计算机网络实验报告

警示

1. **实验报告如有雷同，雷同各方当次实验成绩均以 0 分计。**
2. **当次小组成员成绩只计学号、姓名登录在下表中的。**
3. **在规定时间内未上交实验报告的，不得以其他方式补交，当次成绩按 0 分计。**
4. **实验报告文件以 PDF 格式提交。**

专业	计算机科学与技术（超算）	班 级	19 级行政 3 班
学号	19335074		
学生	黄玟瑜		

UDP 通信程序设计

【实验名称】

基于 UDP 丢包统计程序设计

【实验目的】

选择一个操作系统（Linux 或者 Windows），编制 UDP/IP 通信程序，完成一定的通信功能。

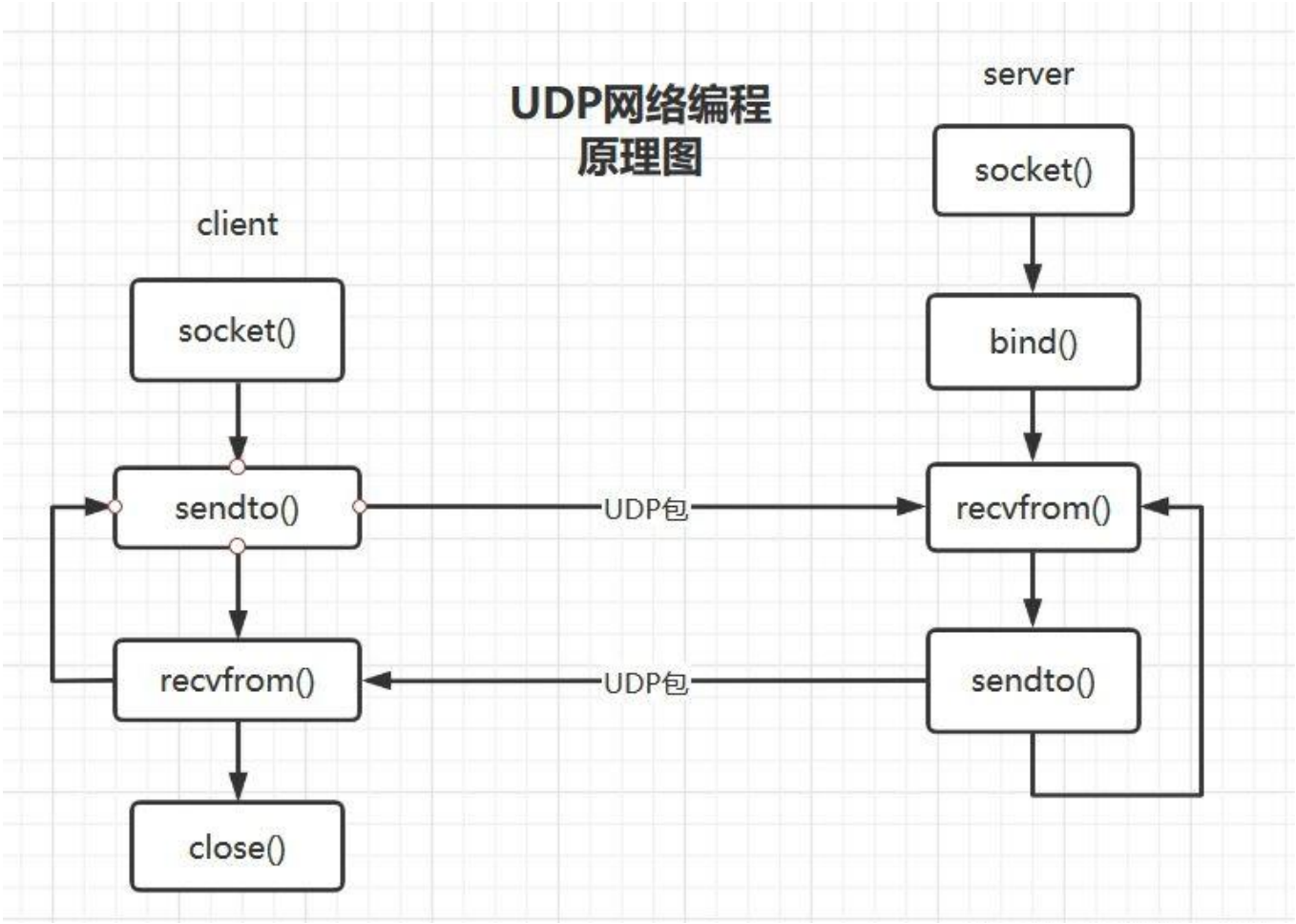
【实验要求】

在发送 UDP 数据包时做一个循环，连续发送 100 个数据包；在接收端统计丢失的数据包。

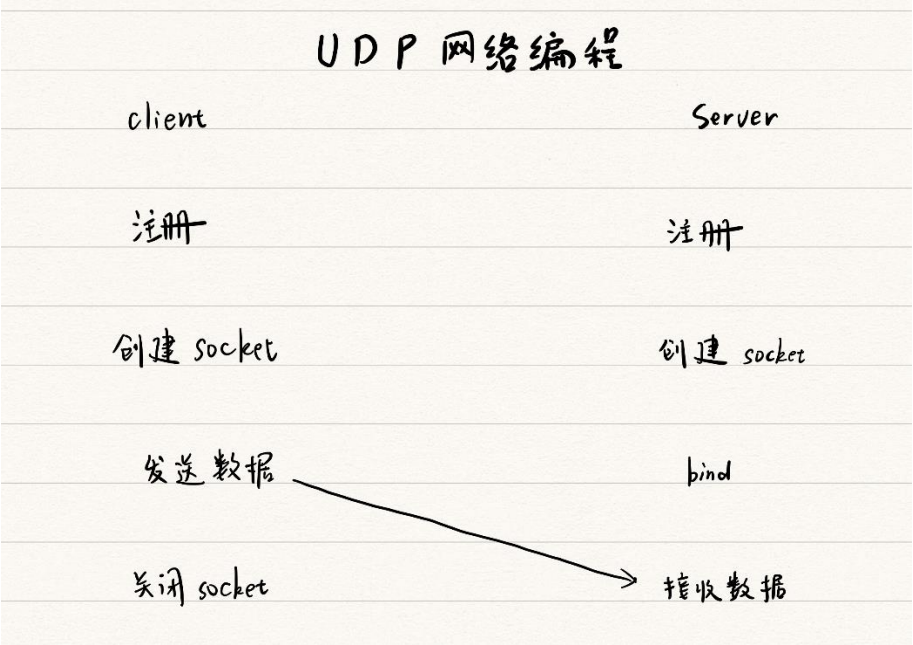
实验时，请运行 Wireshark 软件，对通信时的数据包进行跟踪分析。

【实验原理】

计算机网络实验报告



以上为一般 UDP 网络编程的流程图，在本次实验中仅涉及客户端发送数据和服务器接收数据，因此本次实验的实验流程图如下：



计算机网络实验报告

【实验内容】

根据流程图开始编程，下面进行代码分析：

客户端代码 UDP_Cli.cpp

```
/*创建 Socket*/
    SOCKET sockCli = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sockCli < 0)
    {
        cout << "Failed." << endl;
        return -1;
    }
    cout << "Create socket successfully." << endl;
```

调用库函数 socket 创捷套接字，若返回值<0 则说明创建套接字失败，退出程序。

socket 声明如下：

```
 WINSOCK_API_LINKAGE SOCKET WSAAPI socket(int af,int type,int protocol);
```

第一个参数指明了协议簇，目前支持 5 种协议簇，最常用的有 AF_INET(IPv4 协议)和 AF_INET6(IPv6 协议)；第二个参数指明套接口类型，有三种类型可选：SOCK_STREAM(字节流套接口)、SOCK_DGRAM(数据报套接口)和 SOCK_RAW(原始套接口)；如果套接口类型不是原始套接口，那么第三个参数就为 0。 在本次实验中使用 AF_INET 协议簇，SOCK_DGRAM 数据报接口，第三个参数为 UDP 的 protocol。

```
/*向指定地址和端口收发数据*/
    char recvBuf[BUFSIZE];      //接受数据的缓冲区
    string sendBu= "Hello server! This is a packet. Data: ";      //发送数据的缓冲区
    char tmp[BUFSIZE];
    SOCKADDR_IN addr_server; //服务器的地址数据结构
    addr_server.sin_family = AF_INET;
    addr_server.sin_port = htons(6666);                          //端口号为 6666
    addr_server.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");    //127.0.0.1 为本电脑 IP 地址
    int server_len = sizeof(addr_server);
    for (int i = 1; i <= 100; i++){

        itoa((rand() % 100000), tmp, 10);
        string sendBuf = sendBu + tmp;
```

```
        err = sendto(sockCli, sendBuf.data(), sendBuf.size(), 0, (SOCKADDR *)&addr_server,
 sizeof(SOCKADDR)); //发送
        if (err < 0){
            cout << "Sendto failed."<< endl;
            return -1;
        }
        else{
            cout << "Packet " << i << " has been sent." << endl;
        }
    }
```

使用 sendto 函数向客户端发送 100 个数据包，若发送成功则输出报告，失败则退出程序。每个数据包包括一句固定的问候语和需要发送的数据，在这里为 0~99999 的一个随机数，以字节为单位发送。

```
WINSOCK_API_LINKAGE int WSAAPI sendto(SOCKET s,const char *buf,int len,int flags,const str
uct sockaddr *to,int tolen);
```

sendto 函数：UDP 使用 sendto()函数发送数据，他类似于标准的 write()，但是在 sendto()函数中要指明目的地址。前三个参数等同于函数 read()的前三个参数，flags 参数是传输控制标志。参数 to 指明数据将发往的协议地址，他的大小由 addrlen 参数来指定。

它返回发送数据的长度大于或等于 0 说明发送成功，失败则返回-1。

```
/*关闭套接字*/
    closesocket(sockCli);
```

发送完毕后关闭套接字。

服务端代码 UDP_Ser.cpp

```
/*创建 Socket*/
    int sockSev = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sockSev < 0)
    {
        cout << "Failed to create socket." << endl;
        return -1;
    }
    cout << "Create socket successfully." << endl;
```

过程和客户端大致相同。

```
/*绑定 socket 和端口号*/
    SOCKADDR_IN addr_server;                                    //服务器的地址数据结构
    addr_server.sin_family = AF_INET;
    addr_server.sin_port = htons(6666);                         //端口号为 6666
    addr_server.sin_addr.S_un.S_addr=inet_addr("172.19.1.207");   //172.19.1.207 为本电脑 IP
地址

    if (bind(sockSev, (SOCKADDR *)&addr_server, sizeof(addr_server)) == SOCKET_ERROR)
    {
        cout<<"Failed to bind."<< endl;
        closesocket(sockSev);
        WSACleanup();
        return 0;
    }
    else
        cout << "Bind successfully." << endl;
```

创建服务器的地址数据结构并对其进行协议簇、端口号和 IP 地址的配置，再使用 bind 函数将创建好的 socket 绑定到该地址上。

```
  WINSOCK_API_LINKAGE int WSAAPI bind(SOCKET s,const struct sockaddr *name,int namelen);
```

bind 函数描述：把一个地址族中的特定地址赋给 socket

参数解释：

s：指的是通过 socket()创建的描述字，唯一标识一个 socket。

name：一个指针，指向要绑定的协议地址。

namelen：该地址结构体的长度

```
/*向指定地址和端口收发数据*/
    char recvBuf[BUFSIZE];              //接受数据的缓冲区
    SOCKADDR_IN addr_client;            //用于接收用户的 ip 地址和端口号等信息
    int client_len = sizeof(addr_client);
    int count = 0;
    while(true){
        int last = recvfrom(sockSev, recvBuf, BUFSIZE, 0, (SOCKADDR *)&addr_client, &clien
t_len);
        if (last <= 0)
```

```
        {
            cout << "Recvfrom Error!" << endl;
            continue;
        }
        else{
            cout << "Recvfrom:" << setw(7) << recvBuf;
            cout << "   Count:" << ++count << endl;
        }
    }
```

使用 recvfrom 函数监听发送来的数据，若接收成功则输出结果。同时使用 count 来累计成功接收到的数据包的个数。

```
 WINSOCK_API_LINKAGE int WSAAPI recvfrom(SOCKET s,char *buf,int len,int flags,struct sockaddr *from,int *fromlen);
```

参数解释：

s：标识一个已连接套接口的描述字。

buf：接收数据缓冲区。

len：缓冲区长度。

flags：调用操作方式。

from：（可选）指针，指向装有源地址的缓冲区。

fromlen：（可选）指针，指向 from 缓冲区长度值。

由于 Windows 系统下使用 socket 需进行注册，注册过程如下：

```
/*Winsocket 注册过程*/
    WORD wVersionRequested = MAKEWORD( 2, 0 );  // 请求 WinSock 库，高字节指明副版本，低字节指明主版本
    WSADATA wData;    // 这结构是用于接收 Wjndows Socket 的结构信息的（版本信息）
    int err;
    err = WSAStartup(wVersionRequested, &wData); //Winsock 服务初始化
    if ( err != 0 ) {
        cout << "Initialize failed."<<endl;
        return -1; // 返回值为零时表示成功 WSAStartup
    }
```

这段代码需要被放在客户端和服务器的代码中。

计算机网络实验报告

互联网环境

使用公网上的 UDP 的 echo 服务，将客户端本机地址与套接字绑定，进入监听，以便能接收到 echo。

```
    SOCKADDR_IN addr_client; //服务器的地址数据结构
    addr_client.sin_family = AF_INET;
    addr_client.sin_port = htons(6789);                          //端口号为 6789
    addr_client.sin_addr.S_un.S_addr=inet_addr("127.0.0.1");    //127.0.0.1 为本电脑 IP 地址
    if (bind(sockCli, (SOCKADDR *)&addr_client, sizeof(addr_client)) == SOCKET_ERROR)
    {
        cout<<"Failed to bind."<< endl;
        closesocket(sockCli);
        WSACleanup();
        return 0;
    }
    else
        cout << "Bind successfully." << endl;
```

将目标地址设置为公网上的地址

```
    SOCKADDR_IN addr_server; //服务器的地址数据结构
    addr_server.sin_family = AF_INET;
    addr_server.sin_port = htons(6789);
    addr_server.sin_addr.S_un.S_addr=inet_addr("8.129.101.161");
```

每次发送后接收 echo

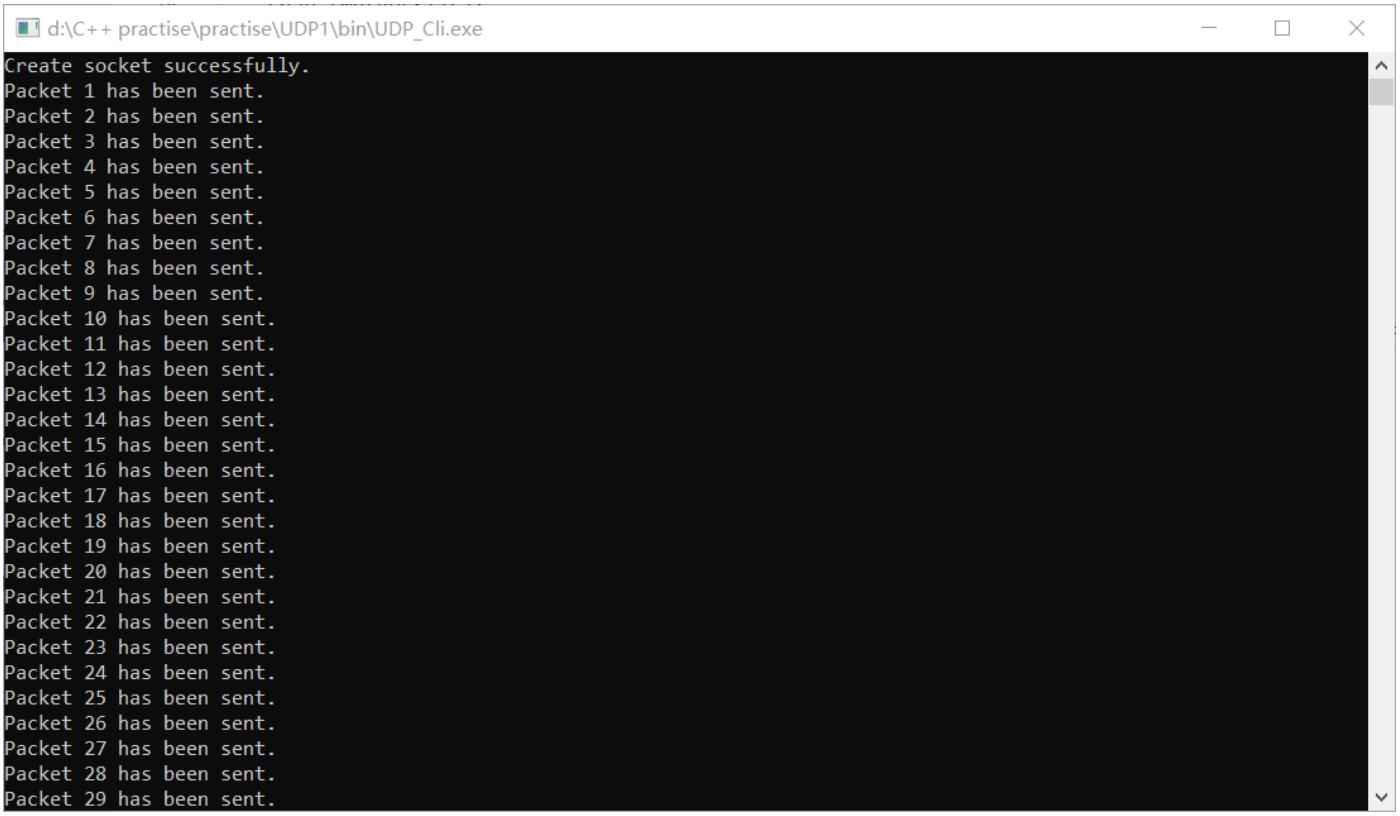
```
    //接收 echo 的数据
        int last = recvfrom(sockCli, recvBuf, BUFSIZE, 0, (SOCKADDR *)&addr_recv, &recv_le
n);
        if (last <= 0)
        {
            cout << "Recvfrom Error!" << endl;
            continue;
        }
        else{
            cout << "Recvfrom:" << setw(7) << recvBuf;
            cout << "   Count:" << ++count << endl;
        }
```

计算机网络实验报告

【实验结果】

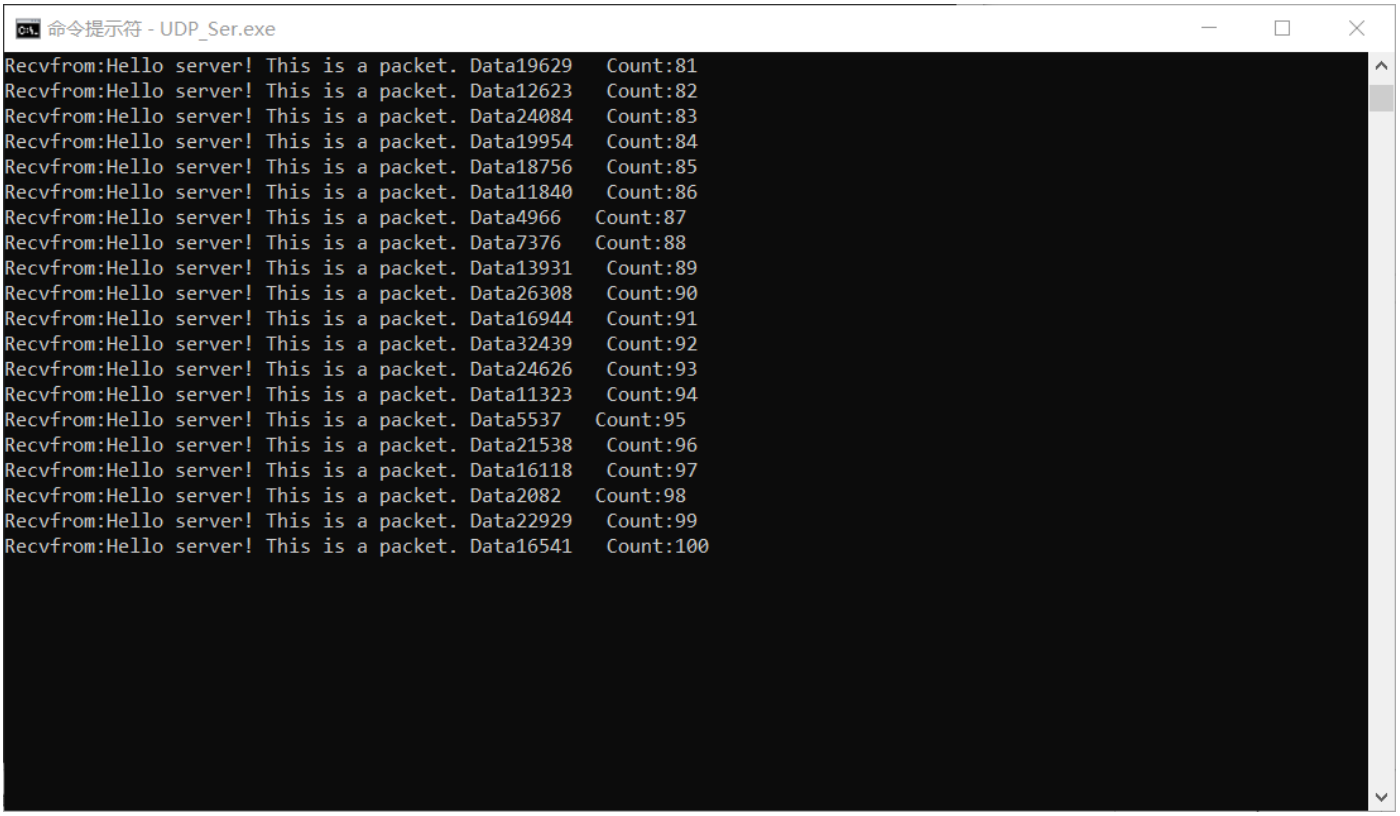
局域网环境下：

同时运行客户端程序和服务器程序，客户端界面如下：

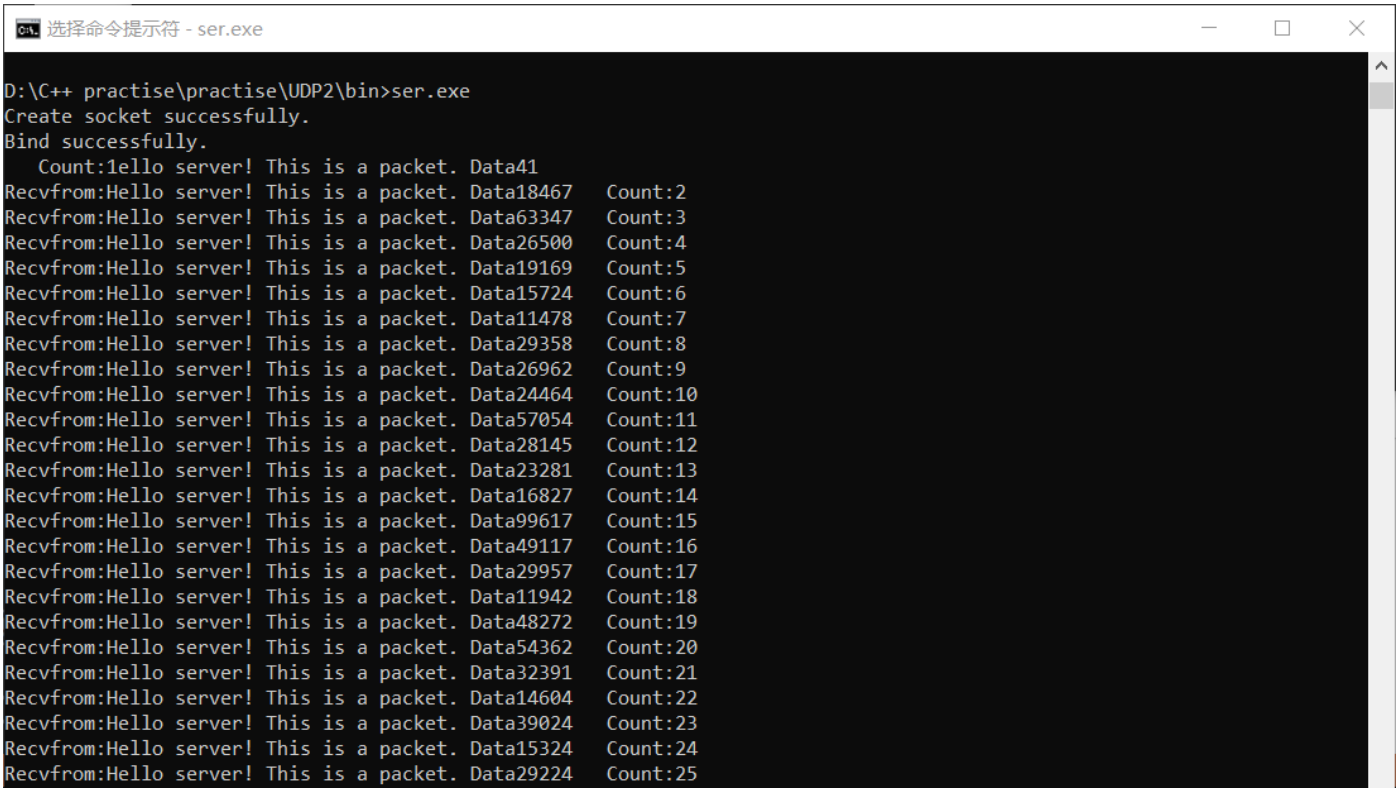


可以看到 socket 创建成功，并陆续向目的地址发送数据包。

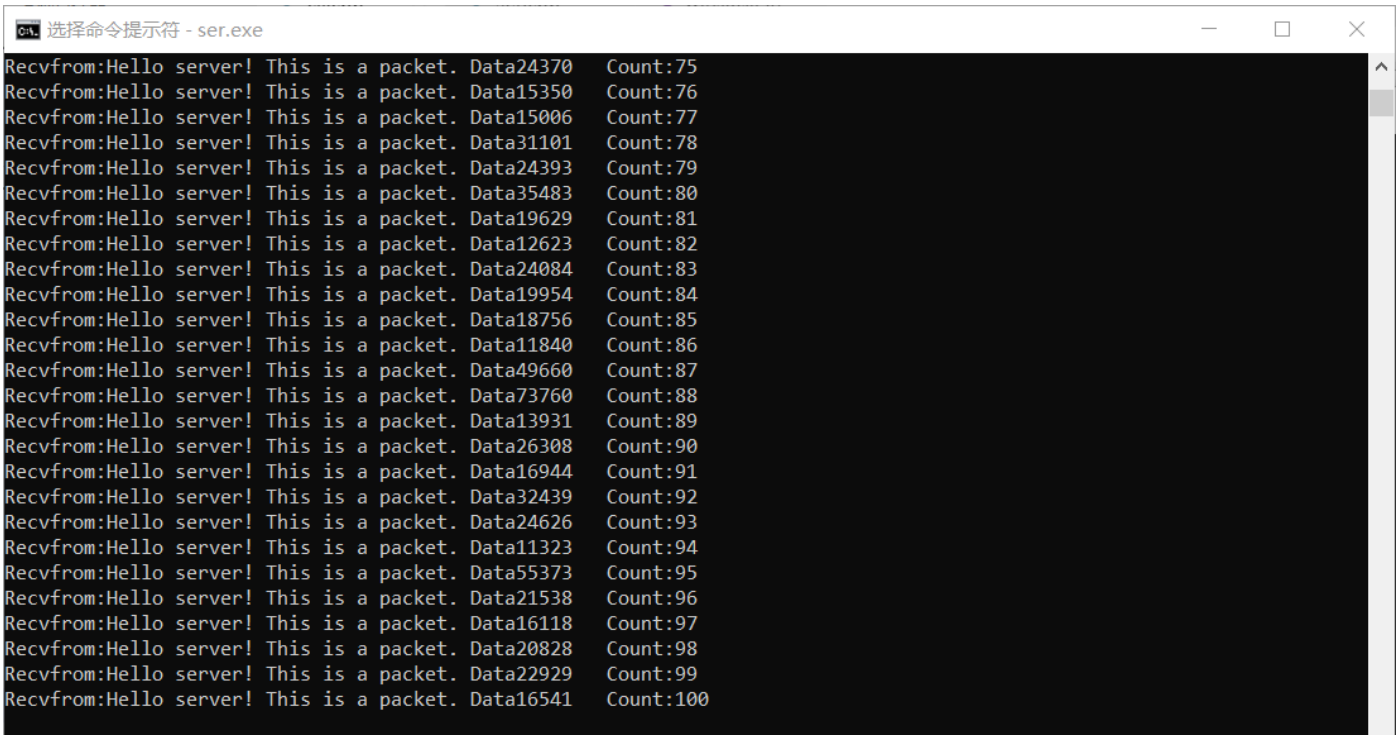
成功发送 100 个数据包，如下所示：



此时在服务器端看到服务器的 socket 创建成功，并成功 bind 上本机地址，开始陆续接收到来自客户端发送的数据，同时统计接收到的数据包的数量。

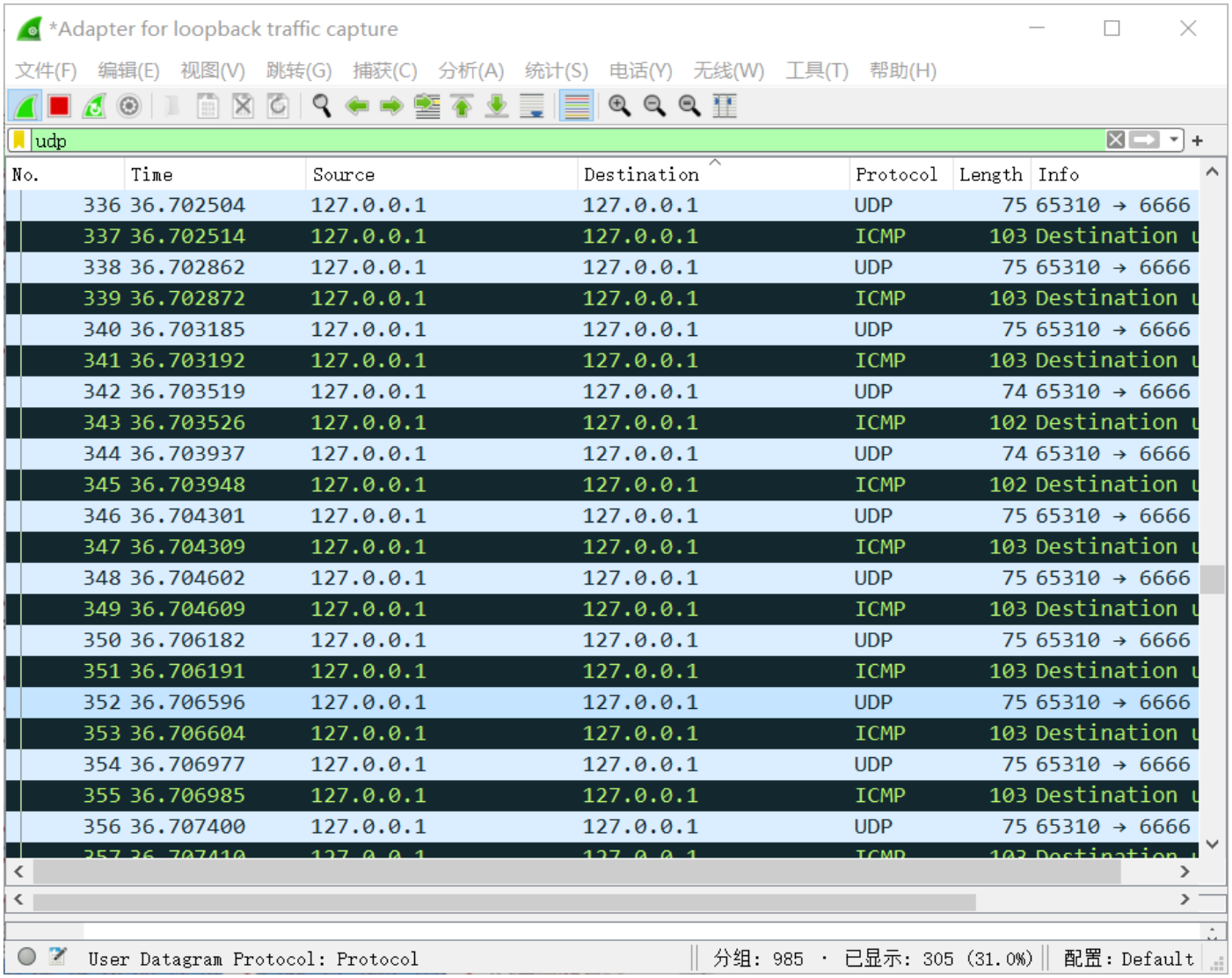


最终成功接收到 100 个数据包，无丢包的情况发生：

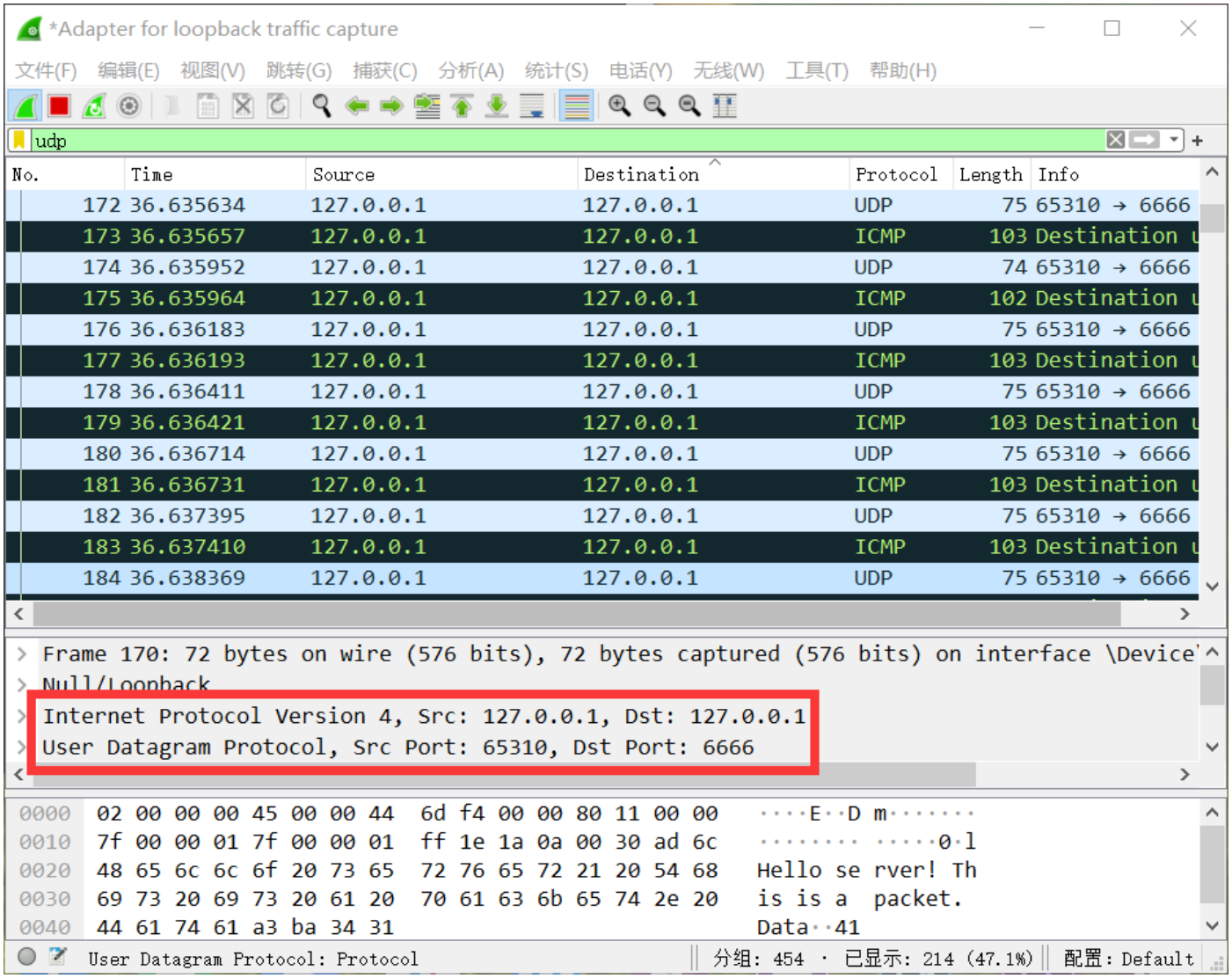


同时使用 wireshark 抓取数据包，可以看到 127.0.0.1（本机地址）发送了 100 个 UDP 类型的数据包：

计算机网络实验报告


*Adapter for loopback traffic capture
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
udp
No. Time Source Destination Protocol Length Info
336 36.702504 127.0.0.1 127.0.0.1 UDP 75 65310 → 6666
337 36.702514 127.0.0.1 127.0.0.1 ICMP 103 Destination u
338 36.702862 127.0.0.1 127.0.0.1 UDP 75 65310 → 6666
339 36.702872 127.0.0.1 127.0.0.1 ICMP 103 Destination u
340 36.703185 127.0.0.1 127.0.0.1 UDP 75 65310 → 6666
341 36.703192 127.0.0.1 127.0.0.1 ICMP 103 Destination u
342 36.703519 127.0.0.1 127.0.0.1 UDP 74 65310 → 6666
343 36.703526 127.0.0.1 127.0.0.1 ICMP 102 Destination u
344 36.703937 127.0.0.1 127.0.0.1 UDP 74 65310 → 6666
345 36.703948 127.0.0.1 127.0.0.1 ICMP 102 Destination u
346 36.704301 127.0.0.1 127.0.0.1 UDP 75 65310 → 6666
347 36.704309 127.0.0.1 127.0.0.1 ICMP 103 Destination u
348 36.704602 127.0.0.1 127.0.0.1 UDP 75 65310 → 6666
349 36.704609 127.0.0.1 127.0.0.1 ICMP 103 Destination u
350 36.706182 127.0.0.1 127.0.0.1 UDP 75 65310 → 6666
351 36.706191 127.0.0.1 127.0.0.1 ICMP 103 Destination u
352 36.706596 127.0.0.1 127.0.0.1 UDP 75 65310 → 6666
353 36.706604 127.0.0.1 127.0.0.1 ICMP 103 Destination u
354 36.706977 127.0.0.1 127.0.0.1 UDP 75 65310 → 6666
355 36.706985 127.0.0.1 127.0.0.1 ICMP 103 Destination u
356 36.707400 127.0.0.1 127.0.0.1 UDP 75 65310 → 6666
User Datagram Protocol: Protocol
分组: 985 · 已显示: 305 (31.0%)
配置: Default

计算机网络实验报告

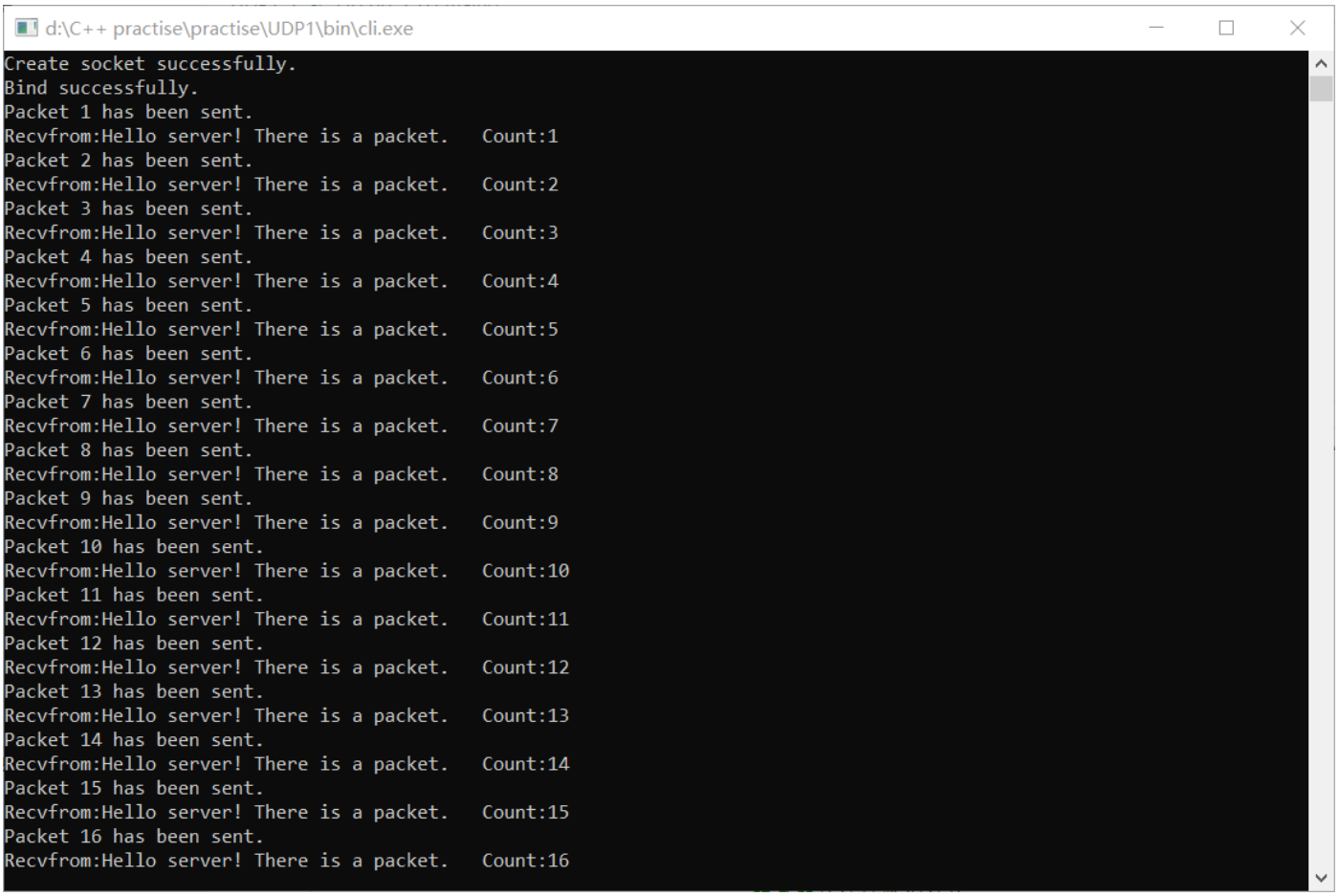


可以看到发送方的 ip 地址（主机地址）目的地的 ip 地址（也是主机地址）均为 127.0.0.1，目的地端口为 6666，正是本次实验所使用的端口。

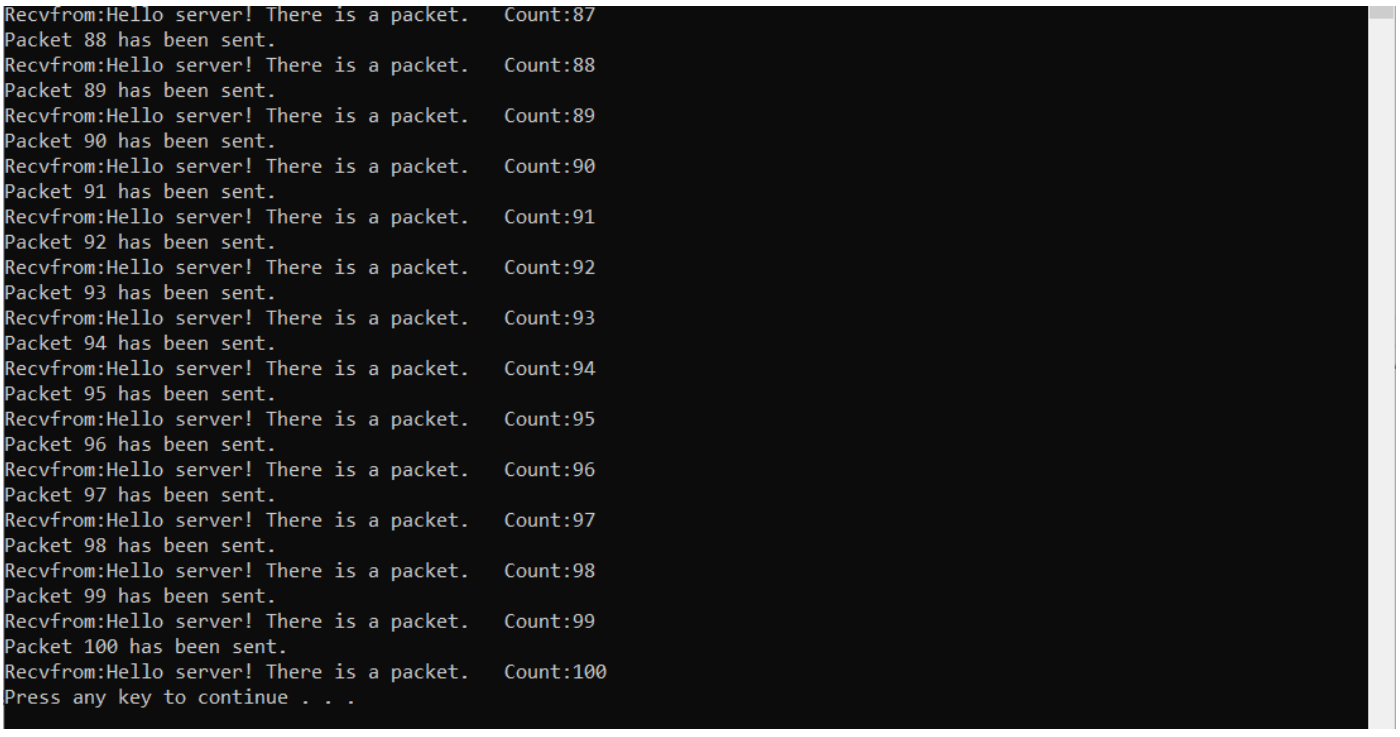
互联网环境下

运行客户端程序，向目的地址发送 100 个数据包，可以看到每发送一个数据包就显示了发送成功，同时监听到发回来的信号：

计算机网络实验报告



最终成功发送 100 个数据包，成功接收到发回的 100 个数据包。



计算机网络实验报告

【实验思考】

引起 UDP 丢包的可能原因是什么?

答：UDP 丢包的可能原因有如下几点：

1、接收端处理时间过长导致丢包：调用 recvfrom 方法接收端收到数据后，处理数据花了一些时间，处理完后再次调用 recvfrom 方法，在这二次调用间隔里,发过来的包可能丢失。对于这种情况可以修改接收端，将包接收后存入一个缓冲区，然后迅速返回继续 recvfrom。

2、发送的包巨大丢包：虽然 sendto 方法会帮你做大包切割成小包发送的事情，但包太大也不行。例如超过 50K 的一个 udp 包,不切割直接通过 sendto 方法发送也会导致这个包丢失。这种情况需要切割成小包再逐个 sendto。

3、发送的包较大，超过接受者缓存导致丢包：几个大的 udp 包可能会超过接收者的缓冲。

4、发送的包频率太快。

UDP 是无连接的，面向消息的数据传输协议，与 TCP 相比，有两个致命的缺点，一是数据包容易丢失，二是数据包无序。UDP 丢包是正常现象，因为它是不安全的。在本次实验中，我尝试了多次都没有丢包的现象，经过分析可能是因为发送的数据包太小（都不超过 1kb），或者是网络环境较好。

参考：

https://blog.csdn.net/yueguanghaidao/article/details/7055985

https://blog.csdn.net/qq_31837203/article/details/112121363