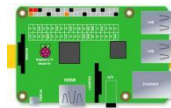



Lesson 17 Introduction to the Principle of TCP

17.1 Overview

In this lesson, we will delve into the principle of TCP (Transmission Control Protocol) and create a simple remote - control system using Python on a Raspberry Pi. By the end of this lesson, you will understand how TCP enables reliable communication between devices, and be able to implement a basic client - server model for remote control applications.

17.2 Required Components

Components	Quantity	Picture
Raspberry Pi	1	
Adeept Robot HAT V3.2	1	

17.3 Principle Introduction

17.3.1 TCP Basics

TCP is a transport - layer protocol in the TCP/IP protocol suite. It provides reliable, ordered, and error - checked delivery of data between applications running on different devices over a network. The key features of TCP include:

Connection - Oriented: Before data transfer, TCP establishes a connection between the client and the server. This is similar to making a phone call, where you need to dial the number (initiate the connection) before you can talk. The connection - establishment process is known as the three - way handshake.

- Step 1: SYN (Synchronize): The client sends a SYN packet to the server, indicating its intention to establish a connection.
- Step 2: SYN - ACK (Synchronize - Acknowledge): The server responds with a SYN - ACK packet, which is an acknowledgment of the client's SYN request and also contains the server's own SYN request.
- Step 3: ACK (Acknowledge): The client sends an ACK packet to the server, completing the connection establishment.

Reliable Delivery: TCP uses sequence numbers and acknowledgments to ensure that all data packets are received correctly. Each packet sent by the sender is assigned a sequence number. The receiver acknowledges the receipt of packets by sending ACK packets back to the sender. If a packet is not acknowledged within a certain time (timeout), the sender will re - send the packet.

Flow Control: TCP has a mechanism to prevent the sender from overwhelming the receiver with data. It uses a sliding window protocol, where the receiver advertises its available buffer space (window size) to the sender. The sender can only send as much data as the receiver's available window size.

17.3.2 Remote - Control Principle

In TCP based remote control systems, the client is the device that initiates control commands. It sends data (control signals) to the server through a TCP connection. After receiving the data, the server will interpret it and perform corresponding operations.

17.4 Wiring Diagram

Since this is a software - based TCP communication tutorial focused on the Raspberry Pi, there is no complex hardware wiring in the traditional sense. However, for the network connection:

Wired (Ethernet): Connect one end of the Ethernet cable to the Ethernet port of the first Raspberry Pi (server) and the other end to the Ethernet port of the second Raspberry Pi (client). If you are using a network switch or router, you can connect both Raspberry Pi devices to the switch/router using Ethernet cables.

Wireless (Wi - Fi): On both Raspberry Pi devices, configure the Wi - Fi settings to connect to the same wireless network. You can do this through the Raspberry Pi's graphical user interface (if available) or by editing the network configuration files in the terminal.

Note: This tutorial will use wireless connectivity as an example to explain.

17.5 Demonstration

1. **Remotely log:** Remote login as a server-side Raspberry Pi terminal.

2. **Navigate to the Program Folder:** Enter the following command in the terminal and press **Enter** to access the folder where the program is located:

```
cd Adeept_AWR-V3/Examples/11_Remote_Control/
```

```
pi@raspberrypi:~ $ cd Adeept_AWR-V3/Examples/11_Remote_Control/  
pi@raspberrypi:~/Adeept_AWR-V3/Examples/11_Remote_Control $
```

3. **View Directory Contents:** Type "ls" in the terminal and press Enter. This will display all the files in the current directory, ensuring that the "**Server.py**" and "**Client.py**" file is present:

```
ls  
  
pi@raspberrypi:~/Adeept_AWR-V3/Examples/11_Remote_Control $ ls  
Client.py  LedClient.py  LedServer.py  Server.py
```

4. Enter the command below and press **Enter** to start the **Server.py** program:

```
sudo python3 Server.py  
  
pi@raspberrypi:~/Adeept_AWR-V3/Examples/11_Remote_Control $ sudo python3 Server.py  
Server has started and is listening for connections...
```

5. To run the client program, you need to provide the server's IP address as a parameter. Use the following command, replacing **<server_ip>** with the actual IP address of the Raspberry Pi running the server:

```
sudo python3 Client.py <server_ip>  
  
pi@raspberrypi:~/Adeept_AWR-V3/Examples/11_Remote_Control $ sudo python3 Client.py 192.168.3.130  
Connected to server 192.168.3.130:8000  
Please enter the message to send (type 'exit' to quit):
```

For example, my Raspberry Pi IP address is "192.168.3.130", and the command to run the client program is as follows:

sudo python3 Client.py 192.168.3.130

- On the Raspberry Pi client, the script will prompt you to enter a message. For example, you can input "Hello". After inputting the message, the client will send it to the server through a TCP connection.

Client - side:

```
pi@raspberrypi:~/Adeept_AWR-V3/Examples/11_Remote_Control $ sudo python3 Client.py 192.168.3.130
Connected to server 192.168.3.130:8000
Please enter the message to send (type 'exit' to quit): Hello
```

- On the server-side terminal, you will see the received commands printed out.

Server - side:

```
pi@raspberrypi:~/Adeept_AWR-V3/Examples/11_Remote_Control $ sudo python3 Server.py
Server has started and is listening for connections...
Accepted connection from ('192.168.3.130', 50298)
Enter message to send to client (type 'exit' to quit): Received from ('192.168.3.130', 50298): Hello
```

- On the Raspberry Pi server, the script will prompt you to enter a message. For example, you can enter "Hello". After inputting the message, the client will send it to the client through a TCP connection.

Server - side:

```
pi@raspberrypi:~/Adeept_AWR-V3/Examples/11_Remote_Control $ sudo python3 Server.py
Server has started and is listening for connections...
Accepted connection from ('192.168.3.130', 50298)
Enter message to send to client (type 'exit' to quit): Received from ('192.168.3.130', 50298): Hello
I am a service
Enter message to send to client (type 'exit' to quit):
```

- On the client terminal, you will see the received command printed out.

Client - side:

```
pi@raspberrypi:~/Adeept_AWR-V3/Examples/11_Remote_Control $ sudo python3 Client.py 192.168.3.130
Connected to server 192.168.3.130:8000
Please enter the message to send (type 'exit' to quit): Hello
Please enter the message to send (type 'exit' to quit): Received from server: I am a service
```

Termination:

When you want to terminate a running program, you can press the "**Ctrl+C**" shortcut key on the keyboard or enter "**exit**" on the keyboard and click "**Enter**"

17.6 Code

Complete code refer to [Server.py](#).

```
01 #!/usr/bin/env/python
02 # File name : Server.py
```

```
03 # Website      : www.Adeept.com
04 # Author       : Adeept
05 # Date        : 2025/03/11
06 import socket
07 import threading
08
09 should_exit = False
10
11 def handle_client(client_socket, client_address):
12     global should_exit
13     while not should_exit:
14         try:
15             client_socket.settimeout(1)
16             data = client_socket.recv(1024)
17             if data:
18                 message = data.decode('utf-8')
19                 print(f"Received from {client_address}: {message}")
20             else:
21                 break
22         except socket.timeout:
23             continue
24         except Exception as e:
25             print(f"Error communicating with {client_address}: {e}")
26             break
27     client_socket.close()
28
29 def send_message(client_socket):
30     global should_exit
31     while not should_exit:
32         try:
33             message = input("Enter message to send to client (type 'exit' to quit): ")
34             if message.lower() == 'exit':
35                 should_exit = True
36                 break
37             client_socket.send(message.encode('utf-8'))
38         except Exception as e:
39             print(f"Error sending message: {e}")
40             break
41
42 # Create a TCP - based socket object
43 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
44
45 # Bind to all available network interfaces, port number set to 8000
46 server_address = ('0.0.0.0', 8000)
47 server_socket.bind(server_address)
48
49 # Start listening, maximum number of connections is 5
50 server_socket.listen(5)
51 server_socket.settimeout(1)
52 print("Server has started and is listening for connections...")
53
54 connected_threads = []
55
56 while not should_exit:
57     try:
58         client_socket, client_address = server_socket.accept()
```

```

59     print(f"Accepted connection from {client_address}")
60     # Start the thread that receives client messages
61     client_thread = threading.Thread(target=handle_client, args=(client_socket, client_address))
62     client_thread.start()
63     connected_threads.append(client_thread)
64     # Start the thread that sends messages to the client
65     send_thread = threading.Thread(target=send_message, args=(client_socket,))
66     send_thread.start()
67     connected_threads.append(send_thread)
68     except socket.timeout:
69         continue
70     except Exception as e:
71         if should_exit:
72             break
73         print(f"Error accepting connection: {e}")
74
75
76 server_socket.close()
77
78 for thread in connected_threads:
79     thread.join()
80 print("Server has stopped.")

```

Complete code refer to [Client.py](#).

```

01 #!/usr/bin/env/python
02 # File name   : Client.py
03 # Website    : www.Adeept.com
04 # Author     : Adeept
05 # Date      : 2025/03/11
06 import socket
07 import sys
08 import threading
09
10 should_exit = False
11
12 def receive_message(client_socket):
13     global should_exit
14     while not should_exit:
15         try:
16             client_socket.settimeout(1)
17             data = client_socket.recv(1024)
18             if data:
19                 message = data.decode('utf-8')
20                 print(f"Received from server: {message}")
21             else:
22                 break
23         except socket.timeout:
24             continue
25         except Exception as e:
26             print(f"Error receiving message: {e}")
27             break
28
29 if len(sys.argv) != 2:
30     print("Please enter the server's IP address when running, for example: python3 Client.py 192.168.1.100")
31     sys.exit(1)

```

```
33
34 server_ip = sys.argv[1]
35 server_port = 8000
36
37 # Create a TCP - based socket object
38 client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
39
40 try:
41     # Connect to the server
42     client_socket.connect((server_ip, server_port))
43     print(f"Connected to server {server_ip}:{server_port}")
44     # Start the thread that receives server messages
45     receive_thread = threading.Thread(target=receive_message, args=(client_socket,))
46     receive_thread.start()
47
48     while True:
49         # Get input from the keyboard
50         message = input("Please enter the message to send (type 'exit' to quit): ")
51         if message.lower() == 'exit':
52             should_exit = True
53             break
54         # Send the message to the server
55         client_socket.send(message.encode('utf-8'))
56 except socket.error as e:
57     print(f"Error connecting to the server: {e}")
58 finally:
59     client_socket.close()
60     if 'receive_thread' in locals():
61         receive_thread.join()
62
```

Code explanation

Server.py

Initialization Stage:

Import the Python socket module. Then, create a TCP socket using IPv4, bind it to 0.0.0.0:8080, and set it to listen with a queue of 5 connections.

Loop Control Process:

After entering an infinite outer while True loop, the following steps are executed:

Stage 1: The outer while True loop continuously waits for new client connections. When a client connects, `server_socket.accept()` returns a new `client_socket` for communicating with the client and the `client_address`.

Stage 2: After that, two threads are created. One thread runs the `handle_client` function to handle incoming messages from the client, and the other runs the `send_message` function to send

messages to the client. This enables the server to handle both sending and receiving messages from the client simultaneously.

Client.py

Initialization Stage:

Import the socket library, prompt the user to enter the server's IP address, and create a TCP socket.

Loop Control Process:

After entering an infinite outer while True loop, the following steps are executed:

Stage 1: Try to connect to the server using `client_socket.connect((server_ip, server_port))`. If the connection is successful, it prints a connection - established message.

Stage 2: Create a thread to run the `receive_message` function. This allows the client to receive messages from the server in the background while the main thread is used for sending messages.

Stage 3: In the main while True loop, the client waits for user input. Once the user enters a message, it is sent to the server using `client_socket.send(message.encode('utf - 8'))`. If the user types 'exit', the loop breaks, and the program stops sending messages.

Stage 4: The try - except - finally block is used to handle socket - related errors. In the finally block, the `client_socket` is closed to release resources.