计算机网络原理(三级项目)——

发送和接收TCP数据包

课 程 名 称：计算机网络原理(三级项目)

2025年4月

目 录

[摘 要 1](#_Toc6986)

[1 前言 1](#_Toc31946)

[1.1 项目背景 1](#_Toc28294)

[1.2 目的和意义 2](#_Toc6384)

[2 研究内容 3](#_Toc6612)

[2.1 TCP数据包发送与接收的基本原理 3](#_Toc4683)

[2.1.1. 项目内容 3](#_Toc1969)

[2.1.2. TCP数据包结构 3](#_Toc31181)

[2.1.3. 数据发送过程 3](#_Toc8323)

[2.1.4. 数据接收过程 3](#_Toc6510)

[2.1.5. 关键技术实现 4](#_Toc31475)

[2.2 研究工具 4](#_Toc30533)

[3 方案设计 5](#_Toc31278)

[3.1 客户端 5](#_Toc26935)

[3.1.1. 常量与类定义 5](#_Toc10786)

[3.1.2. 校验和计算模块 7](#_Toc28541)

[3.1.3. 数据包内容显示 7](#_Toc12424)

[3.1.4. 数据包发送 8](#_Toc32192)

[3.1.5. 数据包接收 9](#_Toc23039)

[3.1.6. 界面相关模块 10](#_Toc23230)

[3.1.7. 主函数 11](#_Toc13436)

[3.2 服务器端 14](#_Toc32258)

[3.2.1. 校验和计算函数 14](#_Toc7459)

[3.2.2. 打印TCP数据包的内容 15](#_Toc3289)

[3.2.3. 主函数初始化 16](#_Toc21978)

[3.2.4. 套接字初始化 16](#_Toc29721)

[3.2.5. 接受客户端连接 16](#_Toc10375)

[3.2.6. 主循环处理客户端请求 16](#_Toc11503)

[3.2.7. 资源清理 19](#_Toc3343)

[4 研究结果 20](#_Toc27984)

[4.1 系统启动流程 20](#_Toc12443)

[4.2 功能实例演示 20](#_Toc18627)

[4.2.1. 身份信息确认（正常情况） 20](#_Toc28583)

[4.2.2. 身份信息确认（余额不足） 21](#_Toc11413)

[4.2.3. 身份信息确认（车牌不存在） 22](#_Toc8680)

[4.2.4. 紧急通讯功能 23](#_Toc15048)

[4.2.5. 显示TCP数据包详细信息功能 24](#_Toc19984)

[4.2.6. 系统关闭 26](#_Toc24875)

[4.3 错误处理情况 26](#_Toc19699)

[4.3.1. 输入无效选项 26](#_Toc13951)

[4.3.2. 服务器未启动时客户端连接 27](#_Toc24225)

[参考文献 29](#_Toc13727)

发送和接收TCP数据包

# 摘 要

本项目以ETC（电子不停车收费）系统为应用场景，设计并实现了一个基于TCP协议的数据包发送与接收程序。客户端模块通过Python实现，手动构造TCP头部与负载数据，包括源端口、目标端口、序列号、标志位及校验和等关键字段，完成数据包封装并发送至服务器端。服务器端接收TCP数据包后进行解析和身份验证，实现车牌信息确认和扣费处理。本系统采用socket和struct库进行套接字通信与二进制数据处理，实现了TCP校验和计算、三次握手连接建立、数据发送与接收、紧急通讯等功能。通过功能测试与错误处理验证，系统能够稳定处理正常身份验证、余额不足、车牌不存在等多种业务场景，并能展示TCP数据包详细信息，便于通信过程分析。本项目有效深化了对TCP协议结构与网络编程原理的理解，为网络应用开发提供了实践基础。

**关键词**：**TCP协议；ETC系统；数据包发送与接收；校验和计算；套接字通信**

# 前言

## 项目背景

随着互联网的迅速普及和计算机网络技术的不断升级，各种数据通信协议在现代信息社会中扮演了举足轻重的角色。其中，TCP（传输控制协议）作为一种面向连接、可靠的数据传输协议，在保障数据传输完整性与顺序性方面具有不可替代的优势。TCP不仅能够通过序列号、确认应答、超时重传以及流量和拥塞控制等机制确保数据在不同网络环境下的可靠传输，而且为电子商务、实时通信、视频传输等多种应用场景提供了坚实的基础。TCP在计算机网络中扮演着至关重要的角色，其重要性主要体现在以下几个方面：

1. 可靠性

TCP是一种面向连接的协议，它通过序列号、确认应答、重传机制等技术保证了数据的可靠传输。这意味着在数据传输过程中，TCP能够检测到丢失、损坏或重复的数据，并进行相应的处理，确保数据的完整性和正确性。

1. 流量控制

TCP能够根据接收端的处理能力和网络拥塞程度来控制发送端的数据发送速率，从而避免了过多数据注入网络造成的拥塞问题。通过TCP的流量控制机制，可以有效地平衡网络各个节点之间的数据传输速率，提高网络的整体性能。

1. 拥塞控制

TCP通过拥塞窗口、慢启动、拥塞避免等算法来控制网络拥塞的发生和程度。当网络出现拥塞时，TCP会降低发送速率，减少数据包的丢失率，从而维持网络的稳定性和吞吐量。

1. 顺序传输

TCP保证数据包按照发送的顺序到达接收端，这对于许多应用程序的正确运行至关重要，如文件传输、网页浏览等。TCP通过序列号和确认应答机制来保证数据包的顺序传输，确保了数据的正确性和完整性。

1. 面向连接

TCP是一种面向连接的协议，它在数据传输前需要建立连接，在传输完成后需要释放连接。这种连接的建立和释放过程可以保证通信的可靠性和安全性，同时也可以为网络管理和资源分配提供便利。

而在现实应用中，如ETC（电子不停车收费）等系统要求高速、高效且安全的数据交换，因此深入理解TCP报文结构、手动构造数据包以及模拟网络通信，对于研究网络编程及提升系统构架设计能力具有重要意义。

## 目的和意义

## 项目设计目的

设计一个发送和接收TCP数据包的程序，其功能是填充一个TCP数据包，发送给目的主机，并在目的主机接收此TCP数据包，将数据字段显示显示在标准输出上。在设计中熟悉TCP数据包的结构以及TCP协议与IP协议的关系，以提高对TCP协议的工作原理的认识。

## 项目设计意义

（一）理解TCP协议的重要性

TCP是计算机网络中最基础、最重要的协议之一。它负责可靠的数据传输，对于网络通信的可靠性和效率至关重要。

（二）实践能力的培养

通过设计这样一个项目，可以实际动手编程实现TCP数据包的发送和接收，从而更深入地理解TCP协议的工作原理。

（三）网络编程的基础

掌握TCP数据包的发送和接收是进行网络编程的基础。这对于想要从事网络开发或系统管理的学生来说是一项必备的技能。

（四）加深对网络通信的理解

通过实际操作，可以更好地理解网络通信的各个环节，包括连接建立、数据传输、错误处理等，从而提升对计算机网络工作原理的整体理解。

（五）培养解决问题的能力

在设计和实现过程中，对于面临各种问题和挑战，需要动手解决。这有助于培养问题解决能力和创新思维。

# 研究内容

## TCP数据包发送与接收的基本原理

## 项目内容

本次项目通过Python实现了TCP协议的数据包发送与接收功能，模拟了TCP协议的基本通信过程。TCP（传输控制协议）是一种面向连接的、可靠的、基于字节流的传输层通信协议，它通过序列号、确认应答、重传机制等确保数据的可靠传输。

## TCP数据包结构

TCP数据包由头部和数据负载两部分组成。头部包含以下关键字段：

源端口和目的端口：标识发送和接收应用程序

序列号：标识数据包的顺序

确认号：期望收到的下一个数据包的序列号

数据偏移：指定TCP头部的长度

标志位：包括SYN、ACK、FIN等控制位

窗口大小：用于流量控制

校验和：用于错误检测

紧急指针：标识紧急数据的位置

## 数据发送过程

(1) 建立连接：首先创建TCP套接字，与接收端建立三次握手连接。发送SYN包，接收SYN-ACK响应，最后发送ACK确认。

(2) 数据封装：将用户输入的消息封装为TCP数据包。设置适当的序列号、标志位（PSH+ACK表示推送数据）等头部字段。

(3) 校验和计算：对TCP头部和数据负载计算校验和，用于接收端验证数据完整性。

(4) 数据发送：将封装好的数据包发送到接收端。

(5) 等待确认：发送后等待接收端的ACK确认，如果收到确认则更新序列号，否则可能需要重传。

(6) 连接终止：当用户输入"quit"时，发送结束消息并关闭连接。

## 数据接收过程

(1) 监听端口：创建套接字并绑定到指定端口，等待连接请求。

(2) 接受连接：完成三次握手过程，建立TCP连接。

(3) 接收数据：从网络接收数据包，首先解析TCP头部信息。

(4) 校验和验证：重新计算校验和并与接收到的校验和比较，确保数据没有损坏。

(5) 数据处理：提取数据负载部分，解码为可读文本并显示。

(6) 发送确认：根据接收到的数据包序列号和长度，计算确认号并发送ACK响应。

(7) 连接管理：收到结束消息后关闭连接。

## 关键技术实现

(1) 序列号管理：每个数据包都有唯一的序列号，接收端通过确认号告知已成功接收的数据。

(2) 确认机制：接收方对每个收到的数据包发送ACK确认，发送方根据ACK 确定数据是否成功送达。

(3) 错误检测：通过校验和机制检测数据传输过程中是否发生错误。

(4) 流量控制：通过窗口字段实现基本的流量控制功能。

(5) 状态显示：详细显示每个数据包的头部信息和内容，便于观察通信过程。

## 研究工具

## 开发工具

表1 项目开发工具

| **工具类别** | **具体工具** | **用途说明** |
| --- | --- | --- |
| 编程语言 | Python 3 | 主要开发语言 |
| 核心库 | socket, struct | 网络通信和二进制打包/解包 |
| IDE环境 | PyCharm | 代码编写与调试 |
| 版本控制 | Git | 代码版本管理 |

## 分析工具

表2 项目分析工具

| **工具名称** | **应用场景** | **关键功能** |
| --- | --- | --- |
| Wireshark | 网络报文捕获 | 验证TCP三次握手/四次挥手过程 |
| tcpdump | 命令行抓包 | 原始数据包记录与分析 |
| netstat | 连接状态监控 | 查看TCP连接状态 |
| ping/traceroute | 网络诊断 | 测试基础网络连通性 |

# 方案设计

## 客户端

ETC车辆端借助Windows Sockets库，先获取用户输入的服务器及车辆自身的IP和端口信息，完成Winsock库初始化和套接字创建连接。然后构造IP、TCP及TCP伪首部数据包，计算校验和。进入循环，根据用户选择的操作指令，实现车辆身份信息确认（含上传车牌及验证扣费）、紧急通讯以及系统关闭等功能，并在结束时清理相关网络资源。

## 常量与类定义

# TCP标志位常量

FIN = 0x01

SYN = 0x02

RST = 0x04

PSH = 0x08

ACK = 0x10

URG = 0x20

# TCP报文头部结构

class TCPHeader:

def \_\_init\_\_(self, src\_port, dst\_port, seq\_num=0, ack\_num=0,

data\_offset=5, reserved=0, flags=0, window=1024,

checksum=0, urgent\_ptr=0):

self.src\_port = src\_port

self.dst\_port = dst\_port

self.seq\_num = seq\_num

self.ack\_num = ack\_num

self.data\_offset = data\_offset # 头部长度，单位是4字节

self.reserved = reserved

self.flags = flags

self.window = window

self.checksum = checksum

self.urgent\_ptr = urgent\_ptr

def pack(self):

data\_offset\_reserved\_flags = (self.data\_offset << 12) + (self.reserved << 6) + self.flags

header = struct.pack('!HHLLHHHH',

self.src\_port,

self.dst\_port,

self.seq\_num,

self.ack\_num,

data\_offset\_reserved\_flags,

self.window,

self.checksum,

self.urgent\_ptr)

return header

@staticmethod

def unpack(packed\_header):

unpacked = struct.unpack('!HHLLHHHH', packed\_header)

src\_port = unpacked[0]

dst\_port = unpacked[1]

seq\_num = unpacked[2]

ack\_num = unpacked[3]

data\_offset\_reserved\_flags = unpacked[4]

data\_offset = (data\_offset\_reserved\_flags >> 12) & 0xF

reserved = (data\_offset\_reserved\_flags >> 6) & 0x3F

flags = data\_offset\_reserved\_flags & 0x3F

window = unpacked[5]

checksum = unpacked[6]

urgent\_ptr = unpacked[7]

return TCPHeader(src\_port, dst\_port, seq\_num, ack\_num,

data\_offset, reserved, flags, window,

checksum, urgent\_ptr)

def get\_flags\_description(self):

"""返回TCP标志位的描述信息"""

flags\_str = []

if self.flags & FIN:

flags\_str.append("FIN")

if self.flags & SYN:

flags\_str.append("SYN")

if self.flags & RST:

flags\_str.append("RST")

if self.flags & PSH:

flags\_str.append("PSH")

if self.flags & ACK:

flags\_str.append("ACK")

if self.flags & URG:

flags\_str.append("URG")

return "|".join(flags\_str) if flags\_str else "无"

def print\_header\_info(self, packet\_type="未知"):

"""打印TCP头部详细信息"""

print("=" \* 60)

print(f"TCP {packet\_type}详细信息:")

print("-" \* 60)

print(f"源端口: {self.src\_port}")

print(f"目标端口: {self.dst\_port}")

print(f"序列号: {self.seq\_num}")

print(f"确认号: {self.ack\_num}")

print(f"头部长度: {self.data\_offset \* 4} 字节")

print(f"标志位: {self.flags:06b} ({self.get\_flags\_description()})")

print(f"窗口大小: {self.window}")

print(f"校验和: 0x{self.checksum:04x}")

print(f"紧急指针: {self.urgent\_ptr}")

print("=" \* 60)

该模块定义了TCP标志位常量如FIN、SYN等。TCPHeader类用于处理TCP报文头部，包含构造函数、打包解包方法，能获取标志位描述信息并打印头部详细信息。

## 校验和计算模块

# 计算校验和

def calculate\_checksum(data):

if len(data) % 2 == 1:

data += b'\x00'

checksum = 0

for i in range(0, len(data), 2):

word = (data[i] << 8) + data[i + 1]

checksum += word

checksum = (checksum & 0xFFFF) + (checksum >> 16)

checksum = ~checksum & 0xFFFF

return checksum

calculate\_checksum函数保障数据可靠传输。它先处理数据长度奇偶性，再按双字节分组累加，经特定算法防止溢出，取反截取得校验和。接收方以此对比判断数据传输有无错误，确保数据完整性。

## 数据包内容显示

# 显示数据包内容（简化或详细版）

def dump\_packet\_data(data, prefix="", detailed=False):

if not detailed:

try:

# 提取TCP头后的数据作为消息

if len(data) > 20: # 至少有TCP头

message = data[20:].decode('utf-8')

print(f"{prefix}数据内容: \"{message}\"")

else:

print(f"{prefix}无有效数据")

except UnicodeDecodeError:

print(f"{prefix}无法解码的数据")

return

# 详细模式

print(f"{prefix}数据包内容 ({len(data)} 字节):")

print("-" \* 60)

# 显示头部和部分数据

display\_size = min(len(data), 40)

hex\_view = ""

ascii\_view = ""

for i in range(display\_size):

if i % 16 == 0 and i > 0:

print(f"{hex\_view} {ascii\_view}")

hex\_view = ""

ascii\_view = ""

hex\_view += f"{data[i]:02x} "

ascii\_view += chr(data[i]) if 32 <= data[i] <= 126 else "."

# 打印最后一行

if hex\_view:

hex\_view += " " \* (16 - (display\_size % 16))

print(f"{hex\_view} {ascii\_view}")

if len(data) > display\_size:

print(f"... (还有 {len(data) - display\_size} 字节未显示)")

print("-" \* 60)

dump\_packet\_data函数按detailed参数展示数据包。简化模式下提取并解码TCP头后数据，详细模式以十六进制和ASCII码对照展示，超40字节会提示剩余，助于分析数据格式、排查编码及损坏问题。

## 数据包发送

# 发送TCP数据包

def send\_tcp\_packet(sock, src\_port, dst\_port, seq\_num, message, detailed=False):

data = message.encode('utf-8')

# 构造TCP头部

header = TCPHeader(

src\_port=src\_port,

dst\_port=dst\_port,

seq\_num=seq\_num,

flags=PSH | ACK, # PSH+ACK

window=1024

)

# 计算校验和

header\_bytes = header.pack()

header.checksum = calculate\_checksum(header\_bytes + data)

header\_bytes = header.pack()

# 显示详细信息（如果需要）

if detailed:

header.print\_header\_info("发送数据包")

# 发送数据包

packet = header\_bytes + data

if detailed:

dump\_packet\_data(packet, "发送", detailed)

else:

print(f"发送: \"{message}\"")

sock.send(packet)

return len(data)

send\_tcp\_packet函数构建并发送 TCP 数据包。接收套接字等参数，编码消息，构造 TCPHeader，计算校验和，按需打印详细信息，拼接数据后发送，返回消息长度，完成数据包发送流程。

## 数据包接收

# 接收TCP数据包

def recv\_tcp\_packet(sock, detailed=False):

data = sock.recv(1024)

if not data:

return None, None

# 如果数据包长度小于TCP头部长度，返回无效

if len(data) < 20:

print("接收到无效的数据包（长度小于20字节）")

return None, None

# 解析TCP头部

header = TCPHeader.unpack(data[:20])

payload = data[20:]

# 显示详细信息（如果需要）

if detailed:

header.print\_header\_info("接收数据包")

dump\_packet\_data(data, "接收", detailed)

else:

dump\_packet\_data(data, "接收", False)

try:

message = payload.decode('utf-8')

return header, message

except UnicodeDecodeError:

print("接收到无法解码的数据")

return header, None

recv\_tcp\_packet函数从服务器接收并解析TCP数据包。接收数据后，先判断长度，再解析TCP头部，尝试解码负载数据，调用时若指定detailed会打印详细信息，方便排查数据问题。

## 界面相关模块

# 清屏函数

def clear\_screen():

os.system('cls' if os.name == 'nt' else 'clear')

# 显示菜单

def display\_menu():

print("\n" + "-" \* 60)

print("ETC收费不停车系统 - 车辆端")

print("-" \* 60)

print("1. 身份验证与扣费")

print("2. 紧急通讯")

print("3. 显示TCP数据包详细信息")

print("4. 退出系统")

print("-" \* 60)

choice = input("请选择功能 (1-4): ")

return choice

clear\_screen函数依操作系统清屏，Windows用cls，类Unix用clear，提供清晰界面。display\_menu函数展示功能菜单，接收用户选择并返回，是用户与系统交互的入口。

## 主函数

def main():

clear\_screen()

print("\n\*\*\*\*\* ETC收费不停车系统 - 车辆端 \*\*\*\*\*\n")

# 获取服务器连接信息

server\_ip = input("请输入服务器IP地址: ")

server\_port = int(input("请输入服务器端口号: "))

print("\n正在连接到服务器...")

# 创建TCP套接字并连接

try:

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((server\_ip, server\_port))

print("连接成功!")

except socket.error as e:

print(f"连接失败: {e}")

sys.exit(1)

src\_port = client\_socket.getsockname()[1] # 获取本地分配的端口

seq\_num = 0

# 主循环

while True:

choice = display\_menu()

if choice == "1": # 身份验证

clear\_screen()

print("\n\*\*\* 身份验证与扣费 \*\*\*\n")

# 发送身份验证请求

send\_tcp\_packet(client\_socket, src\_port, server\_port, seq\_num, "1")

seq\_num += 1

# 接收服务器确认

header, response = recv\_tcp\_packet(client\_socket)

if not header or not response:

continue

# 输入车牌号

plate = input("请输入车牌号: ")

send\_tcp\_packet(client\_socket, src\_port, server\_port, seq\_num, plate)

seq\_num += len(plate)

# 接收验证结果

header, result = recv\_tcp\_packet(client\_socket)

if header and result:

print("\n验证结果:", result)

input("\n按Enter键返回主菜单...")

clear\_screen()

elif choice == "2": # 紧急通讯

clear\_screen()

print("\n\*\*\* 紧急通讯 \*\*\*\n")

# 发送通讯请求

send\_tcp\_packet(client\_socket, src\_port, server\_port, seq\_num, "2")

seq\_num += 1

# 接收服务器确认

header, response = recv\_tcp\_packet(client\_socket)

if not header or not response:

continue

print("\n已建立通讯。输入'exit'结束通讯。")

# 通讯循环

while True:

message = input("\n发送消息: ")

send\_tcp\_packet(client\_socket, src\_port, server\_port, seq\_num, message)

seq\_num += len(message)

if message.lower() == "exit":

# 接收服务器的通讯结束确认

header, end\_msg = recv\_tcp\_packet(client\_socket)

break

# 接收服务器回复

header, reply = recv\_tcp\_packet(client\_socket)

if not header or not reply:

break

input("\n按Enter键返回主菜单...")

clear\_screen()

elif choice == "3": # 显示TCP数据包详细信息

clear\_screen()

print("\n\*\*\* TCP数据包详细信息 \*\*\*\n")

# 发送数据包详情请求

send\_tcp\_packet(client\_socket, src\_port, server\_port, seq\_num, "3")

seq\_num += 1

# 接收服务器确认

header, response = recv\_tcp\_packet(client\_socket)

if not header or not response:

continue

print(f"服务器响应: {response}")

# 让用户输入要发送的测试数据

test\_message = input("\n请输入要发送的测试数据: ")

print("\n----发送数据包详细信息----")

send\_tcp\_packet(client\_socket, src\_port, server\_port, seq\_num, test\_message, detailed=True)

seq\_num += len(test\_message)

# 接收服务器的响应数据包

print("\n----接收数据包详细信息----")

header, server\_response = recv\_tcp\_packet(client\_socket, detailed=True)

if header and server\_response:

print(f"\n服务器回复消息: {server\_response}")

input("\n按Enter键返回主菜单...")

clear\_screen()

elif choice == "4": # 退出系统

print("\n正在关闭系统...")

send\_tcp\_packet(client\_socket, src\_port, server\_port, seq\_num, "4")

# 等待服务器确认

try:

header, response = recv\_tcp\_packet(client\_socket)

print("服务器回复:", response if response else "无响应")

except:

pass

break

else:

print("无效的选择，请重新输入!")

time.sleep(1)

clear\_screen()

# 关闭连接

client\_socket.close()

print("\n系统已关闭")

if \_\_name\_\_ == "\_\_main\_\_":

try:

main()

except KeyboardInterrupt:

print("\n程序被用户中断")

sys.exit(0)

main函数把控客户端核心流程。首先引导用户输入服务器信息并建立连接，随即进入一个循环。在循环中，程序展示用户菜单，根据用户选择执行不同功能。若用户选择身份验证，系统会校验用户身份信息；选择紧急通讯，则开启与服务器的紧急联络通道。只要用户不选择退出，此循环就会持续，不断协调各模块，实现ETC客户端与服务器的交互操作。

* 1. **服务器端**

ETC服务器端程序先获取用户输入端口，进行套接字创建与绑定等操作，接着监听客户端连接请求。成功连接后，接收客户端指令，根据指令执行对应操作，如处理车牌信息验证与扣费、进行紧急通讯交互、接收并显示客户端发送的测试数据包，收到结束指令时关闭系统，并在程序结束时清理相关资源。

## 校验和计算函数

def calculate\_checksum(data):

    if len(data) % 2 == 1:

        data += b'\x00'

    checksum = 0

    for i in range(0, len(data), 2):

        word = (data[i] << 8) + data[i + 1]

        checksum += word

        checksum = (checksum & 0xFFFF) + (checksum >> 16)

    checksum = ~checksum & 0xFFFF

    return checksum

该calculate\_checksum函数用于计算缓冲区数据的校验和，通过累加16位数据，处理奇数长度数据，多次相加高低16位并取反得到校验和。

## 打印TCP数据包的内容

def dump\_packet\_data(data, prefix="", detailed=False):

    if not detailed:

        try:

            # 提取TCP头后的数据作为消息

            if len(data) > 20:  # 至少有TCP头

                message = data[20:].decode('utf-8')

                print(f"{prefix}数据内容: \"{message}\"")

            else:

                print(f"{prefix}无有效数据")

        except UnicodeDecodeError:

            print(f"{prefix}无法解码的数据")

        return

    # 详细模式

    print(f"{prefix}数据包内容 ({len(data)} 字节):")

    print("-" \* 60)

    # 显示头部和部分数据

    display\_size = min(len(data), 40)

    hex\_view = ""

    ascii\_view = ""

    for i in range(display\_size):

        if i % 16 == 0 and i > 0:

            print(f"{hex\_view}  {ascii\_view}")

            hex\_view = ""

            ascii\_view = ""

        hex\_view += f"{data[i]:02x} "

        ascii\_view += chr(data[i]) if 32 <= data[i] <= 126 else "."

    # 打印最后一行

    if hex\_view:

        hex\_view += "   " \* (16 - (display\_size % 16))

        print(f"{hex\_view}  {ascii\_view}")

    if len(data) > display\_size:

        print(f"... (还有 {len(data) - display\_size} 字节未显示)")

    print("-" \* 60)

函数dump\_packet\_data用于以两种模式（普通模式和详细模式）打印数据包的内容。普通模式仅显示数据包中TCP头后的UTF-8解码内容，而详细模式则显示数据包的十六进制和ASCII视图。

## 主函数初始化

def main():

    clear\_screen()

    print("\n\*\*\*\*\* ETC收费不停车系统 - 服务器端 \*\*\*\*\*\n")

    # 设置监听参数

    port = int(input("请输入监听端口号: "))

ETC服务器端的启动和监听部分，提示用户输入要监听的端口号。

## 套接字初始化

try:

        server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

        server\_socket.bind(('', port))

        server\_socket.listen(1)

        print(f"\n服务器启动成功，正在监听端口 {port}...")

        print("-" \* 60)

        print("等待车辆连接...\n")

    except socket.error as e:

        print(f"服务器启动失败: {e}")

        sys.exit(1)

创建基于TCP协议的服务器套接字，将套接字绑定到指定IP和端口并开始监听客户端连接请求。根据连接结果打印相应信息。

## 接受客户端连接

  client\_socket, client\_addr = server\_socket.accept()

    print(f"车辆已连接: {client\_addr[0]}:{client\_addr[1]}")

    print("-" \* 60)

使用accept函数接受客户端的连接请求，若成功则输出对应客户端信息。

## 主循环处理客户端请求

while True:

        header, command = recv\_tcp\_packet(client\_socket)

        if not header or not command:

            print("连接已断开")

            break

        # 根据命令进行处理

        if command == "1":  # 身份验证功能

            # 发送确认消息

            print("收到身份验证请求")

            send\_tcp\_packet(client\_socket, src\_port, header.src\_port, seq\_num, "请发送车牌号")

            seq\_num += 12

            # 接收车牌号

            plate\_header, plate = recv\_tcp\_packet(client\_socket)

            if not plate\_header or not plate:

                continue

            print(f"收到车牌号: {plate}")

            # 验证车牌号

            found = False

            response = ""

            for vehicle in vehicles:

                if vehicle.plate == plate:

                    found = True

                    if vehicle.balance >= 5.0:

                        vehicle.balance -= 5.0

                        response = f"身份验证成功，已扣费5元，余额{vehicle.balance:.2f}元"

                    else:

                        response = "余额不足，请走人工通道"

                    break

            if not found:

                response = "身份验证失败，请走人工通道"

            # 发送验证结果

            send\_tcp\_packet(client\_socket, src\_port, header.src\_port, seq\_num, response)

            seq\_num += len(response)

        elif command == "2":  # 通讯功能

            print("收到通讯请求")

            send\_tcp\_packet(client\_socket, src\_port, header.src\_port, seq\_num, "已建立通讯连接")

            seq\_num += 16

            while True:

                # 接收客户端消息

                msg\_header, message = recv\_tcp\_packet(client\_socket)

                if not msg\_header or not message:

                    break

                if message.lower() == "exit":

                    send\_tcp\_packet(client\_socket, src\_port, header.src\_port, seq\_num, "通讯已结束")

                    seq\_num += 12

                    break

                # 发送回复

                reply = input("请输入回复消息: ")

                send\_tcp\_packet(client\_socket, src\_port, header.src\_port, seq\_num, reply)

                seq\_num += len(reply)

        elif command == "3":  # 详细显示TCP数据包

            print("收到数据包详情请求")

            send\_tcp\_packet(client\_socket, src\_port, header.src\_port, seq\_num, "请输入测试数据")

            seq\_num += 12

            # 等待接收客户端测试数据包

            print("等待客户端发送测试数据包...")

            test\_header, test\_message = recv\_tcp\_packet(client\_socket, detailed=True)

            if test\_header and test\_message:

                print(f"收到客户端测试数据: {test\_message}")

                # 发送响应数据包

                response\_message = f"服务器已收到您的测试数据: {test\_message}"

                print("\n发送响应数据包...")

                send\_tcp\_packet(client\_socket, src\_port, test\_header.src\_port, seq\_num, response\_message, detailed=True)

                seq\_num += len(response\_message)

        elif command == "4":  # 退出系统

            print("收到退出请求，系统即将关闭")

            send\_tcp\_packet(client\_socket, src\_port, header.src\_port, seq\_num, "服务器已断开连接")

            break

        else:

            print(f"收到未知命令: {command}")

            send\_tcp\_packet(client\_socket, src\_port, header.src\_port, seq\_num, "未知命令")

            seq\_num += 8

    # 关闭连接

    client\_socket.close()

    server\_socket.close()

    print("\n服务器已关闭")

进入无限循环，接收客户端发送的TCP数据包，解析出头部信息和命令判断是否断开连接。若指令为1，允许客户端上传车牌信息数据包，接收数据包并解析车牌号，在预存数组中查找车辆信息，根据余额情况返回相应消息；若指令为2，允许客户端进入通讯，与客户端进行消息交互后结束通讯；若指令为3，接收客户端发送的测试数据包，并显示详细信息，然后发送响应数据包；若指令为4，则关闭系统。

## 资源清理

# 关闭连接

client\_socket.close()

    server\_socket.close()

    print("\n服务器已关闭")

关闭服务器和客户端套接字。

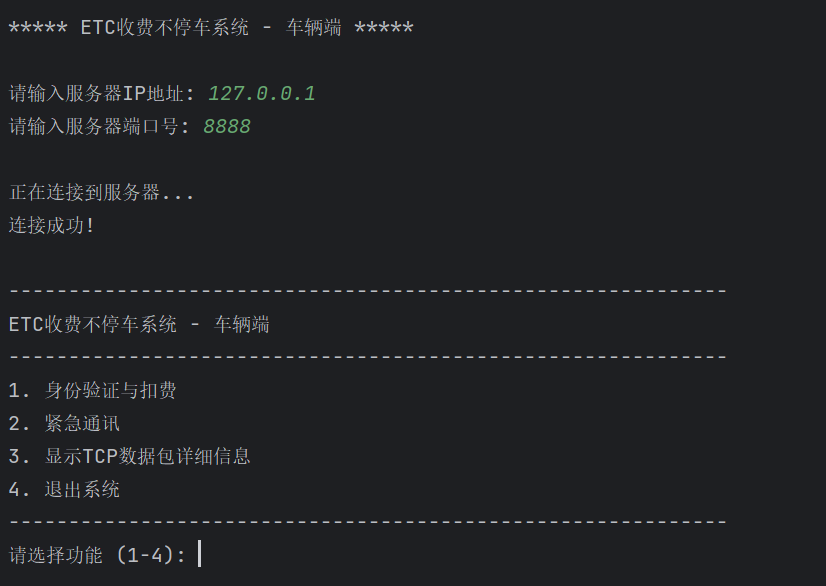
# 研究结果

## 系统启动流程

服务器端启动：

  
图4-1 服务器端启动图

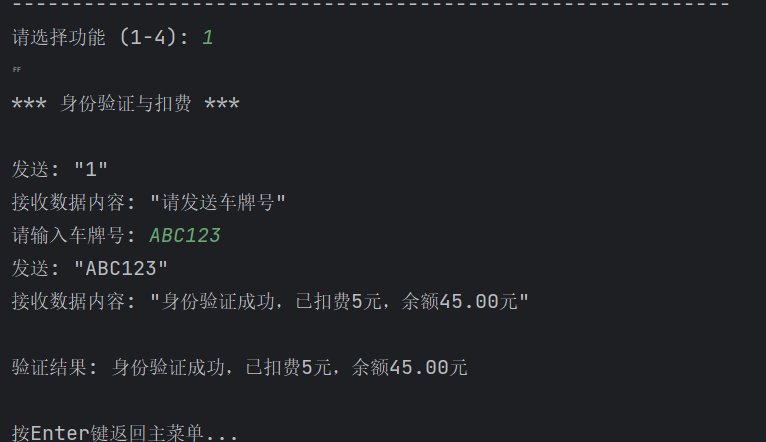
客户端启动：

  
图4-2 客户器端启动图

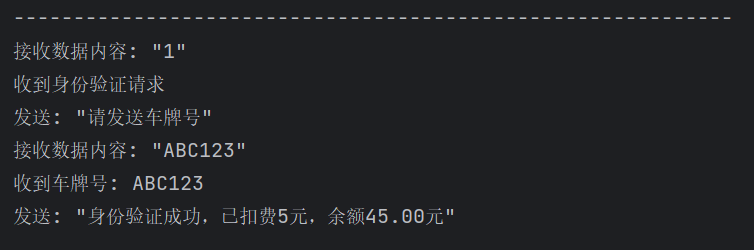
## 功能实例演示

## 身份信息确认（正常情况）

客户端操作：

  
图4-3 客户端发送身份信息图

服务器端显示：

  
图4-4 服务器确认身份信息图

## 身份信息确认（余额不足）

客户端操作：

 图4-5 客户端发送身份信息图

服务器显示：

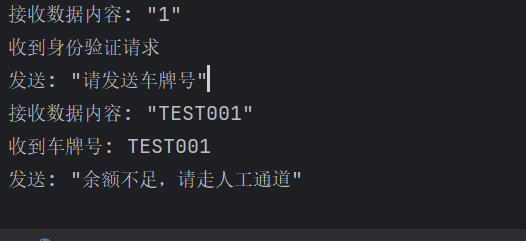


图4-6 服务器确认身份信息图

## 身份信息确认（车牌不存在）

客户端操作：



图4-7 客户端发送身份信息图

服务器显示：

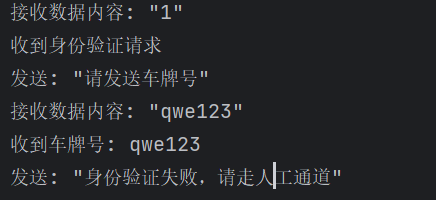
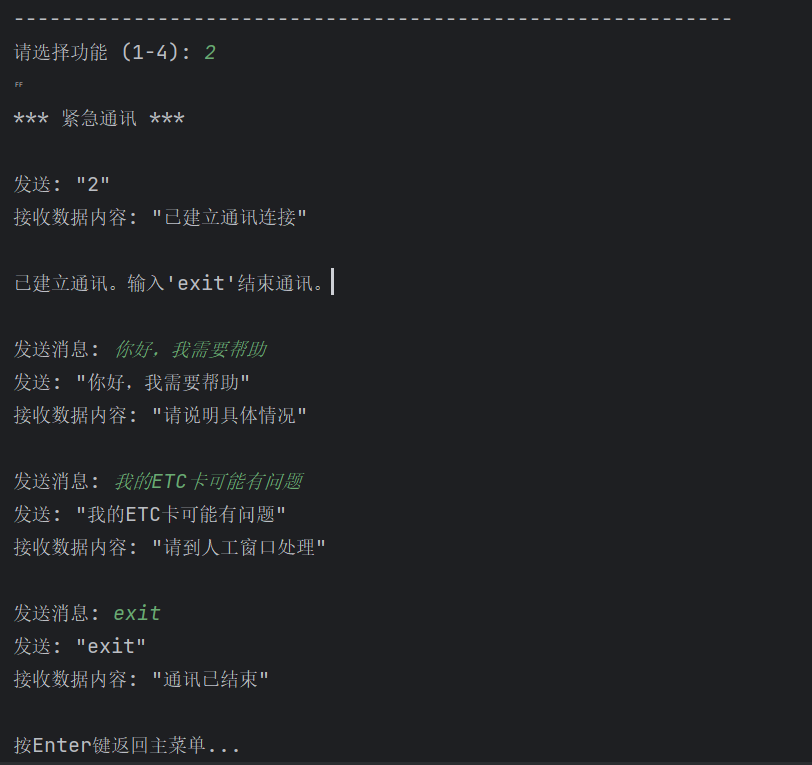


图4-8 服务器确认身份信息图

## 紧急通讯功能

客户端操作：

图4-9 客户端发起紧急通讯图

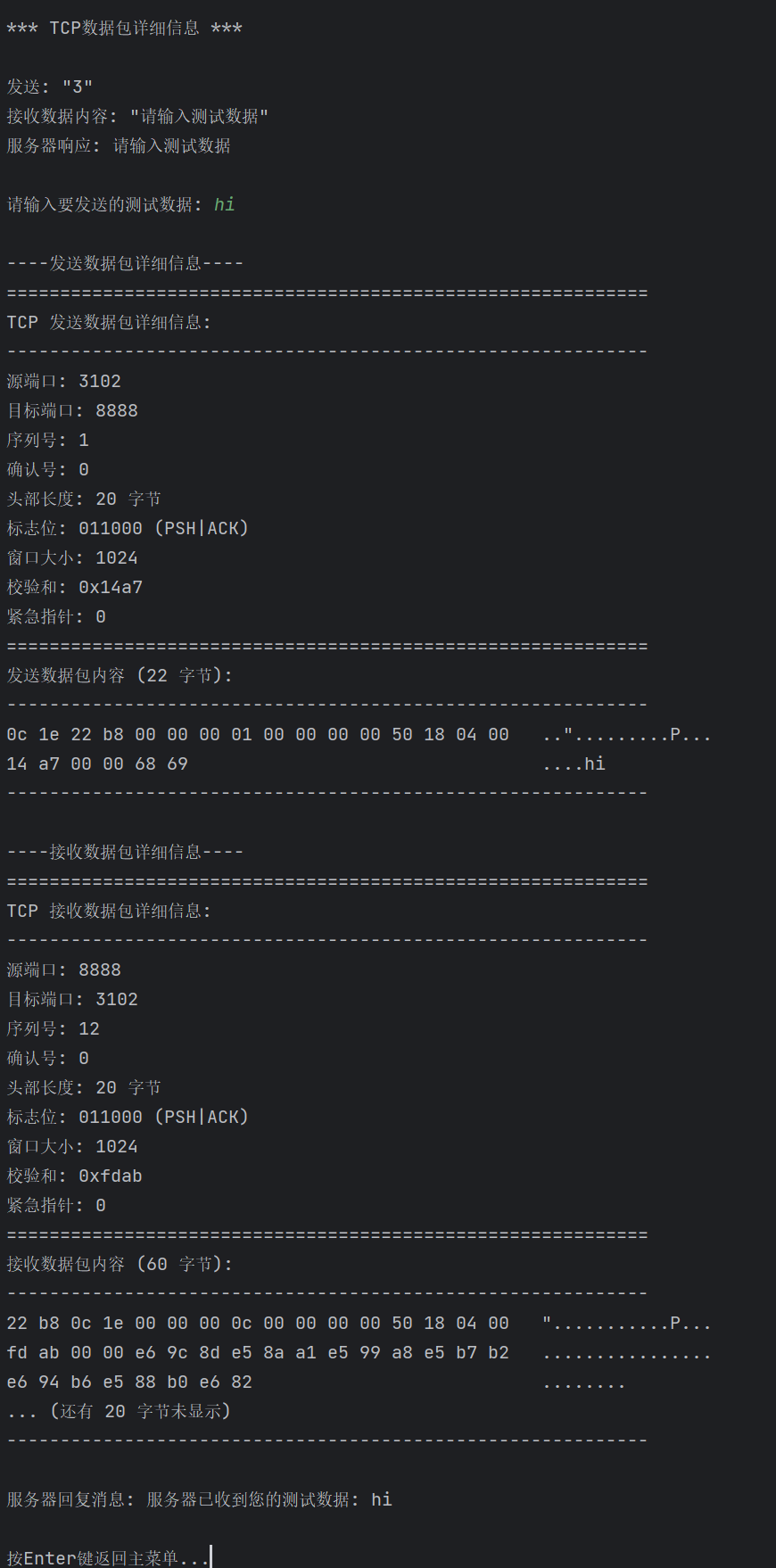
服务器显示：



图4-10 服务器端给客户端相应紧急通讯图

## 显示TCP数据包详细信息功能

客户端操作：

图4-11 客户端显示TCP数据包详细信息功能图

服务器显示：

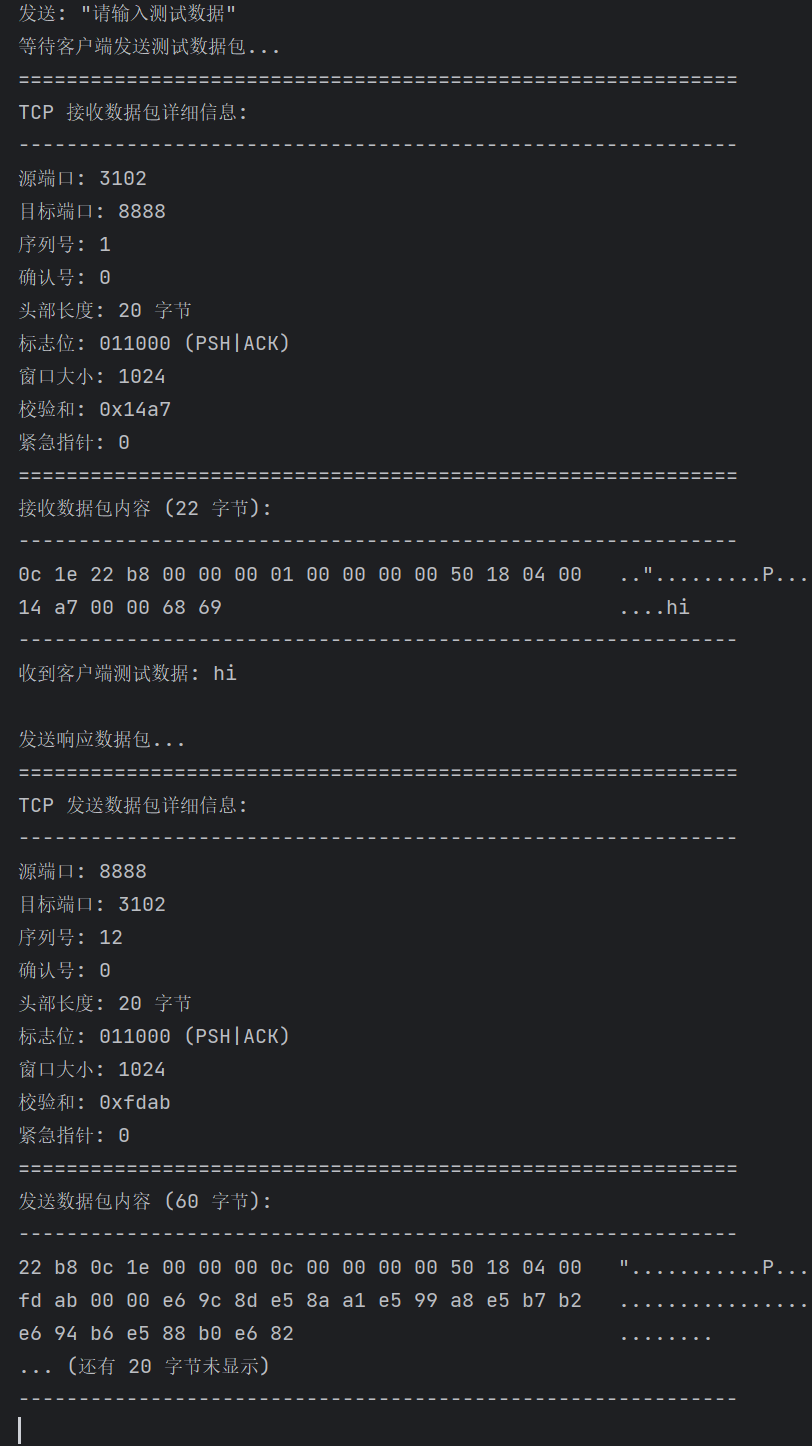


图4-12 服务器端显示TCP数据包详细信息功能图

## 系统关闭

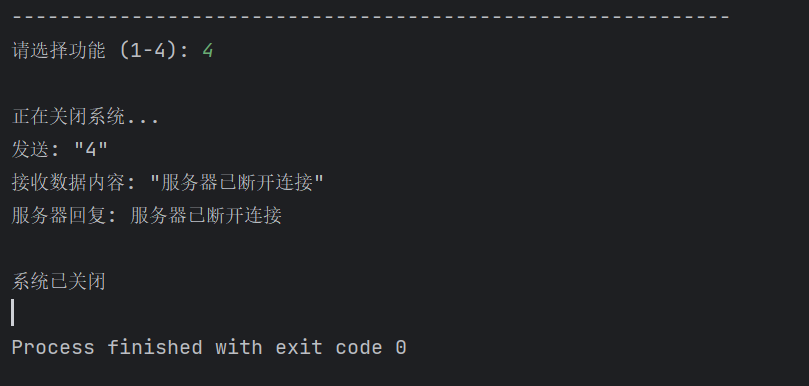
客户端操作：

图4-13 客户端关闭系统图

服务器显示：

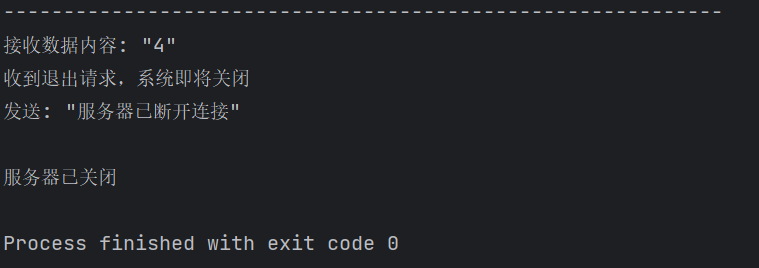


图4-14 服务器端接收到客户端的退出指令图

## 错误处理情况

## 输入无效选项

客户端操作：

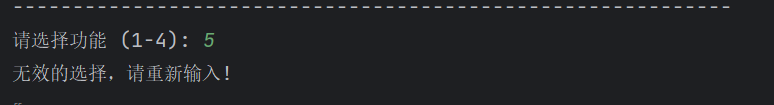


图4-15客户端输入的无效指令图

## 服务器未启动时客户端连接

**客户端显示：**

图4-16客户端连接服务器失败图

总 结

本项目围绕ETC系统中车辆与服务器之间的通信需求，成功设计并实现了一个基于TCP协议的数据包发送与接收程序。通过Python编程语言与socket库，实现了从客户端构造数据包、发送请求到服务器端接收解析、业务处理的完整流程。

在客户端部分，项目实现了TCPHeader类用于处理TCP报文头部，包含构造函数、打包解包方法及校验和计算等功能。设计了数据包发送模块能够构建并发送带有PSH和ACK标志的TCP数据包，同时实现了数据包接收模块解析返回的响应信息。通过界面交互模块，系统提供了身份验证与扣费、紧急通讯、TCP数据包详情显示以及系统关闭等功能选项，使用户能够直观操作。

在服务器端，实现了套接字初始化、客户端连接接受以及主循环处理客户端请求的核心功能。服务器能够根据不同指令执行对应操作，如处理车牌信息验证与扣费（支持余额检查）、进行紧急通讯交互、展示TCP数据包详细信息等。系统还设计了错误处理机制，能够应对无效选项输入和连接异常等情况。

通过本项目实践，我们深入理解了TCP数据包结构，包括源端口、目标端口、序列号、确认号、数据偏移、标志位、校验和等字段的作用与设置方法。掌握了校验和计算原理与实现技术，理解了TCP通信中的序列号管理、确认机制、错误检测和流量控制等核心概念。同时，通过实现ETC系统的功能场景，将理论知识应用于实际业务处理，提升了网络编程的实践能力。

未来可在本项目基础上继续扩展功能，如引入UDP协议进行对比研究、增强数据安全性设计、优化用户界面交互、增加并发连接处理能力、实现完整的三次握手和四次挥手过程模拟等，进一步深化对计算机网络协议原理的理解与应用。本项目为后续网络安全、协议分析与网络应用开发等方向提供了坚实的技术基础。

# 参考文献

1. 谢希仁. 计算机网络[M]. 第8版. 北京：电子工业出版社，2021. 1–428.
2. W. Richard Stevens, Bill Fenner, Andrew M. Rudoff. UNIX Network Programming Volume 1: The Sockets Networking API[M]. 3rd ed. Boston：Addison-Wesley, 2004. 1–973.
3. Kurose J F, Ross K W. Computer Networking: A Top-Down Approach[M]. 8th ed. Boston：Pearson Education, 2020. 1–864.
4. Forouzan B A. TCP/IP Protocol Suite[M]. 4th ed. New York：McGraw-Hill Education, 2009. 1–944.
5. 陈海林，胡丹. 网络程序设计实训教程[M]. 第2版. 北京：清华大学出版社，2019. 1–253.
6. 朱少民. Wireshark网络分析就这么简单[M]. 第2版. 北京：电子工业出版社，2020. 1–352.
7. 郭志伟，张宇. TCP/IP协议原理与应用[M]. 北京：人民邮电出版社，2020. 1–285.
8. 郑莉，杨晓捷. 计算机网络实验与实践教程[M]. 北京：清华大学出版社，2018. 63–98.
9. 尹湘萍，林湘照. 网络协议分析与安全实验教程[M]. 北京：电子工业出版社，2021. 91–117.
10. 王达. 计算机网络原理与应用[M]. 北京：清华大学出版社，2022. 88–108.
11. 胡春明. Windows网络编程实战[M]. 北京：机械工业出版社，2019. 46–70.
12. 张晨，李伟，等. 一种基于TCP协议的网络数据包构造与检测方法[J]. 通信技术，2020，53(8)：1432–1437.