**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.   1. **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.     *Figure (1) Web listening*   1. Parse HTTP request messages, primarily GET requests, extracting the requested file name and method.     *Figure (2) HTTP request messages*   1. Construct the server file path, check file existence and whether it's a file (not a directory), and read file content.     *Figure (3) read file content*   1. Based on the request, build an HTTP response message, including a status line, header information, and the actual file content.     *Figure (4): The target file exists and has been successfully read.*    Figure (5): Failure to find the target file results in an error message.   1. Send the HTTP response message to the client via the socket.     *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.     *Figure (7):client send ping messages to the server.*   1. The server receives messages from the client, converts them to uppercase, and sends them back to the client.     *Figure (8):server receives messages.*   1. 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.     *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.     *Figure (10):Create a TCP socket.*   1. Accept client connection requests and await file name requests.     *Figure (11):Accept client connection requests.*   1. Check if the requested file exists; if it does, send the file content, otherwise, send an error message.     *Figure (12):Check if the requested file exists.*  Client-Side:   1. Create a TCP socket and connect to the server.     *Figure (13):Create a TCP socket.*   1. Send a file name request to the server.     *Figure (14):Send a file name request to the server.*   1. Receive the file content or error message from the server and respond accordingly.     *Figure (15):Receive*   1. **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*    *Figure (17): Server-side Response*    *Figure (18): Content within index.html*  In the situation where the target file is not found, an error message is returned:    *Figure (19): Page Display*    *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.    *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.    *Figure (22): The phenomena of client operation.(Packet loss rate is 20%)*   1. The client simulates packet loss, potentially losing some messages according to the specified loss rate.     *Figure (23): Packet loss rate is 1%.*    *Figure (24): Packet loss rate is 70%.*   1. The client logs the loss information and prints "Request Timed Out" if there is no response from the server within 1 second.     *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.     *Figure (26): When the file exists.*    *Figure (27): When the file doesn't exist.* |
| 1. **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： |