Assignment: Implement TCP Handshake in Python3 Using Raw Sockets

Objective:

Demonstrate your understanding of the TCP protocol by implementing the TCP 3-way handshake process using Python and raw sockets. Your implementation should establish a connection between a client and a server without relying on Python's high-level networking libraries like socket.connect() or socket.listen().

Assignment Overview:

1. Create a TCP-like Server:
   - Use raw sockets to listen for incoming SYN packets.
   - Respond with a SYN-ACK packet when a SYN packet is received.
   - Wait for an ACK from the client to complete the handshake.

2. Create a TCP-like Client:
   - Use raw sockets to send a SYN packet to the server.
   - Wait for a SYN-ACK response from the server.
   - Send an ACK packet to complete the handshake.

3. Deliverables:
   - Separate Python scripts for the client and server.
   - Handshake debug logs showing each step of the process.
   - An explanation of your code (in comments or a markdown file).

Requirements:

1. Packet Construction:

- Build TCP packets manually, including crafting the headers (source port, destination port, sequence number, acknowledgment number, flags, etc.).

- Use checksum calculations to ensure the integrity of packets.

2. Raw Sockets:

- Use Python's socket library to create raw sockets.

- Ensure your script has the necessary permissions (raw sockets typically require administrative privileges).

3. 3-Way Handshake Flow:

- SYN: Client sends a TCP packet with the SYN flag set.

- SYN-ACK: Server responds with a TCP packet with both SYN and ACK flags set.

- ACK: Client sends a TCP packet with the ACK flag set to finalize the handshake.

4. Error Handling:

- Handle unexpected packets or connection failures gracefully.

Hints and Resources:

1. Raw Sockets Setup:

- Use socket.AF_INET and socket.SOCK_RAW to create raw sockets.

- You may need to work with the IP and TCP headers separately.

2. Header Construction:

- Refer to RFC 793 for details about the TCP header structure.

- Use the struct module in Python to pack and unpack binary data.

3. Checksum Calculation:

- Implement the checksum algorithm described in the TCP RFC. Remember, the checksum includes the pseudo-header, TCP header, and data.

4. Debugging:

- Use tools like Wireshark to inspect your packets and verify their correctness.

Expected Debug Logs:

Logs should clearly demonstrate each step of the handshake:

[Client] Sending SYN to 192.168.1.1:5000

[Server] Received SYN, sending SYN-ACK

[Client] Received SYN-ACK, sending ACK

[Server] Received ACK, handshake completed

Evaluation Criteria:

1. Functionality:

- Does the handshake work as expected between client and server?
- Are the packets correctly crafted and validated?

2. Adherence to TCP Standards:

- Does your implementation follow the specifications outlined in RFC 793?

3. Code Quality:

- Is your code clean, well-commented, and modular?

4. Debugging and Testing:

- Are your debug logs informative?
- Did you test the implementation thoroughly?

Stretch Goals:

- Implement connection teardown (FIN-ACK).

- Add support for retransmissions if a packet is lost.

- Simulate multiple simultaneous connections.

This assignment will help you solidify your understanding of TCP and the intricacies of protocol design. Good luck!