Project report of Computer Networks

Instant Messaging System

Group members:

Uzair Rehman (2020509)
Atif Khan (2020092)
Zaeem Shakir (2020487)

Submitted to:

Lecturer Salman Saeed

Introduction

The instant messaging system is a fundamental tool for real-time communication over the internet. This report presents the design and implementation of an instant messaging system comprising both server and client components. The system facilitates text-based communication between multiple users and supports real-time message exchange.

Implementation

System Architecture

The instant messaging system follows a client-server architecture. The server component manages message transfers and client connections, while the client component interacts with the server to send and receive messages.

Design

The system is designed to support real-time communication between two users. It utilizes sockets for network communication and threading for concurrent message handling. The server maintains a dictionary of connected clients and forwards messages between clients as required.

Components

- **Server Component (Server.py)**: Handles client connections, message routing, and communication with clients.
- **Client Component (Client.py)**: Provides a graphical user interface (GUI) for users to send and receive messages.

Workflow

1. The server starts listening for incoming connections on a specified host and port.
2. Clients connect to the server and can send messages to each other through the server.
3. The server receives messages from clients, determines the recipients, and forwards the messages accordingly.
4. Clients receive messages from the server and display them in the user interface.

Deployment Instructions

- Ensure Python is installed on your system.
- Run the Server.py script to start the server.
- Run the Client.py script to start the client with the Tkinter GUI.
- Enter messages in the GUI and click the send button to send messages.

Conclusion

The implemented instant messaging system provides a basic framework for real-time communication between multiple users. While the system meets the requirements outlined in the project statement, there is room for further enhancement and optimization, such as implementing user authentication, improving message reliability, and adding support for additional features like file sharing.

Appendix: Source Code Files

Server.py

```
import socket
import threading

# Global variables
clients = {}  # Dictionary to store client connections

def handle_client(client_socket, client_address):
    try:
        while True:
            message = client_socket.recv(4096).decode('utf-8')
            if message:
                print(f"Received message from {client_address}: {message}")
                send_to_other_clients(client_socket, message)
            else:
                print(f"Client {client_address} disconnected.")
                del clients[client_socket]
                break
    except Exception as e:
        print(f"Error handling client {client_address}: {e}")
        del clients[client_socket]
        client_socket.close()

def send_to_other_clients(sender_socket, message):
    for client_socket in clients:
        if client_socket != sender_socket:
            try:
                client_socket.send(message.encode('utf-8'))
            except Exception as e:
                print(f"Error sending message to client: {e}")

def start_server(host, port):
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((host, port))
    server.listen(5)
    print("Server listening on", host, "port", port)

    while True:
        client_socket, client_address = server.accept()
        print("Accepted connection from", client_address)
        clients[client_socket] = client_address

        # Start a new thread to handle the client
        client_thread = threading.Thread(target=handle_client,
args=(client_socket, client_address))
        client_thread.start()

if __name__ == "__main__":
    HOST = '127.0.0.1'  # Server IP address
    PORT = 8080  # Server port
    start_server(HOST, PORT)
```

Client.py

```
import socket
import threading
import tkinter as tk
class ClientUI:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.root = tk.Tk()
        self.root.title("Instant Messaging Client")
        self.message_frame = tk.Frame(self.root)
        self.scrollbar = tk.Scrollbar(self.message_frame)
        self.scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.message_list = tk.Listbox(self.message_frame, height=15,
width=50, yscrollcommand=self.scrollbar.set)
        self.message_list.pack(side=tk.LEFT, fill=tk.BOTH)
        self.message_frame.pack()
        self.message_entry = tk.Entry(self.root, width=50)
        self.message_entry.pack()
        self.send_button = tk.Button(self.root, text="Send",
command=self.send_message)
        self.send_button.pack()
        # Connect to server
        self.client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.client_socket.connect((self.host, self.port))
        # Start receiving messages in a separate thread
        receive_thread = threading.Thread(target=self.receive_messages)
        receive_thread.daemon = True
        receive_thread.start()
    def send_message(self):
        message = self.message_entry.get()
        if message:
            self.client_socket.send(message.encode('utf-8'))
            self.message_entry.delete(0, tk.END)
    def receive_messages(self):
        while True:
            try:
                message = self.client_socket.recv(4096).decode('utf-8')
                if message:
                    self.message_list.insert(tk.END, message)
                    self.message_list.yview(tk.END)
            except Exception as e:
                print(f"Error receiving message: {e}")
                break

    def run(self):
        self.root.mainloop()
if __name__ == "__main__":
    HOST = '127.0.0.1'  # Server IP address
    PORT = 8080  # Server port
    client_ui = ClientUI(HOST, PORT)
    client_ui.run()
```