



哈尔滨工业大学
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						



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”。

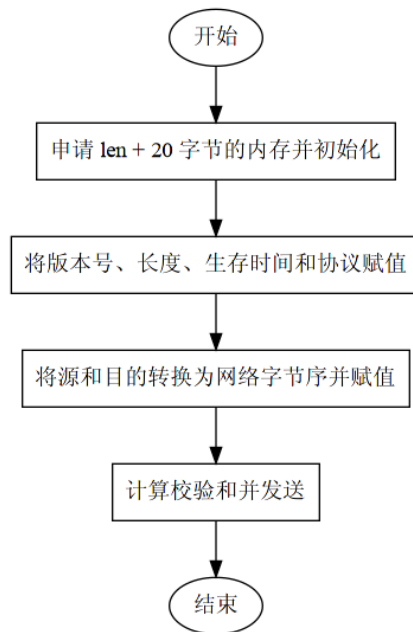
3) IPv4 分组的转发。

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。

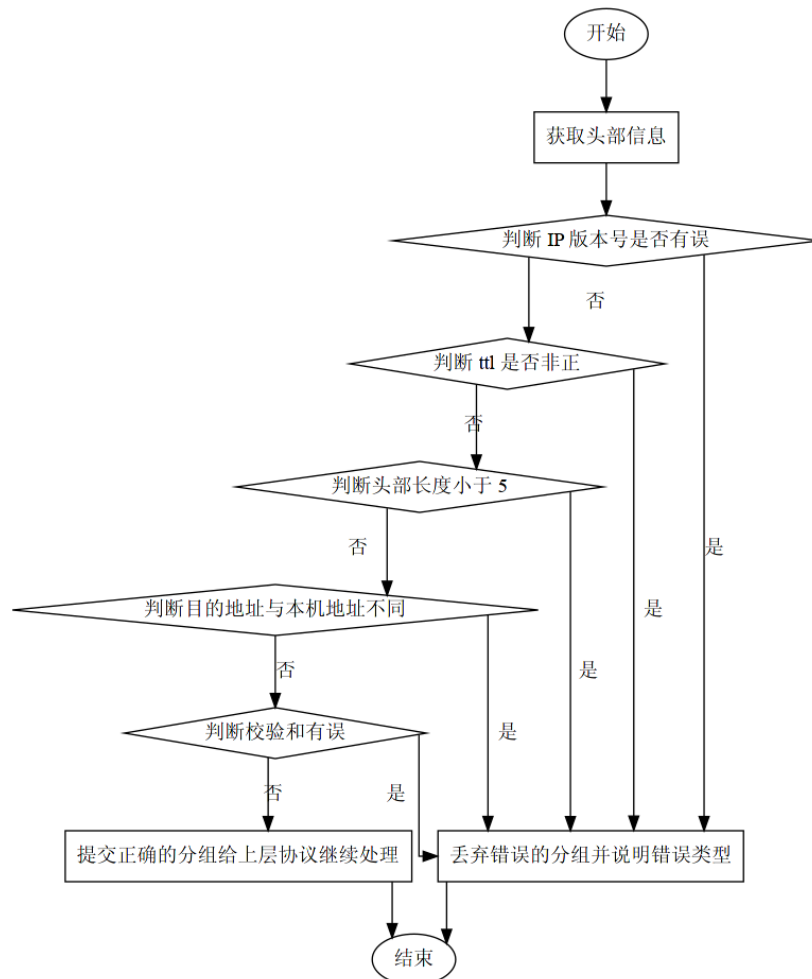
实验过程:

1. IPv4 分组收发实验

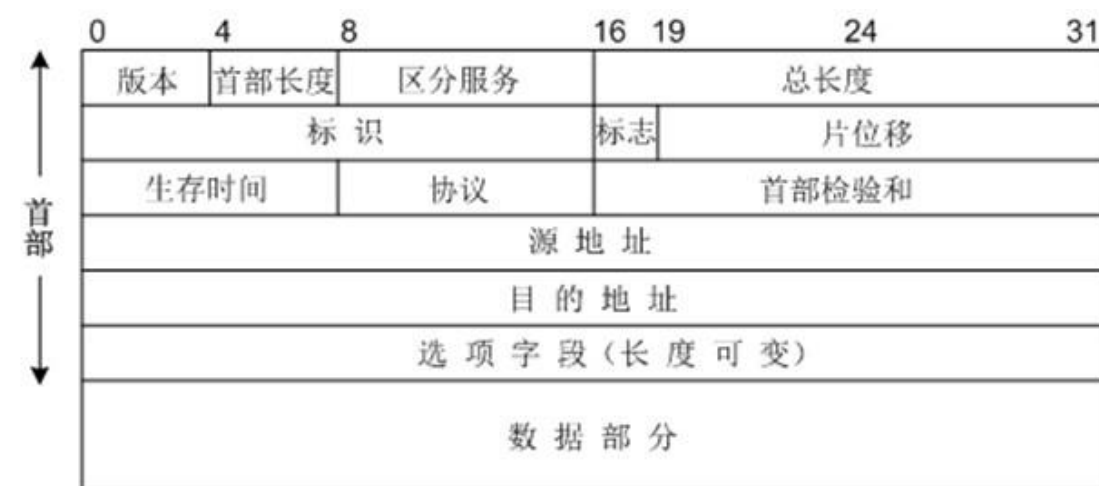
发送函数流程图如下：



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

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: 8 bytes (bogus, must be at least 20)

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: 0x69E1 [correct]

Source: 10.0.0.1

Destination : 10.0.0.3

Data(12 bytes)(invalid TCP header)

0000 00 0D 03 00 00 0A 00 0D 01 00 00 0A 08 00 42 00
0010 00 14 00 00 00 00 40 06 69 E1 0A 00 00 01 0A 00
0020 00 03

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

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: 0

Protocol: TCP (0x06)

Header checksum: 0xA6E1 [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 45 00
0010 00 14 00 00 00 00 00 06 A6 E1 0A 00 00 01 0A 00
0020 00 03

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

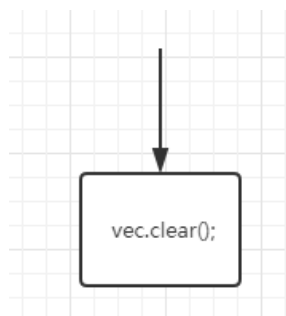
2. IPv4 分组转发实验

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;  
}
```

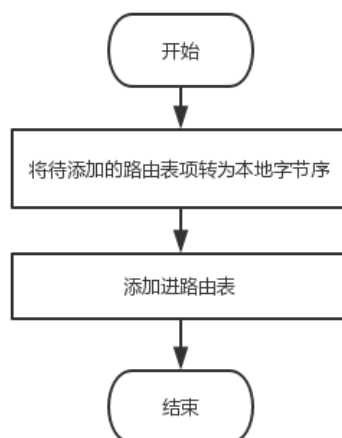


2.2 路由表增加

`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 = ntohs(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 = ntohs(proute->dest);  
    newTableItem.nextthop = ntohs(proute->nextthop);  
    m_table.push_back(newTableItem);  
    return;  
}
```



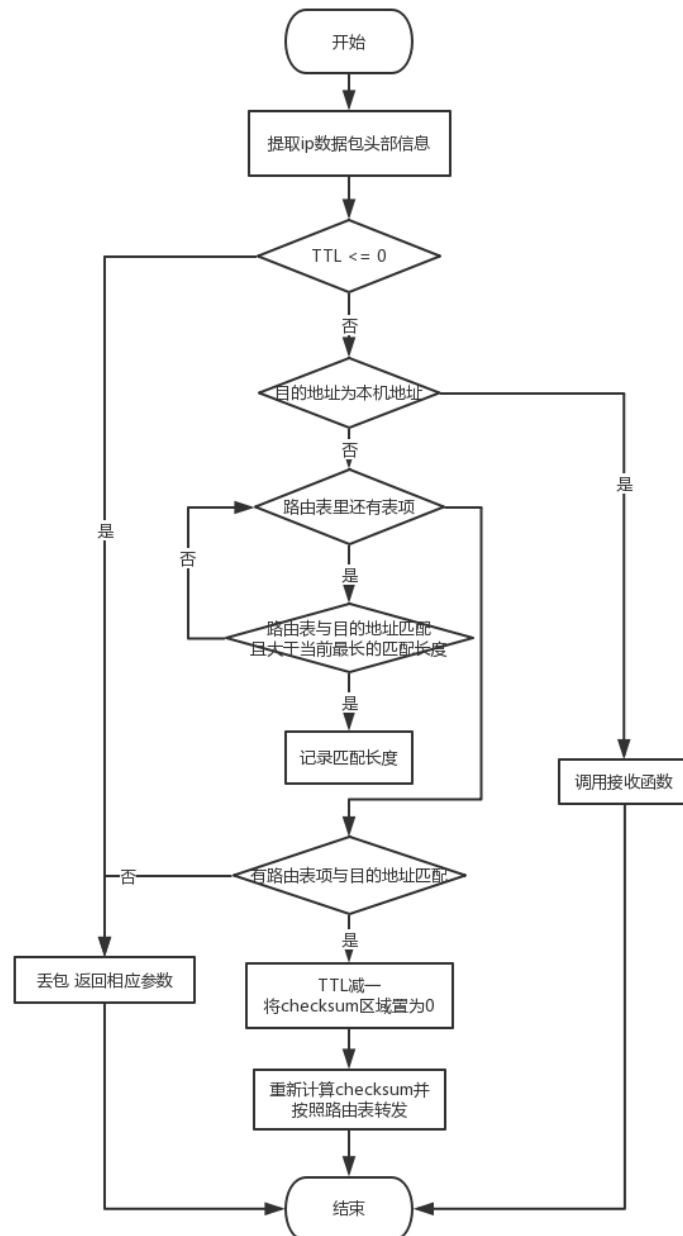
2.3 路由转发

`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

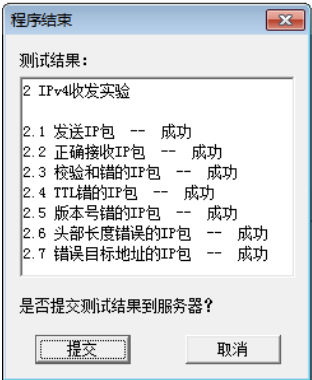
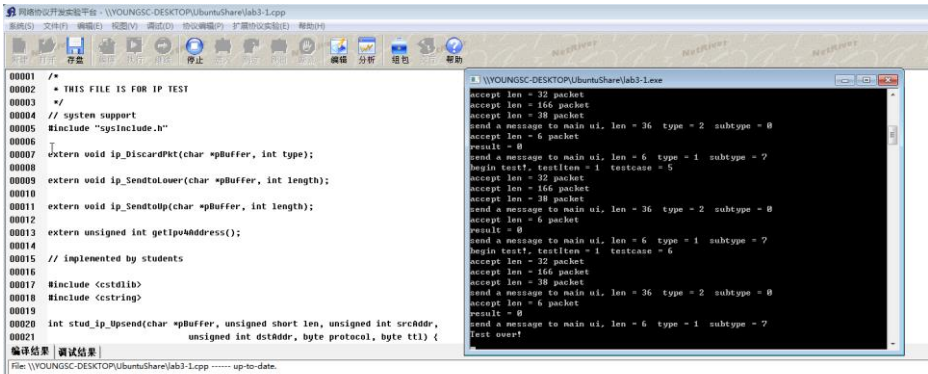
```
struct routeTableItem {
    unsigned int destIP; //目的IP
    unsigned int mask;    // 掩码
    unsigned int masklen; // 掩码长度
    unsigned int nexthop; // 下一跳
};
```

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。

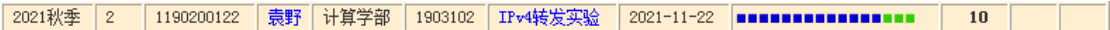
验证过程以及实验结果：

1. IPV4分组收发实验



The screenshot displays a Windows desktop environment. At the top, a taskbar contains icons for File Explorer, Visual Studio, and a terminal window. The main area is divided into two windows. The left window, titled 'lab3-2.cpp', shows C++ source code for a network packet forwarding test. The code includes headers, defines a buffer, and implements functions for sending and discarding packets. The right window, titled 'lab3-2.exe', shows the output of the program's execution, which simulates a network test with various packet sizes and types. The output shows successful transmission of packets of different lengths and types, followed by a 'test over!' message.

```
00001 /*
00002  * THIS FILE IS FOR IP FORWARD TEST
00003  */
00004 #include "sysInclude.h"
00005 #include <vector>
00006 using std::vector;
00007 #include <iostream>
00008 using std::cout;
00009 // system support
00010 extern void fwd_LocalRcv(char *pBuffer, int length);
00011
00012 extern void fwd_SendtoLower(char *pBuffer, int length, unsigned int nextHop);
00013
00014 extern void fwd_DiscardPKT(char *pBuffer, int type);
00015
00016 extern unsigned int getpwnAddress();
00017
00018 struct routeLabelItem {
00019     unsigned int destIP; //目的IP
00020     unsigned int mask; //掩码
00021     unsigned int masklen; //掩码长度
00022 };
00023
00024 #endif
00025
00026 #endif
00027
00028 #endif
00029
00030 #endif
00031
00032 #endif
00033
00034 #endif
00035
00036 #endif
00037
00038 #endif
00039
00040 #endif
00041
00042 #endif
00043
00044 #endif
00045
00046 #endif
00047
00048 #endif
00049
00050 #endif
00051
00052 #endif
00053
00054 #endif
00055
00056 #endif
00057
00058 #endif
00059
00060 #endif
00061
00062 #endif
00063
00064 #endif
00065
00066 #endif
00067
00068 #endif
00069
00070 #endif
00071
00072 #endif
00073
00074 #endif
00075
00076 #endif
00077
00078 #endif
00079
00080 #endif
00081
00082 #endif
00083
00084 #endif
00085
00086 #endif
00087
00088 #endif
00089
00090 #endif
00091
00092 #endif
00093
00094 #endif
00095
00096 #endif
00097
00098 #endif
00099
00100 #endif
00101
00102 #endif
00103
00104 #endif
00105
00106 #endif
00107
00108 #endif
00109
00110 #endif
00111
00112 #endif
00113
00114 #endif
00115
00116 #endif
00117
00118 #endif
00119
00120 #endif
00121
00122 #endif
00123
00124 #endif
00125
00126 #endif
00127
00128 #endif
00129
00130 #endif
00131
00132 #endif
00133
00134 #endif
00135
00136 #endif
00137
00138 #endif
00139
00140 #endif
00141
00142 #endif
00143
00144 #endif
00145
00146 #endif
00147
00148 #endif
00149
00150 #endif
00151
00152 #endif
00153
00154 #endif
00155
00156 #endif
00157
00158 #endif
00159
00160 #endif
00161
00162 #endif
00163
00164 #endif
00165
00166 #endif
00167
00168 #endif
00169
00170 #endif
00171
00172 #endif
00173
00174 #endif
00175
00176 #endif
00177
00178 #endif
00179
00180 #endif
00181
00182 #endif
00183
00184 #endif
00185
00186 #endif
00187
00188 #endif
00189
00190 #endif
00191
00192 #endif
00193
00194 #endif
00195
00196 #endif
00197
00198 #endif
00199
00200 #endif
00201
00202 #endif
00203
00204 #endif
00205
00206 #endif
00207
00208 #endif
00209
00210 #endif
00211
00212 #endif
00213
00214 #endif
00215
00216 #endif
00217
00218 #endif
00219
00220 #endif
00221
00222 #endif
00223
00224 #endif
00225
00226 #endif
00227
00228 #endif
00229
00230 #endif
00231
00232 #endif
00233
00234 #endif
00235
00236 #endif
00237
00238 #endif
00239
00240 #endif
00241
00242 #endif
00243
00244 #endif
00245
00246 #endif
00247
00248 #endif
00249
00250 #endif
00251
00252 #endif
00253
00254 #endif
00255
00256 #endif
00257
00258 #endif
00259
00260 #endif
00261
00262 #endif
00263
00264 #endif
00265
00266 #endif
00267
00268 #endif
00269
00270 #endif
00271
00272 #endif
00273
00274 #endif
00275
00276 #endif
00277
00278 #endif
00279
00280 #endif
00281
00282 #endif
00283
00284 #endif
00285
00286 #endif
00287
00288 #endif
00289
00290 #endif
00291
00292 #endif
00293
00294 #endif
00295
00296 #endif
00297
00298 #endif
00299
00300 #endif
00301
00302 #endif
00303
00304 #endif
00305
00306 #endif
00307
00308 #endif
00309
00310 #endif
00311
00312 #endif
00313
00314 #endif
00315
00316 #endif
00317
00318 #endif
00319
00320 #endif
00321
00322 #endif
00323
00324 #endif
00325
00326 #endif
00327
00328 #endif
00329
00330 #endif
00331
00332 #endif
00333
00334 #endif
00335
00336 #endif
00337
00338 #endif
00339
00340 #endif
00341
00342 #endif
00343
00344 #endif
00345
00346 #endif
00347
00348 #endif
00349
00350 #endif
00351
00352 #endif
00353
00354 #endif
00355
00356 #endif
00357
00358 #endif
00359
00360 #endif
00361
00362 #endif
00363
00364 #endif
00365
00366 #endif
00367
00368 #endif
00369
00370 #endif
00371
00372 #endif
00373
00374 #endif
00375
00376 #endif
00377
00378 #endif
00379
00380 #endif
00381
00382 #endif
00383
00384 #endif
00385
00386 #endif
00387
00388 #endif
00389
00390 #endif
00391
00392 #endif
00393
00394 #endif
00395
00396 #endif
00397
00398 #endif
00399
00400 #endif
00401
00402 #endif
00403
00404 #endif
00405
00406 #endif
00407
00408 #endif
00409
00410 #endif
00411
00412 #endif
00413
00414 #endif
00415
00416 #endif
00417
00418 #endif
00419
00420 #endif
00421
00422 #endif
00423
00424 #endif
00425
00426 #endif
00427
00428 #endif
00429
00430 #endif
00431
00432 #endif
00433
00434 #endif
00435
00436 #endif
00437
00438 #endif
00439
00440 #endif
00441
00442 #endif
00443
00444 #endif
00445
00446 #endif
00447
00448 #endif
00449
00450 #endif
00451
00452 #endif
00453
00454 #endif
00455
00456 #endif
00457
00458 #endif
00459
00460 #endif
00461
00462 #endif
00463
00464 #endif
00465
00466 #endif
00467
00468 #endif
00469
00470 #endif
00471
00472 #endif
00473
00474 #endif
00475
00476 #endif
00477
00478 #endif
00479
00480 #endif
00481
00482 #endif
00483
00484 #endif
00485
00486 #endif
00487
00488 #endif
00489
00490 #endif
00491
00492 #endif
00493
00494 #endif
00495
00496 #endif
00497
00498 #endif
00499
00500 #endif
00501
00502 #endif
00503
00504 #endif
00505
00506 #endif
00507
00508 #endif
00509
00510 #endif
00511
00512 #endif
00513
00514 #endif
00515
00516 #endif
00517
00518 #endif
00519
00520 #endif
00521
00522 #endif
00523
00524 #endif
00525
00526 #endif
00527
00528 #endif
00529
00530 #endif
00531
00532 #endif
00533
00534 #endif
00535
00536 #endif
00537
00538 #endif
00539
00540 #endif
00541
00542 #endif
```



如何提高路由表的查找效率？

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 <stdlib.h>
#include <string.h>

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_rcv(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;
}
```