



本科实验报告

课程名称: 计算机网络基础

实验名称: 基于 Socket 接口实现自定义协议通信

姓 名: 李秋宇

学 院: 计算机学院

系: 计算机

专 业: 计算机科学与技术

学 号: 3220103373

指导教师: 邱劲松

2024 年 11 月 05 日

浙江大学实验报告

实验名称: 基于 Socket 接口实现自定义协议通信 实验类型: 编程实验

同组学生: 万镇杰 实验地点: 计算机网络实验室

一、 实验目的

- 学习如何设计网络应用协议
- 掌握 Socket 编程接口编写基本的网络应用软件

二、 实验内容

根据自定义的协议规范，使用 Socket 编程接口编写基本的网络应用软件。

- 掌握 C 语言形式的 Socket 编程接口用法，能够正确发送和接收网络数据包
- 开发一个客户端，实现人机交互界面和与服务器的通信
- 开发一个服务端，实现并发处理多个客户端的请求
- 程序界面不做要求，使用命令行或最简单的窗体即可
- 功能要求如下：
 1. 运输层协议采用 TCP 或 UDP，客户端和服务端必须同时支持两种协议，在程序运行时，通过配置或命令行参数选择使用其中一种
 2. 客户端采用交互菜单形式，用户可以选择以下功能：
 - a) 连接：请求连接到指定地址和端口的服务端。服务端可以运行多个实例（绑定不同的地址或端口），客户端可以同时与多个服务端保持连接，一旦连接成功，给服务端分配一个 ID，用于后续操作。
 - b) 断开连接：断开与指定 ID 的服务端的连接。
 - c) 获取城市名字：向指定 ID 的服务端请求给出某个区号对应的中英文城市名称。
 - d) 获取气象信息：向指定 ID 的服务端请求给出指定日期、指定城市的气象信息，气象信息包括气温、阴/晴/雨状态、风向、风力、湿度等。
 - e) 活动连接列表：向指定 ID 的服务端请求给出当前连接的所有客户端信息（编号、IP 地址、端口等）。
 - f) 发消息：请求服务端把消息转发给对应编号的客户端，该客户端收到后显示在屏幕上
 - g) 退出：断开所有服务端的连接并退出客户端程序。

同时，客户端后台接收服务端推送的消息，解码后以人类可读的形式，实时显示在屏幕上。

服务端推送的消息包括：

- a) 其他客户端发送的消息
 - b) 其他客户端上线、下线的消息
 - c) 气象预警信息，包括地震、台风等。
3. 服务端接收到客户端请求后，根据客户端传过来的指令完成特定任务：
 - a) 服务端可以同时接受多个客户端连接和请求，一旦客户端连接成功，给客户端分配一个 ID，用于后续操作。

- b) 向客户端传送所请求的区号对应的城市名称。服务端从一个配置文件中读取城市和区号的对照列表，配置文件初始时至少有 20 个城市，后期可增加城市，不同的服务端实例可以有不同的配置数据，首次运行时加载。如果请求的区号不在配置列表中，返回相应的错误信息。
- c) 向客户端传送服务端所请求的日期、城市的气象信息。服务端从一个配置文件中读取所有城市七天内的气象信息，配置文件初始时至少有 20 个城市的七天内气象信息，后期可增加城市，不同的服务端实例可以有不同的配置数据，首次运行时加载。如果请求的城市或者日期不在配置列表中，返回相应的错误信息。
- d) 向客户端传送当前连接的所有客户端信息（ID、IP 地址、端口等）。
- e) 当客户端连接成功后，向其他在线的客户端推送该客户端上线的消息。当客户端断开连接后，向其他在线的客户端推送该客户端下线的消息。
- f) 将某客户端发送过来的内容转发给指定 ID 的其他客户端，如果 ID 不存在或未连接，返回相应的错误信息。
- g) 采用异步多线程编程模式，正确处理多个客户端同时连接，同时发送消息的情况

同时，服务端提供人机交互界面，实现人工触发向所有在线的客户端推送气象预警消息的功能。气象预警信息包括地震和台风两类。地震信息包括地震时间、震中经纬度、震级。台风信息包括台风中心经纬度、级别、预计登录地点和时间。

- 根据上述功能要求，设计一个客户端和服务端之间的应用通信协议
- **本实验涉及到网络数据包发送部分不能使用任何的 Socket 封装类，只能使用最底层的 C 语言形式的 Socket API。程序应使用标准 C 或 C++ 语言。**
- 本实验应组成二人小组，共同完成协议的设计。服务端和客户端由不同人来完成。

三、 主要仪器设备

- 联网的 PC 机、Wireshark 软件
- C/C++ 集成开发环境。

四、 操作方法与实验步骤

- 设计数据包的格式，至少要考虑如下问题：
 - a) 考虑 TCP 与 UDP 的差异，实现基于 TCP 和 UDP 的不同设计
 - b) 可以采用二进制或文本形式，针对 TCP 模式，需要考虑如何在一串连续的字节流中区分多个数据包
 - c) 定义数据包总体结构，考虑结束标记、各字段之间的分隔形式（固定顺序、长度控制或采用分隔符）、长度、包体、校验等。数据包总体分为请求、指示（服务器主动发给客户端的）、响应三类，每一类可以有自己的格式。
 - d) 所有数据包中都要设置一个校验字段，其值为将除校验字段以外的数据包各相关字段与开发者学号连接后通过 MD5 散列计算的结果，即=MD5（数据包各相关字段+学号），参与校验的数据包字段可自行选择，但应包含足够多的信息且双方要一致。客户端和服务端均要记录对方的学号，收到每个数据包首先要对校验字段进行验证（同样算一遍，看结果是否一致）。校验不正确的数据包要丢弃（可选择是否返回错误信息）。
 - e) 定义每种数据包包体，应包含**控制部分**和**数据部分**。控制部分要包含**数据包类型、序号、数据长度**，根据请求、指示、响应的不同特点设计不同的结构。数据部分要根据不同类型设计不同的结构（特别是考虑客户端列表数据如何表达）。

- 客户端编写步骤（**需要采用多线程模式**）
 - a) 运行初始化，调用 `socket()`，向操作系统申请 `socket` 句柄
 - b) 编写一个菜单功能，列出 7 个选项
 - c) 等待用户选择
 - d) 根据用户选择，做出相应的动作（未连接时，只能选连接功能和退出功能）
 1. 选择连接功能：请用户输入服务端 IP 和端口，然后调用 `connect()`，等待返回结果并打印。连接成功后设置连接状态为已连接。为该服务端分配一个 ID，**然后创建一个接收数据的子线程，循环调用 `receive()`，如果收到了一个完整的响应数据包，就通过线程间通信（如消息队列）发送给主线程，然后继续调用 `receive()`，直至收到主线程通知退出。**
 2. 选择断开功能：调用 `close()`，并设置连接状态为未连接。通知并等待子线程关闭。将服务端 ID 设置为无效。
 3. 选择获取城市名称功能：请用户输入要查询的城市区号，然后组装请求数据包，类型设置为城市名称请求，数据参数设置为输入的区号，然后调用 `send()` 将请求数据包发送给服务端，**接着等待接收数据的子线程返回结果**，并根据响应数据包的内容，打印城市名称信息。
 4. 选择获取气象信息功能：请用户输入要查询的城市区号、日期，然后组装请求数据包，类型设置为气象信息请求，数据参数设置为输入的城市区号、日期，然后调用 `send()` 将数据发送给服务端，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印气象信息。
 5. 选择获取客户端列表功能：组装请求数据包，类型设置为列表请求，然后调用 `send()` 将数据发送给服务器，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印客户端列表信息（编号、IP 地址、端口等）。
 6. 选择发送消息功能（选择前需要先获得客户端列表）：请用户输入客户端的列表编号和要发送的内容，然后组装请求数据包，类型设置为消息请求，然后调用 `send()` 将数据发送给服务端，接着等待接收数据的子线程返回结果，并根据响应数据包的内容，打印消息发送结果（是否成功送达另一个客户端）。
 7. 选择退出功能：判断是否存在已连接的服务端，是则先将所有服务端断开，然后再退出程序。否则，直接退出程序。
 8. 主线程除了在等待用户的输入外，还在处理子线程的消息队列，如果有消息到达，则进行处理，如果是响应消息，则打印响应消息的数据内容（比如时间、名字、客户端列表等）；如果是指示消息，则打印指示消息的内容（比如服务器转发的别的客户端的消息内容、发送者编号、IP 地址、端口等）。
 9. 针对 UDP 模式，需要定义特殊类型的请求和响应，用于建立连接和释放连接。客户端收到服务端连接成功的响应后记录状态。如果处于未连接状态，只能发送连接请求，不能发送其他请求。接收数据时调用 `recvfrom()`，发送数据时调用 `sendto()`。
- 服务端编写步骤（**需要采用多线程模式**）
 - a) 运行初始化，解析运行配置或命令行参数，加载城市区号配置文件、气象信息配置文件。
 - b) 调用 `socket()`，向操作系统申请 `socket` 句柄
 - c) 调用 `bind()`，绑定监听端口（**使用学号的后 4 位作为服务器的默认监听端口，同时可通过配置文件或者命令行参数设置为别的**），接着调用 `listen()`，设置连接等待队列长度
 - d) 主线程循环调用 `accept()`，直到返回一个有效的 `socket` 句柄，在客户端列表中增加一个新客户端的项目，分配一个新的 ID，并记录下该客户端句柄和连接状态、端口。然后创建

一个子线程后继续调用 accept()。该子线程的主要步骤是(刚获得的句柄要传递给子线程，子线程内部要使用该句柄发送和接收数据)：

- ◆ 调用 send(), 发送一个握手 (hello) 消息给客户端 (可选)
- ◆ 循环调用 receive(), 如果收到了一个完整的请求数据包, 根据请求类型做相应的动作：
 1. 请求类型为获取城市名称：从数据包中提取请求的区号参数, 然后将对应的城市名称数据组装进响应数据包, 调用 send()发给客户端。如果区号不正确, 将错误代码和出错描述信息组装进响应数据包, 调用 send()发回源客户端。
 2. 请求类型为获取气象信息：从数据包中提取请求的区号、日期参数, 将对应的气象信息 (气温、阴/晴/雨状态) 组装进响应数据包, 调用 send()发给客户端。如果区号、日期不正确, 将错误代码和出错描述信息组装进响应数据包, 调用 send()发回源客户端。
 3. 请求类型为获取客户端列表：读取客户端列表数据, 将 ID、IP 地址、端口等数据组装进响应数据包, 调用 send()发给客户端。
 4. 请求类型为发送消息：根据 ID 读取客户端列表数据, 如果 ID 不存在, 将错误代码和出错描述信息组装进响应数据包, 调用 send()发回源客户端；如果 ID 存在并且状态是已连接, 则将要转发的消息组装进指示数据包。调用 send()发给接收客户端 (使用接收客户端的 socket 句柄), 发送成功后组装转发成功的响应数据包, 调用 send()发回源客户端。
- e) 主线程还负责检测退出指令 (如用户按退出键或者收到退出信号), 检测到后即通知并等待各子线程退出。最后关闭 Socket, 主程序退出。
- f) 针对 UDP 模式, 需要定义特殊类型的请求和响应, 用于建立连接和释放连接。服务端检测该客户端是否处于已连接状态, 如果未连接时收到除了连接/释放连接请求之外的其他请求, 则发回未连接的错误响应。重复收到连接/释放连接请求时, 根据当前连接状态返回适合的响应。接收数据时调用 recvfrom(), 发送数据时调用 sendto()。
- 在一次连接中, 双方对于接收到的过期序号、重复序号的数据包应予丢弃, 序号在一定范围内滚动使用。重新连接后状态复位。建立连接时, 默认起始序号为 0, 也可以进行协商。
- 编程结束后, 双方程序运行, 检查是否实现功能要求, 如果有问题, 查找原因, 并修改, 直至满足功能要求。
- 运行多个服务端实例 (IP 地址、端口之一要不同), 用一个客户端同时连接多个服务端, 测试功能是否正确。
- 使用多个客户端同时连接服务端, 检查并发性。
- 使用 Wireshark 抓取每个功能的交互数据包。

五、实验数据记录和处理

请将以下内容和本实验报告一起打包成一个压缩文件上传：

- 源代码：客户端和服务端的代码分别在一个目录
- 可执行文件：可运行的.exe 文件或 Linux 可执行文件，客户端和服务端各一个
- 编译说明（编译开关、编译环境）和运行说明（参数说明）
- 配置文档说明（如有）

- 描述数据包的总体格式（包含数据包的边界、数据包类型、校验等要素）

字段	字节数量	含义	值或范围	备注
type	1	数据包类型	1~3	1: 请求, 2: 响应, 3:指示
content	1	数据包内容	1~6	数据包内容
id	1	数据包 ID	0~255	重复滚动
length	2	数据包长度	0~65535	包体部分的长度
args	N	包体	N 个字节	包体部分, N=长度字段
checksum	16	校验部分	大写十六进制	对包体部分+学号后的 MD5 值
结束标记	1	结束标记	\n	用于区分数据包的结束

- 描述请求数据包包体的格式（用表格或画图说明），包含每一种请求类型的包体定义

根据请求数据包的请求内容确定包体的参数数量及意义：

请求类型	参数列表	格式说明
获取城市名	参数 1：城市区号	非负整数
获取气象信息	参数 1：城市区号	非负整数
	参数 2：年份	4 个字符, YYYY 格式
	参数 3：月份	2 个字符, MM 格式
	参数 4：日期	2 个字符, DD 格式
获取在线客户列表	无参数	
请求发送消息	参数 1：接收客户端 ID	非负整数
	参数 2：消息内容	字符串
请求建立连接	无参数	
请求断开连接	无参数	

- 描述响应数据包包体的格式（用表格或画图说明），包含每一种响应类型的定义

根据响应内容决定参数数量及意义：

响应类型	参数列表	格式说明
响应城市名	参数 1：是否有正确结果	1 个字符, 0/1
	参数 2： - 有正确结果则返回城市名, - 无正确结果则返回错误信息	字符串
响应气象信息	参数 1：是否有正确结果	1 个字符, 0/1
	参数 2： - 有正确结果则返回城市名	字符串

	<ul style="list-style-type: none"> - 无正确结果则返回错误信息 	
	<p>参数 3:</p> <ul style="list-style-type: none"> - 有正确结果则返回气象信息 - 无正确结果则为空 	字符串
响应在线客户列表	<p>参数 1: 是否请求成功</p>	1 个字符, 0/1
	<p>参数 2:</p> <ul style="list-style-type: none"> - 请求成功则返回活跃客户端数量 - 请求失败则返回错误原因 	非负整数或字符串
	<p>参数 3~n:</p> <ul style="list-style-type: none"> - 请求成功则返回每个活跃客户端的 ID - 请求失败则为空 	非负整数
响应发送消息	<p>参数 1: 是否成功发送消息</p>	1 个字符, 0/1
	<p>参数 2: 发送消息失败时返回错误原因</p>	字符串
响应建立连接	<p>参数 1: 是否成功建立连接</p>	1 个字符, 0/1
响应断开连接	<p>参数 1: 是否成功断开连接</p>	1 个字符, 0/1

- 描述指示数据包的格式（画图说明），包含每一种指示类型的定义

指示类型	参数列表	格式说明
发送消息	参数 1: 消息	字符串
通知所有客户端有新上线的客户端	参数 1: 消息	字符串
通知所有客户端有新下线的客户端	参数 1: 消息	字符串
发布气象预警广播	参数 1: 消息	字符串

2) 以下为关键代码截图，请附上简要说明（图上直接标注，或者文字说明），根据本人从事的开发工作，
服务端和客户端选一个进行说明即可。

- 客户端的人机交互关键代码截图

```

1 /* 客户端采用交互菜单形式，用户可以选择以下功能：
2    a) 连接：请求连接到指定地址和端口的服务端。服务端可以运行多个实例（拥有不同的地址或端口），客户端可以同时与多个服务端保持连接，一旦连接成功，给服务端分配一个ID，用于后续操作。
3    b) 断开连接：断开与指定ID的服务端的连接。
4    c) 获取城市名字：向指定ID的服务端请求输出某个区县对应的中英文城市名称。
5    d) 获取气象信息：向指定ID的服务端请求输出指定日期、指定城市的气象信息。气象信息包括气温、阴/晴/雨状态、风向、风力、湿度等。
6    e) 活动连接列表：向指定ID的服务端请求输出当前连接的所有客户端信息（编号、IP地址、端口等）。
7    f) 发消息：请求服务端把消息转发给对应编号的客户端，该客户端收到后显示在屏幕上。
8    g) 退出：断开所有服务端的连接并退出客户端程序。
9 */
10 #include "Interface.h"
11
12 using namespace std;
13
14 extern std::mutex cout_mtx;
15
16 /**
17 * 选择协议界面
18 */
19 void protocolInterface()
20 {
21     cout << "Please select the protocol you want to use" << endl;
22     cout << "1. TCP" << endl;
23     cout << "2. UDP" << endl;
24 }
25
26 /**
27 * 主界面，格式化输出菜单，使用互斥锁保护输出
28 * @status: 客户端连接状态
29 */
30 void mainInterface(int status)
31 {
32     lock_guard<mutex> lock(cout_mtx);
33     {
34         cout << "\n===== MENU =====";
35         if (status == DISCONNECTED) {
36             cout << "1. Connect to Server" << endl;
37             cout << "2. Exit" << endl;
38         }
39         else {
40             cout << "1. Connect to Server" << endl;
41             cout << "2. Disconnect from Server" << endl;
42             cout << "3. Get City Name" << endl;
43             cout << "4. Get Weather Information" << endl;
44             cout << "5. Get Client List" << endl;
45             cout << "6. Send Message" << endl;
46             cout << "7. Exit" << endl;
47         }
48         cout << "=====";
49     }
50 }
51
52 /**
53 * 显示已连接的服务端
54 */
55 void showConnectedServers()
56 {
57     cout << "\n===== SERVERS =====";
58     if (serverConnections.empty()) {
59         cout << "No server connected." << endl;
60         cout << "=====";
61         return;
62     }
63     cout << "Allocated Server ID: ";
64     for (auto it = serverConnections.begin(); it != serverConnections.end(); it++)
65     {
66         cout << "Server ID: " << it->first << " status: " << (it->second.connected ? "\033[32mConnected\033[0m" : "\033[31mDisconnected\033[0m") << endl;
67     }
68     cout << "=====";
69 }

```

● 客户端的接收数据子线程关键代码截图

```

void worker_UDP(int serverID)
{
    char buffer[1024];
    serverConnection* conn = serverConnections[serverID];
    ssize_t rc;
    struct sockaddr_in addr;
    socklen_t addr_len = sizeof(addr);

    while (1) {
        // cout << "Debug: serverID = " << serverID << endl;
        memset(buffer, 0, sizeof(buffer));
        rc = recvfrom(conn.sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *) &addr, &addr_len);

        // cout << "[Debug] addr: " << inet_ntoa(addr.sin_addr) << ":" << ntohs(addr.sin_port) << endl;
        // cout << "[Debug] conn.addr: " << inet_ntoa(conn.addr.sin_addr) << ":" << ntohs(conn.addr.sin_port) << endl;

        if (rc <= 0) {
            perror("recvfrom failed");
            close(conn.sockfd);
            break;
        }

        // 判断 addr 的地址是否来自邻居，不是则是新
        bool found = false;
        if (serverConnections.find(serverID) != serverConnections.end()) {
            if (serverConnections[serverID].addr.sin_addr.s_addr == addr.sin_addr.s_addr && serverConnections[serverID].addr.sin_port == addr.sin_port) {
                found = true;
            }
        }

        if (!found) {
            cout << "Received message from unknown address, ignored." << endl;
            continue;
        }

        Packet p("3373");
        if (!p.decode(buffer)) {
            cout << "\033[31m[ERROR] Failed to decode message.\033[0m" << endl;
            continue;
        }

        // cout << "[Thread] Received message from server " << serverID << ":" << "(" << buffer << ")" << endl;

        if (p.getContent() == ContentType::ResponseMakeConnection) {
            if (p.getArgs()[0] == "Connection established.") {
                cout << "\033[32m[INFO] Connection established.\033[0m" << endl;
                conn.connected = true;
                messageFlag = true;
                cv.notify_all();
            }
            else {
                cout << "Failed to connect to server ID " << serverID << endl;
                conn.connected = false;
                messageFlag = true;
                cv.notify_all();
            }
        }

        else if (p.getContent() == ContentType::ResponseCloseConnection) {
            if (p.getArgs()[0] == "x") {
                cout << "Received close connection ACK from server ID " << serverID << endl;
                cout << "Connection closed." << endl;
                conn.connected = false;
                messageFlag = true;
                cv.notify_all();
                break;
            }
            else {
                cout << "Failed to close connection to server ID " << serverID << endl;
                conn.connected = true;
                messageFlag = true;
                cv.notify_all();
            }
        }

        else if (conn.connected) {
            lock_guard<mutex> lock(mutex);
            message_queue.push(buffer);
            cv.notify_all();
        }
    }
}

```

```
/*
 * TCP 通信线程，用于接收服务器消息，将消息放入消息队列
 * 使用 互斥锁 和 条件变量 实现线程同步
 * @param s socket file descriptor
 * @return void
 */
void worker(int serverID)
{
    char buffer[1024];
    serverConnection& conn = serverConnections[serverID];
    ssize_t rc;

    while (conn.connected) {

        memset(buffer, 0, sizeof(buffer));
        rc = recv(conn.sockfd, buffer, sizeof(buffer), 0);

        if (rc > 0) {
            string msg(buffer, rc);
            {
                lock_guard<mutex> lock(mtx);
                message_queue.push(msg);
            }
            // 通知主线程
            cv.notify_one();
        } else {
            cout << "Server ID " << serverID << " disconnected." << endl;
            conn.connected = false;
        }
    }

    cout << "\n\033[34m[Thread] Thread for server ID " << serverID << " exited." << "\033[0m" << endl;
}
```

- 服务端的人机交互关键代码截图

```

54 void Server::cmds()
55 {
56     std::string command;
57     while (true) {
58         std::cin >> command;
59         if (command == "init") {
60             server->init();
61         } else if (command == "run" || command == "start") {
62             server->run();
63         } else if (command == "stop") {
64             server->stop();
65         } else if (command == "broadcast") {
66             server->weatherWarning();
67         } else if (command == "help") {
68             help();
69         } else if (command == "exit" || command == "quit") {
70             server->quit();
71             break;
72         } else {
73             std::cout << "Invalid server commands!!! Try again." << std::endl;
74         }
75     }
76     return ;
77 }
```

```

42 /**
43 * Server initializer by creating a socket, setting socket options,
44 * and binding it to the specified IP address and port.
45 */
46 */
47 void init();

48 /**
49 * Server runner.
50 */
51 virtual void run() = 0;

52 /**
53 * Stop the server.
54 */
55 void stop();

56 /**
57 * Stop and exit the server.
58 */
59 void quit();

60 /**
61 * Broadcast weather warning message to all active clients.
62 */
63 void weatherWarning();
```

```

15 void Server_Base::init()
16 {
17     printMessage(ServerMsgType::NOTE, "Server is initializing...");
18     getSocket();
19     setOptions();
20     bindAddress();
21     serverStatus = ServerStatus::READY;
22     printMessage(ServerMsgType::NOTE, "Server initialization completed.");
23 }
24
25 void Server_Base::stop()
26 {
27     switch (serverStatus) {
28         case ServerStatus::RUN:
29             closeServer();
30             printMessage(ServerMsgType::NOTE, "Server shuts down.");
31             break;
32         default:
33             printMessage(ServerMsgType::WARNING, "Server is not running.");
34     }
35     return ;
36 }
37
38 void Server_Base::quit()
39 {
40     serverStatus = ServerStatus::EXIT;
41     printMessage(ServerMsgType::NOTE, "Exiting the server...");
42 }
43
44 void Server_Base::weatherWarning()
45 {
46     std::srand(std::time(0));
47     broadcastMessage(CONTENT_TYPE::AssignmentWeatherWarning, WeatherWarning.at(std::rand() % WarningRecordNums));
48 }
49
50

```

- 服务端的客户端处理子线程关键代码截图

```

57 ~ void Server_TCP::worker()
58 {
59     // Server starts running, wait and accept.
60     serverStatus = ServerStatus::RUN;
61     while (serverStatus == ServerStatus::RUN) {
62         int clientSocket = accept(serverSocket, nullptr, nullptr);
63         if (clientSocket < 0) printMessage(ServerMsgType::ERROR, "Failed to accept client.");
64         startClientThread(clientSocket);
65     }
66     return ;
67 }
68
69 ~ void Server_TCP::startClientThread(int clientSocket)
70 {
71     try {
72         std::thread clientThread(&Server_TCP::process, this, clientSocket);
73         clientThread.detach();
74     } catch (const std::system_error& e) {
75         printMessage(ServerMsgType::ERROR, e.what());
76     }
77     return ;
78 }

```

3) 以下为运行效果截图，请附上简要说明（图上直接标注，或者文字说明），根据本人从事的开发工作，服务端和客户端选一个进行说明即可。请准备好 Wireshark 开始捕获后再运行。

- 初始运行后显示的菜单选项（TCP 模式和 UDP 模式各一组）

客户端：

```
(base) lqy@LAPTOP-SDV26TG8:~/Z/ZJU-Computer-Network-Socket>./client
Please select the protocol you want to use
1. TCP
2. UDP
```

服务端：

```
(base) lqy@LAPTOP-SDV26TG8:~/Z/ZJU-Computer-Network-Socket>./server TCP
[WELCOME] Welcome to the server!!!
*-----*
| SERVER HELP MENU |
*-----*
| AUTHOR: !EEExp3rt |
*-----*
| CMDS | USAGE |
*-----*
| init | Initialize the server |
*-----*
| run | |
*-----* Start the server and run |
| start | |
*-----*
| stop | Stop the server |
*-----*
| broadcast | Broadcast weather warning |
*-----*
| help | Show this help message |
*-----*
| exit | |
*-----* Exit the server program |
| quit | |
*-----*
```

```
(base) lqy@LAPTOP-SDV26TG8:~/Z/ZJU-Computer-Network-Socket>./server UDP
[WELCOME] Welcome to the server!!!
*-----*
| SERVER HELP MENU |
*-----*
| AUTHOR: !EEExp3rt |
*-----*
| CMDS | USAGE |
*-----*
| init | Initialize the server |
*-----*
| run | |
*-----* Start the server and run |
| start | |
*-----*
| stop | Stop the server |
*-----*
| broadcast | Broadcast weather warning |
*-----*
| help | Show this help message |
*-----*
| exit | |
*-----* Exit the server program |
| quit | |
*-----*
```

- 客户端选择连接功能时，显示内容截图（TCP 模式和 UDP 模式各一组）。

客户端：

```
(base) lqy@LAPTOP-SDV26TG8:~/Z/ZJU-Computer-Network-Socket>./client
Please select the protocol you want to use
1. TCP
2. UDP
1

===== SERVERS =====
No server connected.
=====

===== MENU =====
1. Connect to Server
2. Exit
=====
[Client] Enter your cmd: 1
Please enter the server IP address: 127.0.0.1
Please enter the server port: 3373
Connected to server successfully. Server ID: 1

===== SERVERS =====
Allocated Server ID:
Server ID: 1 status: Connected
=====
```

```
(base) lqy@LAPTOP-SDV26TG8:~/Z/ZJU-Computer-Network-Socket>./client
Please select the protocol you want to use
1. TCP
2. UDP
2

===== SERVERS =====
No server connected.
=====

===== MENU =====
1. Connect to Server
2. Exit
=====
[Client] Enter your cmd: 1
Please enter the server IP address: 127.0.0.1
Please enter the server port: 3373
Please enter the local port: 8080
Connecting to server...
```

服务端:

```
(base) lqy@LAPTOP-SDV26TG8:~/Z/ZJU-Computer-Network-Socket>./server TCP
[WELCOME] Welcome to the server!!!
*-----*
|       SERVER HELP MENU      |
*-----*
|       AUTHOR: !EEExp3rt     |
*-----*
|   CMDS    |      USAGE      |
*-----*-----*
|   init    | Initialize the server|
*-----*-----*
|   run     |                   |
*-----*-----* Start the server and run
|   start   |                   |
*-----*-----*
|   stop    | Stop the server   |
*-----*-----*
| broadcast| Broadcast weather warning|
*-----*-----*
|   help    | Show this help message |
*-----*-----*
|   exit    |                   |
*-----*-----* Exit the server program
|   quit   |                   |
*-----*-----*
init
[Server] Server is initializing...
[Server] Get server socket: 3.
[Server] Set server waiting timeout: 10.000000.
[Server] Bind server to IP: 127.0.0.1, port: 3373.
[Server] Server initialization completed.
run
[Server] Start listening on 127.0.0.1:3373.
[Server] The server is now running...
[Server] New client (socketFd = 4) from 127.0.0.1:37282 connected, assigned id = 0.
```

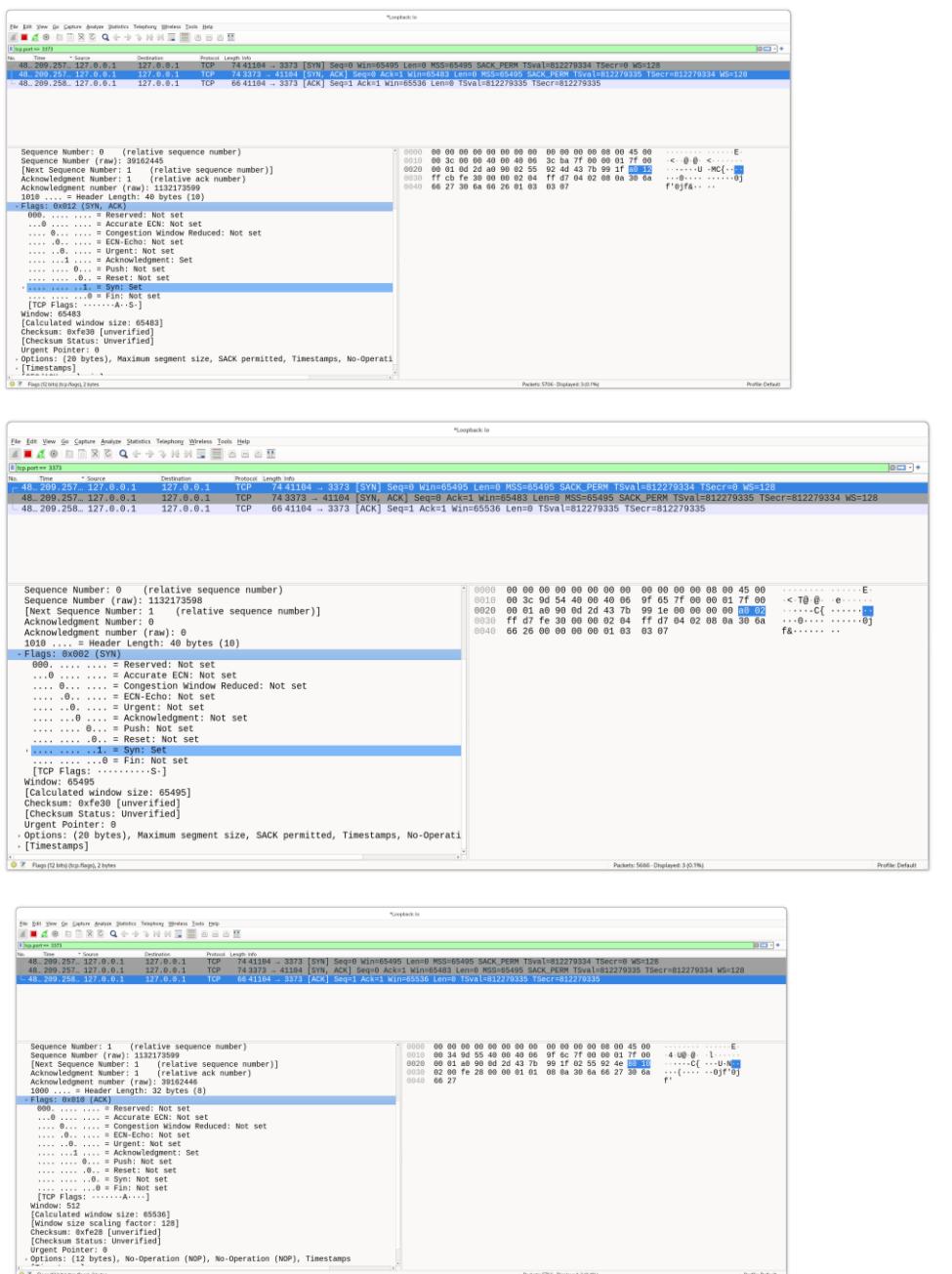
```
(base) lqy@LAPTOP-SDV26TG8:~/Z/ZJU-Computer-Network-Socket>./server UDP
[WELCOME] Welcome to the server!!!
*-----*
|       SERVER HELP MENU      |
*-----*
|       AUTHOR: !EEExp3rt     |
*-----*
|   CMDS    |      USAGE      |
*-----*-----*
|   init    | Initialize the server|
*-----*-----*
|   run     |                   |
*-----*-----* Start the server and run
|   start   |                   |
*-----*-----*
|   stop    | Stop the server   |
*-----*-----*
| broadcast| Broadcast weather warning|
*-----*-----*
|   help    | Show this help message |
*-----*-----*
|   exit    |                   |
*-----*-----* Exit the server program
|   quit   |                   |
*-----*-----*
init
[Server] Server is initializing...
[Server] Get server socket: 3.
[Server] Set server waiting timeout: 10.000000.
[Server] Bind server to IP: 127.0.0.1, port: 3373.
[Server] Server initialization completed.
run
[Server] The server is now running...
[Server] Client from 127.0.0.1:8080 connected.
```

Wireshark 抓取的数据包截图（TCP 模式下，找到 TCP 三次握手建立连接的数据包，展开 TCP

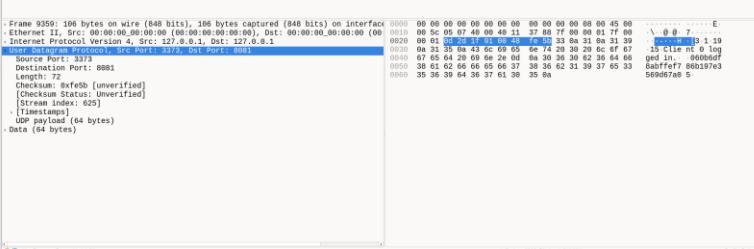
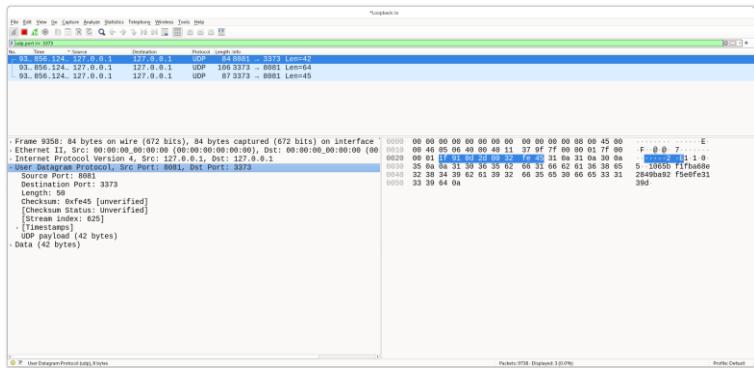
协议层的 Flags 字段，分别标记三个数据包的 SYN 标志位和 ACK 标志位。UDP 模式下，展开 UDP 的数据字段，标记双方发送的连接请求和响应数据）：

客户端：

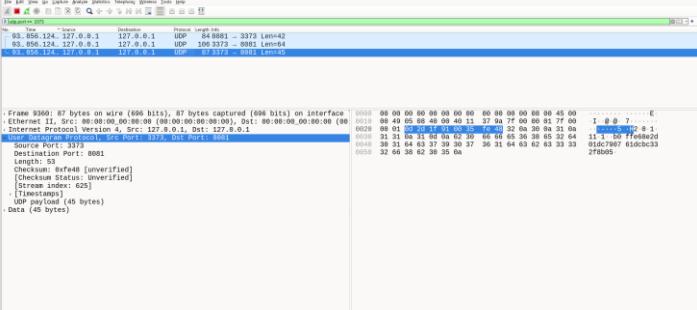
服务端：



UDP:



Packets 3714 - Displayed 3 (0.0%)



Packets 3840 - Displayed 3 (0.0%)

- 客户端选择获取城市名称功能时，显示内容截图。

客户端：

```
===== MENU =====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
=====
[Client] Enter your cmd: 3
Please enter the server ID you want to get city name: 1
Please enter the area code: 12
Request sent successfully.
[Server] City name: Jinan
```

服务端:

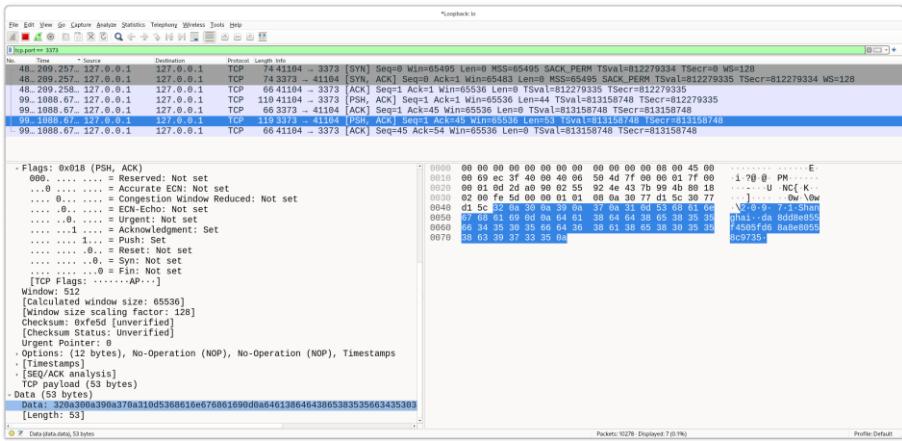
```
*-----* Show this help message *-----*
|   exit  |
*-----* Exit the server program *
|   quit  |
*-----*-----*

init
[Server] Server is initializing...
[Server] Get server socket: 3.
[Server] Set server waiting timeout: 10.000000.
[Server] Bind server to IP: 127.0.0.1, port: 3373.
[Server] Server initialization completed.
run
[Server] Start listening on 127.0.0.1:3373.
[Server] The server is now running...
[Server] New client (socketFd = 4) from 127.0.0.1:37612 connected, assigned id = 0.
[Server] Client 0 requested city name for ID 12.
[]
```

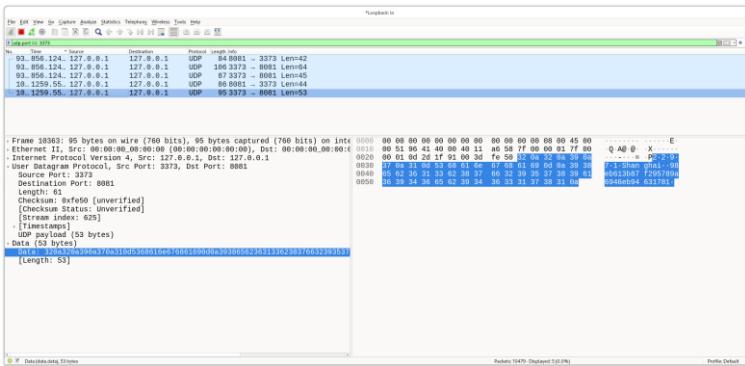
Wireshark 抓取的数据包截图（展开应用层数据包，标记请求、响应类型、返回的城市名称数据对应的位置、校验字段，TCP 模式和 UDP 模式各一组）：

客户端:

服务端:



UDP:



- 客户端选择获取气象信息功能时，显示内容截图。

客户端：

```
=====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
=====
[Client] Enter your cmd: 4
Please enter the server ID you want to get weather info: 1
Please enter the area code: 12
Please enter the date (YYYY-MM-DD): 2024-11-05
Request sent successfully.
[Server] Weather info in Jinan: 29 degree, Sunny, 41 humidity, 41 wind speed, East wind direction.
```

服务端：

```

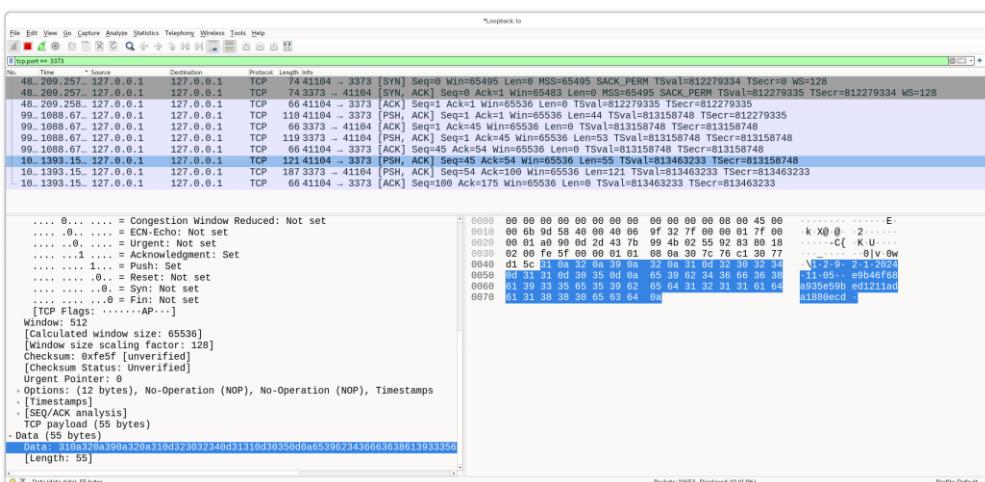
| quit |
*-----*
init
[Server] Server is initializing...
[Server] Get server socket: 3.
[Server] Set server waiting timeout: 10.000000.
[Server] Bind server to IP: 127.0.0.1, port: 3373.
[Server] Server initialization completed.
run
[Server] Start listening on 127.0.0.1:3373.
[Server] The server is now running...
[Server] New client (socketFd = 4) from 127.0.0.1:37612 connected, assigned id = 0.
[Server] Client 0 requested city name for ID 12.
[Server] Client 0 requested weather information for city ID 12 and date 2024-11-05.

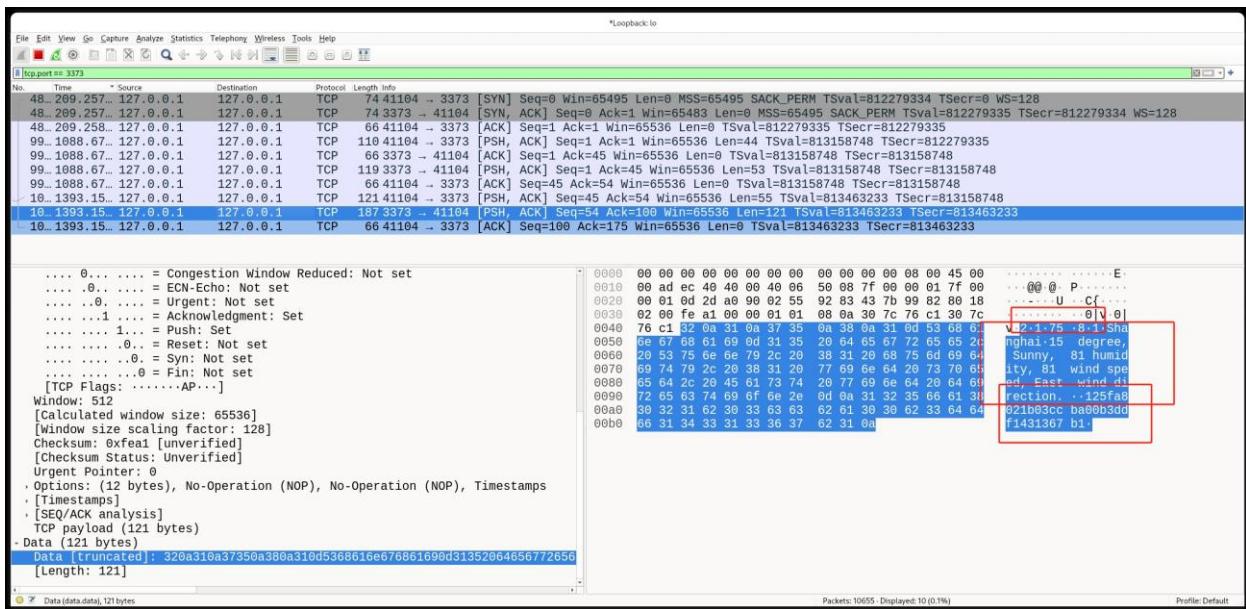
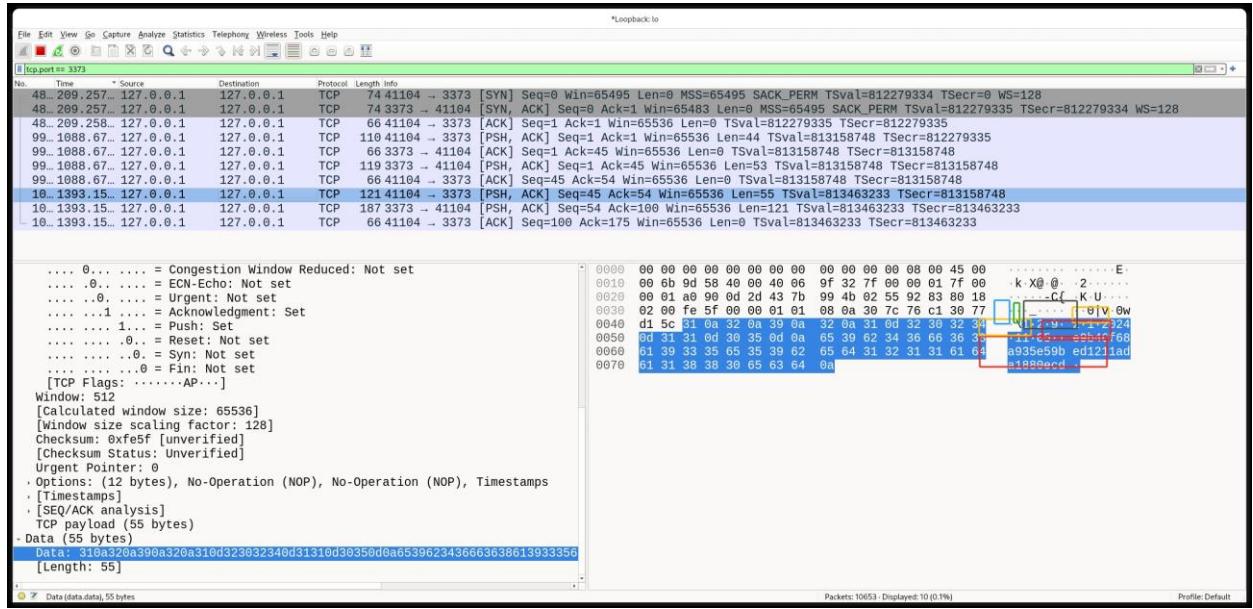
```

Wireshark 抓取的数据包截图（展开应用层数据包，标记请求、响应类型、返回的气象信息数据对应的位置、校验字段，TCP 模式和 UDP 模式各一组）：

客户端：

服务端：





- 客户端选择获取客户端列表功能时，显示内容截图

客户端：

```
===== MENU =====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
=====
[Client] Enter your cmd: 5
Please enter the server ID you want to get client list: 1
Request sent successfully.
[Server] Client list number: 1
0,127.0.0.1:37612
```

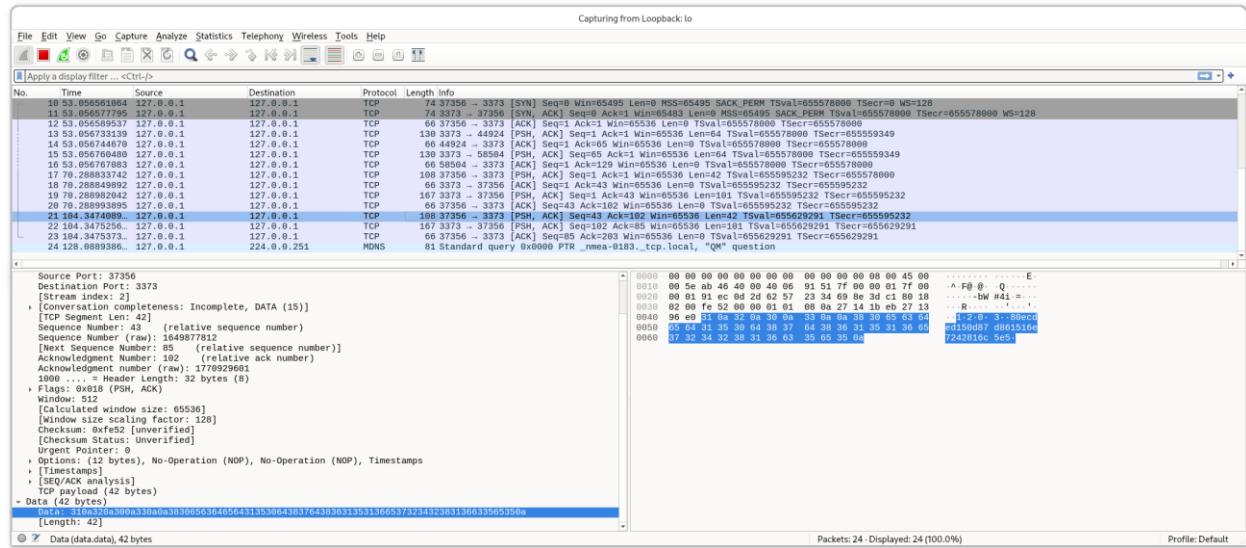
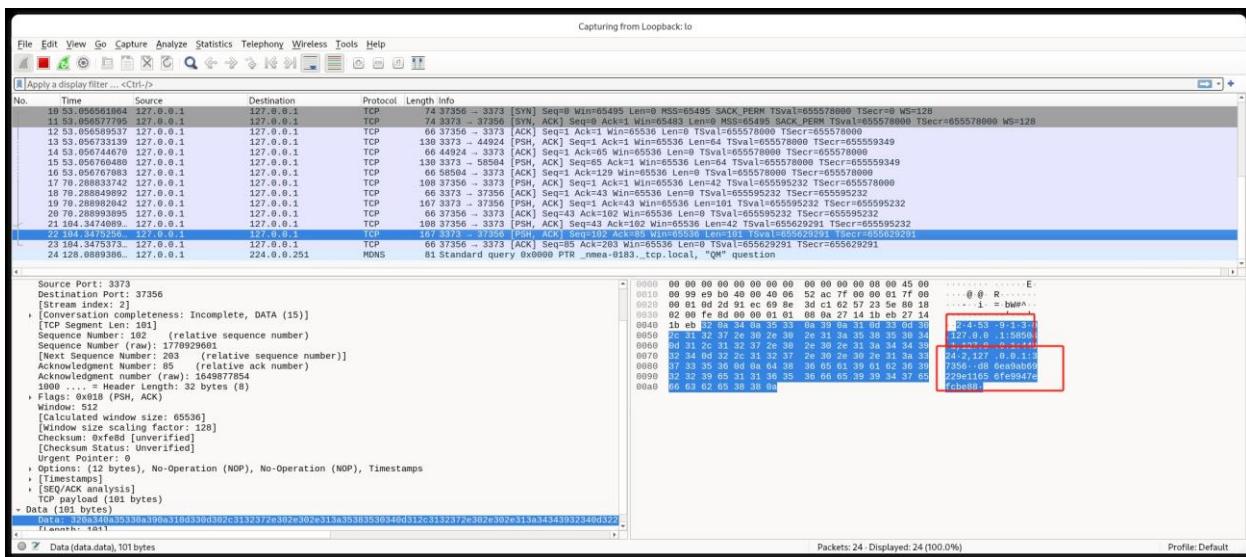
服务端:

```
| help   | Show this help message   |
*-----*-----*
| exit   |                               |
*-----* Exit the server program   |
| quit   |                               |
*-----*-----*
init
[Server] Server is initializing...
[Server] Get server socket: 3.
[Server] Set server waiting timeout: 10.000000.
[Server] Bind server to IP: 127.0.0.1, port: 3373.
[Server] Server initialization completed.
run
[Server] Start listening on 127.0.0.1:3373.
[Server] The server is now running...
[Server] New client (socketFd = 4) from 127.0.0.1:37612 connected, assigned id = 0.
[Server] Client 0 requested city name for ID 12.
[Server] Client 0 requested weather information for city ID 12 and date 2024-11-05.
[Server] Client 0 requested client list.
[]
```

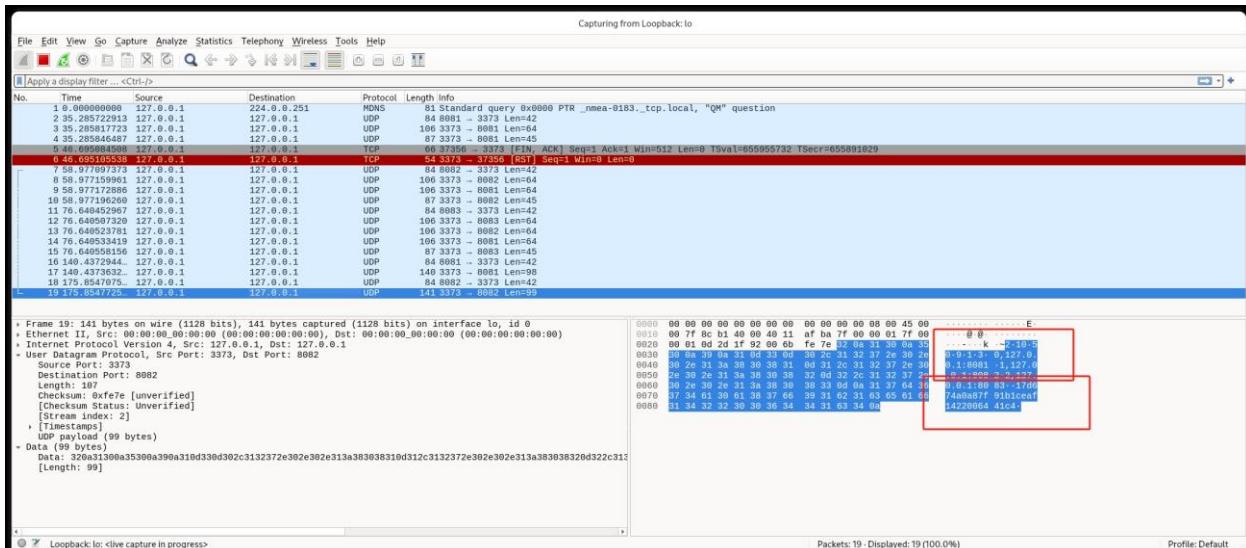
Wireshark 抓取的数据包截图（展开应用层数据包，标记请求、响应类型、返回的客户端列表数据对应的位置、校验字段，TCP 模式和 UDP 模式各一组）：

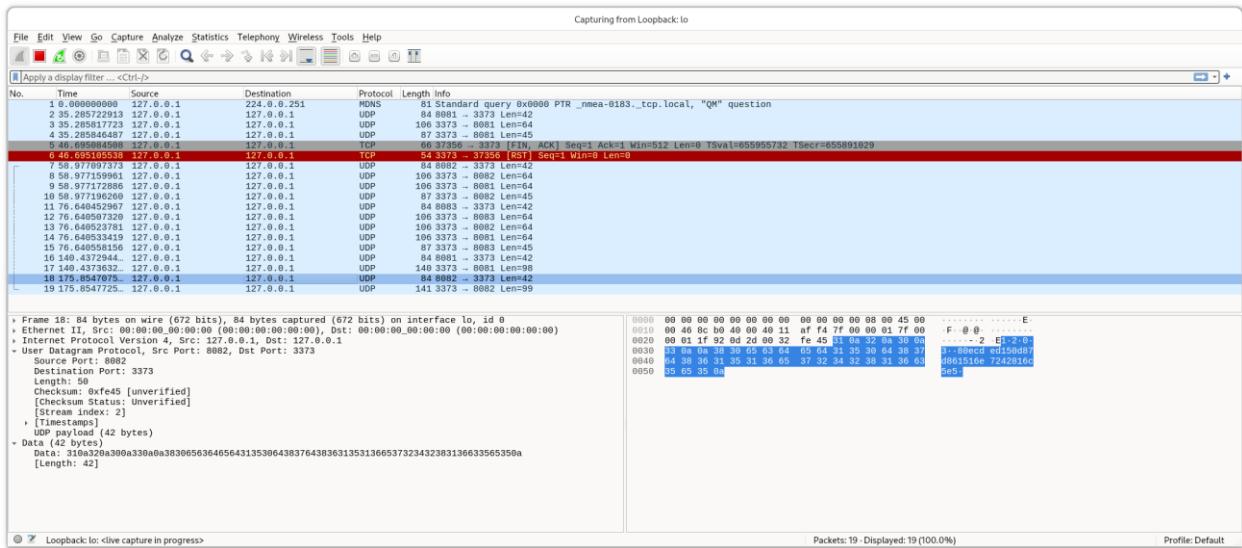
客户端:

服务端:



UDP:





- 客户端选择发送消息功能时，显示内容截图。

发送消息的客户端：

```
=====
 1. Connect to Server
 2. Disconnect from Server
 3. Get City Name
 4. Get Weather Information
 5. Get Client List
 6. Send Message
 7. Exit
=====

[Client] Enter your cmd: 6
Please enter the server ID you want to send message: 1
Please enter the receiver ID: 0
Please enter the message: HELLO WORLD
Request sent successfully.
[Server] Message sent successfully.
```

服务端：

```
[Server] New client (socketFd = 5) from 127.0.0.1:38242 connected, assigned id = 1.
[Server] Client 1 requested to send message to client 0.
```

接收消息的客户端：

```

===== SERVERS =====
Allocated Server ID:
Server ID: 1 status: Connected
=====

===== MENU =====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
=====

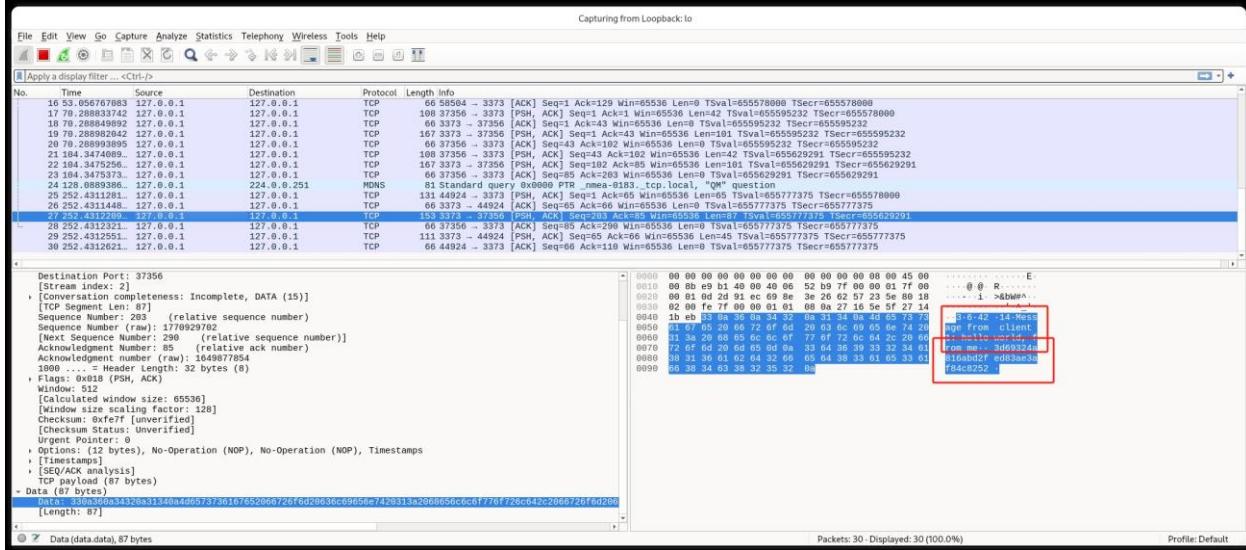
[Client] Enter your cmd: [Server] Client 1 logged in.

[Server] Message: Message from client 1: HELLO WORLD

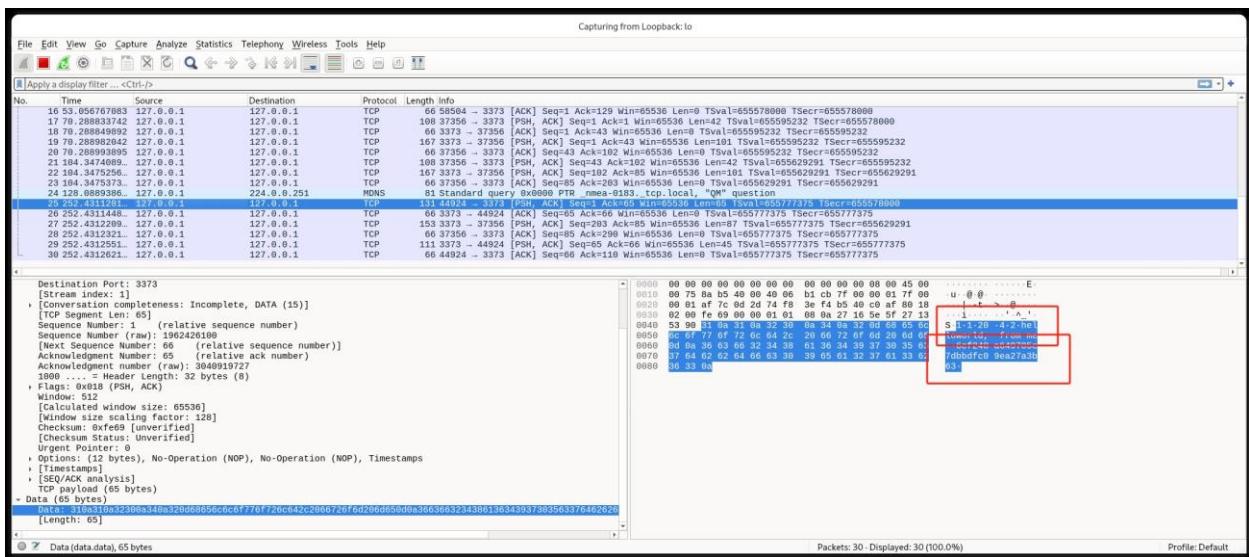
```

Wireshark 抓取的数据包截图(对各字段进行标记、含校验字段, TCP 模式和 UDP 模式各一组):

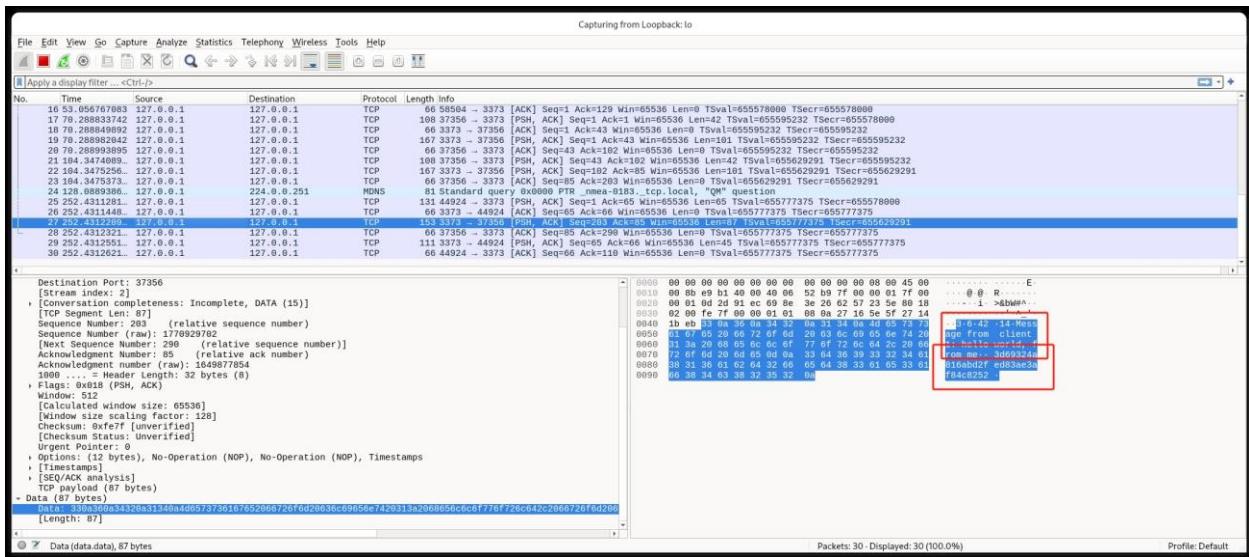
发送消息的客户端:



服务端:

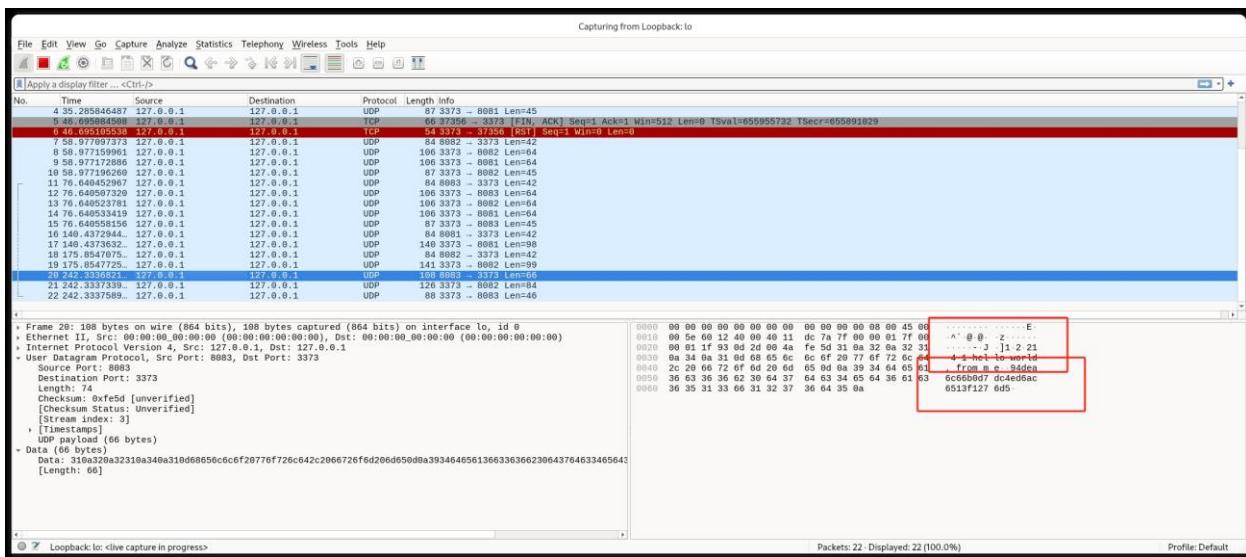


接收消息的客户端:

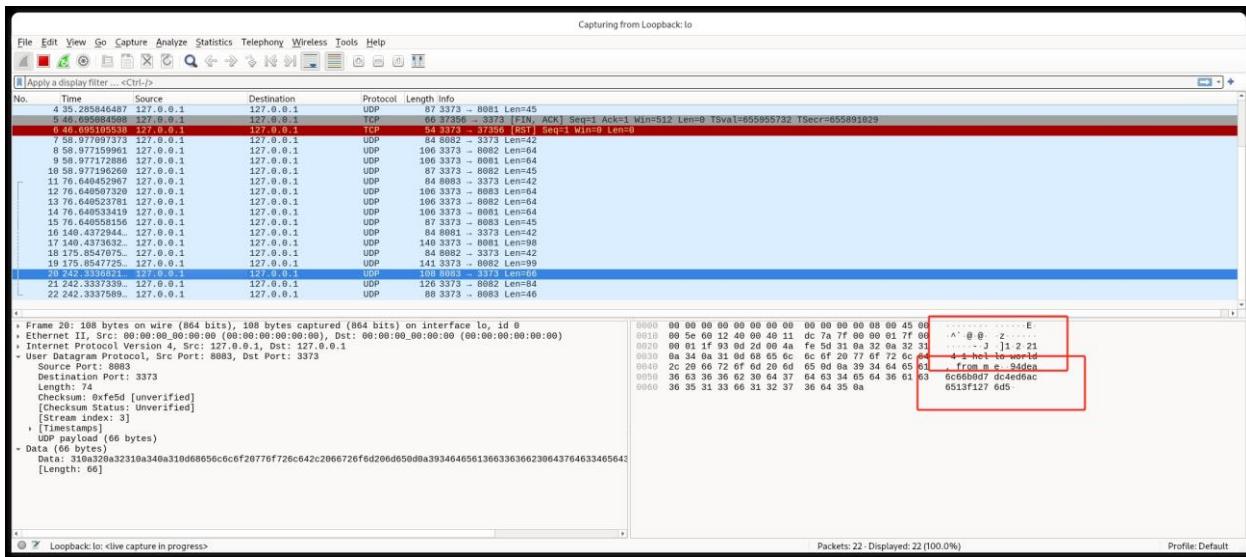


UDP:

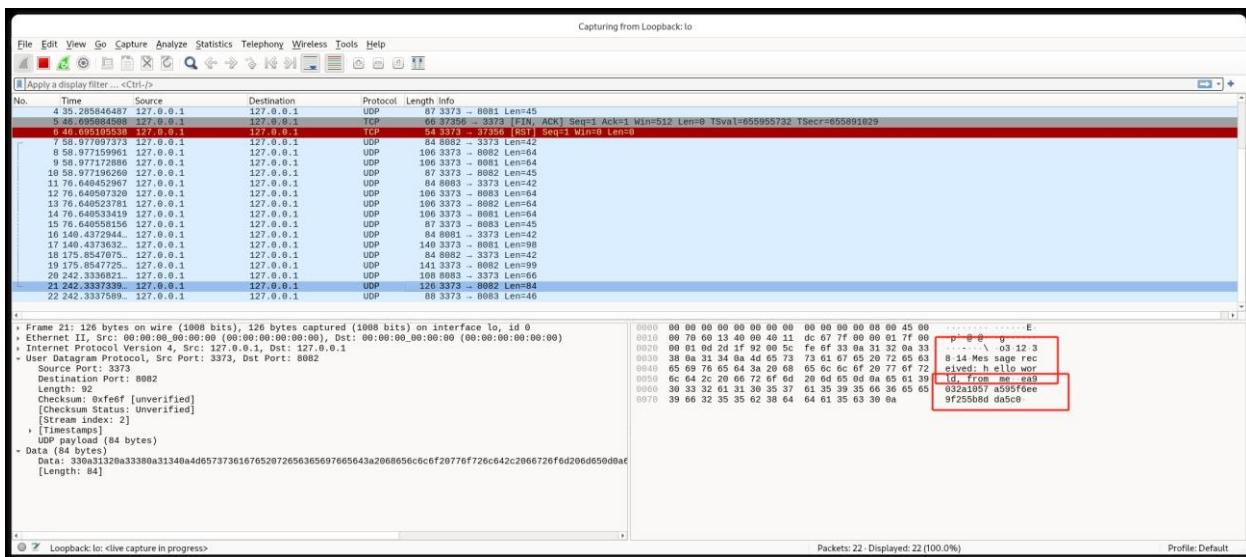
发送消息的客户端:



服务端：



接收消息的客户端：



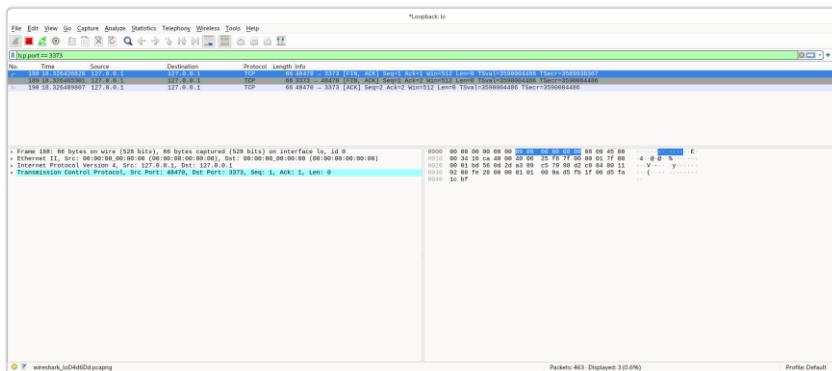
4) 以下为异常和批量测试，均以 TCP 模式运行。同时记录客户端和服务端的截图。

- 拔掉客户端的网线，然后退出客户端程序。观察客户端的 TCP 连接状态，并使用 Wireshark 观察客户端是否发出了 TCP 连接释放的消息。同时观察服务端的 TCP 连接状态在较长时间内（10 分钟以上）是否发生变化。

拔掉客户端网线时，客户端的 TCP 连接状态保持 ESTABLISHED，因为 TCP 连接不会立刻察觉到网络断开

退出客户端程序后，操作系统回收了客户端的 socket，并且会向服务端发送一个 TCP 连接释放的请求，主动告知服务端请求断开，所以会发出 TCP 连接释放的消息

服务端收到这个请求之后会响应，也发一个包表示自己准备关闭连接，但是由于客户端已经下线，服务端收不到 ACK 包，所以所以基于 TCP 协议的重传机制会多次重试发送 ACK 包，直到超过上限后才最终确认连接已经关闭，因此在较长时间内 TCP 的连接状态不会发生变化



- 再次连上客户端的网线，重新运行客户端程序。选择连接功能，连上后选择获取客户端列表功能，查看之前异常退出的连接是否还在。选择给这个之前异常退出的客户端连接发送消息，出现了什么情况？

如果此时重新连接客户端，尝试查看客户端列表，由于之前的连接状态可能存活较长时间（几分钟），因此此时可能可以看到这些

连接状态

此时，尝试向这些已经失效的连接发送消息，服务端会发现这些连接依然处于 ESTABLISH 状态，但由于实际的客户端已经不再连接，因此消息不会成功发送给客户端

- 部署运行多个服务端实例（配置的城市信息和气象信息区别开），一个客户端同时连接多个服务端，向不同服务器请求不同的城市名称、气象信息时，客户端和服务端的运行截图。

```

@ (base) lqy@LAPTOP-SDV26TGB:~/ZJU-Computer-Network-Sock
et> ./server      (main) 01:42
Usage: ./server <server_type> [ip] [port] [max_connectio
ns] [message_buffer_size] [wait_timeout]
@ (base) lqy@LAPTOP-SDV26TGB:~/ZJU-Computer-Network-Sock
et[1]> ./server TCP 127.0.0.1 33
73
[WELCOME] Welcome to the server!!!
*-----*
| SERVER HELP MENU |
*-----*
| AUTHOR: !EEExp3rt |
*-----*
| CMDS | USAGE |
*-----*
| init | Initialize the server |
| run | |
| start | |
| stop | Stop the server |
| broadcast | Broadcast weather warning |
| help | Show this help message |
| exit | |
| quit | |
*-----* Exit the server program |
init
[Server] Server is initializing...
[Server] Get server socket: 3.
[Server] Set server waiting timeout: 10.000000.
[Server] Bind server to IP: 127.0.0.1, port: 3373.
[Server] Server initialization completed.
run
[Server] Start listening on 127.0.0.1:3373.
[Server] The server is now running...
[Server] New client (socketFd = 4) from 127.0.0.1:50430
connected, assigned id = 0.
[Server] Client 0 requested city name for ID 10.

```



```

@ ...e) lqy@LAPTOP-SDV26TGB:~/ZJU-Computer-Network-Sock
et./server TCP 127.0.0.1 3374
[WELCOME] Welcome to the server!!!
*-----*
| SERVER HELP MENU |
*-----*
| AUTHOR: !EEExp3rt |
*-----*
| CMDS | USAGE |
*-----*
| init | Initialize the server |
| run | |
| start | |
| stop | Stop the server |
| broadcast | Broadcast weather warning |
| help | Show this help message |
| exit | |
| quit | |
*-----* Exit the server program |
init
[Server] Server is initializing...
[Server] Get server socket: 3.
[Server] Set server waiting timeout: 10.000000.
[Server] Bind server to IP: 127.0.0.1, port: 3374.
[Server] Server initialization completed.
run
[Server] Start listening on 127.0.0.1:3374.
[Server] The server is now running...
[Server] New client (socketFd = 4) from 127.0.0.1:41832
connected, assigned id = 0.
[Server] Client 0 requested city name for ID 2.

```



```

@ (base) lqy@LAPTOP-SDV26TGB:~/ZJU-Computer-Network-Sock
et[1]> ./server TCP 127.0.0.1 33
73
[WELCOME] Welcome to the server!!!
*-----*
| SERVER HELP MENU |
*-----*
| AUTHOR: !EEExp3rt |
*-----*
| CMDS | USAGE |
*-----*
| init | Initialize the server |
| run | |
| start | |
| stop | Stop the server |
| broadcast | Broadcast weather warning |
| help | Show this help message |
| exit | |
| quit | |
*-----* Exit the server program |
init
[Server] Server is initializing...
[Server] Get server socket: 3.
[Server] Set server waiting timeout: 10.000000.
[Server] Bind server to IP: 127.0.0.1, port: 3373.
[Server] Server initialization completed.
run
[Server] Start listening on 127.0.0.1:3373.
[Server] The server is now running...
[Server] New client (socketFd = 4) from 127.0.0.1:50430
connected, assigned id = 0.
[Server] Client 0 requested city name for ID 10.
[Server] Client 0 requested weather information for city
ID 4 and date 2024-11-07.

```



```

@ ...e) lqy@LAPTOP-SDV26TGB:~/ZJU-Computer-Network-Sock
et./server TCP 127.0.0.1 3374
[WELCOME] Welcome to the server!!!
*-----*
| SERVER HELP MENU |
*-----*
| AUTHOR: !EEExp3rt |
*-----*
| CMDS | USAGE |
*-----*
| init | Initialize the server |
| run | |
| start | |
| stop | Stop the server |
| broadcast | Broadcast weather warning |
| help | Show this help message |
| exit | |
| quit | |
*-----* Exit the server program |
init
[Server] Server is initializing...
[Server] Get server socket: 3.
[Server] Set server waiting timeout: 10.000000.
[Server] Bind server to IP: 127.0.0.1, port: 3374.
[Server] Server initialization completed.
run
[Server] Start listening on 127.0.0.1:3374.
[Server] The server is now running...
[Server] New client (socketFd = 4) from 127.0.0.1:41832
connected, assigned id = 0.
[Server] Client 0 requested city name for ID 2.

```



```

Please enter the server ID you want to get city name: 1
Please enter the area code: 10
Request sent successfully.
[Server] City name: Harbin
===== SERVERS =====
Allocated Server ID:
Server ID: 1 status: Connected
Server ID: 2 status: Connected
===== MENU =====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
8. Send Batch Test Data
===== [Client] Enter your cmd: 3 =====
Please enter the server ID you want to get city name: 2
Please enter the area code: 2
Request sent successfully.
[Server] City name: Guangzhou
===== SERVERS =====
Allocated Server ID:
Server ID: 1 status: Connected
Server ID: 2 status: Connected
===== MENU =====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
8. Send Batch Test Data
===== [Client] Enter your cmd: 4 =====
Please enter the server ID you want to get weather info: 2
Please enter the area code: 8
Please enter the date (YYYY-MM-DD): 2024-11-11
Request sent successfully.
[ERROR] No weather information for date 2024-11-11.
===== SERVERS =====
Allocated Server ID:
Server ID: 1 status: Connected
Server ID: 2 status: Connected
===== MENU =====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
8. Send Batch Test Data

```

- 修改客户端，增加一个批量获取城市名称功能，用户选择 1 次，程序内自动发送 100 次请求（序号不同）。验证服务端是否正常处理了 100 次请求，截取客户端收到的响应（通过程序计数一下是否有 100 个响应回来），并使用 Wireshark 抓取数据包，观察实际发出的数据包个数。

```

[./bin/server ZJU-Computer-Network-Socket X]
[Server] Server initialization completed.
run
[Server] Start listening on 10.162.150.48:3373.
[Server] The server is now running...
[Server] New client (socketFd = 4) from 10.162.150.48:45249 connected, assigned id = 0.
[Server] Client 0 requested city name for ID 0.
[Server] Client 0 requested city name for ID 1.
[Server] Client 0 requested city name for ID 2.
[Server] Client 0 requested city name for ID 3.
[Server] Client 0 requested city name for ID 4.
[Server] Client 0 requested city name for ID 5.
[Server] Client 0 requested city name for ID 6.
[Server] Client 0 requested city name for ID 7.
[Server] Client 0 requested city name for ID 8.
[Server] Client 0 requested city name for ID 9.
[Server] Client 0 requested city name for ID 10.
[Server] Client 0 requested city name for ID 11.
[Server] Client 0 requested city name for ID 12.
[Server] Client 0 requested city name for ID 13.
[Server] Client 0 requested city name for ID 14.
[Server] Client 0 requested city name for ID 15.
[Server] Client 0 requested city name for ID 16.
[Server] Client 0 requested city name for ID 17.
[Server] Client 0 requested city name for ID 18.
[Server] Client 0 requested city name for ID 19.
[Server] Client 0 requested city name for ID 20.
[Server] Client 0 requested city name for ID 21.
[Server] Client 0 requested city name for ID 22.
[Server] Client 0 requested city name for ID 23.
[Server] Client 0 requested city name for ID 24.
[Server] Client 0 requested city name for ID 25.
[Server] Client 0 requested city name for ID 26.
[Server] Client 0 requested city name for ID 27.
[Server] Client 0 requested city name for ID 28.
[Server] Client 0 requested city name for ID 29.
[Server] Client 0 requested city name for ID 30.
[Server] Client 0 requested city name for ID 31.
[Server] Client 0 requested city name for ID 32.
[Server] Client 0 requested city name for ID 33.
[Server] Client 0 requested city name for ID 34.
[Server] Client 0 requested city name for ID 35.
[Server] Client 0 requested city name for ID 36.
[Server] Client 0 requested city name for ID 37.
[Server] Client 0 requested city name for ID 38.

[./bin/client ZJU-Computer-Network-Socket X]
=====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
8. Send Batch Test Data
=====

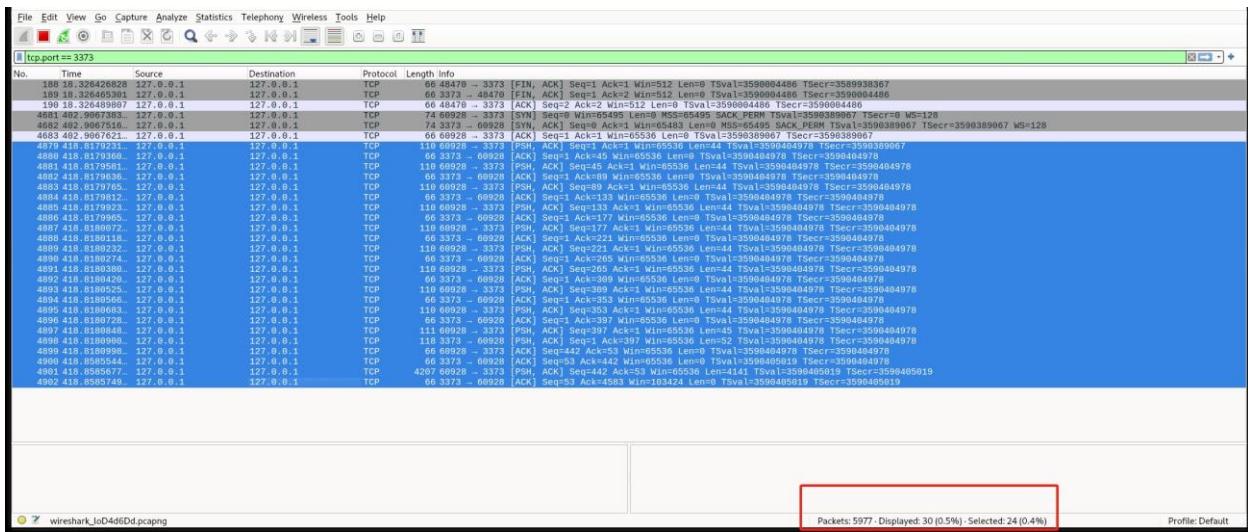
[Client] Enter your cmd: 8
Please enter the server ID you want to send message: 1
Please enter the count of test packet: 100
Request sent successfully.
[Server] City name: Beijing
Request sent successfully.
[Server] City name: Shanghai
Request sent successfully.
[Server] City name: Guangzhou
Request sent successfully.
[Server] City name: Shenzhen
Request sent successfully.
[Server] City name: Hangzhou
Request sent successfully.
[Server] City name: Nanjing
Request sent successfully.
[Server] City name: Chengdu
Request sent successfully.
[Server] City name: Wuhan
Request sent successfully.
[Server] City name: Chongqing
Request sent successfully.
[Server] City name: Suzhou
Request sent successfully.
[Server] City name: Harbin
Request sent successfully.
[Server] City name: Qingdao
Request sent successfully.
[Server] City name: Jinan
Request sent successfully.
[Server] City name: Tianjin
Request sent successfully.
[Server] City name: Changchun

[./bin/server ZJU-Computer-Network-Socket X]
[Server] Client 0 requested city name for ID 57.
[Server] Client 0 requested city name for ID 58.
[Server] Client 0 requested city name for ID 59.
[Server] Client 0 requested city name for ID 60.
[Server] Client 0 requested city name for ID 61.
[Server] Client 0 requested city name for ID 62.
[Server] Client 0 requested city name for ID 63.
[Server] Client 0 requested city name for ID 64.
[Server] Client 0 requested city name for ID 65.
[Server] Client 0 requested city name for ID 66.
[Server] Client 0 requested city name for ID 67.
[Server] Client 0 requested city name for ID 68.
[Server] Client 0 requested city name for ID 69.
[Server] Client 0 requested city name for ID 70.
[Server] Client 0 requested city name for ID 71.
[Server] Client 0 requested city name for ID 72.
[Server] Client 0 requested city name for ID 73.
[Server] Client 0 requested city name for ID 74.
[Server] Client 0 requested city name for ID 75.
[Server] Client 0 requested city name for ID 76.
[Server] Client 0 requested city name for ID 77.
[Server] Client 0 requested city name for ID 78.
[Server] Client 0 requested city name for ID 79.
[Server] Client 0 requested city name for ID 80.
[Server] Client 0 requested city name for ID 81.
[Server] Client 0 requested city name for ID 82.
[Server] Client 0 requested city name for ID 83.
[Server] Client 0 requested city name for ID 84.
[Server] Client 0 requested city name for ID 85.
[Server] Client 0 requested city name for ID 86.
[Server] Client 0 requested city name for ID 87.
[Server] Client 0 requested city name for ID 88.
[Server] Client 0 requested city name for ID 89.
[Server] Client 0 requested city name for ID 90.
[Server] Client 0 requested city name for ID 91.
[Server] Client 0 requested city name for ID 92.
[Server] Client 0 requested city name for ID 93.
[Server] Client 0 requested city name for ID 94.
[Server] Client 0 requested city name for ID 95.
[Server] Client 0 requested city name for ID 96.
[Server] Client 0 requested city name for ID 97.
[Server] Client 0 requested city name for ID 98.
[Server] Client 0 requested city name for ID 99.

===== SERVERS =====
Allocated Server ID:
Server ID: 1 status: Connected
===== MENU =====

```

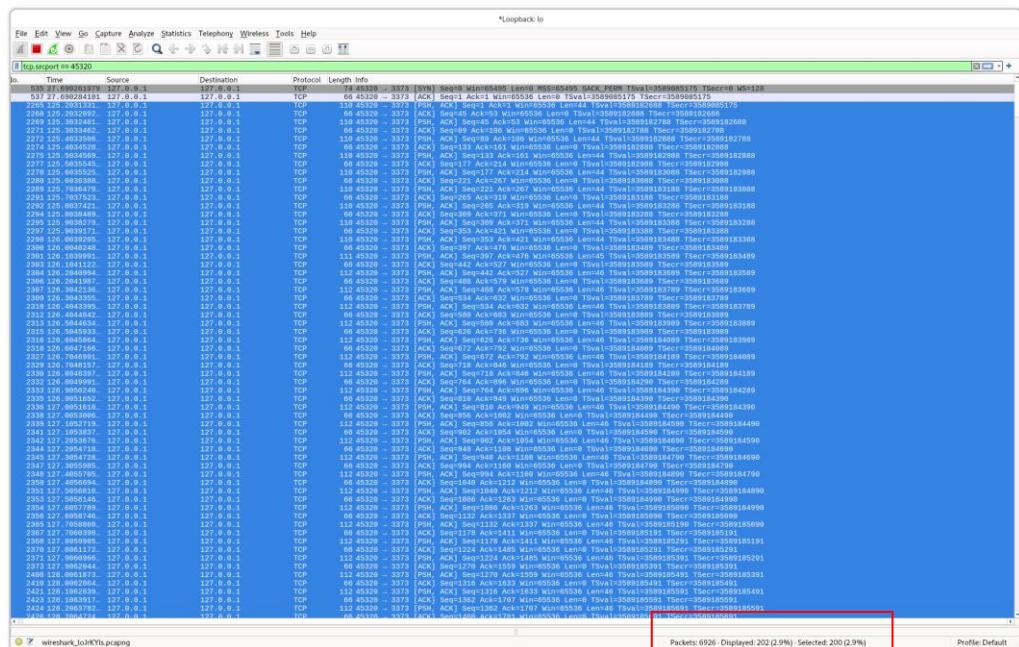
这里红色部分是因为请求的城市信息不在配置文件中，所以服务端向客户端返回错误信息，表明此次请求没有正确的城市名称



一开始客户端连续调用 send 发送 100 个请求包时，服务端实际收到的包少于 100 个，出现了问题，而引起这个问题的原因是调用 send 时操作

系统会先把数据送到 TCP 缓冲区，等合适的时机才发送数据，因此后来会收到少于 100 个包，导致了错误

修改方法是客户端添加一个等待操作，每隔 0.1 秒调用一次 send，以区分不同的包



- 多个客户端同时连接同一个服务端，同时发送批量获取城市名称请求（每个客户端程序内自动连续调用 100 次请求，序号不同），服务端和客户端的运行截图。

```

[Server] Client 0 requested city name for ID 76.
[Server] Client 1 requested city name for ID 81.
[Server] Client 0 requested city name for ID 77.
[Server] Client 1 requested city name for ID 82.
[Server] Client 0 requested city name for ID 78.
[Server] Client 1 requested city name for ID 83.
[Server] Client 0 requested city name for ID 79.
[Server] Client 1 requested city name for ID 84.
[Server] Client 0 requested city name for ID 80.
[Server] Client 1 requested city name for ID 85.
[Server] Client 0 requested city name for ID 81.
[Server] Client 1 requested city name for ID 86.
[Server] Client 0 requested city name for ID 82.
[Server] Client 1 requested city name for ID 87.
[Server] Client 0 requested city name for ID 83.
[Server] Client 1 requested city name for ID 88.
[Server] Client 0 requested city name for ID 84.
[Server] Client 1 requested city name for ID 89.
[Server] Client 0 requested city name for ID 85.
[Server] Client 1 requested city name for ID 90.
[Server] Client 0 requested city name for ID 91.
[Server] Client 1 requested city name for ID 92.
[Server] Client 0 requested city name for ID 93.
[Server] Client 1 requested city name for ID 94.
[Server] Client 0 requested city name for ID 95.
[Server] Client 1 requested city name for ID 96.
[Server] Client 0 requested city name for ID 97.
[Server] Client 1 requested city name for ID 98.
[Server] Client 0 requested city name for ID 99.
[Server] Client 0 requested city name for ID 99.

===== SERVERS =====
Allocated Server ID:
Server ID: 1 status: Connected
===== MENU =====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
8. Send Batch Test Data
===== MENU =====
1. Connect to Server
2. Disconnect from Server
3. Get City Name
4. Get Weather Information
5. Get Client List
6. Send Message
7. Exit
8. Send Batch Test Data
[Client] Enter your cmd: 

```

六、实验结果与分析

- TCP 模式下，客户端是否需要调用 bind 操作？如果不绑定，它的源端口是如何产生的？每一次调用 connect 时客户端的端口是否都保持不变？UDP 模式下呢？

在 TCP 模式下，客户端一般不需要显式调用 bind，直接调用 connect

调用 connect 时，操作系统会为客户端套接字随机分配一个未占用的临时端口，使用客户端 IP 地址和这个临时端口作为源地址，建立与服务器的连接

在实验中，我们重新创建套接字并调用 connect，可观察到端口改变，因为操作系统会随机分配一个新的端口。但如果客户端在同一个会话中复用同一个套接字（没有关闭它），则可以保持端口不变。

在 UDP 模式下，客户端也可以选择不调用 bind。如果没有绑定本地端口，当首次调用 sendto 或 connect 时，操作系统会自动分配一个临时端口。如果显式调用了 bind，则 UDP 的源端口固定为绑定的端口，不会随请求变化，并且如果客户端不关闭套接字，那么在整个生命周期中，这个分配的源端口也会保持不变。

- 假设在服务端调用 listen 和调用 accept 之间设了一个调试断点，暂停在此断点时，此时客户端调用 connect 后是否马上能连接成功？

客户端调用 connect 后不会马上连接成功

在 TCP 连接建立的过程中，connect 调用会触发 TCP 三次握手。

但因为服务器在断点处暂停，没有调用 accept，此时客户端的 SYN 包会发出并等待服务器的响应。

客户端的 connect 调用会在等待服务器响应过程中超时并重试几次（视系统设置和重试机制而定）。如果在等待时间内，服务器没有继续执行 accept，则客户端会最终抛出连接超时错误

- 连续快速 send 多次数据后，通过 Wireshark 抓包看到的发送的 Tcp Segment 次数是否和 send 的次数完全一致？

实验中我们发现 send() 系统调用的次数和实际发送的 TCP segment 数量并不总是完全一致。在进行批量测试

时，客户端调用了 5 次 send，按理说会到达 5 个 packet，有从服务器反馈的信息和 wireshark 抓包情况来看，5 分数据被合成为了一个 TCP segment。

查阅资料后发现，TCP 协议中有一个称为 Nagle 算法（Nagle's Algorithm）的机制。它会尝试将多个小的写操作（即多次调用 send()）合并成一个大的数据包，以提高网络传输效率。Nagle 算法会让数据先在本地缓冲区积累，直到缓冲区足够大或收到确认（ACK）信号后，才会发送出去。因此，如果快速连续调用 send()，可能会看到多个调用合并为一个或几个 TCP segment。

- 服务器在同一个端口接收多个客户端的数据，如何能区分数据包是属于哪个客户端的？

TCP 连接中，使用 Socket 区分数据包属于哪个客户端

每个已连接套接字对应一个唯一的四元组标识符，即（服务器 IP，服务器端口，客户端 IP，客户端端口），这样可以唯一标识连接的每一个客户端。服务器通过每个已连接套接字接收和发送数据，因此可以知道数据来自哪个客户端

- 客户端主动断开连接后，当时的 TCP 连接状态是什么？这个状态保持了多久？（可以使用 netstat -an 查看）

客户端主动断开连接时，连接状态通常会经历 FIN_WAIT_1、FIN_WAIT_2 和 TIME_WAIT 等状态，最终进入 CLOSE 状态

实验中进行多次测试，我们认为 TIME_WAIT 状态持续了大约 4 分钟

- 客户端断网后异常退出，服务器的 TCP 连接状态有什么变化吗？服务器该如何检测连接是否继续有效？

客户端断网后，使用 netstat -an 检查连接状态，发现服务器的连接状态不会立即改变，仍然保持 ESTABLISHED 状态，直到服务器检测到连接失效。

服务器检测连接是否有效的方式：

1. 启用 TCP Keepalive 机制，定期探测连接的有效性。
2. 超时机制和错误检测，比如 recv() 返回错误或超时来判断连接是否有效。
3. 利用 TCP 重传机制，如果长时间没有收到响应，则认为连接已经丢失

七、讨论、心得

非常复杂的漫长的实验

因为上课的时候听老师讲要求就觉得内容非常多，而且又是开发的项目，所以我们组在 10.01 国庆当天就建立了仓库，开始了开发工作

总体来说，前期进度不算太多，主要是基本的 Server & client 框架的搭建，参照了很多的资料，进行了很多测试，这里其实难度不大内容不算特别多，所以没有什么多的感受

到了后期，也就是 ddl 接近这几天，我们开始全力冲刺项目收尾，才感觉到这个实验的不一样

除了搭建聊天系统外还要实现一些特别独特的功能，比如说气象信息等等，实验的要求特别多，内容很繁琐，需要添加很多的业务代码，因此焦头烂额

最后也是成功完成项目，通过了测试，但是对我们两人的消耗非常大。。。

而且实验报告的撰写要求也非常多，导致连着好几天熬到两三点才睡

非常不容易，这个项目花费了我们一个多月的时间，投入了巨大的时间和身体精力，其它专业课的实验、作业都没有怎么顾及，有点顾此失彼了，但是我们还是非常认真的完成了这个实验，建立了有效的合作机制

总体来说收获比较大，如果能有更多时间就好了