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。

院系	数据科学与计算机学院	班 级	软工教务二班	组长	郑卓民
学号	18342138	18342077			
学生	郑卓民	南樟			

编程实验

【实验内容】

（1）完成实验教程实例 3-2 的实验（考虑局域网、互联网两种实验环境），回答实验提出的问题及实验思考。（P103）。（于实验过程部分中体现）

（2）注意实验时简述设计思路。（于实验思考第二题中体现）

（3）引起 UDP 丢包的可能原因是什么？（于实验思考第九题体现）

实验 3-2 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

选择一个操作系统环境（Linux，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。

实验过程

实验准备：

实验选用的实验环境为 window 环境，UDP 通信程序使用的编程语言为 C 语言。

实验考虑两种主机网络环境，一为两台主机位于一个局域网下，二为两台主机位于两个不同的局域网下。

无论是局域网环境还是互联网环境，服务器和客户端的 C 程序代码都是一样的，不同的只是各自监听或发送目的地的 IP 和端口号不同，此外在互联网环境下的 UDP 通信，由于两台主机位于不同的局域网下，需要使用到内网穿透技术才能进行 UDP 通信。

内网穿透技术介绍与原理：

在网络编码中会发现程序在局域网中是可以适用的，但是在外网与内网之间和内网与内网之间就不可行。此时就要用到内网穿透，即 NAT 穿透，网络连接时术语，计算机是局域网内时，外网与内网的计算机节点需要连接通信，有时就会出现不支持内网穿透。就是说映射端口，能让外网的电脑找

到处于内网的电脑，提高下载速度。不管是内网穿透还是其他类型的网络穿透，都是网络穿透的统一方法来研究和解决。

由于内网穿透的实现较为复杂，本实验中的内网穿透借助第三方软件（NATAPP）进行，在此软件中，可以创建一个基于 UDP 协议的隧道，并且会分配一个公网域名与远程端口，使得内网主机相当于拥有了一个公网 IP 的能力，其他局域网中的主机对该域名和端口的 UDP 访问，将会映射到服务器本地的端口 127.0.0.1 的对应已设置的端口。

实验代码部分，主要利用到了 C 语言的 winsock 库进行 Socket 通信，完整代码贴在实验报告末尾，此处截取 UDP 通信过程中部分关键功能对应的函数的编写：

初始化网络环境：

```
WSADATA wsa;
if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
{
    printf("WSAStartup failed\n");
    return -1;
}
```

创建 UDP 的 Socket：

```
//建立一个UDP的socket
SOCKET sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (sock == SOCKET_ERROR)
{
    printf("create socket failed\n");
    return -1;
}
```

服务器绑定地址信息：

```
//绑定地址信息
sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT);
serverAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
```

客户端配置服务器地址信息：

```
// 申明一个网络地址信息的结构体，保存服务器的地址信息
sockaddr_in addr = { 0 };
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.S_un.S_addr = inet_addr("112.74.89.58");
```

客户端收发信息：

发送：

```
//发送数据
for(int i=1; i<=n; i++) {
    int dwSent = sendto(sockClient, buf, strlen(buf), 0, (SOCKADDR *)&addr, sizeof(SOCKADDR));
    if (dwSent <= 0)
    {
        printf("send failed\n");
        cnt++;
    } // else printf("Send msg: %s\n", buf);
}
```

接受：

```
for(int i=1;i<=n-cnt;i++){
    // 接收数据
    int dwRecv = recvfrom(sockClient, recvBuf, 512, 0, (SOCKADDR *)&addrSever,&nServerAddrLen);
    if(dwRecv<=0) {
        printf("timeout:\"%s\" %d\n", inet_ntoa(addrSever.sin_addr),i);
        cnt2++;
    } else printf("Recv msg from server [%s] : %s\n", inet_ntoa(addrSever.sin_addr), recvBuf);
}
```

服务器收发信息：

```
while (TRUE)
{
    memset(buf, 0, 512);
    // 网络节点的信息，用来保存客户端的网络信息
    sockaddr_in clientAddr;
    memset(&clientAddr, 0, sizeof(sockaddr_in));

    int clientAddrLen = sizeof(SOCKADDR);
    //接收客户端发来的数据
    int ret = recvfrom(sock, buf, 512, 0,(SOCKADDR*) &clientAddr,&clientAddrLen );

    printf("Recv msg:%s from IP:[%s] Port:[%d]\n", buf,inet_ntoa(clientAddr.sin_addr),ntohs(clientAddr.sin_port));
    // 发一个数据包返回给客户
    sendto(sock, "Hello Client!", strlen("Hello Client!"), 0, (SOCKADDR*)&clientAddr, clientAddrLen);
    printf("Send msg back to IP:[%s] Port:[%d]\n", inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
}
```

内网穿透软件配置好、服务器和客户端的代码都编写完成后，就可以进行实验了。下面分局域网环境和互联网环境分别记录实验过程。

局域网环境：

局域网环境下的实验由两部分组成，案例一为一台主机既做服务器也做客户端，案例二为一台主机做服务器，另设一台主机做客户端。

案例一：

服务器 sockaddr 的配置：

绑定本机 IP 对应端口（可取 8080）

```
//绑定地址信息
sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT);
serverAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
bind(sock, (SOCKADDR*)&serverAddr, sizeof(SOCKADDR));
```

客户端 sockaddr 的配置：

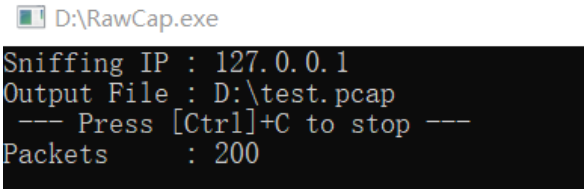
本机 IP 地址，以及前面服务器里设置的监听的 PORT

```
// 申明一个网络地址信息的结构体，保存服务器的地址信息
sockaddr_in addr = { 0 };
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
```

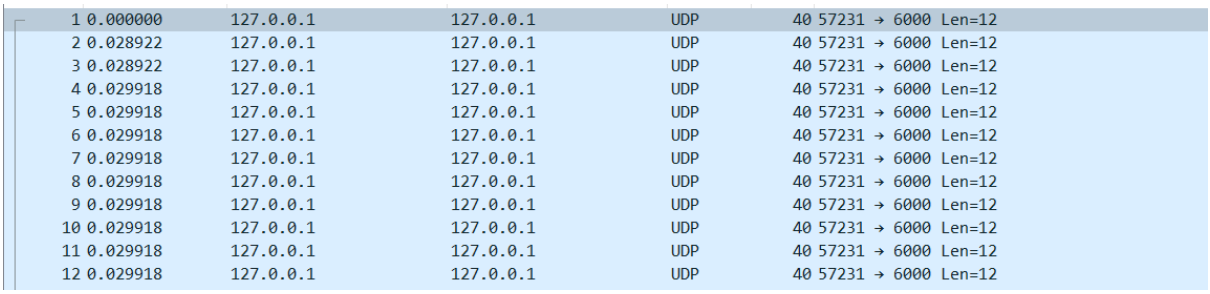
Wireshark 抓包：

由于 Wireshark 虽然强大，但是无法抓取本地回环数据。故使用 RawCap 软件来捕获本地回环的数据包，然后用 Wireshark 打开查看结果。

RawCap 抓包情况 ：发了 100 个收到 100 个包，符合实际。

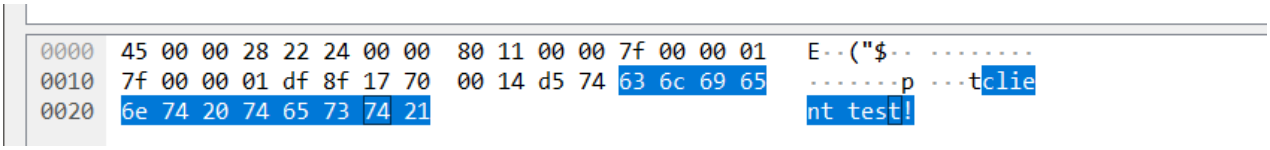


Wireshark 查看：



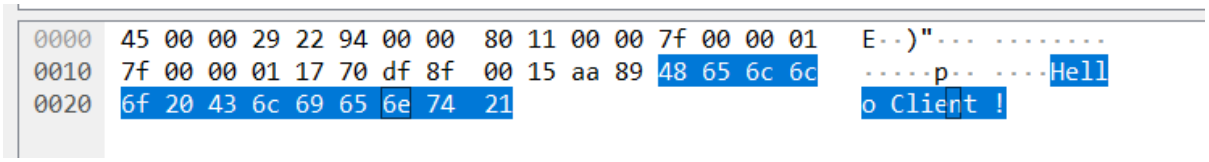
1	0.000000	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
2	0.028922	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
3	0.028922	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
4	0.029918	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
5	0.029918	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
6	0.029918	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
7	0.029918	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
8	0.029918	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
9	0.029918	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
10	0.029918	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
11	0.029918	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12
12	0.029918	127.0.0.1	127.0.0.1	UDP	40 57231 → 6000 Len=12

打开其中一个数据包：



```
0000  45 00 00 28 22 24 00 00  80 11 00 00 7f 00 00 01   E··("$·· ········
0010  7f 00 00 01 df 8f 17 70  00 14 d5 74 63 6c 69 65   ·······p ···tclie
0020  6e 74 20 74 65 73 74 21                           nt test!
```

在 DATA 中可以看到外面发送的 Client test！数据。



```
0000  45 00 00 29 22 94 00 00  80 11 00 00 7f 00 00 01   E··)"··· ········
0010  7f 00 00 01 17 70 df 8f  00 15 aa 89 48 65 6c 6c   ·····p·· ····Hell
0020  6f 20 43 6c 69 65 6e 74  21                       o Client !
```

以及服务端发送回来的 Hello Client！数据。

丢包率及简要分析：无丢包情况，局域网下通信十分稳定。

案例二：

服务器 sockaddr 的配置：

```
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT);
serverAddr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);
```

客户端 sockaddr 的配置：

```
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.S_un.S_addr = inet_addr("192.168.5.4");
```

Wireshark 抓包：

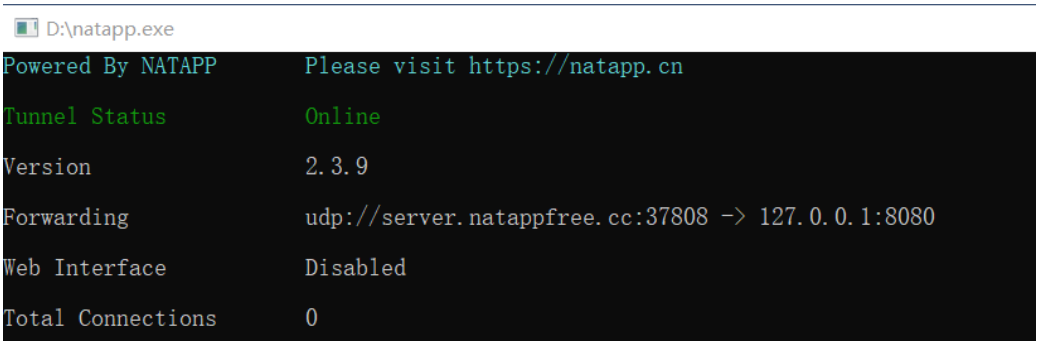
35	7.434899	192.168.5.9	192.168.5.4	UDP	60 60932 → 37670 Len=12
36	7.434901	192.168.5.9	192.168.5.4	UDP	60 60932 → 37670 Len=12
37	7.434903	192.168.5.9	192.168.5.4	UDP	60 60932 → 37670 Len=12
38	7.434904	192.168.5.9	192.168.5.4	UDP	60 60932 → 37670 Len=12
39	7.435406	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12
40	7.436019	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12
41	7.436252	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12
42	7.436447	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12
43	7.436489	192.168.5.9	192.168.5.4	UDP	60 60932 → 37670 Len=12
44	7.436491	192.168.5.9	192.168.5.4	UDP	60 60932 → 37670 Len=12
45	7.436492	192.168.5.9	192.168.5.4	UDP	60 60932 → 37670 Len=12
46	7.436675	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12
47	7.436872	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12
48	7.437060	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12
49	7.437252	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12
50	7.437443	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12
51	7.437636	192.168.5.4	192.168.5.9	UDP	54 37670 → 60932 Len=12

丢包率及简要分析：100 个包接收到了 100 个，说明在局域网中即便使用 udp 通信，丢包率极低。

互联网环境：

此处首先要开启内网穿透软件（即获取一个服务器域名和端口号），然后配置客户端程序中应该请求的 IP 地址和端口号。

NATAPP 运行截图：其中的 server.natappfree.cc 域名对应的 ip 地址以及后面的 37808 端口即我们客户端要发送信息的目的地，其能将信息映射到内网服务器主机的 8080 端口。



服务器 sockaddr 的配置：监听本机的 8080 端口。

```
//绑定地址信息
sockaddr_in serverAddr;
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT); // 8080
serverAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
bind(sock, (SOCKADDR*)&serverAddr, sizeof(SOCKADDR));
```

客户端 sockaddr 的配置：向 112.74.89.58 的 37808 端口发送 UDP 包。

```
// 申明一个网络地址信息的结构体，保存服务器的地址信息
sockaddr_in addr = { 0 };
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT); // 37808
addr.sin_addr.S_un.S_addr = inet_addr("112.74.89.58");
```

Wireshark 抓包：

32	11.445625	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
33	11.445786	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
34	11.445846	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
35	11.445896	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
36	11.445949	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
37	11.445996	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
38	11.446043	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
39	11.446091	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
40	11.446139	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
41	11.446187	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
42	11.446233	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
43	11.446279	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
44	11.446326	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
45	11.446370	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
46	11.446419	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
47	11.446465	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12
48	11.446512	192.168.88.105	112.74.89.58	UDP	54 61285 → 37808 Len=12

```
0000  78 44 76 00 5f df 00 e0  4c 36 01 e8 08 00 45 00   xDv·_··· L6····E·
0010  00 28 b5 97 00 00 80 11  00 00 c0 a8 58 69 70 4a   ·(······ ····XipJ
0020  59 3a ef 65 93 b0 00 14  e2 bb 63 6c 69 65 6e 74   Y:·e···· ··client
0030  20 74 65 73 74 21                                  test!
```

166	11.505173	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
171	11.530088	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
176	11.561716	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
184	11.599696	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
191	11.634921	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
195	11.647491	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
202	11.674936	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
208	11.705472	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
212	11.733646	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
216	11.755191	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
221	11.783266	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13
228	11.806913	112.74.89.58	192.168.88.105	UDP	60 37808 → 61285 Len=13

```
0000  00 e0 4c 36 01 e8 78 44  76 00 5f df 08 00 45 00   ··L6··xD v·_···E·
0010  00 29 17 e2 40 00 31 11  4f 4c 70 4a 59 3a c0 a8   ·)··@·1· OLpJY:··
0020  58 69 93 b0 ef 65 00 15  39 df 48 65 6c 6c 6f 20   Xi···e·· 9·Hello
0030  43 6c 69 65 6e 74 21 00  58 69 93 b0               Client!· Xi··
```

丢包率及简要分析：

发送 100 个数据包的情况：丢包最高不超过 5 个，虽然比局域网下的 UDP 丢包更多，但是情况算是比较理想。

```
send 100 msg.
timeout:"0.0.0.0" 1
timeout:"112.74.89.58" 74
timeout:"112.74.89.58" 86
Recv 97 msg.
```

发送 500 个数据包的情况：下图可见，500 个数据包丢了两百多个，随着短时间发送的数据包的数量越来越多，丢包只会越来越多，丢包率越来越高。

```
send 500 msg.
timeout:"112.74.89.58" 293
timeout:"112.74.89.58" 294
timeout:"112.74.89.58" 295
```

```
timeout:"112.74.89.58" 498
timeout:"112.74.89.58" 499
timeout:"112.74.89.58" 500
Recv 292 msg.
```

如何解决？

在连续发送数据包的循环里面，增加一个 Sleep 函数，即每个数据包的发送之间相隔 50ms。就能实现丢包率为 0，或者丢极少的包。

```
send 500 msg.
Recv 500 msg.
```

```
for(int i=1; i<=n; i++) {
    Sleep(50);
    int dwSent = sendto(sockClie
```

计算机网络实验报告

实验思考

1. 说明在实验过程中遇到的问题和解决办法。

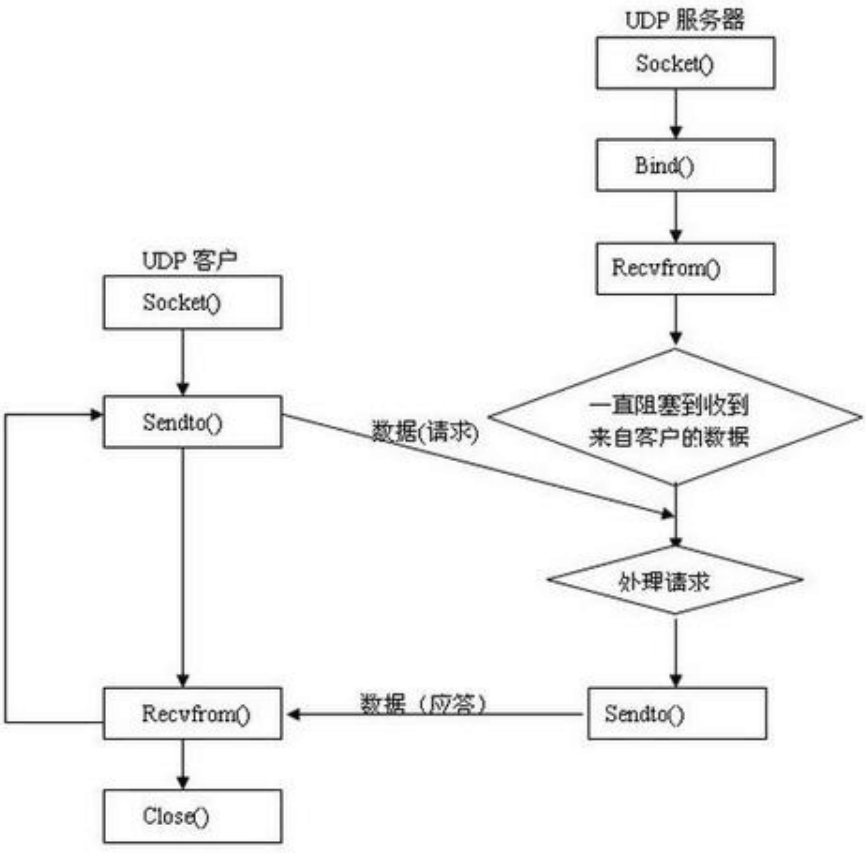
1.系统不匹配：首先我们是在 window 下用 C 语言进行编程，考虑到局域网内的通信，我们首先想到的是用虚拟机代替第二台主机进行通信，在拷贝文件后发现部分库的函数仅在 window 操作系统下存在，需要改变库或者改写程序。最后我借助家中的台式与本身的笔记本进行 udp 通信。

2.系统版本和位数不匹配：其中由于台式过于老久，其系统为 32 位系统，我们不得不改变编译器使得其产生 32 位的可执行文件来给台式进行通信。

3. 本次实验中尝试本地主机既作为服务器，也作为客户端，虽然一切运行正常，但是使用 wireshark 进行抓包的时候发现并不能抓到相关的数据包，通过网络检索，原来 wireshark 是不支持本地回环数据包捕获的，但是 RawCap 却可以抓取本地回环数据，使用命令“RawCap.exe 127.0.0.1 dumpfile.pcap”，将捕获到的数据包使用 wireshark 打开就可以分析数据包了。

4.公网连接困难：由于默认的路由器只分配给电脑内网地址，所以在我们准备进行互联网 udp 通讯的时候遇到了巨大的阻碍。一开始我们单纯地以为知道对方的公网地址后，往公网地址发送即可，后证实此方法不行；然后我们采取了路由器端口转接的方法，但发现端口转接后仍无法接收到对方发来的请求；最后我们采取了 NAT 穿透技术，利用短期免费的隧道使得我们能够通过公网进行连接，从而完成 udp 通信。

2. 给出程序详细的流程图和对程序关键函数的详细说明。



sendto()
简述：向一指定目的地发送数据。

#include <winsock.h>

int PASCAL FAR sendto(SOCKET s, const char FAR* buf, int len, int flags, const struct sockaddr FAR* to, int tolen);

s：一个标识套接口的描述字。 buf：包含待发送数据的缓冲区。 len：buf 缓冲区中数据的长度。 flags：调用方式标志位。 to：（可选）指针，指向目的套接口的地址。 tolen：to 所指地址的长度。

注释：

sendto()适用于已连接的数据报或流式套接口发送数据。对于数据报类套接口，必需注意发送数据长度不应超过通讯子网的 IP 包最大长度。IP 包最大长度在 WSAStartup()调用返回的 WSAData 的 iMaxUdpDg 元素中。如果数据太长无法自动通过下层协议，则返回 WSAEMSGSIZE 错误，数据不会被发送。 请注意成功地完成 sendto()调用并不意味着数据传送到达。 sendto()函数主要用于 SOCK_DGRAM 类型套接口向 to 参数指定端的套接口发送数据报。对于 SOCK_STREAM 类型套接口，to 和 tolen 参数被忽略；这种情况下 sendto()等价于 send()。 为了发送广播数据（仅适用于 SOCK_DGRAM），in 参数所含地址应该把特定的 IP 地址 INADDR_BROADCAST（winsock.h 中有定义）和终端地址结合起来构造。通常建议一个广播数据报的大小不要大到以致产生碎片，也就是说数据报的数据部分（包括头）不超过 512 字节。 如果传送系统的缓冲区空间不够保存需传送的数据，除非套接口处于非阻塞 I/O 方式，否则 sendto()将阻塞。对于非阻塞 SOCK_STREAM 类型的套接口，实际写的数据数目可能在 1 到所需大小之间，其值取决于本地和远端主机的缓冲区大小。可用 select()调用来确定何时能够进一步发送数据。 在相关套接口的选项之上，还可通过标志位 flag 来影响函数的执行方式。也就是说，本函数的语义既取决于套接口的选项也取决于标志位。

recvfrom()
简述：接收一个数据报并保存源地址。

#include <winsock.h>

int PASCAL FAR recvfrom(SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen);

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。

注释：

本函数由于从（已连接）套接口上接收数据，并捕获数据发送源的地址。 对于 SOCK_STREAM 类型的套接口，最多可接收缓冲区大小个数据。如果套接口被设置为线内接收带外数据（选项为 SO_OOBINLINE），且有带外数据未读入，则返回带外数据。应用程序可通过调用 ioctlsocket()的 SOCATMARK 命令来确定是否有带外数据待读入。对于 SOCK_STREAM 类型套接口，忽略 from 和 fromlen 参数。 对于数据报类套接口，队列中第一个数据报中的数据被解包，但最多不超过缓冲区的大小。如果数据报大于缓冲区，那么缓冲区中只有数据报的前面部分，其他的数据都丢失了，并且 recvfrom()函数返回 WSAEMSGSIZE 错误。 若 from 非零，且套接口为 SOCK_DGRAM 类型，则发送数据源的地址被复制到相应的 sockaddr 结构中。fromlen 所指向的值初始化时为这个结构的大小，

当调用返回时按实际地址所占的空间进行修改。 如果没有数据待读，那么除非是非阻塞模式，不然的话套接口将一直等待数据的到来，此时将返回 SOCKET_ERROR 错误，错误代码是 WSAEWOULDBLOCK。用 select()或 WSAAsynSelect()可以获知何时数据到达。 如果套接口为 SOCK_STREAM 类型，并且远端“优雅”地中止了连接，那么 recvfrom()一个数据也不读取，立即返回。如果立即被强制中止，那么 recv()将以 WSAECONNRESET 错误失败返回。 在套接口的所设选项之上，还可用标志位 flag 来影响函数的执行方式。也就是说，本函数的语义既取决于套接口选项，也取决于标志位参数。

3. 使用 Socket API 开发通信程序中的客户端程序和服务器程序时，各需要哪些不同的函数？

客户端：closesocket() ：由于服务器是一直开着的，故对于 Socket 服务可以选择关闭或不关闭，而客户端的 Socket 则在使用完后需要进行关闭。

服务端：bind() ：服务器绑定侦听端口，使用 bind()函数，将套接字文件描述符和一个地址类型变量进行绑定。

4. 解释 connect(), bind() 等函数中 struct sockaddr * addr 参数各个部分的含义，并用具体的数据举例说明。

struct sockaddr_in 的各个参数意义：

```
struct sockaddr_in {
short int sin_family; /* 地址族，AF_xxx 在 socket 编程中只能是 AF_INET */
unsigned short int sin_port; /* 端口号 （使用网络字节顺序） */
struct in_addr sin_addr; /* 存储 IP 地址 4 字节 */
unsigned char sin_zero[8]; /* 总共 8 个字节，实际上没有什么用，只是为了和 struct sockaddr 保持一样的长度 */
};
```

具体实例：

```
serverAddr.sin_family = AF_INET;（只能是这个）
serverAddr.sin_port = htons(PORT);（绑定当前指定的端口）
serverAddr.sin_addr.S_un.S_addr = htonl(INADDR_ANY);（将所有网卡从主机字节顺序变成网络字节顺序并进行绑定）
```

5. 说明面向连接的客户端和面向非连接的客户端在建立 Socket 时有什么区别。

面向连接的客户端需要先与服务端构建一条连接，然后在进行通信传输。

面向非连接的客户端则不需要预先构建一条连接，可以直接通信传输。

6. 说明面向连接的客户端和面向非连接的客户端在收发数据时有什么区别。面向非连接的客户端又是如何判断数据发送结束的？

面对连接的客户端在收发数据时是由客户端指定一个端口进行收发

面对非连接的客户端在收发数据时是由系统分配一个空余的端口进行收发。

由于非连接的客户端收到的包是无序的，所以我们可以采用一段时间内假如没有收到新的包来判断数据发送结束。或者采用每包确认的方法，让发送端每发送一个包，客户端收到就确认，最终判断数据发送结束。

7. 比较面向连接的通信和无连接通信，他们各有什么优点和缺点？适合在哪种场合下使用？

无连接套接字传输效率高，但是不可靠，有丢失数据包、捣乱数据的风险。

有连接套接字非常可靠，万无一失，但是传输效率低，耗费资源多。

对可靠性要求比较高，必须数据包能够完整无误地送达，那就得选择有连接的套接字（TCP 服务），比如 HTTP、FTP 等；而假如并不需要那么高的可靠性，效率和实时才是它们所关心的，那就可以选择无连接的套接字（UDP 服务），比如 DNS、即时聊天工具等

8. 实验过程中使用 Socket 时是工作在阻塞方式还是非阻塞方式？通过网络检索阐述这两种操作方式的不同。

实验过程中 Socket 在工作时是阻塞方式。

在阻塞模式下，在 I/O 操作完成前，执行的操作函数一直等候而不会立即返回，该函数所在的线程会阻塞在这里。相反，在非阻塞模式下，套接字函数会立即返回，而不管 I/O 是否完成，该函数所在的线程会继续运行。

在阻塞模式的套接字上，调用任何一个 Windows Sockets API 都会耗费不确定的等待时间。图所示，在调用 recv()函数时，发生在内核中等待数据和复制数据的过程。

当调用 recv()函数时，系统首先查是否有准备好的数据。如果数据没有准备好，那么系统就处于等待状态。当数据准备好后，将数据从系统缓冲区复制到用户空间，然后该函数返回。在套接应用程序中，当调用 recv()函数时，未必用户空间就已经存在数据，那么此时 recv()函数就会处于等待状态。

把套接字设置为非阻塞模式，即通知系统内核：在调用 Windows Sockets API 时，不要让线程睡眠，而应该让函数立即返回。在返回时，该函数返回一个错误代码。一个非阻塞模式套接字多次调用 recv()函数的过程。前三次调用 recv()函数时，内核数据还没有准备好。因此，该函数立即返回 WSAEWOULDBLOCK 错误代码。第四次调用 recv()函数时，数据已经准备好，被复制到应用程序的缓冲区中，recv()函数返回成功指示，应用程序开始处理数据。

9. 引起 UDP 丢包的原因是什么？

1.接收端处理时间过长导致丢包：调用 recv 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 recv 方法，在这二次调用间隔里,发过来的包可能丢失。

2.发送的包巨大丢包：例如超过 50K 的一个 udp 包，不切割直接通过 send 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 send。

3.发送的包较大，超过接受者缓存导致丢包：

4.发送的包频率太快：虽然每个包的大小都小于 mtu size 但是频率太快，例如 40 多个 mtu size 的包连续发送中间不 sleep，也有可能导致丢包。

学生	学号	自评分
南樟	18342077	100
郑卓民	18342138	100

计算机网络实验报告

代码：

Server.c:

```c
#include <stdio.h>
#include <winsock2.h>
#include <Windows.h>

#pragma comment(lib,"ws2_32.lib")
#define  PORT 8080

typedef struct sockaddr_in sockaddr_in;

int main(int argc, char* argv[])
{
    //初始化网络环境
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    {
        printf("WSAStartup failed\n");
        return -1;
    }
    //建立一个UDP的socket
    SOCKET sock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sock == SOCKET_ERROR)
    {
        printf("create socket failed\n");
        return -1;
    }
    //绑定地址信息
    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(PORT); // 8080
    serverAddr.sin_addr.S_un.S_addr = inet_addr("127.0.0.1");
    bind(sock, (SOCKADDR*)&serverAddr, sizeof(SOCKADDR));
    char buf[512];
    while (TRUE)
    {
        memset(buf, 0, 512);
        // 网络节点的信息，用来保存客户端的网络信息
        sockaddr_in clientAddr;
        memset(&clientAddr, 0, sizeof(sockaddr_in));
        int clientAddrLen = sizeof(SOCKADDR);
        //接收客户端发来的数据
        int ret = recvfrom(sock, buf, 512, 0,(SOCKADDR*) &clientAddr,&clientAddrLen );
        printf("Recv msg:%s from IP:[%s] Port:[%d]\n", buf,inet_ntoa(clientAddr.sin_addr),ntohs(clientAddr.sin_port));
        // 发一个数据包返回给客户
        sendto(sock, "Hello Client!", strlen("Hello Client!"), 0, (SOCKADDR*)&clientAddr, clientAddrLen);
        printf("Send msg back to IP:[%s] Port:[%d]\n", inet_ntoa(clientAddr.sin_addr), ntohs(clientAddr.sin_port));
    }
    return 0;
}
```

计算机网络实验报告

Client.c:

```c
#include <stdio.h>
#include <winsock2.h>
#include <Windows.h>
#pragma comment(lib,"ws2_32.lib")
#define  PORT 37808
typedef struct sockaddr_in sockaddr_in;

int main(int argc, char* argv[])
{   //初始化网络环境
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0) {
        printf("WSAStartup failed\n");
        return -1;
    }
    //建立一个UDP的socket
    SOCKET sockClient = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (sockClient == INVALID_SOCKET) {
        printf("create socket failed\n");
        return -1;
    }
    int nNetTimeout=50;//50ms  recv timeout
    setsockopt(sockClient,SOL_SOCKET,SO_RCVTIMEO,(char*)&nNetTimeout,sizeof(int));
    // 申明一个网络地址信息的结构体，保存服务器的地址信息
    sockaddr_in addr = { 0 };
    addr.sin_family = AF_INET;
    addr.sin_port = htons(PORT); // 37808
    addr.sin_addr.S_un.S_addr = inet_addr("112.74.89.58");
    int n = 500;
    int cnt = 0;
    char buf[] = "client test!";
    for(int i=1; i<=n; i++) {//发送数据
        Sleep(50);
        int dwSent = sendto(sockClient, buf, strlen(buf), 0, (SOCKADDR *)&addr, sizeof(SOCKADDR));
        if (dwSent <= 0)
        {
            printf("send failed\n");
            cnt++;
        } // else printf("Send msg: %s\n", buf);
    }
    printf("send %d msg.\n", n-cnt);
    char recvBuf[512];
    memset(recvBuf, 0, 512);
    sockaddr_in addrSever = { 0 };
    int nServerAddrLen=sizeof(sockaddr_in);
    int cnt2 = 0;
    for(int i=1;i<=n-cnt;i++){// 接收数据
        int dwRecv = recvfrom(sockClient, recvBuf, 512, 0, (SOCKADDR *)&addrSever,&nServerAddrLen);
        if(dwRecv<=0) {
            printf("timeout:\"%s\" %d\n", inet_ntoa(addrSever.sin_addr),i);
            cnt2++;
        } // else printf("Recv msg from server [%s] : %s\n", inet_ntoa(addrSever.sin_addr), recvBuf);
    }
    printf("Recv %d msg.\n", n-cnt-cnt2);
    closesocket(sockClient);//关闭SOCKET连接
    WSACleanup();//清理网络环境
    return 0;
}
```

【交实验报告】

上传实验报告：aceralon@qq.com　　截止日期（不迟于）：1 周之内

上传包括两个文件：

（1）小组实验报告。上传文件名格式：小组号_ 编程实验.pdf （由组长负责上传）

例如: 文件名“10_ Ftp 协议分析实验.pdf”表示第 10 组的 Ftp 协议分析实验报告

（2）小组成员实验体会。每个同学单独交一份只填写了实验体会的实验报告。只需填写自己的学号和姓名。

文件名格式：小组号_学号_姓名_ 编程实验.pdf （由组员自行上传）

例如: 文件名“10_05373092_张三_ Ftp 协议分析实验.pdf”表示第 10 组的 Ftp 协议分析实验报告。

注意：不要打包上传！