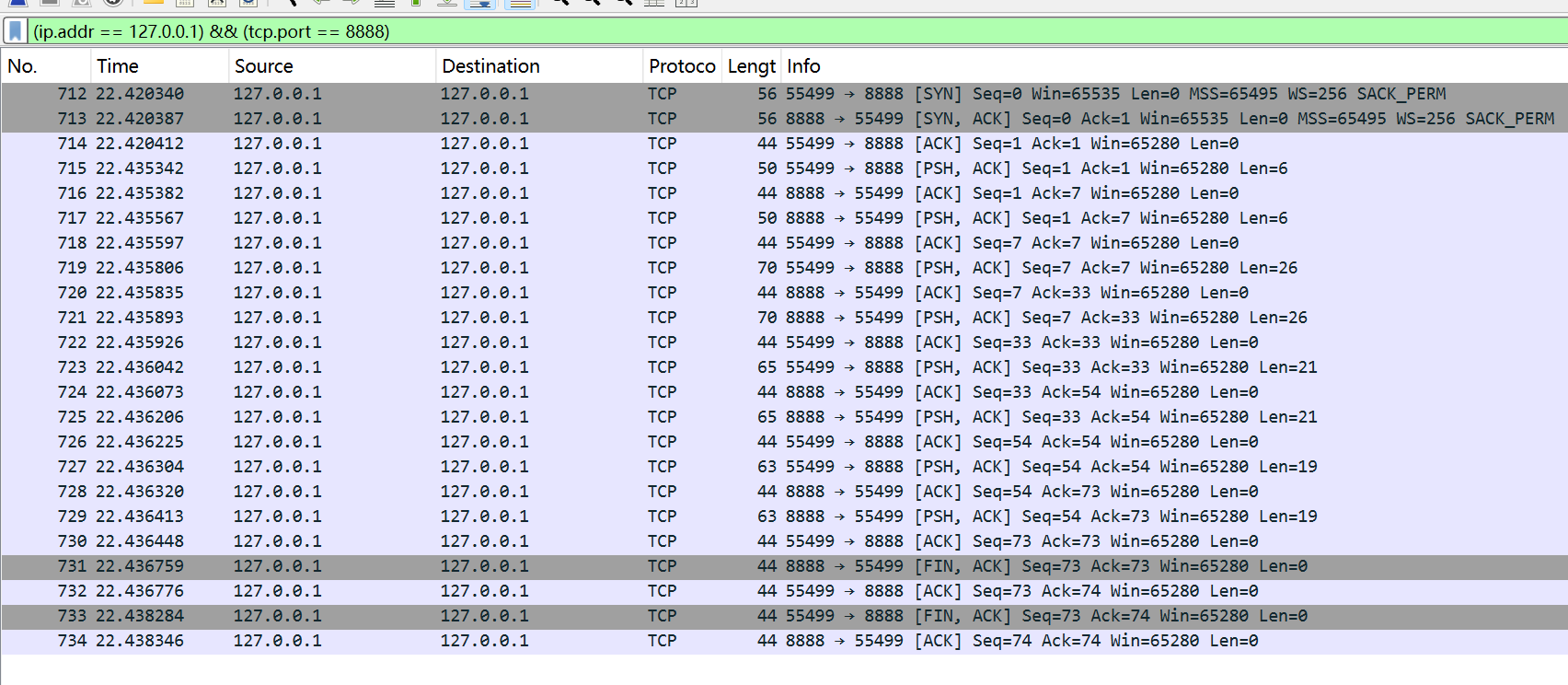
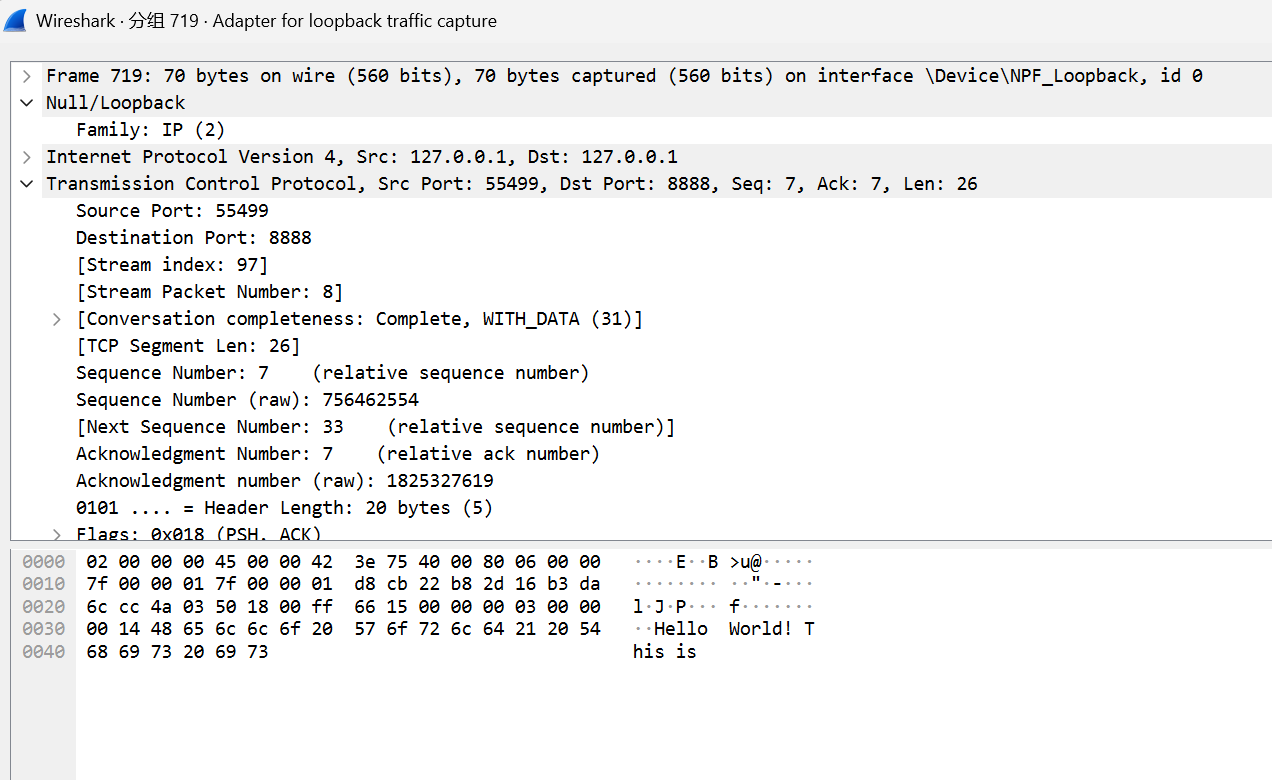
A.wireshark抓包截图





1. 代码关键点和解决方案：

服务器端（TCPserver.py）

端口验证：确保服务器监听的端口号在 1024 到 65535 之间。

多线程处理：使用线程来处理多个客户端连接，提高服务器的并发处理能力。

数据包格式：定义了不同类型的数据包（初始化包、同意包、请求包、响应包），并使用 struct 模块进行打包和解包。

错误处理：对各种可能的异常情况（如连接重置、数据包解析错误等）进行了处理。

客户端端（TCPclient.py）

用户输入验证：验证服务器 IP 地址、端口号、分块大小等用户输入的合法性。

文件读取与分块：读取输入文件内容，并将其分割成多个数据块。

数据包交互：与服务器进行数据包交互，包括发送初始化包、请求包，接收同意包、响应包。

数据处理与输出：处理服务器返回的反转数据，并将其按逆序保存到输出文件中。

流程展示：

初始化服务器

验证端口号

启动服务器监听

循环处理客户端连接:

接受客户端连接

创建线程处理客户端请求

线程处理客户端请求:

接收初始化包

验证初始化包类型

发送同意包

循环处理数据块:

接收请求包

验证请求包类型

接收请求数据

反转数据

发送响应包

关闭服务器

1. 用到并且掌握的知识点:

1网络编程

TCP 套接字编程：使用 socket 模块创建 TCP 套接字，实现客户端和服务器之间的通信。服务器端通过 socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) 创建监听套接字，并使用 bind、listen 和 accept 方法监听客户端连接；客户端使用 connect 方法连接到服务器。例如，在 connect\_to\_server 方法中，客户端创建套接字并连接到服务器。

2多线程处理：服务器端使用 threading 模块创建线程来处理多个客户端连接，提高了服务器的并发处理能力。每个客户端连接由一个独立的线程处理，互不干扰。例如，在 handle\_connections 方法中，为每个客户端连接创建一个新的线程。

3. 数据处理与传输

数据包格式与封装：定义了不同类型的数据包（初始化包、同意包、请求包、响应包），并使用 struct 模块进行打包和解包。通过 struct.pack 和 struct.unpack 方法将数据转换为字节流进行传输，确保数据的正确解析。例如，在 send\_initialization\_packet 方法中，使用 struct.pack 方法将数据包类型和块数量打包成字节流发送给服务器。

数据分块与处理：客户端将文件内容分割成多个数据块进行传输，服务器对每个数据块进行反转处理后返回给客户端。通过分块传输，可以提高数据传输的效率和可靠性。例如，在 split\_into\_chunks 方法中，客户端将文件内容分割成多个数据块。

1. git的url
2. 实验结果截图

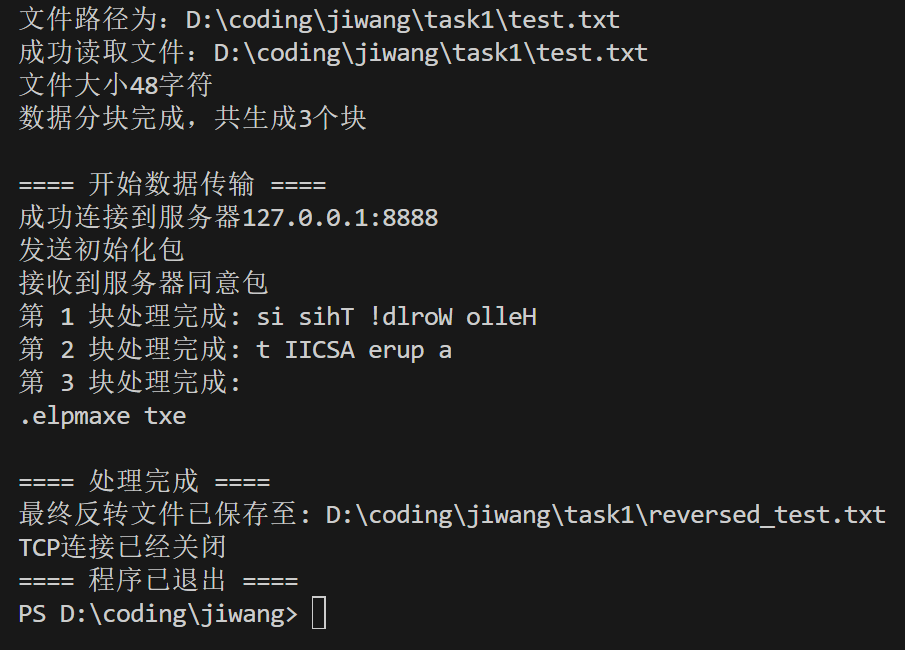


图1 Client端截图

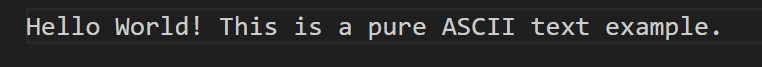


图2 原字符串



图3 反转字符串