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 协议转发实验

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

通过前面的实验，我们已经深入了解了 IPv4 协议的分组接收和发送处理流程。本实验需要将实验模块的角色定位从通信两端的主机转移到作为中间节点的路由器上，在 IPv4 分组收发处理的基础上，实现分组的路由转发功能。

网络层协议最为关注的是如何将 IPv4 分组从源主机通过网络送达目的主机，这个任务就是由路由器中的 IPv4 协议模块所承担。路由器根据自身所获得的路由信息,将收到的 IPv4 分组转发给正确的下一跳路由器。如此逐跳地对分组进行转发，直至该分组抵达目的主机。IPv4 分组转发是路由器最为重要的功能。

本实验设计模拟实现路由器中的 IPv4 协议,可以在原有 IPv4 分组收发实验的基础上，增加 IPv4 分组的转发功能。对网络的观察视角由主机转移到路由器中，了解路由器是如何为分组选择路由，并逐跳地将分组发送到目的主机。本实验中也会初步接触路由表这一重要的数据结构，认识路由器是如何根据路由表对分组进行转发的。

二、实验要求

在前面 IPv4 分组收发实验的基础上，增加分组转发功能。具体来说，

对于每一个到达本机的 IPv4 分组，根据其目的 IPv4 地址决定分组的处理行为，对该分组进行如下的几类操作：

1) 向上层协议上交目的地址为本机地址的分组；

2) 根据路由查找结果，丢弃查不到路由的分组；

3) 根据路由查找结果，向相应接口转发不是本机接收的分组。

三、实验内容

实验内容主要包括：

1) 设计路由表数据结构。

设计路由表所采用的数据结构。要求能够根据目的 IPv4 地址来确定分组处理行为（转发情况下需获得下一跳的 IPv4 地址）。路由表的数据结构和查找算法会极大的影响路由器的转发性能，有兴趣的同学可以深入思考和探索。

2) IPv4 分组的接收和发送。

对前面实验（IP 实验）中所完成的代码进行修改，在路由器协议栈的 IPv4 模块中能够正确完成分组的接收和发送处理。具体要求不做改变，参见“IP 实验”。

3) IPv4 分组的转发。

对于需要转发的分组进行处理，获得下一跳的 IP 地址，然后调用发送接口函数做进一步处理。

四、实验过程

1. 路由表 RouteTable 实现

为了高效完成路由表的查询 query 和插入 insert 操作，此处使用了 C++的 STL 容器 std::map。本质为红黑树。其中每个 pair<int, int>中存储了<DstAddr, NextHop>。

2. 初始化 stud_Route_Init()

无需过多操作，只需将 map RouteTable 初始化即可。

3. 添加条目 stud_route_add()

首先 proute 中读出 DstAddr, NextHop, masklen；而后计算出 DstAddr 和 NextHop 的主机顺序的值；最后构建 pair<DstAddr, NextHop>，使用 insert() 方法将其插入路由表中。

4. 接收转发 stud_fwd_deal()

该函数的主要处理流程如下：

- 根据 pBuffer 读入 IPv4 头的数据
- 若当前主机为接收地址，则调用 fwd_LocalRcv() 向上层发送数据
- 若该包超出生存时间，调用 fwd_DiscardPkt() 抛弃该包
- 在路由表中查找对应包的 DstAddr 的 nexthop 地址。若路由表中不存在对应条目，则调用 fwd_DiscardPkt() 抛弃该包
- 若路由表中存在对应条目，则修改 TTL 和报头校验和，调用 fwd_SendtoLower()转发给下一跳地址

五、附录——代码

```
#include "sysInclude.h"
#include<map>
using std::map;
using std::pair;

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();

/* RouteTable <DstAddr, nexthop> */
map<int, int> RouteTable;

/* Initializing the RouteTable */
void stud_Route_Init() {
     RouteTable.clear();
     return;
}

/* Adding a new entry to the RouteTable */
void stud_route_add(stud_route_msg* proute) {
     const int DstAddr = (ntohl(proute->dest)) & (0xFFFFFFFF << (32 -
htonl(proute->masklen)));
     const int NextHop = ntohl(proute->nexthop);
     RouteTable.insert(std::pair<int, int>(DstAddr, NextHop));
     return;
}

/* Dealing with the reception and forwarding */
int stud_fwd_deal(char* pBuffer, int length) {
     const int IHL = pBuffer[0] & 0xf;
     const int TTL = (int)pBuffer[8];
     int DstAddr = ntohl(*(unsigned *)(&pBuffer[16]));

     if (DstAddr == getIpv4Address()) {
          fwd_LocalRcv(pBuffer, length);
          return 0;
     }
     if (TTL <= 0) {
          fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR);
          return -1;
     }
```

```
    map<int, int>::iterator map_iterator = RouteTable.find(DstAddr);

    if (map_iterator == RouteTable.end()) {
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE);
        return -1;
    }

    unsigned char* Buffer = (unsigned char *)malloc(length);
    memcpy(Buffer, pBuffer, length);

    Buffer[8] = TTL - 1;

    unsigned int HeaderCheckSum = 0;
    memset(&Buffer[10], 0, sizeof(short));
    for (int i = 0; i < 4 * IHL; i += 2) {
        HeaderCheckSum += ((Buffer[i] & 0xFF) << 8) + (Buffer[i + 1] & 0xFF);
    }
    HeaderCheckSum += (HeaderCheckSum >> 16);
    HeaderCheckSum = ~HeaderCheckSum;

    Buffer[10] = (unsigned short)HeaderCheckSum >> 8;
    Buffer[11] = (unsigned short)HeaderCheckSum & 0xFF;

    fwd_SendtoLower((char *)Buffer, length, (*map_iterator).second);

    return 0;
}
```