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	软件工程	班 级	19 级软件工程	组长	冼子婷
学号	18338072	18346019	18322043		
学生	冼子婷	胡文浩	廖雨轩		

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）。

（2）注意实验时简述设计思路。

（3）引起 UDP 丢包的可能原因是什么？

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）

一、实验目的

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

二、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。

三、实验过程

操作系统环境：Windows

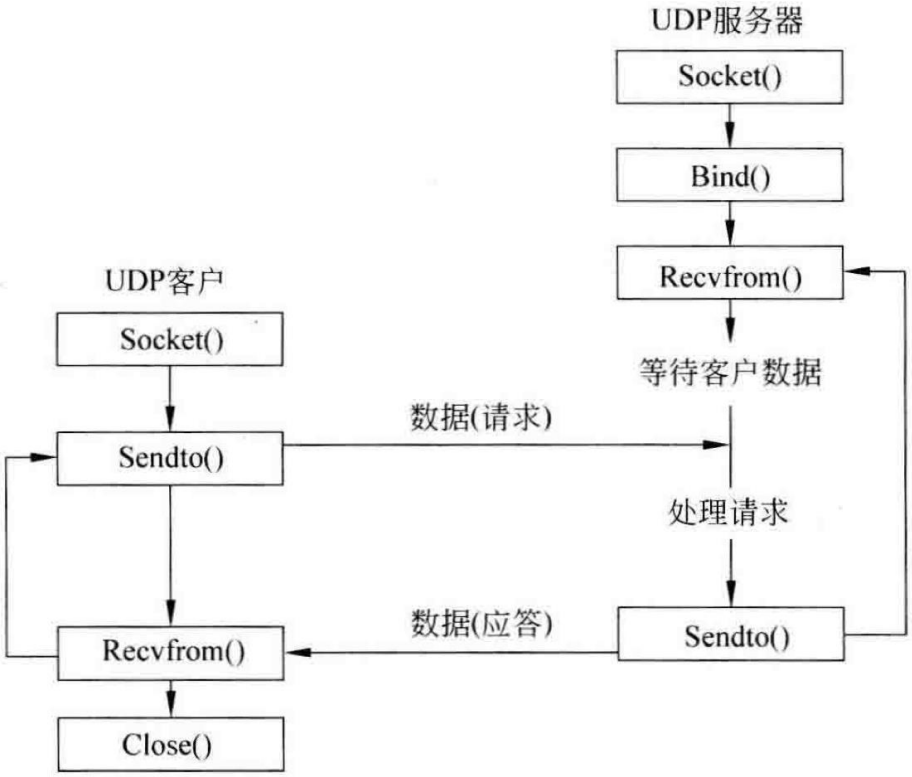
实验步骤：

1） 实验设计思路：

通过研读书上的 TCP 程序例子，以及搜索与 UDP 有关的资料，完成本次实验设计，以下为具体的实验设计思路：

在无连接的 Client / Server 结构中，服务器使用 socket() 和 bind() 函数调用建立和连接 Socket。由于此时的 Socket 是无连接的，服务器使用 recvfrom() 函数从 Socket 接受数据。客户端也只调用 bind() 函数而不调用 connect() 函数。由于无连接的协议不在两个端口之间建立点对点的连接，因此 sendto() 函数要求程序在一个参数中指明目的地址。recvfrom() 函数不需要建立连接，它对到达相连协议端口的任何数据作出相应。当 recvfrom() 函数从 Socket 收到一个数据报时，它将保存发送此数据包的进程的网络地址以及数据包本身。程序（服务器和客户）用保存的地址去确定发送（客户）进程。在必要的条件下，服务器将其应答数据报送到从 recvfrom() 函数调用中所得到的网络地址中去

2） 程序流程图：



3) 程序源码及其关键函数的说明

下面代码第一份为 UDPServer.c，第二份为 UDPClient.c

计算机网络实验报告

```c
/*
    **compile**
    gcc UDPServer.c -lwsock32 -o UDPServer
    **execute**
    ./UDPServer
*/
#include <stdio.h>
#include <winsock2.h>

#define true 1
#define PORT 8000                   // The port on which to listen for incoming data
#define SIZE 1024                   // Max length of buffer
#pragma comment(lib, "ws2_32.lib")  // Winsock Library

int main() {
    SOCKET udpSocket;                   // windows socket
    WSADATA winSocketApi;               // windows socket api
    struct sockaddr_in server, client;  // socket address
    int clientLen = sizeof(client);

    // Prepare the sockaddr_in struct
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT);

    // Initializing winsock
    printf("Initialising Winsock...\n");
    if (WSAStartup(MAKEWORD(2, 2), &winSocketApi) != 0) {
        printf("Initializeing failed, with error Code: %d\n", WSAGetLastError());
        exit(EXIT_FAILURE);
    }

    // Creating socket
    printf("Creating Socket...\n");
    if ((udpSocket = socket(AF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET) {
        printf("Could not create socket: %d\n", WSAGetLastError());
        exit(EXIT_FAILURE);
    }

    // Bind
    printf("Binding...\n");
    if (bind(udpSocket, (struct sockaddr *)&server, sizeof(server)) == SOCKET_ERROR) {
        printf("Bind failed with error code: %d\n", WSAGetLastError());
        exit(EXIT_FAILURE);
    }       Adbean, a day ago • finished program

    int total = 0;
    printf("Listening...\n");
    printf("Enter buffer size: ");
    int bufferLen;
    scanf("%d", &bufferLen);
    char *buffer = malloc(sizeof(char) * bufferLen);
    while (true) {
        // printf("Waiting for packet...\n");
        // receive packets, which is a blocking call
        // the third parameter of the sendto function is the len of the packet
        int receiveLen = recvfrom(udpSocket, buffer, bufferLen, 0, (struct sockaddr *)&client, &clientLen);
        // fflush(stdout);
        // memset(buffer, '\0', bufferLen);
        if (receiveLen == SOCKET_ERROR) {
            printf("recvfrom() failed with error code: %d\n", WSAGetLastError());
            continue;
        }

        if (strcmp("quit", buffer) == 0) {
            printf("total: %d\n", total);
            break;
        }

        printf("Receiving packets from %s:%d\n", inet_ntoa(client.sin_addr), ntohs(client.sin_port));
        printf("Packet[%d] received:\n%s\n", ++total, buffer);

        // Reply the client with the same data
        // if (sendto(udpSocket, buffer, receiveLen, 0, (struct sockaddr *)&client, clientLen) == SOCKET_ERROR) {
        //     printf("sendto() failed with error code: %d\n", WSAGetLastError());
        //     // exit(EXIT_FAILURE);
        //     // failed++, total++;
        // }
    }
    closesocket(udpSocket);
    WSACleanup();
    return 0;
}
```

计算机网络实验报告

```c
/*
    **compile**
    gcc UDPClient.c -lwsock32 -o UDPClient
    **execute**
    ./UDPClient      Adbean, a day ago • finished program
*/
#include <stdio.h>
#include <winsock2.h>

#pragma comment(lib, "ws2_32.lib")  // Winsock Library
#define true 1
// #define SERVER "39.104.16.218"  // ip address of udp server
#define SERVER "127.0.0.1"  // ip address of udp server
#define SIZE 1024           // Max length of buffer
#define PORT 8000           // The port on which to listen for incoming data

int main(void) {
    SOCKET winSocket;
    WSADATA winSocketApi;
    struct sockaddr_in server;

    int serverLen = sizeof(server);
    // char buffer[SIZE], message[SIZE];

    printf("Initializing windows socket...\n");
    if (WSAStartup(MAKEWORD(2, 2), &winSocketApi) != 0) {
        printf("Failed. Error Code: %d\n", WSAGetLastError());
        exit(EXIT_FAILURE);
    }

    // Create socket
    if ((winSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == SOCKET_ERROR) {
        printf("Creating socket failed with error code: %d\n", WSAGetLastError());
        exit(EXIT_FAILURE);
    }

    // Setup address struct
    memset((char *)&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_port = htons(PORT);
    server.sin_addr.S_un.S_addr = inet_addr(SERVER);

    int count = 100;
    printf("Enter buffer size: ");
    int bufferLen;
    scanf("%d", &bufferLen);
    printf("Enter message: ");
    char *buffer = malloc(sizeof(char) * bufferLen);
    char *message = malloc(sizeof(char) * bufferLen);
    scanf("%s", message);
    getchar();
    printf("Sending message for %d times...\n", count);
    while (count--) {
        // sending the message to the server
        // the third parameter of the sendto function is the len of the packet
        if (sendto(winSocket, message, bufferLen, 0, (struct sockaddr *)&server, serverLen) == SOCKET_ERROR) {
            printf("sendto() failed with error code: %d\n", WSAGetLastError());
            continue;
        }

        // receive a reply and print it
        // memset(buffer, '\0', bufferLen);  // clear the buffer
        // try to receive data, which is a blocking call
        // if (recvfrom(winSocket, buffer, bufferLen, 0, (struct sockaddr *)&server, &serverLen) == SOCKET_ERROR) {
        //     printf("recvfrom() failed with error code: %d\n", WSAGetLastError());
        //     continue;
        // }
        printf("sending packets %d\n", count);
        // Sleep(1);
    }
    // strcpy(message, "quit");
    // if (sendto(winSocket, message, strlen(message), 0, (struct sockaddr *)&server, serverLen) == SOCKET_ERROR) {
    //     printf("sendto() failed with error code : %d\n", WSAGetLastError());
    // }
    closesocket(winSocket);
    WSACleanup();
    return 0;
}
```

a) sendto function (winsock.h)

```
int sendto(
  SOCKET         s,          // 套接口文件描述符。
  const char     *buf,       // 指向包含要传输数据的缓冲区的指针
  int            len,        // 缓冲区数据的长度（以字节为单位）
  int            flags,      // 用于指定进行呼叫方式的标志，一般为 0
  const sockaddr *to,        // 指向 sockaddr 结构的可选指针，该结构包含目标套接字的地址
  int            tolen       // 参数所指向的地址的大小（以字节为单位）
);
```

sendto()函数将数据发送到指定目标设备使用套接字的端口。如果没有发生错误，sendto()返回发送的字节总数，它可以小于 len 所指示的字节数。否则，将返回套接字错误的值，并且可以通过调用 WSAGetLastError 来检索特定的错误代码。

b) recvfrom function (winsock.h)

```
int recvfrom(
  SOCKET   s,           // 标识绑定套接字的描述符。
  char     *buf,        // 指向包含要传输的数据的缓冲区的指针
  int      len,         // 缓冲区数据的长度（以字节为单位）
  int      flags,       //为关联套接字指定的选项之外，还可以修改函数调用的行为
  sockaddr *from,       // 指向 sockaddr 结构缓冲区的可选指针，该缓冲区将在返回时保留源地址
  int      *fromlen     // 指向 from 参数所指向的缓冲区大小的可选指针，以字节为单位
);
```

recvfrom()函数监听本机套接字绑定的端口，接收端口所收到的数据包，并保存数据包的源地址。由于使用 UDP 协议通信的双方没有进行连接，接收端必须使用 recvfrom()函数进行实时的阻塞式监听。一旦侦测到有发送到的数据包则立即进行接收与解析。如果未发生错误，则 recvfrom 返回接收到的字节数。 如果已正常关闭连接，则返回值为零。 否则，将返回 SOCKET_ERROR 的值，并且可以通过调用 WSAGetLastError 来检索特定的错误代码。

c) socket function(winsock.h)

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

socket()函数创建一个绑定到特定传输服务提供者的套接字。

d) bind function(winsock.h)

```
int bind(
  SOCKET         s,
  const sockaddr *addr,
  int            namelen
);
```

bind()函数将本地指定端口号与套接字相关联。

e) WSAStartup function(winsock.h)

```
int WSAStartup(
  WORD      wVersionRequired,
  LPWSADATA lpWSAData
);
```

WSAStartup()函数启动进程对 Winsock DLL 的使用。

实验结果及分析：

1. 局域网：

指定客户端 sendto()函数和服务端 recvfrom()函数缓冲区大小为 512 字节时

ip.src == 172.18.34.142

No.	Time	Source	Destination	Protocol	Length	Info
5099	17.912740	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5100	17.912996	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5101	17.912996	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5102	17.913230	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5103	17.913465	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5104	17.913684	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5105	17.913996	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5106	17.914220	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5107	17.914434	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5108	17.914641	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5109	17.914641	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5110	17.914857	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5111	17.915066	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5112	17.915066	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5113	17.915288	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5114	17.915523	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5115	17.915523	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512
5116	17.915736	172.18.34.142	172.18.32.128	UDP	554	63410 → 8000 Len=512

已显示: 100 (1.2%)

```
Packet[96] received:
512
Receiving packets from 172.18.34.142:63410
Packet[97] received:
512
Receiving packets from 172.18.34.142:63410
Packet[98] received:
512
Receiving packets from 172.18.34.142:63410
Packet[99] received:
512
Receiving packets from 172.18.34.142:63410
Packet[100] received:
512
```

指定客户端 sendto()函数和服务端 recvfrom()函数缓冲区大小为 1024 字节时

计算机网络实验报告

ip.src == 172.18.34.142

No.	Time	Source	Destination	Protocol	Length	Info
128	5.169965	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
129	5.169965	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
130	5.169965	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
131	5.170333	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
132	5.170597	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
133	5.170864	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
134	5.171108	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
135	5.171617	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
136	5.172808	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
137	5.173150	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
138	5.173603	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
139	5.173847	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
140	5.174637	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
141	5.175345	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
142	5.175971	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
143	5.176313	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024
144	5.176711	172.18.34.142	172.18.32.128	UDP	1066	52326 → 8000 Len=1024

· 已显示：100 (23.7%)

```
Packet[96] received:
1024
Receiving packets from 172.18.34.142:52326
Packet[97] received:
1024
Receiving packets from 172.18.34.142:52326
Packet[98] received:
1024
Receiving packets from 172.18.34.142:52326
Packet[99] received:
1024
Receiving packets from 172.18.34.142:52326
Packet[100] received:
1024
```

指定客户端 sendto()函数和服务端 recvfrom()函数缓冲区大小为 2048 字节时（校园网夜间繁忙时段）

ip.src == 172.18.34.142 && udp

No.	Time	Source	Destination	Protocol	Length	Info
532	14.530757	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
534	14.531065	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
536	14.531065	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
538	14.531371	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
540	14.531371	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
542	14.531644	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
544	14.531857	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
546	14.532077	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
548	14.532077	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
550	14.532304	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
552	14.532304	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
554	14.532574	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048
556	14.532819	172.18.34.142	172.18.32.128	UDP	610	53227 → 8000 Len=2048

· 已显示：100 (12.4%)

```
Packet[54] received:
2048
Receiving packets from 172.18.34.142:53227
Packet[55] received:
2048
Receiving packets from 172.18.34.142:53227
Packet[56] received:
2048
Receiving packets from 172.18.34.142:53227
Packet[57] received:
2048
Receiving packets from 172.18.34.142:53227
Packet[58] received:
2048
```

指定客户端 sendto()函数和服务端 recvfrom()函数缓冲区大小为 2048 字节且进行 sleep 时

ip.src == 172.18.34.142 && udp

No.	Time	Source	Destination	Protocol	Length	Info
63	3.894789	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
65	3.900774	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
67	3.915824	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
69	3.931090	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
72	3.946369	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
75	3.961705	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
77	3.977250	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
79	3.992518	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
82	4.007831	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
84	4.022966	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
87	4.038121	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
89	4.053720	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048
91	4.068393	172.18.34.142	172.18.32.128	UDP	610	65124 → 8000 Len=2048

· 已显示: 100 (28.2%)

```
Packet[96] received:
2048
Receiving packets from 172.18.34.142:65124
Packet[97] received:
2048
Receiving packets from 172.18.34.142:65124
Packet[98] received:
2048
Receiving packets from 172.18.34.142:65124
Packet[99] received:
2048
Receiving packets from 172.18.34.142:65124
Packet[100] received:
2048
```

2. 互联网：

指定客户端 sendto()函数和服务端 recvfrom()函数缓冲区大小为 512 字节时

计算机网络实验报告

ip.src == 120.236.174.162 && udp

No.	Time	Source	Destination	Protocol	Length	Info
1497	13.406015	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1498	13.406244	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1499	13.406307	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1500	13.406475	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1501	13.406786	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1502	13.406929	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1503	13.406974	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1504	13.407205	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1505	13.407205	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1506	13.407309	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1507	13.407417	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1508	13.407561	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1509	13.407848	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1510	13.407956	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1511	13.408255	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1512	13.408632	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1513	13.408948	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1514	13.409013	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1515	13.409033	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1516	13.409435	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512
1517	13.409435	120.236.174.162	172.16.91.185	UDP	554	16868 → 8000 Len=512

已显示: 100 (1.8%)

```
Packet[95] received:
512
Receiving packets from 120.236.174.162:16868
Packet[96] received:
512
Receiving packets from 120.236.174.162:16868
Packet[97] received:
512
Receiving packets from 120.236.174.162:16868
Packet[98] received:
512
Receiving packets from 120.236.174.162:16868
Packet[99] received:
512
Receiving packets from 120.236.174.162:16868
Packet[100] received:
512
```

指定客户端 sendto()函数和服务端 recvfrom()函数缓冲区大小为 1024 字节时

计算机网络实验报告

ip.src == 120.236.174.162 && udp

No.	Time	Source	Destination	Protocol	Length	Info
1306	28.934904	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1307	28.935063	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1308	28.935063	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1309	28.935203	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1310	28.935373	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1311	28.935490	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1312	28.935563	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1313	28.935828	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1314	28.935886	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1315	28.935934	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1316	28.936021	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1317	28.936096	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1318	28.936131	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1319	28.936192	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1320	28.936443	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1321	28.938409	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1322	28.939432	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1323	28.939955	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1324	28.940037	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1325	28.940504	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024
1326	28.940596	120.236.174.162	172.16.91.185	UDP	1066	7355 → 8000 Len=1024

已显示：100 (4.5%)

```
Receiving packets from 120.236.174.162:7355
Packet[95] received:
1024
Receiving packets from 120.236.174.162:7355
Packet[96] received:
1024
Receiving packets from 120.236.174.162:7355
Packet[97] received:
1024
Receiving packets from 120.236.174.162:7355
Packet[98] received:
1024
Receiving packets from 120.236.174.162:7355
Packet[99] received:
1024
Receiving packets from 120.236.174.162:7355
Packet[100] received:
1024
```

指定客户端 sendto()函数和服务端 recvfrom()函数缓冲区大小为 2048 字节时

ip.src == 120.236.174.162 && udp

No.	Time	Source	Destination	Protocol	Length	Info
845	10.853578	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
847	10.854046	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
849	10.854554	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
851	10.855083	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
853	10.855568	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
855	10.856073	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
857	10.856591	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
859	10.857084	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
861	10.857605	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
863	10.858112	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
865	10.858617	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
867	10.859174	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
869	10.859722	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
871	10.860232	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
873	10.860666	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
875	10.861228	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
877	10.861282	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
879	10.861282	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
881	10.861390	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
883	10.861390	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048
885	10.861390	120.236.174.162	172.16.91.185	UDP	610	30852 → 8000 Len=2048

· 已显示: 98 (5.0%)

```
2048
Receiving packets from 120.236.174.162:30852
Packet[66] received:
2048
Receiving packets from 120.236.174.162:30852
Packet[67] received:
2048
Receiving packets from 120.236.174.162:30852
Packet[68] received:
2048
Receiving packets from 120.236.174.162:30852
Packet[69] received:
2048
Receiving packets from 120.236.174.162:30852
Packet[70] received:
2048
```

指定客户端 sendto()函数和服务端 recvfrom()函数缓冲区大小为 2048 字节且进行 sleep 时

计算机网络实验报告

ip.src == 120.236.174.162 && udp

No.	Time	Source	Destination	Protocol	Length	Info
1484	18.118486	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1488	18.133698	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1490	18.149574	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1492	18.165455	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1505	18.181207	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1509	18.196295	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1512	18.211970	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1516	18.227199	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1524	18.242260	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1526	18.258102	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1530	18.274352	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1532	18.290147	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1541	18.306796	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1543	18.320811	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1547	18.336482	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1551	18.352249	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1556	18.367497	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1558	18.383111	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1563	18.398415	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1568	18.414084	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1572	18.428814	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1577	18.445153	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048
1581	18.460788	120.236.174.162	172.16.91.185	UDP	610	22501 → 8000 Len=2048

已显示: 100 (4.6%)

```
2048
Receiving packets from 120.236.174.162:22501
Packet[96] received:
2048
Receiving packets from 120.236.174.162:22501
Packet[97] received:
2048
Receiving packets from 120.236.174.162:22501
Packet[98] received:
2048
Receiving packets from 120.236.174.162:22501
Packet[99] received:
2048
Receiving packets from 120.236.174.162:22501
Packet[100] received:
2048
```

1.引起 UDP 丢包的可能原因是什么？

从上述实验中可以看出，当指定指定客户端 sendto()函数和服务端 recvfrom()函数缓冲区大小从 512 字节到 2048 字节时，丢包率会上升，这是因为发送的包大小太大且客户端发送频率过快，而导致服务端处理不及时导致丢包。

在设置缓冲区大小为 2048 字节时，我们又对客户端进行了每发送一次进行 Sleep(1)睡眠 1 毫秒的设置，可以发现其丢包率会大大降低，这是因为降低了客户端发送包的频率的同时，服务器有足够的时间去处理收到的包，丢包率就大大改善了。当然，也可以对服务器和客户端多设置一个回拨功能，这样也能大大改善丢包率，这个功能在

我们源码中被注释掉了，打开即可。

同时我们发现，在不同的时间节点进行局域网间 UDP 协议传输，其丢包率也有不同，我们猜测是在夜间 9 点左右校园网使用人数骤升，网络处于繁忙状态，导致丢包率上升。

但在实验过程中，发生丢包时，我们发现 Wireshark 软件探测到的 UDP 包数和服务器端收到的数量是不同的，我们猜测这跟 Wireshark 实现的原理有关，Wireshark 是基于 winpcap 对 windows 底层网络进行访问抓包的，导致其抓包结果可能有些不同。

综上，当使用 UDP 协议传输时，不论对于服务器端的 recvfrom()函数还是客户端的 sendto()函数的第三个参数 len，最好选择合适的大小，比如 strlen(buffer)，这样可以大大降低丢包率：

```
56          // the third parameter of the sendto function is the len of the packet
57          int receiveLen = recvfrom(udpSocket, buffer, strlen(buffer), 0, (struct sock
58          // fflush(stdout);
59          // memset(buffer, '\0', bufferLen);
60          if (receiveLen == SOCKET_ERROR) {
61              printf("recvfrom() failed with error code: %d\n", WSAGetLastError());
```

问题 输出 调试控制台 终端

```
204
Receiving packets from 172.18.34.142:63178
Packet[97] received:
204
Receiving packets from 172.18.34.142:63178
Packet[98] received:
204
Receiving packets from 172.18.34.142:63178
Packet[99] received:
204
Receiving packets from 172.18.34.142:63178
Packet[100] received:
204
```

四、实验思考

1. 实验中遇到的问题及解决方法：

1) 使用 C 语言编程时，直接进行编译会存在大量未定义的参数，原因是编译时没有链接到 Windows Socket API 库。

```
$ gcc UDPClient.c -o UDPClient
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0x3a): undefined reference to `__imp_WSAStartup'
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0x47): undefined reference to `__imp_WSAGetLastError'
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0x78): undefined reference to `__imp_socket'
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0x92): undefined reference to `__imp_WSAGetLastError'
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0xd5): undefined reference to `__imp_htons'
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0xe9): undefined reference to `__imp_inet_addr'
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0x1ae): undefined reference to `__imp_sendto'
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0x1bc): undefined reference to `__imp_WSAGetLastError'
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0x206): undefined reference to `__imp_closesocket'
C:\Users\MXDAM\AppData\Local\Temp\ccs6StzJ.o:UDPClient.c:(.text+0x20f): undefined reference to `__imp_WSACleanup'
collect2.exe: error: ld returned 1 exit status
```

在编译时加入参数-lwsock32，使得编译时动态链接到 Windows Socket API 库，此时能够成功编译。

```
MXDAM@DESKTOP-HA026BV MINGW64 /d/study/软工/大二下/计算机网络/实验/ComputerNetworkExperiment/实验2 (main)
$ gcc UDPClient.c -lwsock32 -o UDPClient
```

2) 对 sendto()，recvfrom()函数的参数与缓冲区不熟悉，以及对 UDP 协议概念不理解

通过查询 Winsock.h 库的官方文档，了解到 recvfrom()和 sendto()函数的原型与定义，通过实践后逐渐会使用 winsock 库进行 UDP 协议传输。

4) 校园内仅有局域网的实验环境，互联网下的实验环境需要另外寻找

我们使用阿里云提供的云服务器 ESC 配置互联网下的实验环境。

5) 校园网无线设备不能互联

在使用校园网 WIFI 无线网时，发现无线设备间无法进行互联的情况，我们将实验设备接入校园网有线网络进行实验。

2. connect()、bind()等函数中 struct sockaddr *addr 参数各个部分的含义及具体的数据举例

（1）sockaddr 结构体的定义如下：

```
struct sockaddr {
    sa_family_t sin_family;  // 地址族（Address Family），也就是地址类型
    char sa_data[14];        // IP 地址和端口号
};
```

sockaddr 是一种通用的结构体，可以用来保存多种类型的 IP 地址和端口号。

（2）bind() 函数的原型为：

```
int bind(int sock, struct sockaddr *addr, socklen_t addrlen);  //Linux
int bind(SOCKET s, const sockaddr *addr, int namelen);         //Windows
```

在 Windows 中：

s 是标识未绑定套接字的描述符， addr 指向要分配给绑定套接字的本地地址的 sockaddr 结构的指针，namelen 是 name 参数所指向的值的长度（以字节为单位）。

bind()函数通常用于绑定到面向连接的（流）套接字或无连接的（数据报）套接字。该绑定函数也可以用于结合原始套接字（套接字被通过调用所生成的套接字与函数型参数集 SOCK_RAW）

下面的代码，将创建的套接字与 IP 地址 127.0.0.1、端口 1234 绑定：

```
// 创建套接字
int serv_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

// 创建 sockaddr_in 结构体变量
struct sockaddr_in serv_addr;
memset(&serv_addr, 0, sizeof(serv_addr));           //每个字节都用 0 填充
serv_addr.sin_family = AF_INET; /                  /使用 IPv4 地址
serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");  //具体的 IP 地址
```

```
serv_addr.sin_port = htons(1234);               //端口

// 将套接字和 IP、端口绑定
bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
```

（3）connect()函数的原型为：

```
int connect(int sock, struct sockaddr *serv_addr, socklen_t addrlen);    //Linux
int connect(SOCKET sock, const sockaddr *name, int namelen);             //Windows
```

以 Windows 为例：

s 是标识未连接套接字的描述符， addr 指向应建立连接的 sockaddr 结构的指针，namelen 是 name 参数所指向的 sockaddr 结构的长度（以字节为单位）。

connect()用于创建到指定的目的地的连接。如果套接字 s 是未绑定的，则系统会将唯一值分配给本地关联，并将套接字标记为已绑定。

3. 面向连接的客户端和面向非连接的客户端在建立 Socket 时的区别

面向连接的客户端在建立 Socket 的时候需要使用 connect()函数请求与服务端建立连接通路，成功后才能进行数据的传输。

面向非连接的客户端建立 Socket 时不需要建立通路，但是需要指定服务端的接受数据的端口才可以进行数据的传输。

4. 面向连接的客户端和面向非连接的客户端收发数据时的区别，面向非连接的客户端判断发送结束的方法

面向连接的客户端（基于 TCP 协议的数据传输）：

- 具有可靠的数据传输，也就说在数据在传输中，无失序、无差错、无丢失、无重复。
- 在数据传输前和传输结束后需要建立连接和断开连接
- 收发数据前，需要通过三次握手过程建立连接，传输结束后，需要通过四次挥手过程断开连接

适用于传输较大的内容或文件，网络良好，需要保证传输可靠性的情况。

面向非连接的非客户端（基于 UDP 协议的数据传输）：

- 不能保证数据传输的可靠性
- 不具有数据连接和断开的过程
- 数据的收发比较自由

适用于网络情况可能产生丢包，对传输可靠性要求低的情况。

面向非连接客户端要判断发送结束需要一个标志位，在实验中我们采用了 `count` 作为计数器。将 count 设定为 100 作为循环开始的初始参数，以 count 为 0 时作为循环结束的条件。每次循环都向目标地址发送一个数据包，同时 count 进行自减操作。

```c
int count = 100;
printf("Enter buffer size: ");
int bufferLen;
scanf("%d", &bufferLen);
printf("Enter message: ");
char *buffer = malloc(sizeof(char) * bufferLen);
char *message = malloc(sizeof(char) * bufferLen);
scanf("%s", message);
getchar();
printf("Sending message for %d times...\n", count);
while (count--) {
    // sending the message to the server      Adbean, a day ago • udp program
    // the third parameter of the sendto function is the len of the packet
    if (sendto(winSocket, message, bufferLen, 0, (struct sockaddr *)&server, serverLen) == SOCKET_ERROR) {
        printf("sendto() failed with error code: %d\n", WSAGetLastError());
        continue;
    }

    // receive a reply and print it
    // memset(buffer, '\0', bufferLen);  // clear the buffer
    // try to receive data, which is a blocking call
    // if (recvfrom(winSocket, buffer, bufferLen, 0, (struct sockaddr *)&server, &serverLen) == SOCKET_ERROR) {
    //     printf("recvfrom() failed with error code: %d\n", WSAGetLastError());
    //     continue;
    // }
    printf("sending packets %d\n", count);
}
```

5. 面向连接的通信和无连接通信的优缺点及场合

- 面向连接通信
  - 在通信双方进行通信之前会建立起一条通信连接，预留端系统间通信沿路径上所需要的资源。
  - 优点
    1. 数据发送的时候会严格按照发送顺序进行传输
    2. 发送的数据会以字节流的形式传送，无需限定发送的长度
    3. 会自动校验发送数据的
  - 缺点
    1. 所需通信链路建立的时间相对比较长
    2. 建立链路之后资源被通信双方占有，当该链路空闲的时候会造成浪费，过多的链路同时存在可能会造成服务器的崩溃。

- 面向无连接通信
  - 将数据包进行分组，然后使用存储转发传输。存储的数据包等待输出线路空闲时再发出。
  - 优点
    1. 不需要建立通信线路，可以随时发送数据包
    2. 传输不是沿着固定的线路进行，可以使用任何空闲的线路进行传输
    3. 客户端无需指定端口
  - 缺点
    1. 不提供无错保证，丢包率相对比较高
    3. 数据包传输会存在乱序和重复的情况

6. 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

本实验中使用的 Socket 默认是工作在阻塞方式的，但是可以通过一些设置使其工作在非阻塞方式。

- 阻塞方式

  当某个操作为阻塞式时，它会一直进入等待状态，直到至前置操作全部完成并有信号通知其可以执行。

  如 recvfrom()函数，当客户端没有发送数据或者数据发送完成之后，服务端会进入等待状态，也即不会执行图中最下方的两个 printf()函数。只有客户端发送的数据包被服务端接收到，recvfrom()函数才会有返回值赋值给 receiveLen，并向下执行 printf()函数。

```
int receiveLen = recvfrom(udpSocket, buffer, bufferLen, 0, (struct sockaddr *)&client, &clientLen);
// fflush(stdout);
// memset(buffer, '\0', bufferLen);
if (receiveLen == SOCKET_ERROR) {
    printf("recvfrom() failed with error code: %d\n", WSAGetLastError());
    continue;
}

// if (strcmp("quit", buffer) == 0) {
//     printf("total: %d\nsuccess: %d\nfailed: %d\n", total, success, failed);
//     break;
// }

printf("Receiving packets from %s:%d\n", inet_ntoa(client.sin_addr), ntohs(client.sin_port));
printf("Packet[%d] received:\n%s\n", ++total, buffer);
```

- 非阻塞方式

  不需要等待任何返回值，按照程序原本设定好的操作顺序执行下去。

学号	学生	自评分
18338072	冼子婷	98
18322043	廖雨轩	98
18346019	胡文浩	98

【交实验报告】

上传实验报告：ftp://172.18.178.1/　　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_ Ftp 协议分析实验.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_ Ftp 协议分析实验.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！