**项目一最小网元设计总报告**

**第6组：蓝苛晋 张洋 邹坤**

**2021年6月**

1. **阶段一**

**1.目标：**完成网元模型分层划分，即需要哪些层级，并介绍每一层级的功能，为之后的工作做准备。

**2.层次划分：**我们小组给网元分为物理层，数据链路层，网络层和应用层。

**3.每层功能：**

1. 物理层：主要充当一个传输01比特流的信道，接收来自上层的信息然后传给对等层，或者接收来自对等层的信息，传输给上层。
2. 数据链路层：完成各种控制功能，包括（1）帧同步功能：在01比特流中判断有效数据的起始位置和终止位置，从而将有效数据提取出来；（2）差错检测和控制功能：检测提取出来的有效数据是否正确，若不正确，则让发送方重新传错误的帧。（3）流量控制 （4）交换机基本功能
3. 网络层：（1）交换和路由功能：为网络中的每一节点编址，在节点启动后，自动建立路由表，完成数据路由和转发；（2）端到端的寻址、差错和恢复功能：根据路由表来为数据包选择路径，实现端到端的数据，文件可靠传输。
4. 应用层：这里的应用层指的是表示层、会话层与应用层。功能包括：信息编码和表示功能：即将信息编制成（二进制）比特流发送给对方，对方能够正确解读。信息编码如传输图片，同时小组为应用层添加了一个友好的用户界面。

**5.通信流程：**

LNK层帧结构：



NET层帧结构：



路由器信息有效判断位为1时表示是该数据包为APP层要求发送的，否则是动态路由算法中路由器交换路由信息的数据包。

**（1）在所有通信前进行ip地址的初始化：**

流程说明：1.路由器首先调用ip\_init将&+子网序号（SN=路由器设备号\*10+端口号）下发给链路层。

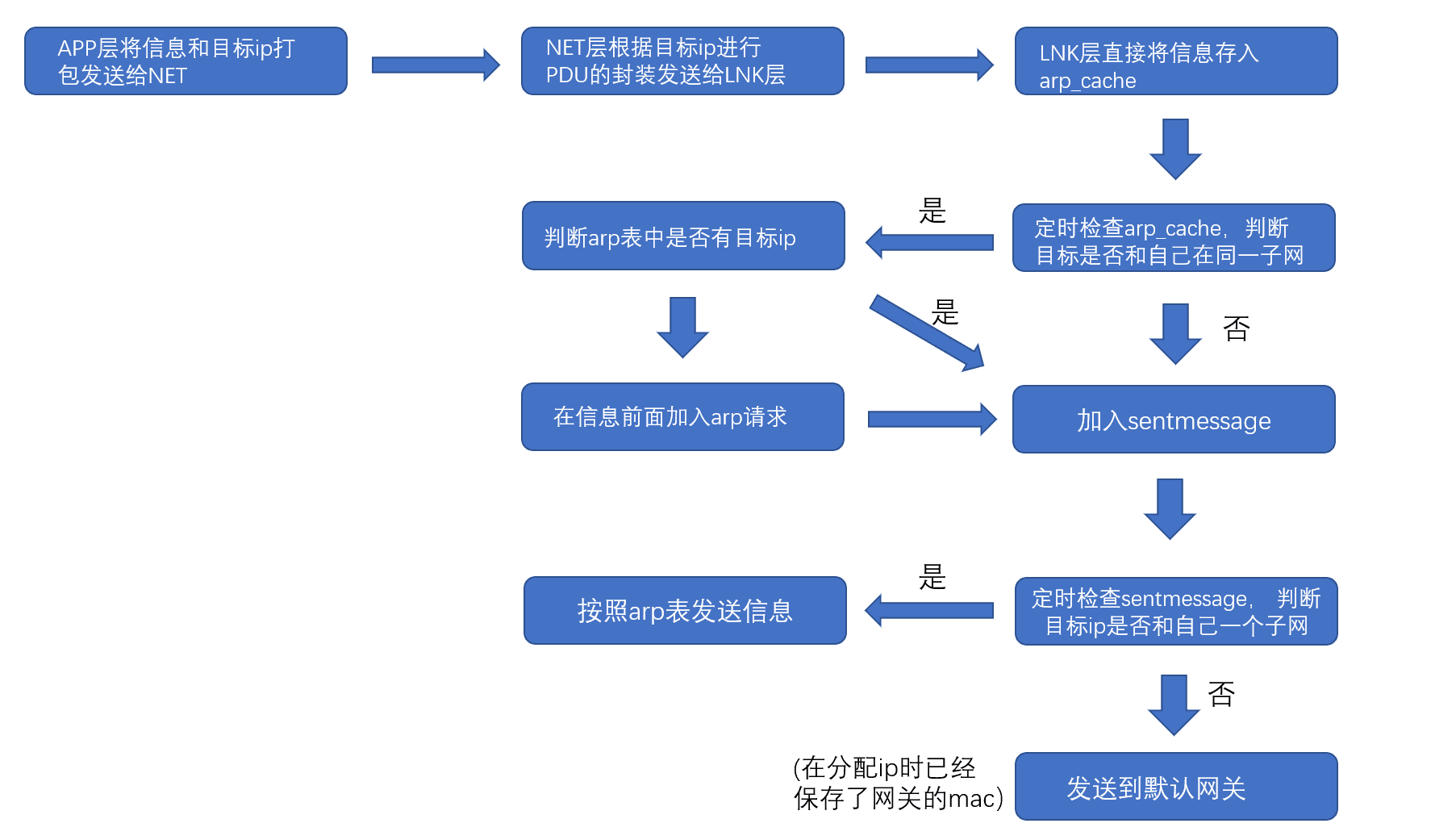
2.接到上层数据后如果端口连的是内网，路由器的LNK层将广播它的mac地址。

3.同一子网的主机收到后从LNK层返回一个ACK，数据内容为“&”表示路由器的LNK层可以分配IP地址

4.路由器LNK层收到ACK后调用lnk\_assign\_ip为每个主机分配ip地址，数据内容为$+SN+cnt（主机号），并将数据传给主机的LNK层。

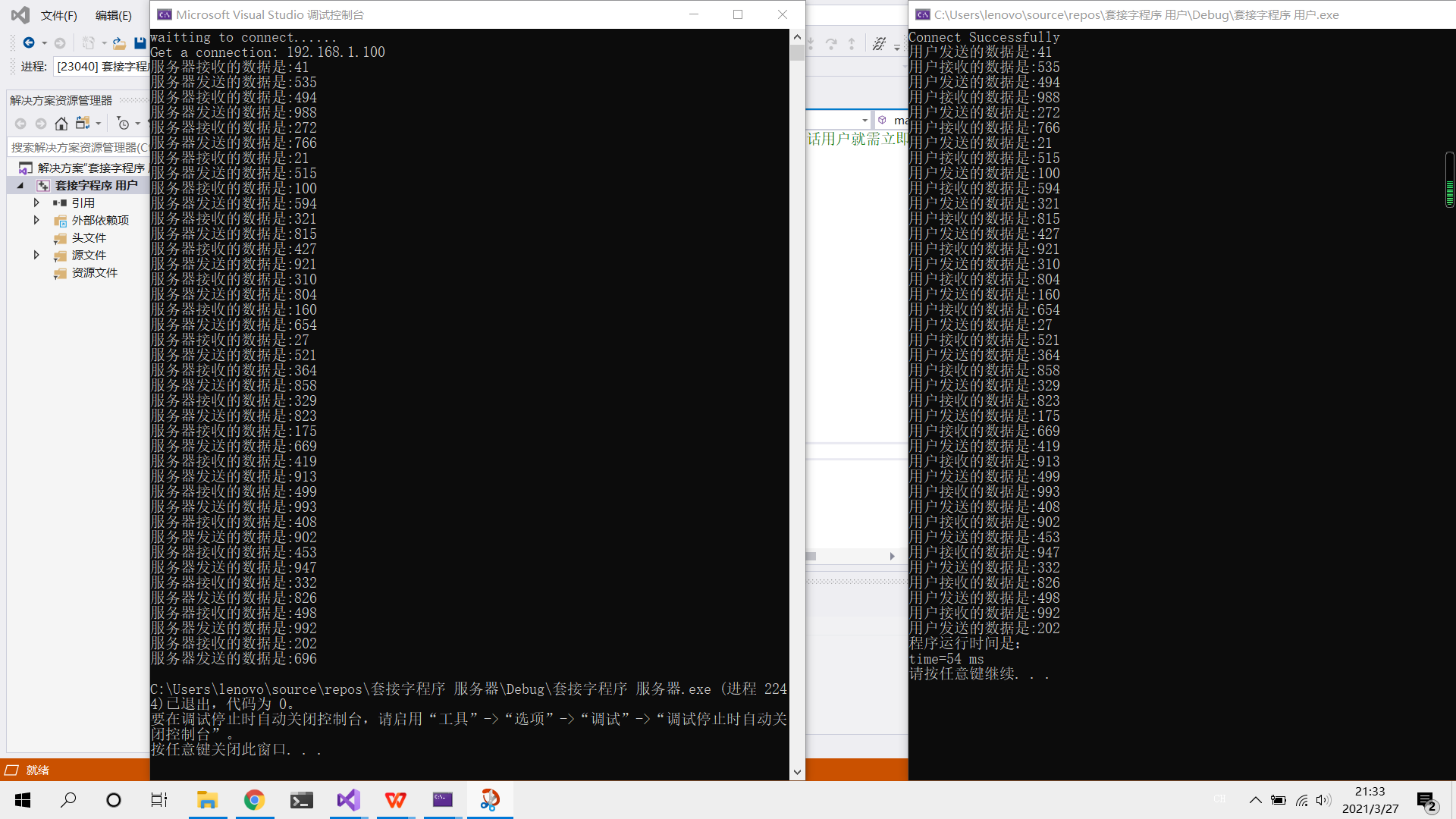
5.主机LNK层收到分配ip的消息（$为标志）后会计算出loc\_ip和默认网关，并在ARP表中存放默认网关与对应的路由器端口mac地址。之后主机LNK层会发送ACK给路由器的LNK层

**（2）通信流程图：**



**6.基于套接字的简单通信程序：**

测试结果：



1. **阶段二**

##### 实验目的

基本要求：

1）两点之间帧同步（定位）

2）差错检测

3）至少实现一种差错控制协议

##### 实验内容

##### （1）成帧

我们在LNK层里的成帧选用的是面向位的首尾定界符法

**发送方**

首先我们把高层传来的字节数组转换成bit数组并将其放到bufSend里，然后在bufSend中进行变换，每出现5个‘1’就插入1个‘0’，然后将变换后的帧放到bufSend2里面，len2就是变换后的帧的长度

然后是在bufSend2的首位添加定界符01111110，这时成帧后的长度就是len2+16了，然后函数就返回成帧后的bit数组bufSend2和长度len2+16

**函数：pair<U8\*, int> framing1(U8\* buf, int len)**

**int judge1(U8\* bufSend, int len)**

**接收方**

首先编写一个函数judge2来判断传来的帧是否出现了定界符‘01111110’，出现的话就返回0，没有出现就返回-1

然后就是将底层传来的数据中的有效帧提取出来。先是用judge2函数判断有效帧的起始位置start和中止位置end，注意这时的有效帧包含成帧变换时插入的‘0’，我们下一步会将其删除；然后用bufSend来存放提取出来的有效帧。最后一句代码是防止特殊情况，即没有检测到定界符，这时就会返回NULL和-1，然后在Receivfromlower函数中如果判断出返回帧为NULL，这时就会报错.

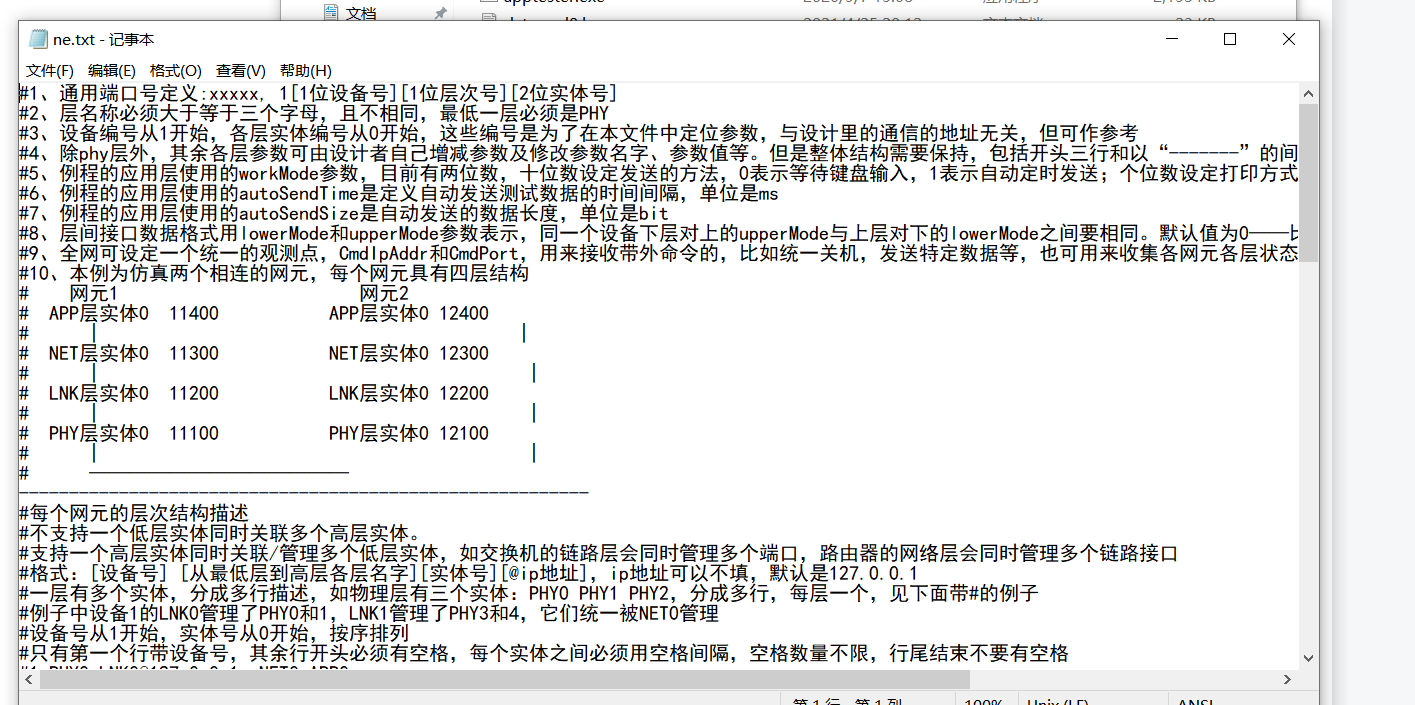
然后是将有效帧中的插入的‘0’删掉，然后返回bufSend和长度len2

然后是另外一种情况，如果底层是字节接口，那么就先要把字节转换成bit，然后再做变换。最后再返回bit数组，再Receivfromlower函数中会将bit转换为字节。因为大致操作和上面类似，就不再赘述

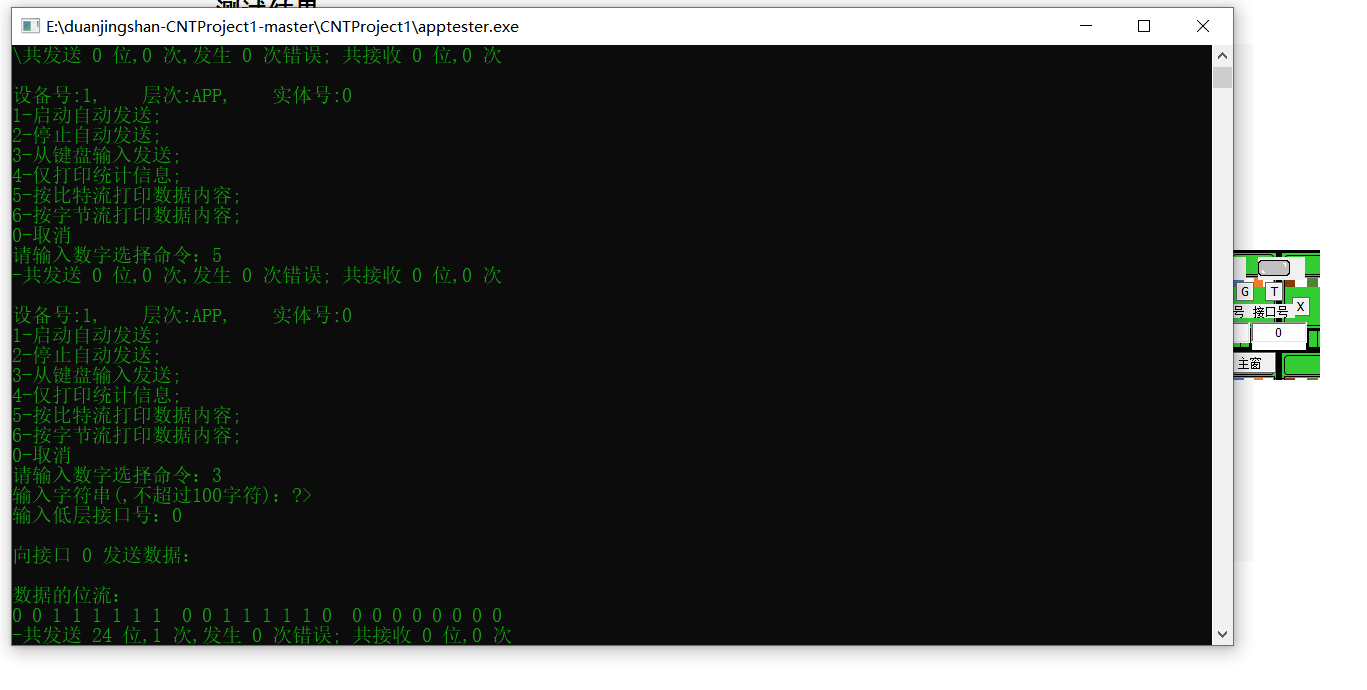
**函数：pair<U8\*, int> framing2(U8\* buf, int len, int ifNo)**

**int judge2(U8\* buf, int start, int end)**

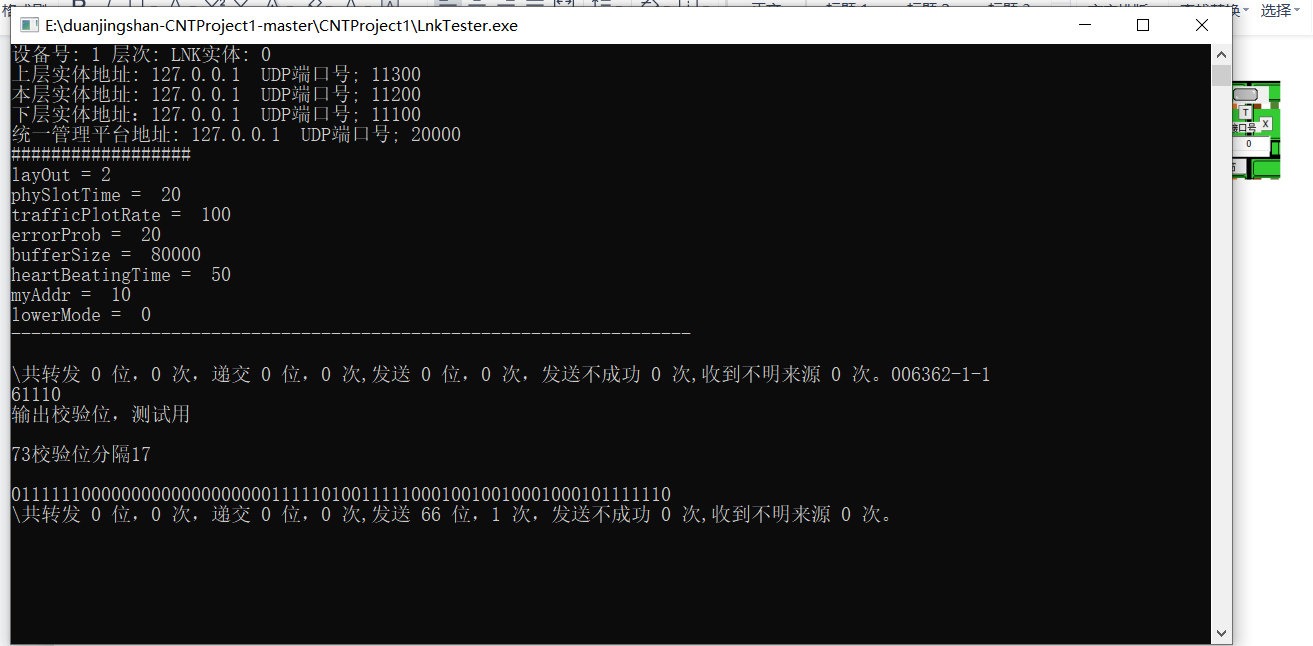
**测试结果**



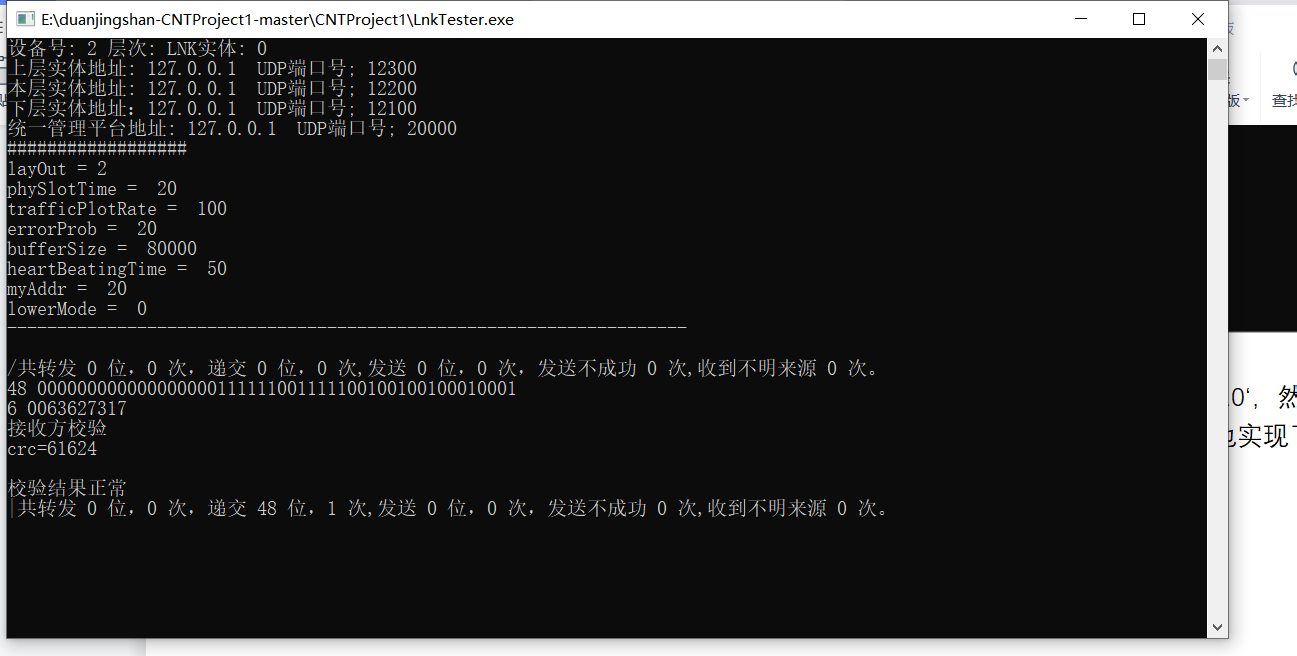
这是配置文件



我首先在设备1的APP层发送了字符‘?>’，其bit流是001111110011111000000000



然后我们可以在设备1的LNK层看到bit流的首尾都加上了定界符’01111110‘，然后我们还可以在定界符包围的有效帧中看到没有连续6个’1‘的存在，说明变换功能也实现了



然后我们可以在设备2的LNK层看到bit流已经是把定界符去掉后，把有效帧提取出来的样子了，说明反变换也已经实现了

#### （2）差错检测

差错检测和控制功能

1、差错校验技术

（1）原理与代码

对于错误率较低的信道，采用差错检测和重传的处理方式更加有效。常见检错码包括奇偶校验、校验和、循环冗余校验(CRC)等。小组决定选择CRC的方式，因为它的检错能力相对较强，在工程中被广泛运用。

相比于CRC的一般代数原理，CRC在工程上的实现有所变化。小组采用CRC16-CCITT标准进行代码编写。CRC16-CCITT的生成多项式为0x1021（生成多项式首位均为1，故略去）。在实现时，为提高效率常采用查表法。

CRC程序实现基本原理：设原始数据为M，校验码有r位，将Mx^r的前r位放入长度位r的寄存器；若寄存器首位为1，将寄存器左移一位再与生成多项式后r位异或，否则仅将寄存器左移1位。不断重复直至将全部Mx^r移入寄存器，寄存器中即为校验码。

基于异或运算的交换律和结合率性质以及与0异或值不改变的性质，我们可以让消息不移入寄存器，而是双重循环内循环时寄存器首位与对应的消息位异或。同时，CRC16-CCITT标准规定计算方式为传输LSB，消息右移进寄存器计算，故改变代码判断寄存器LSB，将0x1021颠倒后的值0x8408与寄存器异或，并通过查表法替代内循环。

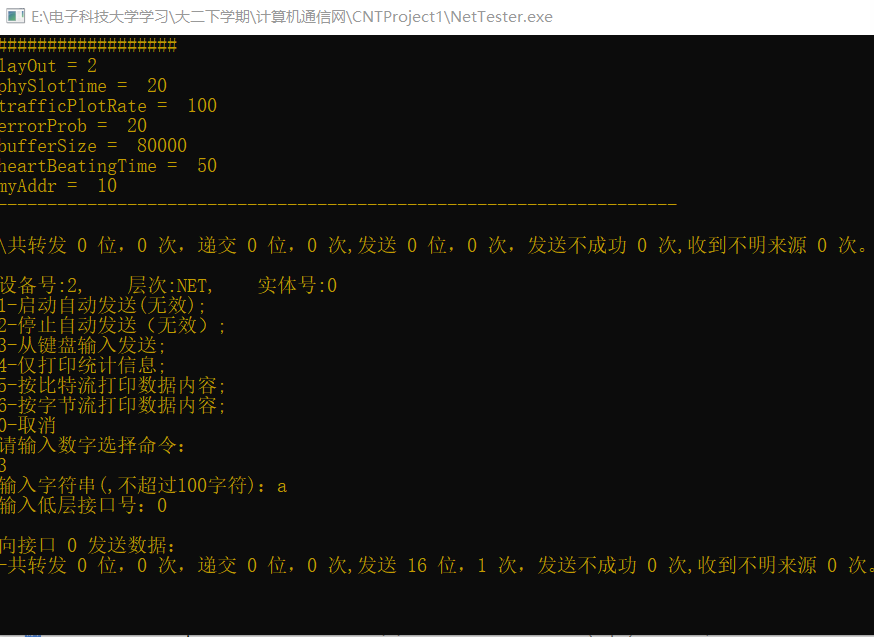
同时，还有几个值得考虑的问题：在数据前错误的增加了1个或多个0字节或者丢失了开始的1个或多个0.因此我们需要将寄存器初始值设为reg\_init（0xffff）而不是0，因为用0处理全是0的位值是不会变的。而为了处理校验码后多增加0或者包括校验码的数据丢失末尾0的问题，CCITT标准要求校验值与0Xfffff异或。因

根据CCITT标准，接收方对所有字节计算校验位，如果没有差错则结果应该为对初始寄存器的值计算校验位的结果（0xf0b8）。因此在接收方获取有效帧后执行差错校验代码，如果有错立即停止接收。

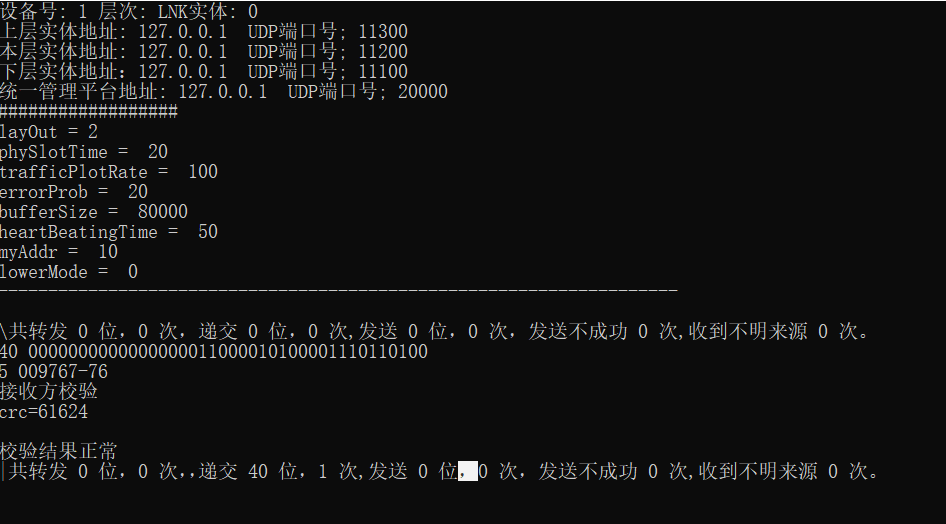
**函数：unsigned short do\_crc(unsigned short reg\_init, char\* message, unsigned int len)**

（2）测试结果展示

在网络层下发字符串a



在接收方输出如下。收到的数据有5个字节，前两个为0，中间为a的ASCII值，后面16位为校验值。接收方计算结果为61624（0xf0b8的十进制数），校验结果正确。



#### （3）差控协议

在LNK层程序我采用了停等协议，首先我在原始数据前面加了两个字节用来判断该帧的属性。

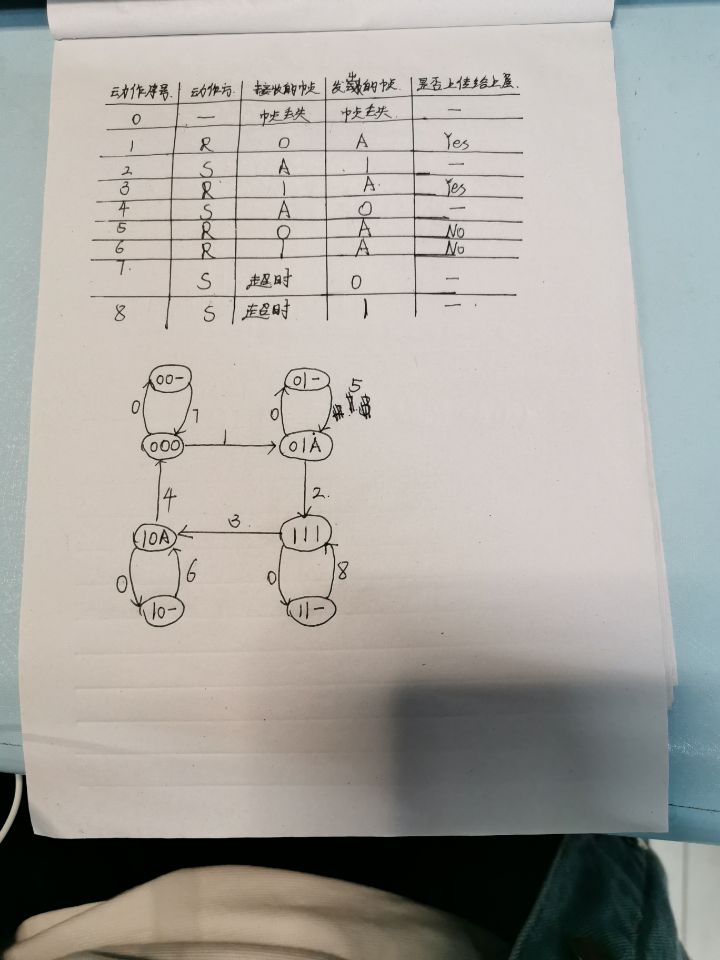
然后通过成帧的函数将buf1封装成帧进行发送，发送后要记录下发送的时间并且将发送信息存在缓存中，

在发送后将ack标志设置成为0，代表现在还没有收到回复，在没有收到回复的时候，如果上层来了信息不能进行转发

在接收方进行对信息的判断，如果该信息是MESSAGE，那么返回一个ACK信息，如果该信息是ACK信息，那么将ack标志设置成为1，代表已经接收到回复，可以进行下一个信息的发送

如果一直没有接收到ACK，那么发送方在一定时间后会进行重新发送

停等协议的有限状态机：



**函数：void TimeOut()（LNKTESTER）**

**(4)流量控制与缓冲区**

建立一个队列进行缓冲：queue<string> sentmessage;在timeout中定时执行相关代码，发送队列顶部的string，在收到ack后将队列顶部的消息弹出（详见阶段四ARP部分）。通过这样的方式实现了一定的流量控制。

**函数：void TimeOut()（LNKTESTER）**

**阶段三**

原函数的封装：

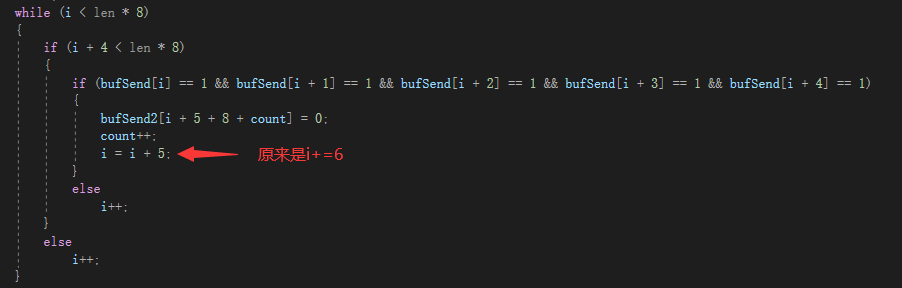
在消息上加一个PDU，

格式如下

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 源地址 | 目的地址 | ACK | 帧序号 | 信息 | 校验位 | 校验位 |

成帧的修改：

原来成帧函数在遇到16个连续的1时会出现提前终止的情况，在修改后可以把每五个1断开



对信息的回复：**void reply(int des\_mac, int frame\_num, int a)**

在receivefromlower中添加了

反向学习功能，将来源和接口进行匹配

判断自己的mac和消息目的mac是否一致，如果不一致，那么交换机进行转发，其他主机将消息丢弃。交换机查表看自己是否保存过对应mac地址，如果没保存那么就进行广播，保存了就进行了定点转发。

广播函数：**void radio(U8\* buf, int len)**

**阶段四**

在这个阶段中我们要实现网络层中的路由功能。小组经讨论决定使用动态路由算法，由于网元结构规模较小，相对简单，所以采取距离矢量路由算法比较合适。

**1、实体编址**

在阶段4我们需要解决各层实体的标定区分和数据投递，经过小组同学商量，具体步骤如下：第一步路由器的NET层给LNK层发送一个分配IP的指令，指令的内容是一个&+一个数字，把子网的序号设置为那个数字。此时路由器的LNK层广播自己的MAC地址，所有主机的LNK层收到后返回一个ACK，ACK中间也加一个&符号，表示路由器的LNK层可以分配IP。路由器用一个计数器为主机分配IP。

在LNK层，使用变量a判断主机是否是发送自己的mac地址，a=1是，否则不是。当LNK层收到物理层传递的数据包时会判断是否返回自己的mac地址。

接下来LNK层会判断自己是不是交换机，如果是交换机将执行交换机功能，不是的时候需要对收到的数据包进一步判断。如果接收的不是要求分配IP的ACK，那么判断是否在指定位置有指示下一位为IP地址的‘$’符号，如果有就将下一位储存下来作为自己的IP地址。

如果loc\_ip>=0,说明自己已经有ip地址了，直接返回。

如果接受到的是主机表示可以分配IP的ACK，将调用lnk\_assign\_ip函数为主机分配ip

**函数：void net\_assign\_ip(int SN, unsigned short port)**

**void ip\_init()**

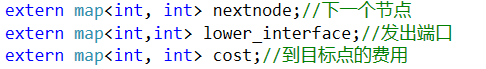
**U8\* addhead(U8\* buf, int len, int desip)**

**int lnk\_assign\_ip(U8\* buf, int len, int port)**

**void save\_ip(U8\* buf)**

**int radio\_mac(U8\* buf, int r, int des\_mac, int src\_mac)**

**2、路由表设计**



张洋同学提出使用C++的STL中的map作为路由表的数据结构，因为路由算法中会需要查询路由表，而在map中查询的时间复杂度为log（n），具有一定的优势。

**3、动态路由实现**

（1）交换路由信息函数

交换时某一路由器的链路信息存储在tem中，tem从第4位开始是路由信息，将cost的键与值都放入tem中，并通过在阶段3中完成的广播函数发送给相邻的路由器。最后更新上一次发送的时间。

（2）TimeOut函数

网元开始运行时lastsend被设为无穷大，因此第一次需要手动触发广播路由表函数，触发后将控制在50个单位实行一次，为避免不停广播将最大广播次数设置为5.

（3）距离矢量算法实现

NET层接收到底层发来的数据包后将进行路由表的更新。直接相连的链路传递所需时延可视为相同，即设为1，到src\_ip的下一站即是scr\_ip自己。如果发现收到的其它路由表的路由信息有自己不知道的节点的cost或者通过scr\_ip到该节点的cost小于自己目前存储的cost，那么更新对应的cost和nextnode.

**函数：void router\_init()**

**void announce()**

**4．ARP实现**

为了模仿真实计算机网络系统，将ip地址与mac地址区分开来（虽然在该项目中不一定非要这么做）而又保持着联系，小组同学完成了类似于ARP协议的功能。

在timeout函数中定期检测LNK层的缓冲，获取每一个报文的目标ip并在ARP表中检查有无表项，如果有直接放入sentmessage队列中。如果没有就调用addarp函数添加一个ARP请求。（可以结合通信流程图）

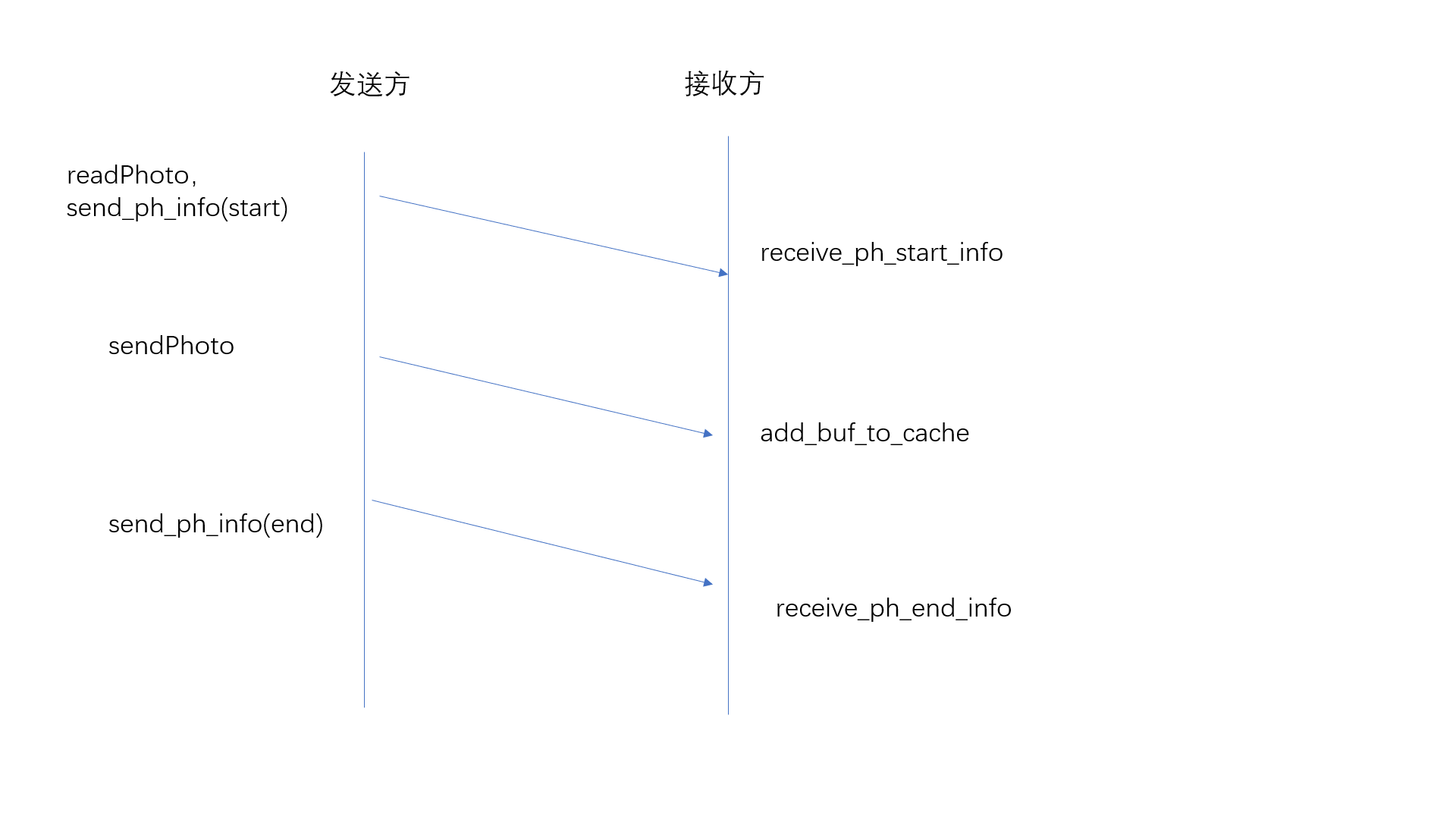
**函数：void savearp(int desmac)**

**void TimeOut()**

**void addarp(int desip)**

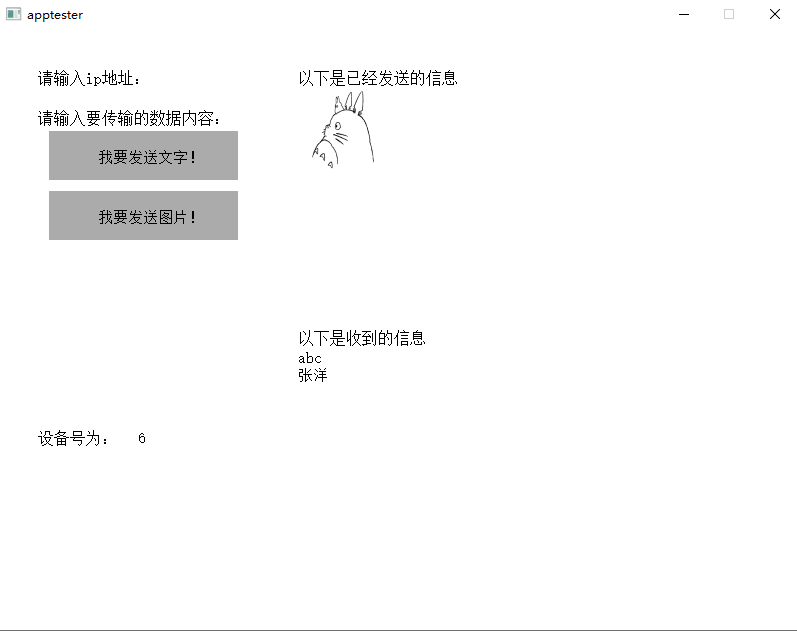
**5．新增功能：传输文件（图片）**

流程图如下：

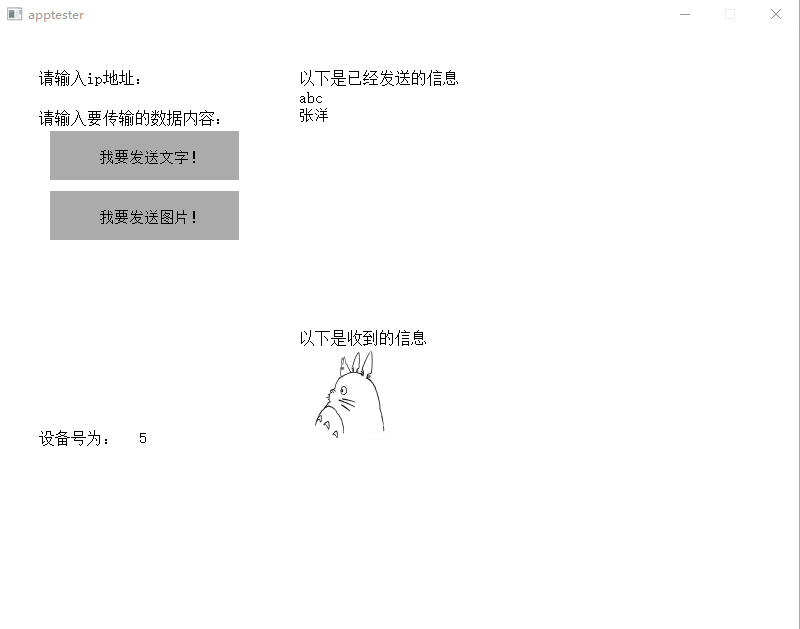


发送方要传输图片，首先将图片读取到buffer中，接下来通知接收方自己要开始传图片了（send\_ph\_info（start）），将图片存储需要的缓存区cache大小告诉接收方。接收方收到开始传图片的通知后即为cache分配相应大小的空间。发送方在send\_ph\_info一段时间后将进行sendPhoto,将buffer采用分段的形式传输，分段大小为100.接收方将按序将收到的数据放入cache中。当发送方发送完成后将send\_ph\_info(end)，告诉接收方自己传完了。接收方收到后将cache数组转换为图片并重置cache。本小组在LNK层有暂存高层数据的功能，因此若需要重传不会打乱分段传输的正确顺序。

**测试结果：**



发送方应用层界面



接收方应用层界面

对比发送和接收的两张图片可以看出传输图片功能成功实现。

**函数：void menu()**

**void RecvfromLower(U8\* buf, int len, int ifNo)**

**pair<U8\*, int> readPhoto()**

**void sendPhoto(U8\* buf, int len,int ip)**

**void send\_ph\_info(int len, int s\_or\_e,int ip)**

**void receive\_ph\_start\_info(U8\* buf, int len)**

**void receive\_ph\_end\_info(U8\* buf, int len)**

**void add\_buf\_to\_cache(U8\* buf, int len)**

总结

1、反思与不足

（1）、主机路由器和交换机混杂，代码阅读困难  
（2）、路由器之间没有实现arp模式，采用一对一链接  
（3）、网络层和链路层有一些服务重叠，ip泄露在LNK层  
（4）、帧结构鲁棒性不够，传输过程会发生意外终止，只能通过特判解决

（5）代码最后还是有不少地方可以进一步封装，最明显的例子就是LNK层无处不在的加两位CRC校验码的代码。

（6）尽管小组同学在编写时注意到变量应该赋予一个有意义的名字的重要性，但有时仍然没有遵循，比如无处不在的buf+一个数字，造成了代码可读性的下降

2、迭代过程  
LNK：  
v1.0:采用一个线性缓冲，只能保存一条信息，在没有接受到ack时，无法接受上层的消息。  
v2.0：采用一个sentmessage二维消息队列，可以将上层消息保存在队列中同时维持消息的有序性  
v3.0：添加了第二重缓冲arp\_cache,保证arp请求和消息的必然顺序，即在发送消息时arp表一定会存在目标ip  
NET：  
v1.0：采用设备号作为ip地址，但是无法隔离广播域。  
v2.0：采用分配模式，将路由器设备号+路由器LNK实体号+主机号用一位字节进行保存，但是在设备数量过多时会出现溢出情况。  
v3.0：用两个字节来保存ip，第一个字节保存设备号+路由器LNK实体号，第二个字节保存主机号，保证ip在传递时的稳定。  
APP：  
v1.0：采用原始控制台模式，只能发送文字信息且无法编码  
v2.0：引入easy-x，创建一个图形界面，保存发送的信息的接受的信息  
v3.0：添加文件发送功能，并将图片在用户界面进行展示。

3、成员分工

张洋：阶段一参与讨论，阶段二停等协议、流量控制，阶段三、阶段四代码主要设计

蓝苛晋：项目一总体报告编写，阶段一参与讨论，阶段二差错检测，阶段三、四部分函数，应用层界面

邹坤：阶段一报告、套接字程序，阶段二帧定位，阶段三、四部分函数，应用层传输图片功能

（阶段二采取分割功能的方式进行分工，阶段三、四听取段老师意见，以张洋同学为代码总设计师，其余两名同学由他分配函数进行编写）

附录：

阶段一套接字实验代码：

//头文件

#include<stdio.h>

#include<winsock2.h>

#include<stdlib.h>

#include<time.h>

//获取链接的动态链接库

#pragma comment(lib,"ws2\_32.lib")

#define random(x) (rand()%x)

int receive(SOCKET clientSocket) //接收函数

{

//接收到的数据

char buf1[10];

memset(&buf1, '0', sizeof(buf1));

//num为数据的长度，num>0表示接收到了数据

int num = recv(clientSocket, buf1, strlen(buf1), 0);

if (num > 0) {

buf1[num] = '\0';

printf("服务器接收的数据是:%s\n", buf1);

//b为字符串数据化为整数

int b = atoi(buf1);

return b;

}

else

return -1;

}

int main(int argc, char\* argv[])

{

int ret;

//初始化套接字

WSADATA wsdata;

if (WSAStartup(MAKEWORD(2, 2), &wsdata) != 0)

{

printf("WSAStartup Error\n");

return 1;

}

//创建套接字

SOCKET serverSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (serverSocket == INVALID\_SOCKET)

{

printf("Create socket error");

WSACleanup();

return 1;

}

//绑定套接字

SOCKADDR\_IN sockAddr;

memset(&sockAddr, 0, sizeof(SOCKADDR\_IN));

sockAddr.sin\_family = AF\_INET;

sockAddr.sin\_addr.S\_un.S\_addr = INADDR\_ANY;

sockAddr.sin\_port = htons(7);

if (bind(serverSocket, (SOCKADDR\*)&sockAddr, sizeof(sockAddr)) == SOCKET\_ERROR) {

printf("Bind socket error\n");

closesocket(serverSocket);

WSACleanup();

return 1;

}

//监听

ret = listen(serverSocket, 10);

if (SOCKET\_ERROR == ret) {

printf("listen() error\n");

closesocket(serverSocket);

WSACleanup();

return 1;

}

//接收数据

SOCKADDR\_IN clientAddr;

int szClientAddr = sizeof(clientAddr);

printf("waitting to connect......\n");

SOCKET clientSocket = accept(serverSocket, (SOCKADDR\*)&clientAddr, &szClientAddr);

if (clientSocket == INVALID\_SOCKET) {

printf("accept error\n");

closesocket(serverSocket);

WSACleanup();

return 1;

}

else

printf("Get a connection: %s\n", inet\_ntoa(clientAddr.sin\_addr));

//收发数据

int b = 0;

//buf2为发送的数据

char buf2[10];

while ((b = receive(clientSocket)) >= 0) {

memset(&buf2, '0', sizeof(buf2));

srand((int)time(0));

//a为服务器在接收到用户的数据后产生的随机数，范围是[1,500]

int a = random(500);

if (a + b > 100) {

itoa(a + b, buf2, 10);

send(clientSocket, buf2, strlen(buf2), 0);

printf("服务器发送的数据是:%s\n", buf2);

}

//如果a+b<=100，则服务器发送10000给用户，其长度num是5，可以在用户程序中作判断

else {

itoa(10000, buf2, 10);

send(clientSocket, buf2, strlen(buf2), 0);

}

}

closesocket(serverSocket);

closesocket(clientSocket);

WSACleanup();

return 0;

}

**用户代码：**

//头文件

#include<stdio.h>

#include<winsock2.h>

#include<stdlib.h>

#include<time.h>

#include<windows.h>

//获取链接的动态链接库

#pragma comment(lib,"ws2\_32.lib")

#define random(x) (rand()%x)

clock\_t begin, end;

int receive(SOCKET clientSocket) //接收函数

{

char buf2[10];

memset(&buf2, '0', sizeof(buf2));

int num = recv(clientSocket, buf2, strlen(buf2), 0);

//若num的值是5，表明服务器发送的是10000，表明相加后的值<=100，表明用户不需立刻发送数据

if (num == 5) {

return 1;

}

//否则用户需立刻发送数据

else {

buf2[num] = '\0';

printf("用户接收的数据是:%s\n", buf2);

return 0;

}

}

int sendto(SOCKET clientSocket)//发送函数

{

char buf1[10];

memset(&buf1, '0', sizeof(buf1));

int a = random(500);

itoa(a, buf1, 10);

send(clientSocket, buf1, strlen(buf1), 0);

printf("用户发送的数据是:%s\n", buf1);

return 0;

}

int main(int argc, char\* argv[])

{

begin = clock();

//初始化套接字

WSADATA wsdata;

if (WSAStartup(MAKEWORD(2, 2), &wsdata) != 0)

{

printf("WSAStartup Error\n");

return 1;

}

//创建套接字

SOCKET clientSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (clientSocket == INVALID\_SOCKET)

{

printf("Create socket error");

WSACleanup();

return 1;

}

// 连接服务端

sockaddr\_in sock\_in;

sock\_in.sin\_family = AF\_INET;

sock\_in.sin\_port = htons(7);

sock\_in.sin\_addr.S\_un.S\_addr = inet\_addr("192.168.1.100");

if (connect(clientSocket, (sockaddr\*)&sock\_in, sizeof(sock\_in)) == SOCKET\_ERROR)

{

printf("Connect error");

return 1;

}

else

printf("Connect Successfully\n");

//收发数据

sendto(clientSocket);//先发送一个数据

srand((int)time(0));

for (int i = 0; i < 19; i++)

{

//之后判断服务器是否发送了数据，发送了的话用户就需立即回发一个数据

if (receive(clientSocket) == 0) {

sendto(clientSocket);

}

//没有的话，用户就隔500ms发送数据

else {

Sleep(500);

sendto(clientSocket);

}

}

closesocket(clientSocket);

WSACleanup();

end = clock();

printf("程序运行时间是：\n");

printf("time=%d ms\n", (end - begin));

system("pause");

return 0;

}