目录

[一、实验要求 2](#_Toc156939541)

[二、协议原理 2](#_Toc156939542)

[2.1 DHCP协议 2](#_Toc156939543)

[2.2 DHCP协议的工作流程 3](#_Toc156939544)

[2.3 DHCP协议特点 4](#_Toc156939545)

[2.4 DHCP服务器IP分配 5](#_Toc156939546)

[2.5 DHCP协议的应用： 5](#_Toc156939547)

[2.6 DHCP服务器管理IP地址租约 6](#_Toc156939548)

[2.7 DHCP协议的场景 7](#_Toc156939549)

[三、分析程序代码 7](#_Toc156939550)

[4.1 导入一些必要的库 7](#_Toc156939551)

[4.2 构建GUI界面启动主程序 8](#_Toc156939552)

[4.3服务端功能 11](#_Toc156939553)

[4.4 客户端功能 14](#_Toc156939554)

[五、总结 17](#_Toc156939555)

[六、参考文献 20](#_Toc156939556)

实验报告

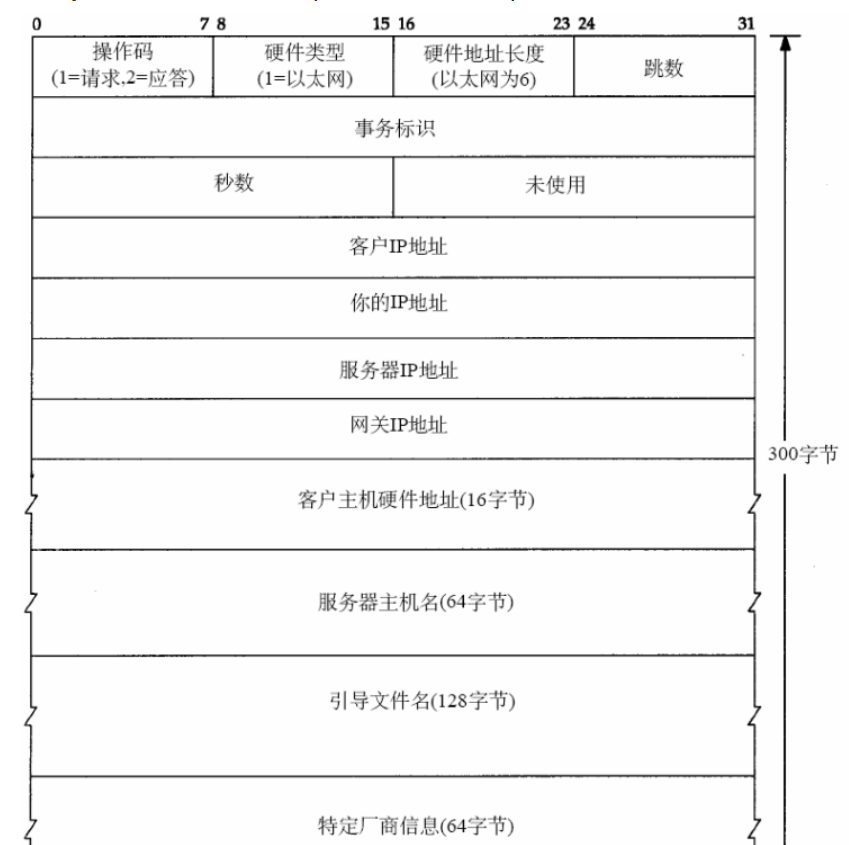
# 一、实验要求

1. 基本要求：在理解 DHCP 协议的基础上，编写一个 DHCP 服务器，为网络中的主机动态分配 IP 地址等信息。2. 设计语言：Python、C/C++。3. 原理：根据 DHCP 工作过程，即 DHCP 正常工作的所需的几种 DHCP报文，在收到的客户的 DHCP 报文之后，服务器正确构造相应的 DHCP 响应报文并发送给 DHCP 客户。4. 技术难点：分析收到 DHCP 客户发送的报文并正确发送响应 DHCP 报文。最终效果：计算机能从运行的 DHCP 服务器程序获取 IP 地址等信息，并能通过 whireshark 抓到相应的交互报文。

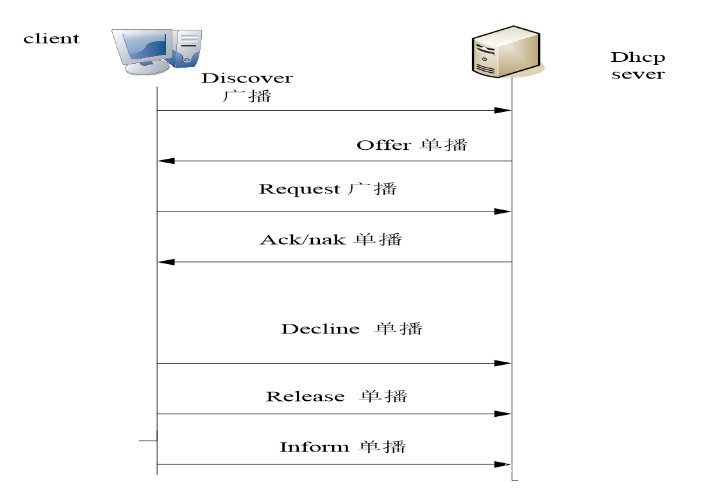
# 二、协议原理

## 2.1 DHCP协议

DHCP（Dynamic Host Configuration Protocol）是一种网络协议，用于自动分配IP地址和其他网络参数给连接到网络的设备。它采用客户端/服务器模型，在网络中的DHCP服务器负责分配IP地址和配置网络参数，而DHCP客户端向服务器请求获取这些信息。



## 2.2 DHCP协议的工作流程



DHCP协议的工作过程如下：

DHCP发现：当设备连接到网络时，它会发送一个DHCP发现消息（DHCP Discover），使用广播方式向网络中的所有设备发送请求。这个消息表明设备正在寻找一个DHCP服务器来获取IP地址和其他配置参数。

DHCP提供：当DHCP服务器收到DHCP发现消息后，它会回复一个DHCP提供消息（DHCP Offer）。这个消息包含一个可用的IP地址以及其他配置参数，如子网掩码、默认网关和DNS服务器等。

DHCP请求：设备收到DHCP提供消息后，可以选择接受其中的IP地址和配置参数，然后发送一个DHCP请求消息（DHCP Request）给指定的DHCP服务器。该消息确认设备接受了服务器提供的IP地址和配置。

DHCP确认：DHCP服务器接收到DHCP请求消息后，会发送一个DHCP确认消息（DHCP Acknowledgment）给设备。这个消息确认设备已成功获取IP地址和配置参数，并指定租约时长，即设备可以使用该IP地址和配置的时间。

DHCP续约：在租约过期之前，设备可以发送DHCP续约消息（DHCP Renewal）向DHCP服务器请求延长租约时间。

DHCP释放：当设备不再需要使用DHCP分配的IP地址时，它可以发送一个DHCP释放消息（DHCP Release），告知DHCP服务器可以将该IP地址回收并重新分配。

通过DHCP协议，网络中的设备可以自动获取有效的IP地址和其他必要的网络配置，简化了网络管理和配置的过程，提高了网络的灵活性和可扩展性。

## 2.3 DHCP协议特点

自动IP地址分配：DHCP协议可以自动为连接到网络的设备分配IP地址，无需手动配置每个设备的IP地址。这简化了网络管理，减少了IP地址冲突的可能性，并提高了网络的可扩展性。 动态配置：DHCP协议支持动态配置，即分配给设备的IP地址和其他网络参数（如子网掩码、默认网关、DNS服务器等）可以根据需求和网络情况进行调整。这使得网络管理员可以更灵活地管理和调整网络资源。集中管理：通过DHCP服务器，网络管理员可以集中管理IP地址池和配置参数。这样，管理员可以更好地控制和监测网络中的设备，并进行集中的IP地址管理和分配。 租约机制：DHCP协议使用租约机制来管理IP地址的分配。每个设备获得的IP地址都有一个租约时间，设备在租约到期前可以续约。这允许设备更灵活地使用IP地址，并确保网络资源的合理分配和回收。减少配置错误：DHCP协议减少了手动配置网络参数的机会，从而减少了配置错误的可能性。网络管理员只需在DHCP服务器上进行正确的配置，而不需要逐个配置每个设备。 支持其他网络参数：除了IP地址外，DHCP协议还可以分配其他网络参数，如子网掩码、默认网关、DNS服务器、NTP服务器等。这使得设备能够获取完整的网络配置，提供更好的网络连接和服务。 总的来说，DHCP协议的特点包括自动IP地址分配、动态配置、集中管理、租约机制、减少配置错误以及支持其他网络参数。这些特点使得DHCP成为网络管理中重要的协议，提供了便捷性、灵活性和可扩展性。2.3 DHCP解决IP地址冲突

当在DHCP网络中发生IP地址冲突时，DHCP协议采取以下步骤进行处理：

1. IP地址检测：DHCP服务器在向客户端提供IP地址之前，会检测该地址是否已经在网络中被使用。这通常通过发送一个ARP请求（Address Resolution Protocol）来检查IP地址是否已被其他设备使用。如果收到ARP响应，表示IP地址已经被另一个设备占用，那么DHCP服务器会认为发生了IP地址冲突。

2. IP地址冲突处理：一旦DHCP服务器检测到IP地址冲突，它会采取以下措施之一来解决冲突：

- 发送DHCP NAK消息：DHCP服务器可以发送一个DHCP NAK消息（Negative Acknowledgment）给客户端，通知其IP地址冲突，并要求客户端重新请求IP地址。客户端接收到DHCP NAK消息后，会放弃使用冲突的IP地址，并重新启动IP地址分配过程。

- 选择新的IP地址：DHCP服务器可以选择一个新的可用IP地址，并将其分配给客户端。这样避免了冲突的IP地址继续被使用，并确保网络中的设备都具有唯一的IP地址。

3. 重新分配IP地址：在发生IP地址冲突后，客户端会重新启动IP地址获取过程。它会发送DHCP Discover消息，请求新的IP地址分配。DHCP服务器会检测并分配一个未被使用的IP地址给客户端。

需要注意的是，DHCP协议本身并不能完全防止IP地址冲突的发生，因为设备之间可能存在其他方式来手动配置IP地址。然而，DHCP协议的IP地址冲突处理机制可以帮助识别和解决冲突，确保网络中的设备获得唯一的IP地址，并减少IP地址冲突对网络正常运行的影响。

## 2.4 DHCP服务器IP分配

DHCP服务器在分配IP地址时，通常遵循以下原则：

IP地址池：DHCP服务器维护一个IP地址池，其中包含可供分配的IP地址范围。这个地址池是网络管理员在DHCP服务器配置中指定的。

首选IP地址：当DHCP服务器接收到客户端的DHCP发现消息后，它从地址池中选择一个可用的IP地址作为首选IP地址。

租约管理：DHCP服务器使用租约管理来控制IP地址的分配。租约是指分配给客户端的IP地址的有效期限。当客户端接受服务器提供的IP地址时，它会与服务器达成一个租约协议，规定了该地址可以使用的时间。

IP地址状态：IP地址可以处于不同的状态，包括已分配（已租出给客户端）、保留（预留给特定设备）、可用（尚未分配）等。DHCP服务器根据地址池中的IP地址状态来确定可用的地址。

IP地址回收：当租约到期或客户端释放IP地址时，DHCP服务器将回收该IP地址，并将其标记为可用状态，以便后续分配给其他设备。

需要注意的是，DHCP服务器的具体配置和行为可能因厂商、软件版本和网络环境而有所不同。网络管理员可以根据实际需求和网络规模进行适当的配置和管理，以确保有效的IP地址分配和优化的网络资源利用。

## 2.5 DHCP协议的应用：

DHCP协议在计算机网络中有广泛的应用，以下是一些主要的应用场景：

1. 局域网（LAN）中的IP地址分配：在企业或家庭网络中，DHCP协议通常用于自动分配IP地址给局域网内的设备。当设备连接到网络时，它们可以通过DHCP请求自动获取IP地址和其他必要的网络参数，而无需手动配置每个设备的IP地址。

2. 公共无线网络：在公共场所，如咖啡馆、机场或酒店等提供的无线网络中，DHCP协议可以用于动态分配IP地址给连接到网络的移动设备。这样用户可以方便地连接到网络而无需手动配置IP地址。

3. VoIP电话系统：在VoIP（Voice over IP）电话系统中，DHCP协议可以用于为IP电话分配IP地址和其他必要的网络配置。当IP电话设备启动时，它可以通过DHCP请求获取与语音通信相关的网络参数。

4. 网络管理：DHCP协议还可以用于网络管理中的一些任务。例如，管理员可以使用DHCP服务器来限制特定设备的访问权限或为特定设备提供特殊的网络配置。

总结起来，DHCP协议的主要应用领域是在计算机网络中自动分配IP地址和配置其他网络参数，以简化网络管理、提高可扩展性，并提供灵活的地址分配和配置机制。

## 2.6 DHCP服务器管理IP地址租约

DHCP服务器通过租约管理来管理IP地址的分配和续约。以下是DHCP服务器如何管理IP地址租约的一般流程：

1. IP地址分配：当DHCP服务器接收到客户端的DHCP发现消息时，它从地址池中选择一个可用的IP地址，并与客户端建立一个租约。服务器将分配的IP地址和其他网络参数发送给客户端。

2. 租约时间：租约包括一个租约时间，即客户端可以使用该IP地址和其他配置的时间。租约时间可以由网络管理员在DHCP服务器配置中设置，默认情况下通常为一定的时间段（如几个小时或几天）。

3. 租约到期前续约：在租约到期之前，客户端可以发送DHCP续约消息给DHCP服务器，请求延长租约时间。DHCP服务器检查续约请求，并根据策略决定是否批准续约。如果续约被批准，租约时间将被延长。

4. 租约到期释放：如果客户端未能在租约到期之前发送续约请求，或者续约请求被拒绝，租约将到期。在这种情况下，DHCP服务器可以将该IP地址标记为可用状态，并可以将该地址重新分配给其他设备。

5. 地址冲突检测：DHCP服务器在分配IP地址之前，通常会进行地址冲突检测，以确保所分配的IP地址在网络中唯一。这可以避免多个设备使用同一IP地址而导致网络通信问题。

通过租约管理，DHCP服务器可以动态地管理IP地址的分配和续约，确保网络中的设备获得有效的IP地址并避免地址冲突。网络管理员可以根据实际需求和网络规模，调整租约时间和其他相关参数，以最佳地管理IP地址资源。

## 2.7 DHCP协议的场景

家庭网络：在家庭网络中，许多家用路由器都充当DHCP服务器的角色。当家庭中的设备（如电脑、智能手机、平板电脑、智能家居设备等）连接到路由器时，DHCP服务器会自动为这些设备分配IP地址和其他网络配置，简化了家庭网络的设置和管理。

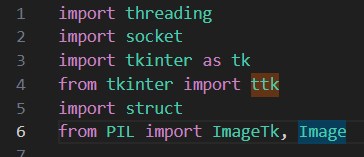
小型办公室/办公楼：在小型办公室或办公楼的局域网中，DHCP协议也被广泛使用。通过DHCP服务器，可以自动为连接到网络的设备分配IP地址和配置，无需手动进行网络设置，提高了网络的管理效率。

公共场所网络：在公共场所，如咖啡馆、酒店、机场等，提供无线网络服务的场所，通常会使用DHCP协议来为连接到网络的客户端设备提供IP地址和网络配置。这样，用户可以方便地连接到网络，无需手动配置网络参数。

企业网络：在大型企业网络中，DHCP协议被广泛用于自动分配IP地址和配置，减轻了网络管理员的负担。通过DHCP服务器集中管理IP地址和配置，可以更好地控制和监测网络中的设备，并提供更高的网络灵活性和可扩展性。

# 三、分析程序代码

## 4.1 导入一些必要的库



上述代码片段引入了几个库：

1. `tkinter`：这是 Python 的标准图形用户界面（GUI）库，用于创建窗口、按钮、标签等用户界面元素，实现图形化的应用程序。它提供了一套简单的接口，用于与用户进行交互。

2. `ttk`：这是 tkinter 的一个模块，提供了一套主题化的用户界面控件，包括按钮、标签、文本框等，可以用于创建更现代化和美观的用户界面。

3. `threading`：这是 Python 的一个内置模块，用于进行多线程编程。它提供了创建和管理线程的功能，可以在程序中同时执行多个任务，提高程序的并发性和响应性。

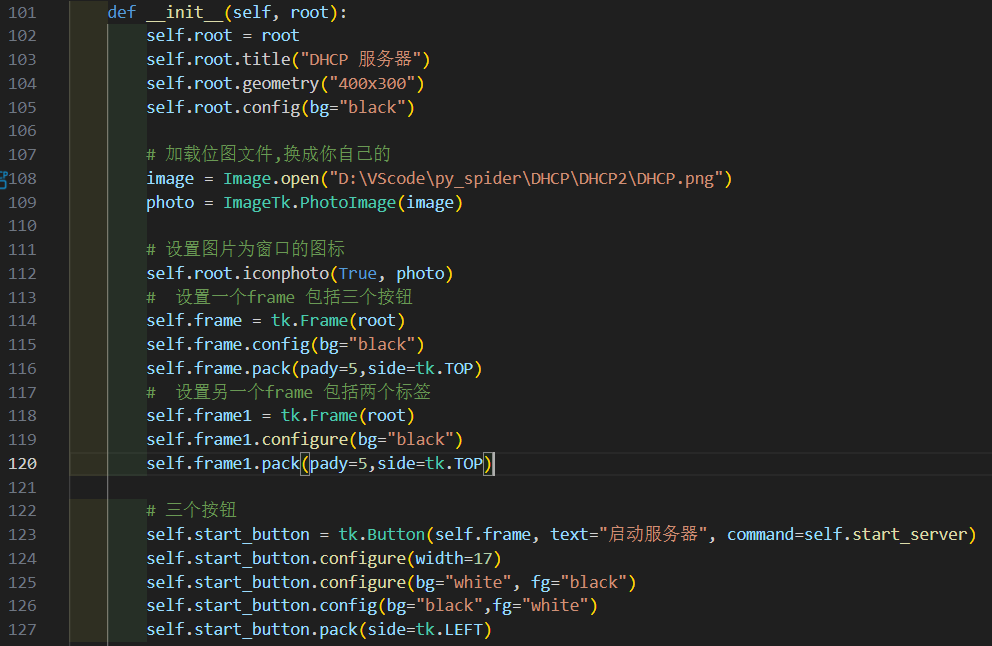
4. `socket`：这是 Python 的一个标准库，用于网络编程。它提供了创建和使用套接字（socket）的功能，用于在网络上进行通信，包括建立连接、发送和接收数据等。

5. `struct`：这也是 Python 的一个标准库，用于处理二进制数据和结构体。它提供了一组函数，用于将数据打包成二进制格式或从二进制格式解析数据，用于处理底层的网络通信和数据传输。

通过引入这些库，可以实现基于 tkinter 的图形化用户界面，同时通过 threading、socket 和 struct 等库来实现网络通信和数据处理的功能。

## 4.2 构建GUI界面启动主程序





上述代码是一个使用tkinter库创建GUI界面的DHCP服务器应用程序的初始化方法。让我逐行解释它：

1. `def \_\_init\_\_(self, root):`: 这是一个类的构造函数，接受一个`root`参数作为根窗口对象。

2. `self.root = root`: 将传入的根窗口对象保存到`self.root`变量中。

3. `self.root.title("DHCP 服务器")`: 设置窗口标题为"DHCP 服务器"。

4. `self.root.geometry("400x300")`: 设置窗口的初始大小为400x300像素。

5. `self.root.config(bg="black")`: 设置窗口的背景色为黑色。

6. `image = Image.open("D:\VScode\py\_spider\DHCP\DHCP2\DHCP.png")`: 打开一个位图文件，文件路径为"D:\VScode\py\_spider\DHCP\DHCP2\DHCP.png"，并将其赋值给`image`变量。

7. `photo = ImageTk.PhotoImage(image)`: 使用`ImageTk`模块将位图文件转换为图像对象`photo`，以便在窗口中显示。

8. `self.root.iconphoto(True, photo)`: 将上一步中创建的图像对象`photo`设置为窗口的图标。

9. `self.frame = tk.Frame(root)`: 创建一个框架对象`self.frame`，它是在根窗口`root`中放置其他组件的容器。

10. `self.frame.config(bg="black")`: 设置框架的背景色为黑色。

11. `self.frame.pack(pady=5,side=tk.TOP)`: 将框架放置在根窗口的顶部，并设置上边距为5个像素。

12. `self.frame1 = tk.Frame(root)`: 创建另一个框架对象`self.frame1`，用于放置标签等组件。

13. `self.frame1.configure(bg="black")`: 设置第二个框架的背景色为黑色。

14. `self.frame1.pack(pady=5,side=tk.TOP)`: 将第二个框架放置在根窗口的顶部，并设置上边距为5个像素。

15. `self.start\_button = tk.Button(self.frame, text="启动服务器", command=self.start\_server)`: 创建一个按钮对象`self.start\_button`，放置在第一个框架中，按钮上显示文本为"启动服务器"，点击按钮时调用`self.start\_server`方法。

16. `self.start\_button.configure(width=17)`: 设置按钮的宽度为17个字符。

17. `self.start\_button.configure(bg="white", fg="black")`: 设置按钮的背景色为白色，前景色（文本颜色）为黑色。

18. `self.start\_button.config(bg="black",fg="white")`: 进一步设置按钮的背景色为黑色，前景色为白色。

19. `self.start\_button.pack(side=tk.LEFT)`: 将启动服务器按钮放置在第一个框架的左侧。

20. `self.stop\_button = tk.Button(self.frame, text="停止服务器", command=self.stop\_server, state=tk.DISABLED)`: 创建另一个按钮对象`self.stop\_button`，放置在第一个框架中，按钮上显示文本为"停止服务器"，点击按钮时调用`self.stop\_server`方法，初始状态设置为禁用状态。

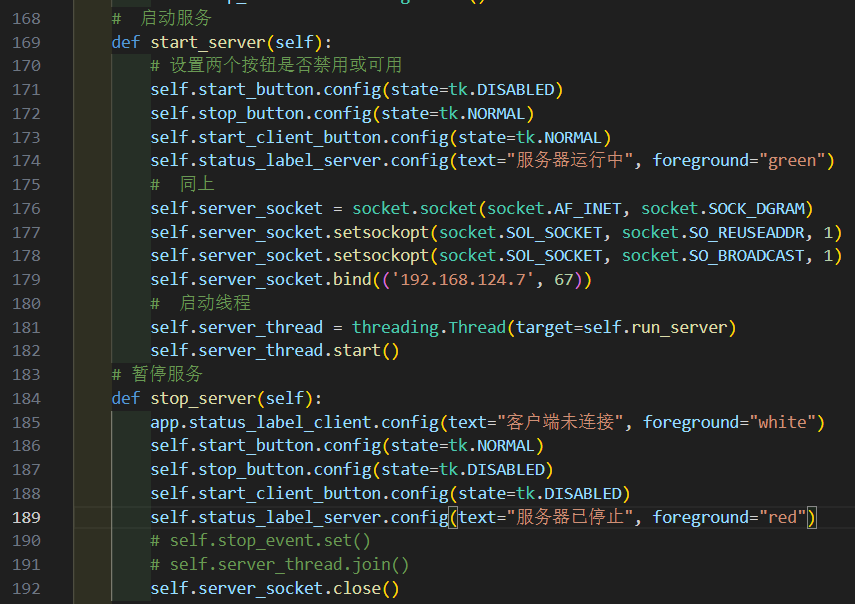
21. `self.stop\_button.configure(bg="black", fg="white")`: 设置停止服务器按钮的背景色为黑色，前景色为白色。

22. `self.stop\_button.pack(side=tk.LEFT)`: 将停止服务器按钮放置在第一个框架的左侧。

23. `self.stop\_button.config(width=17)`: 设置停止服务器按钮的宽度为17个字符。

这段代码创建了一个具有两个按钮的GUI界面，用于启动和停止DHCP服务器。界面还包含了一个位图图像和两个框架，用于放置其他组件。

## 4.3服务端功能



上述代码是一个用于启动和停止DHCP服务器的两个方法：`start\_server`和`stop\_server`。让我逐行解释它们：

`start\_server`方法：

1. `self.start\_button.config(state=tk.DISABLED)`: 禁用启动服务器按钮。

2. `self.stop\_button.config(state=tk.NORMAL)`: 启用停止服务器按钮。

3. `self.start\_client\_button.config(state=tk.NORMAL)`: 启用启动客户端按钮。

4. `self.status\_label\_server.config(text="服务器运行中", foreground="green")`: 设置状态标签为"服务器运行中"，并将文本颜色设置为绿色。

5. `self.server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)`: 创建一个UDP套接字对象。

6. `self.server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)`: 设置套接字选项，允许地址重用。

7. `self.server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)`: 设置套接字选项，允许广播通信。

8. `self.server\_socket.bind(('192.168.124.7', 67))`: 将套接字绑定到IP地址"192.168.124.7"和端口号67。

9. `self.server\_thread = threading.Thread(target=self.run\_server)`: 创建一个线程对象，目标函数为`self.run\_server`，即运行服务器的方法。

10. `self.server\_thread.start()`: 启动服务器线程。

`stop\_server`方法：

1. `app.status\_label\_client.config(text="客户端未连接", foreground="white")`: 设置客户端状态标签为"客户端未连接"，并将文本颜色设置为白色。

2. `self.start\_button.config(state=tk.NORMAL)`: 启用启动服务器按钮。

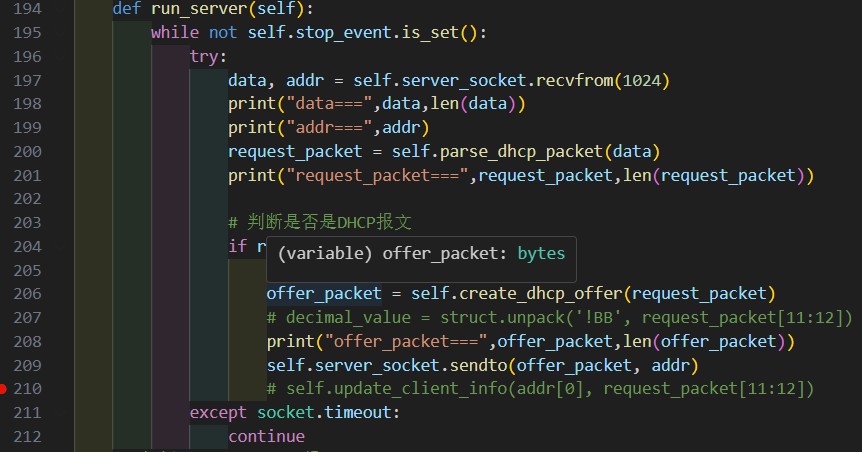
3. `self.stop\_button.config(state=tk.DISABLED)`: 禁用停止服务器按钮。

4. `self.start\_client\_button.config(state=tk.DISABLED)`: 禁用启动客户端按钮。

5. `self.status\_label\_server.config(text="服务器已停止", foreground="red")`: 设置状态标签为"服务器已停止"，并将文本颜色设置为红色。

6. `self.server\_socket.close()`: 关闭服务器的套接字连接。

这些方法实现了通过按钮控制DHCP服务器的启动和停止。在`start\_server`方法中，服务器按钮禁用，停止服务器按钮启用，服务器状态标签更新，并创建一个线程来运行服务器。在`stop\_server`方法中，客户端状态标签更新，服务器按钮启用，停止服务器按钮禁用，并关闭服务器套接字连接。



上述代码是一个名为`run\_server`的方法，用于在DHCP服务器上运行主循环以接收和处理客户端请求。让我逐行解释它：

1. `while not self.stop\_event.is\_set():`: 在停止事件未设置的情况下，执行循环。

2. `data, addr = self.server\_socket.recvfrom(1024)`: 使用服务器套接字从客户端接收数据，最大接收字节数为1024，并将数据和客户端地址存储在`data`和`addr`变量中。

3. `request\_packet = self.parse\_dhcp\_packet(data)`: 调用`parse\_dhcp\_packet`方法解析接收到的DHCP数据报文，并将解析后的结果存储在`request\_packet`变量中。

4. `if request\_packet[0] == 1:`: 检查解析后的DHCP数据报文中的第一个字节是否为1，表示DHCP请求报文。

5. `offer\_packet = self.create\_dhcp\_offer(request\_packet)`: 调用`create\_dhcp\_offer`方法创建DHCP Offer报文，传入解析后的DHCP请求报文作为参数，并将创建的报文存储在`offer\_packet`变量中。

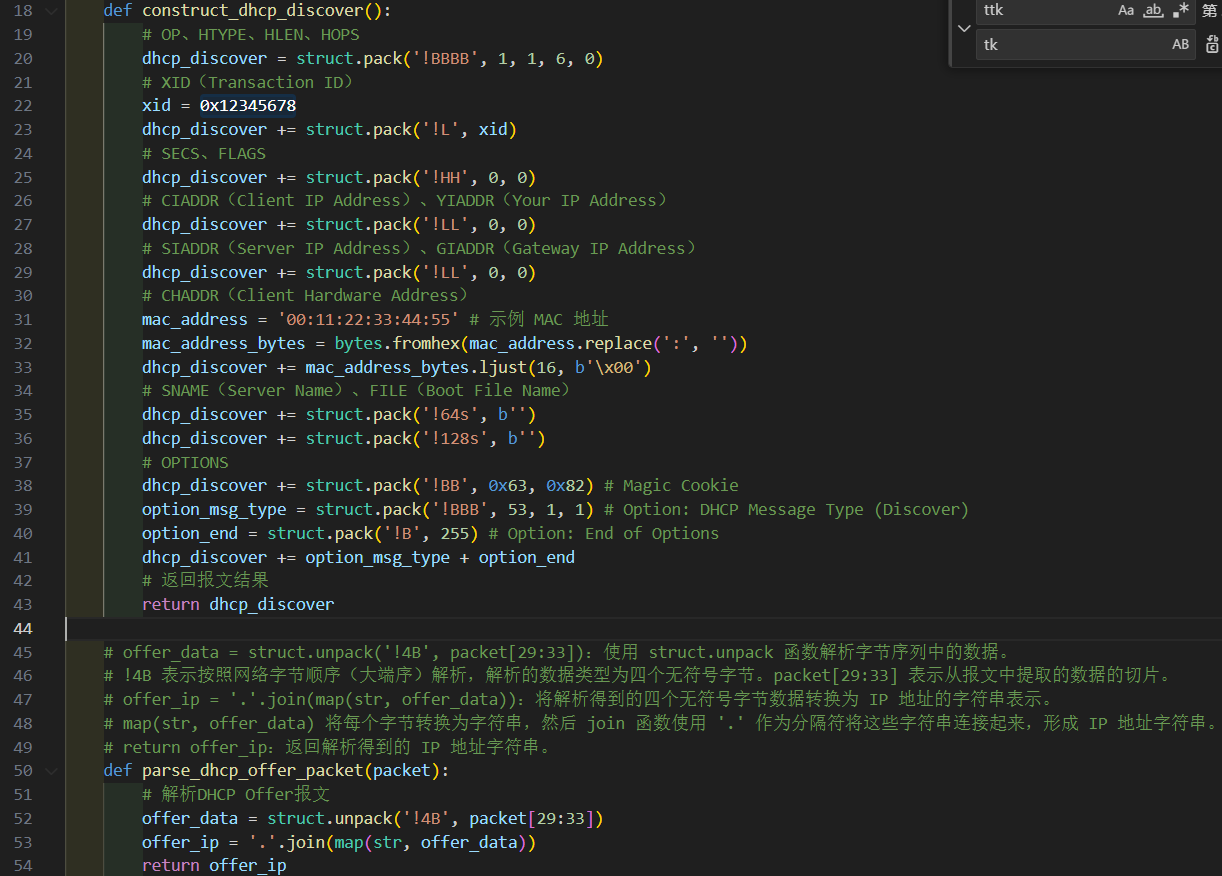
6. `self.server\_socket.sendto(offer\_packet, addr)`: 使用服务器套接字将DHCP Offer报文发送给客户端，发送目标地址为客户端地址。

7. `except socket.timeout: continue`: 如果发生套接字超时异常，继续下一次循环，即继续等待接收客户端的请求。

这段代码实现了DHCP服务器的主循环，它接收从客户端发送的DHCP请求报文，解析请求报文并根据请求创建相应的DHCP Offer报文，然后将Offer报文发送回客户端。循环持续运行，直到停止事件被设置，即服务器停止。



## 4.4 客户端功能



上述代码包含两个函数：`construct\_dhcp\_discover`和`parse\_dhcp\_offer\_packet`。让我逐行解释这两个函数的功能：

`construct\_dhcp\_discover`函数：

该函数用于构建DHCP Discover报文字段。下面是函数的逐行解释：

1. `dhcp\_discover = struct.pack('!BBBB', 1, 1, 6, 0)`: 创建一个字节序列，按照网络字节顺序（大端序）打包了4个无符号字节，分别代表DHCP Discover报文的OP、HTYPE、HLEN和HOPS字段。

2. `xid = 0x12345678`: 设置Transaction ID（XID）为0x12345678。

3. `dhcp\_discover += struct.pack('!L', xid)`: 将XID打包为4个字节的无符号长整型，并将其添加到dhcp\_discover字节序列中。

4. `dhcp\_discover += struct.pack('!HH', 0, 0)`: 将两个16位的无符号整数（SECS和FLAGS）打包为4个字节，并将其添加到dhcp\_discover字节序列中。

5. `dhcp\_discover += struct.pack('!LL', 0, 0)`: 将两个32位的无符号长整型（CIADDR和YIADDR）打包为8个字节，并将其添加到dhcp\_discover字节序列中。

6. `mac\_address = '00:11:22:33:44:55'`: 设置示例的MAC地址。

7. `mac\_address\_bytes = bytes.fromhex(mac\_address.replace(':', ''))`: 将MAC地址转换为字节序列，去除其中的冒号，并将其存储在mac\_address\_bytes变量中。

8. `dhcp\_discover += mac\_address\_bytes.ljust(16, b'\x00')`: 将MAC地址字节序列添加到dhcp\_discover字节序列中，并使用\x00字节填充到总长度为16字节。

9. `dhcp\_discover += struct.pack('!64s', b'')`: 将一个64字节的空字节序列添加到dhcp\_discover字节序列中，表示SNAME（Server Name）字段。

10. `dhcp\_discover += struct.pack('!128s', b'')`: 将一个128字节的空字节序列添加到dhcp\_discover字节序列中，表示FILE（Boot File Name）字段。

11. `dhcp\_discover += struct.pack('!BB', 0x63, 0x82)`: 将两个字节的数值（0x63和0x82）打包到dhcp\_discover字节序列中，表示Magic Cookie。

12. `option\_msg\_type = struct.pack('!BBB', 53, 1, 1)`: 将三个字节的数值（53、1和1）打包到option\_msg\_type字节序列中，表示DHCP Message Type（Discover）选项。

13. `option\_end = struct.pack('!B', 255)`: 将一个字节的数值（255）打包到option\_end字节序列中，表示Options结束。

14. `dhcp\_discover += option\_msg\_type + option\_end`: 将option\_msg\_type和option\_end字节序列添加到dhcp\_discover字节序列中，表示OPTIONS字段。

15. `return dhcp\_discover`: 返回构建的DHCP Discover报文字段。

`parse\_dhcp\_offer\_packet`函数：

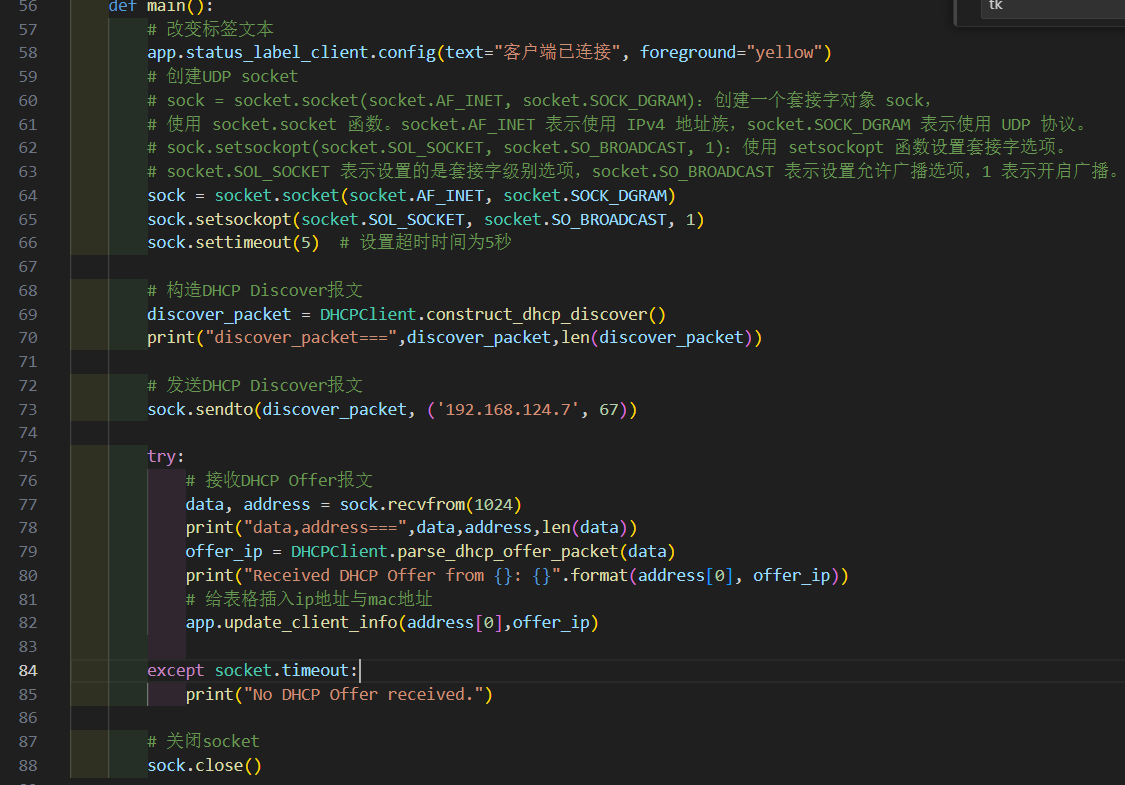
该函数用于解析DHCP Offer报文，并提取其中的IP地址字段。下面是函数的逐行解释：

1. `offer\_data = struct.unpack('!4B', packet[29:33])`: 使用`struct.unpack`函数按照网络字节顺序（大端序）解析从报文中提取的4个字节数据，并将解析结果存储在`offer\_data`变量中。

2. `offer\_ip = '.'.join(map(str, offer\_data))`: 将解析得到的4个无符号字节数据转换为IP地址的字符串表示。`map(str, offer\_data)`将每个字节转换为字符串，然后使用`.`作为分隔符将这些字符串连接起来形成IP地址字符串。

3. `return offer\_ip`: 返回解析得到的IP地址字符串。

这两个函数一起工作，`这两个函数一起工作，`construct\_dhcp\_discover`用于构建DHCP Discover报文字段，而`parse\_dhcp\_offer\_packet`用于解析DHCP Offer报文并提取其中的IP地址字段。这些功能通常用于DHCP客户端和服务器之间的通信，其中客户端使用Discover报文寻找可用的IP地址，并接收服务器发送的Offer报文以获取分配的IP地址。



上述代码定义了一个名为`main`的函数。让我逐行解释该函数的功能：

1. `app.status\_label\_client.config(text="客户端已连接", foreground="yellow")`: 该行代码用于更改`app`对象的`status\_label\_client`标签的文本内容为"客户端已连接"，并将文本颜色设置为黄色。

2. `sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)`: 创建一个UDP套接字对象`sock`，用于进行网络通信。`socket.AF\_INET`表示使用IPv4地址族，`socket.SOCK\_DGRAM`表示使用UDP协议。

3. `sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_BROADCAST, 1)`: 使用`setsockopt`函数设置套接字选项。`socket.SOL\_SOCKET`表示设置套接字级别选项，`socket.SO\_BROADCAST`表示设置允许广播选项，值为1表示开启广播。

4. `sock.settimeout(5)`: 设置套接字的超时时间为5秒，即在接收数据时，如果在5秒内没有接收到数据，将引发`socket.timeout`异常。

5. `discover\_packet = DHCPClient.construct\_dhcp\_discover()`: 调用`DHCPClient`类的`construct\_dhcp\_discover`函数构造DHCP Discover报文，将构造的报文字段存储在`discover\_packet`变量中。

6. `sock.sendto(discover\_packet, ('192.168.124.7', 67))`: 使用套接字`sock`将DHCP Discover报文发送到目标地址`('192.168.124.7', 67)`，其中67是DHCP服务器的端口号。

7. `try:`: 开始异常处理块，用于捕获可能发生的异常。

8. `data, address = sock.recvfrom(1024)`: 使用套接字`sock`接收来自DHCP服务器的DHCP Offer报文，接收到的数据存储在`data`变量中，发送方的地址存储在`address`变量中。

9. `offer\_ip = DHCPClient.parse\_dhcp\_offer\_packet(data)`: 调用`DHCPClient`类的`parse\_dhcp\_offer\_packet`函数解析DHCP Offer报文并提取其中的IP地址字段，将解析得到的IP地址存储在`offer\_ip`变量中。

10. `print("Received DHCP Offer from {}: {}".format(address[0], offer\_ip))`: 打印从DHCP服务器接收到的DHCP Offer报文的源地址和提取得到的IP地址。

11. `app.update\_client\_info(address[0], offer\_ip)`: 调用`app`对象的`update\_client\_info`方法，向表格中插入提取得到的IP地址和MAC地址。

12. `except socket.timeout:`: 如果在接收DHCP Offer报文时发生超时，即在指定的时间内没有接收到数据，将执行下面的代码块。

13. `print("No DHCP Offer received.")`: 打印提示信息，表示没有接收到DHCP Offer报文。

14. `sock.close()`: 关闭套接字，释放资源。

总体来说，该`main`函数的功能是作为DHCP客户端，向DHCP服务器发送DHCP Discover报文，并接收并解析DHCP Offer报文，从中提取IP地址，并将IP地址和相关信息显示在表格中。

# 五、总结

根据DHCP协议服务器实验，并结合上述App类中的代码，以下是一篇总结并指出不同之处：

总结：

DHCP（动态主机配置协议）是一种网络协议，用于自动分配IP地址和其他网络配置参数给网络中的设备。通过实验和上述App类中的代码，我们可以总结如下：

1. DHCP客户端功能：上述App类中的代码实现了一个简单的DHCP客户端功能。它通过构造DHCP Discover报文，向DHCP服务器发送请求，并接收和解析DHCP Offer报文以获取分配的IP地址。

2. 自动IP地址分配：DHCP服务器实验中，DHCP服务器负责接收DHCP Discover报文，并为客户端分配一个可用的IP地址。客户端通过发送DHCP Discover报文来请求IP地址，并在收到DHCP Offer报文后获取分配的IP地址。

3. 网络配置参数：DHCP协议还可以分配其他网络配置参数，如子网掩码、默认网关、DNS服务器等。这些参数可以通过DHCP服务器进行配置，并在DHCP Offer报文中发送给客户端。

不同之处：

尽管上述App类中的代码实现了基本的DHCP客户端功能，但在与实际的DHCP服务器实验相比较时，可能存在以下不同之处：

1. DHCP交互过程：实际的DHCP交互过程可能更加复杂。DHCP客户端通常需要进行多轮的请求和响应，包括DHCP Discover、DHCP Offer、DHCP Request、DHCP Ack等过程。上述代码可能只涵盖了其中的一部分。

2. DHCP选项和参数：DHCP协议支持许多可选的配置参数和选项，如租约时间、域名服务器、NTP服务器等。上述App类中的代码可能没有实现对所有这些选项和参数的处理和解析。

3. 错误处理和容错性：在实际的DHCP实验中，需要考虑更多的错误处理和容错机制。例如，处理服务器响应超时、重试机制、处理服务器拒绝请求等情况。

4. 用户界面：上述App类中的代码提到了GUI表格，用于显示IP地址和相关信息。然而，具体的GUI实现并未提供，因此无法评估其与实际情况的差异。

综上所述，上述App类中的代码提供了一个基本的DHCP客户端功能，但与实际的DHCP服务器实验相比较时，可能存在一些不同之处，如DHCP交互过程的复杂性、支持的选项和参数、错误处理和容错性等方面。在实际应用中，需要根据具体的需求和情况进行相应的调整和扩展。

上述代码实现了一个基本的DHCP客户端功能。它通过发送DHCP Discover报文来寻找可用的IP地址，并接收并解析DHCP Offer报文以获取分配的IP地址。以下是对代码的总结和不足之处：

总结：

- 代码通过使用Python的socket模块创建UDP套接字，并设置套接字选项，实现了与DHCP服务器进行通信的能力。

- 使用构造函数`construct\_dhcp\_discover`构建了DHCP Discover报文的字段，并通过套接字将其发送到指定的DHCP服务器地址和端口。

- 使用`parse\_dhcp\_offer\_packet`函数解析接收到的DHCP Offer报文，并提取其中的IP地址字段。

- 代码通过更新GUI表格显示IP地址和相关信息，提供了一个简单的用户界面。

不足之处：

1. 缺乏错误处理：代码中缺少对各种可能的错误情况的处理，如套接字连接失败、发送失败、接收超时等。在实际应用中，应该考虑添加适当的错误处理机制，以增强代码的健壮性和容错性。

2. 硬编码的地址和端口：代码中直接在`sock.sendto`函数中硬编码了DHCP服务器的地址和端口。这限制了代码的灵活性和可移植性。应该考虑将这些地址和端口作为参数传递给函数，或者从配置文件或用户输入中获取。

3. 缺乏注释和文档：代码中缺乏注释和文档，不清楚每个函数的作用和输入输出。为了增加代码的可读性和可维护性，应该添加适当的注释和提供文档说明。

4. 缺乏代码复用性：代码中的功能实现比较简单，并没有充分考虑到代码的复用性。例如，构造DHCP报文和解析报文的功能可以抽象成独立的函数或类，以便在其他项目中重复使用。

综上所述，虽然代码能够实现基本的DHCP客户端功能，但还有一些改进的空间，包括添加错误处理、提高灵活性、增加注释和文档，以及提高代码的复用性。

# 六、参考文献

以下是一些与DHCP协议相关的论文文献的示例：

1. Droms, R., & Rabil, G. (1997). Dynamic host configuration protocol. Internet Engineering Task Force (IETF), Request for Comments: 2131. [链接](https://tools.ietf.org/html/rfc2131)

2. Lemon, T., & Cheshire, S. (2003). Dynamic Host Configuration Protocol for IPv6 (DHCPv6). Internet Engineering Task Force (IETF), Request for Comments: 3315. [链接](https://tools.ietf.org/html/rfc3315)

3. Droms, R., & Arbaugh, W. (1997). Authentication for DHCP messages. Internet Engineering Task Force (IETF), Request for Comments: 3118. [链接](https://tools.ietf.org/html/rfc3118)

4. Stapp, M., & Lemon, T. (2009). DHCPv6 relay agent. Internet Engineering Task Force (IETF), Request for Comments: 3315. [链接](https://tools.ietf.org/html/rfc3315)

5. Droms, R., & Lemon, T. (2003). The ISC Dynamic Host Configuration Protocol Distribution. Internet Systems Consortium, Inc. [链接](https://www.isc.org/dhcp/)

6. Droms, R., Ed. (2011). DHCPv6 for IPv4-IPv6 Coexistence. Internet Engineering Task Force (IETF), Request for Comments: 6424. [链接](https://tools.ietf.org/html/rfc6424)