哈尔滨工业大学

<<计算机网络>>

实验报告

(2017 年度春季学期)

姓名:	张茗帅
学号:	**1140310606**
学院:	计算机科学与技术学院
教师:	聂兰顺

实验三 IPv4 分组转发实验

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。

实验环境：

接入到Netriver网络实验系统服务器的主机

Windows XP、Windows7/8/10或苹果OS

开发语言：C语言

二、实验内容

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；

2 根据路由查找结果，向相应接口转发不是本机接收的分组。

实验要点包括：

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。

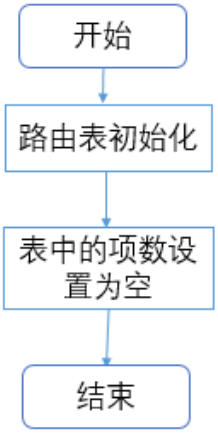
对于需要转发的分组进行处理，获得下一跳的 IP 地址，然后调用发送接口函数做进一步处理

三、实验过程及结果

实验原理说明：

1 主要函数流程图

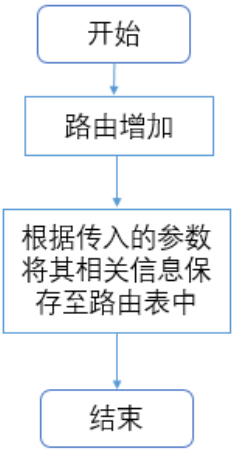
路由表初始化



说明：

路由表的初始化设置相对简单，只是将路由表清空即可

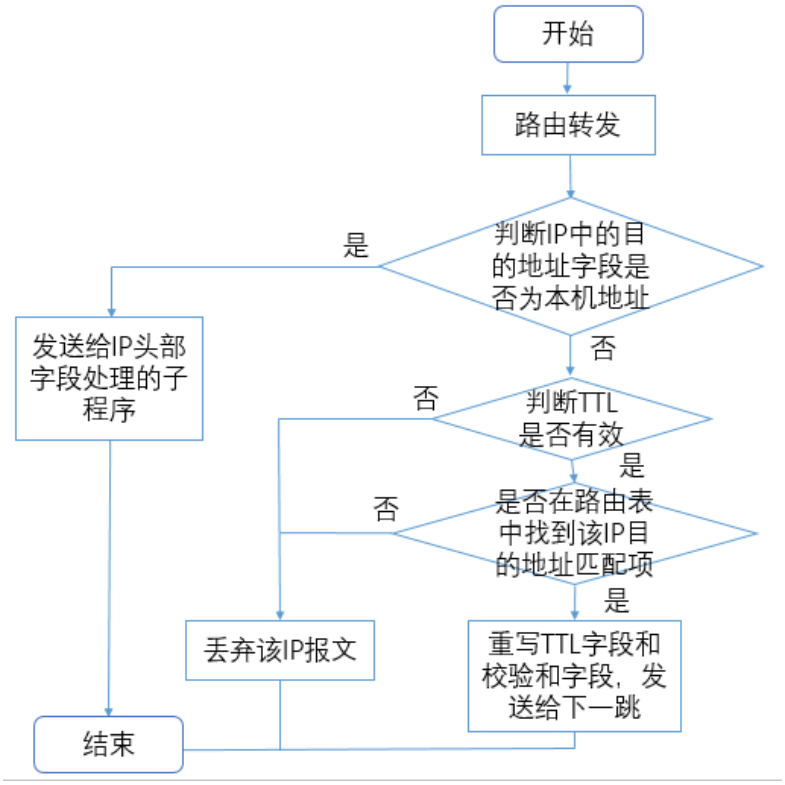
路由表添加项



说明：

根据系统已经规定的参数进行传入，将其相关的信息保存到路由表中，例如目的地址、子网掩码的长度、子网掩码的值以及下一跳的位置。其中子网掩码的值可以通过移位运算和子网掩码的长度计算得到。

路由转发



说明：

本地获取到 IP 报文后，首先对头部字段提取出 TTL 和目的 IP 地址，如果 TTL 小于 0，那么直接调用 fwd_DiscardPkt()丢弃该 IP 报文，同时判断目的 IP 地址是否为本机地址，如果是本机地址，则调用本地 IP 的处理判断子程序 fwd_LocalRcv()进行其头部的深度判断，如果目的地址不是本机地址，则根据此目的地址寻找路由表看是否能够找到匹配项，如果找到匹配项，则修改 IP 头部字段中 TTL 的值并且重写校验和，再将修改过的 IP 报文发送给数据链路层，用于在网络中传输。如果没有在路由表中没有找到匹配项，则丢弃该 IP 报文。

2 新建数据结构的说明

在这里新建的数据结构主要就是路由表这个数据结构，主要用于存贮其中每一项的信息，包括目的 IP 地址、子网掩码、下一跳地址。这里面的目的 IP 地址一般都指的是子网地址，其中子网掩码的长度用于路由匹配中找到一个最优（大）匹配，子网掩码具体的值用于和具体的目的 IP 地址做位与运算得到子网地址，从而才可以在路由表中找到相对应匹配的项。

```
struct routeTable   //路由表
{
   unsigned int destIP;  //目的 IP 地址
   unsigned int mask;   //子网掩码对应的整数值
   unsigned int masklen; //子网掩码 1 的个数
   unsigned int nexthop; //下一跳地址
};
```

3 大量分组提高路由器转发效率的问题

主要采用两种方法：路由聚合和多线程查找路由表

路由聚合：

路由聚合（也叫汇总）是让路由选择协议能够用一个地址通告众多网络，旨在缩小路由器中路由选择表的规模，以节省内存，并缩短 IP 对路由选择表进行分析以找出前往远程网络的路径所需的时间。

路由聚合的方法

（1）Inter-area 路由聚合

Inter-area 路由聚合在 ABR 上进行，对来自 AS 内部的路由起作用。对通过路由重新分发而引入的外部路由不起作用。为了利用路由聚合这个特性;在一个区域中的网络地址应当连续，这些成块的地址可以形成一个范围。

（2）External 路由聚合

External 路由聚合是指通过路由重新分发将 External 路由引入 OSPF 区域中。同样，要确保要聚合的 External 路由的范围是连续的。如果从两个不同的路由器聚合的路由含有相同部分，则在报文转发到目的地址过程中会出错的。

路由聚合算法

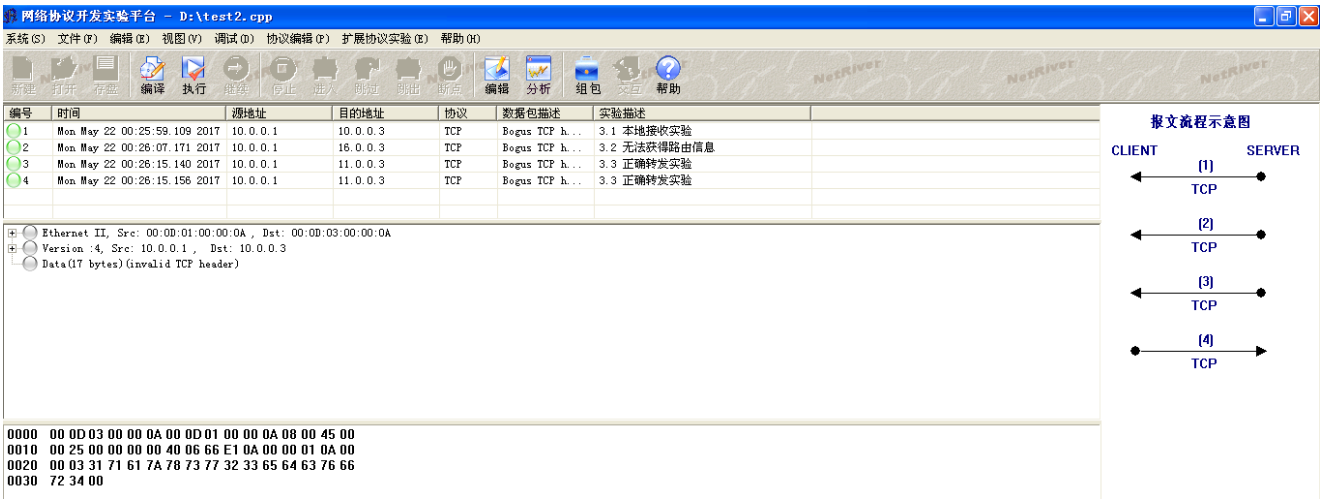
汇总（路由聚合）算法比较简单，因为只需要知道块的大小，

例如:网络 192.168.16.0——192.168.31.0 块大小是多少呢？刚好是 16 个 C 类 网络，块大小 16 就满足，由于通告汇总地址带的网络地址总是块中的第一个网络地址，这里是 192.168.16.0。确定子网掩码，什么样的子网子网掩码提供块大小为 16 呢？答案是 240 也就是/20 因此子网掩码为 255.255.240.0。

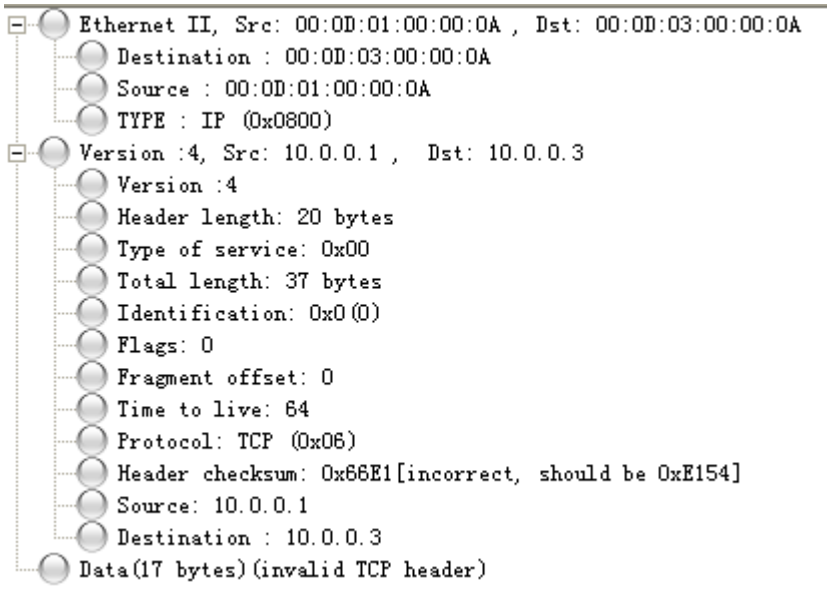
多线程查找路由表：

通过多个线程并行进行路由查找的任务,提高路由查找的吞吐能力和效率

实验结果截图：



报文分析图如上，具体的每个报文分析很简单，在我上一个报告（IP 分发与转发）有详细地写到。大概来讲，就是你要看哪个报文的相关信息，就双击哪个报文，下面的二进制数字从 000E 处开始便是 IP 的头部字段的第一个字节，一直读到 0021 为 IP 头部字段的最后一个字节。根据这些二进制数值，得到 IP 头部各字段具体的值，按 Version 旁边的加号既可以得到显示。如下图，然后再按照其值进行分析即可。



源代码（含详细注释）：

```
/*
* THIS FILE IS FOR IP FORWARD TEST
*/

#include <iostream>
#include <vector>
#include "sysInclude.h"

// 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

struct routeTable    //路由表
{
   unsigned int destIP;    //目的 IP 地址
   unsigned int mask;      //子网掩码对应的整数值
   unsigned int masklen; //子网掩码 1 的个数
   unsigned int nexthop; //下一跳地址
};
std::vector<routeTable> m_table; //用 vector 保存每一个路由表中信息

void stud_Route_Init()          //路由表初始化  清空
{
   m_table.clear();
   return;
}

void stud_route_add(stud_route_msg *proute) //路由表添加每一项的信息，
{
   routeTable newTableItem;
   newTableItem.masklen = ntohl(proute->masklen);
   newTableItem.mask = (1<<31)>>(ntohl(proute->masklen)-1);
   newTableItem.destIP = ntohl(proute->dest)&newTableItem.mask;
   newTableItem.nexthop = ntohl(proute->nexthop);
   m_table.push_back(newTableItem);
   return;
}

int stud_fwd_deal(char *pBuffer, int length)        //获取 IP 报文后进行处理
{

   int IHL = pBuffer[0] & 0xf;                      //获取头部字段长度
   int TTL = (int)pBuffer[8];                       //获取生存时间 TTL
   int destIP = ntohl(*(unsigned int*)(pBuffer+16));    //获得目的 IP 地址

   if(destIP == getIpv4Address())                   //如果目的 IP 地址是本机地址，则直接发送给
本机进行处理
```

```
  {
    fwd_LocalRcv(pBuffer, length);
    return 0;
  }

  if(TTL <= 0)                    //如果 TTL 小于 0 则丢弃该 IP 报文
 {
    fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR);
    return 1;
  }

  bool isMatch = false;       //用于判断是否在路由表中找到了对应的匹配项
  unsigned int longestMatchLen = 0; //用于寻找最长的匹配
  int bestMatch = 0;              //用于保存最终匹配对应的路由表中的条目号

  for(int i = 0; i < m_table.size(); i ++) //寻找路由表中的最佳匹配
  {
    if(m_table[i].masklen > longestMatchLen && m_table[i].destIP == (destIP &
m_table[i].mask))
    {                             //目标 IP 地址和子网掩码按位取&结果为子网的地址（即可
对应到路由表中的项）
      bestMatch = i;
      isMatch = true;
      longestMatchLen = m_table[i].masklen;
    }
  }

  if(isMatch)             //如果在路由表中找到了匹配项，则构造 IP 数据包发送
  {
    char *buffer = new char[length];
    memcpy(buffer,pBuffer,length);
    buffer[8]--; //TTL - 1          //重新 TTL 字段
    int sum = 0;
    unsigned short int localCheckSum = 0;
    for(int j = 0; j < 2 * IHL; j ++) //重写校验和字段
    {
      if (j != 5) {
        sum = sum + (buffer[j*2]<<8) + (buffer[j*2+1]);
      }

    }

    while((unsigned(sum) >> 16) != 0)
    sum = unsigned(sum) >> 16 + sum & 0xffff;
```

```
        localCheckSum = htons(0xffff - (unsigned short int)sum);
        memcpy(buffer+10, &localCheckSum, sizeof(unsigned short));

        fwd_SendtoLower(buffer, length, m_table[bestMatch].nexthop);
        return 0;                       //发送 IP 报文给下一层
    }
    else
    {
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE);
        return 1;
    }
    return 1;
}
```

四、实验心得

通过本次实验，我对路由器，这个网络层中十分重要的设备有了十分深刻的认识，对路由的具体流程有了更为深入的理解。与此同时，对子网掩码的印象得到了再一次的加深，现在心中已经熟记如何利用子网掩码和具体的目的 IP 地址来计算子网地址。知道了路由时的最大匹配原则，同时在实验中实现了它，再一次增强了自己的代码能力，收获颇丰。