《信息安全工程实践一》
实践报告

姓　　名	
班　　级	
程序实践名称	
程序实践内容	
开设学期	
开设时间	
报告日期	
评定成绩	

东北大学软件学院

实验 3.1 基于 Socket 编程的时间服务器

一、程序实践概述

1、题目名称：

Socket 编程基础：时间服务器

2、时间进度：

2022.06.20 - 2022.06.28

3、开发环境：

Linux Ubuntu 18.04.6，Linux CentOS release 7.0(Final)

Visual studio code C/C++ May 2022 (version 1.68)

二、问题分析

1、功能说明：

客户端向服务端发起连接，服务器给客户端返回时间。

2、解决方案：

程序的整体主要分为三个部分。首先是服务器和客户端建立连接，然后是客户端获取服务器的基准时间,最后是服务器根据客户端访问情况将对应的客户端的 IP 地址和客户端申请的基准时间写入到文件中。

三、方案设计

1、模块结构：

客户端：timeclient.c：

函数	说明
int main(void)	负责建立和服务器的连接，并且从服务器端获取基准时间

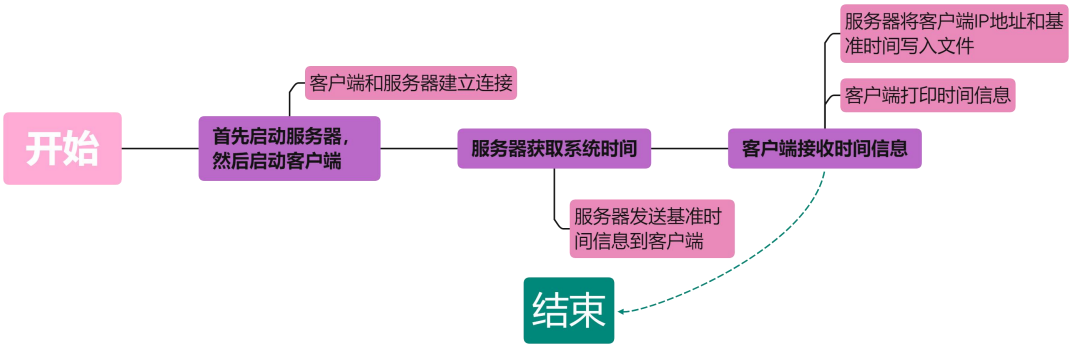
服务端：timeserver.c：

函数	说明
int main(void)	负责接受申请连接的客户端，并与之建立连接。一旦连接建立成功，则发送服务器基准时间给客户端
void write_file(const char* filename,char* ip_address,char* time)	根据传入的文件名称，写入相应的 ip 地址和时间。日志功能

2、数据结构：

```
#define PORT 13333//The server listens on the port number, because the port number is less than 10000 is the port used by the operating system
#define BACKLOG 10//Maximum number of simultaneous connection requests
```

3、 总体流程：



4、 关键算法：

(1)日志记录功能

```
void write_file(const char* filename,char* ip_address,char* time){
    FILE *fd=fopen(filename,"a+");
    fprintf(fd,"IP Address: %s | Time: %s \n",ip_address,time);
    fclose(fd);
}
```

根据传入的文件名称 filename，写入相应的 ip 地址 ip_address 和时间 time。

(2)服务器端循环监听

```
while(true){
    addrlen = sizeof(client);
    if((connectfd = accept(sockfd, (struct sockaddr *)&client, &addrlen )) == -1){
        perror("accept() error!");
        continue;
    }
    t=time(NULL);
    cout<<"************************************************"<<endl;
    cout<<"Received a connection from :"<<inet_ntoa(client.sin_addr)<<endl;
    cout<<"Timeclient's IP is: "<<inet_ntoa(client.sin_addr)<<endl;
    cout<<"The current time is: "<<ctime(&t)<<endl;
    cout<<"************************************************"<<endl;
    const char* filename="/home/neu_ljh/timerecords";
    write_file(filename,inet_ntoa(client.sin_addr),ctime(&t));
    if(!fork()){
        if(send(connectfd,(time_t *)&t,sizeof(time_t),0)==-1){
            perror("send() error!");
        }
        close(connectfd);
        return 0;
    }
    close(connectfd);
}
```

此部分为 while 死循环，服务器不断地等待、接收客户端的连接请求，如果接收到客户端的连接请求，则立即获取当前的时间戳。然后发送当前时间戳给客户端。并且调用函数 void write_file(const char* filename,char* ip_address,char* time)进行日志记录。

(3) 客户端入口参数

```
/*get ip address*/
if((host= gethostbyname(argv[1])) == NULL){
    herror("gethostbyname() error!\n");
    return 0;
}
```

客户端 main 函数接收一个入口参数，IP 地址

四、调试记录

1. 启动服务器

```
[neu_ljh@localhost c]$ sudo ./timeserver
[sudo] neu_ljh 的密码：
Waiting for the client to connect to this server...
```

2.启动客户端

```
[neu_ljh@localhost c]$ sudo ./timeclient
[sudo] neu_ljh 的密码：
please input the ip address while computing the program!
[neu_ljh@localhost c]$
```

客户端 main 函数需要一个入口参数，即 IP 地址，若未输入则无法正常使用时间功能。上图为失败界面。下图为成功界面。

```
[neu_ljh@localhost c]$ sudo ./timeclient 127.0.0.1
[sudo] neu_ljh 的密码：
According to the TimeServer:Time is ---- Sat Jun 25 01:54:33 2022

[neu_ljh@localhost c]$
```

同时，服务端也会同步打印数据：

```
[neu_ljh@localhost c]$ sudo ./timeserver
[sudo] neu_ljh 的密码：
Waiting for the client to connect to this server...
************************************************
Received a connection from :127.0.0.1
Timeclient's IP is: 127.0.0.1
The current time is: Sat Jun 25 01:54:33 2022

************************************************
```

3.服务器循环监听客户端的连接请求

客户端：

```
[neu_ljh@localhost c]$ sudo ./timeclient
[sudo] neu_ljh 的密码：
please input the ip address while computing the program!
[neu_ljh@localhost c]$ sudo ./timeclient 127.0.0.1
[sudo] neu_ljh 的密码：
According to the TimeServer:Time is ---- Sat Jun 25 01:54:33 2022

[neu_ljh@localhost c]$ sudo ./timeclient 127.0.0.1
According to the TimeServer:Time is ---- Sat Jun 25 01:58:24 2022

[neu_ljh@localhost c]$
```

服务端：

```
[neu_ljh@localhost c]$ sudo ./timeserver
[sudo] neu_ljh 的密码：
Waiting for the client to connect to this server...
************************************************
Received a connection from :127.0.0.1
Timeclient's IP is: 127.0.0.1
The current time is: Sat Jun 25 01:54:33 2022

************************************************
************************************************
Received a connection from :127.0.0.1
Timeclient's IP is: 127.0.0.1
The current time is: Sat Jun 25 01:58:24 2022

************************************************
```

五、创新说明

1. 增加服务端日志记录

实验 3.2 基于 Socket 编程的远程文件备份服务器

一、程序实践概述

1、题目名称：

Socket 编程基础：远程文件备份服务器

2、时间进度：

2022.06.20 - 2022.06.28

3、开发环境：

Linux Ubuntu 18.04.6，Linux CentOS release 7.0(Final)

Visual studio code C/C++ May 2022 (version 1.68)

二、问题分析

1、功能说明：

(1)客户端指定一个文件并将文件名发送给服务端，通过 Socket 通信将文件内容发送给服务端。服务端进行同步接受。

(2)服务端创建响应的文件，并将服务端发送过来的文件内容同步写入到创建的文件并保存。

2、解决方案：

程序的整体主要分为三个部分。首先是服务器和客户端建立连接，然后是客户端获取用户输入的文件绝对路径，然后客户端读取此文件并将文件内容发送给服务端。最后是服务器获取文件的备份路径，根据客户端发送过来的内容同步的写入到文件中并保存。

三、方案设计

1、模块结构：

客户端：fileclient.cpp：

函数	说明
int main(void)	**负责建立和服务器的连接，客户端获取用户输入的文件绝对路径，然后客户端读取此文件并将文件内容发送给服务端**

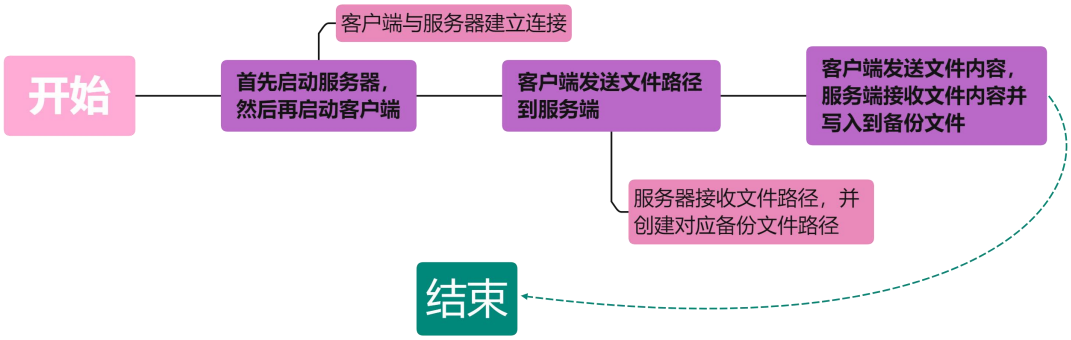
服务端：fileserver.cpp：

函数	说明
int main(void)	**负责接受申请连接的客户端，并与之建立连接。一旦连接建立成功，则服务器获取文件的备份路径，根据客户端发送过来的内容同步的写入到文件中并保存**

2、数据结构：

```
#define PORT 13334
#define BACKLOG 5
#define MAXSIZE 32
```

3、总体流程：



4、关键算法：

(1)客户端获取文件路径并发送文件内容

```
cout<<"connect success!"<<endl;
cout<<"Please input file's directory: "<<endl;
cin>>dir;
FILE *fd= fopen(dir.c_str(),"r+");
while((num=fread(buf,1,MAXSIZE,fd))>0){
    send(sockfd,buf,num,0);
}
```

客户端获取用户输入的文件绝对路径，打开文件并发送文件内容。

(2)服务端获取备份文件路径并接收文件内容

```
cout<<"Please input directory of copy: "<<endl;
cin>>dir;
fd=fopen(dir.c_str(),"wb");
while(num=recv(connectfd,buf,1,0)>0)
{
    fwrite(buf,1,num,fd);
}
```

服务端建立根据备份文件路径建立备份文件，并将接收到的数据写入到备份文件里。

(3)客户端入口参数

```
/*get ip address*/
if((host= gethostbyname(argv[1])) == NULL){
    herror("gethostbyname() error!\n");
    return 0;
}
```

(4)反馈通信

客户端：

```
cout<<"Please input file's directory: "<<endl;
cin>>dir;
FILE *fd= fopen(dir.c_str(),"r+");
if(fd==NULL){
    cout<<"The refered file didn't exist!"<<endl;
    strcpy(res,"404");
    cout<<"response: "<<res<<endl;
}else{
    strcpy(res,"200");
    cout<<"response: "<<res<<endl;
}
write(sockfd,res,strlen(res));
```

判断文件路径是否合法。合法则发送“200”给服务端，否则发送“404”。

服务端：

```
if(strcmp("404",response)==0){
    cout<<"The server detected that the file specified by the client does not exist! File backup
    return 0;
}else{
    cout<<"Please input directory of copy: "<<endl;
    cin>>dir;
    fd=fopen(dir.c_str(),"wb");
    while(num=recv(connectfd,buf,1,0)>0)
    {
        fwrite(buf,1,num,fd);
    }
    fclose(fd);
    close(connectfd);
    cout<<"The file was successfully received by the server and the backup was successful!"<<end
    cout<<"please check in about 10 seconds, or later..."<<endl;
}
```

服务端接收到“200”时才会开始文件备份功能。否则直接退出。

四、调试记录

1. 启动服务器

```
[neu_ljh@localhost c]$ g++ -o fileserver fileserver.cpp
[neu_ljh@localhost c]$ ./fileserver
Current Time: Sat Jun 25 02:37:53 2022

Waiting for the client to connect to this server...
```

2.启动客户端

```
[neu_ljh@localhost c]$ g++ -o fileclient fileclient.cpp
[neu_ljh@localhost c]$ ./fileclient
Current Time: Sat Jun 25 02:38:35 2022

The client is connecting to the server...
please in put the ip address!
[neu_ljh@localhost c]$
```

客户端 main 函数需要一个入口参数，即 IP 地址，若未输入则无法正常使用文件备份功能。

上图为失败界面。下图为成功界面。

```
[neu_ljh@localhost c]$ g++ -o fileclient fileclient.cpp
[neu_ljh@localhost c]$ ./fileclient
Current Time: Sat Jun 25 02:38:35 2022

The client is connecting to the server...
please in put the ip address!
[neu_ljh@localhost c]$ ./fileclient 127.0.0.1
Current Time: Sat Jun 25 02:40:07 2022

The client is connecting to the server...
connect success!
Please input file's directory:
```

服务端显示连接成功：

```
[neu_ljh@localhost c]$ g++ -o fileserver fileserver.cpp
[neu_ljh@localhost c]$ ./fileserver
Current Time: Sat Jun 25 02:37:53 2022

Waiting for the client to connect to this server...
It's Connected!
```

3.文件备份功能

选择备份主目录下的 timerecords 文件，文件内容如下：

客户端传输文件：

```
[neu_ljh@localhost c]$ ./fileclient 127.0.0.1
Current Time: Sat Jun 25 02:47:04 2022

The client is connecting to the server...
connect success!
Please input file's directory:
/home/neu_ljh/timerecords
The file specified by the user has been sent to the server!
please check in about 10 seconds, or later...
[neu_ljh@localhost c]$
```

服务端接收文件并保存文件：

```
[neu_ljh@localhost c]$ ./fileserver
Current Time: Sat Jun 25 02:46:44 2022

Waiting for the client to connect to this server...
It's Connected!
Please input directory of copy:
/home/neu_ljh/timerecords_copy
The file was successfully received by the server and the backup was successful!
please check in about 10 seconds, or later...
[neu_ljh@localhost c]$
```

查看文件，发现文件备份成功！：

五、创新说明

1. 增加服务端时间记录

```
[neu_ljh@localhost c]$ ./fileserver
Current Time: Sat Jun 25 13:27:53 2022

Waiting for the client to connect to this server...
It's Connected!
```

2.新增对于客户端文件路径不存在的及时反馈

例如我们在客户端输入一个不存在的文件：/home/neu_ljh/nosuchfile

```
[neu_ljh@localhost c]$ g++ -o fileclient fileclient.cpp
[neu_ljh@localhost c]$ ./fileclient 127.0.0.1
Current Time: Sat Jun 25 13:28:04 2022

The client is connecting to the server...
connect success!
Please input file's directory:
/home/neu_ljh/nosuchfile
The refered file didn't exist!
response: 404
The refered file didn't exist.Program terminated!
```

客户端会提示文件不存在，并且将消息“404”发送给服务端(消息“200”表示路径存在)。然后服务端根据客户端的消息是“200”还是“400”来判断下一步的逻辑操作。当服务端接收到“404”信号时：

```
[neu_ljh@localhost c]$ ./fileserver
Current Time: Sat Jun 25 13:27:53 2022

Waiting for the client to connect to this server...
It's Connected!
response: 404
The server detected that the file specified by the client does not exist! Fil
e backup failed!
```

服务端接收到“404”，提示文件不存在，终止程序。

实验 3.3 基于 Socket 编程的带身份认证功能的远程数据备份服务器

一、程序实践概述

1、题目名称：

Socket 编程基础：带身份认证的远程数据备份服务器

2、时间进度：

2022.06.20 - 2022.06.28

3、开发环境：

Linux Ubuntu 18.04.6，Linux CentOS release 7.0(Final)

Visual studio code C/C++ May 2022 (version 1.68)

二、问题分析

1、功能说明：

(1)对需要使用数据备份的新用户提供注册功能：注册功能完成后，用户可以选择登录。用户成功登录后，可以正常使用数据备份功能。

(2)用户在登录功能中，实现身份认证的功能：服务端连接 RSA 密钥系统生成公钥 public_key_rsa 和私钥 private_key_rsa，并将公钥 public_key_rsa 传输给客户端。客户端接收来自服务端的 public_key_rsa 后，连接 DES 密钥系统生成密钥 des_key,并使用 public_key_rsa 加密 des_key,再发送给服务端。服务端接收后使用 private_key_rsa 解密得到 des_key。服务器和客户端都连接 MD5 系统。客户端对输入的用户名和密码分别先后进行 MD5 加密、DES 加密得到数字签名的结果，再结合原本未处理的用户输入的用户名和密码一起发送给服务端。服务端对数字签名进行解密得到结果 A，并使用 MD5 加密得到的未处理的用户名和密码得到结果 B，将结果 A 和结果 B 进行比对。若 A 和 B 不相等，则用户的身份验证失败；否则，则身份验证成功。身份验证成功后，服务器会对用户的用户名和密码进行匹配检查，若档案库中存在用户信息，则登陆成功。用户成功登录后，可以正常使用数据备份功能。

(3)提供数据备份功能：客户端指定一个文件并将文件名发送给服务端，通过 Socket 通信将文件内容发送给服务端。服务端进行同步接受。

(4)提供数据备份功能：服务端创建响应的文件，并将服务端发送过来的文件内容同步写入到创建的文件并保存。

2、解决方案：

程序的整体主要分为三个部分。

①服务器和客户端建立连接。

②新用户需要注册才能正常使用数据备份功能。提示用户，并引导用户完成信息的注册。

③用户的登录功能中，模拟了数字签名的流程，新增了身份管理和身份认证的功能。

④身份验证成功后，客户端获取用户输入的文件绝对路径，然后客户端读取此文件并将文件内容发送给服务端。

⑤服务器获取文件的备份路径，根据客户端发送过来的内容同步的写入到文件中并保存。

三、方案设计

1、模块结构：

MD5 系统：md5.h,md5.cpp

函数	说明
void MD5Init(MD5_CTX *context);	MD5 结构体的初始化
void MD5Update(MD5_CTX *context,unsigned char *input,unsigned int inputlen);	MD5 结构体的更新操作
void MD5Final(MD5_CTX *context,unsigned char digest[16]);	根据输入内容得到 MD5 最终的加密结果
void MD5Transform(unsigned int state[4],unsigned char block[64]);	MD5 操作的移位、变换和迭代
void MD5Encode(unsigned char *output,unsigned int *input,unsigned int len);	MD5 的加密操作
void MD5Decode(unsigned int *output,unsigned char *input,unsigned int len);	MD5 的解密操作
string get_md5(string hexbuf)	返回一个字符串作为 MD5 最终的加密结果

RSA 系统：rsa.cpp

函数	说明
vector<string> split(const string& str, const string& delim)	自定义的字符串分割函数，根据入口参数 delim 来对字符串 str 进行切割。并返回一个字符串数组
string int_to_string(int x)	自定义的将 int 类型变量转换为 string 类型变量
int get_inverse(int e, int n)	求取 e 关于模 n 的逆元
int Gcd(int a, int b)	求最大公因数
int getrand(int p,int q)	产生满足条件的随机数 d
string Encode(int e,int n,string plaintext)	RSA 的加密操作
string get_ciphertext_no_comma(string ciphertext_comma)	RSA 的获取优化后密文的操作
string Decode(int d, int n,string ciphertext_comma)	RSA 的解密操作

int isPrime(int n)	**判断入口参数 n 是否是质数**
void get_prime_num(int &p,int &q)	**得到随机的不相同的两个随机数 p 和 q**

DES 系统：des.cpp

函数	说明
string int2BinString(int n)	**int 转四位 string 和 int 十进制转 string 二进制**
string hexToTwo(string str)	**string 十六进制转 string 二进制**
int binToDec(string bin)	**string 二进制转 int 十进制**
int str2Dec(string str)	**01 字符转十进制**
string Bin2Hex(string strBin)	**64 位密文转十六进制**
string exchange(string str, int rule[], int x)	**利用交换表进行置换**
string circleMove(string str, int j)	**依据移位表进行移位**
string spiltShift(string str, int j)	**左右两部分移位**
string XOR(string str1, string str2)	**string 异或操作**
string SBoxWork(string str, int SBox[][4][16])	**S 盒工作**
string encryption(string MingWen,string Key)	**DES 的加密操作**
string decryption(string MiWen,string Key)	**DES 的解密操作**
string get_des_key()	**随机生成 DES 的密钥**

客户端：fileclientplus.cpp：

函数	说明
int main(void)	**负责建立和服务器的连接和其他杂项功能。完成接收并解析 RSA 公钥，发送 DES 密钥。读取用户选择，当满足条件时，客户端获取用户输入的文件绝对路径，然后客户端读取此文件**

	并将文件内容发送给服务端。
void login(int &sockfd,string des_key)	客户端提供的登录功能。读取用户输入的账号密码并进行加密传输，同时接收来自服务端的关于身份验证的响应
void regist(int &sockfd,string des_key)	客户端提供的注册功能。读取用户输入的账号密码并进行加密传输。
void tips(int &sockfd,string des_key)	选项函数。根据用户需要进行的功能，读取用户的选择。分别调用函数 void login 或者 void regist

服务端：fileserverplus.cpp：

函数	说明
int main(void)	负责接受申请连接的客户端，并与之建立连接。一旦连接建立成功，发送 RSA 公钥，接受并解析 DES 密钥，接收来自客户端用户对于功能的选择以达到功能同步的效果，进行身份管理。在登录功能下，服务端对用户进行身份验证，若身份验证通过，则继续验证身份是否合法，若合法，则服务器获取文件的备份路径，根据客户端发送过来的内容同步的写入到文件中并保存。
void RSA_get_key()	随机生成 RSA 的公钥和私钥
void write_file(const char* filename,const char* username,const char* password)	将入口参数的用户名和密码写入到指定文件路径中

2、数据结构：

(1)Socket 宏定义

```
#define MIN 15
#define MAX 100 //随机数产生的范围
#define PORT 13334
#define BACKLOG 5
#define MAXSIZE 32
#define MAXDATASIZE 1000
```

(2)DES 部分的基本数据结构

```
const static int Key_SIZE = 16;
const static int ShiftTable_SIZE = 16;
const static int PC_2_SIZE = 48;
```

```
const static int IP_SIZE = 64;
const static int E_SIZE = 48;
const static int P_SIZE = 32;
const static int IP_1_SIZE = 64;
const static int ExchangeRules_SIZE = 56;
const static string Bin_Hex[16]=
{
    "0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"
};
//交换规则表（8*7）
    int ExchangeRules[56] =
{
    57, 49, 41, 33, 25, 17,  9,
    1, 58, 50, 42, 34, 26, 18,
    10,  2, 59, 51, 43, 35, 27,
    19, 11,  3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
    7, 62, 54, 46, 38, 30, 22,
    14,  6, 61, 53, 45, 37, 29,
    21, 13,  5, 28, 20, 12,  4
};
//移位表
int ShiftTable[16] =
{
    1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
};
//PC-2（8*6）
int PC_2[48] =
{
    14, 17, 11, 24,  1,  5,
    3, 28, 15,  6, 21, 10,
    23, 19, 12,  4, 26,  8,
    16,  7, 27, 20, 13,  2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
};
//IP（8*8）
int IP[64] =
{
    58, 50, 42, 34, 26, 18, 10,  2,
    60, 52, 44, 36, 28, 20, 12,  4,
    62, 54, 46, 38, 30, 22, 14,  6,
```

```
    64, 56, 48, 40, 32, 24, 16,  8,
    57, 49, 41, 33, 25, 17,  9,  1,
    59, 51, 43, 35, 27, 19, 11,  3,
    61, 53, 45, 37, 29, 21, 13,  5,
    63, 55, 47, 39, 31, 23, 15,  7
};
//扩展置换 E（8*6）
int E[48] =
{
    32,  1,  2,  3,  4,  5,
    4,  5,  6,  7,  8,  9,
    8,  9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
    28, 29, 30, 31, 32,  1
};
```

```
//S 盒
int SBox[8][4][16] =
{
    {
        {14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7},
        {0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8},
        {4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0},
        {15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13}
    },
    {
        {15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10},
        {3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5},
        {0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15},
        {13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9}
    },
    {
        {10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8},
        {13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1},
        {13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7},
        {1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12}
    },
    {
        {7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15},
        {13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9},
        {10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4},
```

```
        {3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14}
    },
    {
        {2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9},
        {14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6},
        {4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14},
        {11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3}
    },
    {
        {12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11},
        {10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8},
        {9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6},
        {4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13}
    },
    {
        {4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1},
        {13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6},
        {1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2},
        {6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12}
    },
    {
        {13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7},
        {1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2},
        {7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8},
        {2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11}
    }
};
//P 盒（8*4）
int P[32] =
{
    16,  7, 20, 21,
    29, 12, 28, 17,
    1, 15, 23, 26,
    5, 18, 31, 10,
    2,  8, 24, 14,
    32, 27,  3,  9,
    19, 13, 30,  6,
    22, 11,  4, 25
};
//IP-1（8*8）
int IP_1[64] =
{
    40, 8, 48, 16, 56, 24, 64, 32,
    39, 7, 47, 15, 55, 23, 63, 31,
```

```
    38, 6, 46, 14, 54, 22, 62, 30,
    37, 5, 45, 13, 53, 21, 61, 29,
    36, 4, 44, 12, 52, 20, 60, 28,
    35, 3, 43, 11, 51, 19, 59, 27,
    34, 2, 42, 10, 50, 18, 58, 26,
    33, 1, 41,  9, 49, 17, 57, 25
};
```

(3)MD5 部分的基本数据结构

1)结构体

```
typedef struct
{
    unsigned int count[2];
    unsigned int state[4];
    unsigned char buffer[64];
}MD5_CTX;
```

2)基本运算的宏定义

```
#define F(x,y,z) ((x & y) | (~x & z))
#define G(x,y,z) ((x & z) | (y & ~z))
#define H(x,y,z) (x^y^z)
#define I(x,y,z) (y ^ (x | ~z))
#define ROTATE_LEFT(x,n) ((x << n) | (x >> (32-n)))
#define FF(a,b,c,d,x,s,ac) \
          { \
          a += F(b,c,d) + x + ac; \
          a = ROTATE_LEFT(a,s); \
          a += b; \
          }
#define GG(a,b,c,d,x,s,ac) \
          { \
          a += G(b,c,d) + x + ac; \
          a = ROTATE_LEFT(a,s); \
          a += b; \
          }
#define HH(a,b,c,d,x,s,ac) \
          { \
          a += H(b,c,d) + x + ac; \
          a = ROTATE_LEFT(a,s); \
          a += b; \
          }
#define II(a,b,c,d,x,s,ac) \
          { \
          a += I(b,c,d) + x + ac; \
          a = ROTATE_LEFT(a,s); \
```

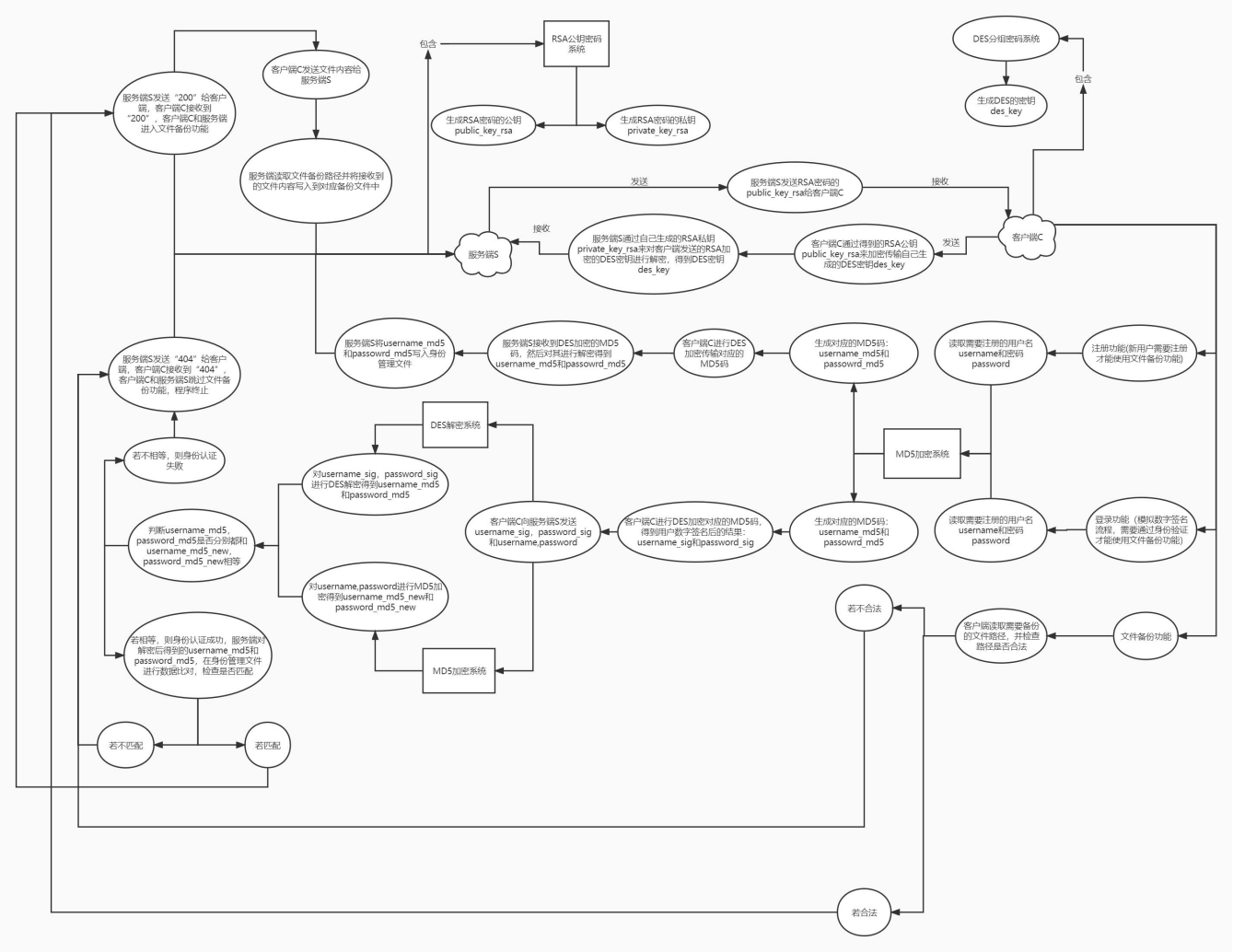
```
        a += b; \
        }
```

3)偏移量

```
unsigned char PADDING[]={0x80,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                         0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

3、总体流程：

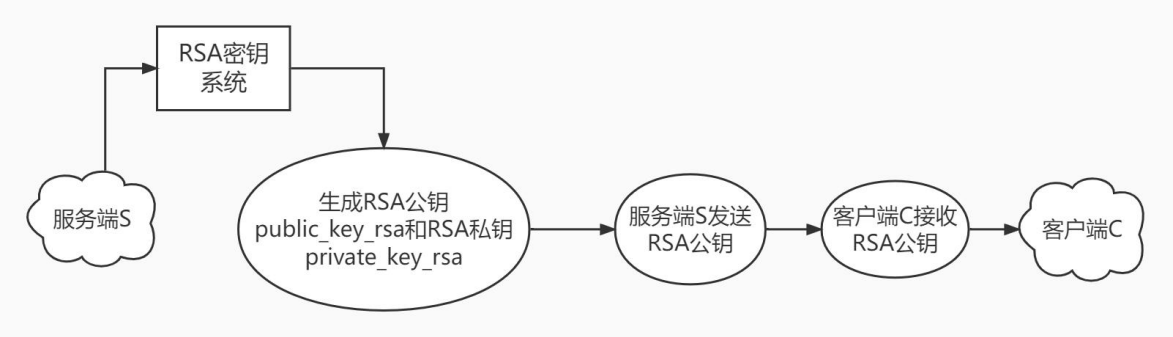
(1)实验方案总流程



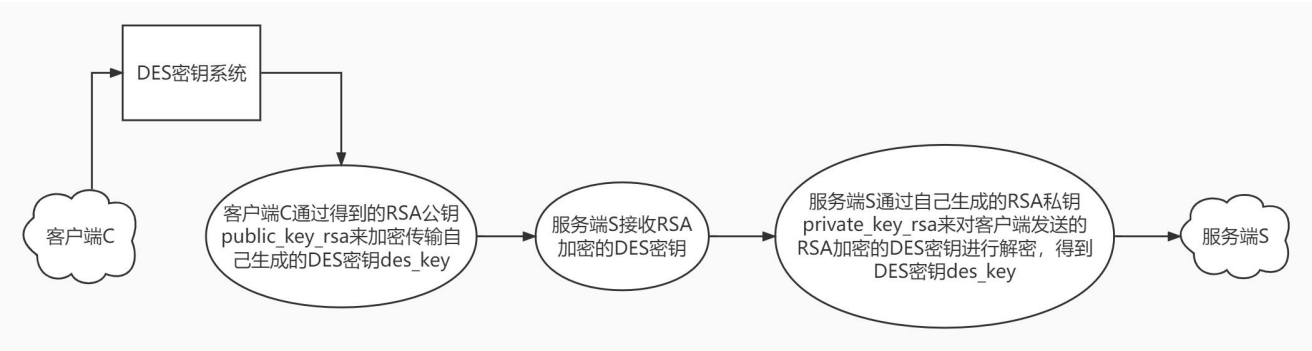
本流程图要素点过多，主要涵盖 6 个方面。下面分别给出此 6 个重要部分的流程图(清晰流程图图片打包在附件，老师可以自行查看。)

(2)功能详细流程图

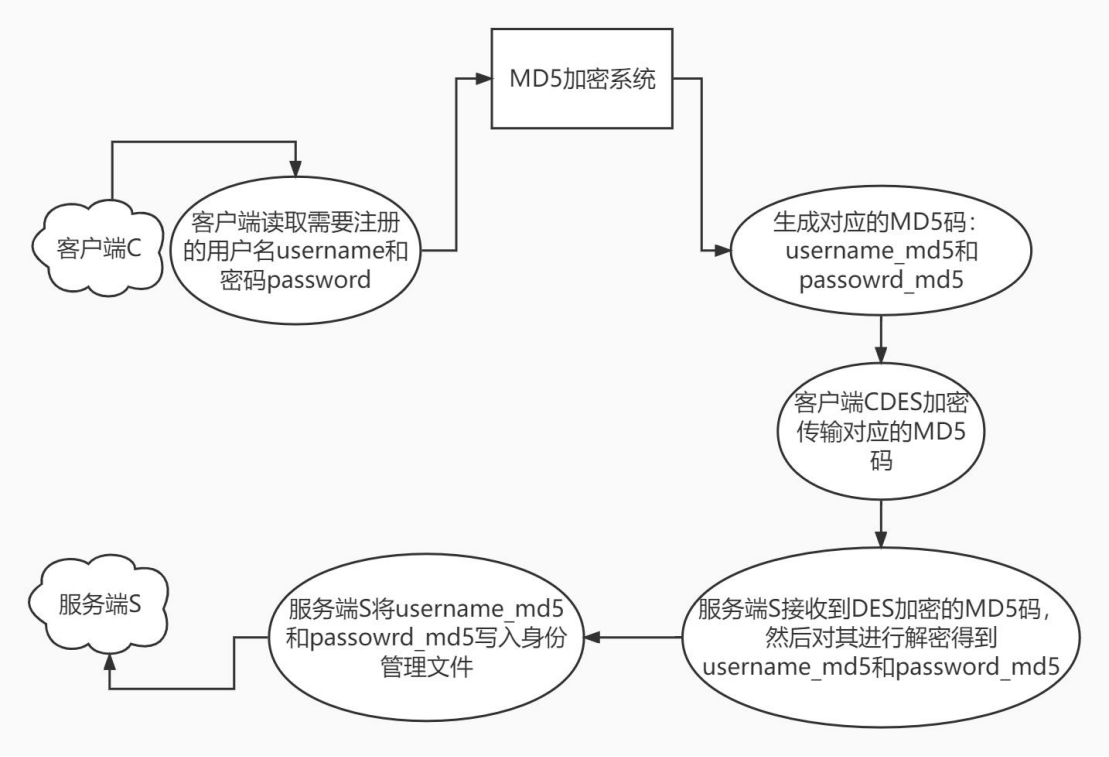
①RSA 密钥的生成与发送



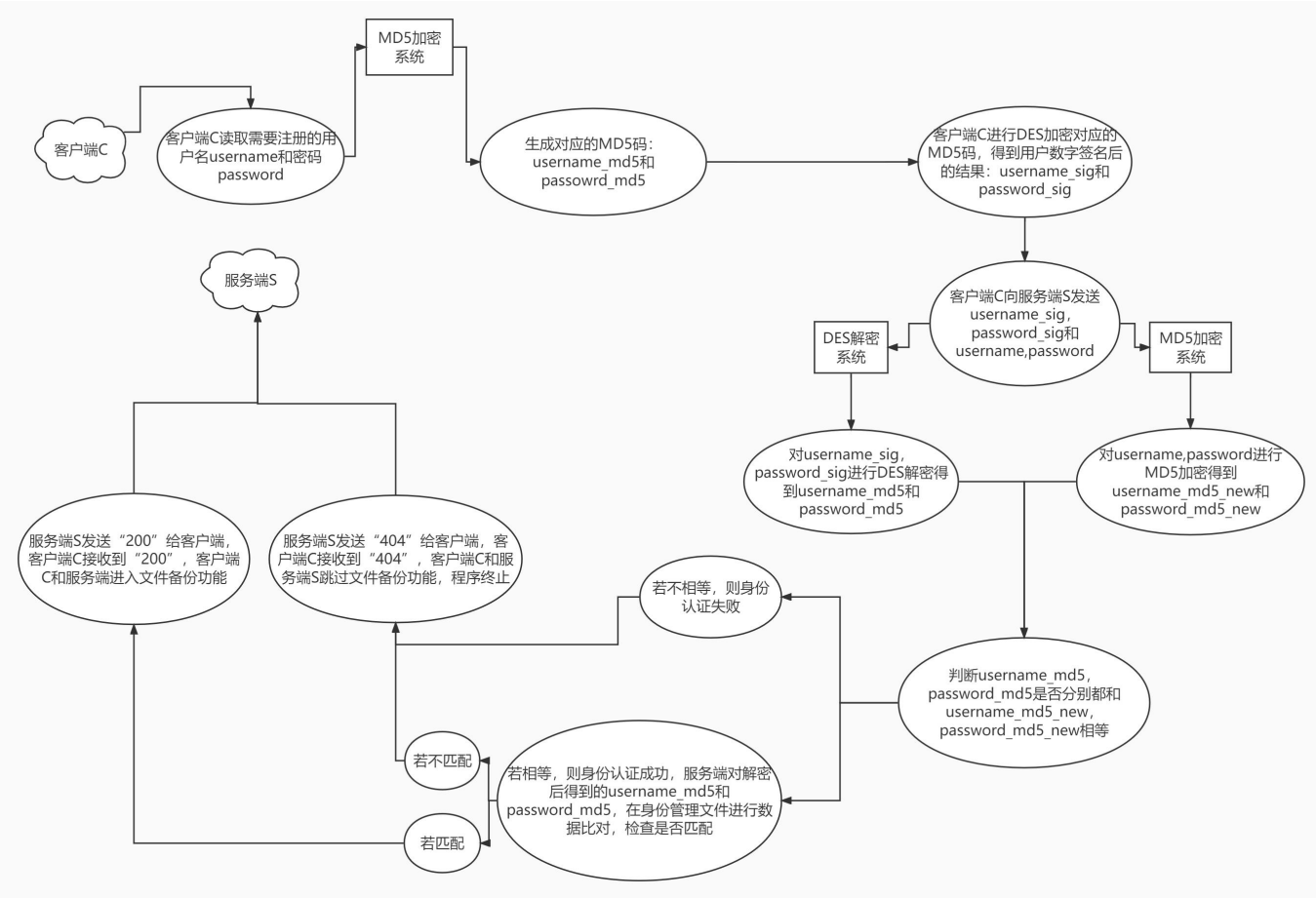
②DES 密钥的生成与发送



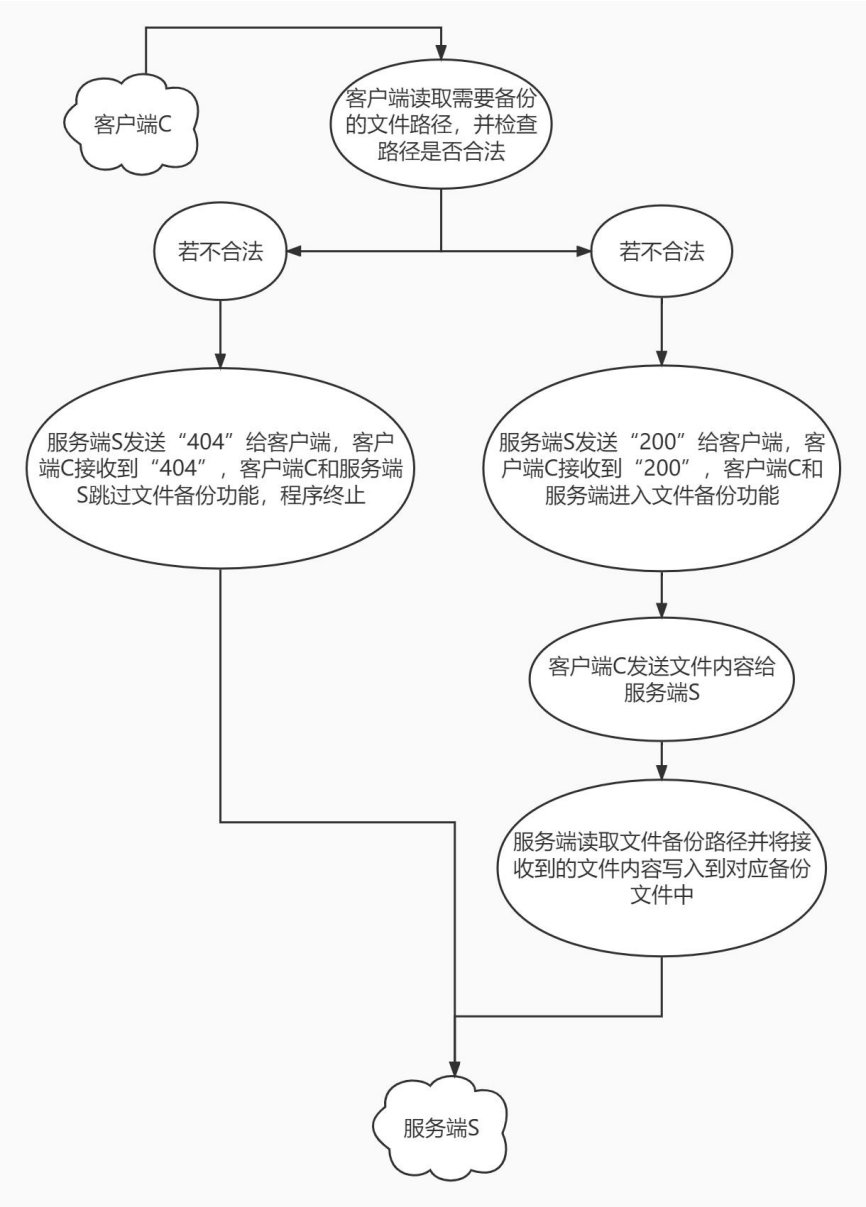
③注册功能的实现(含身份管理)



④登录功能的实现(含身份认证）



⑤数据备份功能


客户端C
客户端读取需要备份的文件路径，并检查路径是否合法
若不合法
若不合法
服务端S发送“404”给客户端，客户端C接收到“404”，客户端C和服务端S跳过文件备份功能，程序终止
服务端S发送“200”给客户端，客户端C接收到“200”，客户端C和服务端进入文件备份功能
客户端C发送文件内容给服务端S
服务端读取文件备份路径并将接收到的文件内容写入到对应备份文件中
服务端S

4、关键算法：

(1)RSA 密钥的随机生成

```
void get_prime_num(int &p,int &q){
    int i;
    int a;
    srand((unsigned)time(NULL));
    while(1){
        a=MIN + (rand() % (MAX-MIN));
        if(isPrime(a)){
            p=a;
            break;
        }else{
            continue;
        }
    }
    while(1){
        a=MIN + (rand() % (MAX-MIN));
        if(isPrime(a)){
            q=a;
            if(q==p){
                continue;
            }else{
                break;
            }
        }else{
            continue;
        }
    }
}
```

首先是生成两个随机的不等的素数 p 和 q(素数的大小范围可以根据宏定义的 MIN 和 MAX 数值来调整)。然后根据这两个素数 p 和 q，来生成 RSA 的公钥和私钥，并进行封装：

```
void RSA_get_key(){
    RSA rsa;
    rsa.get_prime_num(prime_p,prime_q);
    n=prime_p*prime_q;
    m=(prime_p-1)*(prime_q-1);
    d=rsa.getrand(prime_p,prime_q);
    e=rsa.get_inverse(rsa.getrand(prime_p,prime_q),m);
    private_key=private_key+rsa.int_to_string(d);
    public_key=public_key+rsa.int_to_string(e);

    private_key=private_key+","+rsa.int_to_string(n);
    public_key=public_key+","+rsa.int_to_string(n);
}
```

(2)DES 密钥的随机生成

```
string get_des_key(){
    int i;
    string res="";
    srand((unsigned)time(NULL));
    for(i=0;i<16;i++){
        res=res+Bin_Hex[rand()%16];
    }
    return res;
}
```

(3)RSA 的加密和解密

加密：

```
string Encode(int e,int n,string plaintext) {
    string ciphertext_comma="";
    int flag = 1;
    for (int i = 0; i < plaintext.size(); i++) {
        for (int j = 0; j < e; j++) {
            flag = flag * (int)plaintext.at(i) % n;
        }
        //ciphertext=ciphertext+to_string(flag);
        if(i!=0){
            ciphertext_comma=ciphertext_comma+","+int_to_string(flag);
        }else{
            ciphertext_comma=ciphertext_comma+int_to_string(flag);
        }
        flag = 1;
    }
    return ciphertext_comma;
}
```

此函数会得到带','的密文，还需要进一步处理得到最终的密文：

```
string get_ciphertext_no_comma(string ciphertext_comma){
    vector<string> res=split(ciphertext_comma,",");
    string ciphertext="";
    for(int i=0;i<res.size();i++){
        ciphertext=ciphertext+res[i];
    }
    return ciphertext;
}
```

解密：

```
string Decode(int d, int n,string ciphertext_comma){
    int flag = 1;
    string plaintext;
    vector<string> res=split(ciphertext_comma,",");
    for (int i = 0; i < res.size(); i++) {
        for (int j = 0; j < d; j++) {
            flag = (flag * atoi(res[i].c_str()))%n;
        }
        plaintext=plaintext+char(flag);
        flag = 1;
    }
    return plaintext;
}
```

(4)DES 的加解密

加密：

```
   string encryption(string MingWen,string Key)
   {
       string M = hexToTwo(MingWen);
       string K = hexToTwo(Key);
       string KKK = exchange(K, ExchangeRules, ExchangeRules_SIZE);/***处理密钥，生成 16 个子密
钥 **/
       int i = 0; /* 利用规则交换表（8*7）将 K 转换成 K0 ；  K0(56 位) = C0(28 位) + D0(28 位) */
       string K_arr[Key_SIZE+1];/* 利用移位表转换得 C1D1----C16D16,存入 K_arr */
       K_arr[0] = KKK;
       for(i=1; i<=Key_SIZE; i++)
       {
           K_arr[i] = spiltShift(K_arr[i-1], ShiftTable[i-1]);
       }
       string Key_arr[Key_SIZE];/* Kn（48 位）= PC-2（8*6）处理 CnDn 得 16 个子密钥，存入 Key_arr */
       for(i=0; i<Key_SIZE; i++)
       {
           Key_arr[i] = exchange(K_arr[i+1], PC_2, PC_2_SIZE);
       }/*** 用子密钥对明文加密**/
       string IP_M = exchange(M, IP, IP_SIZE);/* 通过 IP（8*8）处理 M 得 L0（32 位）  R0（32 位） */
       string L[Key_SIZE+1];/* Ln= R(n-1); Rn= L(n-1) + f(R(n- 1); Kn)迭代 16 次 */
       string R[Key_SIZE+1];
       L[0] = IP_M.substr(0, M.length()/2);
       R[0] = IP_M.substr(M.length()/2);
       string it = "";
       for(i=1; i<=Key_SIZE; i++){
           it = exchange(R[i-1], E, E_SIZE);//将 R0 通过扩展置换 E（8*6）从 32 位扩展到 48 位
           it = XOR(it, Key_arr[i-1]);//R0（48 位）与 K1 异或得 E0（48 位）
           it = SBoxWork(it, SBox);//将 E0（48 位）通过 S 盒转换成 32 位
```

```
            it = exchange(it, P, P_SIZE);//P 盒（8*4）置换，得 P0
            it = XOR(it, L[i-1]);//P0 与 L0 进行异或，得 J0
            L[i] = R[i-1];//左右交换位置，即 R1 = J0; L1 = R0
            R[i] = it;
        }/* 对 R16 L16 进行一次 IP-1（8*8）排序得密文 */
        string res = "";
        res += R[16];
        res += L[16];
        string finalRes = Bin2Hex(exchange(res, IP_1, IP_1_SIZE));
        return finalRes;
    }
```

解密：

```
string decryption( string MiWen,string Key)
    {
        string M = hexToTwo(MiWen);
        string K = hexToTwo(Key);
        string KKK = exchange(K, ExchangeRules, ExchangeRules_SIZE); /***处理密钥，生成 16 个子密
钥 **/
        int i = 0;/* 利用规则交换表（8*7）将 K 转换成 K0 ; K0(56 位) = C0(28 位) + D0(28 位) */
        string K_arr[Key_SIZE+1];/* 利用移位表转换得 C1D1----C16D16,存入 K_arr */
        K_arr[0] = KKK;
        for(i=1; i<=Key_SIZE; i++)
        {
            K_arr[i] = spiltShift(K_arr[i-1], ShiftTable[i-1]);
        }/* Kn（48 位）= PC-2（8*6）处理 CnDn 得 16 个子密钥，存入 Key_arr */
        string Key_arr[Key_SIZE];
        for(i=0; i<Key_SIZE; i++)
        {/*** 用子密钥对明文加密**/
            Key_arr[i] = exchange(K_arr[i+1], PC_2, PC_2_SIZE);
        }/* 通过 IP（8*8）处理 M 得 L0（32 位）  R0（32 位） */
        string IP_M = exchange(M, IP, IP_SIZE);
        string L[Key_SIZE+1];/* Ln= R(n-1); Rn= L(n-1) + f(R(n- 1); Kn)迭代 16 次 */
        string R[Key_SIZE+1];
        L[0] = IP_M.substr(0, M.length()/2);
        R[0] = IP_M.substr(M.length()/2);
        string it = "";
        for(i=1; i<=Key_SIZE; i++)
        {
            it = exchange(R[i-1], E, E_SIZE);//将 R0 通过扩展置换 E（8*6）从 32 位扩展到 48
位
            it = XOR(it, Key_arr[16-i]); //R0（48 位）与 K1 异或得 E0（48 位）
            it = SBoxWork(it, SBox);//将 E0（48 位）通过 S 盒转换成 32 位
            it = exchange(it, P, P_SIZE);//P 盒（8*4）置换，得 P0
```

```
            it = XOR(it, L[i-1]); //P0 与 L0 进行异或，得 J0
            L[i] = R[i-1];
            R[i] = it;//左右交换位置，即 R1 = J0; L1 = R0
        }  /* 对 R16 L16 进行一次 IP-1（8*8）排序得密文 */
        string res = "";
        res += R[16];
        res += L[16];
        string finalRes = Bin2Hex(exchange(res, IP_1, IP_1_SIZE));
        return finalRes;
    }
```

(5)数字签名与身份认证

```
if(strcasecmp(decry_username.c_str(),md5.get_md5(username_log_ori_string).substr(0,16).c_str()
)==0
            &&strcasecmp(decry_password.c_str(),md5.get_md5(password_log_ori_string).substr(0,
16).c_str())==0){
                yanzheng=true;
            }
            if(yanzheng){
                cout<<"Client authentication successful, verifying whether username and password
match..."<<endl;
                FILE *fp;
                char line[MAXDATASIZE];
                fp=fopen(filename,"r");
                if(fp==NULL){
                    printf("can not load file!");
                    return 1;
                }
                while(!feof(fp)){
                    fgets(line,1000,fp);
                    char username[MAXDATASIZE];
                    char password[MAXDATASIZE];
                    int index=0;
                    int jndex=0;
                    bool flag=true;
                    while(line[index]!='\0'&&line[index]!='\n'){
                        if(flag){
                            username[index]=line[index];
                            index++;
                        }else{
                            password[jndex]=line[index];
                            index++;
                            jndex++;
                        }
```

```
                    if(line[index]==','){
                        username[index]='\0';
                        flag=false;
                        index++;
                    }
                }
                password[jndex]='\0';
                if(strcmp(password,decry_password.c_str())==0&&strcmp(username,decry_usern
ame.c_str())==0){
                    auther=true;
                    break;
                }
            }
            fclose(fp);
        }else{
            cout<<"Client authentication failed!"<<endl;
        }
        if(auther){
            response_login="200";
            state=1;
        }else{
            response_login="404";
            state=2;
        }
        write(connectfd,response_login.c_str(),response_login.size());
```

客户端对输入的用户名和密码分别先后进行 MD5 加密、DES 加密得到数字签名的结果，再结合原本未处理的用户输入的用户名和密码一起发送给服务端。服务端对数字签名进行解密得到结果 A，并使用 MD5 加密得到的未处理的用户名和密码得到结果 B，将结果 A 和结果 B 进行比对。若 A 和 B 不相等，则用户的身份验证失败；否则，则身份验证成功。身份验证成功后，服务器会对用户的用户名和密码进行匹配检查，若档案库中存在用户信息，则登陆成功。除登陆成功，服务器向客户端发送反馈信号“200”外，其他情况均为登陆失败情况，发送反馈信号“404”。

(6)客户端获取文件路径并发送文件内容

```
cout<<"connect success!"<<endl;
cout<<"Please input file's directory: "<<endl;
cin>>dir;
FILE *fd= fopen(dir.c_str(),"r+");
while((num=fread(buf,1,MAXSIZE,fd))>0){
    send(sockfd,buf,num,0);
}
```

客户端获取用户输入的文件绝对路径，打开文件并发送文件内容。

(7)服务端获取备份文件路径并接收文件内容

```
cout<<"Please input directory of copy: "<<endl;
cin>>dir;
fd=fopen(dir.c_str(),"wb");
while(num=recv(connectfd,buf,1,0)>0)
{
    fwrite(buf,1,num,fd);
}
```

服务端建立根据备份文件路径建立备份文件，并将接收到的数据写入到备份文件里。

(8)客户端入口参数

```
/*get ip address*/
if((host= gethostbyname(argv[1])) == NULL){
    herror("gethostbyname() error!\n");
    return 0;
}
```

(9)反馈通信

客户端：

```
cout<<"Please input file's directory: "<<endl;
cin>>dir;
FILE *fd= fopen(dir.c_str(),"r+");
if(fd==NULL){
    cout<<"The refered file didn't exist!"<<endl;
    strcpy(res,"404");
    cout<<"response: "<<res<<endl;
}else{
    strcpy(res,"200");
    cout<<"response: "<<res<<endl;
}
write(sockfd,res,strlen(res));
```

判断文件路径是否合法。合法则发送“200”给服务端，否则发送“404”。

服务端：

```
if(strcmp("404",response)==0){
    cout<<"The server detected that the file specified by the client does not exist! File backup
    return 0;
}else{
    cout<<"Please input directory of copy: "<<endl;
    cin>>dir;
    fd=fopen(dir.c_str(),"wb");
    while(num=recv(connectfd,buf,1,0)>0)
    {
        fwrite(buf,1,num,fd);
    }
    fclose(fd);
    close(connectfd);
    cout<<"The file was successfully received by the server and the backup was successful!"<<end
    cout<<"please check in about 10 seconds, or later..."<<endl;
}
```

服务端接收到“200”时才会开始文件备份功能。否则直接退出。

四、调试记录

1. 启动服务器

```
[neu_ljh@localhost c]$ g++ -o fileserverplus fileserverplus.cpp
[neu_ljh@localhost c]$ ./fileserverplus
Current Time: Tue Jun 28 10:49:56 2022

Waiting for the client to connect to this server...
```

2.启动客户端

```
[neu_ljh@localhost c]$ g++ -o fileclientplus fileclientplus.cpp
[neu_ljh@localhost c]$ ./fileclientplus 127.0.0.1
Current Time: Tue Jun 28 10:50:36 2022

The client is connecting to the server...
connect success!
```

等待系统初始化，完成各项密钥的存储管理，发现客户端打印出相关数据，代表系统初始化完成：

```
[neu_ljh@localhost c]$ g++ -o fileclientplus fileclientplus.cpp
[neu_ljh@localhost c]$ ./fileclientplus 127.0.0.1
Current Time: Tue Jun 28 10:50:36 2022

The client is connecting to the server...
connect success!
Accept RSA: public_key successfully!
RSA public_key: 739,1003
e: 739
n: 1003
DES key: B542D99767E64613
Send DES: key successfully!
The system detected that this host is not logged in!
If you do not have account,you cannot use any function!
Press 0 to register,or press 1 to login:
```

服务端也打印出相关数据：

```
[neu_ljh@localhost c]$ g++ -o fileserverplus fileserverplus.cpp
[neu_ljh@localhost c]$ ./fileserverplus
Current Time: Tue Jun 28 10:49:56 2022

Waiting for the client to connect to this server...
It's Connected!
Send RSA: public_key successfully!
public key: 739,1003
private key: 491,1003
Accept DES: key successfully!
Receive DES key from Client(encrypted state): 264892562747374454454302143086821456221496168
After decode: DES key= B542D99767E64613
```

4.注册功能(含身份管理)

在客户端输入相关信息。

客户端：

```
The client is connecting to the server...
connect success!
Accept RSA: public_key successfully!
RSA public_key: 739,1003
e: 739
n: 1003
DES key: B542D99767E64613
Send DES: key successfully!
The system detected that this host is not logged in!
If you do not have account,you cannot use any function!
Press 0 to register,or press 1 to login:
0
*************** Register Interface ******************
please input the username: neu

please input the password: neu

register account successfully!
Congratulations, you have completed the registration, restart the client to u
se the file backup function!
```

服务端：

```
[neu_ljh@localhost c]$ g++ -o fileserverplus fileserverplus.cpp
[neu_ljh@localhost c]$ ./fileserverplus
Current Time: Tue Jun 28 10:49:56 2022

Waiting for the client to connect to this server...
It's Connected!
Send RSA: public_key successfully!
public key: 739,1003
private key: 491,1003
Accept DES: key successfully!
Receive DES key from Client(encrypted state): 264892562747374454454302143086821456221496168
After decode: DES key= B542D99767E64613
The user has chosen to register, so the file backup function cannot continue to be used...
```

当注册完之后，服务端写入对应信息，然后程序结束。


```
123,123
456,456
123456,123456
202cb962ac59075b964b07152d234b70,202cb962ac59075b964b07152d234b70
4B7F98C072961DDC,4B7F98C072961DDC
67F87C9B32834504,67F87C9B32834504
83E71D83B2DCEA23,83E71D83B2DCEA23
```

最后一条为客户端注册而新加入的用户信息。

5.登录功能(含身份验证)

重新运行程序，客户端输入 1 选择登录功能。

```
[neu_ljh@localhost c]$ ./fileclientplus 127.0.0.1
Current Time: Tue Jun 28 11:03:19 2022

The client is connecting to the server...
connect success!
Accept RSA: public_key successfully!
RSA public_key: 139,1147
e: 139
n: 1147
DES key: BB359D7BAE5CD6DC
Send DES: key successfully!
The system detected that this host is not logged in!
If you do not have account,you cannot use any function!
Press 0 to register,or press 1 to login:
1
```

输入刚才的注册信息：

客户端：

```
Press 0 to register,or press 1 to login:
1
please log in first...
*************** Login Interface ******************
please input the username: neu
username_log: neu

please input the password: neu
password_log: neu

encry_username: 24FE7454DBF5560B
encry_passoword: 24FE7454DBF5560B
md5_username: 83e71d83b2dcea23dcab3b64182494cc
md5_password: 83e71d83b2dcea23dcab3b64182494cc
decry_username: 83E71D83B2DCEA23
decry_password: 83E71D83B2DCEA23
waiting for the Certification from server......
login successfully!
Please input file's directory:
```

服务端：

```
username_log_ori: neu
password_log_ori: neu
username_log: FE56B9AA1A87E1E2
password_log: FE56B9AA1A87E1E2
decry_username: 83E71D83B2DCEA23
decry_password: 83E71D83B2DCEA23
md5_username: 83e71d83b2dcea23
md5_password: 83e71d83b2dcea23
Client authentication successful, verifying whether username and password mat
ch...
username and password match!state: 1
```

身份核验通过！

6.文件备份功能

选择备份文件 timerecords,提前查看文件内容：

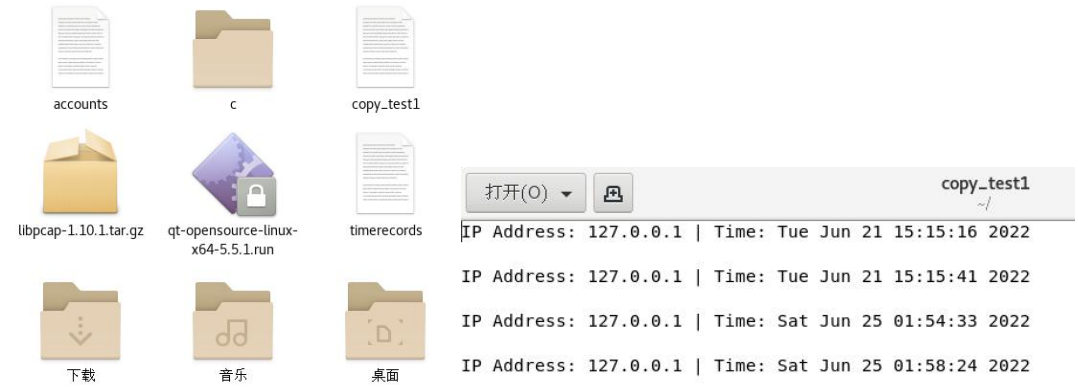
客户端：

```
Please input file's directory:
/home/neu_ljh/timerecords
response: 200
The file specified by the user has been sent to the server!
please check in about 10 seconds, or later...
[neu_ljh@localhost c]$
```

服务端：

```
response: 200
Please input directory of copy:
/home/neu_ljh/copy_test1
The file was successfully received by the server and the backup was successfu
l!
please check in about 10 seconds, or later...
[neu_ljh@localhost c]$
```

查看文件备份情况：



文件创建成功并备份成功。

五、创新说明

1. 增加服务端时间记录

```
[neu_ljh@localhost c]$ ./fileserver
Current Time: Sat Jun 25 13:27:53 2022

Waiting for the client to connect to this server...
It's Connected!
```

2.新增对于客户端文件路径不存在的及时反馈

例如我们在客户端输入一个不存在的文件：/home/neu_ljh/nosuchfile

```
login successfully!
Please input file's directory:
/home/neu_ljh/nosuchfile
The refered file didn't exist!
response: 404
The refered file didn't exist.Program terminated!
[neu_ljh@localhost c]$
```

客户端会提示文件不存在，并且将消息“404”发送给服务端(消息“200”表示路径存在)。然后服务端根据客户端的消息是“200”还是“400”来判断下一步的逻辑操作。当服务端接收到“404”信号时：

```
Client authentication successful, verifying whether username and password mat
ch...
username and password match!state: 1
response: 404
The server detected that the file specified by the client does not exist! Fil
e backup failed!
[neu_ljh@localhost c]$
```

服务端接收到“404”，提示文件不存在，终止程序。

3.数字签名、身份管理和身份认证，信息的加密传输

客户端：

```
encry_username: 96B4C9F97C377B1E
encry_passoword: 96B4C9F97C377B1E
md5_username: 83e71d83b2dcea23dcab3b64182494cc
md5_password: 83e71d83b2dcea23dcab3b64182494cc
decry_username: 83E71D83B2DCEA23
decry_password: 83E71D83B2DCEA23
```

服务端：

```
Send RSA: public_key successfully!
public key: 487,1271
private key: 823,1271
Accept DES: key successfully!
Receive DES key from Client(encrypted state): 335841101105651615841141110565110445445145412
After decode: DES key= 779AA2D98A2A4451
username_log_ori: neu
password_log_ori: neu
username_log: 96B4C9F97C377B1E
password_log: 96B4C9F97C377B1E
decry_username: 83E71D83B2DCEA23
decry_password: 83E71D83B2DCEA23
md5_username: 83e71d83b2dcea23
md5_password: 83e71d83b2dcea23
```

实验 4　基于 Libpcap 的网络数据包捕获与嗅探程序

一、程序实践概述

1、题目名称：

Libpcap 开发包使用

2、时间进度：

2022.06.20 - 2022.06.28

3、开发环境：

Linux Ubuntu 18.04.6，Linux CentOS release 7.0(Final)

Visual studio code C/C++ May 2022 (version 1.68)

QT.creator 5.9.8

二、问题分析

1、功能说明：

（1）获取网络接口名字和掩码等信息

（2）捕获数据包（单个数据包和多个数据包两种情况）

（3）以太网数据报捕获

（4）ARP 数据包捕获

（5）IP 数据包捕获

（6）TCP 数据包捕获

（7）UDP 数据包捕获

（8）ICMP 数据包捕获

（9）总数据包的捕获

2、解决方案：

(1)网络设备查找

(2)打开网络设备

(3)获取网络参数

(4)编译过滤策略

(5)设置过滤器

(6)利用回调函数捕获数据包

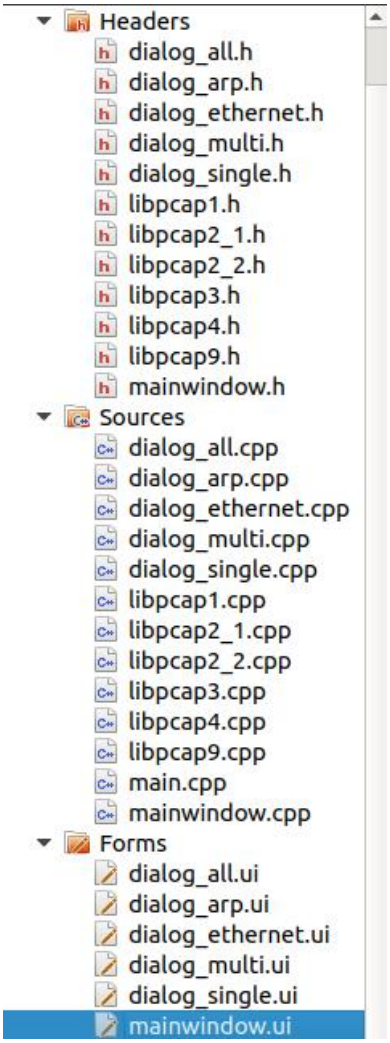
(7)数据可视化呈现

(8)关闭网络设备

本实验采用了 QT 编程，具有合理的操作界面和操作逻辑。

三、方案设计

1、模块结构：



主页面：**mainwindow.cpp**

函数	说明
void MainWindow::on_get_base_infor_clicked()	点击获取主机的基本信息
void MainWindow::on_btn_single_clicked()	点击跳转页面，到达获取单个数据包页面
void MainWindow::on_btn_multiple_clicked()	点击跳转页面，到达获取多个数据包页面
void MainWindow::on_btn_ethernet_clicked()	点击跳转页面，到达获取以太网数据包页面
void MainWindow::on_btn_arp_clicked()	点击跳转页面，到达获取 ARP 数据包页面
void MainWindow::on_btn_all_clicked()	点击跳转页面，到达获取全部数据包页面

获取单个数据包页面：dialog_single.cpp

函数	说明
void Dialog_single::on_btn_single_start_clicked()	点击获取单个数据包

获取多个数据包页面：dialog_multi.cpp

函数	说明
void Dialog_multi::on_btn_mul_start_clicked()	点击获取多个数据包

获取以太网数据包页面：dialog_ethernet.cpp

函数	说明
void Dialog_ethernet::on_btn_start_clicked()	点击获取以太网数据包

获取 ARP 数据包页面：dialog_arp.cpp

函数	说明
void Dialog_arp::on_btn_start_clicked()	点击获取第一个 ARP 数据包
void Dialog_arp::on_btn_next_clicked()	点击获取下一个 ARP 数据包

获取所有数据包页面(包含以太网，ARP,IP,TCP,UDP,ICMP)：dialog_all.cpp

函数	说明
void Dialog_all::on_pushButton_clicked()	点击获取第一个详细数据包
void Dialog_all::on_btn_next_clicked()	点击获取下一个详细数据包

2、数据结构：

获取网络接口名字和掩码等信息：libpcap1.h

```
class libpcap1
{
public:
    libpcap1() {}
    static char* net_interface;
    static char* net_ip_string;
    static char* net_mask_string;
};
```

获取单个数据包：libpcap2_1.h

```
class libpcap2_1{
public:
    static string net_interface;
    static int length;
};
```

获取多个数据包：libpcap2_2.h

```
class libpcap2_2
{
public:
    libpcap2_2() {}
    static string res;
    static int number;
};
```

获取以太网数据包：libpcap3.h

```
class libpcap3
{
public:
    libpcap3() {}
    static string net_interface;
    static string time;
    static int length;
    static u_short ethernet_type;
    static string protocol;
    static string mac_source;
    static string mac_des;
};
```

获取 ARP 数据包：libpcap4.h

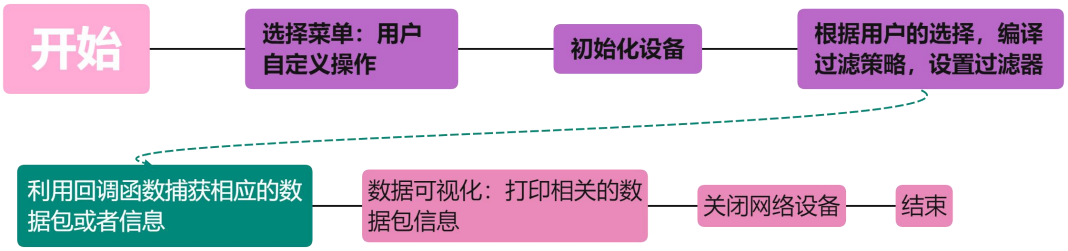
```
class libpcap4{
public:
    static int number;
    static vector<string> ethernet_type;
    static vector<string> mac_source;
    static vector<string> mac_des;
    static vector<unsigned short> hardware_type;
    static vector<unsigned short> protocol_type;
    static vector<unsigned char> hardware_length;
    static vector<unsigned char> protocol_length;
    static vector<unsigned short> operation_code;
    static vector<string> protocol;
    static vector<string> eth_source;
    static vector<string> ip_source;
    static vector<string> eth_des;
    static vector<string> ip_des;
};
```

获取所有数据包：libpcap9.h

```
class libpcap9{
public:
    static int number;
    //ethernet
    static vector<string> ethernet_type;
    static vector<string> mac_source;
    static vector<string> mac_des;
     //ethernet
     //arp
    static vector<unsigned short> hardware_type;
    static vector<unsigned short> protocol_type;
    static vector<unsigned char> hardware_length;
    static vector<unsigned char> protocol_length;
    static vector<unsigned short> operation_code;
    static vector<string> protocol;
    static vector<string> eth_source;
    static vector<string> ip_source;
    static vector<string> eth_des;
    static vector<string> ip_des;
    //arp
    //ip
    static vector<unsigned char> ip_version;
    static vector<int> ip_header_length;
    static vector<unsigned char> ip_tos;
    static vector<unsigned short> ip_total_length;
    static vector<unsigned short> ip_id;
    static vector<unsigned int> ip_offset;
    static vector<int> ip_ttl;
    static vector<unsigned char> ip_protocol;
    static vector<int> ip_checksum;
    static vector<string> ip_source_address;
    static vector<string> ip_des_adddress;
    //ip
```

```
    //tcp
    static vector<int> tcp_source_port;
    static vector<int> tcp_des_port;
    static vector<string> tcp_protocol;
    static vector<unsigned int> tcp_seq_num;
    static vector<unsigned int> tcp_ack_num;
    static vector<int> tcp_header_length;
    static vector<int> tcp_reserved;
    static vector<string> tcp_flags;
    static vector<int> tcp_win_size;
    static vector<int> tcp_checksum;
    static vector<int> tcp_u_pointer;
    //tcp
    //udp
    static vector<int> udp_source_port;
    static vector<int> udp_des_port;
    static vector<string> udp_service;
    static vector<int> udp_length;
    static vector<int> udp_checksum;
    //udp
    //icmp
    static vector<int> icmp_type;
    static vector<string> icmp_protocol;
    static vector<int> icmp_code;
    static vector<int> icmp_id;
    static vector<int> icmp_seq_num;
    static vector<int> icmp_checksum;
    //icmp
```

3、总体流程：



4、关键算法：

(1)获取主机信息和 IP 地址

```
void get_ip_address(){
    char* interface;
    char* ip;

    char error_content[PCAP_ERRBUF_SIZE];//1
    struct in_addr net_ip_address;//4

    u_int32_t net_ip;//3
    u_int32_t net_mask;//3
    libpcap1::net_interface=pcap_lookupdev(error_content);
    pcap_lookupnet(libpcap1::net_interface,&net_ip,&net_mask,error_content);
    interface=libpcap1::net_interface;

    net_ip_address.s_addr=net_ip;
    libpcap1::net_ip_string=inet_ntoa(net_ip_address);
    ip=libpcap1::net_ip_string;
}
```

(2)获取子网掩码

```
void get_mask_address(){
    char* interface;
    char* mask;
    char error_content[PCAP_ERRBUF_SIZE];//1

    struct in_addr net_mask_address;
    u_int32_t net_ip;//3
    u_int32_t net_mask;//3
    libpcap1::net_interface=pcap_lookupdev(error_content);
    pcap_lookupnet(libpcap1::net_interface,&net_ip,&net_mask,error_content);
    interface=libpcap1::net_interface;

    net_mask_address.s_addr=net_mask;
    libpcap1::net_mask_string=inet_ntoa(net_mask_address);
    mask=libpcap1::net_mask_string;
}
```

(3)获取数据包的主体函数

```
void solution()
{
    char error_content[PCAP_ERRBUF_SIZE];
    pcap_t* pcap_handle;
    char* net_interface;
    struct bpf_program bpf_filter;
    char bpf_filter_string[] = "";
    bpf_u_int32 net_mask;
    bpf_u_int32 net_ip;
    net_interface = pcap_lookupdev(error_content);

    pcap_lookupnet(net_interface, &net_ip, &net_mask, error_content);

    pcap_handle = pcap_open_live(net_interface, BUFSIZ, 1, 0, error_content);

    pcap_compile(pcap_handle, &bpf_filter, bpf_filter_string, 0, net_ip);

    pcap_setfilter(pcap_handle, &bpf_filter);

    if (pcap_datalink(pcap_handle) != DLT_EN10MB) {
        return;
    }

    pcap_loop(pcap_handle, libpcap9::number, ethernet_protocol_packet_callback, NULL);

    pcap_close(pcap_handle);
}
```

调用回调函数 ethernet_protocol_packet_callback，处理、分析和获取以太网协议相关的信息；如果需要更深层次的协议信息，则在 ethernet_protocol_packet_callback 中，调用回调函数 arp_protocol_packet_callback 来获取 ARP 协议的相关信息或者 ip_protocol_packet_callback 来获取 IP 协议的相关信息；如果调用的是 IP 协议，则可以继续调用回调函数 tcp_protocol_packet_callback 来获取 TCP 协议的相关信息，或者调用回调函数 udp_protocol_packet_callback 来获取 UDP 协议的相关信息,或者调用回调函数 icmp_protocol_packet_callback 来获取 ICMP 协议的相关信息。

(4)回调函数

这里拿一个简单的回调函数举例子：

```
static void ip_protocol_packet_callback(u_char* argument, const struct pcap_pkthdr* packet_header, const u_char* packet_content) {
        struct ip_header* ip_protocol;
        u_int header_length;
        u_int offset;
        u_char tos;
        u_int16_t checksum;
        ip_protocol = (struct ip_header*)(packet_content + 14);
        checksum = ntohs(ip_protocol->ip_checksum);
        header_length = ip_protocol->ip_header_length * 4;
        tos = ip_protocol->ip_tos;
        offset = ntohs(ip_protocol->ip_off);
        printf("---------- IP Protocol (Network Layer) ----------");
        printf("IP version: %d\n", ip_protocol->ip_version);
        printf("Header length: %d\n", header_length);
        printf("TOS: %d\n", tos);
```

```
printf("Total length: %d\n", ntohs(ip_protocol->ip_length));
printf("Identification: %d\n", ntohs(ip_protocol->ip_id));
printf("Offset: %d\n", (offset & 0x1fff) * 8);
printf("TTL: %d\n", ip_protocol->ip_ttl);
printf("Protocol: %d\n", ip_protocol->ip_protocol);
switch (ip_protocol->ip_protocol) {
case 6:
	printf("The Transport Layer Protocol is TCP\n");
	break;
case 17:
	printf("The Transport Layer Protocol is UDP\n");
	break;
case 1:
	printf("The Transport Layer Protocol is ICMP\n");
	break;
default:
	break;
}
printf("Header checksum: %d\n", checksum);
printf("Source address: %s\n", inet_ntoa(ip_protocol->ip_source_address));
printf("Destination address: %s\n", inet_ntoa(ip_protocol->ip_destination_address));
switch (ip_protocol->ip_protocol) {
case 6:
	tcp_protocol_packet_callback(argument,packet_header,packet_content);
	break;
case 17:
	udp_protocol_packet_callback(argument,packet_header,packet_content);
	break;
case 1:
	icmp_protocol_packet_callback(argument,packet_header,packet_content);
	break;
default:
	break;
}
libpcap9::ip_version.push_back(ip_protocol->ip_version);
libpcap9::ip_header_length.push_back(header_length);
libpcap9::ip_tos.push_back(tos);
libpcap9::ip_total_length.push_back(ntohs(ip_protocol->ip_length));
libpcap9::ip_id.push_back(ntohs(ip_protocol->ip_id));
libpcap9::ip_offset.push_back((offset & 0x1fff) * 8);
libpcap9::ip_ttl.push_back(ip_protocol->ip_ttl);
libpcap9::ip_protocol.push_back(ip_protocol->ip_protocol);
libpcap9::ip_checksum.push_back(checksum);
libpcap9::ip_source_address.push_back(inet_ntoa(ip_protocol->ip_source_address));
```

libpcap9::ip_des_adddress.push_back(inet_ntoa(ip_protocol->ip_destination_address));
}

(5)信息可视化逻辑

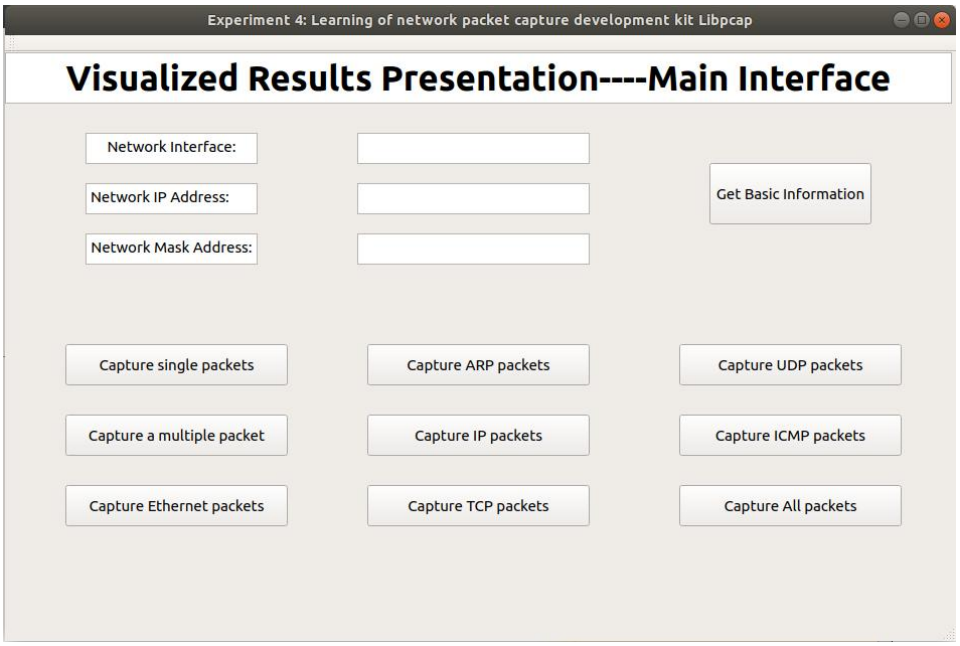
```
void Dialog_arp::on_btn_start_clicked()
{
    QString number_string=ui->te_number->toPlainText();
    int number;
    if(number_string==""){
        number=1;
    }else{
        number=number_string.toInt();
    }
    libpcap4::number=number;

    libpcap4_cpp pcap4;
    pcap4.solution();

    ui->te_id->setText(libpcap4::ethernet_type[Dialog_arp::index].c_str());
    ui->te_mac_source->setText(libpcap4::mac_source[Dialog_arp::index].c_str());
    ui->te_mac_des->setText(libpcap4::mac_des[Dialog_arp::index].c_str());
    ui->te_ht->setText(to_string(libpcap4::hardware_type[Dialog_arp::index]).c_str());
    ui->te_pt->setText(to_string(libpcap4::protocol_type[Dialog_arp::index]).c_str());
    ui->te_hl->setText(to_string(libpcap4::hardware_length[Dialog_arp::index]).c_str());
    ui->te_pl->setText(to_string(libpcap4::protocol_length[Dialog_arp::index]).c_str());
    ui->te_operation->setText(to_string(libpcap4::operation_code[Dialog_arp::index]).c_st
    ui->te_protocol->setText(libpcap4::protocol[Dialog_arp::index].c_str());
    ui->te_eth_source->setText(libpcap4::eth_source[Dialog_arp::index].c_str());
    ui->te_ip_source->setText(libpcap4::ip_source[Dialog_arp::index].c_str());
    ui->te_eth_des->setText(libpcap4::eth_des[Dialog_arp::index].c_str());
    ui->te_ip_des->setText(libpcap4::ip_des[Dialog_arp::index].c_str());

    string str="The "+to_string(Dialog_arp::index)+" ARP packet is captured";
    ui->order->setText(str.c_str());
}
```

(6)获取多个数据包时的 NEXT 操作

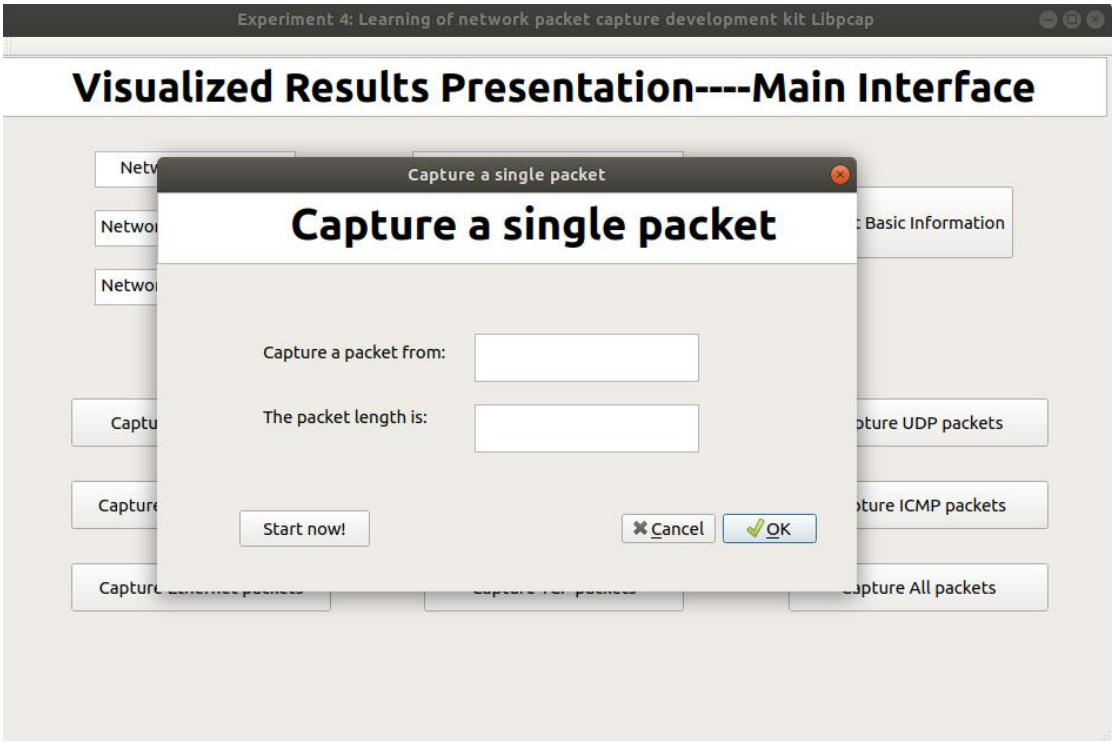
```
void Dialog_arp::on_btn_next_clicked()
{
    Dialog_arp::index++;
    if(Dialog_arp::index==libpcap4::number){
        Dialog_arp::index=0;
    }

    string str="The "+to_string(Dialog_arp::index+1)+" ARP packet is captured";
    ui->order->setText(str.c_str());

    ui->te_id->setText(libpcap4::ethernet_type[Dialog_arp::index].c_str());
    ui->te_mac_source->setText(libpcap4::mac_source[Dialog_arp::index].c_str());
    ui->te_mac_des->setText(libpcap4::mac_des[Dialog_arp::index].c_str());
    ui->te_ht->setText(to_string(libpcap4::hardware_type[Dialog_arp::index]).c_str());
    ui->te_pt->setText(to_string(libpcap4::protocol_type[Dialog_arp::index]).c_str());
    ui->te_hl->setText(to_string(libpcap4::hardware_length[Dialog_arp::index]).c_str());
    ui->te_pl->setText(to_string(libpcap4::protocol_length[Dialog_arp::index]).c_str());
    ui->te_operation->setText(to_string(libpcap4::operation_code[Dialog_arp::index]).c_st
    ui->te_protocol->setText(libpcap4::protocol[Dialog_arp::index].c_str());
    ui->te_eth_source->setText(libpcap4::eth_source[Dialog_arp::index].c_str());
    ui->te_ip_source->setText(libpcap4::ip_source[Dialog_arp::index].c_str());
    ui->te_eth_des->setText(libpcap4::eth_des[Dialog_arp::index].c_str());
    ui->te_ip_des->setText(libpcap4::ip_des[Dialog_arp::index].c_str());

}
```

5、界面设计：

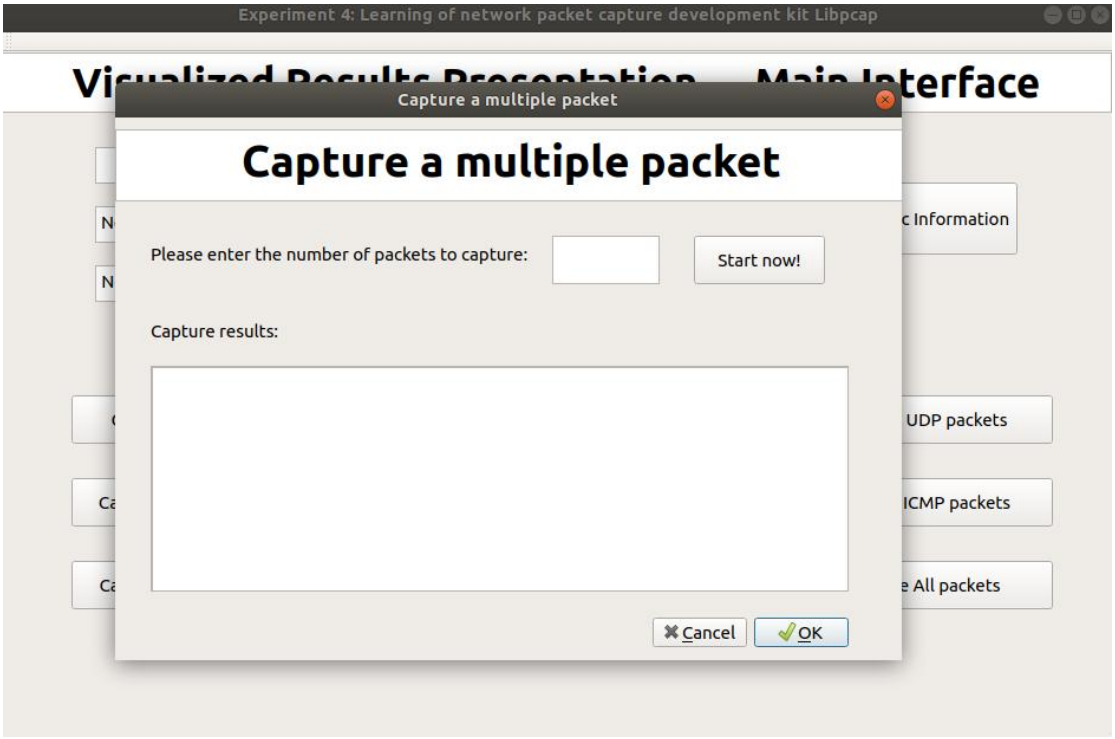
主页面：



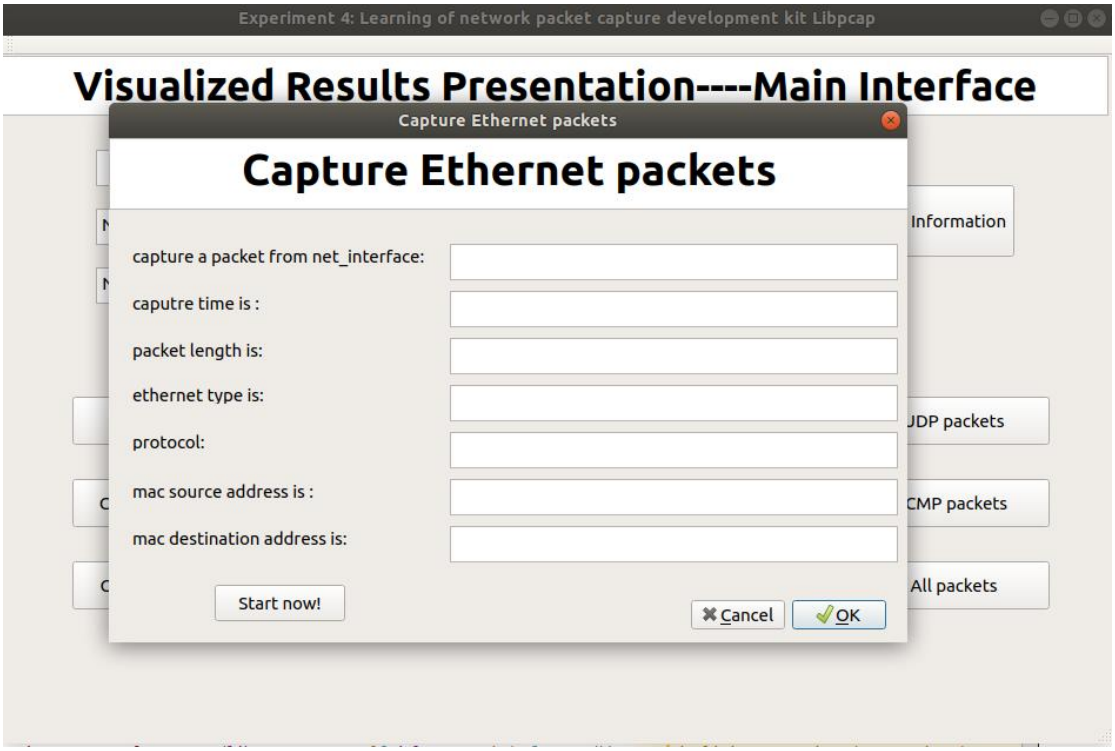
获取单个数据包页面(均采用弹窗实现)：



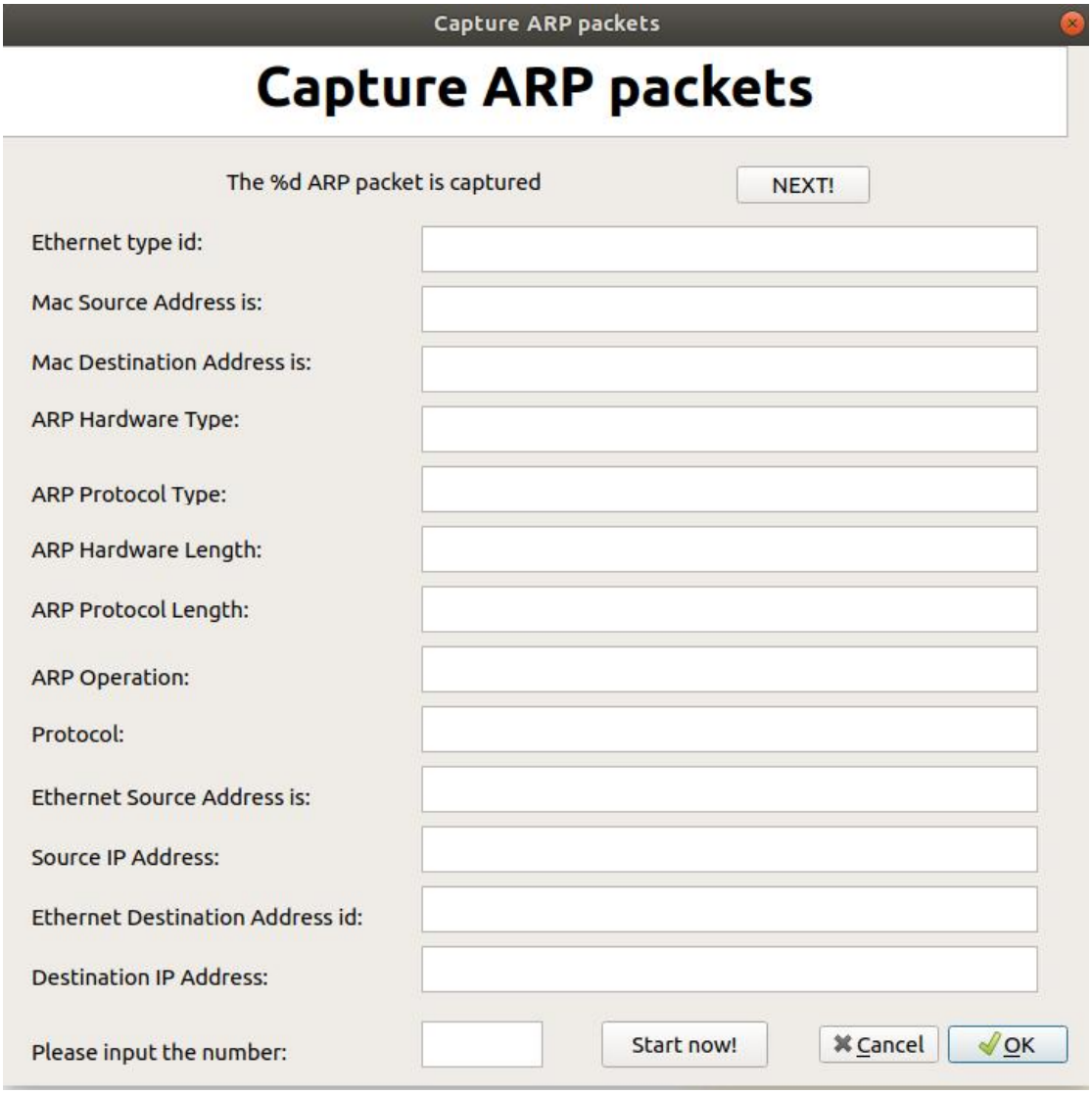
获取多个数据包页面：



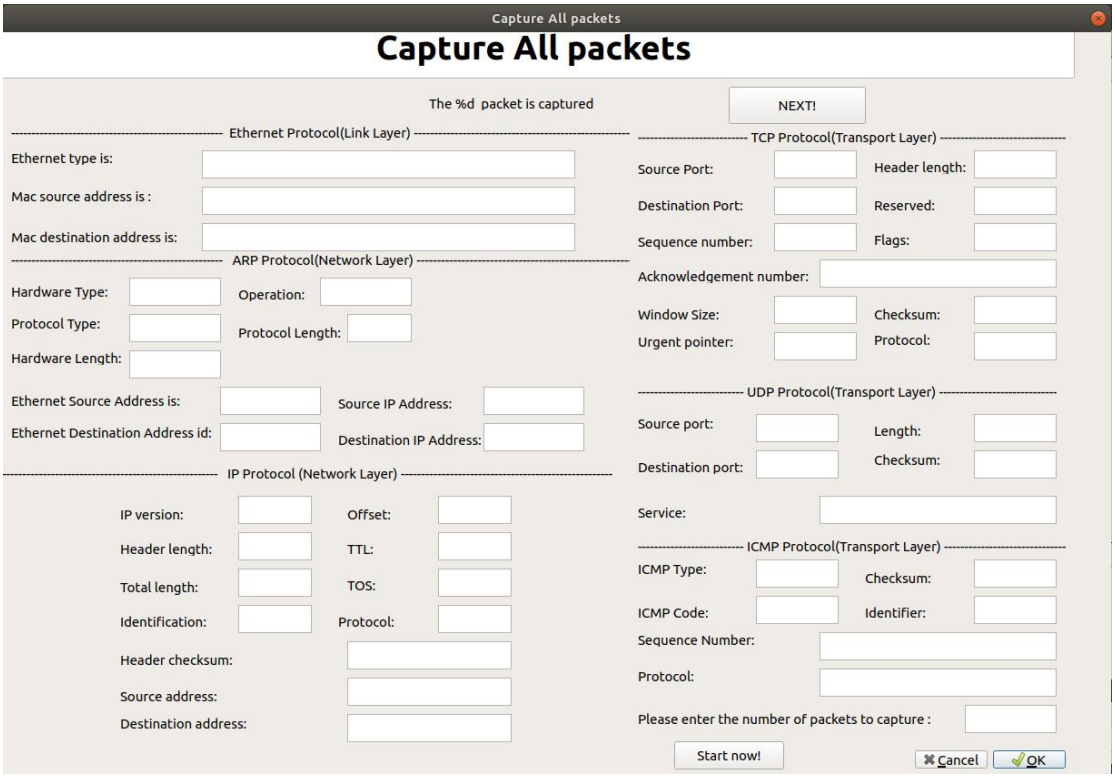
获取以太网数据包界面：



获取 ARP 数据包界面：

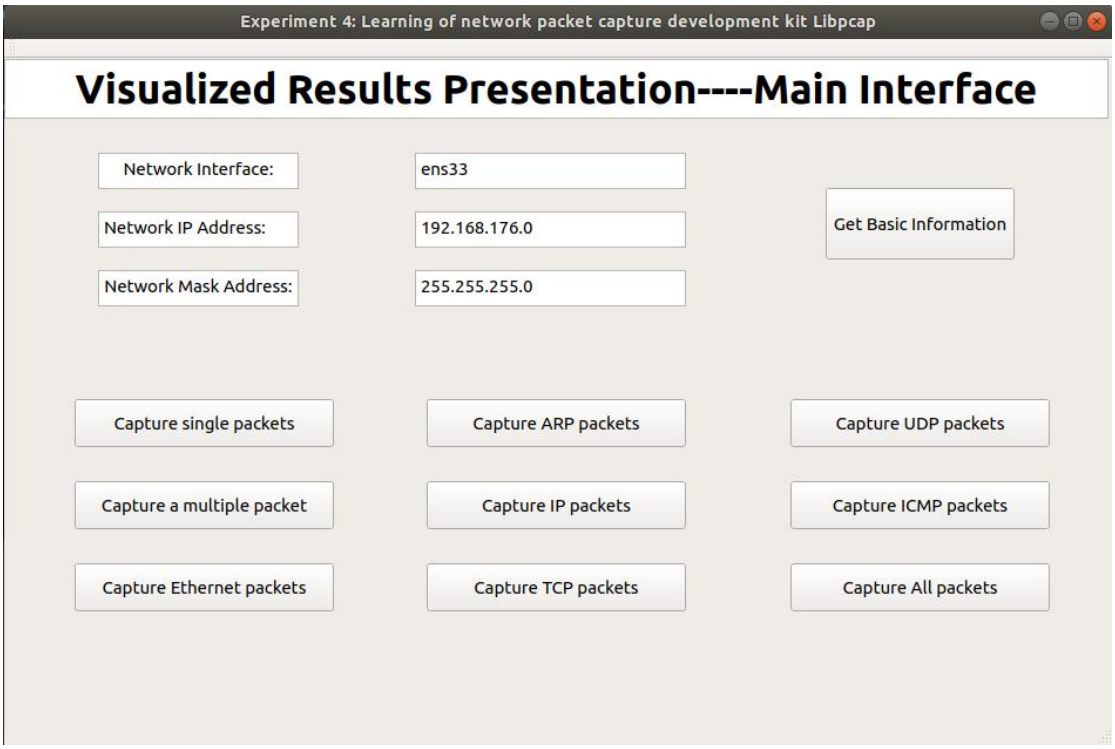

Capture ARP packets
Capture ARP packets
The %d ARP packet is captured
NEXT!
Ethernet type id:
Mac Source Address is:
Mac Destination Address is:
ARP Hardware Type:
ARP Protocol Type:
ARP Hardware Length:
ARP Protocol Length:
ARP Operation:
Protocol:
Ethernet Source Address is:
Source IP Address:
Ethernet Destination Address id:
Destination IP Address:
Please input the number:
Start now!
Cancel
OK

总数据包的获取页面：

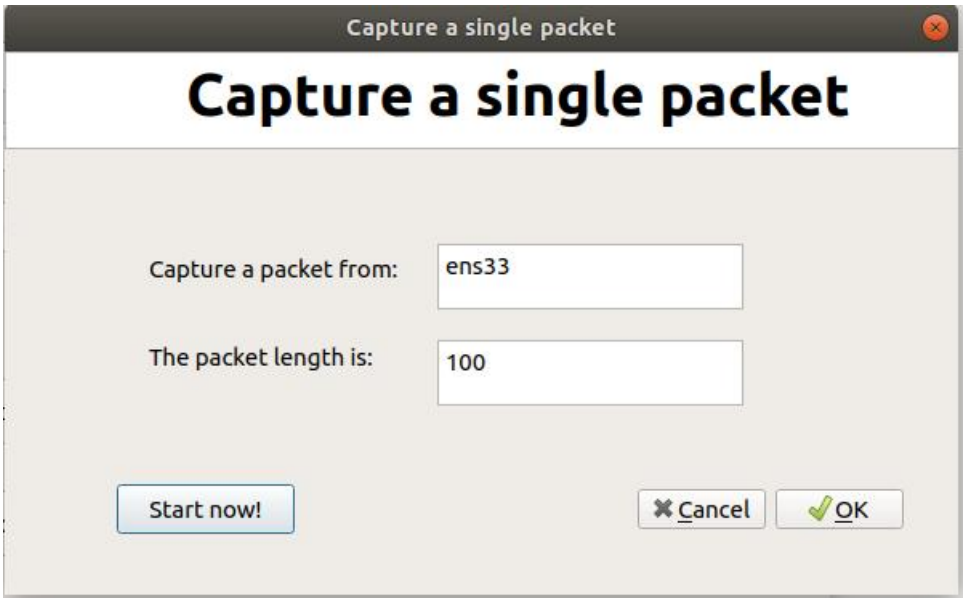


四、调试记录

（1）获取网络接口名字和掩码等信息

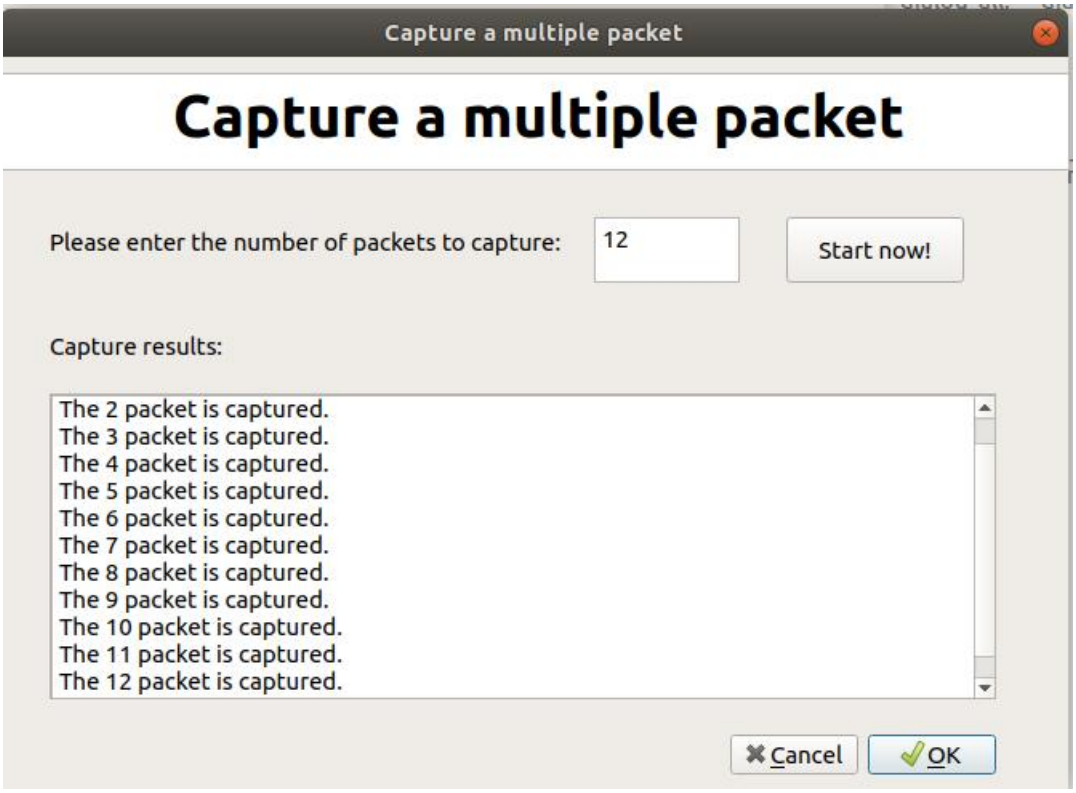


（2）捕获单个数据包

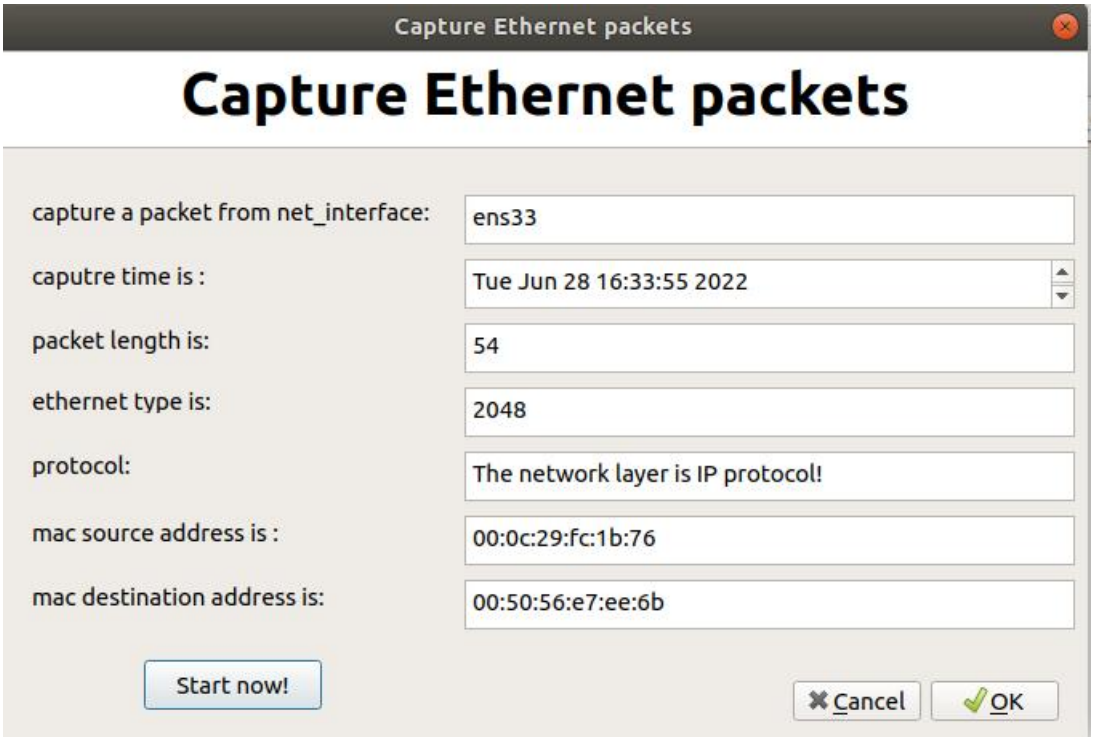


（3）捕获多个数据包

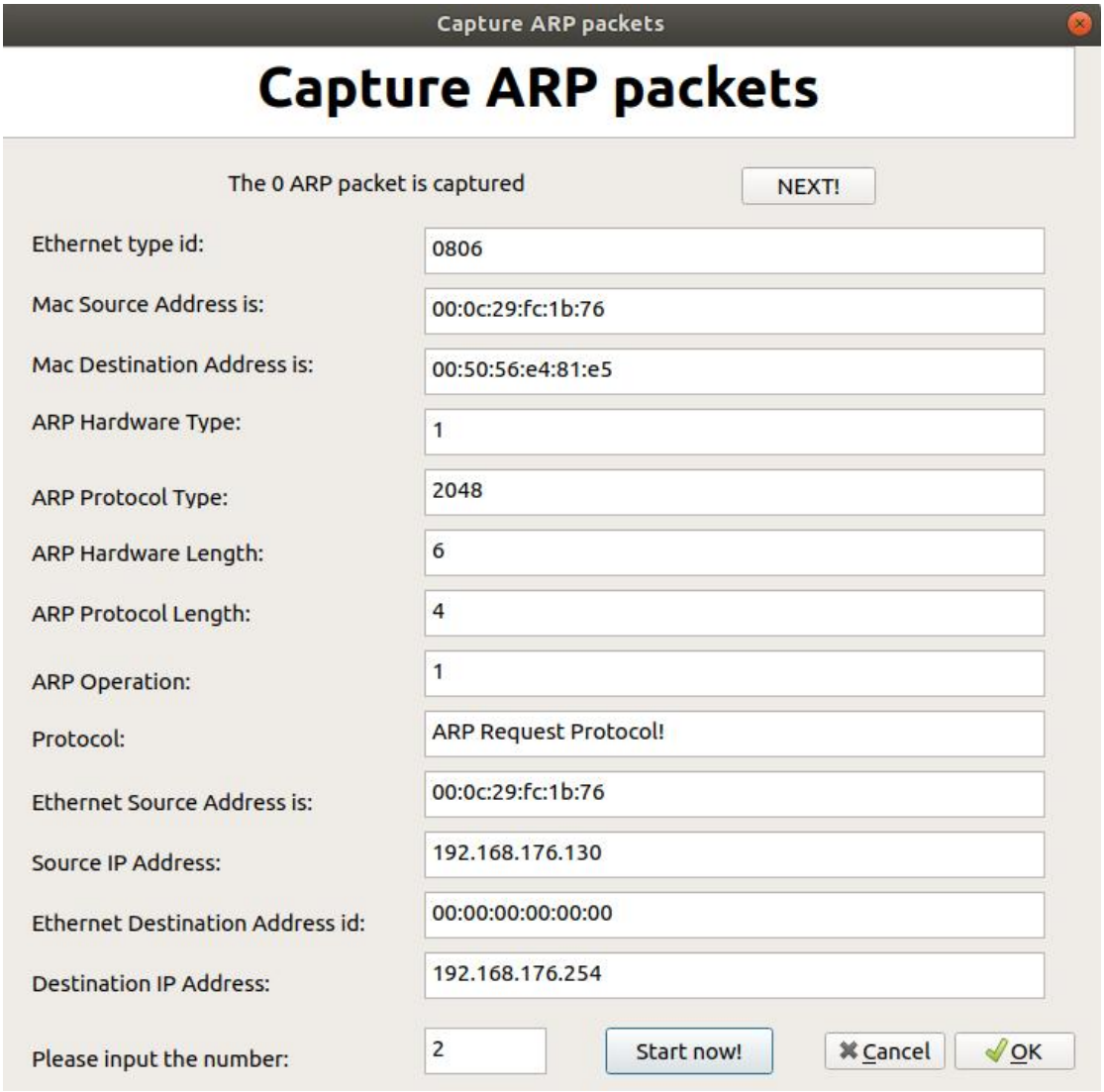
支持用户自定义数据包的个数



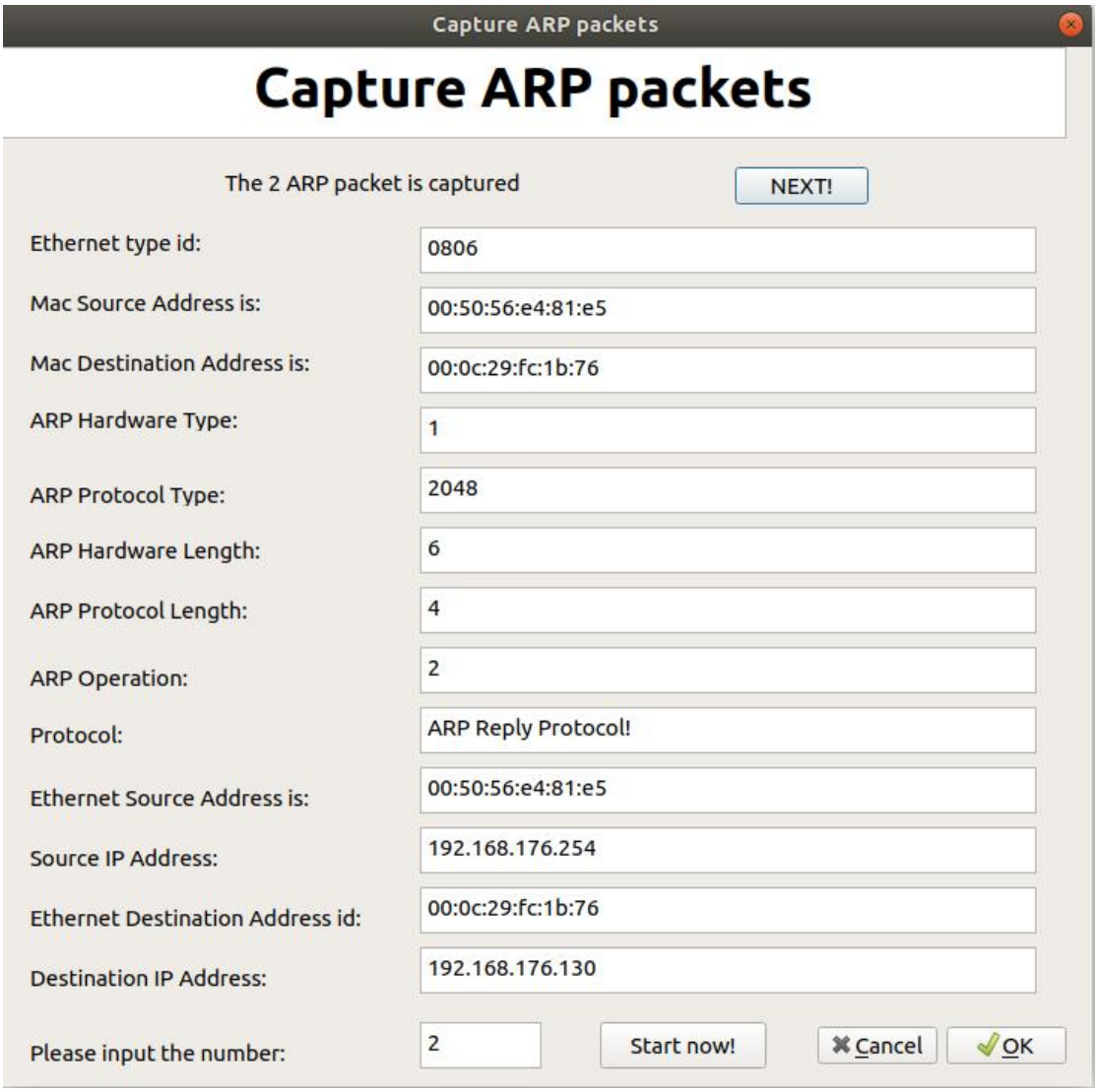
（4）以太网数据报捕获


Capture Ethernet packets
Capture Ethernet packets
capture a packet from net_interface:
ens33
caputre time is :
Tue Jun 28 16:33:55 2022
packet length is:
54
ethernet type is:
2048
protocol:
The network layer is IP protocol!
mac source address is :
00:0c:29:fc:1b:76
mac destination address is:
00:50:56:e7:ee:6b
Start now!
Cancel
OK

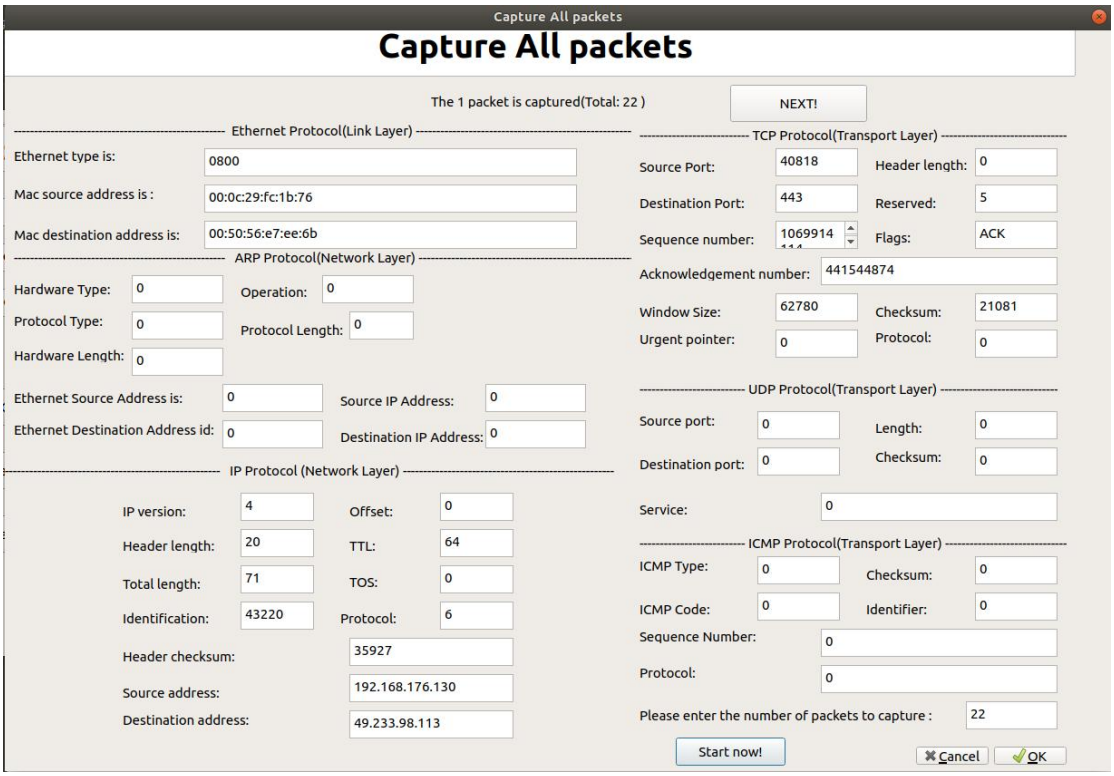
（5）ARP 数据包捕获


Capture ARP packets
Capture ARP packets
The 0 ARP packet is captured
NEXT!
Ethernet type id:
0806
Mac Source Address is:
00:0c:29:fc:1b:76
Mac Destination Address is:
00:50:56:e4:81:e5
ARP Hardware Type:
1
ARP Protocol Type:
2048
ARP Hardware Length:
6
ARP Protocol Length:
4
ARP Operation:
1
Protocol:
ARP Request Protocol!
Ethernet Source Address is:
00:0c:29:fc:1b:76
Source IP Address:
192.168.176.130
Ethernet Destination Address id:
00:00:00:00:00:00
Destination IP Address:
192.168.176.254
Please input the number:
2
Start now!
Cancel
OK

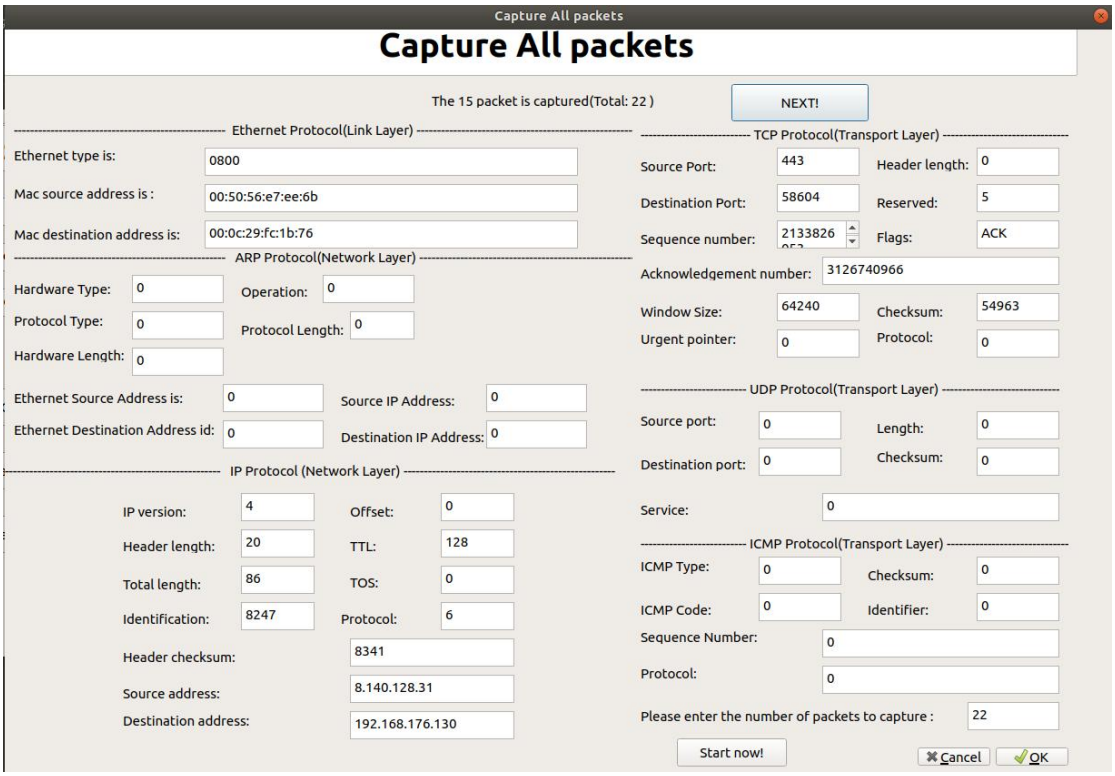
点击 NEXT：


Capture ARP packets
Capture ARP packets
The 2 ARP packet is captured
NEXT!
Ethernet type id:
0806
Mac Source Address is:
00:50:56:e4:81:e5
Mac Destination Address is:
00:0c:29:fc:1b:76
ARP Hardware Type:
1
ARP Protocol Type:
2048
ARP Hardware Length:
6
ARP Protocol Length:
4
ARP Operation:
2
Protocol:
ARP Reply Protocol!
Ethernet Source Address is:
00:50:56:e4:81:e5
Source IP Address:
192.168.176.254
Ethernet Destination Address id:
00:0c:29:fc:1b:76
Destination IP Address:
192.168.176.130
Please input the number:
2
Start now!
Cancel
OK

（6）总数据包的捕获(此处包括 IP 数据包捕获，TCP 数据包捕获，UDP 数据包捕获和 ICMP 数据包捕获)

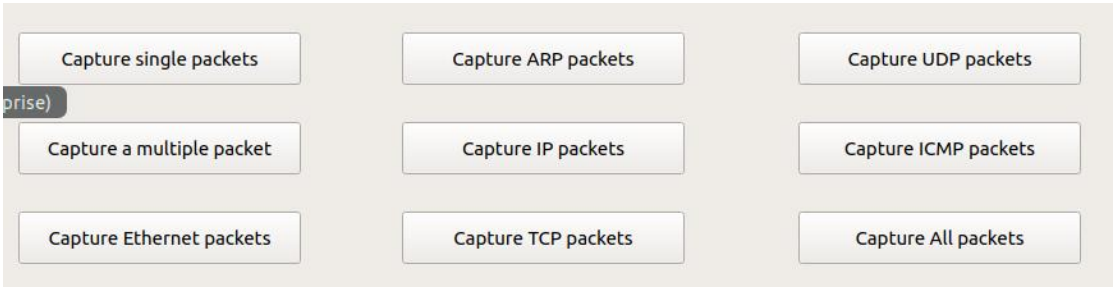


总共获取 22 个数据包，我们可以点击 NEXT 来查看第 15 个数据包：

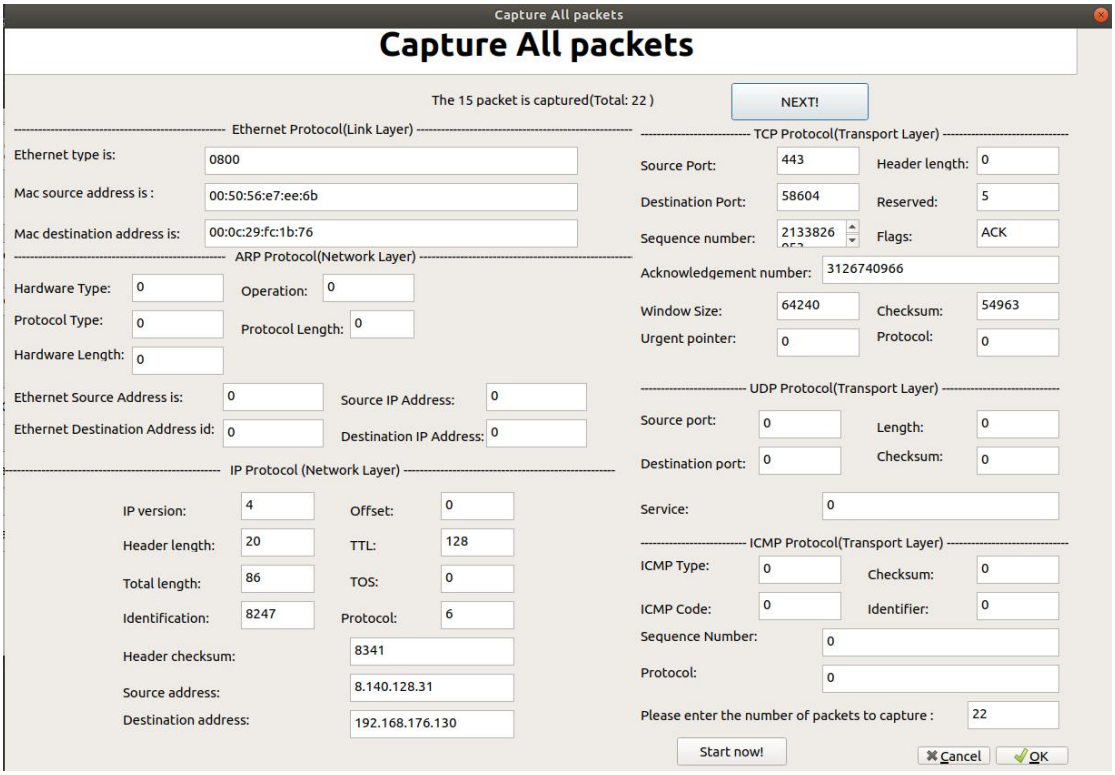


五、创新说明

1. 可以自主选择需要的操作，可自主选择抓取指定的数据包。



2. 使用 QT 编程，支持信息的可视化



评分表

考核标准	得分
能够理解实验原理，独立编写出程序，程序正确运行，并可以给出正确运行结果（40%）	
正确完成实践报告，实践报告内容完整、准确、格式规范（50%）	
创新性：算法创新、实现创新、界面创新等（10%）	
成绩小计：	