CIS505 Project 3
Team: Code Damn It!
Shruthi Ashok Kumar
Adrian Diaz
Abhishek Ravichandran

Protocol and Design Specifications

- **Protocols**
  - UDP Protocol: We plan to build off of the UDP implementation in Project 2
  - Total Ordering Protocol:
    - The leader maintains a sequence number counter. Every time it is sent a message it assigns that message's sequence number to the counter and increments the counter. It then sends the multicast to all participants with the sequence number
    - When a participant joins, the leader sends them the current sequence number. This helps the participant realise that some packets might have been dropped.
    - When a participant receives a message from the leader's multicast, it only displays the message to the client if the sequence number is one more than the current sequence number. If not, it puts it in a hold back queue
    - Dealing with delayed or lost messages:
      - When a message is placed in the hold back queue, after 5 seconds if the hold back queue is not empty, it will send a message to the leader requesting the message of the lost sequence number
      - If the leader does not respond within 5 seconds, it determines the leader has failed and holds an election
      - If leader is still active, it will respond with the message
    - When the leader receives ack's from all participants. It can remove the message from its message array. Until then, it keeps the messages in case a message was lost and it was not received by a participant
    - Dealing with delayed or lost ack:
      - We will use the same time out protocol to determine whether a participant has failed. If after 5 seconds, the leader does not receive an ack/request for the lost packet, it will send a message to the participant

            - If the leader does not hear back from the participant in 5 seconds, it will notify all users that the participant has left that chat
    - Leader Election Protocol
        - All the users are aware of the IP addresses and port numbers of the other users currently in the chat.
        - After 60 seconds of inactivity (i.e. participant has not received a message or notification from the leader), it will send a message to the leader to make sure the leader is awake
        - When a participant detects that a leader has failed via a lost message or a lack of acknowledgements to “are you awake” messages, it decides to hold an election. We plan to follow the “Bully Algorithm” protocol described in class.
- **Message Types:** all message types should have a unique identifier describing the type of message. For example, a user joined message will have the identifier “joined” and this will be the first line in the payload
    - New participant joins a chat (this could be a message to any participant):
        - Identifier : “joined”
        - User name
        - IP Address
        - Port Number
    - Reply to Join
        - Identifier: “reply-join”
        - Leader user name
        - Leader IP address
        - Leader port number
    - Notify leader you have joined
        - Identifier: “add me”
        - User name
        - IP Address
        - Port number
    - Leader replies to participant that just joined
        - Identifier: “welcome”
        - Current sequence number
    - Left chat (participant sends this to the leader)
        - Identifier : “left”
        - User name
        - IP Address
        - Port Number

    - Participant sends message to leader
        - Identifier: “message”
        - User name
        - IP Address of sender
        - Port Number of sender
        - Message
    - Leader sends message to participant (note this could be a message or a notice that someone has joined/left)
        - Identifier: “multicast”
        - User name
        - IP Address of sender
        - Port Number of sender
        - Message
    - Participant acknowledges receipt of message
        - Identifier: “message ack”
        - Sequence number
    - Leader enquires about a delayed or lost ack
        - Identifier: “ack not received”
        - Sequence number
    - Participant requests “lost” message
        - Identifier: “lost message”
        - Sequence number: the sequence number of the missing message
    - Send bully message that participant has higher id
        - Identifier: “Back off”
    - Leader notifies all other participants that they are the leader
        - Identifier: “I am the leader”
        - User name of leader
        - IP Address of leader
        - Port number
        - Is_leader flag set to 1
    - Participant checks if leader is awake
        - Identifier:”are you awake”
    - Leader responds to awake check
        - Identifier: “yes”
- **Data Structures**
    - User struct:
        - Name
        - IP address
        - Port

        - Is_leader flag
    - Array of all user structs also in chat: each client will maintain this list and the first element of the array is the data associated with the leader.
    - Message struct
        - Is_notification? Flag to determine if message is a message from user or notification (e.g. someone joining or leaving)
        - User struct (will be nil if notification)
        - Message string
        - Message sequence number for total ordering
    - Hold back queue which is an array of undelivered message structs
    - Array of message structs: this is maintained by the leader for all messages that have not been acknowledged by all participants
- **Division of Labor**
    - Implementation of data structures and overall code organization: Abhishek
    - Implementation of Total Ordering : Shruthi
    - Implementation of Leader Election : Adrian