哈尔滨工业大学

<<计算机网络>>

实验报告

(2018 年度春季学期**)**

姓名:	
学号:	
学院:	计算机科学与技术学院
教师:	

实验四 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。

二、实验内容

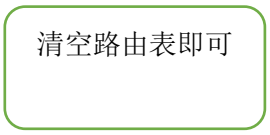
在前面 IPv4 分组收发实验的基础上，增加分组转发功能。具体来说，对于每一个到达本机的 IPv4 分组，根据其目的 IPv4 地址决定分组的处理行为，对该分组进行如下的几类操作：

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；
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。

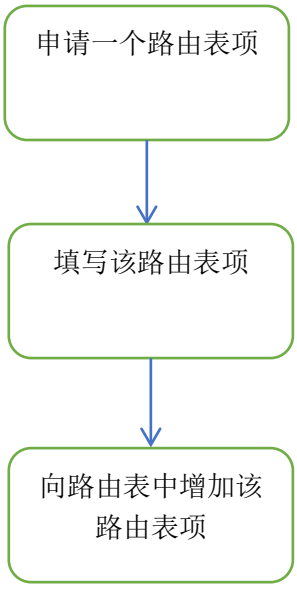
三、实验过程及结果

1. 路由表初始化、路由增加、路由转发三个函数的实现流程图：

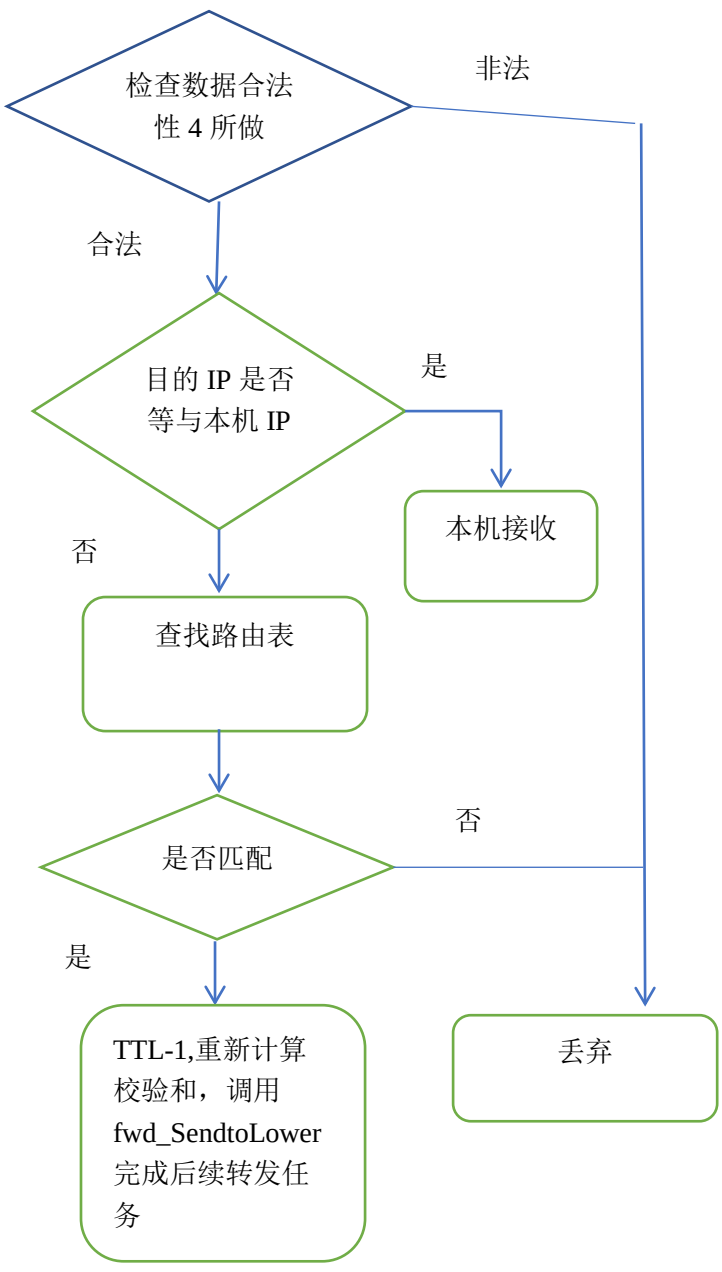
路由表初始化



路由增加：



路由转发函数



2. 自定义数据结构说明

路由表项

```
struct routeTable
{
unsigned int destIP;
unsigned int mask;
unsigned int masklen;
unsigned int nexthop;
};
vector table      //路由器转发表
```

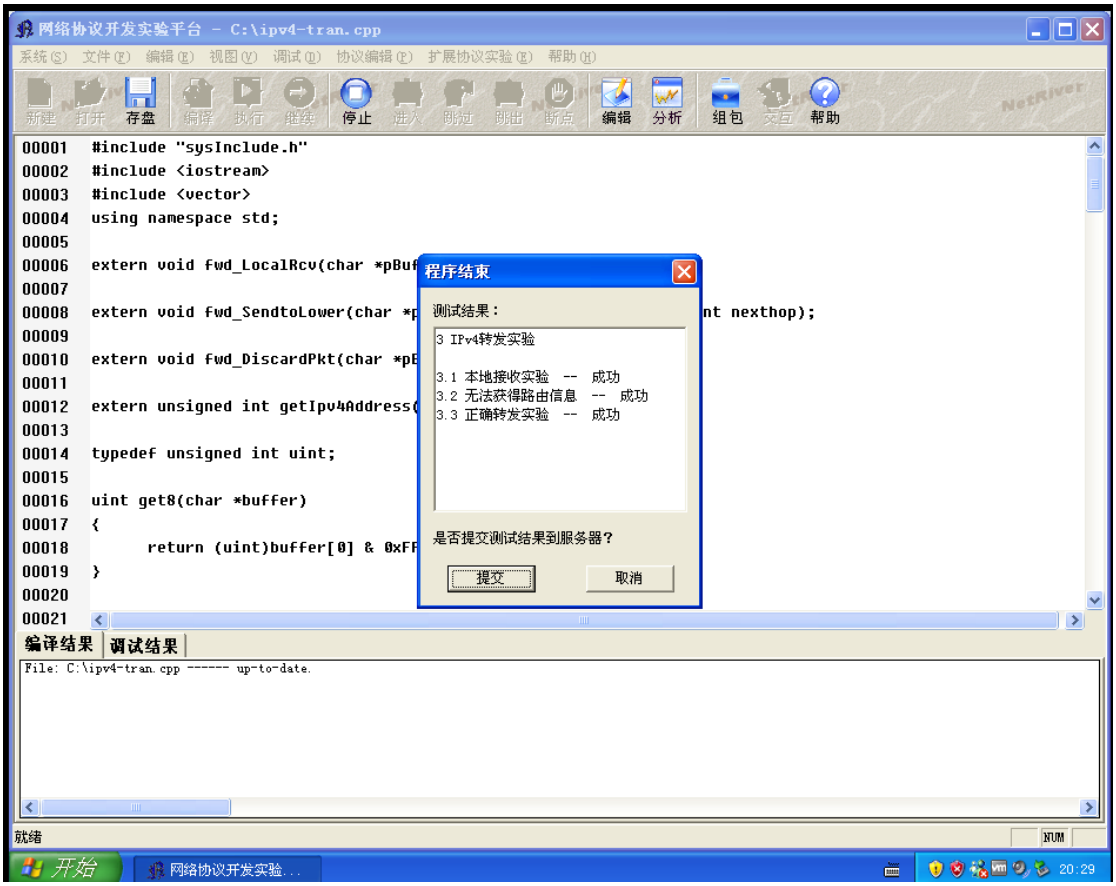
3. 如何提高转发效率

a. 路由表存储结构由线性结构改为树形结构，可采用 B+树，提高匹配效率。

b. 每次在转发分组时，都要检测数据合法性，计算校验和等操作，我们可以把这些分成几个阶段，然后采用流水线的机制，并行的操作几个分组，就像体系结构中所讲的指令流水一样，这样就能提高转发效率。当然这由硬件来实现。

c. 经过路由器的前后分组间的相关性很大，具有相同目的地址和源地址的分组往往连续到达，快速转发过程中，只需对一组具有相同目的地址和源地址的分组的前几个分组进行传统的路由转发处理，并把成功转发的分组的目的地址、源地址和下一网关地址（下一路由器地址）放人转发缓存中。当其后的分组要进行转发时，茵先查看转发缓存，如果该分组的目的地址和源地址与转发缓存中的匹配，则直接根据转发缓存中的下一网关地址进行转发，而无须经过传统的复杂操作，大大减轻了路由器的负担，达到了提高路由器吞吐量的目标。

4. 实验结果



四、实验心得

分组转发是路由器最重要的功能。分组转发的依据是路由信息，依次将目的地址不同的分组发送到相应的接口上，逐跳转发，并最终到达目的主机，通过此次实验，我对路由器的分组转发功能有了更加切实的体验，也对其匹配转发的过程更加的清楚了。

五、源代码

```
#include "sysInclude.h"
#include<stdlib>

using std::vector;
using std::cout;
```

```
// 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
{
unsigned int destIP;
unsigned int mask;
unsigned int masklen;
unsigned int nexthop;
};
vector table;

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

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

int stud_fwd_deal(char *pBuffer, int length)
{

int IHL = pBuffer[0] & 0xf;

int TTL = (int)pBuffer[8];
int headerChecksum = ntohl(*(unsigned short*)(pBuffer+10));
int destIP = ntohl(*(unsigned int*)(pBuffer+16));



if(destIP == getIpv4Address())
{
fwd_LocalRcv(pBuffer, length);
return 0;
}

if(TTL <= 0)
{
fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR);
return 1;
}

bool isMatch = false;
```

```
unsigned int longestMatchLen = 0;
int bestMatch = 0;

for(int i = 0; i < table.size(); i ++)
{
if(table[i].masklen > longestMatchLen && table[i].destIP == (destIP & table[i].mask))
{
bestMatch = i;
isMatch = true;
longestMatchLen = table[i].masklen;
//cout << "find one" << endl;
}
}

if(isMatch)
{
char *buffer = new char[length];
memcpy(buffer,pBuffer,length);
buffer[8]--; //TTL - 1
int sum = 0;
unsigned short int localCheckSum = 0;
for(int j = 0; j < 2 * IHL; j ++)
{
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, table[bestMatch].nexthop);
        return 0;
        }
        else
        {
        fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE);
        return 1;
        }
        return 1;
}
```