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 收发和转发实验				
姓名	袁野		院系	计算学部计算机科学与技术专业	
班级	1903102		学号	1190200122	
任课教师	刘亚维		指导教师	刘亚维	
实验地点	格物 207		实验时间	2021.11.14	
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：

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。在我们日常使用的计算机的主机协议栈中，IPv4 协议必不可少，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。

IPv4 分组收发实验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。

另外，通过本实验，学生可以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。

IPv4 分组转发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。

IPv4 分组转发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。

实验内容：

在 IPv4 分组收发实验中，我们需要：

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。

在 IPv4 分组转发实验中，我们需要：

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”。

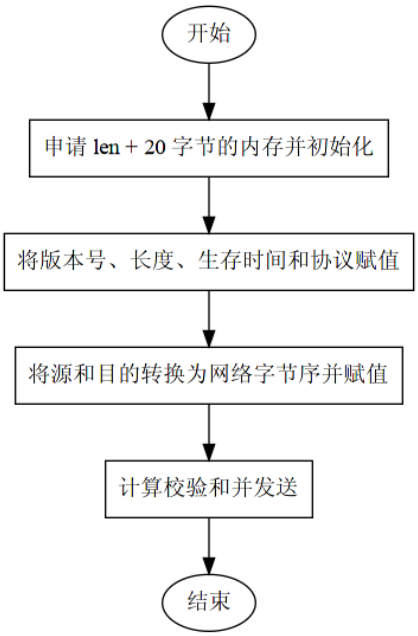
3) IPv4 分组的转发。

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。

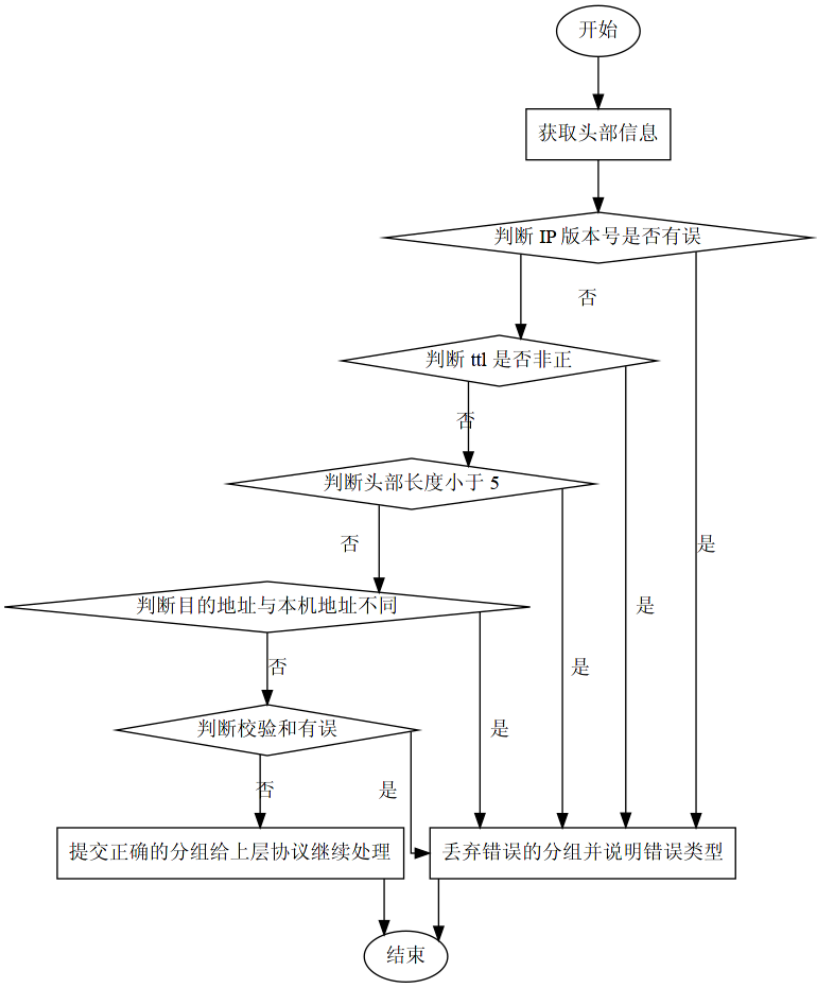
实验过程：

1. IPv4 分组收发实验

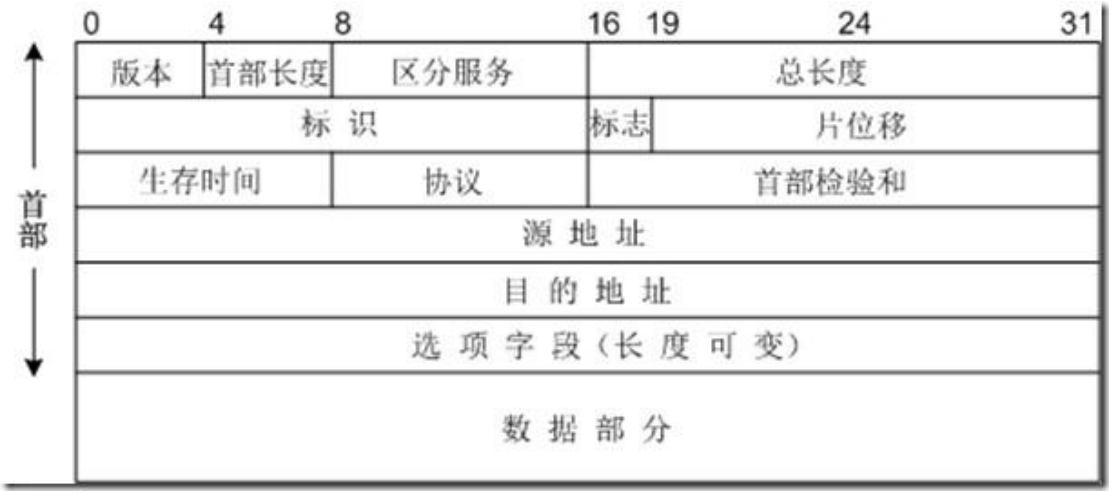
发送函数流程图如下：



接收函数流程图如下：



1.1 IPV4 数据包的头部信息



1.2 接收检验

我们可以利用位运算来获取版本等头部信息。版本号和头部长度的信息在第一个字节，然后通过位运算来分离两个信息。生存时间字段在 IPv4 数据包的头部第八个字节。源地址和目的地址在 IPv4 数据包的第十二和第十六个字节开始（获取时需要转换网络字节）。我们需要检验版本号是否为 4，头部长度是否大于 5（因为最少的 IPv4 数据包的头部信息为 20 字节），生存时间是否为正（如果 TTL 非正则说明其已经过期），目的地址是否为本机地址。

```
unsigned short version = pBuffer[0] >> 4;                          // 版本号
unsigned short head_length = pBuffer[0] & 0xf;                     // 头部长度
unsigned short ttl = pBuffer[8];                                   // 生存时间
unsigned short checksum = ntohs(*(unsigned short *)(pBuffer + 10)); // 校验和
unsigned int dest = ntohl(*(unsigned int *)(pBuffer + 16));        // 目的地址

if (version != 4) {
    // IP 版本号错
    ip_DiscardPkt(pBuffer, STUD_IP_TEST_VERSION_ERROR);
    return 1;
}
else if (ttl <= 0) {
    // TTL 值非正
    ip_DiscardPkt(pBuffer, STUD_IP_TEST_TTL_ERROR);
    return 1;
}
else if (head_length < 5) {
    //头部长度错
    ip_DiscardPkt(pBuffer, STUD_IP_TEST_HEADLEN_ERROR);
    return 1;
}
else if (dest != getIpv4Address()) {
    //目的地址错
    ip_DiscardPkt(pBuffer, STUD_IP_TEST_DESTINATION_ERROR);
    return 1;
}
```

头部校验和字段在 IPv4 数据包的头部第十个字节，并且根据之前进行计算时取反的性质，将所有的头部信息进行与 checksum 生成时相同的计算步骤，得到的结果应该为全 1；否则说明头部校验和错误。

```
unsigned long sum = 0;
for (int i = 0; i < 20; i += 2) {
    sum += (unsigned char)buffer[i] << 8;
    sum += (unsigned char)buffer[i + 1];
}
unsigned short l_word = sum & 0xffff;
unsigned short h_word = sum >> 16;
unsigned short checksum = l_word + h_word;
checksum = ~checksum;

unsigned short header_checksum = htons(checksum);
memcpy(buffer + 10, &header_checksum, sizeof(unsigned short));
memcpy(buffer + 20, pBuffer, len);
ip_SendtoLower(buffer, ip_length);
return 0;
```

1.3 版本号错误

未知版本号3：

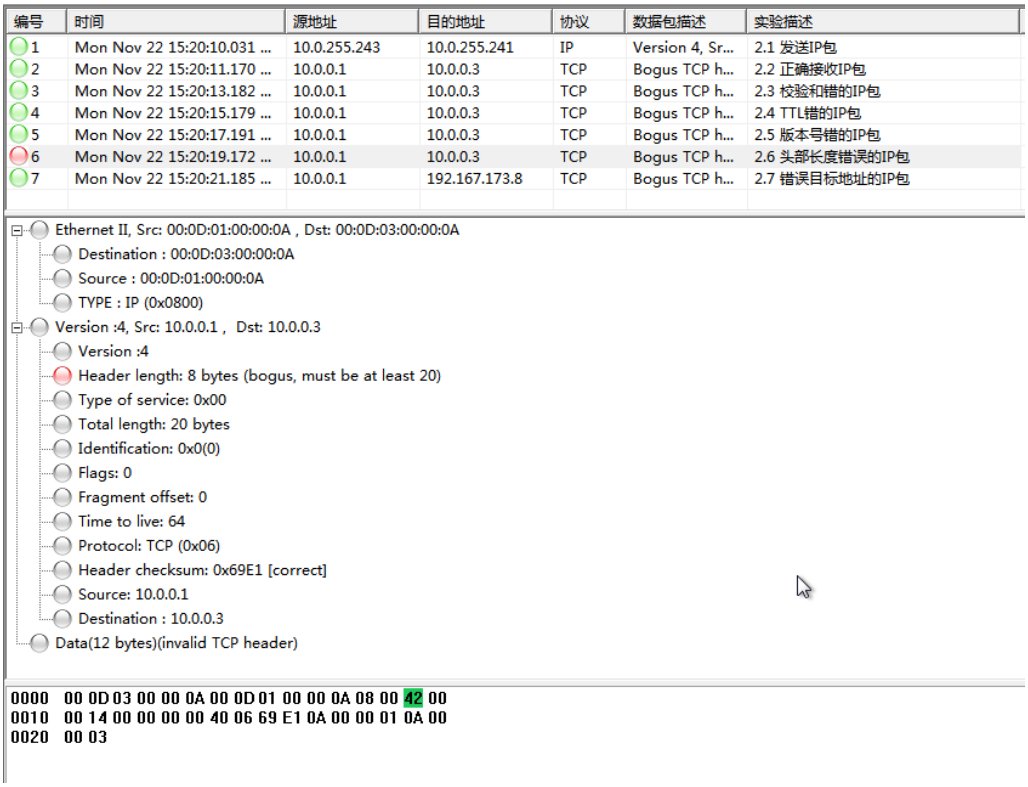
编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Mon Nov 22 15:20:10.031 ...	10.0.255.243	10.0.255.241	IP	Version 4, Sr...	2.1 发送IP包
2	Mon Nov 22 15:20:11.170 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Mon Nov 22 15:20:13.182 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Mon Nov 22 15:20:15.179 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Mon Nov 22 15:20:17.191 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Mon Nov 22 15:20:19.172 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.6 头部长度错误的IP包
7	Mon Nov 22 15:20:21.185 ...	10.0.0.1	192.167.173.8	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

```
Ethernet II, Src: 00:0D:01:00:00:0A , Dst: 00:0D:03:00:00:0A
    Destination : 00:0D:03:00:00:0A
    Source : 00:0D:01:00:00:0A
    TYPE : IP (0x0800)
Version :3, Src: 10.0.0.1 ,  Dst: 10.0.0.3
    Version :3 (Unknown Version)
    Header length: 20 bytes
    Type of service: 0x00
    Total length: 20 bytes
    Identification: 0x0(0)
    Flags: 0
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (0x06)
    Header checksum: 0x76E1 [correct]
    Source: 10.0.0.1
    Destination : 10.0.0.3
```

```
0000   00 0D 03 00 00 0A 00 0D 01 00 00 0A 08 00 35 00
0010   00 14 00 00 00 00 40 06 76 E1 0A 00 00 01 0A 00
0020   00 03
```

1.4 头部长度错误

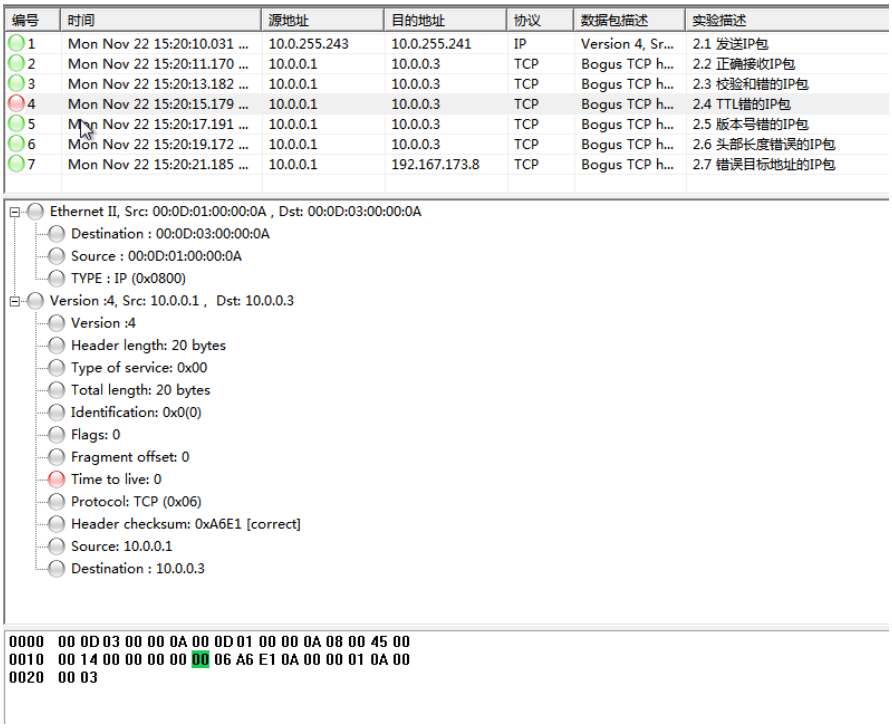
这里我们将头部长度错误的修改为八个字节



编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Mon Nov 22 15:20:10.031 ...	10.0.255.243	10.0.255.241	IP	Version 4, Sr...	2.1 发送IP包
2	Mon Nov 22 15:20:11.170 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Mon Nov 22 15:20:13.182 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Mon Nov 22 15:20:15.179 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Mon Nov 22 15:20:17.191 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Mon Nov 22 15:20:19.172 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.6 头部长度错误的IP包
7	Mon Nov 22 15:20:21.185 ...	10.0.0.1	192.167.173.8	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

1.5 TTL错误

TTL错误地为0



编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Mon Nov 22 15:20:10.031 ...	10.0.255.243	10.0.255.241	IP	Version 4, Sr...	2.1 发送IP包
2	Mon Nov 22 15:20:11.170 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Mon Nov 22 15:20:13.182 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Mon Nov 22 15:20:15.179 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Mon Nov 22 15:20:17.191 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Mon Nov 22 15:20:19.172 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.6 头部长度错误的IP包
7	Mon Nov 22 15:20:21.185 ...	10.0.0.1	192.167.173.8	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

1.6 目标地址错误

编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Mon Nov 22 15:20:10.031 ...	10.0.255.243	10.0.255.241	IP	Version 4, Sr...	2.1 发送IP包
2	Mon Nov 22 15:20:11.170 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Mon Nov 22 15:20:13.182 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Mon Nov 22 15:20:15.179 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Mon Nov 22 15:20:17.191 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Mon Nov 22 15:20:19.172 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.6 头部长度错误的IP包
7	Mon Nov 22 15:20:21.185 ...	10.0.0.1	192.167.173.8	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

- Ethernet II, Src: 00:0D:01:00:00:0A , Dst: 00:0D:03:00:00:0A
  - Destination : 00:0D:03:00:00:0A
  - Source : 00:0D:01:00:00:0A
  - TYPE : IP (0x0800)
- Version :4, Src: 10.0.0.1 , Dst: 192.167.173.8
  - Version :4
  - Header length: 20 bytes
  - Type of service: 0x00
  - Total length: 20 bytes
  - Identification: 0x0(0)
  - Flags: 0
  - Fragment offset: 0
  - Time to live: 64
  - Protocol: TCP (0x06)
  - Header checksum: 0x0334 [correct]
  - Source: 10.0.0.1
  - Destination : 192.167.173.8

```
0000  00 0D 03 00 00 0A 00 0D 01 00 00 0A 08 00 45 00
0010  00 14 00 00 00 00 40 06 03 34 0A 00 00 01 C0 A7
0020  AD 08
```

1.7 校验和错误

编号	时间	源地址	目的地址	协议	数据包描述	实验描述
1	Mon Nov 22 15:20:10.031 ...	10.0.255.243	10.0.255.241	IP	Version 4, Sr...	2.1 发送IP包
2	Mon Nov 22 15:20:11.170 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.2 正确接收IP包
3	Mon Nov 22 15:20:13.182 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.3 校验和错的IP包
4	Mon Nov 22 15:20:15.179 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.4 TTL错的IP包
5	Mon Nov 22 15:20:17.191 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.5 版本号错的IP包
6	Mon Nov 22 15:20:19.172 ...	10.0.0.1	10.0.0.3	TCP	Bogus TCP h...	2.6 头部长度错误的IP包
7	Mon Nov 22 15:20:21.185 ...	10.0.0.1	192.167.173.8	TCP	Bogus TCP h...	2.7 错误目标地址的IP包

- Ethernet II, Src: 00:0D:01:00:00:0A , Dst: 00:0D:03:00:00:0A
  - Destination : 00:0D:03:00:00:0A
  - Source : 00:0D:01:00:00:0A
  - TYPE : IP (0x0800)
- Version :4, Src: 10.0.0.1 , Dst: 10.0.0.3
  - Version :4
  - Header length: 20 bytes
  - Type of service: 0x00
  - Total length: 20 bytes
  - Identification: 0x0(0)
  - Flags: 0
  - Fragment offset: 0
  - Time to live: 64
  - Protocol: TCP (0x06)
  - Header checksum: 0x00C8[incorrect, should be 0x2E19]
  - Source: 10.0.0.1
  - Destination : 10.0.0.3

```
0000  00 0D 03 00 00 0A 00 0D 01 00 00 0A 08 00 45 00
0010  00 14 00 00 00 00 40 06 00 C8 0A 00 00 01 0A 00
0020  00 03
```

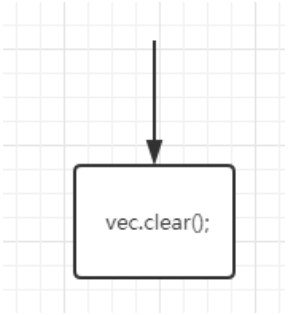
2. IPv4 分组转发实验

2.1 路由表初始化

void stud_route_init():

在 stud_route_init 函数中，我们需要初始化路由表。由于我们使用 STL 中的 vector 类维护路由表，所以我们只需要一条语句即可完成。

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

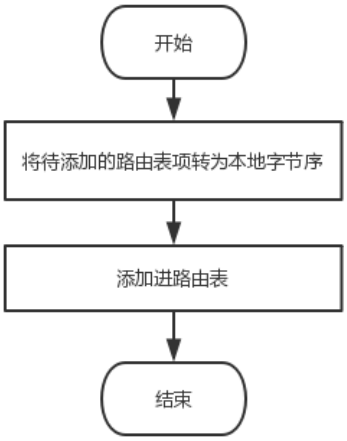


2.2 路由表增加

void stud_route_add(stud_route_msg *proute)

在 stud_route_add 函数中，我们只需要将网络字节转化为本地字节之后赋值即可。

```
void stud_route_add(stud_route_msg *proute) {
    routeTableItem newTableItem;
    newTableItem.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
    newTableItem.mask = (1 << 31) >> (ntohl(proute->masklen) - 1); //
    newTableItem.destIP = ntohl(proute->dest);
    newTableItem.nexthop = ntohl(proute->nexthop);
    m_table.push_back(newTableItem);
    return;
}
```



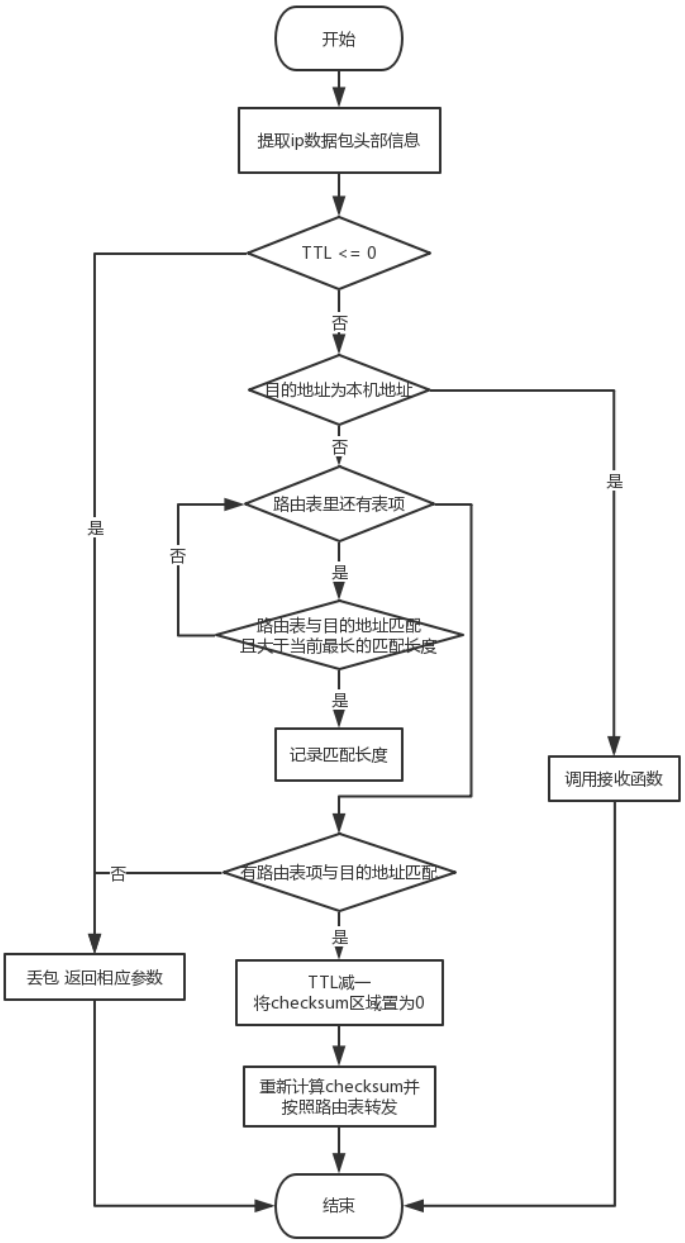
2.3 路由转发

int stud_fwd_deal(char *pBuffer, int length)

在 stud_fwd_deal() 函数中，需要完成下列分组接收处理步骤：

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 fwd_DiscardPkt() 进行丢弃，上交分组调用接口函数 fwd_LocalRcv() 提交给上层协议继续处理，转发分组进行转发处理。注意，转发分组还要从路由表项中获取下一跳的 IPv4 地址。

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 fwd_SendtoLower()进行发送处理。



2.4 新建数据结构

```
struct routeTableItem {
    unsigned int destIP;  //目的IP
    unsigned int mask;    // 掩码
    unsigned int masklen; // 掩码长度
    unsigned int nexthop; // 下一跳
};
```

2.5 在存在大量分组的情况下如何提高转发效率，如果代码中有相关功能实现，请给出具体原理说明

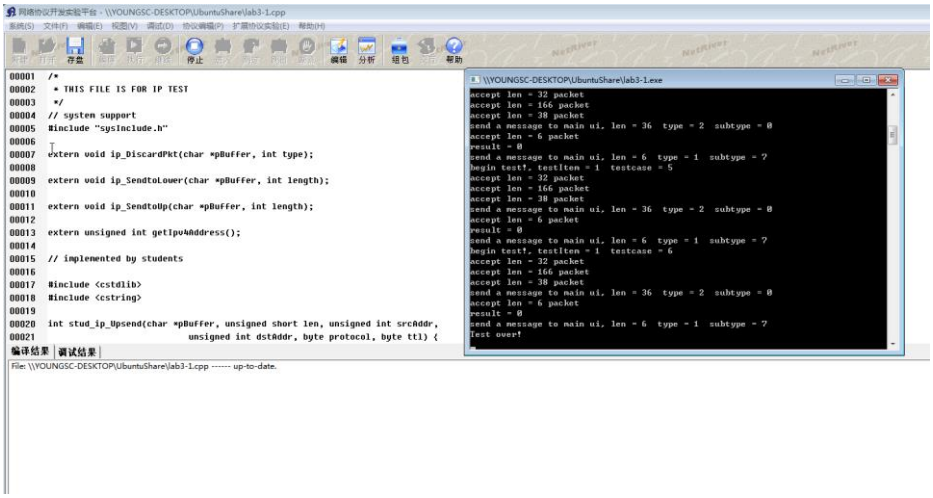
2.5.1 **树形结构匹配路由表项**：路由表存储结构由线性结构改为树形结构，提高匹配效率。

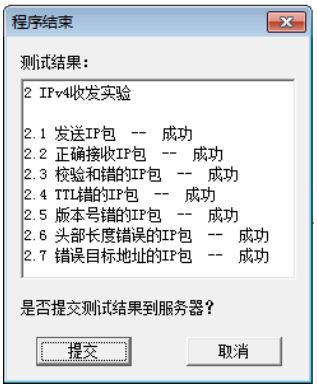
2.5.2 **并行检查**：每次在转发分组时，都要检测数据合法性，计算校验和等操作，我们可以并行操作，就能提高转发效率，由硬件来实现。

2.5.3 **缓存分组**：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。

验证过程以及实验结果：

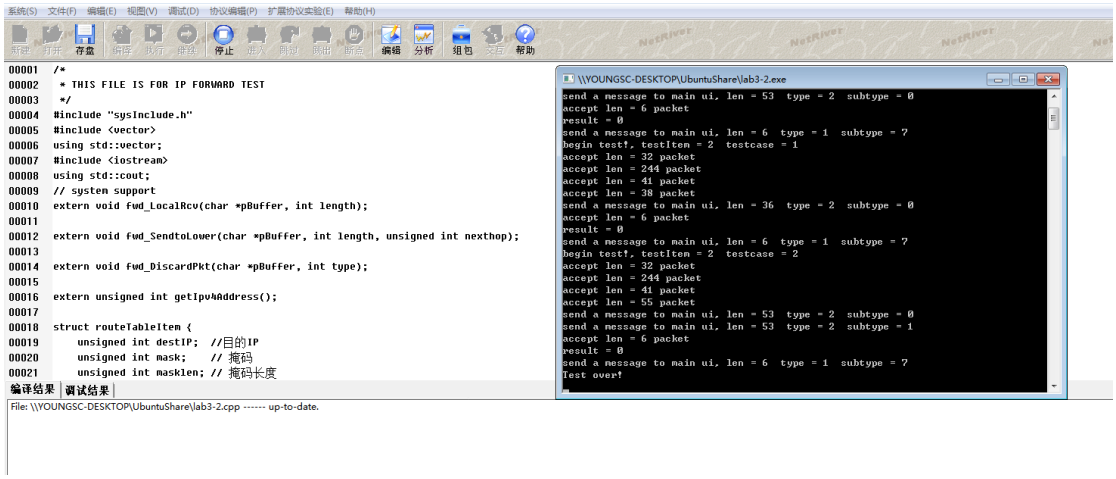
1. IPV4分组收发实验

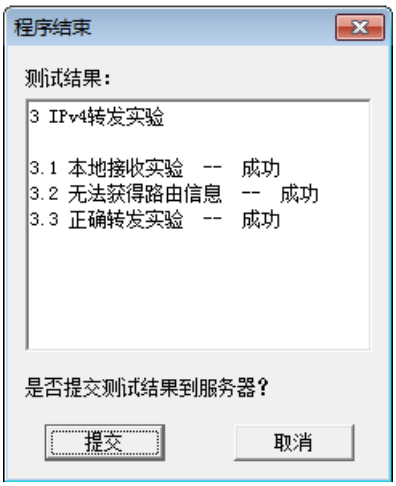




2021秋季	1	1190200122	袁野	计算学部	1903102	IPv4收发实验	2021-11-22		10		

2. IPV4分组转发实验





2021秋季	2	1190200122	袁野	计算学部	1903102	IPv4转发实验	2021-11-22		10		

问题讨论：

如何提高路由表的查找效率？

在IP匹配中，我们可以将IP地址看作等长的01字符串，因此我们可以给予最大匹配原则，将所有的IP地址储存为一棵01字典树，将需要查找的IP地址在字典树上由根向下进行匹配进行查找，这样我们可以由原先的将所有表项均进行查找并对比的线性复杂度，优化为只需要比较一个IP地址长度的常数复杂度，并能将储存空间进行优化，极大的提高其效率。

心得体会：

经过此次实验，我了解路由器是如何为分组选择路由，并逐跳地发送到目的主机。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，初步接触互联网协议栈的结构和计算机网络实验系统。

IPV4 分组收发实验

```
/*
  * THIS FILE IS FOR IP TEST
  */
// system support
#include "sysInclude.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();

#include <cstdlib>
#include <cstring>

int stud_ip_Upsend(char *pBuffer, unsigned short len, unsigned int srcAddr,
                   unsigned int dstAddr, byte protocol, byte ttl) {

    // 默认头部长度为 20 Bytes
    unsigned short ip_length = len + 20;

    char *buffer = (char *)malloc(ip_length * (sizeof(char)));
    memset(buffer, 0, ip_length * sizeof(char));
    buffer[0] = 0x45;       // 版本号和长度
    buffer[8] = ttl;    // 生存时间
    buffer[9] = protocol; // 协议

    // 转换为网络字节序
    unsigned short network_length = htons(ip_length);
    unsigned int src = htonl(srcAddr);
    unsigned int dest = htonl(dstAddr);
    memcpy(buffer + 2, &network_length, sizeof(unsigned short));
    memcpy(buffer + 12, &src, sizeof(unsigned int));
    memcpy(buffer + 16, &dest, sizeof(unsigned int));

    unsigned long sum = 0;
    for (int i = 0; i < 20; i += 2) {
        sum += (unsigned char)buffer[i] << 8;
        sum += (unsigned char)buffer[i + 1];
    }
```

```
    unsigned short l_word = sum & 0xffff;
    unsigned short h_word = sum >> 16;
    unsigned short checksum = l_word + h_word;
    checksum = ~checksum;

    unsigned short header_checksum = htons(checksum);
    memcpy(buffer + 10, &header_checksum, sizeof(unsigned short));
    memcpy(buffer + 20, pBuffer, len);
    ip_SendtoLower(buffer, ip_length);
    return 0;
}

int stud_ip_recv(char *pBuffer, unsigned short length) {

    unsigned short version = pBuffer[0] >> 4;                                    // 版本号
    unsigned short head_length = pBuffer[0] & 0xf;                               // 头部长度
    unsigned short ttl = pBuffer[8];                                             // 生存时间
    unsigned short checksum = ntohs(*(unsigned short *)(pBuffer + 10)); // 校验和
    unsigned int dest = ntohl(*(unsigned int *)(pBuffer + 16));                  // 目的地址

    if (version != 4) {
        // IP 版本号错
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_VERSION_ERROR);
        return 1;
    }
    else if (ttl <= 0) {
        // TTL 值非正
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_TTL_ERROR);
        return 1;
    }
    else if (head_length < 5) {
        //头部长度错
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_HEADLEN_ERROR);
        return 1;
    }
    else if (dest != getIpv4Address()) {
        //目的地址错
        ip_DiscardPkt(pBuffer, STUD_IP_TEST_DESTINATION_ERROR);
        return 1;
    }

    unsigned long sum = 0;
    for (int i = 0; i < head_length * 2; ++i) {
        sum += (unsigned char)pBuffer[i * 2] << 8;
```

```
            sum += (unsigned char)pBuffer[i * 2 + 1];
        }
        unsigned short l_word = sum & 0xffff;
        unsigned short h_word = sum >> 16;
        if (l_word + h_word != 0xffff) {
            // IP 校验和出错
            ip_DiscardPkt(pBuffer, STUD_IP_TEST_CHECKSUM_ERROR);
            return 1;
        }

        ip_SendtoUp(pBuffer, length);
        return 0;
}
```

IPV4 分组转发实验

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

struct routeTableItem {
    unsigned int destIP;    //目的 IP
    unsigned int mask;      // 掩码
    unsigned int masklen; // 掩码长度
    unsigned int nexthop; // 下一跳
};

vector<routeTableItem> m_table;
void stud_Route_Init() {
    m_table.clear();
    return;
```

```
}

void stud_route_add(stud_route_msg *proute) {
	routeTableItem newTableItem;
	newTableItem.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
	newTableItem.mask = (1 << 31) >> (ntohl(proute->masklen) - 1); //
	newTableItem.destIP = ntohl(proute->dest);
	newTableItem.nexthop = ntohl(proute->nexthop);
	m_table.push_back(newTableItem);
	return;
}

int stud_fwd_deal(char *pBuffer, int length) {
	int TTL = (int)pBuffer[8]; //存储 TTL
	int headerChecksum = ntohl(*(unsigned short *)(pBuffer + 10));
	int DestIP = ntohl(*(unsigned int *)(pBuffer + 16));
	int headsum = pBuffer[0] & 0xf;
	if (DestIP == getIpv4Address()) {   //判断分组地址与本机地址是否相同
		fwd_LocalRcv(pBuffer, length); //将 IP 分组上交本机上层协议
		return 0;
	}
	if (TTL <= 0) {   // TTL 判断 小于 0 不能转发 丢弃 IP 分组
		fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_TTLERROR); //丢弃 IP 分组
		return 1;
	}
	//设置匹配位
	bool Match = false;
	unsigned int longestMatchLen = 0;
	int bestMatch = 0;
	// 判断掩码是否匹配
	for (int i = 0; i < m_table.size(); i++) {
		if (m_table[i].masklen > longestMatchLen && m_table[i].destIP == (DestIP &
m_table[i].mask)) {
			bestMatch = i;
			Match = true;
			longestMatchLen = m_table[i].masklen;
		}
	}
	if (Match) {	//匹配成功
		char *buffer = new char[length];
		memcpy(buffer, pBuffer, length);
		buffer[8]--; // TTL - 1
		int sum = 0;
```

```
            unsigned short int localCheckSum = 0;
            for (int j = 1; j < 2 * headsum + 1; j++) {
                if (j != 6) {
                    sum = sum + (buffer[(j - 1) * 2] << 8) + (buffer[(j - 1) * 2 + 1]);
                    sum %= 65535;
                }
            }
            //重新计算校验和
            localCheckSum = htons(~(unsigned short int)sum);
            memcpy(buffer + 10, &localCheckSum, sizeof(unsigned short));
            // 发给下一层协议
            fwd_SendtoLower(buffer, length, m_table[bestMatch].nexthop);
            return 0;
        }
        else {   //匹配失败
            fwd_DiscardPkt(pBuffer, STUD_FORWARD_TEST_NOROUTE); //丢弃 IP 分组
            return 1;
        }
        return 1;
}
```