ipv4 协议转发实验报告

黄道吉

2020 年 5 月 12 日

IPv4 协议是互联网的核心协议。这次实验要求实现 IPv4 协议中转发数据包的部分。与上次实验不同的是，本次实验实现的主要是路由器的功能而不是主机的功能。实验要求在原有收发功能之外，另加 IPv4 分组的转发代码，要求能够实现路由表的数据结构，实现路由器选择路由的功能。

1 数据结构

这次实验的中心是设计路由表的结构。在我的实现中，路由表被设置为一个路由消息 (route_msg) 到下一跳的映射，利用 c++ 的标准库实现。另外增加一个函数判断某个地址是否落在路由消息的目的地址之内。这样对路由表的初始化、添加和查找方法分别实现为：

- 初始化：设置路由表为空。
- 添加路由消息：先查找路由表中有没有冗余表项，没有的话再添加进路由表中。注意对于 masklen 不相同的路由信息，即便目的地址相同，也不能够认为他们相互有包含/被包含的关系，因为在进行最大长度匹配时会忽略掉 masklen 更小的表项。为尽量压缩路由表的大小，实现的时候认为目的地址相同（加掩码的意义下），并且 a) 掩码长度相同或者 b) 待添加的表项掩码更长但是下一条相同的路由信息相同的路由信息是冗余的信息。在 a) 情况下，它的目的地址和某个已经存在的表项完全相同，b) 的情况下它的目的地址和下一跳被已有表项完全覆盖。
- 查找匹配的路由：迭代 map 中各个元素，查看目的地址是否和路由消息的目的地址相等（在加上掩码的意义下），在所有相等的路由消息中选择最长匹配的。

2 实现细节

本节介绍我的实现的细节和只参考手册和查阅资料没有解决的问题。

2.1 函数逻辑

这部分介绍主函数 stud_fwd_deal 实现的功能。在收到需要转发/接受的分组时，首先判断分组的存活时间，为 0 的应当抛弃。因为手册中只给出两种丢包的错误，所以不再判断分组头部的其他域。对于目的地址是自己（本机 ip 地址或广播地址）的数据包应当接收。对于其他的数据分组，则需要迭代路由表每一个表项，找到最大长度匹配的表项（因此每一次都需要迭代整个路由表），决定转发的接口。最后参考上一个实验的代码，重组分组头部（ttl 减一，计算 checksum），送到下层。

2.2 残存问题

实现的过程中仍然没有解决的问题有

- 和上一次实验相同，本机地址是否需要包含诸如 127.0.0.0/8 的地址，还是测试样例中并不会包含这些地址
- 路由消息的各个域的字节序是什么

此外，为了更加有效的压缩路由表的体积，一个可行的方法可能是添加表项时检查/定期扫描有没有可以合并的表项，e.g. <162.0.0.0/8, 0> 和 <163.0.0.0/8, 0> 可以合并为 <162.0.0.0/7, 0>，但并不确定 a) 这种合并方式在测试样例下的正确性，e.g. 合并后再添加表项 <162.0.0.0/8, 1> 时并不会认为和已有的表项冲突，和 b) 计算的复杂性是否合适。

3 代码实现

```
#include <map>
using namespace std; // for its map data structure

/* API provided by system */

void fwd_LocalRcv(char *pBuffer, int length);
void fwd_SendtoLower(char *pBuffer, int length, unsigned int nexthop);
void fwd_DiscardPkt(char * pBuffer, int type);
UINT32 getIpv4Address( );

/* routetable: route_message -> nexthop */
map<stud_route_msg, int> routetable;

/* helper function(s) */
int helper_route_include(unsigned int destination_address, stud_route_msg route_message){
    // destination_address == route_message.dest under a certain mask
    return (destination_address - route_message.dest) & ((~0) << (route_message.masklen)) == 0;
}

int stud_fwd_deal(char * pBuffer, int length){

/* code to check ttl and dstaddr is borrowed from ipv4_receive.c from here */

    unsigned int time_to_live = ((unsigned int)pBuffer[8]);
    if(time_to_live == 0){
        ip_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR);
        return -1;
    }

    unsigned int destination_address = ntohl(*(unsigned int* )(pBuffer + 16)); // might be wrong

/* to here */

    if(destination_address == getIpv4Address() || destination_address != (~0)){
        fwd_LocalRcv(pBuffer, length);
        return 0;
    }
```

```
    int nexthop = -1, maxlengthmatch = -1; // assume nexthop != -1 for all route_message
    for(map<stud_route_msg, int>::iterator it = routetable.begin();
        it != routetable.end();
        it++){
        /* assume masklen is the 'n' in x.y.z.m/n */
        if(helper_route_include(destination_address, *it)){
            // a match found!
            if(maxlengthmatch < 32 - it->masklen){
                maxlengthmatch = 32 - masklen;
                nexthop = it-> nexthop;
            }
        }
    }

    if(nexthop == -1){
        /* no match found */
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE);
        return -1;
    }


    unsigned char* total_buffer = malloc(length);
    memcpy(total_buffer, pBuffer, length);
    total_buffer[8] = time_to_live - 1;

/* code to compute checksum is borrowed from ipv4_receive.c from here */
    unsigned int header_checksum = 0;
    for(int i = 0; i < 20; i += 2){
        header_checksum += (total_buffer[i] << 8) + total_buffer[i + 1];
    }
    while(header_checksum >> 16){
        header_checksum == (header_checksum >> 16) + (header_checksum & 0xFFFF);
    }
    header_checksum = ~header_checksum;
    total_buffer[10] = header_checksum >> 8;
    total_buffer[11] = header_checksum & 0xFF;
/* to here */
    fwd_SendtoLower(total_buffer, length, nexthop);

    return 0;
}

void stud_route_add(stud_route_msg* proute){
    /* subject to change: what is the byteorder of dest, masklen and nexthop? */
    for(map<stud_route_msg, int>::iterator it = routetable.begin();
        it != routetable.end();
        it++){
        /* assume masklen is the 'n' in x.y.z.m/n */
        if(helper_route_include(proute->destination_address, *it)
           && (proute->masklen == it->masklen
               || (proute->masklen >= it->masklen && proute->nexthop == it->nexthop)
              )
           ){
            // a match found! no need to update route table
            return ;
        }
    }
```

```
    routetable.insert(std::pair<stud_route_msg, int>(*proute, proute->nexthop));
    return ;
}

void stud_Route_Init(){
    routetable.clear();
    return ;
}
```