



# 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。

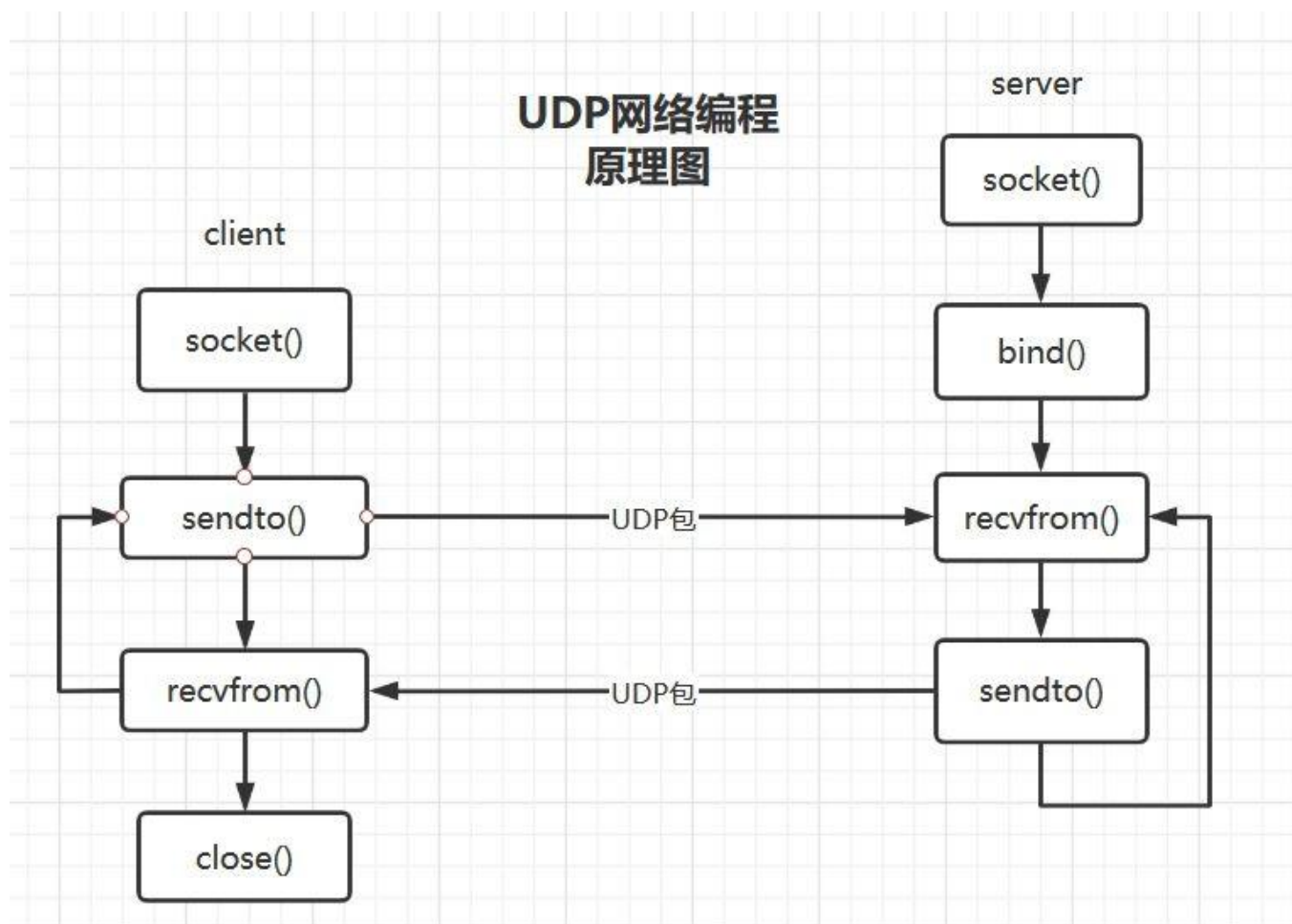
### 【实验要求】

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。

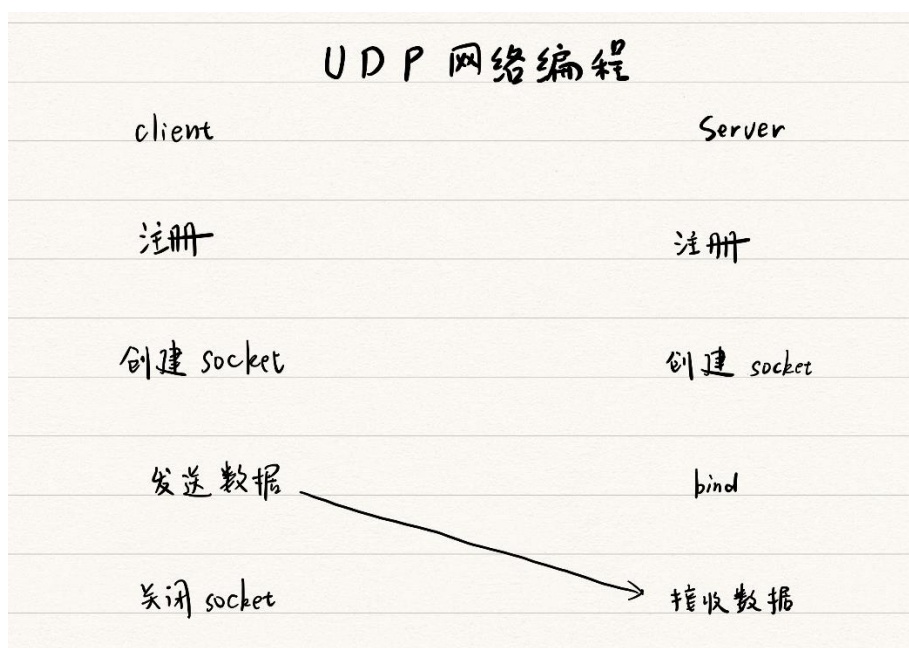
实验时，请运行 Wireshark 软件，对通信时的数据包进行跟踪分析。

### 【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 创建套接字，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 WINAPI 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;
```



# 计算机网络实验报告

```
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 WINAPI sendto(SOCKET s,const char *buf,int len,int flags,const struct 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。



# 计算机网络实验报告

/\*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 WINAPI 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, &client_len);
    if (last <= 0)
```



# 计算机网络实验报告

```
{
    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个数。

```
WINSOCK_API_LINKAGE int WINAPI recvfrom(SOCKET s,char *buf,int len,int flags,struct socka
ddr *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 = WSASStartup(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 WSASStartup
}
```

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_len);
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：

```
d:\C++ practise\practise\UDP1\bin\UDP_Cli.exe
Create socket successfully.
Packet 1 has been sent.
Packet 2 has been sent.
Packet 3 has been sent.
Packet 4 has been sent.
Packet 5 has been sent.
Packet 6 has been sent.
Packet 7 has been sent.
Packet 8 has been sent.
Packet 9 has been sent.
Packet 10 has been sent.
Packet 11 has been sent.
Packet 12 has been sent.
Packet 13 has been sent.
Packet 14 has been sent.
Packet 15 has been sent.
Packet 16 has been sent.
Packet 17 has been sent.
Packet 18 has been sent.
Packet 19 has been sent.
Packet 20 has been sent.
Packet 21 has been sent.
Packet 22 has been sent.
Packet 23 has been sent.
Packet 24 has been sent.
Packet 25 has been sent.
Packet 26 has been sent.
Packet 27 has been sent.
Packet 28 has been sent.
Packet 29 has been sent.
```

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：





# 计算机网络实验报告

```
C:\ 命令提示符 - UDP_Ser.exe

Recvfrom:Hello server! This is a packet. Data19629 Count:81
Recvfrom:Hello server! This is a packet. Data12623 Count:82
Recvfrom:Hello server! This is a packet. Data24084 Count:83
Recvfrom:Hello server! This is a packet. Data19954 Count:84
Recvfrom:Hello server! This is a packet. Data18756 Count:85
Recvfrom:Hello server! This is a packet. Data11840 Count:86
Recvfrom:Hello server! This is a packet. Data4966 Count:87
Recvfrom:Hello server! This is a packet. Data7376 Count:88
Recvfrom:Hello server! This is a packet. Data13931 Count:89
Recvfrom:Hello server! This is a packet. Data26308 Count:90
Recvfrom:Hello server! This is a packet. Data16944 Count:91
Recvfrom:Hello server! This is a packet. Data32439 Count:92
Recvfrom:Hello server! This is a packet. Data24626 Count:93
Recvfrom:Hello server! This is a packet. Data11323 Count:94
Recvfrom:Hello server! This is a packet. Data5537 Count:95
Recvfrom:Hello server! This is a packet. Data21538 Count:96
Recvfrom:Hello server! This is a packet. Data16118 Count:97
Recvfrom:Hello server! This is a packet. Data2082 Count:98
Recvfrom:Hello server! This is a packet. Data22929 Count:99
Recvfrom:Hello server! This is a packet. Data16541 Count:100
```

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。

```
C:\ 选择命令提示符 - ser.exe

D:\C++ practise\practise\UDP2\bin>ser.exe
Create socket successfully.
Bind successfully.
Count:1ello server! This is a packet. Data41
Recvfrom:Hello server! This is a packet. Data18467 Count:2
Recvfrom:Hello server! This is a packet. Data63347 Count:3
Recvfrom:Hello server! This is a packet. Data26500 Count:4
Recvfrom:Hello server! This is a packet. Data19169 Count:5
Recvfrom:Hello server! This is a packet. Data15724 Count:6
Recvfrom:Hello server! This is a packet. Data11478 Count:7
Recvfrom:Hello server! This is a packet. Data29358 Count:8
Recvfrom:Hello server! This is a packet. Data26962 Count:9
Recvfrom:Hello server! This is a packet. Data24464 Count:10
Recvfrom:Hello server! This is a packet. Data57054 Count:11
Recvfrom:Hello server! This is a packet. Data28145 Count:12
Recvfrom:Hello server! This is a packet. Data23281 Count:13
Recvfrom:Hello server! This is a packet. Data16827 Count:14
Recvfrom:Hello server! This is a packet. Data99617 Count:15
Recvfrom:Hello server! This is a packet. Data49117 Count:16
Recvfrom:Hello server! This is a packet. Data29957 Count:17
Recvfrom:Hello server! This is a packet. Data11942 Count:18
Recvfrom:Hello server! This is a packet. Data48272 Count:19
Recvfrom:Hello server! This is a packet. Data54362 Count:20
Recvfrom:Hello server! This is a packet. Data32391 Count:21
Recvfrom:Hello server! This is a packet. Data14604 Count:22
Recvfrom:Hello server! This is a packet. Data39024 Count:23
Recvfrom:Hello server! This is a packet. Data15324 Count:24
Recvfrom:Hello server! This is a packet. Data29224 Count:25
```



# 计算机网络实验报告

最终成功接收到 100 个数据包，无丢包的情况发生：

```
选择命令提示符 - ser.exe
Recvfrom:Hello server! This is a packet. Data24370 Count:75
Recvfrom:Hello server! This is a packet. Data15350 Count:76
Recvfrom:Hello server! This is a packet. Data15006 Count:77
Recvfrom:Hello server! This is a packet. Data31101 Count:78
Recvfrom:Hello server! This is a packet. Data24393 Count:79
Recvfrom:Hello server! This is a packet. Data35483 Count:80
Recvfrom:Hello server! This is a packet. Data19629 Count:81
Recvfrom:Hello server! This is a packet. Data12623 Count:82
Recvfrom:Hello server! This is a packet. Data24084 Count:83
Recvfrom:Hello server! This is a packet. Data19954 Count:84
Recvfrom:Hello server! This is a packet. Data18756 Count:85
Recvfrom:Hello server! This is a packet. Data11840 Count:86
Recvfrom:Hello server! This is a packet. Data49660 Count:87
Recvfrom:Hello server! This is a packet. Data73760 Count:88
Recvfrom:Hello server! This is a packet. Data13931 Count:89
Recvfrom:Hello server! This is a packet. Data26308 Count:90
Recvfrom:Hello server! This is a packet. Data16944 Count:91
Recvfrom:Hello server! This is a packet. Data32439 Count:92
Recvfrom:Hello server! This is a packet. Data24626 Count:93
Recvfrom:Hello server! This is a packet. Data11323 Count:94
Recvfrom:Hello server! This is a packet. Data55373 Count:95
Recvfrom:Hello server! This is a packet. Data21538 Count:96
Recvfrom:Hello server! This is a packet. Data16118 Count:97
Recvfrom:Hello server! This is a packet. Data20828 Count:98
Recvfrom:Hello server! This is a packet. Data22929 Count:99
Recvfrom:Hello server! This is a packet. Data16541 Count:100
```

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：



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 Unreachable
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 Unreachable
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 Unreachable
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 Unreachable
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 Unreachable
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 Unreachable
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 Unreachable
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 Unreachable
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 Unreachable
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 Unreachable
356	36.707400	127.0.0.1	127.0.0.1	UDP	75	65310 → 6666
357	36.707410	127.0.0.1	127.0.0.1	ICMP	103	Destination Unreachable



# 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
172	36.635634	127.0.0.1	127.0.0.1	UDP	75	65310 → 6666
173	36.635657	127.0.0.1	127.0.0.1	ICMP	103	Destination u
174	36.635952	127.0.0.1	127.0.0.1	UDP	74	65310 → 6666
175	36.635964	127.0.0.1	127.0.0.1	ICMP	102	Destination u
176	36.636183	127.0.0.1	127.0.0.1	UDP	75	65310 → 6666
177	36.636193	127.0.0.1	127.0.0.1	ICMP	103	Destination u
178	36.636411	127.0.0.1	127.0.0.1	UDP	75	65310 → 6666
179	36.636421	127.0.0.1	127.0.0.1	ICMP	103	Destination u
180	36.636714	127.0.0.1	127.0.0.1	UDP	75	65310 → 6666
181	36.636731	127.0.0.1	127.0.0.1	ICMP	103	Destination u
182	36.637395	127.0.0.1	127.0.0.1	UDP	75	65310 → 6666
183	36.637410	127.0.0.1	127.0.0.1	ICMP	103	Destination u
184	36.638369	127.0.0.1	127.0.0.1	UDP	75	65310 → 6666

> Frame 170: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface \Device\NPF{...}

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> User Datagram Protocol, Src Port: 65310, Dst Port: 6666

Offset	Hex	ASCII
0000	02 00 00 00 45 00 00 44 6d f4 00 00 80 11 00 00	.....E..D m.....
0010	7f 00 00 01 7f 00 00 01 ff 1e 1a 0a 00 30 ad 6c	.....0.1
0020	48 65 6c 6c 6f 20 73 65 72 76 65 72 21 20 54 68	Hello se rver! Th
0030	69 73 20 69 73 20 61 20 70 61 63 6b 65 74 2e 20	is is a packet.
0040	44 61 74 61 a3 ba 34 31	Data..41

User Datagram Protocol: Protocol | 分组: 454 · 已显示: 214 (47.1%) | 配置: Default

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

```
d:\C++ practise\practise\UDP1\bin\cli.exe
Create socket successfully.
Bind successfully.
Packet 1 has been sent.
Recvfrom:Hello server! There is a packet.    Count:1
Packet 2 has been sent.
Recvfrom:Hello server! There is a packet.    Count:2
Packet 3 has been sent.
Recvfrom:Hello server! There is a packet.    Count:3
Packet 4 has been sent.
Recvfrom:Hello server! There is a packet.    Count:4
Packet 5 has been sent.
Recvfrom:Hello server! There is a packet.    Count:5
Packet 6 has been sent.
Recvfrom:Hello server! There is a packet.    Count:6
Packet 7 has been sent.
Recvfrom:Hello server! There is a packet.    Count:7
Packet 8 has been sent.
Recvfrom:Hello server! There is a packet.    Count:8
Packet 9 has been sent.
Recvfrom:Hello server! There is a packet.    Count:9
Packet 10 has been sent.
Recvfrom:Hello server! There is a packet.    Count:10
Packet 11 has been sent.
Recvfrom:Hello server! There is a packet.    Count:11
Packet 12 has been sent.
Recvfrom:Hello server! There is a packet.    Count:12
Packet 13 has been sent.
Recvfrom:Hello server! There is a packet.    Count:13
Packet 14 has been sent.
Recvfrom:Hello server! There is a packet.    Count:14
Packet 15 has been sent.
Recvfrom:Hello server! There is a packet.    Count:15
Packet 16 has been sent.
Recvfrom:Hello server! There is a packet.    Count:16
```

最终成功发送 100 个数据包，成功接收到发回的 100 个数据包。

```
Recvfrom:Hello server! There is a packet.    Count:87
Packet 88 has been sent.
Recvfrom:Hello server! There is a packet.    Count:88
Packet 89 has been sent.
Recvfrom:Hello server! There is a packet.    Count:89
Packet 90 has been sent.
Recvfrom:Hello server! There is a packet.    Count:90
Packet 91 has been sent.
Recvfrom:Hello server! There is a packet.    Count:91
Packet 92 has been sent.
Recvfrom:Hello server! There is a packet.    Count:92
Packet 93 has been sent.
Recvfrom:Hello server! There is a packet.    Count:93
Packet 94 has been sent.
Recvfrom:Hello server! There is a packet.    Count:94
Packet 95 has been sent.
Recvfrom:Hello server! There is a packet.    Count:95
Packet 96 has been sent.
Recvfrom:Hello server! There is a packet.    Count:96
Packet 97 has been sent.
Recvfrom:Hello server! There is a packet.    Count:97
Packet 98 has been sent.
Recvfrom:Hello server! There is a packet.    Count:98
Packet 99 has been sent.
Recvfrom:Hello server! There is a packet.    Count:99
Packet 100 has been sent.
Recvfrom:Hello server! There is a packet.    Count:100
Press any key to continue . . .
```



# 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](https://blog.csdn.net/qq_31837203/article/details/112121363)