哈爾濱工業大學
Harbin Institute of Technology

计算机网络
课程实验报告

实验名称	IPv4 分组收发/转发实验				
姓名	刘天瑞		院系	未来技术学院	
班级	20W0362		学号	7203610121	
任课教师	刘亚维		指导教师	刘亚维	
实验地点	G001		实验时间	2022.10.19	
实验课表现	出勤、表现得分(10)		实验报告得分(40)	实验总分	
	操作结果得分(50)				
教师评语					

计算机科学与技术学院 SINCE 1956...
School of Computer Science and Technology

实验目的：

（注：实验报告模板中的各项内容仅供参考，可依照实际实验情况进行修改。）

本次实验的主要目的。

答：

1. 通过设计实现主机协议栈中的 IPv4 协议，让学生深入了解网络层协议的基本原理，学习 IPv4 协议基本的分组接收和发送流程。

2. 使学生初步接触互联网协议栈的结构和计算机网络实验系统，为后面进行更为深入复杂的实验奠定良好的基础。

3. 使学生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。

实验内容：

概述本次实验的主要内容，包含的实验项等。

答：

1. 实现 IPv4 分组的基本接收处理功能，对于接收到的 IPv4 分组，检查目的地址是否为本地地址，并检查 IPv4 分组头部中其它字段的合法性。提交正确的分组给上层协议继续处理，丢弃错误的分组并说明错误类型。

2. 实现 IPv4 分组的封装发送根据上层协议所提供的参数，封装 IPv4 分组，调用系统提供的发送接口函数将分组发送出去。

3. 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。

4. 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”。

5. 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。

实验过程：

以文字描述、实验结果截图等形式阐述实验过程，必要时可附相应的代码截图或以附件形式提交。

1.了解实验相关基础知识

1) IPv4 协议

IPv4 协议是互联网的核心协议，它保证了网络节点（包括网络设备和主机）在网络层能够按照标准协议互相通信。IPv4 地址唯一标识了网络节点和网络的连接关系。在日常使用的计算机的主机协议栈中，IPv4 协议必不可少，因为它能够接收网络中传送给本机的分组，同时也能根据上层协议的要求将报文封装为 IPv4 分组发送出去。

在主机协议栈中，IPv4 协议主要承担辨别和标识源 IPv4 地址和目的 IPv4 地址的功能，一方面接收处理发送给自己的分组，另一方面根据应用需求填写目的地址并将上层报文封装发送。IPv4 地址可以在网络中唯一标识一台主机，因而在相互通信时填写在 IPv4 分组头部中的 IPv4 地址就起到了标识源主机和目的主机的作用。

在两个主机端系统通信的环境中，网络的拓扑可以简化为两台主机直接相连，中间的具体连接方式可以抽象为一条简单链路，如下图所示：

图 1 主机端系统通信环境抽象

2) IPv4 协议的分组转发

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，以此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。在实验中，需要按照路由器协议栈的 IPv4 协议功能进行设计实现，接收处理所有收到的分组（而不只是目的地址为本机地址的分组），并根据分组的 IPv4 目的地址结合相关的路由信息，对分组进行转发、接收或丢弃操作。

3) IPv4 报文格式

IPv4 报文格式如下：

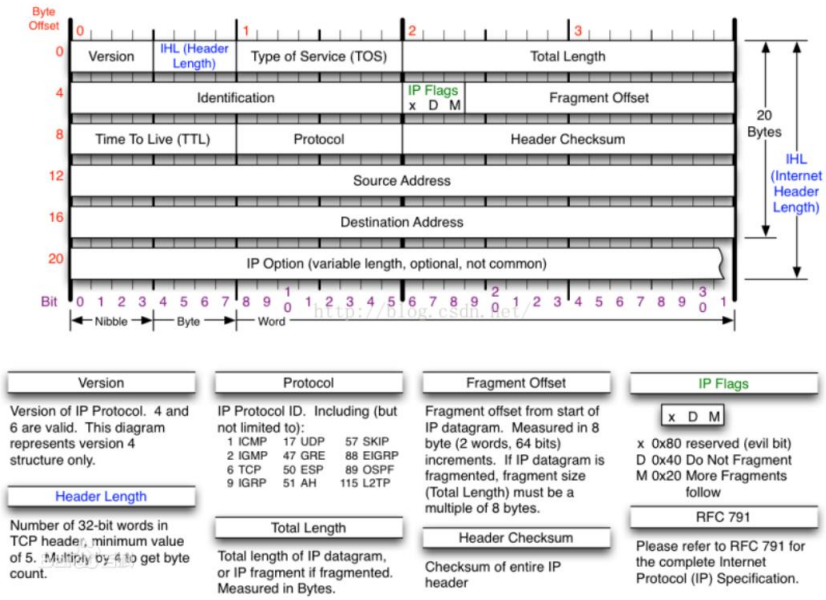


图 2 IPv4 报文具体格式

以下具体说明了 IPv4 报文段中比较重要的部分：

(i)Version（版本）

字段长度为 4 位，标识了数据包的 IP 版本号。0100 表示 IP 版本为 4，0110 表示 IP 版本为 6（其他所有版本号仅作为“历史产物”）。

(ii)IHL（头部长度）

字段长度为 4 位（单位为 4 个字节）,IP 报头的最小长度为 20 个八位组，最大可以扩展到 60 个八位组，通过这个字段可以描述 32 位字长的最大长度。

(iii)Total Length（总长度）

字段长度为 16 位（单位为一个八位组），其中包括 IP 报头。接受者用总长度减去 IP 报头长度，就可以确定数据包数据有效载荷的大小（用十进制表示最大到 65535 个）。

(iv)TTL（生存时间）

字段长度为 8 位，以前为时间，现在为跳数。传输时，每台路由器都会将 TTL 值减一，到 0 会向源点发送错误信息（用于防止环形成以及 trace 追踪工具）。

(v)Header Checksum（头部校验）

是针对 IP 报头的纠错字段。校验和不计算被封装的数据，UDP、TCP 和 ICMP 都有各自的校验和。

4) NetRiver 平台的使用

NetRiver 平台的整体流程如下：

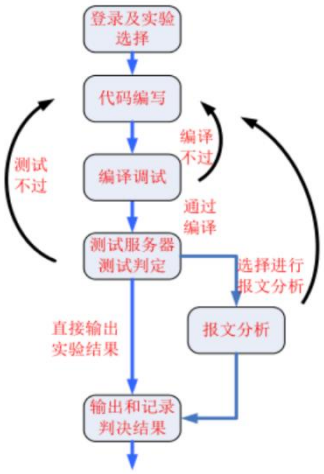


图 3 NetRiver 平台使用流程图

首先，我在 VMware 平台安装了 WindowsXP 虚拟机，进入客户端软件后，选择要进行的实验内容与测试用例，新建文件，复制我们写好的代码进行编译。编译完成后，执行文件，等待一段时间后，便可得到测试结果。

2.分析程序设计思路

1) IPv4 报文接收与发送的程序设计思路

在发送 IPv4 报文的过程中，根据所传参数（如数据大小），来确定分配的存储空间的大小并申请分组的存储空间。按照 IPv4 协议标准填写 IPv4 分组头部各字段，标识符（Identification）字段可以使用一个随机数来填写（注意：部分字段内容需要通过 ntol()函数转换成网络字节顺序）。在完成 IPv4 分组的封装后，调用 ip_SendtoLower()接口函数将数据报文发送给下层协议，最终将分组发送到网络中。

发送函数的程序流程图如下：

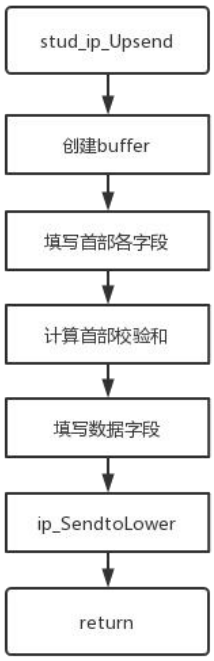


图 4 发送函数程序流程图

在接收 IPv4 报文时，首先要检查接收到的 IPv4 分组头部的字段，包括版号(Version)、头部长度(IP Head length)、生存时间(TTL)以及头校验和(Header checksum)。字段是否符合要求。如果不符合要求，则判定为出错分组，调用 ip_DiscardPkt()函数，以特定的错误类型进行丢弃。随后，要检查 IPv4 分组是否应该由本机接收。如果分组的目的地址是本机地址或广播地址，则说明此分组是发送给本机的，进行对上层协议类型的提取，并调用 ip_SendtoUp()接口函数，将数据报交给上层协议进行后续接收处理；否则说明此 IP 报文虽然正确，但并非发送给本机，需要调用 ip_DiscardPkt()函数丢弃，并说明错误类型。

接收函数的程序流程图如下：

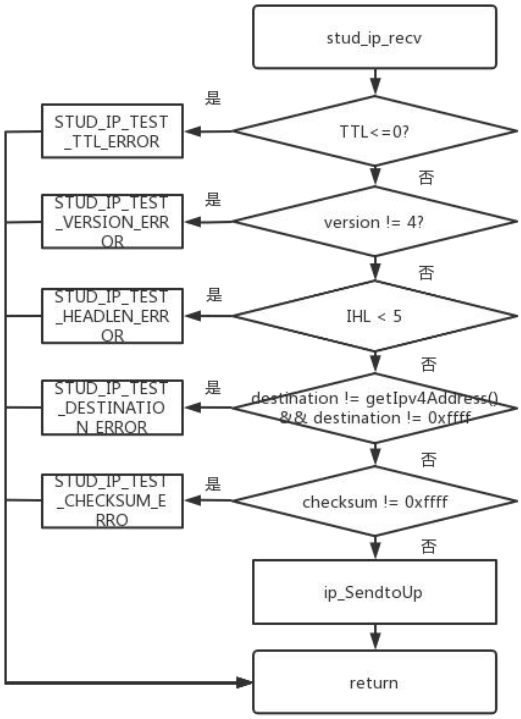


图 5 接收函数程序流程图

2) IPv4 分组转发程序的设计思路

(i)路由表初始化：在 IPv4 分组转发程序中，设定如下的数据结构：

```
17  // 构造路由表结构体
18  struct routingTable
19  {
20      unsigned int dstIP;   // 目的IP
21      unsigned int mask;    // 掩码
22      unsigned int masklen; // 掩码长度
23      unsigned int nexthop; // 下一跳
24  };
```

图 6 定义路由表结构体的部分代码

路由表初始化的程序很简单，即简单对路由表进行清空操作。

(ii)在路由表初始化之后，路由表需要增加路由表项，具体过程为：从 newTableItem 结构中取得 dest, masklen, nextIP，转为网络字节序之后经过处理，构建结构体 route，并添加到 vector 中。路由表增加路由表项的程序框图如下图所示：

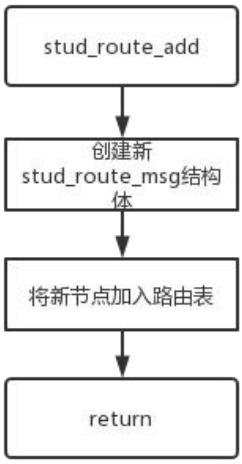


图 7 向路由表增加路由表项的程序流程图

(iii)路由转发：在 stud_fwd_deal ()函数中，需要完成下列分组接收处理步骤：

1. 查找路由表。根据相应路由表项的类型来确定下一步操作，错误分组调用函数进行丢弃，上交分组调用接口函数提交给上层协议继续处理，转发分组进行转发处理。值得注意的是，转发分组还要从路由表项中获取下一跳的 IPv4 地址。
2. 转发处理流程。对 IPv4 头部中的 TTL 字段减 1，重新计算校验和，然后调用下层接口进行发送处理。

路由转发的程序框图如下图所示：

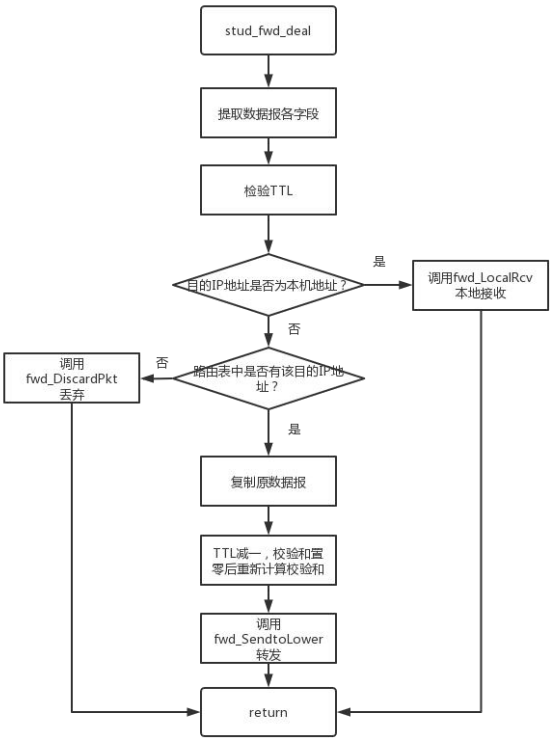


图 8 路由转发处理函数程序流程图

实验结果：

采用演示截图、文字说明等方式，给出本次实验的实验结果。

1. IPv4分组收发实验：

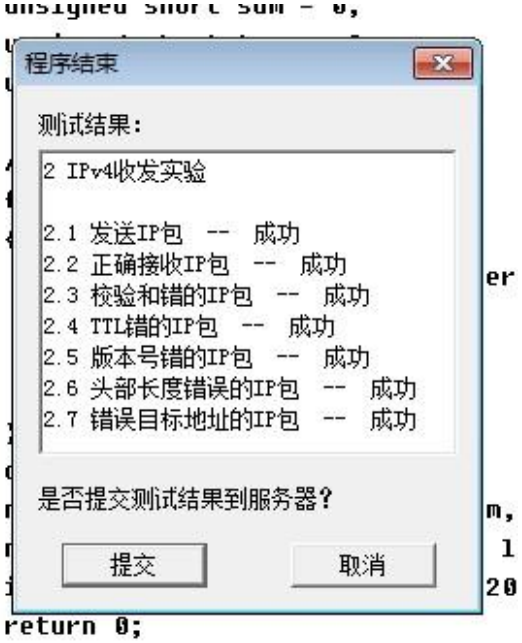


图9 IPv4分组收发实验测试结果

2. IPv4分组转发实验：

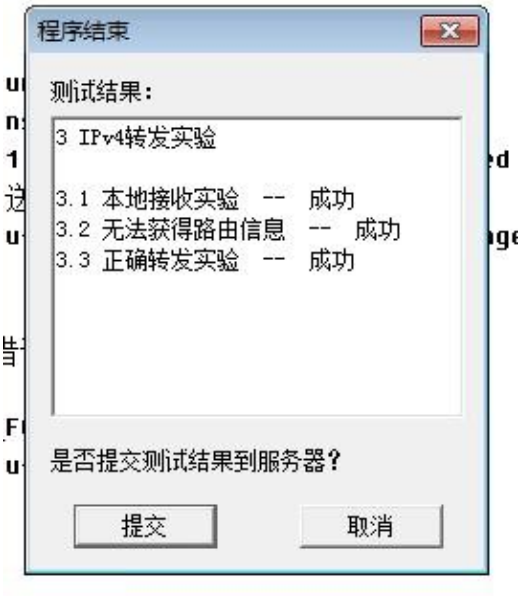
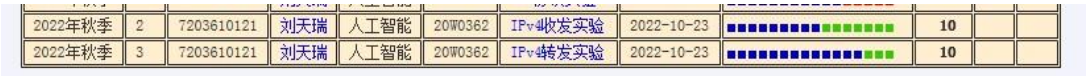


图10 IPv4分组转发实验测试结果

最终实验成绩结果如下所示：



2022年秋季	2	7203610121	刘天瑞	人工智能	20W0362	IPv4收发实验	2022-10-23		10		
2022年秋季	3	7203610121	刘天瑞	人工智能	20W0362	IPv4转发实验	2022-10-23		10		

图11 我的实验成绩

问题讨论：

对实验过程中的思考问题进行讨论或回答。

答：

1. 在IP分组转发实验中，若存在大量分组情况下如何提高转发效率，展开讨论如下：

(i) 首先最直接一点是改进存储结构，本次实验中使用的线性结构，查询时间复杂度为O(n)。若将查询时间优化到O(logn)，可以根据IP目的地址进行有序存储，查询时使用二分查找；若希望查询时间复杂度为O(1)，可以将路由表设置为一个哈希表，达到空间换时间的目的。

(ii) 因为存在大量分组，从硬件层面入手，所以可以采用并行转发。所有的检测、匹配等过程都可以并行处理。

(iii) 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，缓存分组，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，减轻了路由器的负担，提高路由器吞吐量。

2. 发送IPv4分组计算头部校验和时总是出错，展开讨论如下：

因为IPv4分组需要转发给下层然后发送出去，而每次计算校验和时使用unsigned short类型即16位进行计算，需要更改为网络字节序再进行计算，否则在接收端计算校验和时会总是出错。

心得体会：

结合实验过程和结果给出实验的体会和收获。

答：

1. 经过本次实验，我对IPv4的报文结构与检测、转发等功能有了更为深入的认识，也对路由表的建立、维护和工作过程更加了解；
2. 同时我还学习到了IPv4的数据包头部字段内容和格式，IP分组各字段计算方法、接收转发的流程，路由表的建立和路由信息添加、路由器如何为分组选择路由并逐跳地发送到目的主机以及检索，深入了解了网络层协议的基本原理。
3. 深入理解了路由表的数据结构，理解路由器是如何根据路由表对分组进行转发的。

附录

实验源代码如下所示：

(i) IPv4分组收发实验：

```
/*
* THIS FILE IS FOR IP RECEIVE TEST
*/
#include "sysInclude.h"
#include <stdio.h>
#include <string.h>

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

extern void ip_SendtoLower(char *pBuffer, int length);

extern void ip_SendtoUp(char *pBuffer, int length);

extern unsigned int getIpv4Address();
```

```
}

int stud_ip_recv(char *pBuffer, unsigned short length)
{
  int errorType = 0;

  // 检测 version 错误
  int version = pBuffer[0] >> 4;
  if (version != 4)
  {
    errorType = STUD_IP_TEST_VERSION_ERROR;
    ip_DiscardPkt(pBuffer, errorType);
    return 1;
  }

  // 检测 headerLength 错误
  int headerLength = pBuffer[0] & 0xF;
  if (headerLength < 5)
  {
    errorType = STUD_IP_TEST_HEADLEN_ERROR;
    ip_DiscardPkt(pBuffer, errorType);
    return 1;
  }

  // 检测 TTL 错误
  int ttl = (unsigned short)pBuffer[8];
  if (ttl <= 0)
  {
    errorType = STUD_IP_TEST_TTL_ERROR;
    ip_DiscardPkt(pBuffer, errorType);
    return 1;
  }

  // 检测目的地址错误
  unsigned int destIP = ntohl(*(unsigned int *)(pBuffer + 16));
  unsigned int localIP = getIpv4Address();
  if (destIP != 0xFFFFFFFF && destIP != localIP)
  {
    errorType = STUD_IP_TEST_DESTINATION_ERROR;
    ip_DiscardPkt(pBuffer, errorType);
    return 1;
  }

  // 检测 checkSum 错误
```

```
  unsigned short sum = 0;
  unsigned short temp = 0;
  for (int i = 0; i < headerLength * 2; i++)
  {
    temp = ((unsigned char)pBuffer[i * 2] << 8) + (unsigned char)pBuffer[i * 2 + 1]; //
<<8 表示其做高 2 位
    if (sum + temp > 0xFFFF) // 若计算结果 > 0xFFFF，则将高 16 位加在低 16 位上
      sum += 1;
    sum += temp;
  }
  if (sum != 0xFFFF) // 若计算结果 ≠ FFFF，说明数据报发生错误
  {
    errorType = STUD_IP_TEST_CHECKSUM_ERROR;
    ip_DiscardPkt(pBuffer, errorType);
    return 1;
  }

  // 无错误显示，则表示成功接受，再上传给上层
  ip_SendtoUp(pBuffer, length);
  return 0;
}

int stud_ip_Upsend(char *pBuffer, unsigned short len, unsigned int srcAddr, unsigned int
dstAddr, byte protocol, byte ttl)
{
  char *IPBuffer = (char *)malloc((len + 20) * sizeof(char)); // IPBuffer 的每一位对应
着一个字节，20 是报文头的字节数
  memset(IPBuffer, 0, len + 20);
  IPBuffer[0] = 0x45; // 构造版本号与头长度位
  unsigned short totalLength = htons(len + 20); // IPv4 报文分组总长度
  memmove(IPBuffer + 2, &totalLength, 2); // 构造报文头的 totalLength 部分
  IPBuffer[8] = ttl; // 构造 TTL
  IPBuffer[9] = protocol; // 构造协议号
  unsigned int src = htonl(srcAddr);
  unsigned int dst = htonl(dstAddr);
  memmove(IPBuffer + 12, &src, 4); // 源 IP 地址
  memmove(IPBuffer + 16, &dst, 4); // 目的 IP 地址
  unsigned short sum = 0;
  unsigned short temp = 0;
  unsigned short checkSum = 0;

  //计算 checksum
  for (int i = 0; i < 10; i++)
  {
```

```
    temp = ((unsigned char)IPBuffer[i * 2] << 8) + (unsigned char)IPBuffer[i * 2 + 1];
// <<8 表示其做高 2 位
    if (sum + temp > 0xFFFF) // 若结果>0xFFFF，则将高 16 位加在低 16 位上
      sum += 1;
    sum += temp;
  }
  checkSum = htons(0xFFFF - sum); // 取反，得到最终的 checkSum
  memmove(IPBuffer + 10, &checkSum, 2); // 构造报文头的 checkSum 部分
  memmove(IPBuffer + 20, pBuffer, len); // 构造报文的实际内容
  ip_SendtoLower(IPBuffer, len + 20);   // 向下一层协议发送
  return 0;
}
```

(ii) IPv4分组转发实验：

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

// 构造路由表结构体
struct routingTable
{
    unsigned int dstIP;   // 目的 IP
    unsigned int mask; // 掩码
    unsigned int masklen; // 掩码长度
    unsigned int nexthop; // 下一跳
};

// 创建路由表实例
vector<routingTable> routing_table; // 路由表

void stud_Route_Init()
{
    routing_table.clear();
```

```
    return;
}

void stud_route_add(stud_route_msg *proute)
{
    routingTable rt;
    rt.dstIP = ntohl(proute->dest);
    rt.mask = (1 << 31) >> (ntohl(proute->masklen) - 1);
    rt.masklen = ntohl(proute->masklen); //将一个无符号长整形数从网络字节顺序转换为主机字节顺序
    rt.nexthop = ntohl(proute->nexthop);
    routing_table.push_back(rt);
    return;
}

int stud_fwd_deal(char *pBuffer, int length)
{
    int errorType = 0; // 错误编号
    int ttl = pBuffer[8]; // TTL
    int headerLength = pBuffer[0] & 0xF; // 数据报头部长度
    int dstIP = ntohl(*(unsigned int *)(pBuffer + 16)); // 目的 IP 地址
    if (dstIP == getIpv4Address()) // 判断分组地址与本机地址是否相同，若相同，则直接交付报文
    {
        fwd_LocalRcv(pBuffer, length); // 向上层协议交付 IP 分组
        return 0;
    }
    if (ttl <= 0) // 若 TTL < 0，则将该分组丢弃
    {
        errorType = STUD_FORWARD_TEST_TTLERROR;
        fwd_DiscardPkt(pBuffer, errorType);
        return 1;
    }

    // 进行路由查找
    bool match = false;   // 是否完成匹配
    unsigned int maxLen = 0; // 最长前缀匹配的长度
    int longestNum = 0;   // 最长前缀匹配的序号
    // 判断是否存在匹配
    for (int i = 0; i < routing_table.size(); i++)
    {
        if (routing_table[i].masklen > maxLen && routing_table[i].dstIP == (dstIP & routing_table[i].mask)) // 按照最长前缀原则匹配到下一跳，记录相关数据
        {
```

```
            match = true;
            longestNum = i;
            maxLen = routing_table[i].masklen;
        }
    }

    if (match) // 匹配成功，发送至下一跳
    {
        int sum = 0;
        unsigned short int newCheckSum = 0;
        char *buffer = new char[length];
        memmove(buffer, pBuffer, length);
        buffer[8]--; // 让 TTL - 1
        for (int j = 1; j < 2 * headerLength + 1; j++)
        {
            if (j != 6)
            {
                sum += (buffer[(j - 1) * 2] << 8) + (buffer[(j - 1) * 2 + 1]);
                sum %= 65535;
            }
        }
        // 重新计算 checksum
        newCheckSum = htons(~(unsigned short int)sum);
        memmove(buffer + 10, &newCheckSum, sizeof(unsigned short));
        // 向下一层协议发送，将数据传送至下一跳的路由
        fwd_SendtoLower(buffer, length, routing_table[longestNum].nexthop);
        return 0;
    }
    else // 若匹配失败，则进行错误处理
    {
        errorType = STUD_FORWARD_TEST_NOROUTE;
        fwd_DiscardPkt(pBuffer, errorType);
        return 1;
    }
}
```