UDP-Client Example

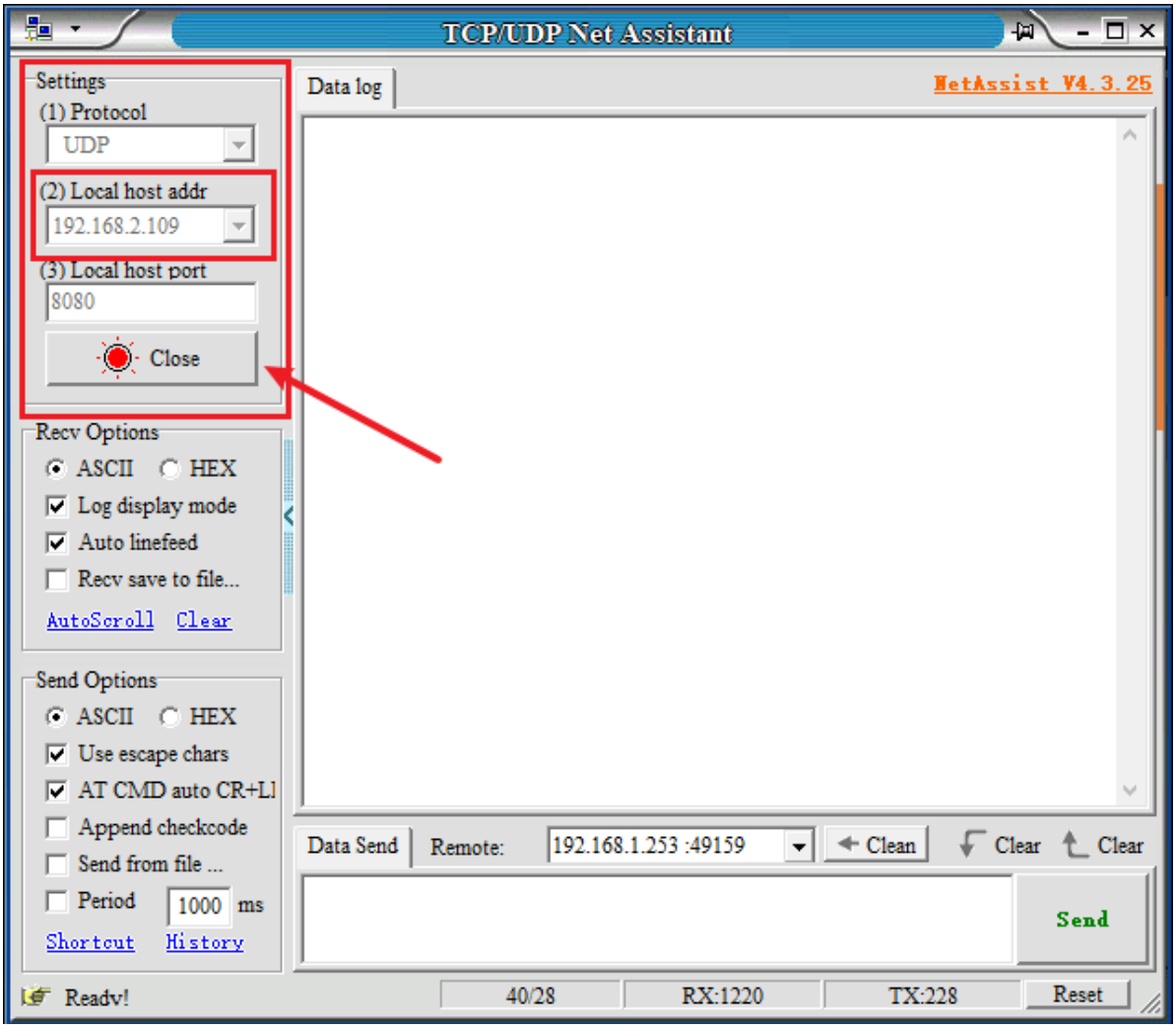
Routine Introduction

In this section, we will introduce how to use K230 as a UDP server

We open the example code in this section and modify the WIFI connection information part in the code to your home WIFI or mobile hotspot

```
sta.connect("WIFI SSID", "WIFI PASSWORD")  # 连接到指定WiFi / Connect to specified WiFi
```

Then we open the network debugging assistant and enter the IP address of the machine (check with ipconfig)



Then we go back to the code and modify this part of the code (modify it to the IP address and port number filled in above)

```
# 设置服务器参数 / Set server parameters
server_ip = '192.168.2.109'
server_port = 8080
```

Click the run button in the lower left corner, and the console will output the information sent to the server.

```
发送字节数 / Bytes sent: 29
发送消息 / Sending: K230 UDP client send test 1

发送字节数 / Bytes sent: 29
发送消息 / Sending: K230 UDP client send test 2

发送字节数 / Bytes sent: 29
发送消息 / Sending: K230 UDP client send test 3

发送字节数 / Bytes sent: 29
发送消息 / Sending: K230 UDP client send test 4

发送字节数 / Bytes sent: 29
发送消息 / Sending: K230 UDP client send test 5

发送字节数 / Bytes sent: 29
发送消息 / Sending: K230 UDP client send test 6
```

```
发送字节数 / Bytes sent: 29
发送消息 / Sending: K230 UDP client send test 7

发送字节数 / Bytes sent: 29
发送消息 / Sending: K230 UDP client send test 8

发送字节数 / Bytes sent: 29
发送消息 / Sending: K230 UDP client send test 9

发送字节数 / Bytes sent: 29
客户端已结束 / Client ended.
```

At the same time, we will go to the network debugging assistant and find that the information sent by K230 has also been received here

```
[2025-02-21 10:37:01.615]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 0


[2025-02-21 10:37:01.679]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 1


[2025-02-21 10:37:01.862]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 2


[2025-02-21 10:37:02.188]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 3


[2025-02-21 10:37:02.378]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 4


[2025-02-21 10:37:02.466]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 5


[2025-02-21 10:37:02.779]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 6


[2025-02-21 10:37:02.869]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 7


[2025-02-21 10:37:03.096]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 8


[2025-02-21 10:37:03.272]# RECV ASCII/30 from 192.168.1.253 :49153 <<<
K230 UDP client send test 9
```

Code Analysis

For complete comments and code, please refer to the file [Source Code/11.Network/04.udp/udp_client.py]

```
import socket
import os
import time
import network
```

At the beginning of the code, the necessary modules are imported:

- `socket`: Provides network communication functions and supports TCP and UDP protocols.
- `os`: Provides functionality for interacting with the operating system, such as exiting a program.
- `time`: Provides time-related functions, such as delay.
- `network`: Provides network related functions, here for WiFi connection.

function `connect_wifi`

```
def connect_wifi(ssid="[WIFI SSID]", password="[WIFI PASSWORD]"):
    """
    连接WiFi并返回IP地址
    Connect to WiFi and return IP address

    参数 / Parameters:
    ssid: WiFi名称 / WiFi name
    password: WiFi密码 / WiFi password

    返回 / Returns:
    str: IP地址 / IP address
    """
```

This function is used to connect to the specified WiFi network and return the IP address after successful connection. Its parameters include the WiFi name (SSID) and password.

```
    wifi_station = network.WLAN(0)  # 创建WiFi站点对象 / Create WiFi station object
    wifi_station.connect(ssid, password)  # 连接到指定WiFi / Connect to specified
WiFi
```

Here a WiFi site object is created and a connection is made to the WiFi network using the provided SSID and password.

```
    # 等待直到获取到IP地址 / Wait until IP address is obtained
    while wifi_station.ifconfig()[0] == '0.0.0.0':
        os.exitpoint()
```

Check if the IP address has been obtained by `ifconfig()` the method. If not (that is, the IP address is `0.0.0.0`), enter the loop and keep checking until a valid IP address is obtained. The here `os.exitpoint()` may be an error, usually it should be `os._exit(1)` or `raise SystemExit`.

```
    return wifi_station.ifconfig()[0]  # 返回IP地址 / Return IP address
```

Once a valid IP address is obtained, it is returned.

function `start_udp_client`

```
def start_udp_client():
    """
    启动UDP客户端，发送测试消息
    Start UDP client and send test messages
    """
```

This function is responsible for starting the UDP client and sending a test message.

```
    # 连接WiFi网络 / Connect to WiFi network
    connect_wifi()
```

Call `connect_wifi` function to connect to WiFi.

```
    # 设置服务器参数 / Set server parameters
    server_ip = '192.168.2.89'
    server_port = 8080
```

Defines the IP address and port of the UDP server to connect to.

```
    # 获取服务器地址信息 / Get server address information
    address_info = socket.getaddrinfo(server_ip, server_port)
    print("地址信息 / Address info:", address_info)

    server_address = address_info[0][-1]
    print("连接地址 / Connect address:", server_address)
```

Use `socket.getaddrinfo()` to obtain the server's address information and extract the actual connection address (the last element of the tuple).

```
    # 创建UDP套接字 / Create UDP socket
    udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Create a UDP socket.

Send a test message

```
    # 发送测试消息 / Send test messages
    try:
        for msg_count in range(10):
            # 构建消息 / Build message
            message = f"K230 UDP client send test {msg_count}\r\n"
            print("发送消息 / Sending:", message)

            # 发送消息并获取发送字节数 / Send message and get bytes sent
            bytes_sent = udp_socket.sendto(message.encode(), server_address)
            print("发送字节数 / Bytes sent:", bytes_sent)

            # 等待200ms / Wait 200ms
            time.sleep(0.2)
```

In `try` the block, a loop sends 10 test messages:

- The message content is a formatted string, including the message number.
- Use `sendto` the method to send a message and print the number of bytes sent.
- After each message is sent, the program will wait 200 milliseconds.

Closing a Socket

```
    finally:
        # 关闭套接字 / Close socket
        udp_socket.close()
        print("客户端已结束 / Client ended.")
```

Regardless of whether the message sending process is successful or not, `finally` the UDP socket will be closed in the block and the end information will be printed.

Main program entry

```
# 启动客户端 / Start client
if __name__ == '__main__':
    start_udp_client()
```

If this script is the main program running, the function is called `start_udp_client()` to start the UDP client.

UDP protocol

1. Basic knowledge of UDP

1.1 What is UDP?

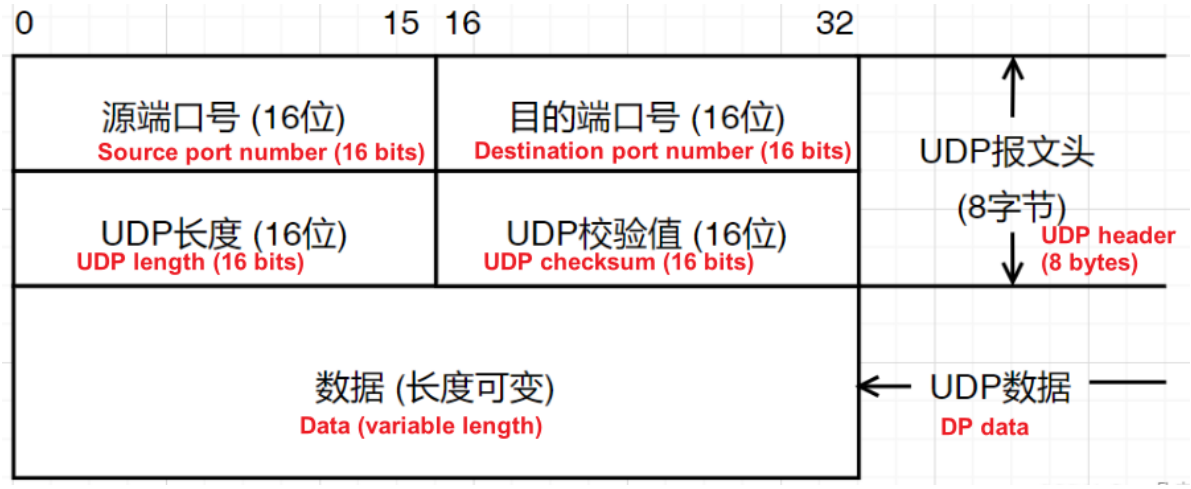
- **Positioning** : Transport layer protocol (OSI layer 4)
- **Features** : connectionless, best effort delivery, lightweight header, low latency
- **Analogy** : Like the post office's "normal postcard service" (no guarantee of delivery, but fast delivery)

1.2 UDP vs TCP Core Differences

characteristic	UDP	TCP
Connection Establishment	No handshake	Three-way handshake
Data transmission guarantee	Order and arrival are not guaranteed	Completely reliable (logically reliable only from this layer)
Header overhead	8 bytes	20-60 bytes
Transport Control	No flow control/congestion control	Fine-grained control
Applicable scenarios	Real-time communication	Reliable transmission

2. UDP protocol structure (diagram)

2.1 Datagram Format



2.2 Header Structure

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Length             |            Checksum           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               Data                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- **Source Port** (16 bits): Optional field, 0 indicates a passive port
- **Destination Port** (16 bits): Required destination port
- **Length** (16 bits): Total number of bytes of header + data (minimum 8 bytes)
- **Checksum** (16 bits): Checksum (optional for IPv4, mandatory for IPv6)

3. UDP key working mechanism

3.1 Connectionless Communication

- **Unicast** : one-to-one transmission
- **Multicast** : one-to-many transmission (IP multicast address range: 224.0.0.0~239.255.255.255)
- **Broadcast** : One-to-all transmission (limited broadcast address 255.255.255.255)

3.2 Checksum calculation

- **Coverage** : pseudo header + UDP header + data
- **Pseudo header structure** :

```
| Source IP address (32 bits) | Destination IP address (32 bits) | Protocol (8 bits) | UDP length (16 bits) |
```

---

4. UDP Advanced Features

4.1 Multiplexing and Demultiplexing

- **Port number** : Different applications are distinguished by port number (0~65535)

4.2 Application layer reliability implementation

- **Custom retransmission** : such as packet loss detection in the QUIC protocol
- **Forward Error Correction** : Adding Redundant Data to Deal with Packet Loss
- **Sequence number mechanism** : add custom sequence numbers in data payload

---

5. Typical UDP application scenarios

5.1 Real-time audio and video transmission

- **Zoom/Teams** : Allowing a small amount of packet loss in exchange for low latency
- **Streaming protocol** : RTP over UDP

5.2 Game Communication

- **MOBA games** : high-frequency position updates (20-30 times per second)
- **FPS shooting game** : quickly transmit operation instructions

5.3 Network Service Discovery

- **DHCP** : Dynamically obtain an IP address
- **mDNS** : LAN device discovery (such as printers)

5.4 IoT Communications

- **CoAP** : A lightweight protocol for constrained devices
- **MQTT-SN** : Sensor Network Protocol