CSC 328 Final Group Project:
Network Program Design

Dr. Dylan Schwesinger
R-E Miller, Elliot Swan, & Matthew Hill

November 21st, 2023

Table of Contents

1) Project Members....3
2) Communication Plan....3
3) Tasks involved....3
4) Programming Language....4
5) Project Requirements/Scope....4
6) Application I/O....5
7) Common Functionalities....6
8) Application Protocols....7
9) Sequence Diagram (Found on page 9)....8
10) Test Plan....10

1) Project Members

R-E Miller - Team Lead - Client Code
Elliot Swan - Library Code
Matthew Hill - Server Code

---

2) Communication Plan

Communication will happen through a combination of in person meetings at the library and online meetings held virtually through discord. The code will be managed and shared on our GitHub repository. (https://github.com/R-E-Miller/CSC328-Final-Project). We plan to meet weekly at least, or more if necessary.

---

3) Tasks involved

Our project involves a plan in which a specific task i assigned to each team member. We have divided the project into key areas - Server, Client, and Library code portions, with each area having its own set of tasks and sub-tasks.

- Server Tasks (Matthew Hill):
    - Creating the Server: This involves setting up the server infrastructure, ensuring it can handle multiple client connections. Estimated time: [X hours/days].
    - Name Verification: Implementing functionality to verify and manage unique client nicknames. Estimated time: [Y hours/days].
    - Handling Multiple Connections: Ensuring the server can efficiently manage simultaneous client connections. Estimated time: [Z hours/days].
    - Graceful Shutdown: Developing a mechanism for the server to shut down gracefully, including sending notifications to all connected clients. Estimated time: [W hours/days].
- Client Tasks (R-E Miller):
    - Connecting to Server: Establishing a reliable connection to the server. Estimated time: [A hours/days].
    - Nickname Decision: Enabling clients to choose and confirm a unique nickname. Estimated time: [B hours/days].
    - Message Sending and Receiving: Facilitating the sending and receiving of messages through the client interface. Estimated time: [C hours/days].
    - Disconnecting: Implementing a smooth and user-friendly disconnection process. Estimated time: [D hours/days].
- Library Code/Curses Integration (Elliot Swan):

- Encoding/Decoding JSON Packets: Developing methods to encode and decode messages in JSON format for efficient communication. Estimated time: [E hours/days].
- Sending/Receiving JSON Packets: Establishing protocols for message transmission and reception. Estimated time: [F hours/days].
- UI Research and Design in Curses: Designing and implementing the user interface using the curses library, ensuring a seamless user experience. Estimated time: [G hours/days].

Side Note - As a group, we will keep track of time as we go, as an actual completion time is submitted at the end of the project. These figures are just estimates.

---

4) Programming Language

In our project, we have thoroughly evaluated the programming language requirements for each component - the client, server, and the shared library code. After some thought and meeting as a group, we have decided to use Python for all aspects of our project. This decision is based on Python's versatility, its wide range of libraries, and ease of integration across different components.

- Client Side: Python will be used to develop the client interface. Its rich set of libraries, especially for network communication and user interface (like curses for terminal-based interfaces), makes it an ideal choice.
- Server Side: The server component will also be implemented in Python. This will enable us to leverage Python's robust networking capabilities and its efficient handling of concurrent connections.
- Library Code: For the shared library, which includes functionalities common to both the client and server (like JSON packet encoding/decoding), Python is again our language of choice. This ensures seamless integration and function calling between the client and server components.

---

5) Project Requirements/Scope

1. Client Requirements:

- Nickname Selection: Clients have the ability to choose a unique nickname. The system will check for nickname availability to avoid duplicates.
- User Interface: A clean and user-friendly interface. This includes clear display of messages and easy navigation.

- Message Log Access: Clients can request access to older messages, potentially through a command like /logs, enhancing the chat experience by allowing users to catch up on missed conversations.
- Seamless Exit: Clients can smoothly exit the chat program either through a specific command or by using a keyboard interrupt/shortcut like ‘CTRL-C’.

2. Optional Features:
   - Clear Messages: Clients may have the option to clear their chat screen using a command like /clear.
   - User Lists: Display a list of active users in the chat.
   - Potential Direct Messaging Capability for clients to send private messages to each other.
   - Potential Moderation Capabilities; The server operator might have the ability to moderate the chat, with functionalities like muting or removing users.
3. Server Requirements:
   - Chat Log Storage: The server will maintain a log of all chat conversations in a text file, ensuring a record of all interactions.
   - Nickname Verification: Ability to verify and manage unique client nicknames in a non-blocking manner to ensure smooth operation.
   - Graceful Shutdown: The server can shut down gracefully, informing clients of the impending shutdown and closing existing connections without data loss.
   - Handling Multiple Messages: Capability to handle and process multiple messages concurrently, ensuring real-time communication.
   - Client Capacity: Initially, the server will handle a maximum of 10 clients simultaneously, with the potential to scale up in the future.

These requirements are established based on the application specifications and are designed to ensure a functional chatroom application. Each feature has been considered for its impact on user experience with optional features included as potential enhancements. The project will focus on meeting these core requirements first, while keeping in mind the possibility of integrating the optional features based on time and resource availability.

---

6) Application I/O

Client Program User Inputs:

- Text Input through Curses Module: Users will be able to type messages and other inputs into a text box within the curses module interface. This includes standard chat messages and potential commands.

- Command Execution: Users can execute specific commands using a "/command" syntax. This feature might restrict certain text combinations to avoid command conflicts.
- Special Key Combinations: Additional functionalities, like exiting the chat, or accessing help menus, can be triggered using key combinations like ‘CTRL-C’.
- Interactive Features: The interface may include additional interactive elements like accessing a user list or enabling direct messaging, depending on development progress and feasibility.

Client Program Outputs to the User:

- Curses Module Interface: The client program will display a curses-based user interface, offering an intuitive and clear layout for chat interactions.
- Dedicated Input Area: At the bottom of the interface, there will be a space allocated for typing messages and commands.
- Message Display Area: Above the input space, messages from other users, system notifications, and other relevant information will be displayed.

Server Program User Inputs (Potentially for Moderation):

- Administrative Commands: If a moderation feature is implemented, server administrators may have commands to manage user activities, like printing user nicknames or disconnecting specific users.
- Moderation Controls: Depending on the final feature set, additional moderation controls may be available to manage the chat environment effectively.

Server Program Outputs to User:

- Connection Notifications: The server will output notifications related to user connections and disconnections, which could be directed to a console or a log file.
- Broadcast Messages: In certain scenarios, such as a server shutdown or important announcements, the server might output broadcast messages to all connected clients.
- Administrative Feedback: The server will provide feedback on the execution of administrative commands, aiding in effective chatroom management.

---

7) Common Functionalities

1. Message Encoding and Decoding:
    - Functionality: A set of functions to encode messages into a standardized format before sending and to decode them upon receipt.
    - Purpose: Ensures consistent message formatting across the network, facilitating clear and reliable communication between clients and the server.
2. Message Formatting:
    - Functionality: Utilities to format messages for display. This includes structuring the message content, timestamp, and sender information.

- Purpose: Provides a uniform appearance for messages on both client and server interfaces, enhancing readability and user experience.

3. Authentication Mechanism:
   - Functionality: A system for authenticating user identities, possibly through username validation.
   - Purpose: To ensure that each user has a unique identifier in the chat, preventing impersonation and maintaining chat integrity.
4. Common Communication Protocols:
   - Functionality: Standard protocols for sending and receiving messages, ensuring compatibility and smooth communication between the client and server.
   - Purpose: To streamline the process of message transmission and reception across the network, reducing the likelihood of errors and miscommunications.
5. User and Message Management:
   - Functionality: Functions to manage user data and message history, potentially including features for retrieving past messages or user lists.
   - Purpose: To provide both the client and server with the necessary tools to manage chat history and user information effectively.

---

8) Application Protocols

Core Communication Method:

- The central element of our communication protocol involves the use of JSON packets. This format is chosen for its simplicity, readability, and wide support across Python.

JSON Packet Structure:

- Each packet will primarily contain two fields:
  - NICK: Represents the user's nickname.
  - MSG: Contains the actual message text.

Client-to-Server Communication:

Initial Connection:

- Upon connecting, the client awaits a confirmation message from the server.
- The client then enters a loop for nickname confirmation.
- Initially, the NICK field is set to None until the server approves a selected nickname.

Nickname Approval:

- The client sends a connection message with the chosen nickname.
- The server responds by either approving the nickname (NICK is set) or requesting a different one.

Message Exchange:

- Regular communication involves sending JSON packets structured as described.
- The client uses the received NICK field to verify its messages were sent.
- If a message contains the /exit command, the client initiates a termination sequence.

Server-to-Client Communication:

Confirmation of Connection:

- The server sends an acknowledgment message upon a client's successful connection.

Nickname Verification:

- The server is responsible for verifying the uniqueness of a user's nickname.
- It communicates approval or rejection of the nickname back to the client.

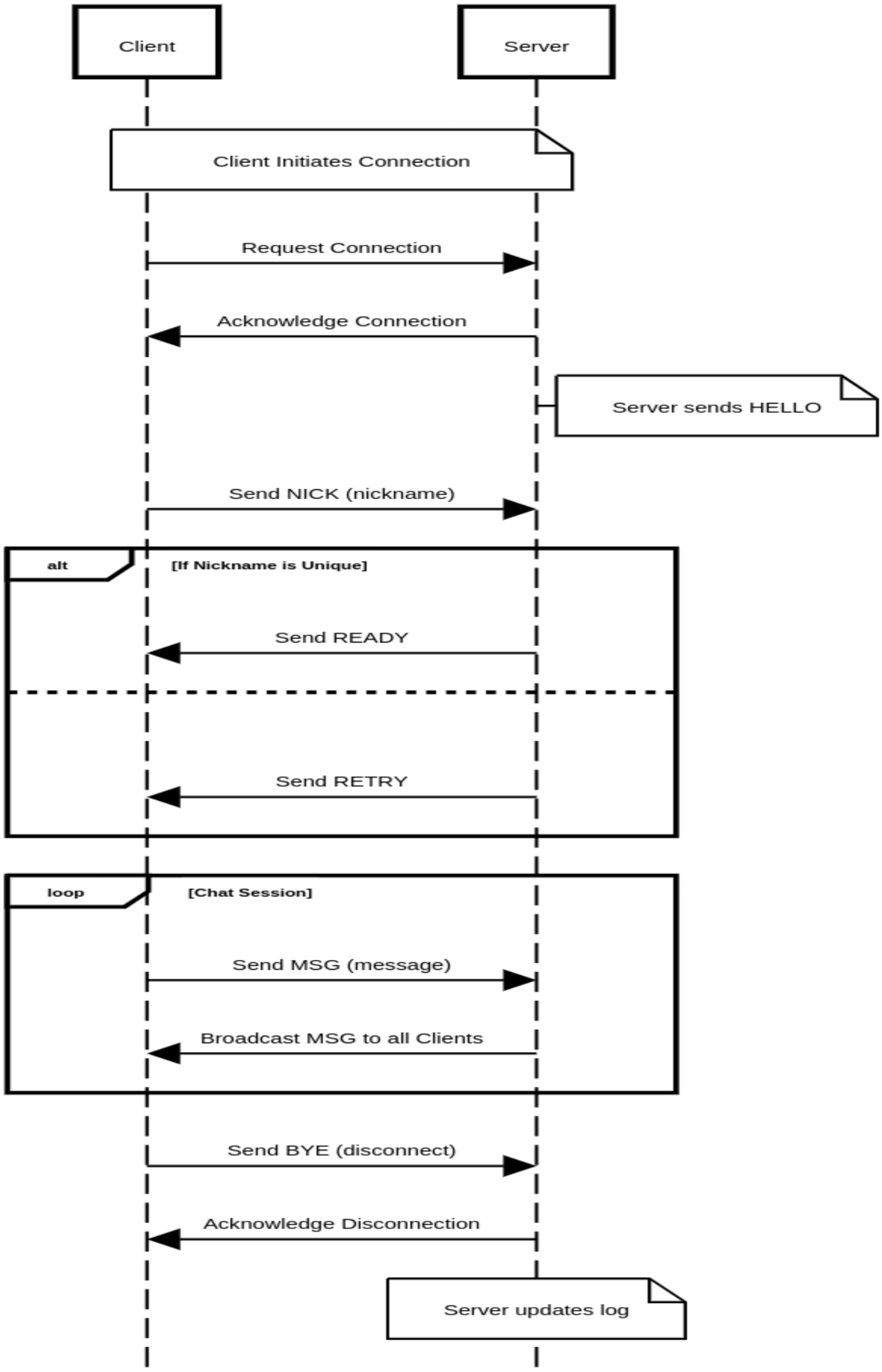
Broadcasting Messages:

- For standard operations, the server receives a JSON packet from one client and broadcasts it to all other connected clients.
- If a packet contains the /exit command, the server terminates that particular client's connection and removes the associated nickname from its active list.

---

9) Sequence Diagram (Found on page 9)

Chat Server Application Interaction



10) Test Plan

Test Case	Brief Description	Input	Expected Output
1	Verify successful login	User selects "myUser"	Successful connection to the chatroom
2	Handle login with a reserved or duplicate nickname	User selects "None"	Prompt to retry with a different username
3	Confirm receipt of a message from another user	Another user sends "Hello World"	All users, including newly joined, receive the message "Hello World"
4	Test sending of a message to the server	User sends "Hello World"	Server logs the message "Hello World"
5	Access chat logs after joining the chat late	User types "/logs"	User receives the recent chat history
6	Validate command input and show error for invalid command	User types "/notcmd"	Display an error message indicating invalid command
7	Ensure user input is not lost during message reception	User is typing a message	User's input remains uninterrupted as new messages are received