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	计算机学院	班 级	计算机科学与技术 1 班
实验	基于 UDP 丢包统计程序设计		
学生	邓栩瀛	学号	21307035

编程实验

【实验内容】

实验步骤（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），回答实验问题及实验思考。

实验目的：选择一个操作系统环境，编写 UDP/IP 通信程序，完成一定的通信功能

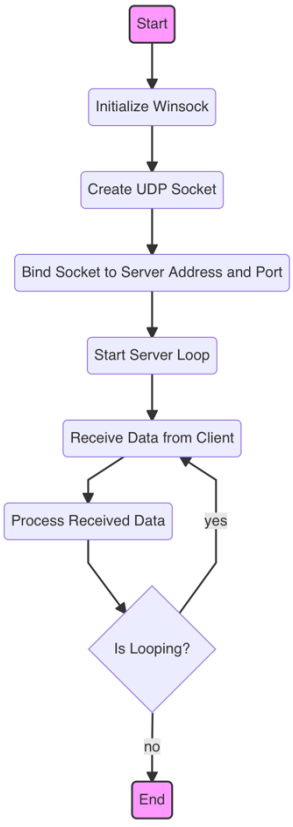
实验要求：在发送 UDP 数据包时做一个循环，连续发送 100 个数据包；在接收端统计丢失的数据包。实验时，请运行 Wireshark 软件，对通信时的数据包进行跟踪分析。

实验思考：

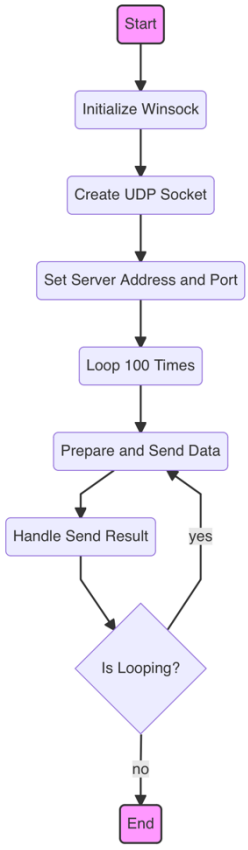
（1）给出程序详细的流程图和对程序关键函数的详细说明

流程图

server.cpp



client.cpp



关键函数：

server.cpp

①socket()函数用于创建套接字，接受三个参数：地址族（AF_INET 表示 IPv4）、套接字类型（SOCK_DGRAM 表示 UDP 套接字）和协议（0 表示默认协议）。在代码中，创建了一个 UDP 套接字，并将其赋值给变量 udpSocket。

```
SOCKET udpSocket = socket(AF_INET, SOCK_DGRAM, 0);
```

②recvfrom()函数用于从套接字接收数据，接受六个参数：套接字、接收缓冲区、缓冲区大小、标志（通常为 0）、客户端地址结构体指针和客户端地址长度。在代码中，使用 recvfrom()函数从 udpSocket 接收数据，并存储在 recvData 缓冲区中。然后根据接收结果判断是否接收成功，并将接收到的数据信息打印出来。

```
int recvResult = recvfrom(udpSocket, recvData, sizeof(recvData), 0, (struct sockaddr*)&clientAddr,
&clientAddrLen);
```

client.cpp

①socket()函数用于创建套接字，接受三个参数：地址族（AF_INET 表示 IPv4）、套接字类型（SOCK_DGRAM 表示 UDP 套接字）和协议（0 表示默认协议）。在代码中，创建了一个 UDP 套接字，并将其赋值给变量 udpSocket。

```
SOCKET udpSocket = socket(AF_INET, SOCK_DGRAM, 0);
```

②sendto()函数用于向套接字发送数据，接受六个参数：套接字、发送缓冲区、缓冲区大小、标志（通常为 0）、目标地址结构体指针和目标地址长度。在代码中，使用 sendto()函数将数据发送到 serverAddr 指定的服务器地址。在循环中，通过 sprintf()函数将要发送的数据存储在 sendData 缓冲区中，然后使用 sendto()发送数据。如果发送失败，会打印错误信息。

```
int sendResult = sendto(udpSocket, sendData, strlen(sendData), 0, (struct sockaddr*)&serverAddr,
sizeof(serverAddr));
```

（2）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？

服务器程序

①socket()：创建套接字，指定地址族（通常为 AF_INET，表示 IPv4）、套接字类型（如 SOCK_STREAM 表示 TCP 套接字，SOCK_DGRAM 表示 UDP 套接字）和协议（通常为 0，表示使用默认协议）。

②bind()：将套接字与特定的 IP 地址和端口号绑定在一起，以便监听客户端连接请求或接收客户端发送的数据。

③listen()：将套接字标记为监听状态，开始监听客户端连接请求。

④accept()：接受客户端连接请求，创建一个新的套接字用于与客户端进行通信。

⑤recvfrom()：从客户端接收数据，用于接收 TCP 或 UDP 传输的数据。

⑥sendto()：向客户端发送数据，用于发送 TCP 或 UDP 传输的数据。

⑦closesocket()：关闭套接字，释放资源。

客户端程序：

①socket()：创建套接字，指定地址族、套接字类型和协议。

②connect()：与服务器建立连接，指定服务器的 IP 地址和端口号。

③sendto()：向服务器发送数据，用于发送 TCP 或 UDP 传输的数据。

④recvfrom()：从服务器接收数据，用于接收 TCP 或 UDP 传输的数据。

⑤closesocket()：关闭套接字，释放资源。

（3）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 {
    short sin_family; // 地址族，如 AF_INET 表示 IPv4
    u_short sin_port; // 端口号，使用网络字节顺序
    struct in_addr sin_addr; // IP 地址
    char sin_zero[8]; // 填充字节，通常设置为 0
};
```

sin_family：指定地址族，常用的是 AF_INET，表示 IPv4 地址。

sin_port：指定端口号，使用网络字节顺序表示，可以通过 htons()函数将主机字节顺序转换为网络字节顺序。

sin_addr：存储 IP 地址，是一个 struct in_addr 类型的结构体，其中的字段 s_addr 是一个 unsigned long 类型，用于存储 32 位的 IPv4 地址，可以通过 inet_addr()函数将点分十进制形式的 IP 地址转换为 unsigned long 类型。

sin_zero：用于填充字节，通常设置为 0。

例如：在本实验中，绑定 IPv4 地址 172.16.11.2 的端口号 12345，可以使用以下代码

```
    struct sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(12345); // 服务器的端口
    serverAddr.sin_addr.s_addr = inet_addr("172.16.11.2"); // 服务器的 IP 地址
    memset(serverAddr.sin_zero, 0, sizeof(serverAddr.sin_zero));
```

serverAddr 是一个 struct sockaddr_in 类型的结构体，通过逐个赋值来设置地址信息。sin_family 设置为 AF_INET 表示 IPv4 地址族，sin_port 设置为 htons(12345)将端口号 12345 转换为网络字节顺序，sin_addr.s_addr 使用 inet_addr("172.16.11.2")将字符串形式的 IPv4 地址转换为 unsigned long 类型，并赋值给 sin_addr。sin_zero 通过 memset()函数将其填充为 0。

```
// 连接服务器
int connectResult = connect(sock, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
```

将 serverAddr 作为参数传递给 connect()函数来连接服务器。

（4）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	面向非连接的客户端

建立连接	使用 connect()函数建立与服务器的连接	不需要建立连接，直接发送数据到目标地址
通信	使用 send()和 recv()函数进行数据传输	使用 sendto()和 recvfrom()函数进行数据传输
可靠性	提供可靠的数据传输和错误处理机制	无法保证数据的可靠性和顺序性
延迟	引入一定的开销和延迟	传输速度快，无额外延迟
应用场景	需要可靠的数据传输、顺序性和错误处理的应用	对传输速度要求较高，对数据可靠性要求较低的应用

（5）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？

	面向连接的客户端	**面向非连接的客户端**
数据传输方式	使用 send()和 recv()函数进行数据传输	使用 sendto()和 recvfrom()函数进行数据传输
数据传输结束判断	可以使用固定长度、特殊字符/标记或协议协商等机制	需要应用层协议中定义判断数据传输结束的机制
数据的可靠性和顺序性	提供可靠性和顺序性保证	无法保证数据的可靠性和顺序性

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

	面向连接通信	无连接通信
优点	可靠性：提供可靠的数据传输和错误处理机制	传输速度快：无需建立连接，数据直接发送到目标地址
	顺序性：保证数据按照发送顺序接收	简单快速：不需要建立连接步骤
	错误处理：提供错误检测和恢复机制	灵活性：适用于实时、快速的数据传输
	流控制：通过滑动窗口等机制控制数据流	
缺点	延迟：建立连接和断开连接的开销	可靠性：无法保证数据的可靠性和顺序性
	开销：需要维护连接状态和缓冲区	无连接性：无法进行错误检测和恢复
	资源消耗：占用较多的服务器资源	
适合使用场景	需要可靠性、顺序性和错误处理的应用	实时性要求高的应用，如音视频传输
	大量数据传输或长时间通信的应用	广播或多播通信
	对延迟要求相对较低的应用	简单的请求-响应通信

（7）实验过程中使用 Socket 时是工作在阻塞方式还是非阻塞方式？通过网络检索阐述着两种操作方式的不同。

本实验使用的是阻塞方式

	阻塞方式	非阻塞方式
工作方式	阻塞主线程，直到操作完成或发生错误	立即返回结果，无论操作是否完成
发送操作	如果发送缓冲区已满，操作阻塞直到有足够空间可用	立即返回结果，表示操作当前不可用
接收操作	如果接收缓冲区为空，操作阻塞直到有新数据可用	立即返回结果，表示操作当前不可用
主线程执行	被阻塞，无法执行其他任务	可以继续执行其他任务
操作状态检查方式	不需要额外的操作状态检查	需要轮询或事件驱动方式进行操作状态检查

（8）说明在实验过程中遇到的问题和解决方法。

一开始直接 g++编译报错，报错信息说明部分库的确实，经研究，发现编译过程中没有添加动态库编译 g++的编译命令加上`-lwsock32` 即可编译成功

实验步骤（2）注意实验时简述设计思路。

①确保实验所需的环境：

- 在发送端和接收端均安装并配置好 Wireshark 软件，以便进行数据包跟踪和分析。
- 确保发送端和接收端之间建立了网络连接，并且可以进行 UDP 通信。

② 在发送端（Client）编写程序：

- 编写一个 UDP 发送程序，用于发送数据包。
- 设置一个循环，连续发送 100 个数据包。

```
    for (int i = 0; i < 100; i++) {
        sprintf(sendData, "Message %d", i + 1);

        int sendResult = sendto(udpSocket, sendData, strlen(sendData), 0, (struct sockaddr*)&serverAddr,
sizeof(serverAddr));
        if (sendResult == SOCKET_ERROR) {
            fprintf(stderr, "sendto failed: %d\n", WSAGetLastError());
            break;
        }
        printf("Sent: %s\n", sendData);
        Sleep(100);
    }
```

- 每次发送一个数据包时，可以在数据包中包含一些标识信息（如序列号），以便在接收端进行统计和识别。

③在接收端（Server）编写程序：

- 编写一个 UDP 接收程序，用于接收数据包。

- 在程序中设置一个循环，接收从发送端发送的数据包。

```
    while (1) {
        struct sockaddr_in clientAddr;
        int clientAddrLen = sizeof(clientAddr);
        int recvResult = recvfrom(udpSocket, recvData, sizeof(recvData), 0, (struct sockaddr*)&clientAddr,
&clientAddrLen);
        if (recvResult == SOCKET_ERROR) {
            fprintf(stderr, "recvfrom failed: %d\n", WSAGetLastError());
            break;
        }
        recvData[recvResult] = '\0';
        printf("Received data from %s:%d: %s\n", inet_ntoa(clientAddr.sin_addr),
ntohs(clientAddr.sin_port), recvData);
    }
```

- 在接收到每个数据包时，进行统计，判断是否有数据包丢失，可以根据数据包中的标识信息进行判断。

④使用 Wireshark 进行数据包跟踪：

- 在发送端和接收端的计算机上同时运行 Wireshark
- 设置好过滤规则，捕获 UDP 通信的数据包
- 开始捕获数据包，并启动发送端和接收端的程序进行通信
- Wireshark 将会记录并显示发送和接收的数据包，可以检查是否有数据包丢失以及其他相关的网络信息

实验步骤（3）引起 UDP 丢包的可能原因是什么？

本实验并没有出现 UDP 丢包的情况，如果出现 UDP 丢包，则有可能为以下几种情况：

a.网络拥塞：当网络中的流量超过其处理能力时，会导致数据包丢失。因为 UDP 不提供拥塞控制机制，因此在拥塞时容易丢包。

b.网络错误：网络中的错误，如传输介质故障、路由器故障或网络设备故障等，都有可能导致 UDP 数据包丢失。

c.数据包过大：UDP 协议没有分包和重组的机制，如果发送的 UDP 数据包超过网络的 MTU，则数据包可能被丢弃。

d.不可靠的接收方：如果接收方的应用程序或设备无法及时处理接收到的 UDP 数据包，或者应用程序没有正确实现丢包处理机制，也可能导致丢包。

e.防火墙配置：防火墙的配置可能会导致 UDP 数据包被过滤或丢弃，某些防火墙可能会限制特定端口或协议的传输。

计算机网络实验报告

实验结果展示

服务器端：

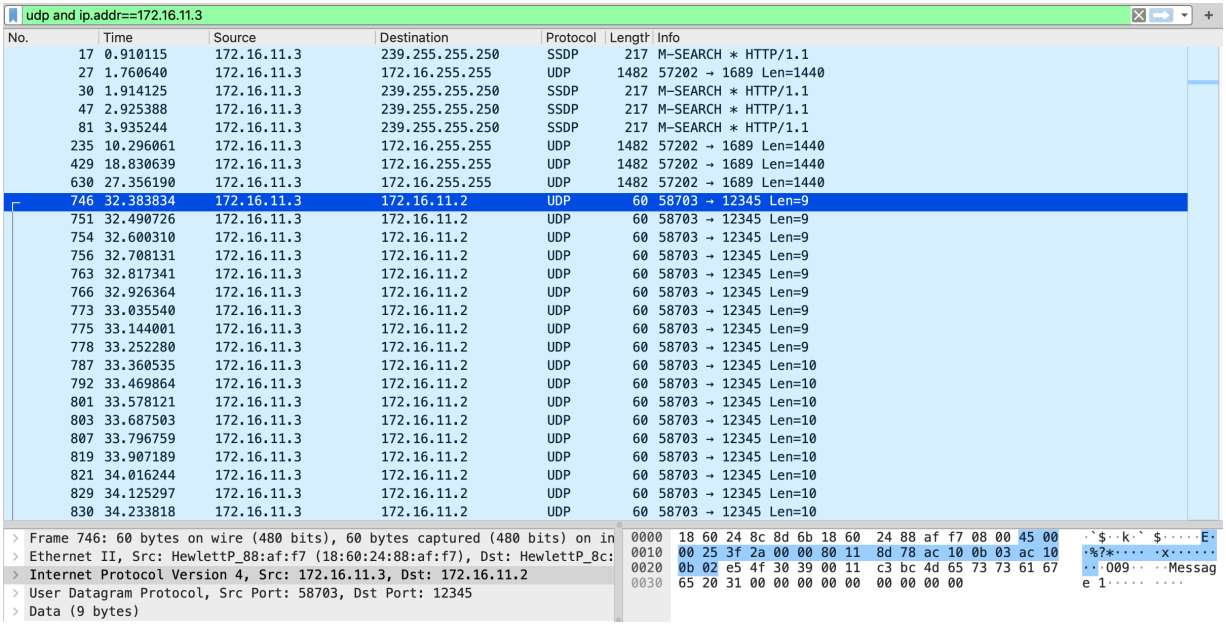
```
Server is running...
Received data from 172.16.11.3:58703: Message 1
Received data from 172.16.11.3:58703: Message 2
Received data from 172.16.11.3:58703: Message 3
Received data from 172.16.11.3:58703: Message 4
Received data from 172.16.11.3:58703: Message 5
Received data from 172.16.11.3:58703: Message 6
Received data from 172.16.11.3:58703: Message 7
Received data from 172.16.11.3:58703: Message 8
Received data from 172.16.11.3:58703: Message 9
Received data from 172.16.11.3:58703: Message 10
Received data from 172.16.11.3:58703: Message 11
Received data from 172.16.11.3:58703: Message 12
Received data from 172.16.11.3:58703: Message 13
Received data from 172.16.11.3:58703: Message 14
Received data from 172.16.11.3:58703: Message 15
Received data from 172.16.11.3:58703: Message 16
Received data from 172.16.11.3:58703: Message 17
Received data from 172.16.11.3:58703: Message 18
Received data from 172.16.11.3:58703: Message 19
Received data from 172.16.11.3:58703: Message 20
Received data from 172.16.11.3:58703: Message 21
Received data from 172.16.11.3:58703: Message 22
Received data from 172.16.11.3:58703: Message 23
Received data from 172.16.11.3:58703: Message 24
Received data from 172.16.11.3:58703: Message 25
Received data from 172.16.11.3:58703: Message 26
Received data from 172.16.11.3:58703: Message 27
Received data from 172.16.11.3:58703: Message 28
Received data from 172.16.11.3:58703: Message 29
Received data from 172.16.11.3:58703: Message 30
Received data from 172.16.11.3:58703: Message 31
Received data from 172.16.11.3:58703: Message 32
Received data from 172.16.11.3:58703: Message 33
Received data from 172.16.11.3:58703: Message 34
Received data from 172.16.11.3:58703: Message 35
Received data from 172.16.11.3:58703: Message 36
Received data from 172.16.11.3:58703: Message 37
Received data from 172.16.11.3:58703: Message 38
Received data from 172.16.11.3:58703: Message 39
Received data from 172.16.11.3:58703: Message 40
Received data from 172.16.11.3:58703: Message 41
Received data from 172.16.11.3:58703: Message 42
Received data from 172.16.11.3:58703: Message 43
Received data from 172.16.11.3:58703: Message 44
Received data from 172.16.11.3:58703: Message 45
Received data from 172.16.11.3:58703: Message 46
Received data from 172.16.11.3:58703: Message 47
Received data from 172.16.11.3:58703: Message 48
Received data from 172.16.11.3:58703: Message 49
Received data from 172.16.11.3:58703: Message 50
Received data from 172.16.11.3:58703: Message 51
Received data from 172.16.11.3:58703: Message 52
Received data from 172.16.11.3:58703: Message 53
Received data from 172.16.11.3:58703: Message 54
Received data from 172.16.11.3:58703: Message 55
Received data from 172.16.11.3:58703: Message 56
Received data from 172.16.11.3:58703: Message 57
Received data from 172.16.11.3:58703: Message 58
Received data from 172.16.11.3:58703: Message 59
Received data from 172.16.11.3:58703: Message 60
Received data from 172.16.11.3:58703: Message 61
Received data from 172.16.11.3:58703: Message 62
Received data from 172.16.11.3:58703: Message 63
Received data from 172.16.11.3:58703: Message 64
Received data from 172.16.11.3:58703: Message 65
Received data from 172.16.11.3:58703: Message 66
Received data from 172.16.11.3:58703: Message 67
Received data from 172.16.11.3:58703: Message 68
Received data from 172.16.11.3:58703: Message 69
Received data from 172.16.11.3:58703: Message 70
Received data from 172.16.11.3:58703: Message 71
Received data from 172.16.11.3:58703: Message 72
Received data from 172.16.11.3:58703: Message 73
Received data from 172.16.11.3:58703: Message 74
Received data from 172.16.11.3:58703: Message 75
Received data from 172.16.11.3:58703: Message 76
Received data from 172.16.11.3:58703: Message 77
Received data from 172.16.11.3:58703: Message 78
Received data from 172.16.11.3:58703: Message 79
Received data from 172.16.11.3:58703: Message 80
Received data from 172.16.11.3:58703: Message 81
Received data from 172.16.11.3:58703: Message 82
Received data from 172.16.11.3:58703: Message 83
Received data from 172.16.11.3:58703: Message 84
Received data from 172.16.11.3:58703: Message 85
Received data from 172.16.11.3:58703: Message 86
Received data from 172.16.11.3:58703: Message 87
Received data from 172.16.11.3:58703: Message 88
Received data from 172.16.11.3:58703: Message 89
Received data from 172.16.11.3:58703: Message 90
Received data from 172.16.11.3:58703: Message 91
Received data from 172.16.11.3:58703: Message 92
Received data from 172.16.11.3:58703: Message 93
Received data from 172.16.11.3:58703: Message 94
Received data from 172.16.11.3:58703: Message 95
Received data from 172.16.11.3:58703: Message 96
Received data from 172.16.11.3:58703: Message 97
Received data from 172.16.11.3:58703: Message 98
Received data from 172.16.11.3:58703: Message 99
Received data from 172.16.11.3:58703: Message 100
```

计算机网络实验报告

客户端：

```
Sent: Message 1
Sent: Message 2
Sent: Message 3
Sent: Message 4
Sent: Message 5
Sent: Message 6
Sent: Message 7
Sent: Message 8
Sent: Message 9
Sent: Message 10
Sent: Message 11
Sent: Message 12
Sent: Message 13
Sent: Message 14
Sent: Message 15
Sent: Message 16
Sent: Message 17
Sent: Message 18
Sent: Message 19
Sent: Message 20
Sent: Message 21
Sent: Message 22
Sent: Message 23
Sent: Message 24
Sent: Message 25
Sent: Message 26
Sent: Message 27
Sent: Message 28
Sent: Message 29
Sent: Message 30
Sent: Message 31
Sent: Message 32
Sent: Message 33
Sent: Message 34
Sent: Message 35
Sent: Message 36
Sent: Message 37
Sent: Message 38
Sent: Message 39
Sent: Message 40
Sent: Message 41
Sent: Message 42
Sent: Message 43
Sent: Message 44
Sent: Message 45
Sent: Message 46
Sent: Message 47
Sent: Message 48
Sent: Message 49
Sent: Message 50
Sent: Message 51
Sent: Message 52
Sent: Message 53
Sent: Message 54
Sent: Message 55
Sent: Message 56
Sent: Message 57
Sent: Message 58
Sent: Message 59
Sent: Message 60
Sent: Message 61
Sent: Message 62
Sent: Message 63
Sent: Message 64
Sent: Message 65
Sent: Message 66
Sent: Message 67
Sent: Message 68
Sent: Message 69
Sent: Message 70
Sent: Message 71
Sent: Message 72
Sent: Message 73
Sent: Message 74
Sent: Message 75
Sent: Message 76
Sent: Message 77
Sent: Message 78
Sent: Message 79
Sent: Message 80
Sent: Message 81
Sent: Message 82
Sent: Message 83
Sent: Message 84
Sent: Message 85
Sent: Message 86
Sent: Message 87
Sent: Message 88
Sent: Message 89
Sent: Message 90
Sent: Message 91
Sent: Message 92
Sent: Message 93
Sent: Message 94
Sent: Message 95
Sent: Message 96
Sent: Message 97
Sent: Message 98
Sent: Message 99
Sent: Message 100
```

抓包分析：



计算机网络实验报告

源 IP：172.16.11.3　　目标 IP：172.16.11.2

源端口：58703　　目标端口：12345

与代码的设定一致

```
serverAddr.sin_port = htons(12345); // 服务器的端口
serverAddr.sin_addr.s_addr = inet_addr("172.16.11.2"); // 服务器的 IP 地址
```

此外，由抓包结果可以看出，客户端向服务器发送 100 个数据，全部被接收，没有出现丢包的情况

实验总结

1、通过实验，深入了解了 UDP 协议的使用和网络通信的基本原理。

2、在发送端使用循环结构来连续发送 100 个 UDP 数据包，从而模拟在真实网络环境中进行数据传输的情况。通过编写程序，可以灵活控制发送数据包的频率和内容，以满足不同的需求。

3、在接收端实现了对接收到的数据包进行统计和分析的功能。通过记录丢失的数据包数量，可以评估网络传输的可靠性，并检查传输过程中是否存在丢包的情况。

4、使用 Wireshark 软件进行数据包的跟踪分析，捕获和分析通信过程中的数据包，查看其详细信息，并获取有关传输延迟、丢包率等性能指标的数据。