IPv4协议收发实验

1600013019 张智涵

一、实验内容

本实验要求实现IPv4协议中的数据报收发环节，模拟客户端的网络层实现。在将数据发送给服务器时，网络层将上层协议提供的数据按照IPv4协议的格式进行封装，调用链路层的接口将数据转发给链路层，由链路层实现数据的收发；收到服务器送回的数据时，网络层会检查数据的合法性并送给上层协议继续处理。

接收数据时，网络层主要检查的参数有：IPv4分组的生存时间、分组头校验和、头部长度、版本号、接收端IP地址等。在发送数据时，上述参数也需要通过网络层和原始数据一起封装，生成符合IPv4协议的分组。本实验要求实现IPv4协议的数据发送、数据正确接收以及五种需要丢弃的错误数据类型：头部长度错误、版本号错误、生存时间错误、头校验和错误或者接收端IP地址错误。

二、编程实现

为了方便数据的读取和封装，手动定义了ipv4结构体作为ipv4头部数据结构的抽象：

```
typedef struct ipv4
{
    unsigned short version_len_service;
    unsigned short total_length;
    unsigned short tag;
    unsigned short flag_offset;
    unsigned short ttl_protocol;
    unsigned short checksum;
    unsigned int source_addr;
    unsigned int dest_addr;
};
```

其中除最后的源IP和目的IP以外，其余字段为了方便均以2字节的short表示。

1.接收数据

接收数据时，测试函数stud_ip_recv会依次检验各参数以确认数据的合法性。

生存时间：分组头部中ttl_protocol字段的前8位，提取出之后和0做比，若大于0则合法，否则调用ip_DiscardPkt丢弃该分组；

版本号：分组头部version_len_service字段中前4位，提取出后和4做比（IPv4的版本号为4），若相等则合法，否则丢弃该分组；

头部长度：以4字节为单位，IPv4分组头部的固定部分长度为20字节，故头部长度的最小值应为5。头部长度为分组头部version_len_service字段中4-8位，提取出后和4做比，若头部长度大于4则合法，否则丢弃该分组；

目的IP：分组头部中的dest_addr字段，提取出之后和本机IP地址做比，若相等则该分组为本机需要接收的分组，否则丢弃该分组；

头部校验和：定义函数compute_checksum求一段unsigned short数组的校验和，具体实现为先求和，再将结果的前16位和后16位循环相加直到前16位为0。将pBuffer转换成(unsigned short *)类型后调用该函数，将函数返回值取反即为校验结果，若等于0则数据正确，否则数据在传输过程中出错，丢弃该分组；

若以上五个参数均正确，则调用ip_SendtoUp函数将pBuffer传输给上层协议。

2.发送数据

发送数据时，需要将源数据封装为IPv4分组，即对IPv4头部中的各参数赋值并和源数据连接起来。

version_len_service字段：包含版本号、头部长度、服务类型三个参数，其中版本号为4，固定头部长度为5（单位：4字节），本实验中服务类型为0。故该字段取值为0x4500。

total_length字段：总长度，包括头部长度和数据长度，故为20+len。

tag字段：标识字段，本实验中取值为0。

flag_offset字段：标志位和片偏置字段，不在本实验考察范围内，取值为0。

ttl_protocol字段：包含生存时间和协议类型两个参数，生存时间为测试函数的参数ttl，协议类型为测试函数的参数protocol，通过位运算连接起来即可。

souce_addr和dest_addr字段：分别为测试函数的参数srcAddr及dstAddr。

checksum字段：先置为0，当所有字段定值后，调用compute_checksum函数计算校验和并取反，填入checksum字段作为头部校验和的值。

头部参数定值完毕后，调用memcpy函数将源数据复制到分组头部的内存地址之后，调用ip_SendtoLower函数将分组传送给链路层进行分发。

实验代码：

```
#include "sysInclude.h"
#include <iostream>
#include <winsock.h>
#include <cstring>
using namespace std;

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

typedef struct ipv4
{
    unsigned short version_len_service;
    unsigned short total_length;
```

```
    unsigned short tag;
    unsigned short flag_offset;
    unsigned short ttl_protocol;
    unsigned short checksum;
    unsigned int source_addr;
    unsigned int dest_addr;
};

// function for calculating the header checksum;
unsigned short compute_checksum(unsigned short *pBuffer, int length)
{
    unsigned int sum = 0;
    for (int i = 0; i < length; i++) {
        sum += ntohs(pBuffer[i]);
        sum = (sum >> 16) + (sum & 0xffff);
    }
    return sum;
};

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

    // check the time to live
    unsigned int ttl = (ntohs(packet->ttl_protocol) >> 8) & 0xff;
    if (ttl <= 0) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_TTL_ERROR);
        return 1;
    }

    // check the version number (supposed to be 4)
    unsigned int version = (ntohs(packet->version_len_service) >> 12) & 0xf;
    if (version != 4) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_VERSION_ERROR);
        return 1;
    }

    // check the head length (supposed to be bigger than 4)
    unsigned int head_len = (ntohs(packet->version_len_service) >> 8) & 0xf;
    if (head_len <= 4) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_HEADLEN_ERROR);
        return 1;
    }

    // check the destination address (supposed to be host ip)
    if (ntohl(packet->dest_addr) != getIpv4Address()) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_DESTINATION_ERROR);
        return 1;
    }

    // check the checksum
    unsigned short checksum_result = ~compute_checksum((unsigned short *)pBuffer, head_len *
2);
    if (checksum_result != 0) {
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_CHECKSUM_ERROR);
        return 1;
    }

    // valid data
```

```
    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)
{
    ipv4 *header = new(ipv4);
    header->version_len_service = htons(0x4500);
    header->total_length = htons(20 + len);
    header->tag = 0;
    header->flag_offset = 0;
    header->ttl_protocol = htons(((unsigned short)ttl << 8) + (unsigned short)protocol);
    header->checksum = 0;
    header->source_addr = htonl(srcAddr);
    header->dest_addr = htonl(dstAddr);

    // compute the checksum
    header->checksum = htons(~compute_checksum((unsigned short *)header, 10));

    // copy the data and send the packet
    memcpy((char *)header + 20, pBuffer, len);
    ip_SendtoLower((char *)header, len + 20);

    return 0;
}
```