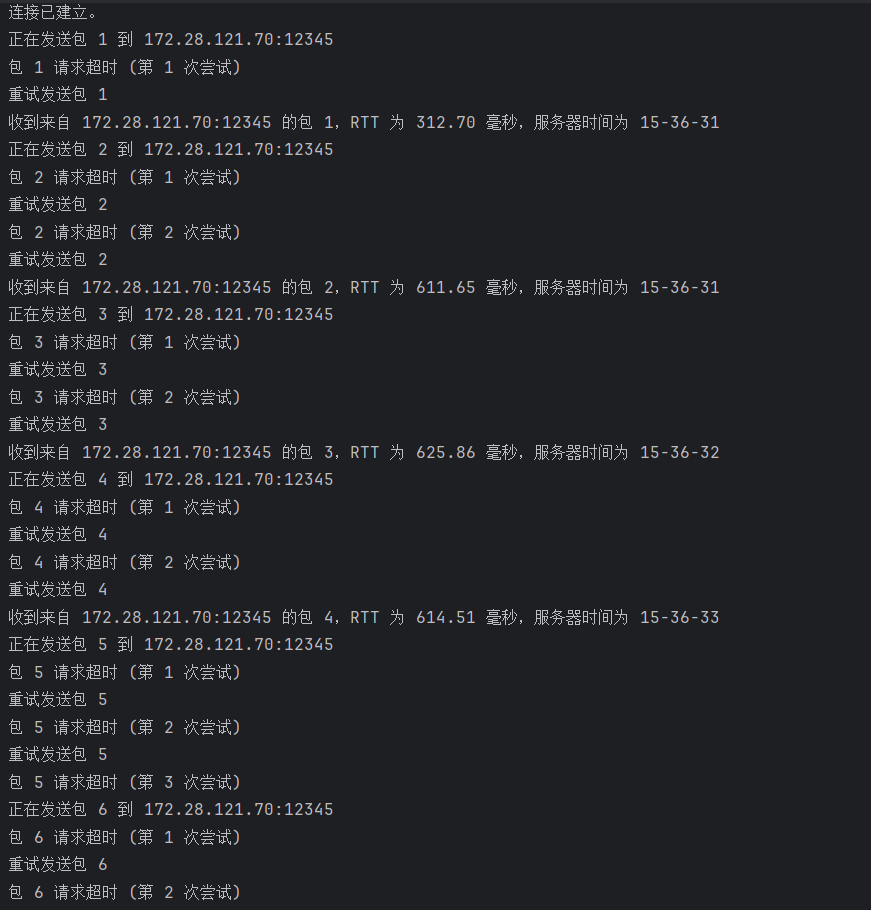
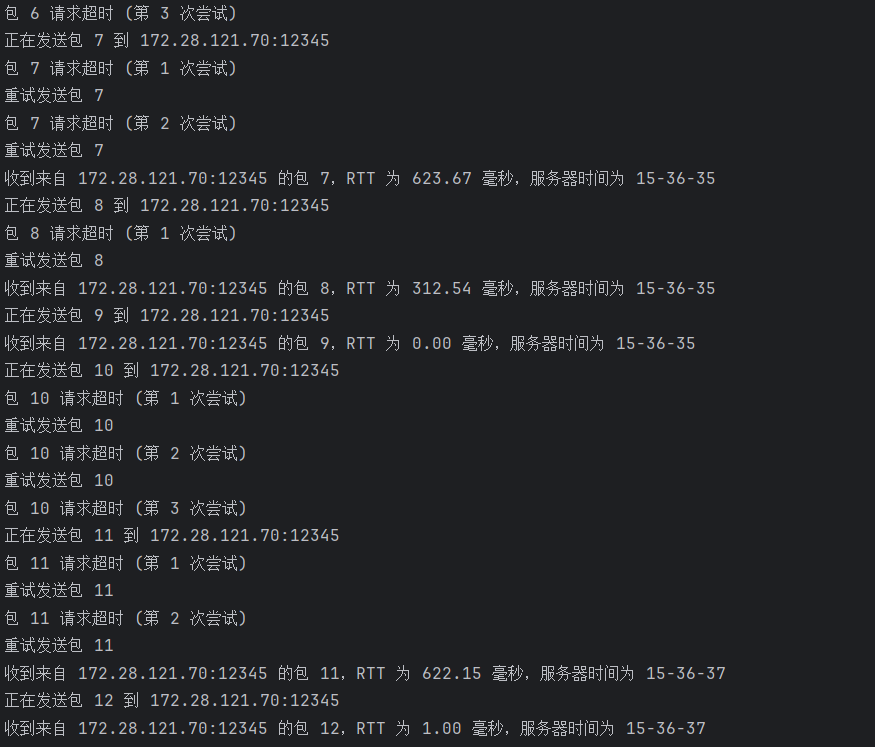
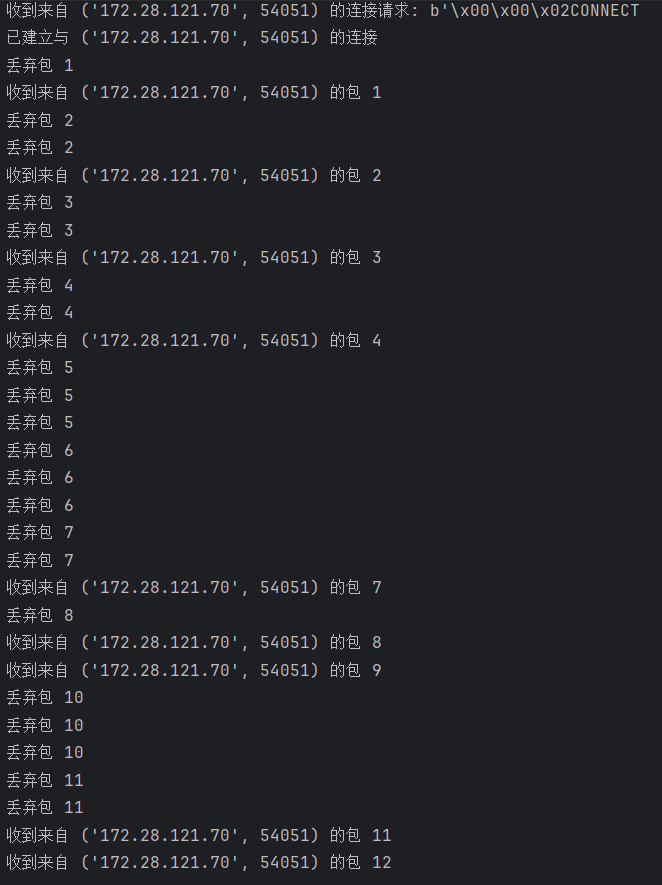
**a.**

当服务器与客户端建立连接后，客户端发送包截图如下：

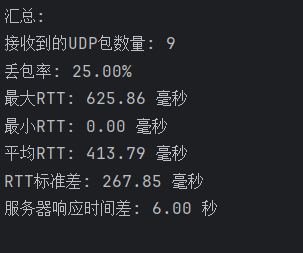




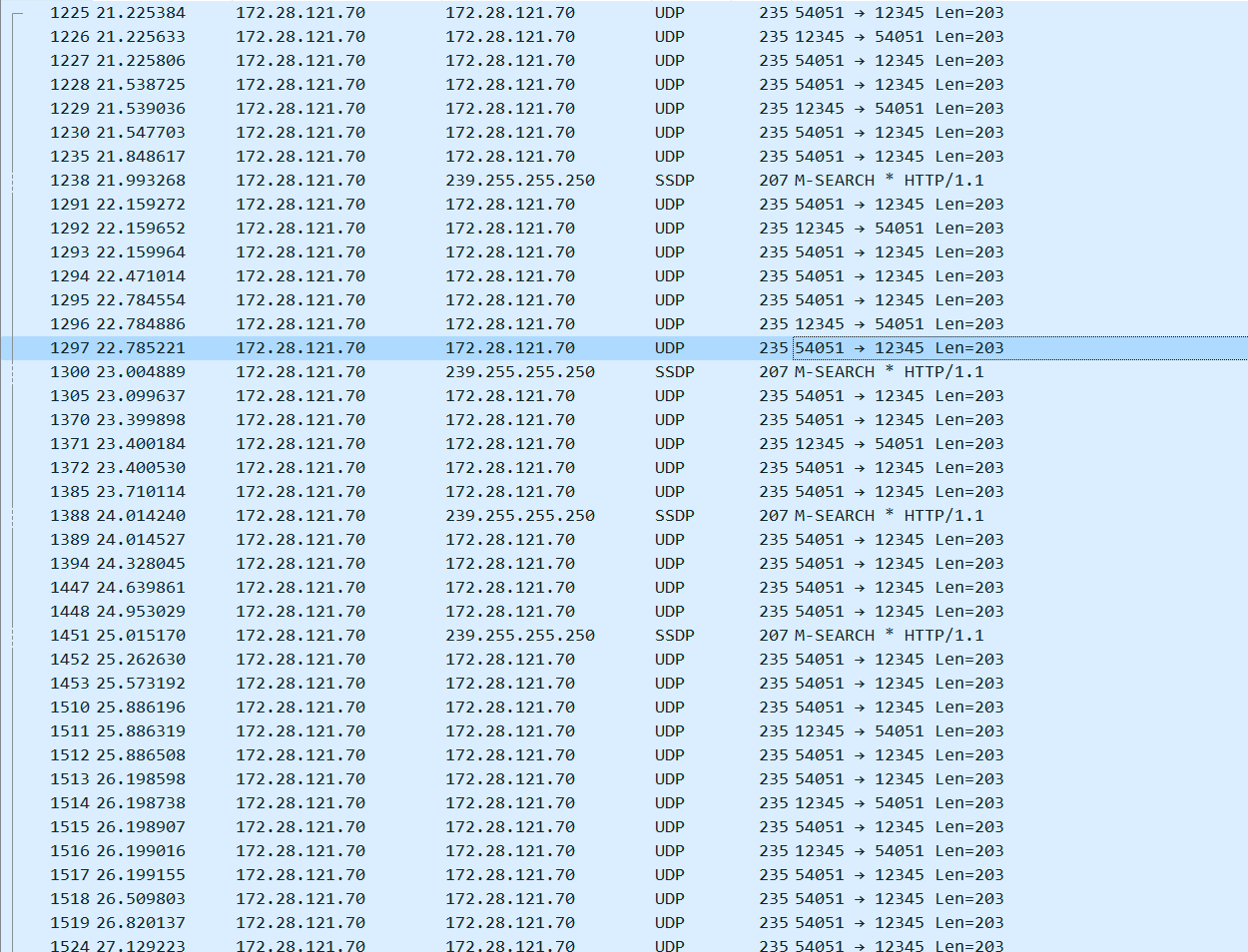
服务器接受包截图如下：



汇总如下：



这是wireshark的截图，可以看到前两条为连接的确认，确认后开始发生包。第一个数据包重传了一次，所以有两次发送一次接受，第二个重传两次，有三次发送一次接受。若三次重传均丢包，则没有接受，如包5，6，10。



**b.关键点和解决办法**

**连接建立和验证：**

关键点：确保服务器只响应有效的连接请求，并且客户端确认连接已经建立。

解决方案：使用预定义的连接请求和响应模式。服务器接收到包含特定标识（如b"\x02"）和b"CONNECT"消息的包后，会发送相同的包回去作为确认。客户端在发送连接请求后，等待服务器的响应来确认连接是否建立成功。

# 服务器端

if data[SEQ\_BYTES:SEQ\_BYTES+VER\_BYTES] == b"\x02" and data[SEQ\_BYTES+VER\_BYTES:].strip() == b"CONNECT":

server\_socket.sendto(data, addr)

print(f"已建立与 {addr} 的连接")

break

# 客户端

connect\_packet=struct.pack(PACKET\_FORMAT,b'\x00\x00',b'\x02', b'CONNECT'.ljust(CONTENT\_BYTES, b' '))

client\_socket.sendto(connect\_packet, (server\_ip, server\_port))try:

data, \_ = client\_socket.recvfrom(2048)

print("收到服务器响应:", data)

print("连接已建立。")except socket.timeout:

print("连接超时。")

exit()

**数据包丢失模拟：**

关键点：模拟实际网络环境中可能出现的数据包丢失。

解决方案：使用随机数生成来决定是否丢弃一个包。在服务器端接收到数据包后，通过random.random()生成一个介于0到1之间的随机数，如果小于预设的丢包率（如0.6），则丢弃该包。

if random.random() < drop\_rate:

print(f"丢弃包 {seq\_number}")

continue

**RTT（Round Trip Time）计算和包重传：**

关键点：计算每个包从发送到接收到的时间（RTT），并在超时时重传未确认的包。

解决方案：在客户端发送包后记录发送时间，并在接收到服务器响应包时记录接收时间，计算RTT。为了处理可能的超时情况，设置多次重试机制，在每次重试前等待一段时间。

for seq in range(1, 13):

send\_time = time.time()

client\_socket.sendto(packet, (server\_ip, server\_port))

for attempt in range(3):

try:

data, \_ = client\_socket.recvfrom(2048)

receive\_time = time.time()

rtt = (receive\_time - send\_time) \* 1000

print(f"收到来自 {server\_ip}:{server\_port} 的包 {received\_sequence\_number}，RTT 为 {rtt:.2f} 毫秒")

rtts.append(rtt)

break

except socket.timeout:

print(f"包 {seq} 请求超时 (第 {attempt + 1} 次尝试)")

if attempt == 2:

break

client\_socket.sendto(packet, (server\_ip, server\_port))

**服务器时间同步和统计计算：**

关键点：记录和计算服务器的响应时间差和统计信息（如RTT最大值、最小值、平均值、标准差）。

解决方案：将服务器返回的时间戳解析为秒数，计算从开始到结束的时间差，并统计RTT相关的数据。

server\_time\_struct = time.strptime(server\_time, "%H-%M-%S")

server\_time\_seconds = server\_time\_struct.tm\_hour \* 3600 + server\_time\_struct.tm\_min \* 60 + server\_time\_struct.tm\_sec

response\_times.append(server\_time\_seconds)

total\_response\_time = end\_time - start\_time if response\_times else 0.0

**丢包率和RTT统计**：

关键点：统计丢包率和RTT相关指标（最大值、最小值、平均值、标准差）。

解决方案：通过记录发送和接收的包数量计算丢包率，并使用统计方法计算RTT的相关指标。

total\_sent = 12

total\_lost = total\_sent - received\_packets

loss\_rate = total\_lost / total\_sent \* 100if rtts:

max\_rtt = max(rtts)

min\_rtt = min(rtts)

avg\_rtt = mean(rtts)

std\_rtt = stdev(rtts) if len(rtts) > 1 else 0.0

**c.掌握的知识点**

**Socket 编程：**

熟悉 Python 的socket模块，用于在服务器和客户端之间建立通信。

使用 UDP 套接字 (socket.SOCK\_DGRAM) 进行无连接的通信。

**数据打包和解包：**

使用struct模块进行数据打包和解包，以确保数据包的格式一致性。

了解不同数据类型的打包格式，如!H表示网络字节序（大端）的无符号短整型。

**网络数据传输：**

熟悉如何使用sendto和recvfrom方法发送和接收数据。

理解 UDP 的无连接特性，以及如何处理丢包和数据重传。

**时间处理：**

使用time模块记录和计算时间，包括发送时间、接收时间和服务器时间解析。

了解如何使用时间戳来计算 RTT（Round Trip Time）和服务器响应时间差。

**错误处理和重试机制：**

处理网络通信中的超时问题，使用settimeout设置超时时间。

实现重试机制，在包请求超时时重发数据包。

d.git的url

https://github.com/hhnhhh/udp