**IPV4协议转发实验上机报告**

1. **实验目的**

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。了解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。

1. **实验要求**

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

1. **实验内容**
2. 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。

1. IPv4 分组的接收和发送。

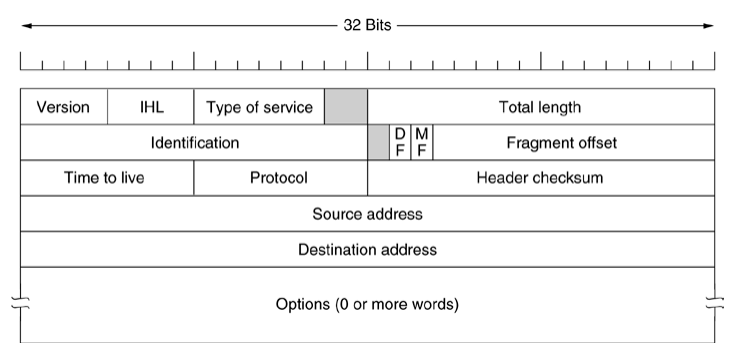
对前面实验（IP 实验） 中所完成的代码进行修改，在路由器协议栈的 IPv4 模块中能够正确完成分组的接收和发送处理。

1. 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

1. **实验原理**

IPv4分组头部格式如图，所以只需从头部各字节提取相应信息，对报文做出检验。根据目的地址决定是上传还是查找路由表转发或丢弃。



本实验中，采用线性表的数据结构存储路由表。

struct routeTableItem

{

unsigned int destIP;

unsigned int mask;

unsigned int masklen;

unsigned int nexthop;

};

//线性路由表

vector<routeTableItem> routeTable;

1. **实验代码**

#include "sysInclude.h"

#include <vector>

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

typedef struct stud\_route\_msg

{

unsigned int dest;

unsigned int masklen;

unsigned int nexthop;

};

\*/

struct routeTableItem

{

unsigned int destIP;

unsigned int mask;

unsigned int masklen;

unsigned int nexthop;

};

//线性路由表

vector<routeTableItem> routeTable;

void stud\_Route\_Init()

{

routeTable.clear();

}

void stud\_route\_add(stud\_route\_msg \*proute)

{

routeTableItem item;

item.masklen = ntohl(proute->masklen);

item.mask = (1<<31)>>(ntohl(proute->masklen)-1);

item.destIP = ntohl(proute->dest)&item.mask;

item.nexthop = ntohl(proute->nexthop);

routeTable.push\_back(item);

}

//计算校验和

unsigned short \_checksum(char \*pBuffer)

{

int sum = 0;

for(int i = 0; i < 10; ++i)

{

if(i != 5)

{

sum += ((unsigned short\*)pBuffer)[i];

}

}

while(sum > 0xffff)

{

sum = (sum & 0xffff) + (sum >> 16);

}

return (unsigned short)(0xffff - sum);

}

int stud\_fwd\_deal(char \*pBuffer, int length)

{

//提取头部信息

int IHL = pBuffer[0] & 0xf;

int TTL = (int)pBuffer[8];

int headerChecksum = ntohl(\*(unsigned short\*)(pBuffer+10));

int destIP = ntohl(\*(unsigned int\*)(pBuffer+16));

//是否过期

if(TTL <= 0)

{

fwd\_DiscardPkt(pBuffer, STUD\_FORWARD\_TEST\_TTLERROR);

return 1;

}

//如果目标地址为本机

if(destIP == getIpv4Address())

{

fwd\_LocalRcv(pBuffer, length);

return 0;

}

bool isMatch = false;

unsigned int longestMatchLen = 0;

int bestMatch = 0;

//匹配最佳的地址

for(int i = 0; i < routeTable.size(); ++i)

{

if(routeTable[i].masklen > longestMatchLen && routeTable[i].destIP == (destIP & routeTable[i].mask))

{

bestMatch = i;

isMatch = true;

longestMatchLen = routeTable[i].masklen;

}

}

//匹配成功，更新TTL和校验和并转发

if(isMatch)

{

char \*buffer = new char[length];

memcpy(buffer,pBuffer,length);

--buffer[8];//TTL

unsigned short localCheckSum = \_checksum(buffer);

memcpy(buffer+10, &localCheckSum, sizeof(unsigned short));

fwd\_SendtoLower(buffer, length, routeTable[bestMatch].nexthop);

return 0;

}

//没有找到目标地址，直接丢弃

else

{

fwd\_DiscardPkt(pBuffer, STUD\_FORWARD\_TEST\_NOROUTE);

return 1;

}

}