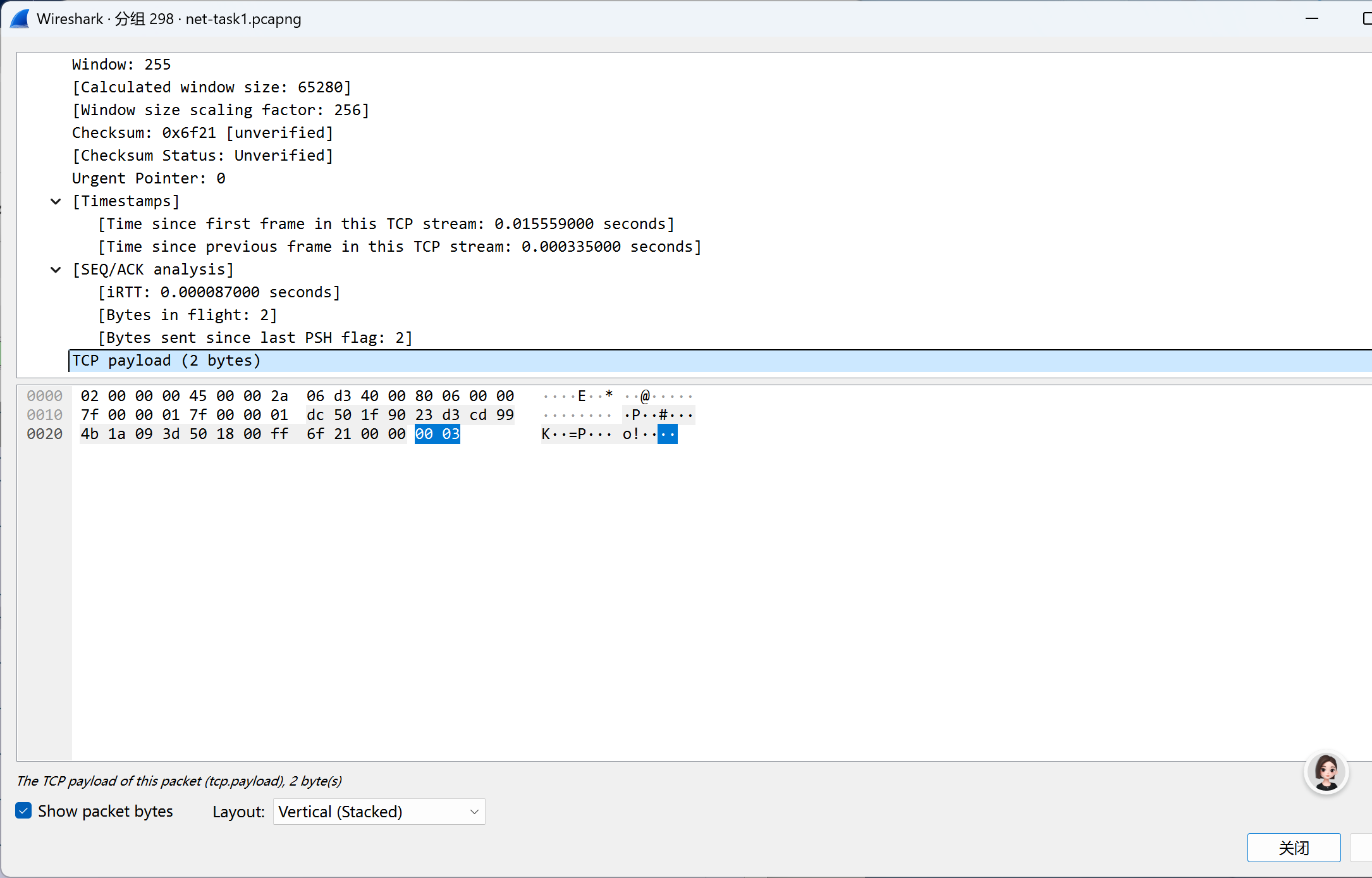
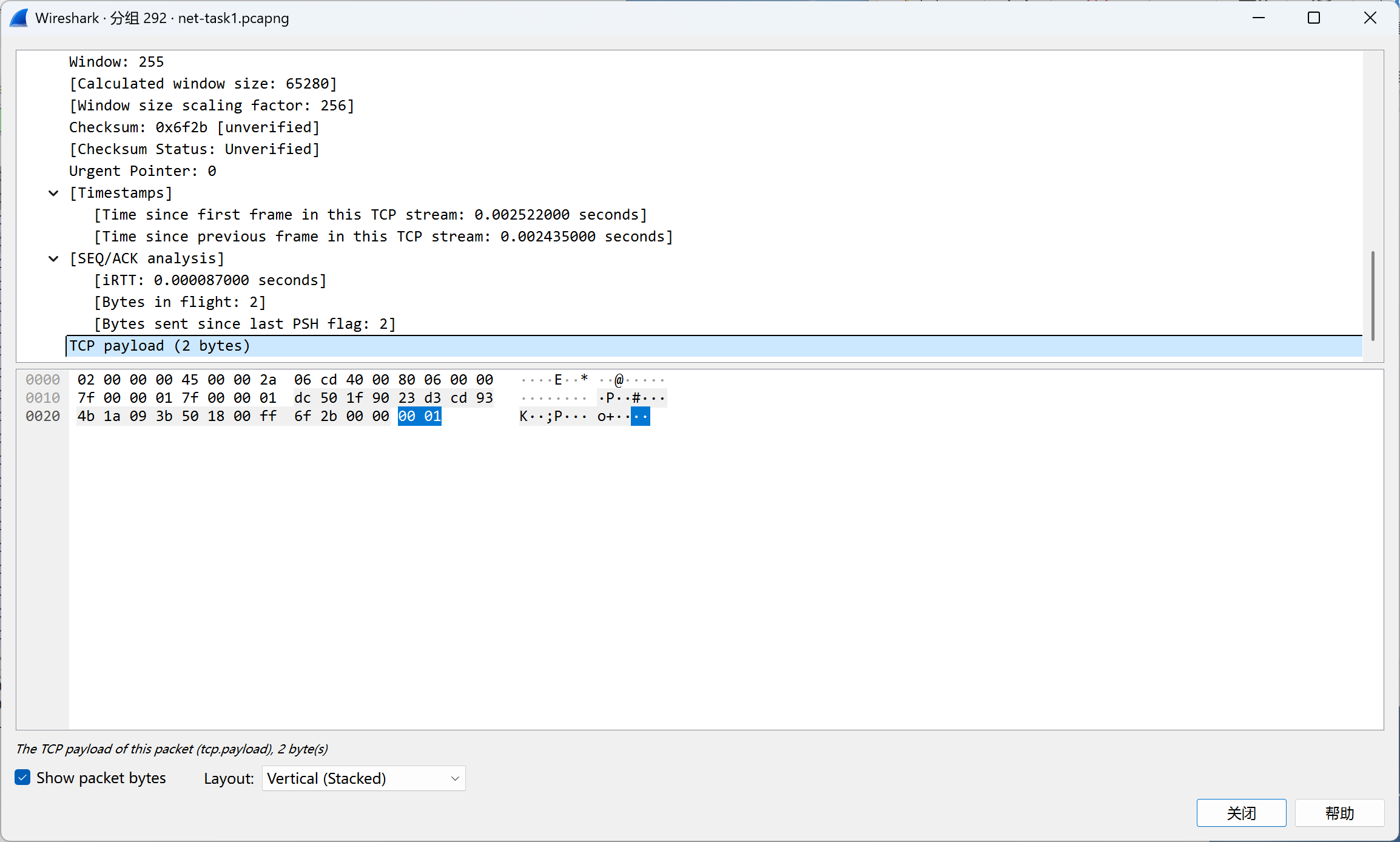
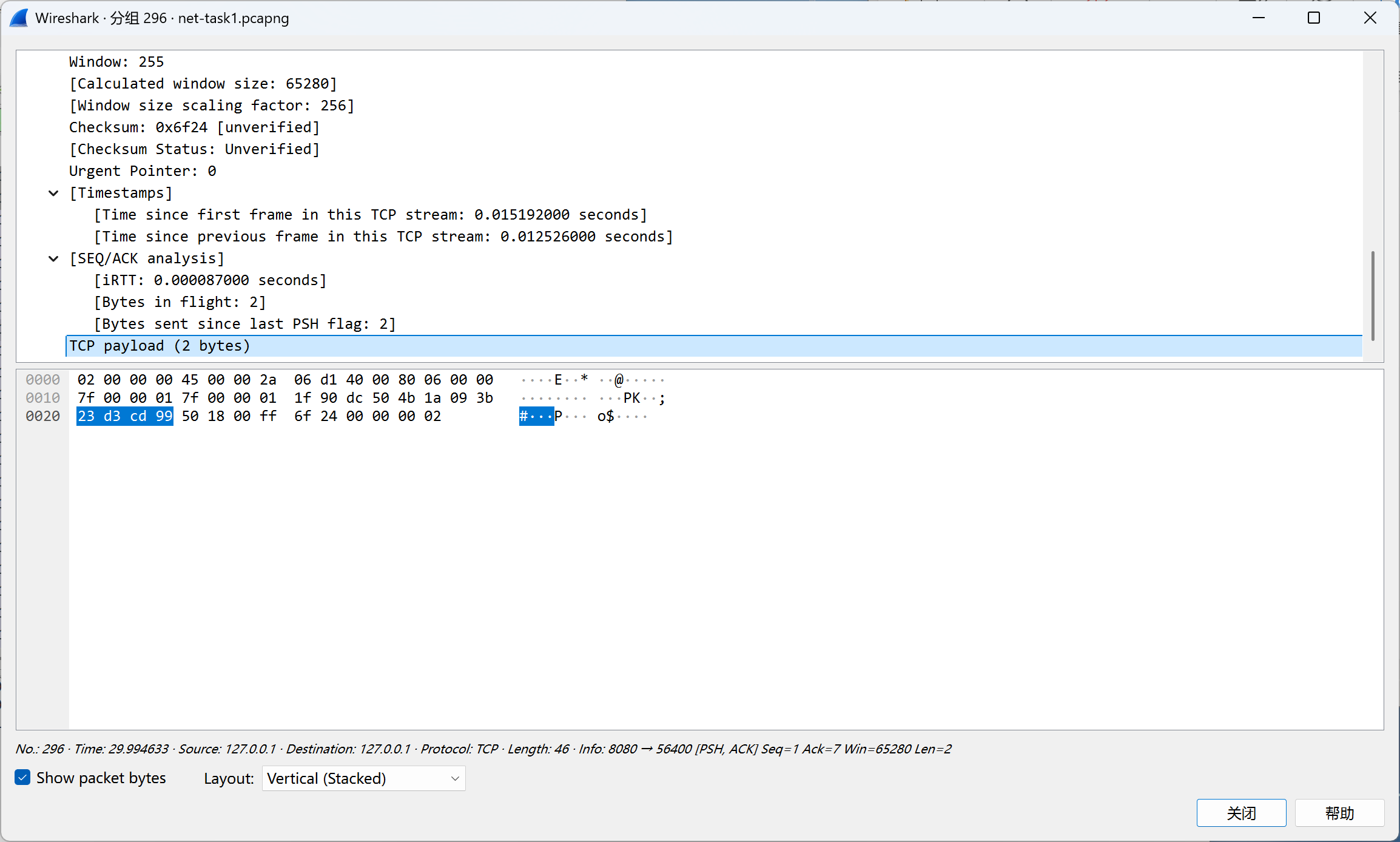
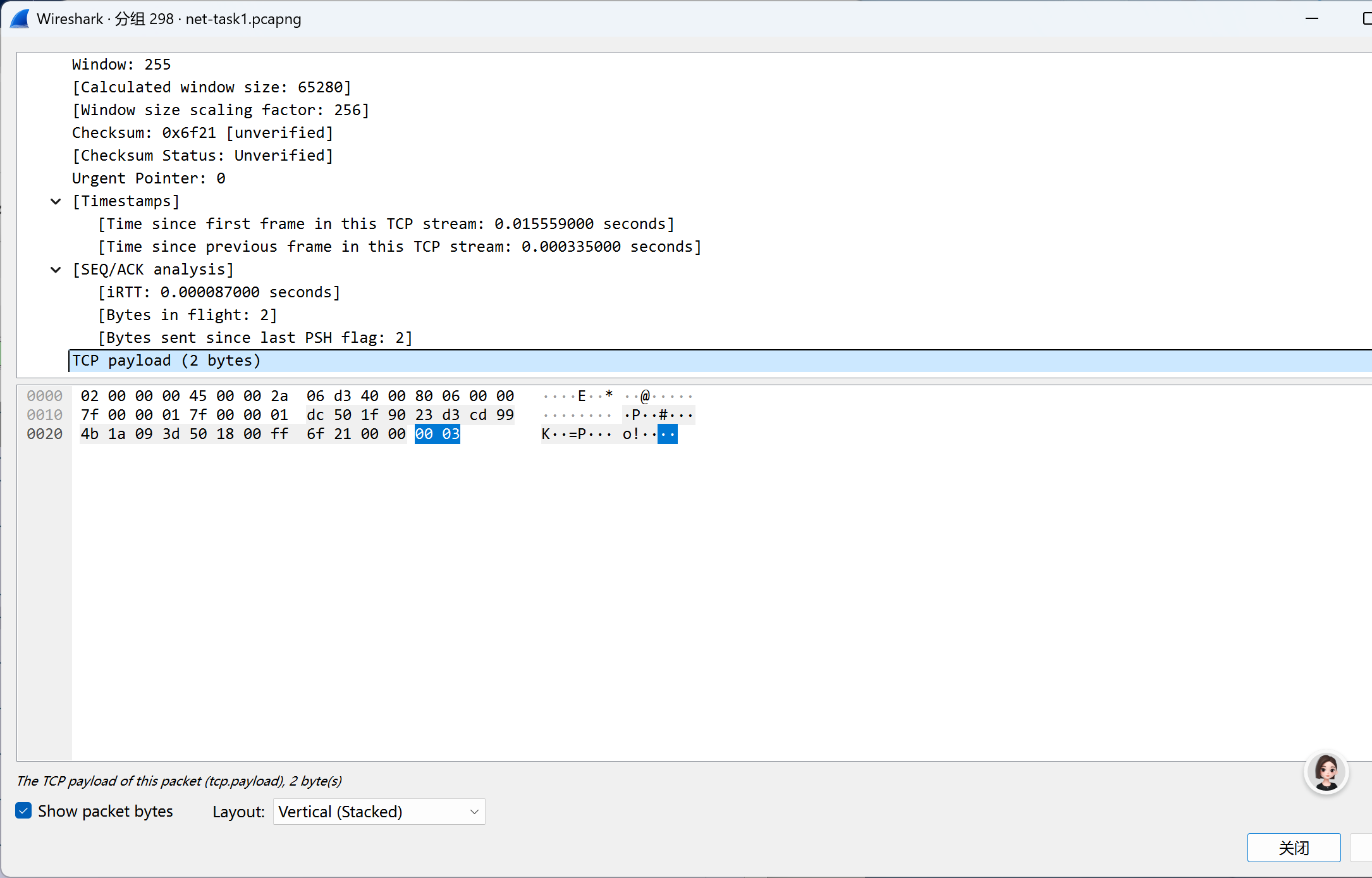
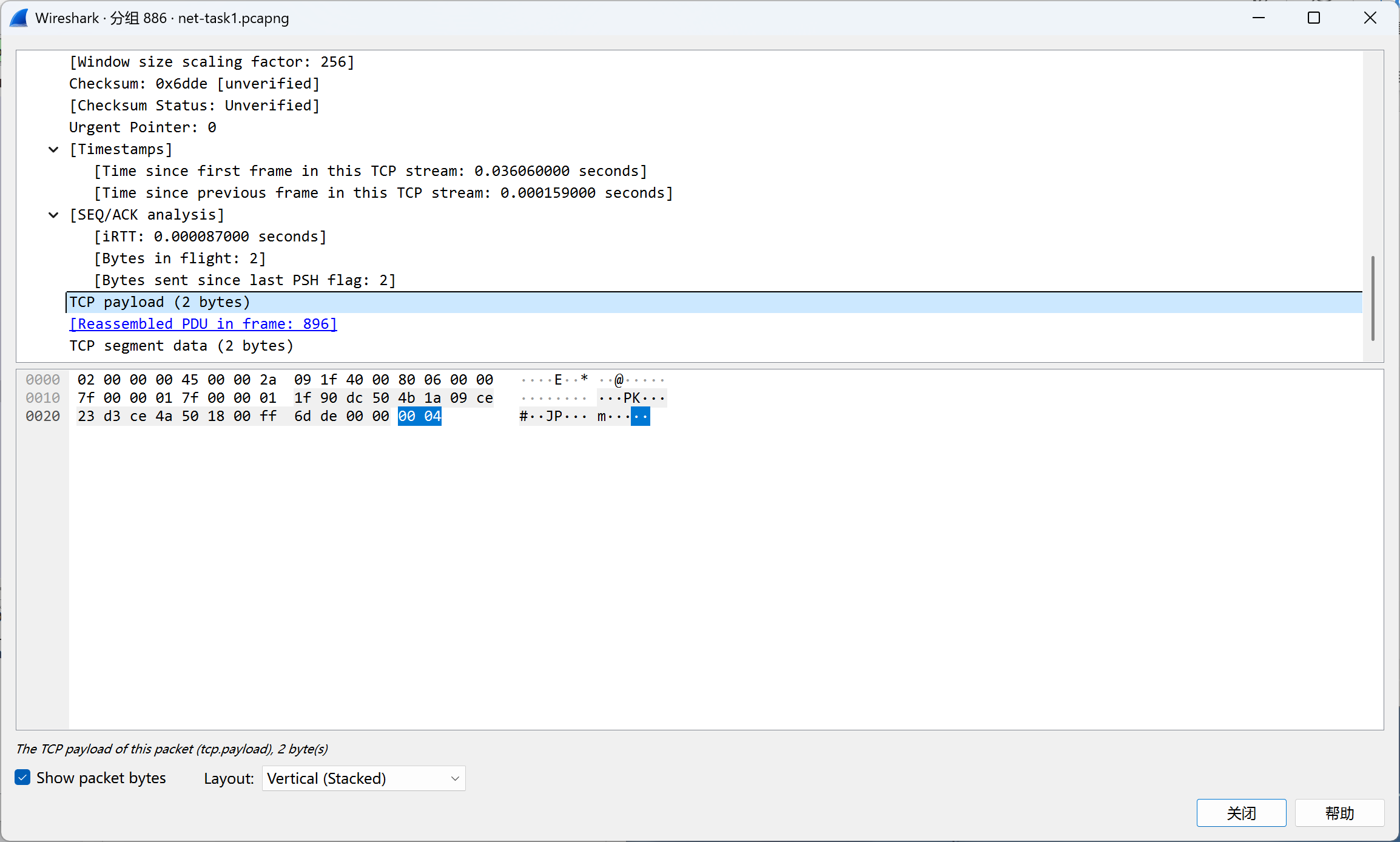
* Initialization 报文（Client→Server）：Type=1



* agree 报文（Server→Client）：Type=2



* reverseRequest 报文（Client→Server）：Type=3
* reverseAnswer 报文（Server→Client）：Type=4



**核心功能模块**

1. **文件处理**：
   * 使用 Files.readAllBytes 读取原始文件（ASCII 编码）
   * 通过 splitFile 方法将文件分割为不定长块：
     + 最后一块为剩余全部内容，其余块长度在 [Lmin, Lmax] 间随机生成
2. **网络交互**：
   * 发送 Initialization 报文（Type=1）和块数 N
   * 循环发送 reverseRequest 报文，包含数据块内容
   * 接收 reverseAnswer 报文并打印反转结果
3. **结果生成**：
   * 通过 combineReversedBlocks 方法将反转块逆序合并
   * 生成最终反转文件（原文件名 + .reversed 后缀）

**关键代码片段**

// 分割文件为不定长块

private static List<String> splitFile(String content, int Lmin, int Lmax) {

List<String> blocks = new ArrayList<>();

Random random = new Random();

int index = 0;

int totalLength = content.length();

while (index < totalLength) {

int maxPossible = totalLength - index;

int blockSize = (maxPossible <= Lmax) ? maxPossible :

Lmin + random.nextInt(Lmax - Lmin + 1);

blocks.add(content.substring(index, index + blockSize));

index += blockSize;

}

return blocks;

}

**服务器实现（reversetcpserver.java）**

**核心功能模块**

1. **多客户端处理**：
   * 使用 Executors.newCachedThreadPool() 创建线程池
   * 每个客户端连接由独立的 ClientHandler 线程处理
2. **报文处理流程**：
   * 接收 Initialization 报文并解析块数 N
   * 发送 agree 报文确认协议
   * 循环接收 reverseRequest 报文，处理数据反转
   * 发送 reverseAnswer 报文返回反转结果
3. **数据反转实现**：
   * 使用 StringBuilder.reverse() 方法高效反转字符串
   * 保持原始字节编码（US-ASCII）

**关键代码片段**

// 客户端处理线程

private static class ClientHandler implements Runnable {

@Override

public void run() {

try {

// 读取初始化报文

short type = in.readShort();

if (type != 1) return;

int blockCount = in.readInt();

// 发送同意报文

out.writeShort(2);

out.flush();

// 处理每个数据块

for (int i = 0; i < blockCount; i++) {

type = in.readShort();

if (type != 3) break;

int length = in.readInt();

byte[] data = new byte[length];

in.readFully(data);

// 反转数据

String reversed = new StringBuilder(new String(data, StandardCharsets.US\_ASCII))

.reverse().toString();

// 发送响应报文

out.writeShort(4);

out.writeInt(reversed.length());

out.writeBytes(reversed);

out.flush();

}

} catch (IOException e) {

// 异常处理

}

}

}

// 发送响应报文

out.writeShort(4);

out.writeInt(reversed.length());

out.writeBytes(reversed);

out.flush();

}

} catch (IOException e) {

// 异常处理

}

}

}

**不定长数据块处理**

* **问题**：客户端需按 [Lmin, Lmax] 分割数据块，最后一块为剩余内容
* **解决方案**：
  + 使用随机数生成中间块长度，最后一块取剩余字节
  + 通过 substring 方法精确截取数据块

**多客户端并发处理**

* **问题**：服务器需同时处理多个客户端请求
* **解决方案**：
  + 使用 ExecutorService 线程池管理客户端连接
  + 每个客户端由独立线程处理，避免阻塞

**报文格式解析**

* **问题**：确保客户端与服务器报文格式一致
* **解决方案**：
  + 严格遵循报文格式定义，使用 DataInputStream/DataOutputStream
  + 先读取 Type 字段验证报文类型，再解析后续字段

**涉及知识点总结**

**已掌握的核心知识点**

1. **TCP 网络编程**：
   * Socket 通信原理（客户端 / 服务器模型）
   * 字节流数据传输与报文格式定义
2. **Java 并发编程**：
   * 线程池（ExecutorService）的使用
   * 多线程资源隔离与异常处理
3. **文件操作**：
   * Java NIO 文件读取（Files.readAllBytes）
   * 文本编码处理（US-ASCII 格式）
4. **数据结构应用**：
   * 列表（ArrayList）存储数据块
   * 字符串反转（StringBuilder.reverse）

**通过本项目新增的知识**

1. **网络协议设计**：自定义应用层报文格式的完整流程
2. **并发服务器实现**：基于线程池的多客户端请求处理方案
3. **Wireshark 抓包分析**：网络报文的捕获与协议解析方法