深 圳 大 学 实 验 报 告

课程名称：计算机网络(Computer Networks)

实验名称：Application Layer Assignment

学院：电子与信息工程学院

专业：电子信息工程

指导教师：毕宿志

报告人：贾苏健 班级：电信 06 学号：2022280485

实验时间：2023.10.30

实验报告提交时间：2023.11.2

教务部制

1. Purpose of experiment

Experiment 1:

(1) Understand and learn how to write a basic web server using Python that can receive, parse, and respond to HTTP request messages.

(2) Implement basic HTTP server functionalities to enable the server to locate and read specific files within the server's file system based on client requests, returning the file contents in the form of HTTP response messages.

(3) Handle GET requests and return the appropriate content or a 404 Not Found error message based on whether the requested file exists.

(4) Practice basic socket programming and network communication, understanding the format and content of HTTP messages.

Experiment 2:

(1) Implement a simple ping program based on UDP to measure Round-Trip Time (RTT) by simulating the exchange of ping and pong messages.

(2) Understand how to write client and server programs using Python to achieve the interaction of ping and pong in UDP communication.

(3) Demonstrate simulating packet loss to replicate potential loss situations in actual networks.

Experiment 3:

(1) Develop a basic TCP-based FTP protocol using Python and the

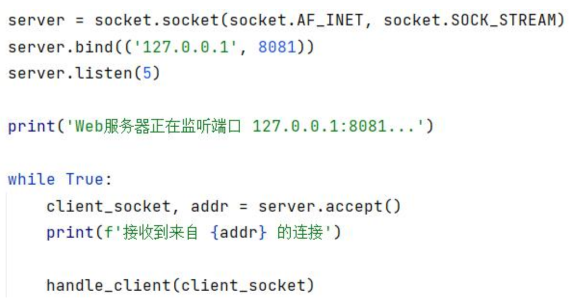
Socket module to achieve simple file transfer functionality.

(2) Demonstrate the fundamental file request and response process between the server and client, including requesting files, checking file existence, and sending files or error messages.

2. Experimental principle

Experiment 1:

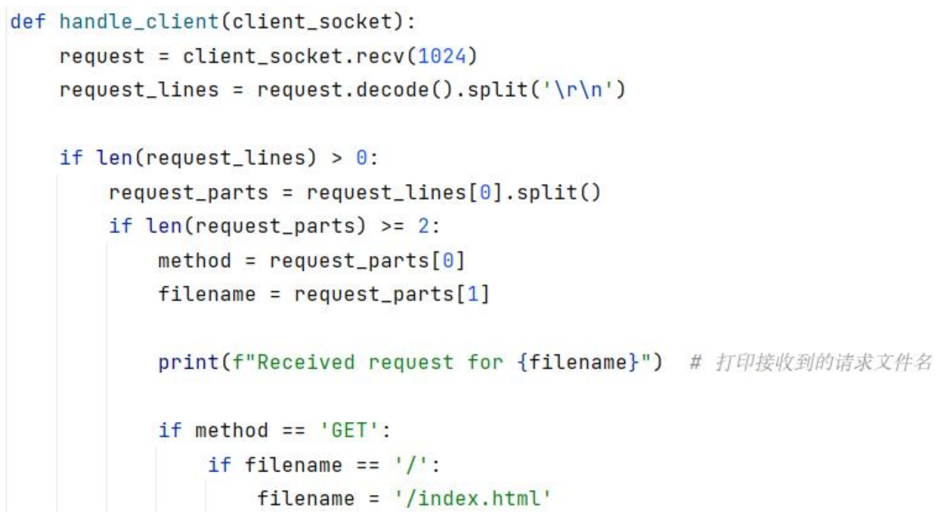
(1) Utilize the socket module in Python to create a basic TCP/IP server, listening on a specific port to accept connections from clients.



```python
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('127.0.0.1', 8081))
server.listen(5)

print('Web服务器正在监听端口 127.0.0.1:8081...')

while True:
    client_socket, addr = server.accept()
    print(f'接收到来自 {addr} 的连接')

    handle_client(client_socket)
```

Figure (1) Web listening

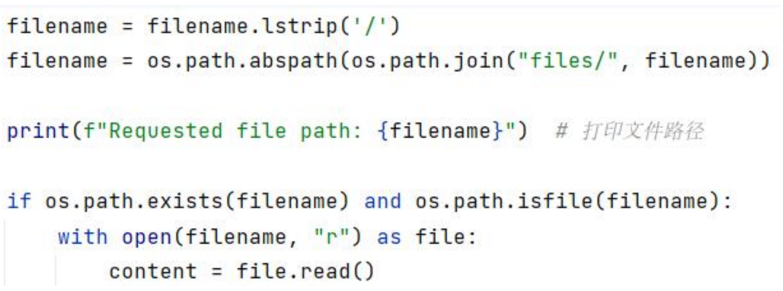
(2) Parse HTTP request messages, primarily GET requests, extracting the requested file name and method.



```python
def handle_client(client_socket):
    request = client_socket.recv(1024)
    request_lines = request.decode().split('\r\n')

    if len(request_lines) > 0:
        request_parts = request_lines[0].split()
        if len(request_parts) >= 2:
            method = request_parts[0]
            filename = request_parts[1]

            print(f"Received request for {filename}")  # 打印接收到的请求文件名

            if method == 'GET':
                if filename == '/':
                    filename = '/index.html'
```

Figure (2) HTTP request messages

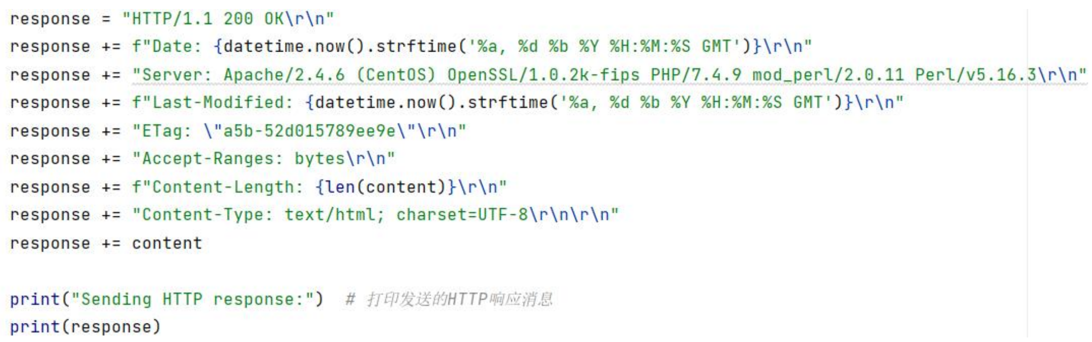
(3) Construct the server file path, check file existence and whether it's a file (not a directory), and read file content.



```
filename = filename.lstrip('/')
filename = os.path.abspath(os.path.join("files/", filename))

print(f"Requested file path: {filename}")  # 打印文件路径

if os.path.exists(filename) and os.path.isfile(filename):
    with open(filename, "r") as file:
        content = file.read()
```

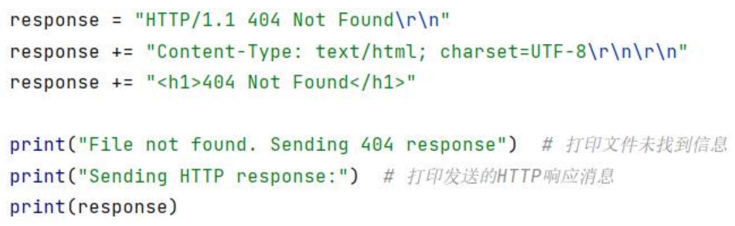
Figure (3) read file content

(4) Based on the request, build an HTTP response message, including a status line, header information, and the actual file content.



```
response = "HTTP/1.1 200 OK\r\n"
response += f"Date: {datetime.now().strftime('%a, %d %b %Y %H:%M:%S GMT')}\r\n"
response += "Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.9 mod_perl/2.0.11 Perl/v5.16.3\r\n"
response += f"Last-Modified: {datetime.now().strftime('%a, %d %b %Y %H:%M:%S GMT')}\r\n"
response += "ETag: \"a5b-52d015789ee9e\"\r\n"
response += "Accept-Ranges: bytes\r\n"
response += f"Content-Length: {len(content)}\r\n"
response += "Content-Type: text/html; charset=UTF-8\r\n\r\n"
response += content

print("Sending HTTP response:")  # 打印发送的HTTP响应消息
print(response)
```

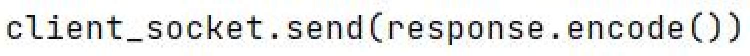
Figure (4): The target file exists and has been successfully read.



```
response = "HTTP/1.1 404 Not Found\r\n"
response += "Content-Type: text/html; charset=UTF-8\r\n\r\n"
response += "<h1>404 Not Found</h1>"

print("File not found. Sending 404 response")  # 打印文件未找到信息
print("Sending HTTP response:")  # 打印发送的HTTP响应消息
print(response)
```

Figure (5): Failure to find the target file results in an error message.

(5) Send the HTTP response message to the client via the socket.

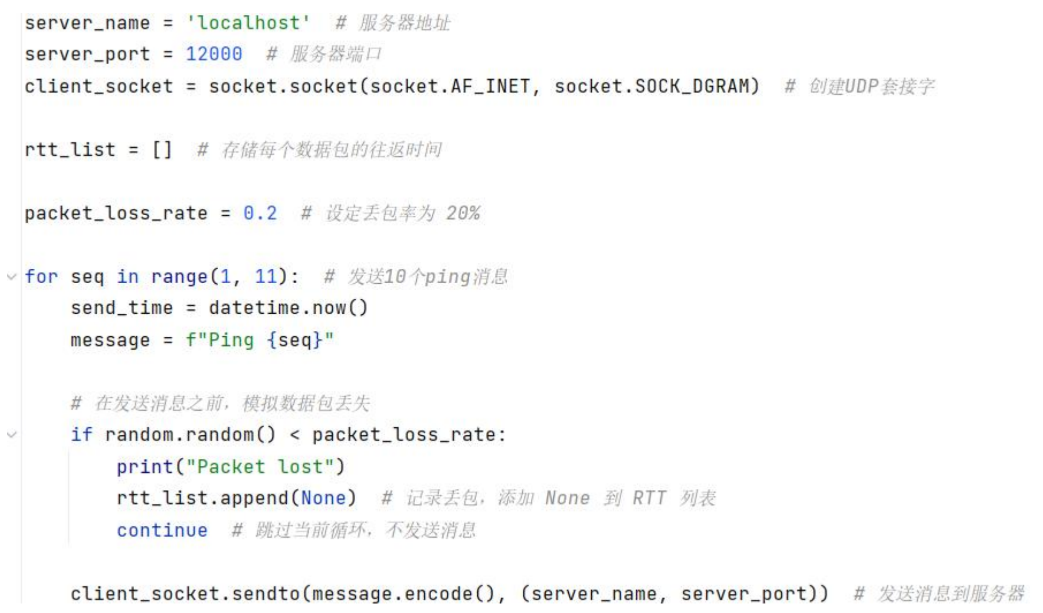


```
client_socket.send(response.encode())
```

Figure (6): Send the HTTP response message.

Experiment 2:

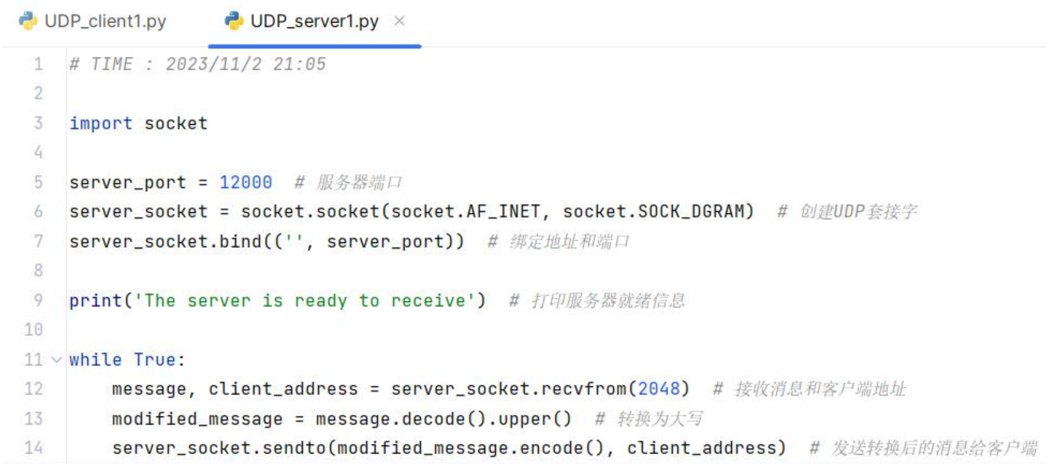
(1) The client uses Python's socket module to create a UDP socket, sending ping messages to the server.



```python
server_name = 'localhost'  # 服务器地址
server_port = 12000  # 服务器端口
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # 创建UDP套接字

rtt_list = []  # 存储每个数据包的往返时间

packet_loss_rate = 0.2  # 设定丢包率为 20%

for seq in range(1, 11):  # 发送10个ping消息
    send_time = datetime.now()
    message = f"Ping {seq}"

    # 在发送消息之前，模拟数据包丢失
    if random.random() < packet_loss_rate:
        print("Packet lost")
        rtt_list.append(None)  # 记录丢包，添加 None 到 RTT 列表
        continue  # 跳过当前循环，不发送消息

    client_socket.sendto(message.encode(), (server_name, server_port))  # 发送消息到服务器
```

Figure (7):client send ping messages to the server.

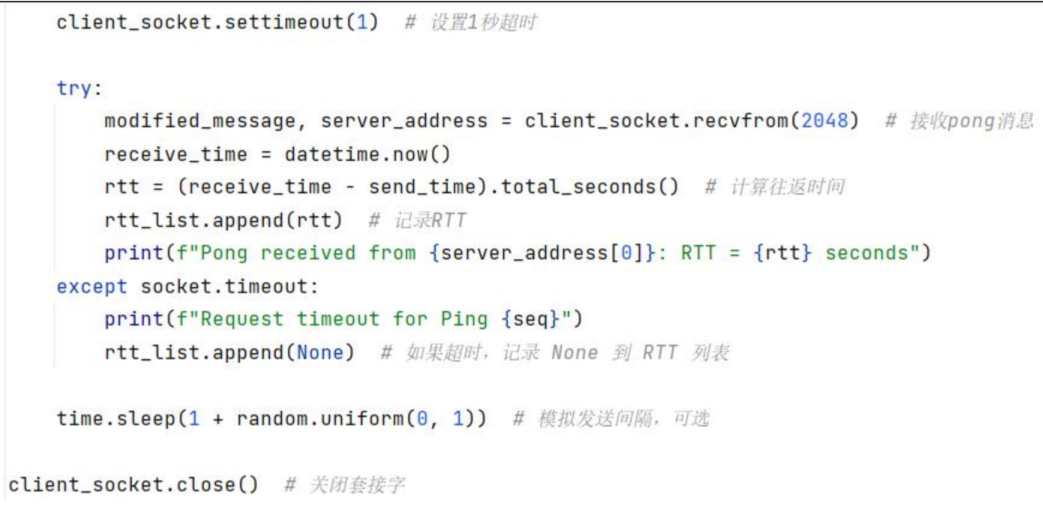
(2) The server receives messages from the client, converts them to uppercase, and sends them back to the client.



```python
# TIME : 2023/11/2 21:05

import socket

server_port = 12000  # 服务器端口
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # 创建UDP套接字
server_socket.bind(('', server_port))  # 绑定地址和端口

print('The server is ready to receive')  # 打印服务器就绪信息

while True:
    message, client_address = server_socket.recvfrom(2048)  # 接收消息和客户端地址
    modified_message = message.decode().upper()  # 转换为大写
    server_socket.sendto(modified_message.encode(), client_address)  # 发送转换后的消息给客户端
```

Figure (8):server receives messages.

(3) The client calculates and prints the Round-Trip Time (RTT) for each received pong message, simulates packet loss, and waits for the server's response within 1 second.



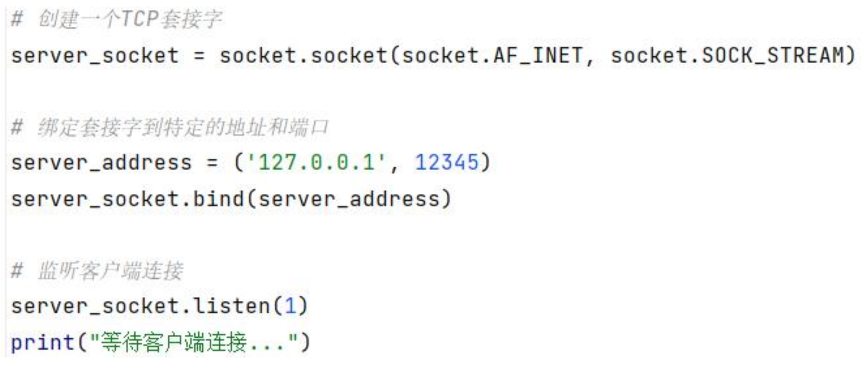
```
    client_socket.settimeout(1)  # 设置1秒超时

    try:
        modified_message, server_address = client_socket.recvfrom(2048)  # 接收pong消息
        receive_time = datetime.now()
        rtt = (receive_time - send_time).total_seconds()  # 计算往返时间
        rtt_list.append(rtt)  # 记录RTT
        print(f"Pong received from {server_address[0]}: RTT = {rtt} seconds")
    except socket.timeout:
        print(f"Request timeout for Ping {seq}")
        rtt_list.append(None)  # 如果超时，记录 None 到 RTT 列表

    time.sleep(1 + random.uniform(0, 1))  # 模拟发送间隔，可选

client_socket.close()  # 关闭套接字
```

Figure (9):Round-Trip Time.

Experiment 3:

Server-Side:

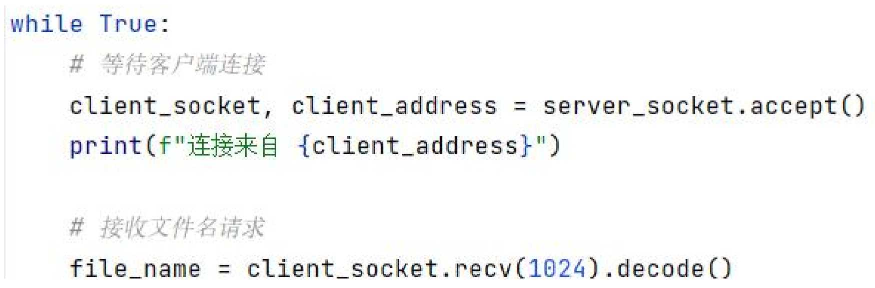
(1) Create a TCP socket, bind it to an address and port, and listen for client connections.



```
# 创建一个TCP套接字
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 绑定套接字到特定的地址和端口
server_address = ('127.0.0.1', 12345)
server_socket.bind(server_address)

# 监听客户端连接
server_socket.listen(1)
print("等待客户端连接...")
```

Figure (10):Create a TCP socket.

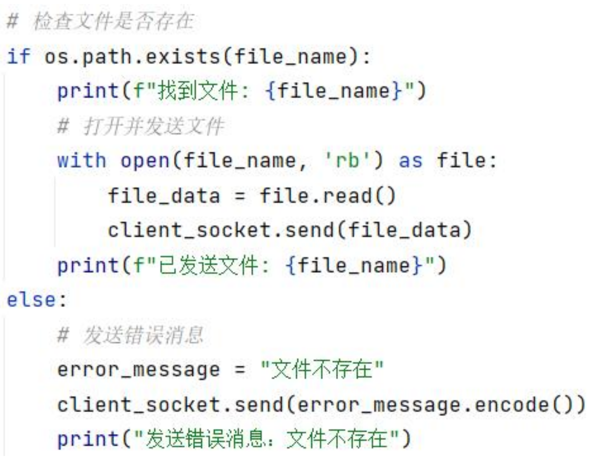
(2) Accept client connection requests and await file name requests.



```
while True:
    # 等待客户端连接
    client_socket, client_address = server_socket.accept()
    print(f"连接来自 {client_address}")

    # 接收文件名请求
    file_name = client_socket.recv(1024).decode()
```

Figure (11):Accept client connection requests.

(3) Check if the requested file exists; if it does, send the file content, otherwise, send an error message.

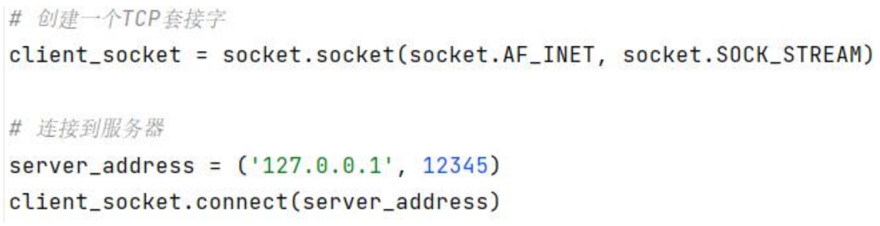


```python
# 检查文件是否存在
if os.path.exists(file_name):
    print(f"找到文件: {file_name}")
    # 打开并发送文件
    with open(file_name, 'rb') as file:
        file_data = file.read()
        client_socket.send(file_data)
    print(f"已发送文件: {file_name}")
else:
    # 发送错误消息
    error_message = "文件不存在"
    client_socket.send(error_message.encode())
    print("发送错误消息：文件不存在")
```

Figure (12):Check if the requested file exists.

Client-Side:

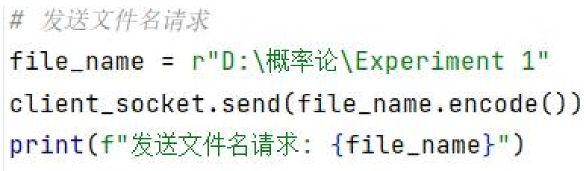
(1) Create a TCP socket and connect to the server.



```python
# 创建一个TCP套接字
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 连接到服务器
server_address = ('127.0.0.1', 12345)
client_socket.connect(server_address)
```

Figure (13):Create a TCP socket.

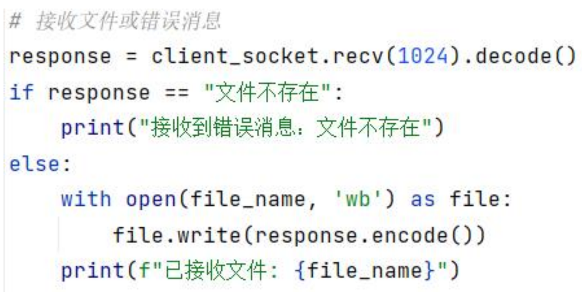
(2) Send a file name request to the server.



```python
# 发送文件名请求
file_name = r"D:\概率论\Experiment 1"
client_socket.send(file_name.encode())
print(f"发送文件名请求: {file_name}")
```

Figure (14):Send a file name request to the server.

(3) Receive the file content or error message from the server and respond accordingly.



```python
# 接收文件或错误消息
response = client_socket.recv(1024).decode()
if response == "文件不存在":
    print("接收到错误消息：文件不存在")
else:
    with open(file_name, 'wb') as file:
        file.write(response.encode())
    print(f"已接收文件: {file_name}")
```

Figure (15):Receive

3. Content

Experiment 1:

(1) Create a TCP socket based on IPv4, bind it to a specific IP address and port, and listen for connection requests from clients.

(2) Upon receiving a connection, receive the HTTP request message from the client.

(3) Parse the request message, extract the requested file name and method, build the file path, check file existence, and read file content.

(4) Construct the appropriate HTTP response message, including a status line, header information, and the actual file content.

(5) Send the HTTP response message to the client via the socket.

(6) For files not found, return an error message "404 Not Found".

In the case of the existence of the target file and successful reading:

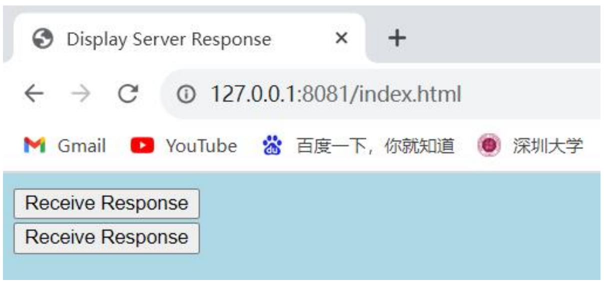
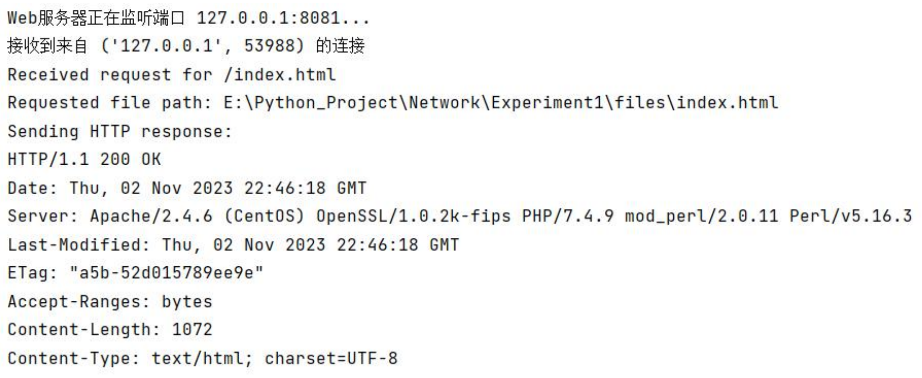
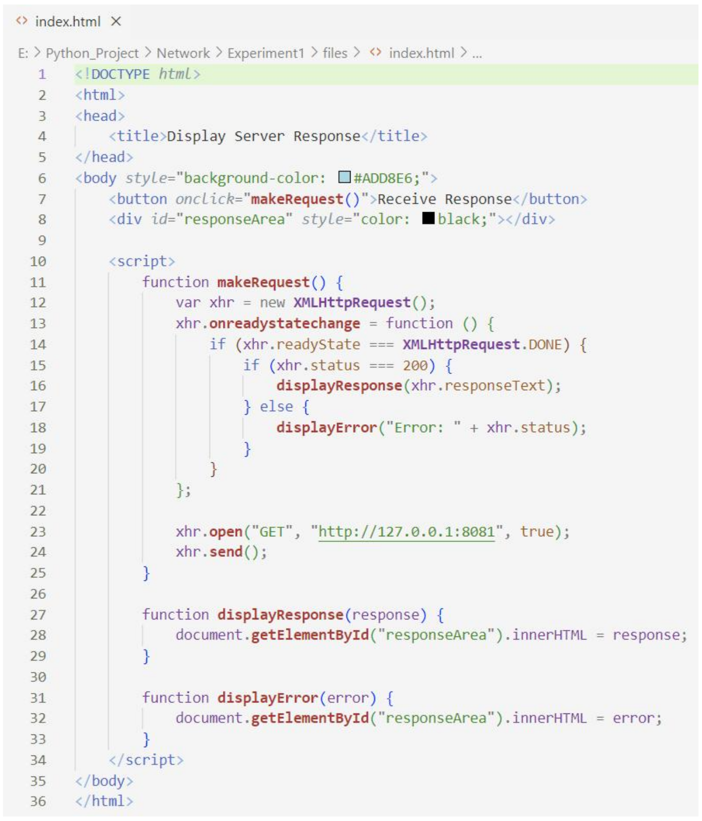


Figure (16): Page Display



```
Web服务器正在监听端口 127.0.0.1:8081...
接收到来自 ('127.0.0.1', 53988) 的连接
Received request for /index.html
Requested file path: E:\Python_Project\Network\Experiment1\files\index.html
Sending HTTP response:
HTTP/1.1 200 OK
Date: Thu, 02 Nov 2023 22:46:18 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/7.4.9 mod_perl/2.0.11 Perl/v5.16.3
Last-Modified: Thu, 02 Nov 2023 22:46:18 GMT
ETag: "a5b-52d015789ee9e"
Accept-Ranges: bytes
Content-Length: 1072
Content-Type: text/html; charset=UTF-8
```

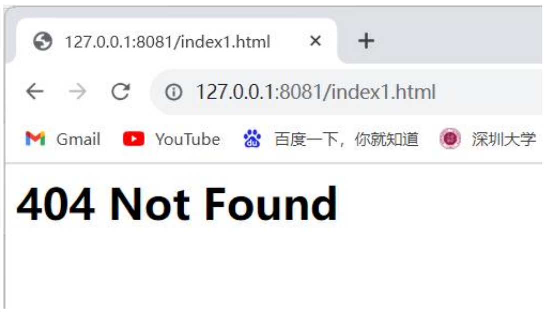
Figure (17): Server-side Response



```html
<!DOCTYPE html>
<html>
<head>
    <title>Display Server Response</title>
</head>
<body style="background-color: #ADD8E6;">
    <button onclick="makeRequest()">Receive Response</button>
    <div id="responseArea" style="color: black;"></div>

    <script>
        function makeRequest() {
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function () {
                if (xhr.readyState === XMLHttpRequest.DONE) {
                    if (xhr.status === 200) {
                        displayResponse(xhr.responseText);
                    } else {
                        displayError("Error: " + xhr.status);
                    }
                }
            };

            xhr.open("GET", "http://127.0.0.1:8081", true);
            xhr.send();
        }

        function displayResponse(response) {
            document.getElementById("responseArea").innerHTML = response;
        }

        function displayError(error) {
            document.getElementById("responseArea").innerHTML = error;
        }
    </script>
</body>
</html>
```

Figure (18): Content within index.html

In the situation where the target file is not found, an error message is returned:



127.0.0.1:8081/index1.html

404 Not Found

Figure (19): Page Display

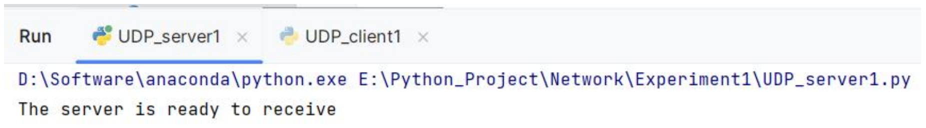
```
接收到来自 ('127.0.0.1', 54311) 的连接
Received request for /index1.html
Requested file path: E:\Python_Project\Network\Experiment1\files\index1.html
File not found. Sending 404 response
Sending HTTP response:
HTTP/1.1 404 Not Found
Content-Type: text/html; charset=UTF-8

<h1>404 Not Found</h1>
Request handled. Waiting for new connections...
```

Figure (20): Server-side Response

Experiment 2:

(1) The client creates a UDP socket and sends 10 ping messages to the server.

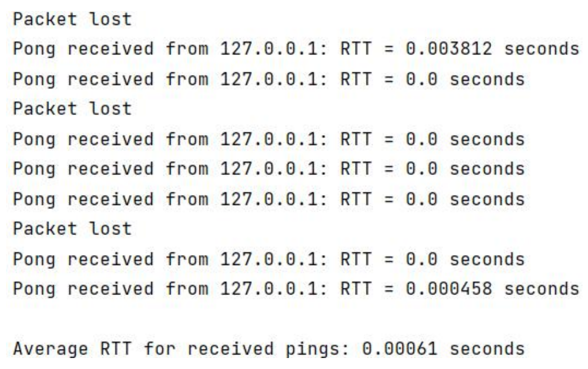
(2) The server receives the ping messages, converts them to uppercase, and returns them to the client.



```
D:\Software\anaconda\python.exe E:\Python_Project\Network\Experiment1\UDP_server1.py
The server is ready to receive
```

Figure (21): The phenomena of server operation.

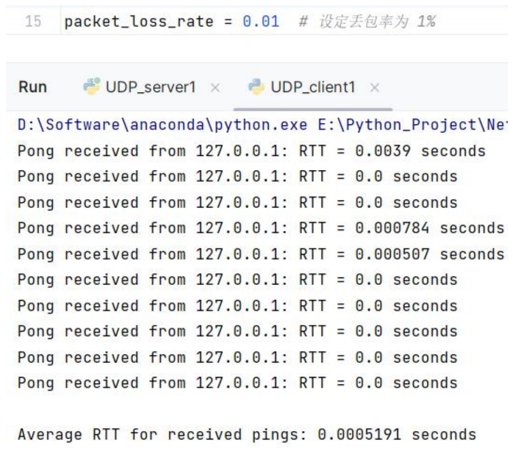
(3) The client receives the server's pong messages, calculates the Round-Trip Time (RTT) for each data packet.



```
Packet lost
Pong received from 127.0.0.1: RTT = 0.003812 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds
Packet lost
Pong received from 127.0.0.1: RTT = 0.0 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds
Packet lost
Pong received from 127.0.0.1: RTT = 0.0 seconds
Pong received from 127.0.0.1: RTT = 0.000458 seconds

Average RTT for received pings: 0.00061 seconds
```

Figure (22): The phenomena of client operation.(Packet loss rate is 20%)

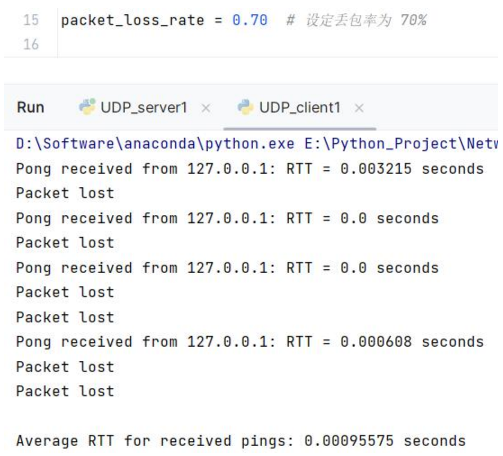
(4) The client simulates packet loss, potentially losing some messages according to the specified loss rate.



```
packet_loss_rate = 0.01  # 设定丢包率为 1%
```

```
D:\Software\anaconda\python.exe E:\Python_Project\Ne
Pong received from 127.0.0.1: RTT = 0.0039 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds
Pong received from 127.0.0.1: RTT = 0.000784 seconds
Pong received from 127.0.0.1: RTT = 0.000507 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds
Pong received from 127.0.0.1: RTT = 0.0 seconds

Average RTT for received pings: 0.0005191 seconds
```

Figure (23): Packet loss rate is 1%.

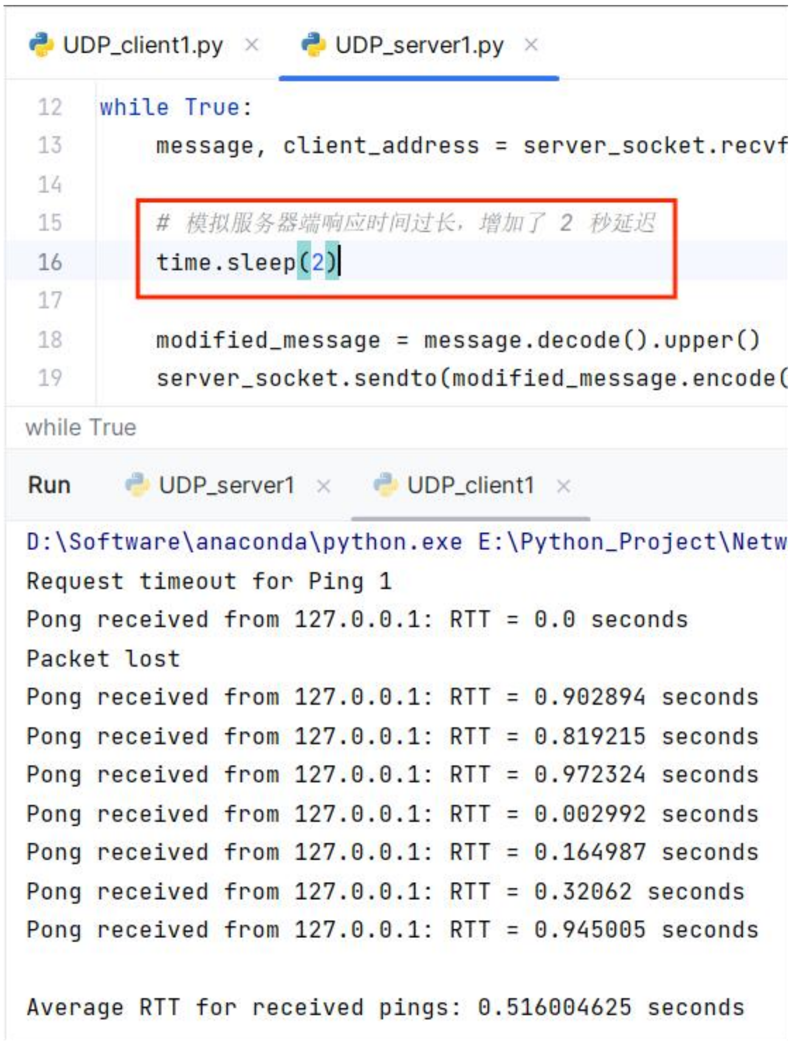


```
packet_loss_rate = 0.70  # 设定丢包率为 70%
```

```
D:\Software\anaconda\python.exe E:\Python_Project\Netw
Pong received from 127.0.0.1: RTT = 0.003215 seconds
Packet lost
Pong received from 127.0.0.1: RTT = 0.0 seconds
Packet lost
Pong received from 127.0.0.1: RTT = 0.0 seconds
Packet lost
Packet lost
Pong received from 127.0.0.1: RTT = 0.000608 seconds
Packet lost
Packet lost

Average RTT for received pings: 0.00095575 seconds
```

Figure (24): Packet loss rate is 70%.

(5) The client logs the loss information and prints "Request Timed Out" if there is no response from the server within 1 second.



```
while True:
    message, client_address = server_socket.recvf

    # 模拟服务器端响应时间过长，增加了 2 秒延迟
    time.sleep(2)

    modified_message = message.decode().upper()
    server_socket.sendto(modified_message.encode(
```

```
D:\Software\anaconda\python.exe E:\Python_Project\Netw
Request timeout for Ping 1
Pong received from 127.0.0.1: RTT = 0.0 seconds
Packet lost
Pong received from 127.0.0.1: RTT = 0.902894 seconds
Pong received from 127.0.0.1: RTT = 0.819215 seconds
Pong received from 127.0.0.1: RTT = 0.972324 seconds
Pong received from 127.0.0.1: RTT = 0.002992 seconds
Pong received from 127.0.0.1: RTT = 0.164987 seconds
Pong received from 127.0.0.1: RTT = 0.32062 seconds
Pong received from 127.0.0.1: RTT = 0.945005 seconds

Average RTT for received pings: 0.516004625 seconds
```

Figure (25): The server introduces a two-second delay.

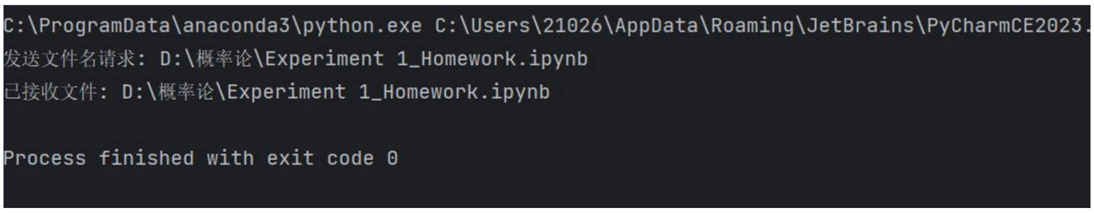
Experiment 3:

(1) Establish a TCP socket and bind/connect to the server on the specified address and port.

(2) For the server, process incoming file requests, checking for file existence, and respond with the file content or an error message.

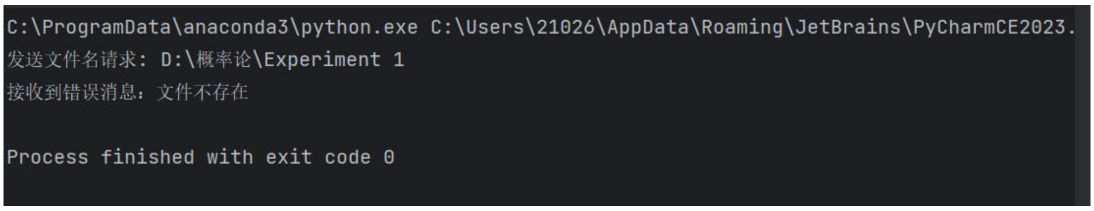
(3) On the client side, send the file name request, receive and handle the file content or an error message accordingly.

(4) Maintain a continuous connection on the server side to listen for incoming requests. Close the client connection after the transaction.



```
C:\ProgramData\anaconda3\python.exe C:\Users\21026\AppData\Roaming\JetBrains\PyCharmCE2023.
发送文件名请求: D:\概率论\Experiment 1_Homework.ipynb
已接收文件: D:\概率论\Experiment 1_Homework.ipynb

Process finished with exit code 0
```

Figure (26): When the file exists.



```
C:\ProgramData\anaconda3\python.exe C:\Users\21026\AppData\Roaming\JetBrains\PyCharmCE2023.
发送文件名请求: D:\概率论\Experiment 1
接收到错误消息: 文件不存在

Process finished with exit code 0
```

Figure (27): When the file doesn't exist.

4. Conclusion and discussion

Experiment 1:

(1) This experiment demonstrates the implementation of a basic Python web server capable of handling simple HTTP GET requests.

(2) Through simple file reading and processing of HTTP messages, the experiment illustrates how the server responds to requests and can return the corresponding content or error messages.

Experiment 2:

(1) The experiment demonstrated a UDP-based ping program, showcasing the calculation of Round-Trip Time (RTT) for ping and pong messages.

(2) It can simulate packet loss, causing some messages to not receive a response, simulating packet loss scenarios in actual networks.

(3) Because UDP is a connectionless protocol without a confirmation mechanism, it cannot ensure the reliable transmission of data packets.

Experiment 3:

This simple FTP protocol example demonstrates basic file transfer functionality based on TCP, exhibiting basic interaction between the server and client.

指导教师批阅意见：

成绩评定：

指导教师签字：

年　　月　　日

备注：