TCP communication

This documentation provides a step-by-step guide to setting up TCP communication between server and client using python.

TCP (Transmission Control Protocol) is an important network protocol that lets two hosts connect and exchange data streams. TCP guarantees the delivery of data and packets in the same order as they were sent.

TCP's role is to ensure the packets are reliably delivered, and error-free. TCP implements congestion control, which means the initial requests start small, increasing in size to the levels of bandwidth the computers, servers, and network can support.

Prerequisites

- Python installed on both machines.
- Both machines must be on the same network.
- Knowledge of IP addresses and basic Python programming.

Server Code
The server will bind to a IP address and port, listen for incoming connections, and acknowledge each message received. (In this case we are using localhost)

Server.py

```python
import socket

def start_server(host="127.0.0.1", port=65433):
    """
    Function to start a TCP server.

    Parameters:
    host (str): The IP address the server listens on. Default is localhost.
    port (int): The port the server listens on. Default is 65433.
    """

    # Create a TCP/IP socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Bind the socket to the address and port
    server_socket.bind((host, port))
```

```
    # Enable the server to accept connections (max backlog of connections is
1)
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}")

    while True:
        print("Waiting for a connection")

        # Wait for a connection
        conn, addr = server_socket.accept()

        try:
            print(f"Connection from {addr}")

            while True:
                # Receive data from the client in chunks of 1024 bytes
                data = conn.recv(1024)

                if data:
                    # Decode and print the received data
                    print(f"Received {data.decode()} from {addr}")

                    # Send acknowledgment to the client
                    acknowledgment = "Message received"
                    conn.sendall(acknowledgment.encode())
                else:
                    # Break the loop if no more data is received
                    break

        finally:
            # Clean up the connection
            conn.close()
            print(f"Connection from {addr} closed")

if __name__ == "__main__":
    # Start the server with default parameters
    start_server()
```

Client Code

The client will connect to the server's IP address and port, send messages, and display acknowledgments from the server.

```
import socket

def start_client(host='127.0.0.1', port=65433):
    """
    Function to start a TCP client.

    Parameters:
    host (str): The IP address of the server to connect to. Default is
localhost.
    port (int): The port on which the server is listening. Default is 65433.
    """

    # Create a TCP/IP socket
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect the socket to the server's port
    client_socket.connect((host, port))
    print(f"Connected to server {host}:{port}")

    try:
        while True:
            # Prompt user for a message to send to the server
            message = input("Enter message to send (type 'exit' to close): ")

            # Exit the loop if the user types 'exit'
            if message.lower() == 'exit':
                break

            # Send the message to the server
            client_socket.sendall(message.encode())

            # Receive the response from the server
            data = client_socket.recv(1024)
            print(f"Received {data.decode()} from server")

    finally:
        # Close the connection to the server
        client_socket.close()
        print("Connection closed")

if __name__ == "__main__":
    # Start the client with default parameters
    start_client()
```

Explanation

Server:

- The server binds to its IP address and a specific port, listens for incoming connections, and waits for data from the client.
- Upon receiving data, it prints the message and sends an acknowledgment back to the client.
- This continues until the client disconnects.

Client:

- The client connects to the server at the specified IP address and port.
- It sends user input to the server and waits for the acknowledgment response.
- The interaction continues until the user types "exit".

Thank you