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。**

院系	数据科学与计算机学院	班 级	周一班	组长	曾妮
学号	16340011	16340013		16340041	
学生	曾妮	曾翔		陈亚楠	

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。**

编程实验

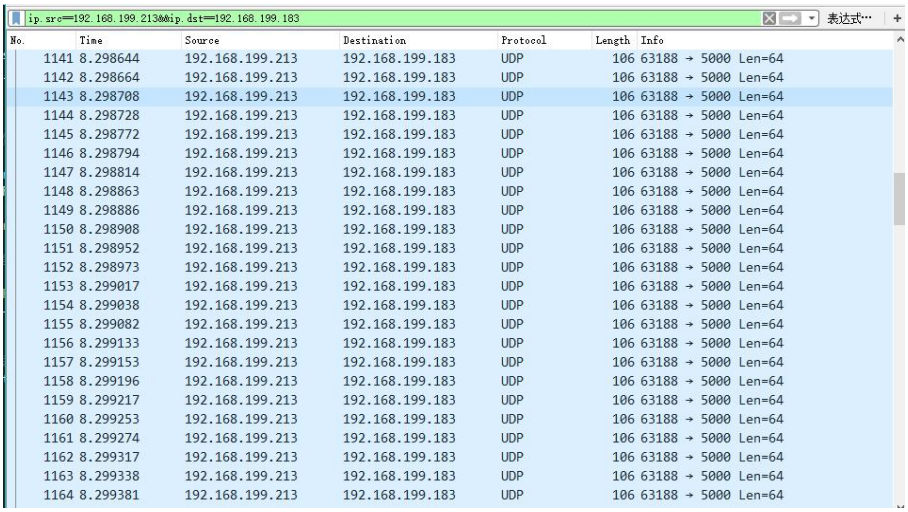
【实验内容】

（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）。

本地局域网下两台机器分别运行客户端与服务端程序，服务端截图如下：

```
Recv From Client:UDP Hello World !
91
Recv From Client:UDP Hello World !
92
Recv From Client:UDP Hello World !
93
Recv From Client:UDP Hello World !
94
Recv From Client:UDP Hello World !
95
Recv From Client:UDP Hello World !
96
Recv From Client:UDP Hello World !
97
Recv From Client:UDP Hello World !
98
Recv From Client:UDP Hello World !
99
Recv From Client:UDP Hello World !
100
```

Wireshark 捕获：



ip.src==192.168.199.213&&ip.dst==192.168.199.183

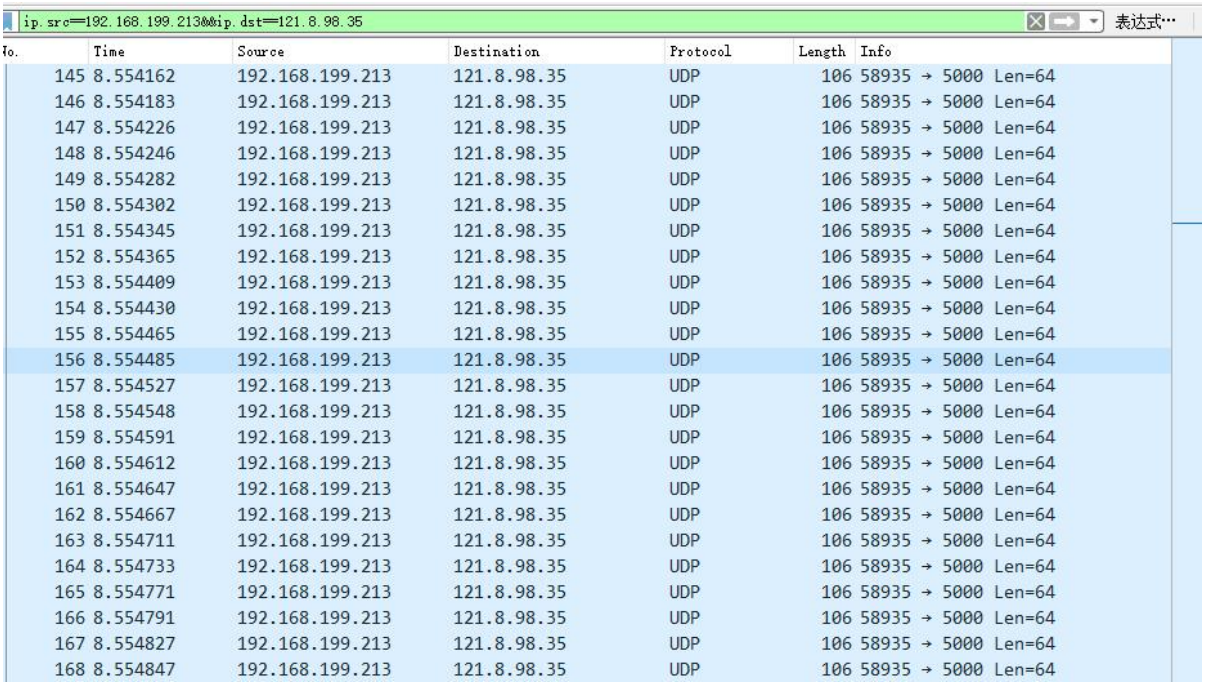
No.	Time	Source	Destination	Protocol	Length	Info
1141	8.298644	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1142	8.298664	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1143	8.298708	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1144	8.298728	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1145	8.298772	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1146	8.298794	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1147	8.298814	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1148	8.298863	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1149	8.298886	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1150	8.298908	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1151	8.298952	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1152	8.298973	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1153	8.299017	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1154	8.299038	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1155	8.299082	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1156	8.299133	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1157	8.299153	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1158	8.299196	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1159	8.299217	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1160	8.299253	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1161	8.299274	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1162	8.299317	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1163	8.299338	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64
1164	8.299381	192.168.199.213	192.168.199.183	UDP	106	63188 → 5000 Len=64

可以看到的是，在本地局域网下，服务端完整的接收到了 100 个数据包，并没有丢包。

至于外网，由于公网 ip 有限，我们尝试获取本机的公网 ip 并发送数据包，wireshark 捕获到了发送出去的数据包，

但实际上，服务端并没有收到这些数据包，至于这些数据包到底去了哪，这是一个问题。

计算机网络实验报告



ip.src==192.168.199.213&&ip.dst==121.8.98.35 表达式…

No.	Time	Source	Destination	Protocol	Length	Info
145	8.554162	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
146	8.554183	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
147	8.554226	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
148	8.554246	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
149	8.554282	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
150	8.554302	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
151	8.554345	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
152	8.554365	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
153	8.554409	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
154	8.554430	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
155	8.554465	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
156	8.554485	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
157	8.554527	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
158	8.554548	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
159	8.554591	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
160	8.554612	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
161	8.554647	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
162	8.554667	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
163	8.554711	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
164	8.554733	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
165	8.554771	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
166	8.554791	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
167	8.554827	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64
168	8.554847	192.168.199.213	121.8.98.35	UDP	106	58935 → 5000 Len=64

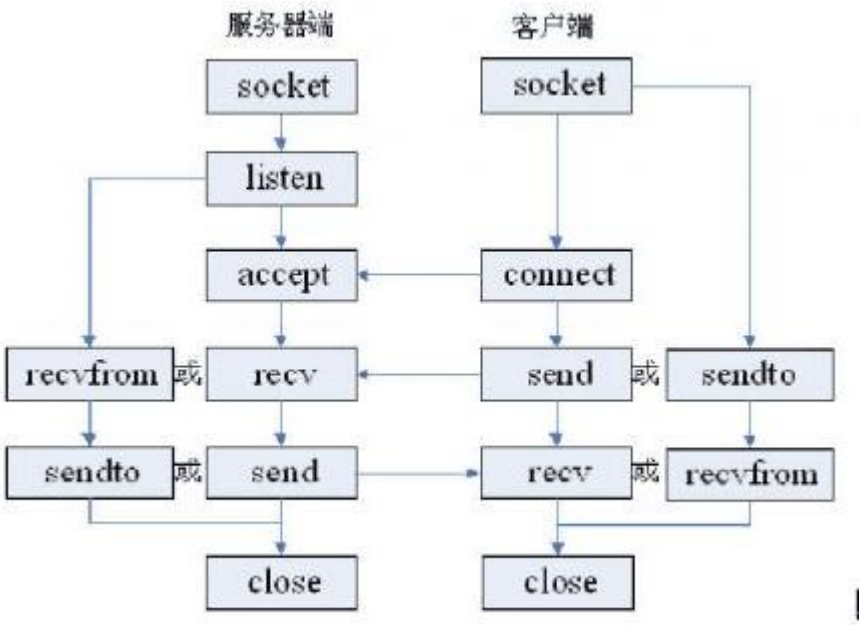
【实验思考】

1）遇到的问题有很多，首先就是代码的问题，由于是第一次接触这种代码，所以不知道从何下手，在网上找到一些参考资料之后终于是成功的跑了起来，但是其中的中文不知为何变成了乱码，找了半天没有找到解决方案，然后把需要输出的地方换成了英文，此时，程序虽然是能跑起来了但是离要求还差一段距离，在这份代码上东改改西改改，最后虽然能通过编译，但是没有任何的输出，此次的代码以失败告终。后来重新在网上找资料，然后自己修改，总算是跑出了应该有的效果。此时进度来到了抓包，由于服务端与客户端都是在本电脑上跑，所以在如何抓包上又是弄了很久，最后找到了方法：1．以管理员身份打开命令提示符 2．输入 route add 本机 ip mask 255.255.255.255 网关 ip 如果不知道本机 ip 和网关 ip，可以在命令行输入 ipconfig 查看 3．将我们程序里面的 localhost 或者 127.0.0.1 替换成本机 ip4．使用 WireShark 即可抓到本地包

注:在测试完之后，使用 route delete 本机 ip mask 255.255.255.255 网关 ip 来删除我们上面的更改，不然我们本机的所有报文都会先经过网卡再回到本机，会比较消耗性能。虽然是找到了方法，也成功的抓去了包，但是服务窗口却没有任何的输出，所以也只能作罢，然后就是在不同电脑之间跑两个程序，得到了在同一局域网下 100%的抓包，以及不同局域网下，一个都没抓到的结果。眼看时限将至，最后只能如此。

2）程序详细的流程图：client：初始化套接字动态库、创建套接字、接受数据、设置服务器套接字地址、向服务器发送数据、接收反馈、关闭套接字、终止套接字动态库的使用。

Server：初始化套接字动态库、创建套接字、设置服务器地址端口、绑定套接字、从客户端接收数据、打印数据、向客户端发送数据、关闭套接字、终止套接字动态库的使用。



关键函数：

socket 函数:为了执行网络输入输出,一个进程必须做的第一件事就是调用 socket 函数获得一个文件描述符。 int socket(int family,int type,int protocol)；返回：非负描述字成功，-1 失败

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。

bind 函数：为套接口分配一个本地 IP 和协议端口，对于网际协议，协议地址是 32 位 IPv4 地址或 128 位 IPv6 地址与 16 位的 TCP 或 UDP 端口号的组合；如指定端口为 0，调用 bind 时内核将选择一个临时端口，如果指定

一个通配 IP 地址，则要等到建立连接后内核才选择一个本地 IP 地址。 int bind(int sockfd, const struct sockaddr * server, socklen_t addrlen)；返回：0 成功，-1 失败

第一个参数是 socket 函数返回的套接口描述字；第二和第第三个参数分别是一个指向特定于协议的地址结构的指针和该地址结构的长度。

recvfrom 函数(经 socket 接收数据):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。

本函数用于从（已连接）套接口上接收数据，并捕获数据发送源的地址。

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。ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr * to, int addrlen);

返回发送数据的长度成功，-1 失败

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。

3）客户端程序需要使用到 WSAStartup()用于加载 ws2_32.dll，Socket()用于创建套接字，Connect()用于发送连接请求，Send()用于发送，Recv()用于接收，Closesocket()关闭套接字，WSACleanup()清理加载的 dll；

服务端程序同样需要使用到 WSAStartup()、Socket()、Recv()、Send()、Closesocket()、WSACleanup()，除此之外，还是使用到了 Bind()用于绑定 IP 地址和端口号，Listen()用于监听，Accept()用于接收请求连接队伍里的一个连接请求。

4）一个 const struct sockaddr *指针，指向要绑定或者连接的给 sockfd 的协议地址。这个地址结构根据地址创建 socket 时的地址协议族的不同而不同，如 ipv4 对应的是：

```
struct    sockaddr_in {
        short           sin_family;        //AF_INET 地址族
        unsigned short    sin_port;        //使用的端口，2 字节
        struct    in_addr sin_addr;        //本机 IP 地址结构体，4 字节
        char            sin_zero[8];       //预留，8 字节
}

/* Internet address. */
struct in_addr {
    uint32_t        s_addr;     /* address in network byte order */
};
```

ipv6 对应的是：

```
struct sockaddr_in6 {
    sa_family_t     sin6_family;    /* AF_INET6 */
    in_port_t       sin6_port;      /* port number */
    uint32_t        sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr;      /* IPv6 address */
    uint32_t        sin6_scope_id; /* Scope ID (new in 2.4) */
};

struct in6_addr {
    unsigned char   s6_addr[16];    /* IPv6 address */
};
```

Unix 域对应的是：

```
#define UNIX_PATH_MAX    108
```

```
struct sockaddr_un {
        sa_family_t sun_family;                 /* AF_UNIX */
        char            sun_path[UNIX_PATH_MAX];    /* pathname */
};
```

本次实验中使用的是 ipv4 的版本，故而在 connect 的时候，如下填写：

```
servAddr.sin_family = AF_INET;
servAddr.sin_addr.S_un.S_addr = inet_addr("120.236.174.153");  //这里填写的服务端的 ip 地址
servAddr.sin_port = htons(5000);  //端口号
```

5）顾名思义，从字面上看，面向连接的客户端在建立 socket 是需要与服务器连接，所以他其中的 type 应该取 SOCK_STREAM，而面向非连接的 socket 则不需要，故 type 应该为 SOCK_DGRAM

6）面向连接的客户端在首发数据时是以比特流形式，而面向非连接的是以数据包形式；通过接收方回传一个确认传输完毕的 udp 包来确认数据发送完成

7）面向连接的通信就是具有顺序、可靠、双向等优点；而无连接的通信不可靠的传输，但却传输十分迅速，所以在视频网络电话是适用于无连接的通信，而当传输重要文件是则应选择面向连接的通信。

8）使用非阻塞，阻塞就是，你发一个消息，知道确认对方接受到你才会发下一条消息，而非阻塞是你不管对方有没有接收到消息，你都可以发完你想法的消息。

（2）注意实验时简述设计思路。

通过书中所给和查找资料了解了 UDP 编程所需要的函数，设计在客户端循环发送数据包，使用 wireshark 进行抓包，客户端循环发送结束后关闭套接字，服务器端接受结束后关闭套接字。

（3）引起 UDP 丢包的可能原因是什么？

可能是发送端发送包频率过快，如果要发送的数据过多或者过大，那么在缓冲区满的那个瞬间要发送的报文就

很有可能被丢失；也可能是 Socket 未开始监听导致接收方丢包。

本次实验完成后，请根据组员在实验中的贡献，请实事求是，自评在实验中应得的分数。（按百分制）

学号	学生	自评分
16340011	曾妮	100
16340013	曾翔	100
16340041	陈亚楠	100