目录

[一、实验要求 1](#_Toc156484484)

[二、协议原理 2](#_Toc156484485)

[2.1 TCP协议 2](#_Toc156484486)

[2.2 TCP三次握手 5](#_Toc156484487)

[2.3 TCP四次挥手 6](#_Toc156484488)

[三、程序功能与流程 8](#_Toc156484489)

[四、分析程序代码 8](#_Toc156484490)

[4.1 构建TCPServer 8](#_Toc156484491)

[4.2 构建TCPClient 13](#_Toc156484492)

[4.3 GUI可视化 14](#_Toc156484493)

[五、总结 16](#_Toc156484494)

[六、参考文献 17](#_Toc156484495)

实验报告

# 一、实验要求

1. 基本要求：理解三次握手、四次挥手过程及序号变化。

2. 设计语言：Python、C/C++。

3. 原理：利用 TCP 报文中的 SYN、SYN+ACK、ACK 报文与服务器某程序（例如端口 80、端口 23）建立 TCP 连接，然后向服务器发送部分数据，最后用四报文挥手释放连接。亦可参考计算机网络综合实验教程中的实验 11，编写一个简单的服务器程序，并与其建立连接、传输数据并释放连接。

4. 技术难点：TCP 握手和挥手过程中 seq 和 ack 的变化情况，准确构建对应的 TCP 报文段，发送这些报文段，接收并分析返回结果。另外，当运行程序的计算机收到服务器发来的 TCP 报文段之后（例如第二次握手的报文），计算机可会发送 RST 报文给服务器，这个 RST 报文必须丢弃，如何丢弃 RST 报文，需要在 linux 中实现（windows 丢弃 RST 包的方法较难），因此，该程序需在 Linux 中实现并运行。

5. [悦学课程参考链接](https://xue.51zhy.cn/detail/v_60fe69c0e4b0bf642ffe55ed/3?from=p_609dd95be4b0fe3220131618&type=6&parent_pro_id=)（三报文握手建立 TCP 连接）

# 二、协议原理

## 2.1 TCP协议

TCP（Transmission Control Protocol）是一种面向连接的、可靠的传输层协议，用于在计算机网络中传输数据。它是互联网协议套件（TCP/IP）中最常用的协议之一，被广泛用于应用层协议（如HTTP、FTP、SMTP等）的可靠数据传输。

下面是对TCP协议及其特点的详细解释：

1. 面向连接：在进行数据传输之前，发送方和接收方需要通过三次握手建立一个连接。连接的建立包括发送方向接收方发送一个连接请求（SYN），接收方向发送方发送一个连接确认（SYN-ACK），以及发送方向接收方发送一个连接确认（ACK）。连接的建立确保了双方都愿意进行数据传输，并为后续的数据传输提供了必要的准备。

2. 可靠性：TCP通过使用序列号、确认应答和重传等机制来确保可靠的数据传输。每个TCP报文段都包含一个序列号，接收方通过确认应答来告知发送方已经接收到哪些数据。如果发送方没有接收到确认应答或者接收到了超时的确认应答，它将会重新发送未确认的数据。这种机制确保了数据的可靠性，即使在网络丢包或出现错误的情况下也能够进行恢复。

3. 流量控制：TCP使用滑动窗口机制来进行流量控制。接收方通过告知发送方自己的缓冲区大小来限制发送方发送的数据量。发送方根据接收方提供的窗口大小来控制发送的数据量，以确保接收方能够及时处理接收到的数据。

4. 拥塞控制：TCP使用拥塞控制算法来避免网络拥塞。发送方通过动态调整发送的数据量来适应网络的状况。当网络拥塞时，发送方减少发送的数据量，以降低网络负载。当网络状况改善时，发送方逐渐增加发送的数据量，以提高数据传输的效率。

5. 有序性：TCP保证数据的有序传输。接收方根据接收到的TCP报文段的序列号对它们进行重新排序，以确保数据按照发送方发送时的顺序进行传输。

6. 双工通信：TCP支持全双工通信，即发送方和接收方可以同时进行数据的发送和接收。这使得双方能够同时进行数据交换，提高了通信的效率。

总结起来，TCP是一种可靠的面向连接的传输协议，通过建立连接、使用序列号和确认应答、流量控制和拥塞控制等机制，提供了可靠的数据传输、有序性、流量控制和拥塞控制等功能。它是互联网上应用层协议进行可靠数据传输的基础。

上述讲到了TCP协议及其特点，此外，TCP协议在互联网中有广泛的应用场景，以下是一些常见的应用场景：

1. 网页浏览：TCP作为HTTP协议的可靠传输层协议，用于在浏览器和服务器之间传输网页内容。当您在浏览器中输入网址并请求网页时，TCP协议负责将网页内容分成多个TCP报文段，并通过互联网将它们从服务器传输到您的浏览器。

2. 文件传输：TCP协议也被用于文件传输协议，如FTP（File Transfer Protocol）。通过TCP的可靠性和流量控制，FTP可以确保文件在客户端和服务器之间的可靠传输。

3. 电子邮件：TCP协议用于电子邮件的传输。当您发送电子邮件时，您的电子邮件客户端使用TCP将邮件传输到邮件服务器，而接收方的电子邮件客户端使用TCP从邮件服务器接收邮件。

4. 远程登录：TCP协议被用于远程登录协议，如Telnet和SSH。这些协议允许用户通过网络远程登录到远程计算机，并与其进行交互。TCP协议提供了可靠的数据传输和双向通信的支持，确保用户与远程计算机之间的交互是稳定和可靠的。

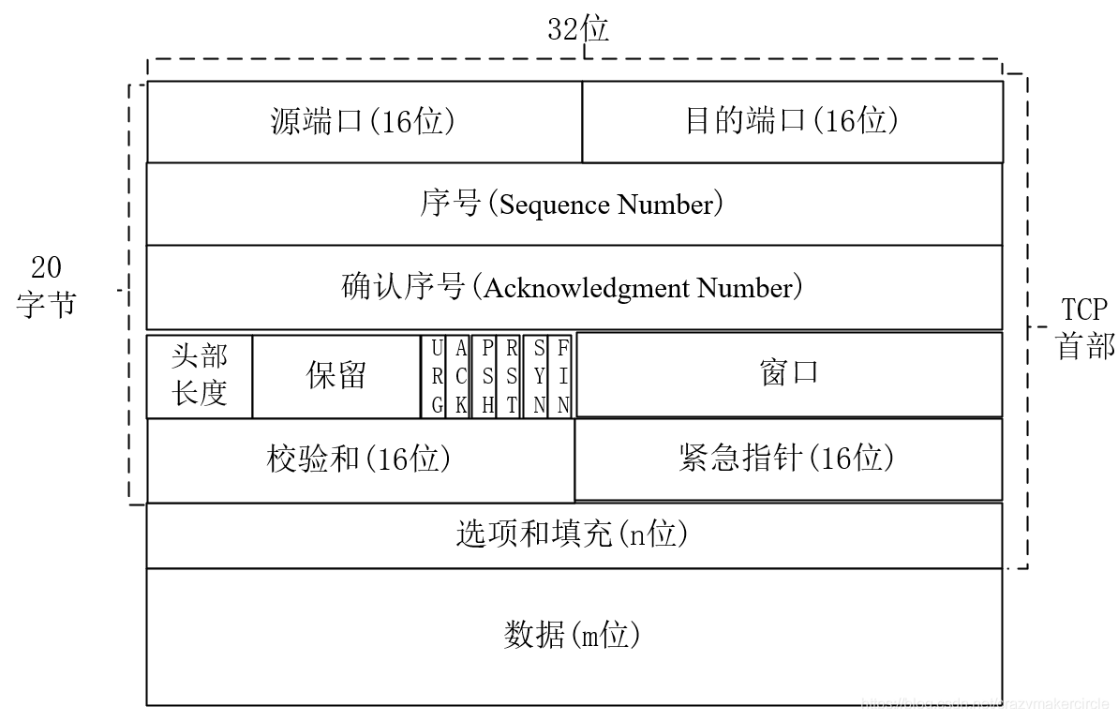
5. 数据库访问：TCP协议常被用于数据库访问，如MySQL和PostgreSQL。通过TCP协议，客户端可以与数据库服务器建立连接，并进行查询、更新和管理数据库中的数据。

6. 实时通信：TCP协议也被用于实时通信应用，如即时通讯（Instant Messaging）和语音通话。通过TCP的可靠性和流量控制，这些应用可以确保消息和音频数据在用户之间的实时传输。

需要注意的是，尽管TCP协议在许多应用中被广泛使用，但对于某些特定的应用，如实时视频流或大规模数据传输等，可能会选择使用UDP协议，因为UDP具有更低的延迟和更高的传输速度，但可靠性较差。因此，在选择使用TCP还是UDP时，需要根据应用的需求来做出权衡。

上述了解了TCP协议、特点还有它适合的场景，为了更加详细的了解他，需要掌握它的报文格式：

TCP（Transmission Control Protocol）报文格式如下所示：



下面是对各个字段的详细解释：

1. 源端口号（Source Port）：16位字段，用于标识发送方的应用程序或服务的端口号。

2. 目标端口号（Destination Port）：16位字段，用于标识接收方的应用程序或服务的端口号。

3. 序列号（Sequence Number）：32位字段，用于对TCP数据流中的每个字节进行编号。序列号用来保证数据的顺序性。

4. 确认号（Acknowledgment Number）：32位字段，用于确认已经收到的数据的序列号。确认号表示期望接收的下一个字节的序列号。

5. 数据偏移（Data Offset）：4位字段，表示TCP报文头部的长度，以4字节为单位。TCP报文头长度最小为20字节。

6. 保留（Reserved）：6位字段，保留用于将来的扩展。

7. URG标志（URG）：1位字段，表示紧急指针字段是否有效。

8. ACK标志（ACK）：1位字段，表示确认号字段是否有效。

9. PSH标志（PSH）：1位字段，表示接收方是否应该尽快将数据交给应用层。

10. RST标志（RST）：1位字段，表示中断连接。

11. SYN标志（SYN）：1位字段，用于建立连接时进行同步。

12. FIN标志（FIN）：1位字段，表示发送方已经完成数据的发送，准备关闭连接。

13. 窗口大小（Window Size）：16位字段，表示接收方可接收的字节数。

14. 校验和（Checksum）：16位字段，用于检验TCP报文的完整性。

15. 紧急指针（Urgent Pointer）：16位字段，用于指示紧急数据的字节偏移量。

16. 选项（Options）：可选字段，用于在TCP报文中添加一些可选的功能或参数。

17. 数据（Data）：可选字段，用于携带应用层的数据。

TCP报文格式中的各个字段共同构成了TCP协议数据传输的头部部分，提供了必要的控制和管理信息，以实现可靠的、有序的数据传输。

## 2.2 TCP三次握手

建立一个连接需要三次握手，而终止一个连接要经过四次挥手（也有将四次挥手叫做四次握手的）。这由TCP的**半关闭**（half-close）造成的。所谓的半关闭，其实就是TCP提供了连接的一端在结束它的发送后还能接收来自另一端数据的能力。

TCP 连接的拆除需要发送四个包，因此称为四次挥手(Four-way handshake)，客户端或服务端均可主动发起挥手动作。

刚开始双方都处于ESTABLISHED 状态，假如是客户端先发起关闭请求。四次挥手的过程如下：

* 第一次挥手：客户端发送一个 FIN 报文，报文中会指定一个序列号。此时客户端处于 FIN\_WAIT1 状态。  
  即发出**连接释放报文段**（FIN=1，序号seq=u），并停止再发送数据，主动关闭TCP连接，进入FIN\_WAIT1（终止等待1）状态，等待服务端的确认。
* 第二次挥手：服务端收到 FIN 之后，会发送 ACK 报文，且把客户端的序列号值 +1 作为 ACK 报文的序列号值，表明已经收到客户端的报文了，此时服务端处于 CLOSE\_WAIT 状态。  
  即服务端收到连接释放报文段后即发出**确认报文段**（ACK=1，确认号ack=u+1，序号seq=v），服务端进入CLOSE\_WAIT（关闭等待）状态，此时的TCP处于半关闭状态，客户端到服务端的连接释放。客户端收到服务端的确认后，进入FIN\_WAIT2（终止等待2）状态，等待服务端发出的连接释放报文段。
* 第三次挥手：如果服务端也想断开连接了，和客户端的第一次挥手一样，发给 FIN 报文，且指定一个序列号。此时服务端处于 LAST\_ACK 的状态。  
  即服务端没有要向客户端发出的数据，服务端发出**连接释放报文段**（FIN=1，ACK=1，序号seq=w，确认号ack=u+1），服务端进入LAST\_ACK（最后确认）状态，等待客户端的确认。
* 第四次挥手：客户端收到 FIN 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 +1 作为自己 ACK 报文的序列号值，此时客户端处于 TIME\_WAIT 状态。需要过一阵子以确保服务端收到自己的 ACK 报文之后才会进入 CLOSED 状态，服务端收到 ACK 报文之后，就处于关闭连接了，处于 CLOSED 状态。  
  即客户端收到服务端的连接释放报文段后，对此发出**确认报文段**（ACK=1，seq=u+1，ack=w+1），客户端进入TIME\_WAIT（时间等待）状态。此时TCP未释放掉，需要经过时间等待计时器设置的时间2MSL后，客户端才进入CLOSED状态。

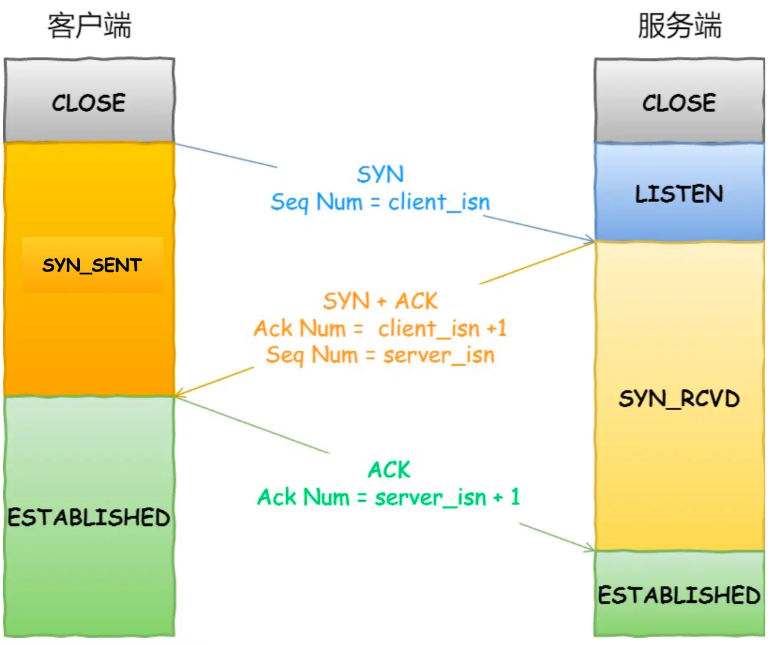
收到一个FIN只意味着在这一方向上没有数据流动。客户端执行主动关闭并进入TIME\_WAIT是正常的，服务端通常执行被动关闭，不会进入TIME\_WAIT状态。

在socket编程中，任何一方执行close()操作即可产生挥手操作。

一旦服务器收到客户端的ACK报文，TCP连接就正式建立起来了。此时，客户端和服务器之间可以开始进行数据传输。

三次握手的目的是确保客户端和服务器都能够发送和接收数据，并且双方已经同意了初始的序列号。通过这个过程，TCP可以建立一个可靠的双向通信通道，以确保数据传输的可靠性和完整性。

需要注意的是，三次握手过程中可能会出现延迟或丢失的情况。如果某个步骤的报文丢失或延迟到达，TCP会根据超时和重传机制进行处理，以确保握手过程的完成。



## 2.3 TCP四次挥手

挥手为什么需要四次？因为当服务端收到客户端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当服务端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉客户端，“你发的FIN报文我收到了”。只有等到我服务端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四次挥手。

TCP（Transmission Control Protocol）使用四次挥手（four-way handshake）来关闭一个已建立的连接。四次挥手确保了通信双方在终止连接时的正常关闭，以避免数据丢失或不完整。

下面是TCP四次挥手的详细过程：

1. 第一步（FIN）：当客户端决定关闭连接时，它发送一个FIN（Finish）报文给服务器。这个报文表示客户端不再发送数据，但仍然可以接收服务器发送的数据。

2. 第二步（ACK）：服务器收到客户端的FIN报文后，发送一个ACK（Acknowledgment）报文作为确认。这个报文表示服务器已经收到了客户端的关闭请求。

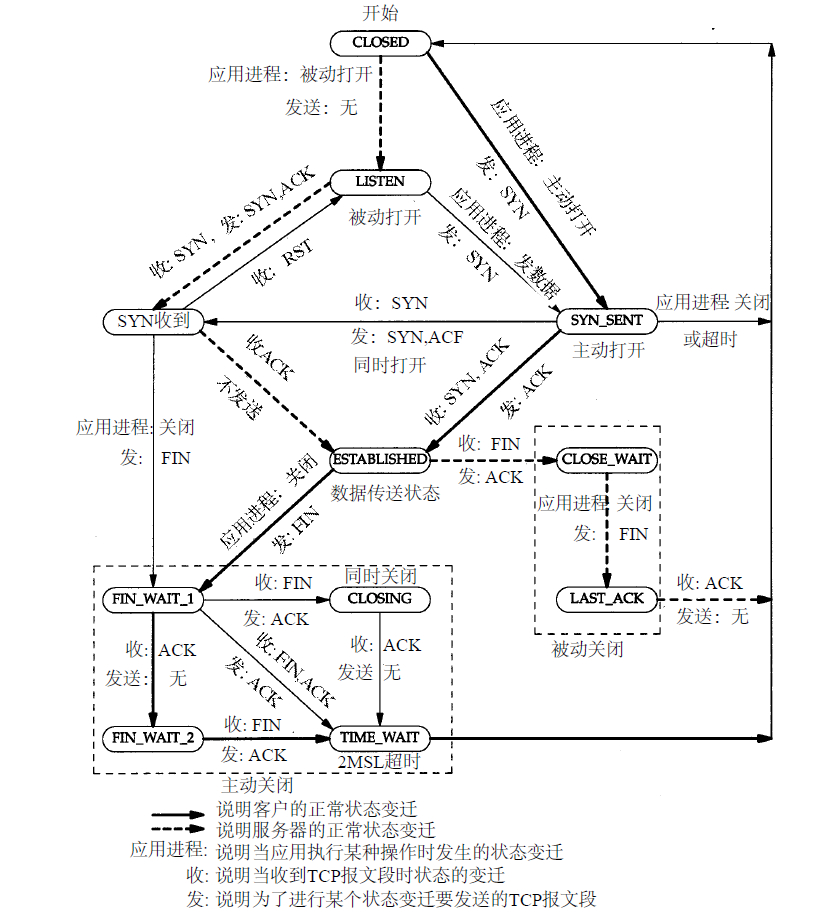
3. 第三步（FIN）：当服务器也准备关闭连接时，它发送一个FIN报文给客户端。这个报文表示服务器不再发送数据。

4. 第四步（ACK）：客户端收到服务器的FIN报文后，发送一个ACK报文作为确认。这个报文表示客户端已经收到了服务器的关闭请求。

一旦双方都发送了FIN和ACK报文，并收到了对方的确认，TCP连接就会正式关闭。此时，双方都不再发送或接收数据。

四次挥手的目的是确保双方都能够完成数据的传输和接收，并在关闭连接时进行协调。通过这个过程，TCP可以正常地关闭连接，释放相关的资源，并确保数据的完整性。

需要注意的是，在四次挥手过程中可能会出现延迟、丢失或重复的报文。TCP通过超时和重传机制来处理这些情况，以确保挥手过程的顺利完成。



TCP状态变迁图

# 三、程序功能与流程

我们首先创建了一个客户端套接字，然后使用 connect() 方法与服务器建立 TCP 连接，并发送 SYN 报文。

接着，我们接收服务器返回的 SYN+ACK 报文，并发送 ACK 报文，完成三次握手，建立了连接。

然后，我们向服务器发送数据，并使用 recv() 方法接收服务器的响应数据。

接着，我们发送四报文挥手释放连接：发送 FIN 报文，接收 FIN+ACK 报文，发送 ACK 报文，完成四次挥手，释放连接。

最后，我们使用tkinter中进行图形化界面GUI，并展示出来。

# 四、分析程序代码

## 4.1 构建TCPServer



上述代码是一个简单的 TCP 服务器类 `TCPServer` 的示例。

解释代码如下：

1. `def \_\_init\_\_(self, port):`: 定义了 `TCPServer` 类的构造函数。在创建 `TCPServer` 类的实例时，需要传入服务器的端口号 `port`。

2. `self.port = port`: 将传入的端口号保存到类的属性 `port` 中。

3. `self.server\_socket = None`: 定义了一个名为 `server\_socket` 的属性，初始值为 `None`。这个属性用于存储服务器的套接字对象。

4. `self.server\_thread = None`: 定义了一个名为 `server\_thread` 的属性，初始值为 `None`。这个属性用于存储用于监听客户端连接的线程对象。

5. `def start(self):`: 定义了一个 `start` 方法，用于启动服务器。

6. 在 `start` 方法中：

- `self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)`: 创建一个服务器套接字对象 `server\_socket`，用于监听客户端连接。

- `self.server\_socket.bind(('192.168.1.5', self.port))`: 将服务器套接字绑定到指定的 IP 地址和端口号。在这个示例中，IP 地址为 `'192.168.1.5'`，端口号为构造函数中传入的 `port`。

- `self.server\_socket.listen(1)`: 开始监听客户端连接请求，参数 `1` 表示最多允许同时连接的客户端数量为 1。

- `print("服务器已启动，等待客户端连接...")`: 打印消息表示服务器已启动，等待客户端连接。

7. `self.server\_thread = threading.Thread(target=self.\_listen)`: 创建一个线程对象 `server\_thread`，该线程将调用 `\_listen` 方法来监听客户端连接。`\_listen` 方法是一个内部方法，用于实际处理客户端连接。

8. `self.server\_thread.start()`: 启动线程，开始监听客户端连接。

9. `def stop(self):`: 定义了一个 `stop` 方法，用于停止服务器。

10. 在 `stop` 方法中：

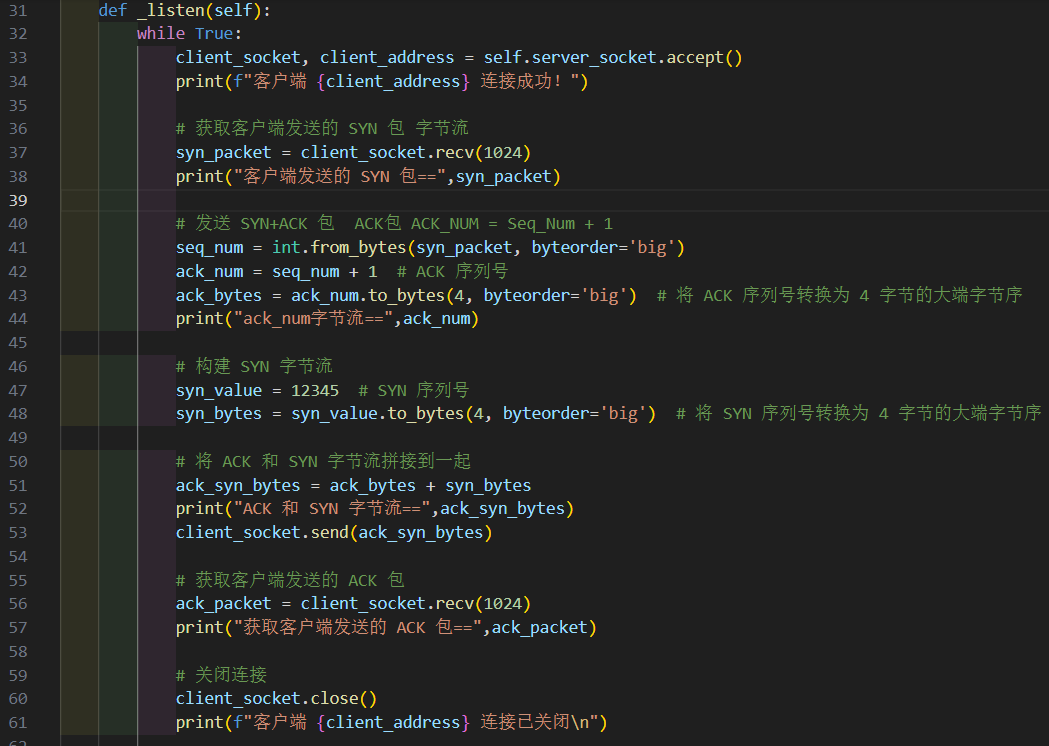
- `if self.server\_socket:`: 检查服务器套接字对象是否存在。

- `self.server\_socket.close()`: 关闭服务器套接字。

- `if self.server\_thread:`: 检查监听线程对象是否存在。

- `self.server\_thread.join()`: 等待监听线程结束。

总体而言，这段代码实现了一个简单的 TCP 服务器类 `TCPServer`，通过调用 `start` 方法可以启动服务器并开始监听客户端连接，通过调用 `stop` 方法可以停止服务器并关闭套接字。



上述代码定义了一个名为 `\_listen` 的方法，用于在服务器上监听客户端连接，并处理 TCP 三次握手的过程。

解释代码如下：

1. `def \_listen(self):`: 定义了一个名为 `\_listen` 的方法，用于监听客户端连接并处理三次握手过程。

2. `while True:`: 进入一个无限循环，持续监听客户端连接。

3. `client\_socket, client\_address = self.server\_socket.accept()`: 通过调用服务器套接字的 `accept()` 方法，接受客户端的连接请求，返回一个新的套接字对象 `client\_socket` 和客户端的地址信息 `client\_address`。

4. `print(f"客户端 {client\_address} 连接成功！")`: 打印消息表示客户端连接成功。

5. `syn\_packet = client\_socket.recv(1024)`: 从客户端套接字接收客户端发送的 SYN 包，最多接收 1024 字节的数据。

6. `print("客户端发送的 SYN 包==", syn\_packet)`: 打印客户端发送的 SYN 包的字节流。

7. `seq\_num = int.from\_bytes(syn\_packet, byteorder='big')`: 将接收到的 SYN 包字节流转换为整数，获取序列号 `seq\_num`。

8. `ack\_num = seq\_num + 1`: 计算 SYN+ACK 包的确认号，即客户端发送的 SYN 包序列号加 1。

9. `ack\_bytes = ack\_num.to\_bytes(4, byteorder='big')`: 将确认号 `ack\_num` 转换为 4 字节的大端字节序。

10. `syn\_value = 12345`: 设置 SYN 包的序列号 `syn\_value`。

11. `syn\_bytes = syn\_value.to\_bytes(4, byteorder='big')`: 将 SYN 包的序列号 `syn\_value` 转换为 4 字节的大端字节序。

12. `ack\_syn\_bytes = ack\_bytes + syn\_bytes`: 将 ACK 字节流和 SYN 字节流拼接在一起，形成 SYN+ACK 包的字节流。

13. `client\_socket.send(ack\_syn\_bytes)`: 将 SYN+ACK 包的字节流发送给客户端。

14. `ack\_packet = client\_socket.recv(1024)`: 从客户端套接字接收客户端发送的 ACK 包，最多接收 1024 字节的数据。

15. `print("获取客户端发送的 ACK 包==", ack\_packet)`: 打印客户端发送的 ACK 包的字节流。

16. `client\_socket.close()`: 关闭客户端套接字。

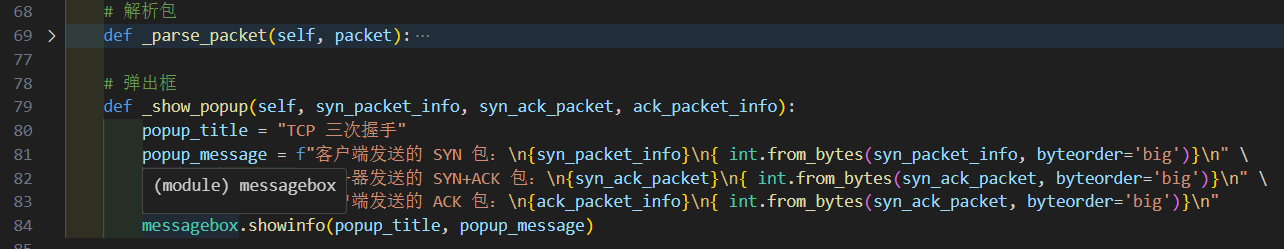
17. `print(f"客户端 {client\_address} 连接已关闭\n")`: 打印消息表示客户端连接已关闭。

18. `self.\_show\_popup(syn\_packet, ack\_syn\_bytes, ack\_packet)`: 调用 `\_show\_popup` 方法，显示弹出框，展示 TCP 三次握手过程中的包信息。

19. `client\_socket.close()`: 再次关闭客户端套接字。

20. `print(f"客户端 {client\_address} 连接已关闭\n")`: 打印消息表示客户端连接已关闭。

总体而言，这段代码在一个循环中监听客户端连接，接收客户端发送的 SYN 包，发送 SYN+ACK 包，接收客户端发送的 ACK 包，并显示弹出框展示三次握手过程中的包信息。连接关闭后，循环继续监听下一个客户端连接。



上述代码定义了一个名为 `\_show\_popup` 的方法，用于显示一个弹出框，展示 TCP 三次握手过程中发送的 SYN 包、SYN+ACK 包和 ACK 包的信息。

解释代码如下：

1. `def \_show\_popup(self, syn\_packet\_info, syn\_ack\_packet, ack\_packet\_info):`: 定义了一个名为 `\_show\_popup` 的方法，它有三个参数：`syn\_packet\_info`、`syn\_ack\_packet` 和 `ack\_packet\_info`，分别表示 SYN 包、SYN+ACK 包和 ACK 包的信息。

2. `popup\_title = "TCP 三次握手"`: 定义了弹出框的标题为 "TCP 三次握手"。

3. `popup\_message = f"客户端发送的 SYN 包：\n{syn\_packet\_info}\n{int.from\_bytes(syn\_packet\_info, byteorder='big')}\n" \`: 构建了弹出框的消息内容。使用了 f-string 格式化字符串，包括客户端发送的 SYN 包信息和对应的整数值。

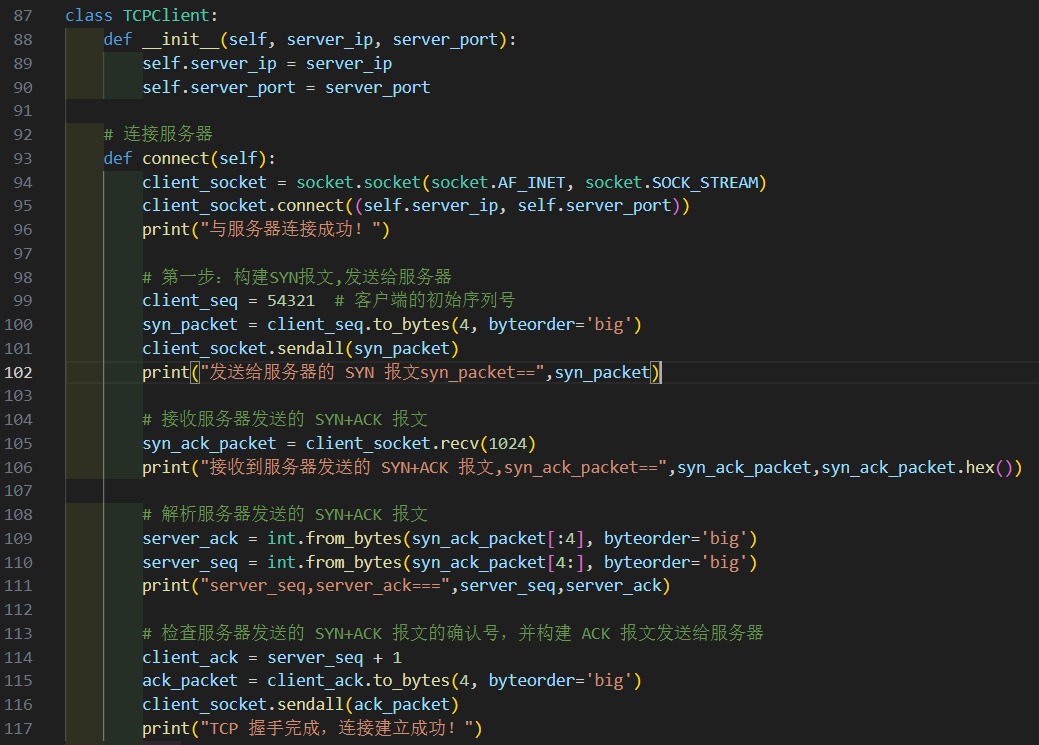
4. `f"服务器发送的 SYN+ACK 包：\n{syn\_ack\_packet}\n{int.from\_bytes(syn\_ack\_packet, byteorder='big')}\n"`: 在消息内容中添加了服务器发送的 SYN+ACK 包的信息和整数值。

5. `f"客户端发送的 ACK 包：\n{ack\_packet\_info}\n{int.from\_bytes(syn\_ack\_packet, byteorder='big')}\n"`: 在消息内容中添加了客户端发送的 ACK 包的信息和整数值。

6. `messagebox.showinfo(popup\_title, popup\_message)`: 调用 `messagebox.showinfo()` 方法显示一个信息弹出框。`popup\_title` 是弹出框的标题，`popup\_message` 是弹出框的消息内容。

总体而言，这段代码定义了一个方法 `\_show\_popup`，用于在图形化界面中显示一个弹出框，展示 TCP 三次握手过程中发送的 SYN 包、SYN+ACK 包和 ACK 包的信息。弹出框显示的内容包括包的原始信息以及将其解析为整数值后的结果。

## 4.2 构建TCPClient



上述代码是一个连接服务器的函数 `connect(self)` 的示例。

解释代码如下：

1. `client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)`: 创建一个客户端套接字对象 `client\_socket`，用于与服务器建立 TCP 连接。

2. `client\_socket.connect((self.server\_ip, self.server\_port))`: 使用 `connect()` 方法连接到指定的服务器 IP 地址和端口号。`self.server\_ip` 和 `self.server\_port` 是类属性，表示服务器的 IP 地址和端口号。

3. `print("与服务器连接成功！")`: 打印消息表示与服务器成功建立连接。

4. 构建 SYN 报文并发送给服务器：

- `client\_seq = 54321`: 客户端的初始序列号。

- `syn\_packet = client\_seq.to\_bytes(4, byteorder='big')`: 将客户端的序列号转换为 4 字节的大端字节序，构建 SYN 报文。

- `client\_socket.sendall(syn\_packet)`: 将 SYN 报文发送给服务器。

5. `print("发送给服务器的 SYN 报文syn\_packet==", syn\_packet)`: 打印发送给服务器的 SYN 报文的内容。

6. 接收服务器发送的 SYN+ACK 报文：

- `syn\_ack\_packet = client\_socket.recv(1024)`: 使用 `recv()` 方法接收服务器发送的 SYN+ACK 报文。`1024` 是指定的接收缓冲区大小。

7. `print("接收到服务器发送的 SYN+ACK 报文, syn\_ack\_packet==", syn\_ack\_packet, syn\_ack\_packet.hex())`: 打印接收到的 SYN+ACK 报文的内容。

8. 解析服务器发送的 SYN+ACK 报文：

- `server\_ack = int.from\_bytes(syn\_ack\_packet[:4], byteorder='big')`: 提取接收到的 SYN+ACK 报文中的确认号。

- `server\_seq = int.from\_bytes(syn\_ack\_packet[4:], byteorder='big')`: 提取接收到的 SYN+ACK 报文中的序列号。

9. `print("server\_seq, server\_ack===", server\_seq, server\_ack)`: 打印服务器的序列号和确认号。

10. 检查服务器发送的 SYN+ACK 报文的确认号，并构建 ACK 报文发送给服务器：

- `client\_ack = server\_seq + 1`: 根据服务器的序列号计算客户端的确认号。

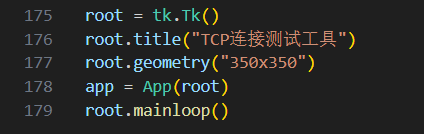
- `ack\_packet = client\_ack.to\_bytes(4, byteorder='big')`: 将客户端的确认号转换为 4 字节的大端字节序，构建 ACK 报文。

- `client\_socket.sendall(ack\_packet)`: 将 ACK 报文发送给服务器。

11. `print("TCP 握手完成，连接建立成功！")`: 打印消息表示 TCP 握手过程完成，连接成功建立。

总体而言，这段代码通过客户端套接字与服务器建立 TCP 连接，进行了三次握手的过程，并打印了握手过程中发送和接收的报文内容。最后，根据服务器发送的 SYN+ACK 报文，构建并发送了客户端的 ACK 报文，完成了连接的建立。

## 4.3 GUI可视化



上述代码用于创建一个基于 tkinter 库的 GUI 应用程序。

解释代码如下：

1. `root = tk.Tk()`: 创建一个名为 `root` 的根窗口对象，并将其赋值给变量 `root`。这个根窗口是 GUI 应用程序的最顶层窗口。

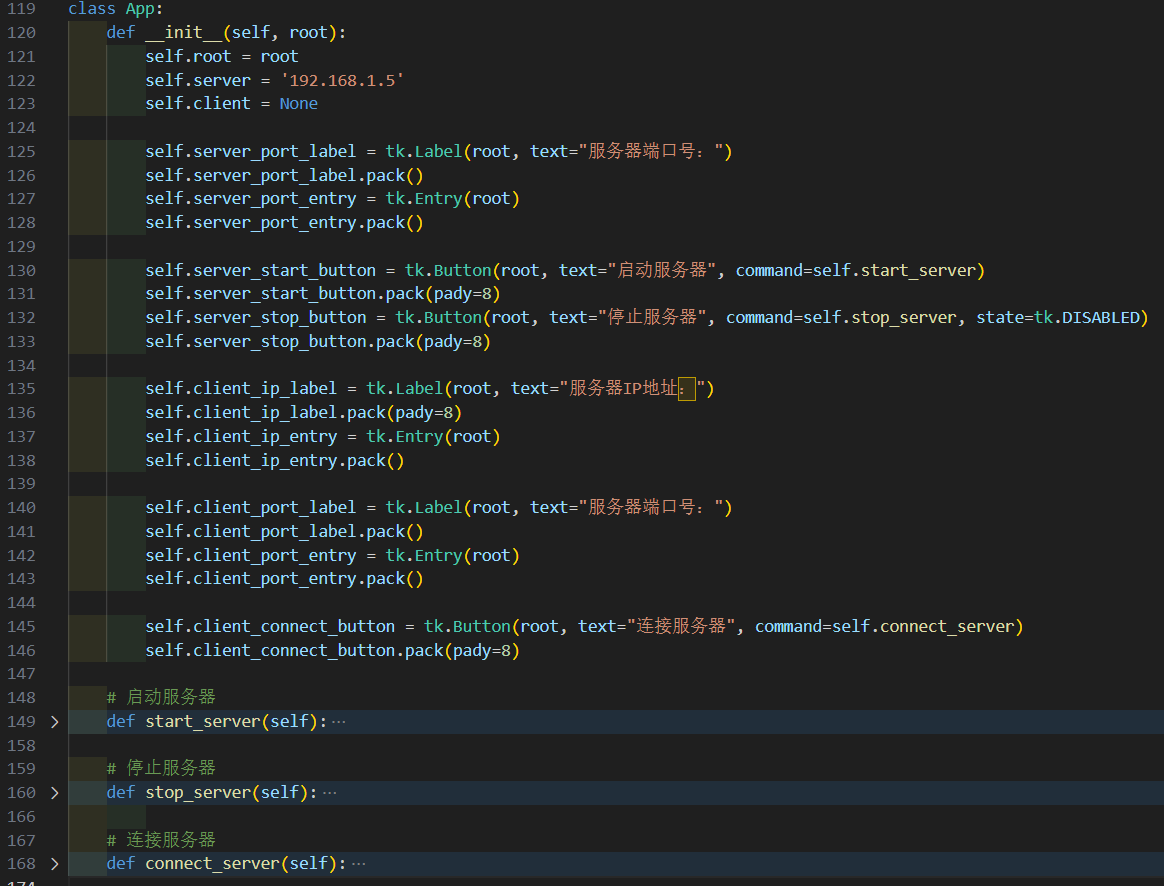
2. `root.title("TCP连接测试工具")`: 设置根窗口的标题为 "TCP连接测试工具"。

3. `root.geometry("350x350")`: 设置根窗口的大小为宽度 350 像素、高度 350 像素。

4. `app = App(root)`: 创建一个 `App` 类的实例，并传入根窗口对象 `root`。这样就将 `App` 类的 GUI 元素添加到了根窗口中。

5. `root.mainloop()`: 进入主事件循环，该函数会一直运行，直到用户关闭窗口。在事件循环中，程序会等待用户的交互事件（如按钮点击、键盘输入等），并调用相应的事件处理函数来响应用户操作。

总体而言，这段代码创建了一个 GUI 应用程序的根窗口，并在根窗口中添加了 `App` 类的 GUI 元素。然后，通过调用 `root.mainloop()` 进入主事件循环，使程序保持运行状态，等待用户的交互操作。



上述代码是一个使用 tkinter 库创建图形用户界面（GUI）的 Python 类 `App` 的示例。

解释代码如下：

1. `class App:`: 定义了一个名为 `App` 的类，用于创建 GUI 应用程序。

2. `def \_\_init\_\_(self, root):`: `\_\_init\_\_` 方法是类的构造函数，在创建 `App` 类的实例时被调用。`self` 表示类的实例本身，`root` 是传入的根窗口对象。

3. `self.root = root`: 将传入的根窗口对象保存到类的属性 `root` 中。

4. `self.server = '192.168.1.5'`: 定义了一个名为 `server` 的字符串属性，表示服务器的 IP 地址。

5. `self.client = None`: 定义了一个名为 `client` 的属性，初始值为 `None`。

6. 接下来是一系列 GUI 元素的创建和配置，包括标签 (`tk.Label`)、文本输入框 (`tk.Entry`) 和按钮 (`tk.Button`)。

- `self.server\_port\_label`, `self.server\_port\_entry`: 用于输入服务器的端口号。

- `self.server\_start\_button`, `self.server\_stop\_button`: 分别用于启动和停止服务器。

- `self.client\_ip\_label`, `self.client\_ip\_entry`: 用于输入服务器的 IP 地址。

- `self.client\_port\_label`, `self.client\_port\_entry`: 用于输入服务器的端口号。

- `self.client\_connect\_button`: 用于连接服务器。

7. `command=self.start\_server`, `command=self.stop\_server`, `command=self.connect\_server`: 这些是按钮的 `command` 参数，表示按钮被点击时要执行的函数。

8. `self.start\_server`, `self.stop\_server`, `self.connect\_server`: 这些是 `App` 类中定义的方法，用于处理按钮点击事件的逻辑。

总体而言，这段代码创建了一个基本的 GUI 应用程序，包含了服务器的启动和停止按钮以及客户端的连接按钮。它为用户提供了输入服务器 IP 地址和端口号的文本框，并通过调用相应的方法处理按钮点击事件。

# 五、总结

上述实验涉及了使用 TCP 协议建立连接、传输数据和释放连接的过程。简要流程如下：

1. 创建客户端套接字，并使用 `connect()` 方法与服务器建立 TCP 连接，发送 SYN 报文。

2. 接收服务器返回的 SYN+ACK 报文，并发送 ACK 报文，完成三次握手，建立连接。

3. 向服务器发送数据，并使用 `recv()` 方法接收服务器的响应数据。

4. 发送四报文挥手释放连接：发送 FIN 报文，接收 FIN+ACK 报文，发送 ACK 报文，完成四次挥手，释放连接。

5. 关闭客户端套接字。

# 六、参考文献

1. "TCP Congestion Control" by Van Jacobson (1988): 这篇论文介绍了 TCP 拥塞控制算法的基本原理，其中包括慢启动、拥塞避免和快速重传/恢复等机制。

2. "TCP/IP Illustrated, Volume 1: The Protocols" by W. Richard Stevens (1994): 这本书并非论文，但它详细介绍了 TCP/IP 协议栈的工作原理和实现细节，对理解 TCP 协议非常有帮助。

3. "TCP/IP Performance Analysis for Space Communications" by Sally Floyd and Van Jacobson (1995): 这篇论文深入分析了 TCP 的性能问题，特别是在高延迟和高丢包率的网络环境下的表现，并提出了一些改进的建议。

4. "TCP Vegas: New Techniques for Congestion Detection and Avoidance" by Lawrence S. Brakmo and Larry L. Peterson (1995): 该论文介绍了 TCP Vegas 拥塞控制算法，它通过测量网络往返时间的变化来检测拥塞，并采取相应的措施避免拥塞。

5. "TCP/IP Performance: A Hierarchy of Models" by Jitendra Padhye, Victor Firoiu, and Don Towsley (2000): 这篇论文提出了一个层次化的模型，用于理解和分析 TCP 的性能问题，包括带宽利用率、时延、丢包和吞吐量等方面。