Report on NET ASSIGNMENT 1

## Protocol Design and Specification

The protocol design encompasses consideration for privacy/confidentiality , authentication, performance and good user experience. The protocol follows a client-server and p2p architecture.It is designed to be scalable, capable of handling a large number of users and channels. Clients connect to the server using transmission control protocol. Once connected clients are able to establish peer to peer connections using the User Datagram Protocol and communicate to each other through channels. A signalling server manages a list of channels listed by clients taking into consideration the level of privacy specified by channel owners. The protocol supports various commands for allowing different actions between client and server or amongst clients.

The protocol supports two types of messages: a chat message, and a command. Clients use commands to communicate with a signalling server to perform different actions. Commands follow a specific structure which the server is able to interpret. The command structure will be discussed further in the command structure section. Once clients have initiated connections with each other they are able to send chat messages and special chat commands to perform different actions like leaving a channel.

## Message format and structure

For messaging, the protocol uses a command approach to facilitate communication between clients and servers and amongst clients. The commands allow users to perform different actions like getting the channels available on the server, leaving chat rooms or modifying visibility on the lookup index.

The commands follow a specific format which includes commands and parameters. The commands specify the action to be performed and the parameters provide the additional information required for the command. Commands start with prefix **/** (backslash) . Each command has special validates, type handlers and custom error handlers that send response messages to the client. This allows users to get feedback after every action they perform

An example of a command format and the actual command used by the client to create a channel is shown below.

**Command Format:**

This command allows clients to create channels with a level of visibility in the server. They can specify the level of privacy by setting the mode of their channel. If they want the secret to be private , a passkey is specified and can be sent out to specific clients. This can be shows by the optional argument which is prefixed with a dash as shown in the actual command supplied by the user **(-s)** for private channel

|  |
| --- |
| /chan <channel\_name> <mode (optional)> <passkey (optional)> |

**Command supplied by users**

|  |
| --- |
| /chan #channel -s password1234 |

This command will create a secret channel with the name channel, and the passkey “password1234.”

**Response to user who created channel**

|  |
| --- |
| Channel #channel was created successfully |

This is an example of one of the many response messages from the server. Unlike commands sent to the server where they follow a specific format , responses are sent over as normal messages with exception of response messages that return data that the client has to perform wrangling upon. These exceptions are prefixed with different symbols which map the different responses for command.. This includes commands like /join #channel that returns the connection details that the client has to use to initiate a UDP connection with clients on the channel. An example of the response structure from calling /join #channel is shown below.

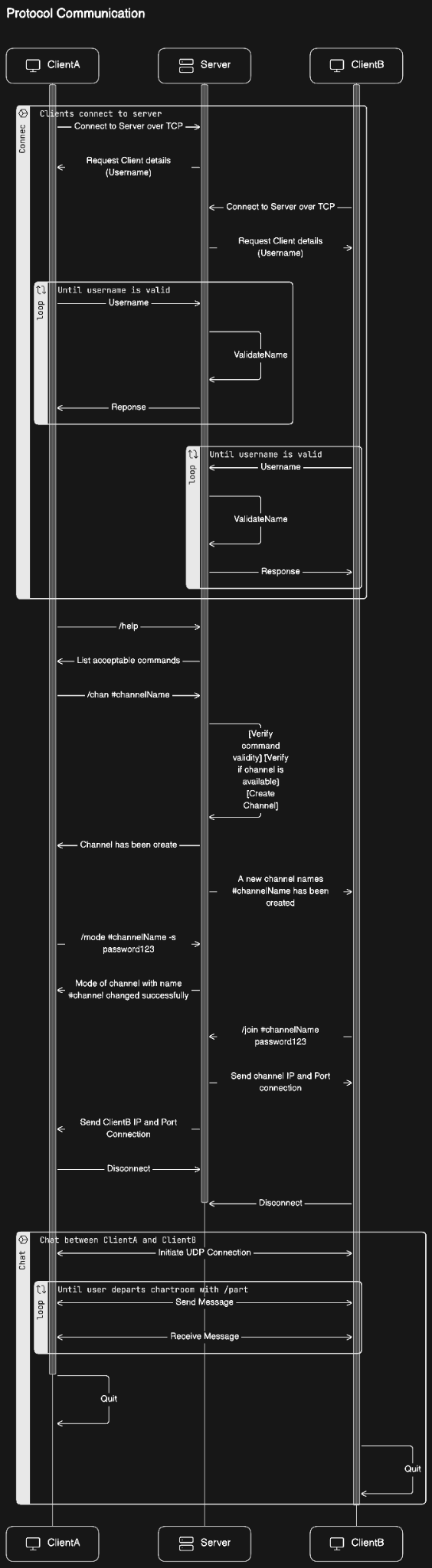
**Response after user send /join #channel**

|  |
| --- |
| >PeerUsername@176.192.0.1:5555 |

This example above is an of the a response a client gets after requesting to join the channel with the name #channel. It allows the client to initiate an UDP connection. The response is prefixed with **>** which maps to the response from the join command. This allows the client to perform special operations for that command. The second part of the command specifies the name of the peer to connect to and the last section is the corresponding address of the peer which includes the IP address and the Port number to connect to.

#### Sequence diagram

This diagram shows two clients connecting to the server through TCP. Performing action to the server with commands. One forms a channel and the other join.They disconnect from server and chat over UDP



## 

## Features

This section highlights the features provided by the protocol and our chat application and their contribution to the overall functionality of the chat app

### Channel Creation

Channel creation allows users to organise conversation based on their interests, topics or privacy. Users can create new channels , join new ones and also manage channel settings. This enhances communication within the application and encourages collaboration within amongst users

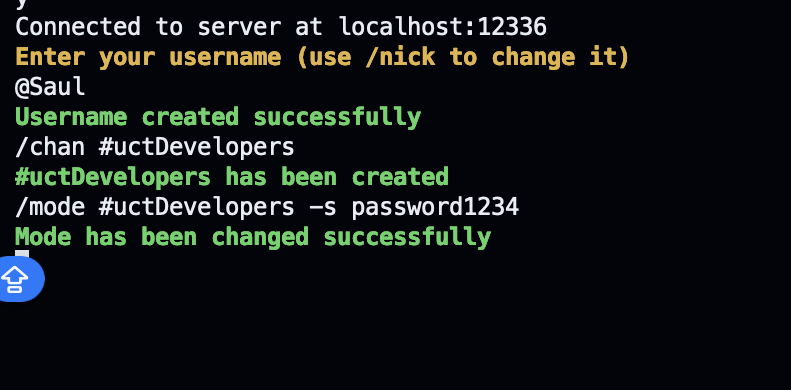
Below is an example of a user creating a channel for UCT developers.



### Channel Privacy Setting

Privacy is at the core of the design. Privacy settings enable users to control access channels to ensure that channels conversations remain private or restricted to specific users. This is done by use of a secret-key that can be shared with specific users.

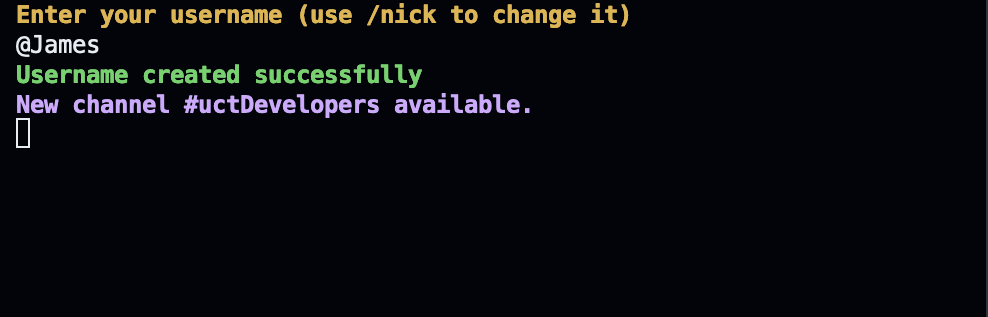
Below is example of a user setting the privacy of a channel and adding a passkey for others to access



### Notification System

This system alerts users about new channel creation or other relevant activities within the app. This ensures that users stay informed and engaged when they are idle.

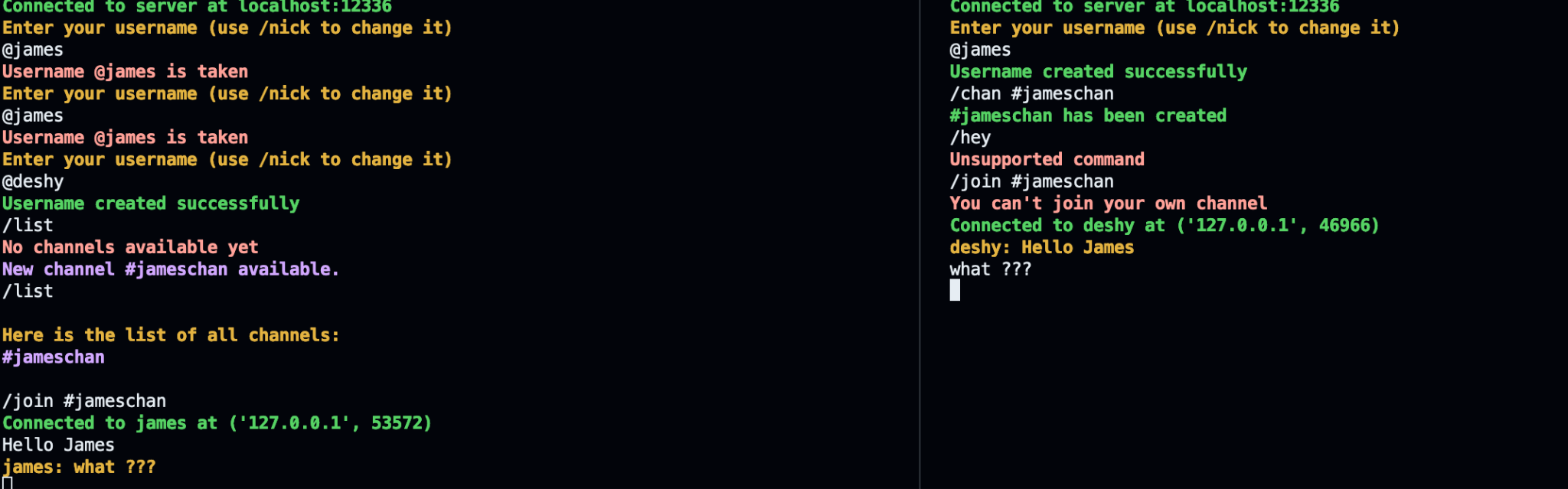
An example is shown below where users are notified of a channel for UCT developers that has been created



### Enhanced CLI Design

The application is accessed through the CLI. It utilises colours to distinguish between different types of response and messages for clients. It helps clients understand the context of each message or notification within the interface. In addition, custom commands are used to perform actions to the server and between clients.

An example is shown below where the user is interacting with the CLI with difference commands and getting the necessary responses.



### Client implementation

1. **Imports and Configuration:**

- Imports necessary modules such as `queue`, `socket`, `sys`, `threading`, and `config`.

- `config` is a module containing configuration details like the server port (`SERVER\_PORT`), which is used to set up connections.

2. **Input Handling:**

- The `input\_thread` function runs on a separate thread to continuously accept user input from the command line. This input is put into a queue for processing.

3. **TCP Connection**:

- The `TCP\_con` function establishes a TCP connection with a server running on `localhost` at the specified port. It sets up separate threads for sending and receiving messages.

- Upon receiving specific messages from the server (starting with ">" or "?"), it triggers the initiation of a UDP connection with another client.

4**. TCP Message Handling:**

- The `send\_messages` function sends messages over the TCP connection, while the `receive\_messages` function receives messages.

- If a message received over TCP indicates the need for a UDP connection, it extracts necessary details like peer IP, username, and port number, initiates a UDP connection, and closes the TCP connection.

5. **UDP Connection**:

- When triggered by the TCP connection, the `UDP\_con` function sets up a UDP connection with another client using threads for sending and receiving messages.

6. **UDP Message Handling**:

- `UDP\_receive\_messages` listens for messages over UDP and prints them out. If it receives a "/part" message indicating a client leaving the chat, it prints a corresponding message and closes the UDP connection.

- `UDP\_send\_messages` sends messages over UDP to the specified peer. If it receives a "/part" message, it sends it to notify the peer and closes the UDP connection.

7. **Utility Functions**:

- `get\_peer\_IP`, `get\_peer\_username`, and `get\_peer\_socknum` are utility functions to extract necessary information from the message received over TCP for initiating a UDP connection.

8**. Main Execution**:

- The `\_\_main\_\_` block initiates the TCP connection when the script is run.