错。

```
// IP校验和出错
#DEFINE STUP_IP_TEST_CHECKSUM_ERROR
// TTL值出错
#DEFINE STUP_IP_TEST_TTL_ERROR
// IP版本号出错
#DEFINE STUP_IP_TEST_VERSION_ERROR
// 头部长度出错
#DEFINE STUP_IP_TEST_HEADLEN_ERROR
// 目的地址出错
#DEFINE STUP_IP_TEST_DESTINATION_ERROR
void ip_DiscardPkt(char *pBuffer, int type);
```

向上层交付 IPv4 分组内容的接口如下，pBuffer 为指向待交付内容的指针，length 为待交付内容的长度。

```
void ip_SendtoUp(char *pBuffer, int length);
```

### C. 实现

IPv4 分组结构同第 II节。

需要检查五种错误，若都通过则可以直接交付，否则丢弃并报告错误编号。

Version 字段必须为 4，否则出错，报告 IP 版本号错误；IHL 字段必须大于等于 5（报文头最少需要 5 个 32 位字），否则出错，报告头部长度错误；TTL 必须非 0，否则出错，报告 TTL 错误；目的地址必须为本机地址或广播地址（0xFFFFFFFF），否则出错，报告目的地址错误；头部校验和的计算方法伪代码如下，若出错，报告校验和错误。

```
unsigned int getCheckSum(char* Buffer)
{
    unsigned int HeaderCheckSum = 0;
    for (half-word i in Header)
        HeaderCheckSum += i;
    HeaderCheckSum += HeaderCheckSum >> 16;
    return HeaderCheckSum;
}
```

程序的 C/C++ 风格的伪代码如下所示。

```
int stud_ip_recv(char* pBuffer, unsigned short length)
{
    // Get all required segments.
    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 1;
    }
    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;
    }
    if ((unsigned short)(~HeaderCheckSum)) {
        ip_DiscardPkt(pBuffer,
                      STUD_IP_TEST_CHECKSUM_ERROR);
        return 1;
    }
    ip_SendtoUp(pBuffer, length);
    return 0;
}
```

## V. 代码

本次作业详细代码请见附录部分。

# 附录

```
/*
* 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

// IPv4 Version = 4
#define VERSION_DEFAULT ((unsigned int)(4));
// IHL = 5
#define IHL_DEFAULT ((unsigned int)(5));

unsigned int getVersion(char* pBuffer) { return (unsigned)pBuffer[0] >> 4; }
unsigned int getIHL(char* pBuffer) { return (unsigned)pBuffer[0] & 0xF; }
unsigned int getTTL(char* pBuffer) { return (unsigned)pBuffer[8]; }
unsigned int getDstAddr(char* pBuffer) { return ntohl(*(unsigned int *)(&pBuffer[16])); }
unsigned int getCheckSum(char* pBuffer)
{
    unsigned int HeaderCheckSum = 0;
    for (int i = 0; i < 20; i += 2) {
        HeaderCheckSum += ((pBuffer[i] & 0xFF) << 8) + (pBuffer[i + 1] & 0xFF);
    }
    HeaderCheckSum += (HeaderCheckSum >> 16);
    return HeaderCheckSum;
}

int stud_ip_recv(char* pBuffer, unsigned short length)
{
    unsigned int Version = getVersion(pBuffer);
    unsigned int IHL = getIHL(pBuffer);
    unsigned int TTL = getTTL(pBuffer);
    unsigned int DstAddr = getDstAddr(pBuffer);
    unsigned int HeaderCheckSum = getCheckSum(pBuffer);

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

    ip_SendtoUp(pBuffer, length);
    return 0;
}

void setVersionIHL(char* Buffer) { Buffer[0] = VERSION_DEFAULT << 4 | IHL_DEFAULT; }
void setLength(char* Buffer, unsigned int TotalLen) { Buffer[2] = TotalLen >> 8; Buffer[3] = TotalLen; }
void setTTL(char* Buffer, byte ttl) { Buffer[8] = ttl; }
void setProtocol(char* Buffer, byte protocol) { Buffer[9] = protocol; }
void setSrcAddr(char* Buffer, unsigned int srcAddr)
{
    Buffer[12] = srcAddr >> 24;
    Buffer[13] = srcAddr >> 16;
    Buffer[14] = srcAddr >> 8;
    Buffer[15] = srcAddr;
}
void setDstAddr(char* Buffer, unsigned int dstAddr)
{
    Buffer[16] = dstAddr >> 24;
    Buffer[17] = dstAddr >> 16;
    Buffer[18] = dstAddr >> 8;
    Buffer[19] = dstAddr;
}
void setCheckSum(char* Buffer)
{
    unsigned int HeaderCheckSum = 0;
    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);
}
void setContent(char* Buffer, char* pBuffer, unsigned short len) { memcpy(Buffer + 20, pBuffer, len); }

int stud_ip_Upsend(char* pBuffer, unsigned short len, unsigned int srcAddr,
                   unsigned int dstAddr, byte protocol, byte ttl)
{
    unsigned int TotalLen = 4 * IHL_DEFAULT + len;
    unsigned char* Buffer = (unsigned char*) malloc(TotalLen);

    memset(Buffer, 0, TotalLen);

    setVersionIHL(Buffer);
    setLength(Buffer, TotalLen);
    setTTL(Buffer, ttl);
    setProtocol(Buffer, protocol);
    setSrcAddr(Buffer, srcAddr);
    setDstAddr(Buffer, dstAddr);
    setCheckSum(Buffer);
    setContent(Buffer, pBuffer, len);

    ip_SendtoLower((char*)Buffer, TotalLen);
```

```
    return 0;
}
```