IPv4协议转发实验

1600013019 张智涵

一、实验内容

本实验要求实现作为通信中间节点的路由器的分组转发功能。路由器可以连接不同的网络，当不同网络的主机相互发送数据时，路由器可作为中间站存在，从发送端接收数据，转发到目的地址对应的网络端口。每个路由器维护一个路由表，保存有不同地址所需要转发的下一跳路由器地址。当路由器接收到数据时，通过查询路由表，根据路由算法找出下一跳的地址并转发数据。若数据出错或者找不到下一跳的地址，则丢弃数据。

本实验分为本机接收数据、丢弃数据、转发数据三个子实验，要求同时实现接收数据和转发数据两种功能：即当数据报的目的地址为本机地址时，直接将数据传送给上层协议；当数据报的目的地址不是本机地址时，根据系统提供的路由表查找下一跳地址并转发。其中，接收分组部分与IPv4收发协议实验中的接收数据部分类似，而转发分组部分相比于IPv4收发实验中的发送数据部分多了查找下一跳地址的步骤。

二、编程实现

模拟路由器首先需要模拟路由表的建立。由于本实验中路由表项是静态表项，因此直接采用vector存放系统给出的路由表项，作为模拟的路由表。在查询路由表时，遍历vector，如果能找到和数据报中目的地址相同的路由表项，则读出对应的下一跳路由器地址，否则丢弃数据报。

与IPv4收发实验相同，为了将传来的参数pBuffer表示成IPv4分组头部的形式，我们定义结构体

```
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;
};
```

作为IPv4分组头部的表示。接收数据之后，首先取出数据报的生存时间（ttl_protocol字段的前8位）判断其是否大于零，否则调用fwd_DiscardPkt函数丢弃数据；若生存时间合法，则将目的地址dest_addr和本机地址相比较，若相等，则直接调用fwd_LocalRcv函数，将数据报传送给上层协议。

若数据报合法并且不是本机接收，则在路由表中查找对应表项，若找不到则调用fwd_DiscardPkt函数丢弃数据；若找到对应表项，则从对应表项中读出下一跳地址nexthop，并将数据报的生存时间减一，将分组头部的校验和部分（checksum字段）清零，并调用compute_checksum函数（实现方法同IPv4收发实验）重新计算校验和再填入分组头部中的对应字段。最后，调用fwd_SendtoLower函数将分组传递给下层协议，继续向下一跳路由器发送数据。

实验代码：

```
/*
* THIS FILE IS FOR IP FORWARD TEST
*/
#include "sysInclude.h"
#include <iostream>
#include <cstring>
#include <vector>
#include <winsock.h>
using namespace std;

// system support
extern void fwd_LocalRcv(char *pBuffer, int length);

extern void fwd_SendtoLower(char *pBuffer, int length, unsigned int nexthop);

extern void fwd_DiscardPkt(char *pBuffer, int type);

extern unsigned int getIpv4Address();

// implemented by students
vector<stud_route_msg> route_list;

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;
};

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;
};

void stud_Route_Init()
{
    route_list.clear();
    return;
}

void stud_route_add(stud_route_msg *proute)
{
    route_list.push_back(*proute);
    return;
}


int stud_fwd_deal(char *pBuffer, int 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) {
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR);
        return 1;
    }

    // should be received by host
    unsigned int dest_addr = ntohl(packet->dest_addr);
    if (dest_addr == getIpv4Address()) {
        fwd_LocalRcv(pBuffer, length);
        return 0;
    }

    stud_route_msg route_msg;
    for (int i = 0; i < route_list.size(); i++) {
        if (ntohl(route_list[i].dest) == dest_addr) {
            route_msg = route_list[i];
            break;
        }
        // cannot find it
        if (i == route_list.size() - 1) {
            fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE);
            return 1;
        }
    }

    // ttl-1, recompute the checksum
    unsigned int nexthop = ntohl(route_msg.nexthop);
    ttl--;
    unsigned int protocol = ntohs(packet->ttl_protocol) & 0xff;
    packet->ttl_protocol = htons(((unsigned short)ttl << 8) + (unsigned short)protocol);
    packet->checksum = 0;
    packet->checksum = htons(~compute_checksum((unsigned short *)packet, 10));
    fwd_SendtoLower((char*)packet, length, nexthop);

    return 0;
}
```