Task2 UDP socket programming

1. wireshark抓包：server在主机运行，client在虚拟机中运行，同时需要运行若干次客户端与服务器抓取丢包行为。
2. 单一客户端时：wireshark抓包的截图如图1所示，在虚拟机中运行的客户端记录如图2所示，其中，红色框内为重传的数据包，且在小于超时次数内被服务器判定为接收到；黄色框内为超时重传后服务器仍然未接受到的数据包，则被服务器丢弃。

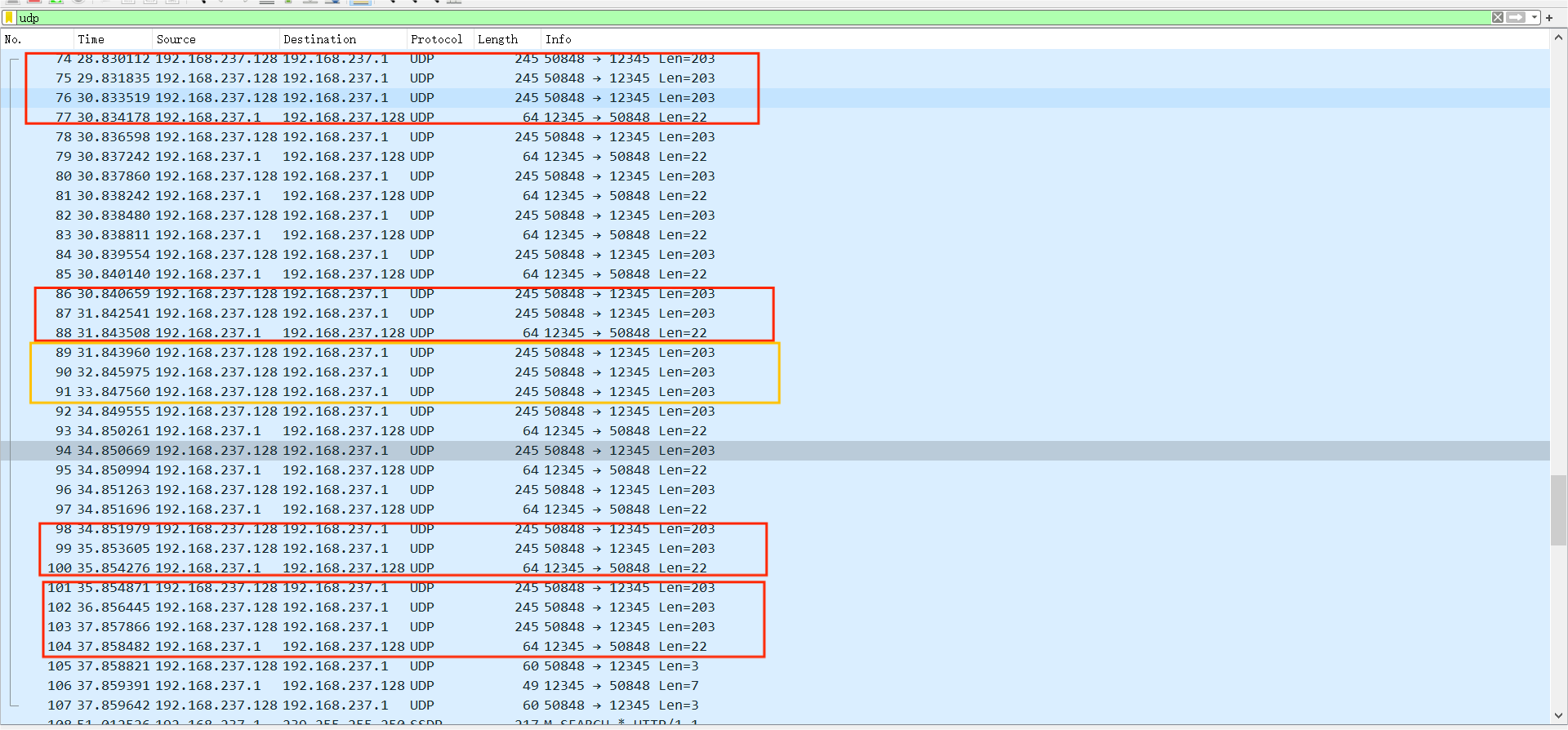


图1 单一客户端时wireshark抓包截图

客户端的界面截图如图2所示：



图2 客户端页面截图

服务器的界面截图如图3所示：

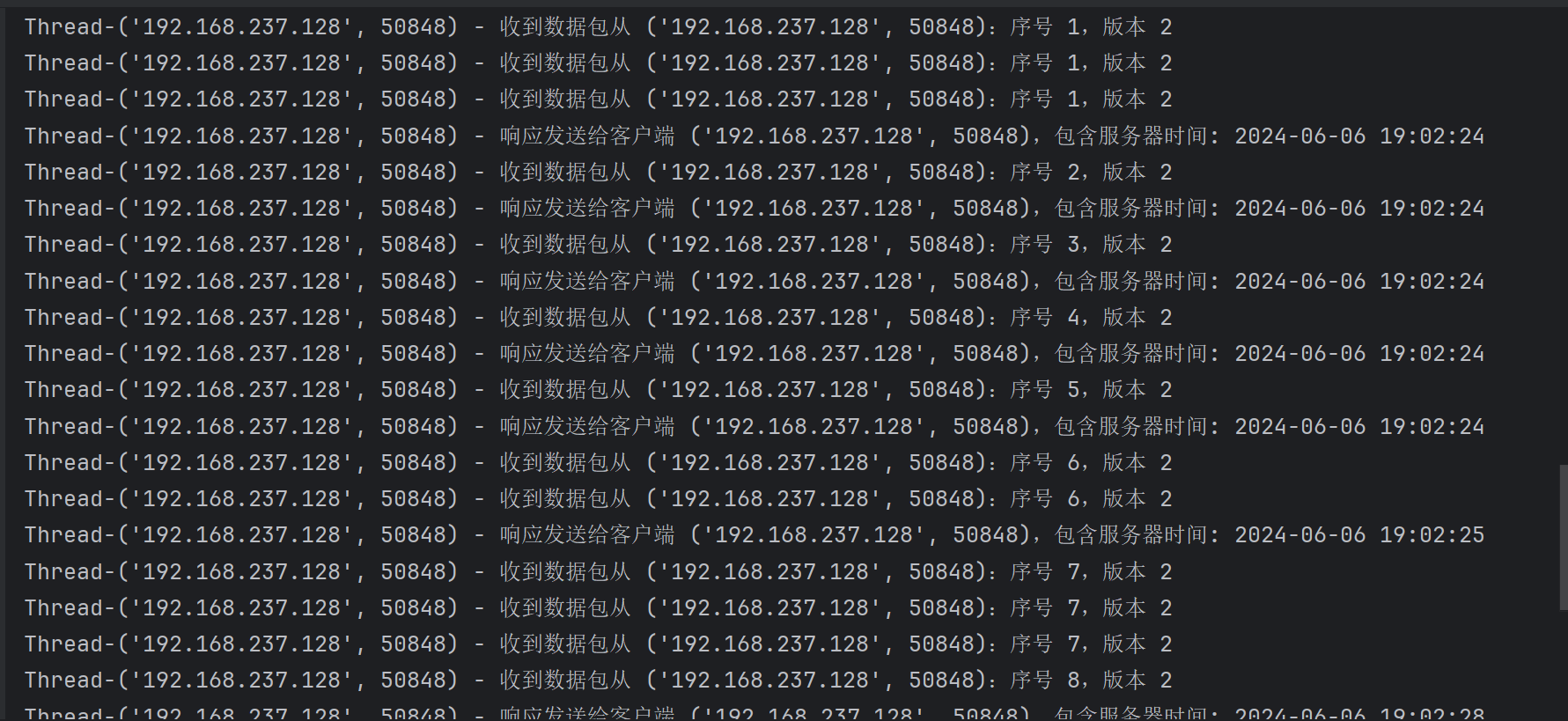


图3 服务器运行截图

1. 两个客户端时：服务器满足支持同时和多个客户端建立连接，当分别运行两个客户端进程时，：wireshark抓包的截图如图3-1和3-2所示，由于两个进程同时运行，我们需要通过客户端的端口号来区分各自的进程，对于本次抓包，红色框中对应pid为41540的进程1重传的数据包，在小于超时次数内被服务器判定为接收到，黄色框中对应进程1超时重传后服务器仍然未接受到的数据包，则被服务器丢弃。

黑色框中对应pid为33222的进程2重传的数据包，在小于超时次数内被服务器判定为接收到，在本次抓包时，进程2没有丢包。

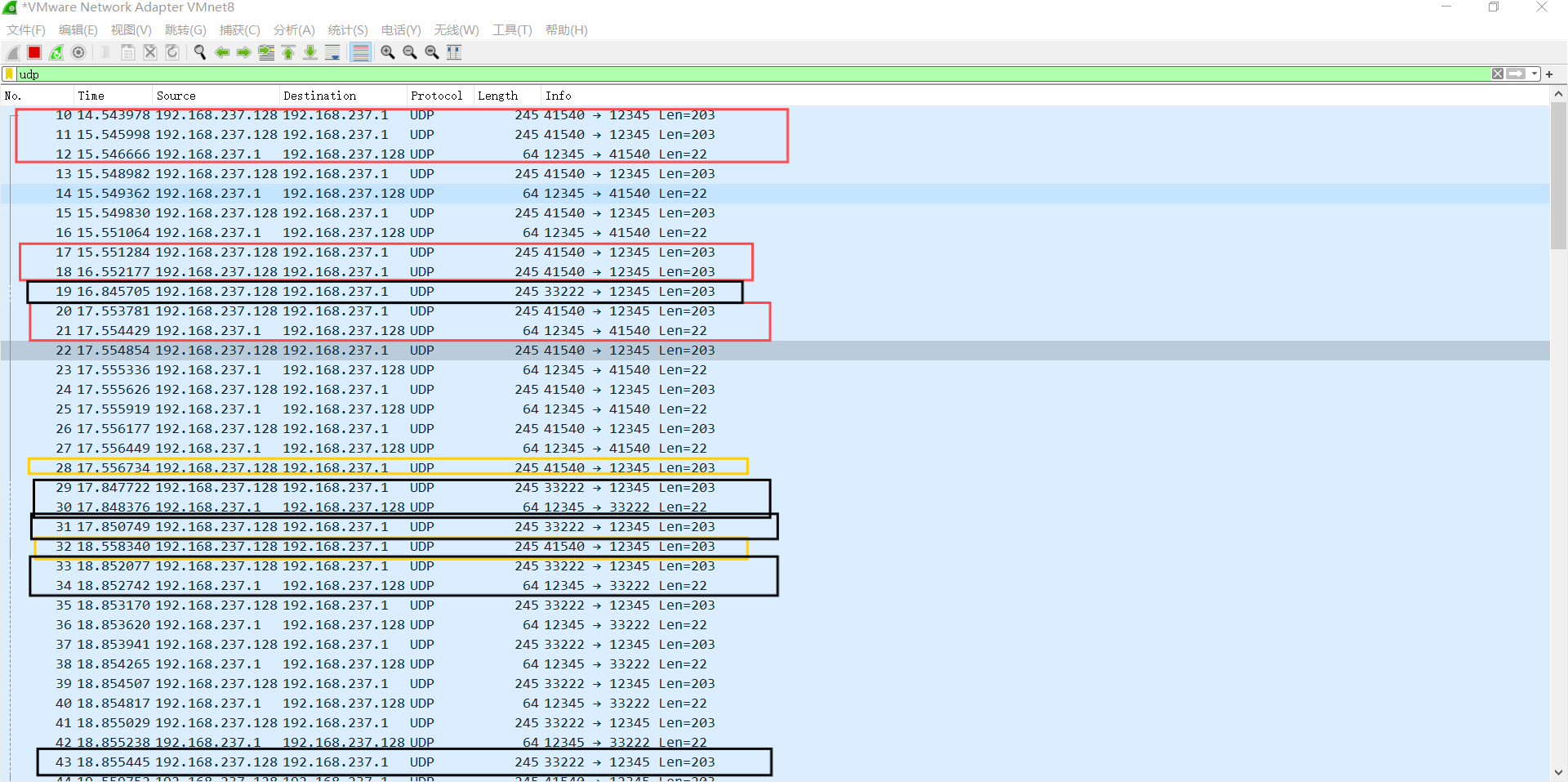


图3-1 多客户端时wireshark抓包截图

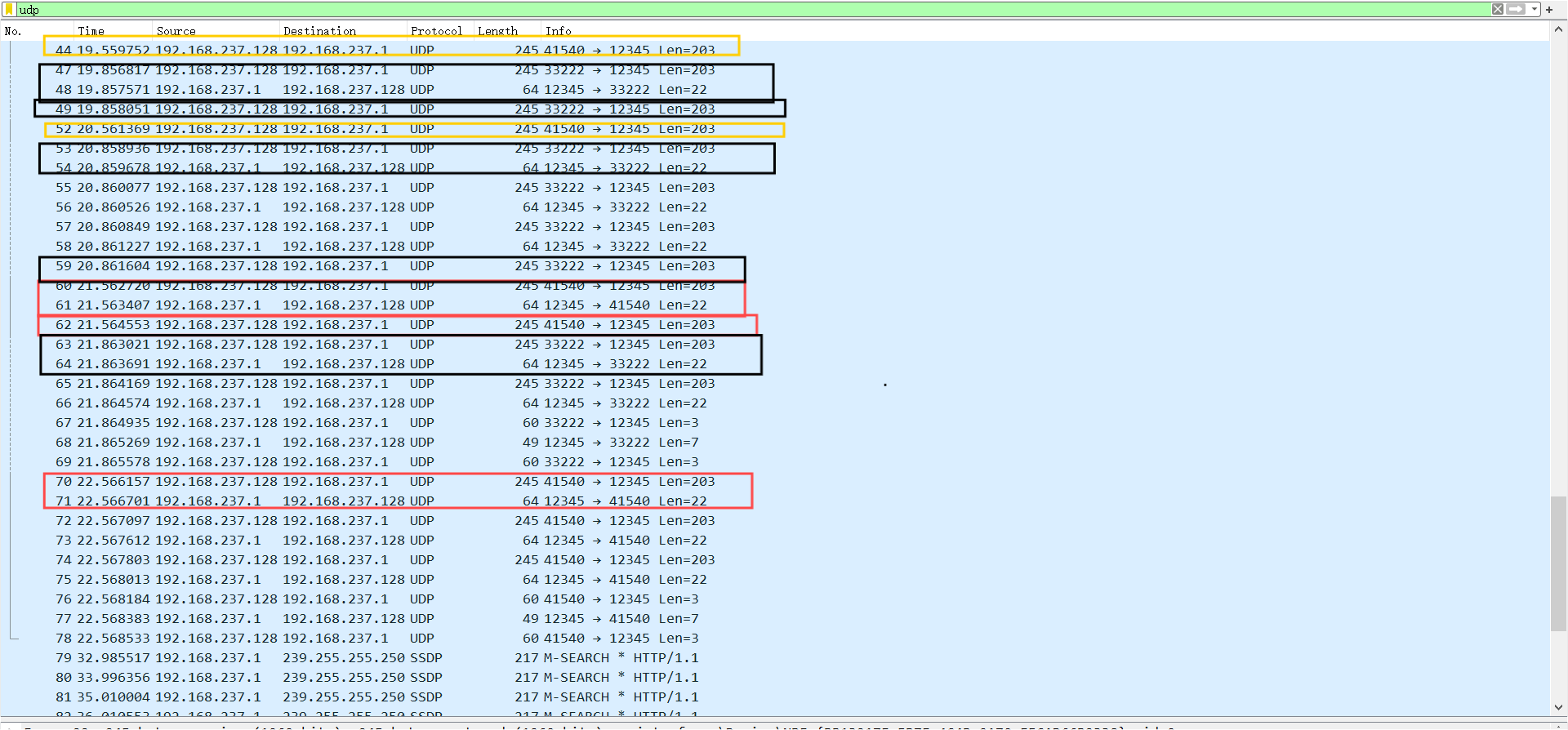


图3-2 多客户端时wireshark抓包截图

两个客户端的界面截图如图所示：

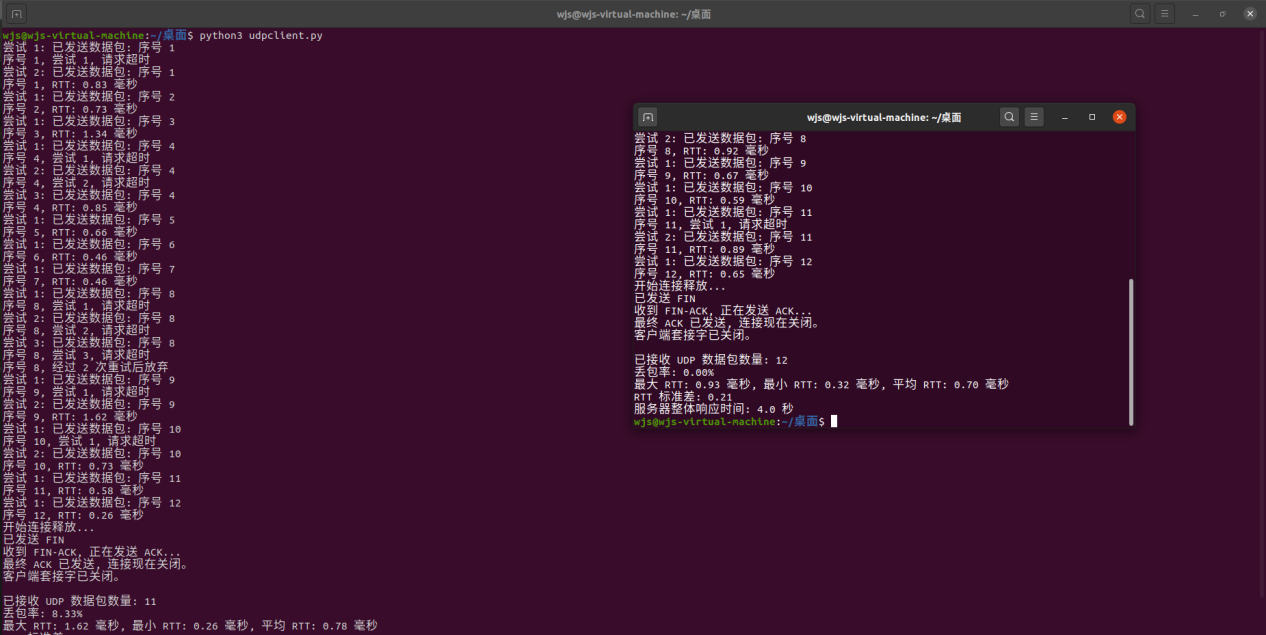


图4 客户端页面截图

服务器收到两个进程的数据界面截图如图5所示：

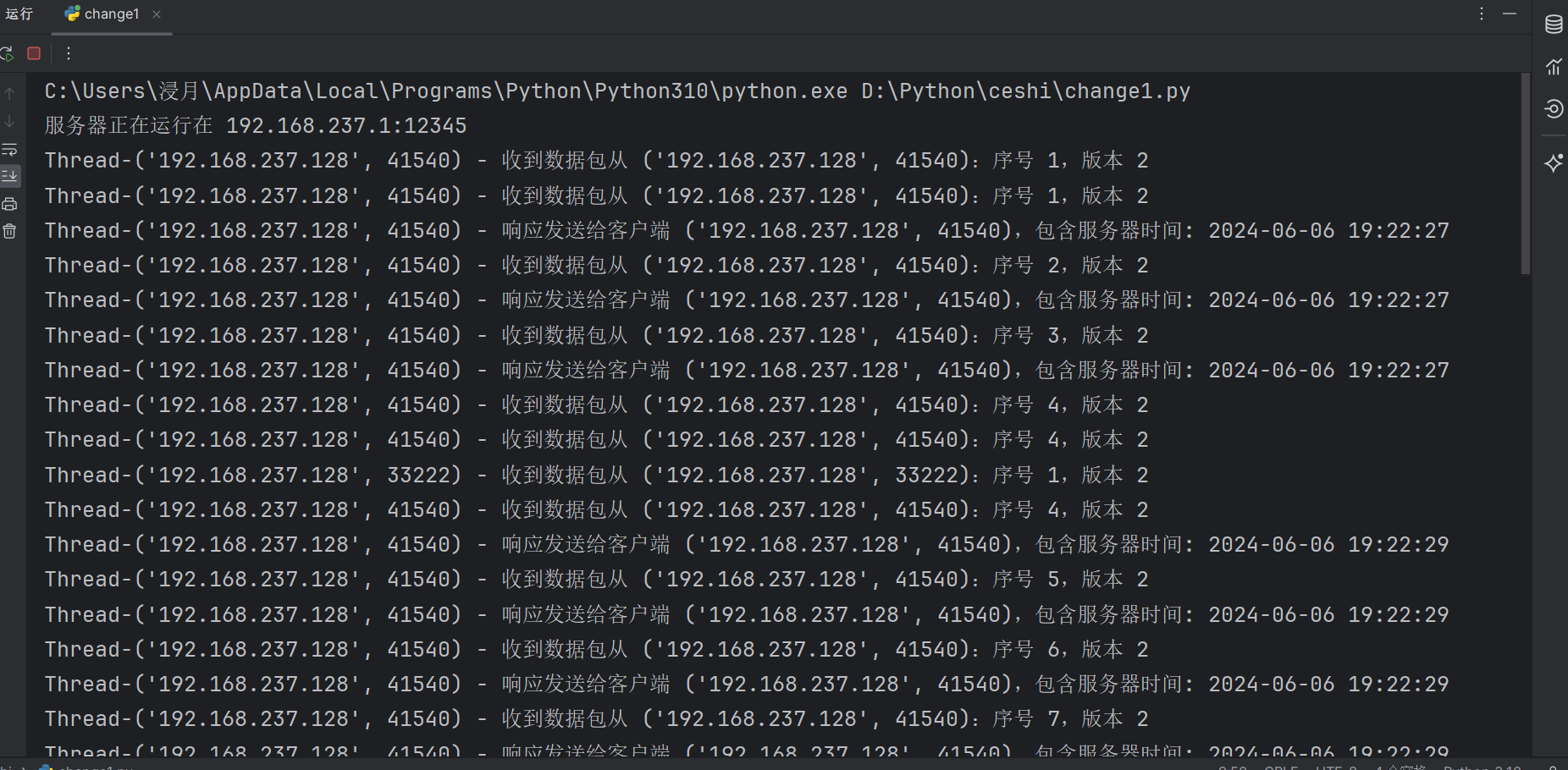


图5 服务器端截图

1. 实现中的关键点
2. server服务器端：

服务器设置与绑定：服务器绑定在特定的IP地址（192.168.237.1）和端口号（12345）上，这意味着服务器将在这个IP地址和端口上监听客户端发送的数据；UDP套接字创建：使用socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)创建了一个UDP套接字。UDP是一种无连接的协议，适用于对实时数据传输要求较高的应用。如图6所示：



图6 服务器设置与绑定与创建UDP套接字

多线程创建：每当接收到一个新的客户端请求时，就创建一个新的线程来处理该请求。这样做的目的是允许服务器同时处理多个客户端请求，提高效率，如图7所示：

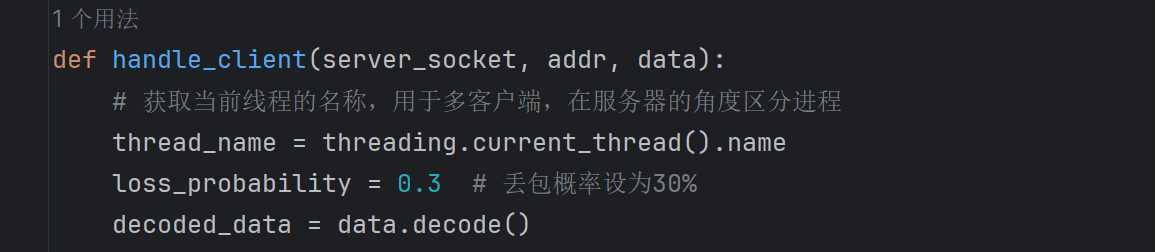


图7 线程创建

数据包数据解析：接收到的数据首先被解码，并根据内容进行不同的处理。如果是FIN，服务器将发送FIN-ACK来结束连接；如果是ACK，则视为连接结束的确认；序列号与版本：数据包前两字节被解析为序列号，第三个字节解析为版本号，服务器据此进行相应的处理；丢包模拟：通过设定30%的概率来随机丢弃一些数据包，这是为了模拟网络不稳定情况下的数据传输，在实际网络环境中可能发生的数据包丢失情况。



图8 数据包解析数据

服务器时间戳响应：服务器将其当前时间作为响应附加到数据包中，然后发送回客户端。可以用于客户端验证服务器响应的实时性或进行时间同步，如图9所示。

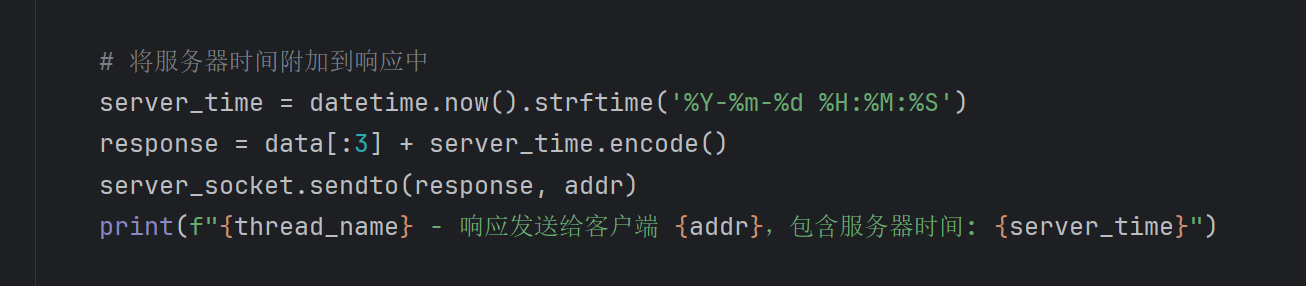


图9 服务器时间戳响应

1. client客户端：

服务器 IP 地址和端口：通过 server\_ip 和 server\_port 指定，分别用于定义目标服务器的网络地址和通信端口；请求的数量：num\_requests 设为 12，表示客户端将发送 12 个数据包到服务器；超时设置：timeout 设为 0.1 秒（100 毫秒），用于定义等待服务器响应的最大时间；最大重试次数：max\_retries 设为 2，意味着每个数据包将在最初的尝试后最多重发两次。如图10所示：

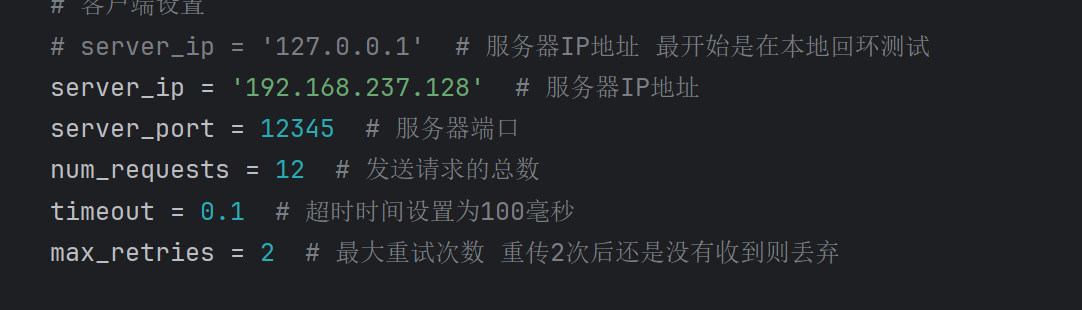


图10 客户端初始化设置

创建 UDP 套接字：使用 socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) 创建一个 UDP 套接字；地址重用：设置 SO\_REUSEADDR 选项，这允许在同一端口上启动多个应用程序而不会遇到地址已在使用的错误；设置超时：使用 settimeout 方法应用超时设置，使得 recvfrom 在指定时间内没有接收到数据时会抛出超时异常。如图11所示：

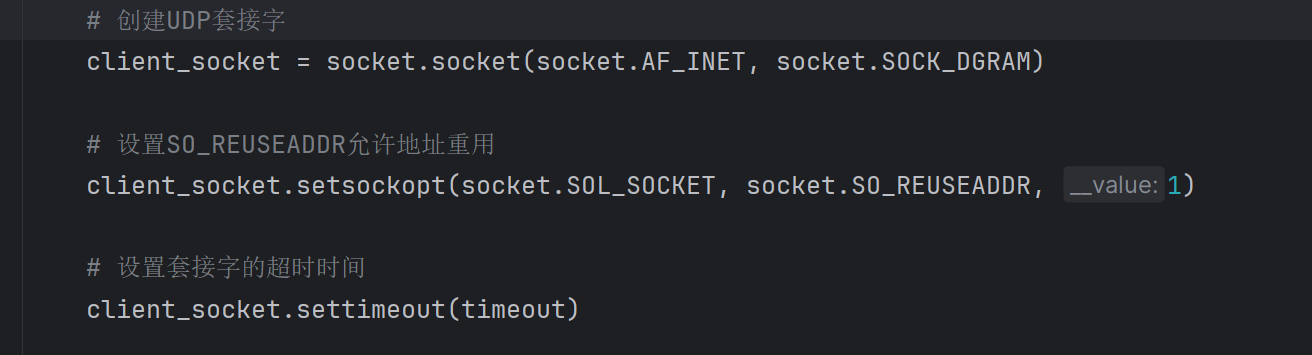


图11 网络套接字配置

数据包的构造和发送：循环构造包含序号、协议版本和填充字节的数据包。序号是以大端格式编码的两字节整数，用于唯一标识每个数据包；接收响应：尝试接收服务器响应并计算往返时间（RTT）。如果在设定的超时时间内未收到响应，则进行重试，最多不超过设定的重试次数；处理超时：在尝试接收数据时，如果发生超时异常，程序将重试发送数据包，最多重试 max\_retries 次。超过这个次数后，程序将放弃当前数据包并记录为丢失。如图12-1和12-2所示：



图12-1 数据发送和接收

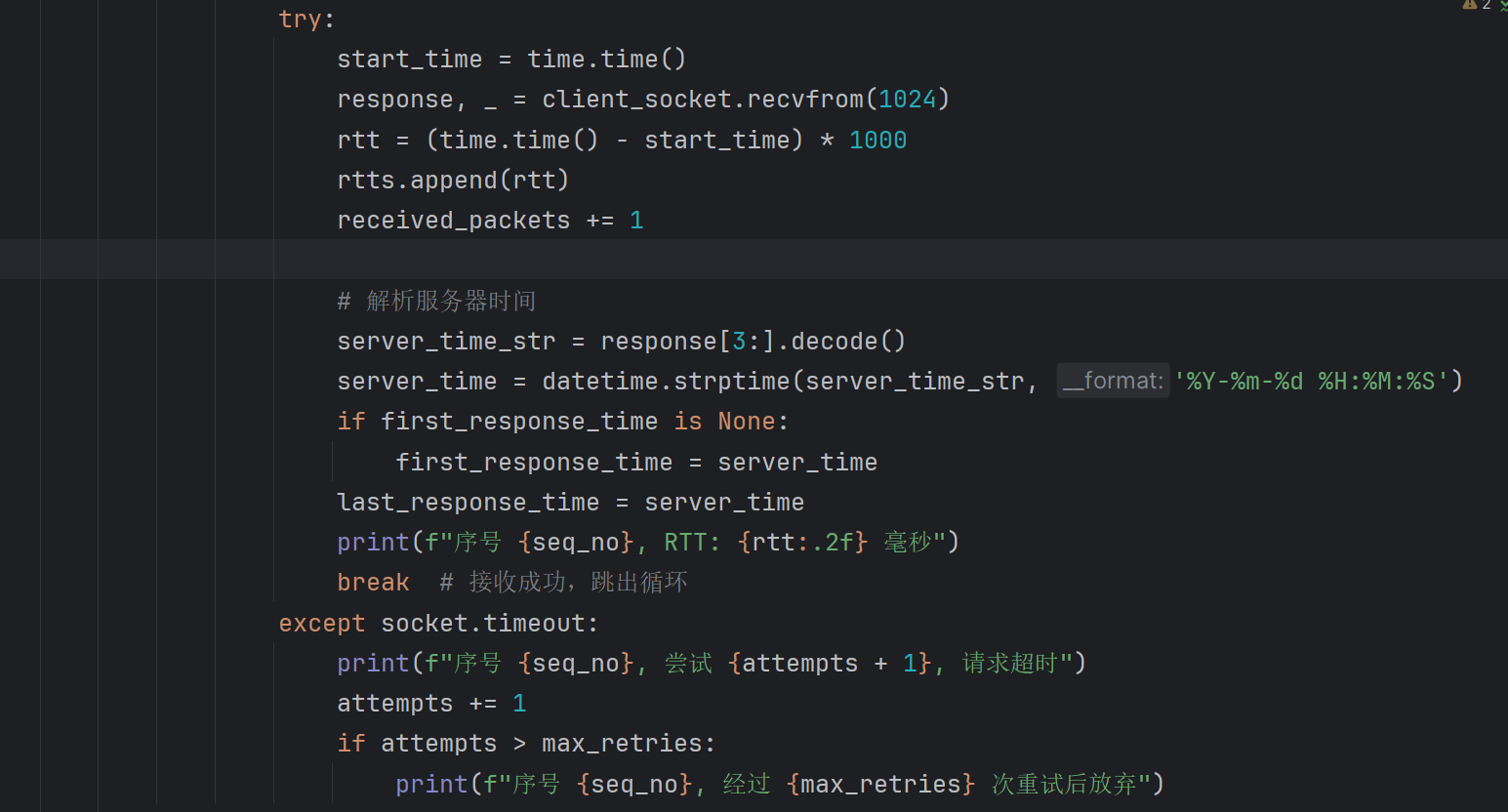


图12-2 超时和重试逻辑

模拟 TCP 的连接释放：通过发送一个特殊的 FIN 数据包来模拟连接终止过程。收到 FIN-ACK 响应后，再发送一个 ACK 来完成三次握手过程的模拟。如图13所示：



图13 模拟连接释放

计算和输出统计数据：在发送所有请求之后，统计接收到的数据包数、计算丢包率、RTT 的最大值、最小值、平均值和标准差（如果有足够的数据）。记录响应时间：记录接收到的第一个和最后一个响应的时间戳，用于评估服务器的整体响应性。如图14所示：



图14 统计数据

1. 程序涉及了一系列网络编程的核心知识点和高级功能。
2. 理解 UDP 协议：本实验通过编写 UDP 客户端和服务器代码，直接操作 UDP 套接字，加深了对 UDP 协议的无连接、不保证可靠交付的特性的理解。通过实际发送和接收数据包，更好地理解 UDP 的工作机制和使用场景。
3. 套接字编程和超时逻辑：学习了如何使用 Python 的 socket 模块创建和管理网络套接字，包括设置套接字选项、绑定地址和端口、以及发送和接收数据。通过实现超时检测和数据重发逻辑，学到了如何在无连接的 UDP 传输中模拟可靠的数据传输机制。这不仅提高了解决实际网络问题的能力，还理解了更复杂的网络协议（如 TCP）背后的一些基本原理。
4. 网络不稳定性：实验中通过在服务器端引入随机丢包逻辑，模拟了真实网络环境中的不稳定性，如丢包和网络延迟。这帮助你了解如何在设计和测试网络应用时考虑这些因素，以及如何编写能够适应这些问题的鲁棒性代码。
5. 多个线程：在一个多线程的服务器模型中，服务器可以为每个新的客户端连接分配一个独立的线程。这样，服务器能够同时处理多个客户端的请求，提高响应效率和吞吐量。通过这种方式，你可以学习如何有效地管理线程，包括创建、执行和终止线程
6. Git链接：https://github.com/826962555/Computer-Network.git