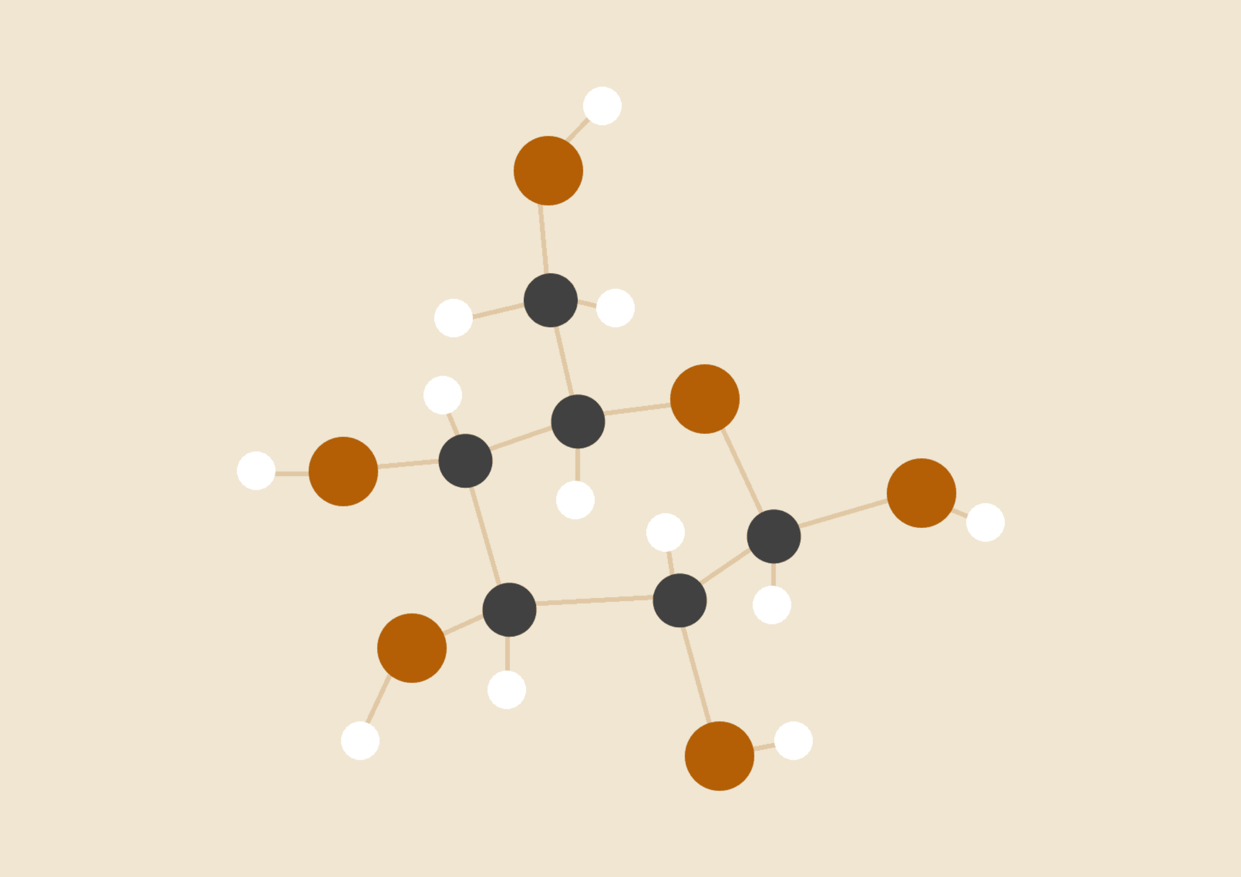
NETWORK ASSIGNMENT 1

TCP, UDP and Protocol Design



WLFETH001, EBRLUK001, LTHMAT005

01.03.2024

NET 41

FUNCTIONALITY

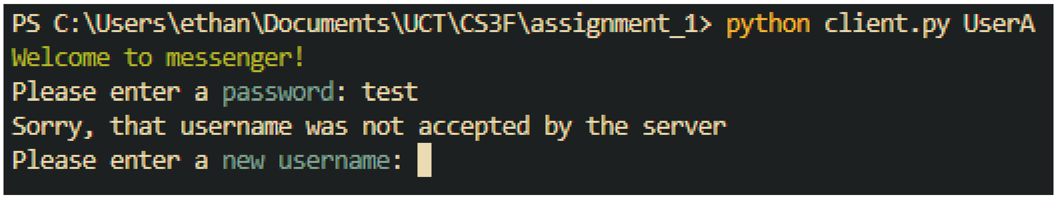
DESCRIPTION

We have built a chat application based on a client-server architecture that allows clients to poll the server for available users and then form peer-to-peer connections with these other users. Authentication has been implemented to ensure confidentiality and all peer-to-peer connections have to be first requested and then accepted before any information is shared. Users can chat with any number of other users, each with a separate chat that saves their previous messages should they move to a different chat.

FEATURES

1. Signup and authentication

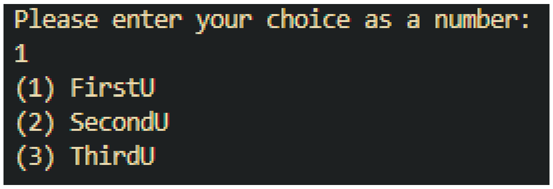
Users sign up with a unique username and password to confirm their identity.



```
PS C:\Users\ethan\Documents\UCT\CS3F\assignment_1> python client.py UserA
Welcome to messenger!
Please enter a password: test
Sorry, that username was not accepted by the server
Please enter a new username: 
```

2. Polling the server for users -

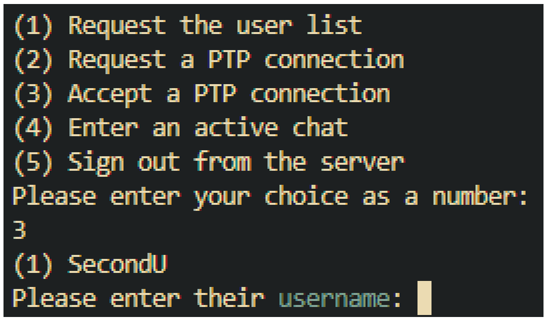
Users can view the other users on the server in order to request connections.



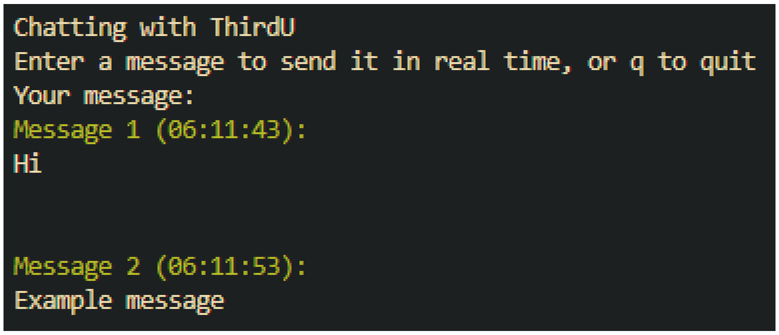
```
Please enter your choice as a number:
1
(1) FirstU
(2) SecondU
(3) ThirdU
```

3. Confidential chats -

Chats require confirmation from both parties to be created and only those two users can view those messages. Chats also save messages to be viewed again if the chat is re-entered.



```
(1) Request the user list
(2) Request a PTP connection
(3) Accept a PTP connection
(4) Enter an active chat
(5) Sign out from the server
Please enter your choice as a number:
3
(1) SecondU
Please enter their username:
```



```
Chatting with ThirdU
Enter a message to send it in real time, or q to quit
Your message:
Message 1 (06:11:43):
Hi


Message 2 (06:11:53):
Example message
```

SERVER IMPLEMENTATION

The server and clients communicate via TCP sockets, with the server opening a new socket and spinning up a new thread for each client. Each thread continuously receives and responds to messages, retrieving shared data from the main server class. Locks allow data structures to be shared safely between threads.

CLIENT IMPLEMENTATION

The client takes the server ip address and port number as command-line arguments and connects to it via TCP. From there, it uses commands to communicate with the server and secure peer-to-peer connections, within which data transfer messages are used to communicate text. The client communicates with the server as required to service user requests, but also regularly checks for any requested or accepted peer-to-peer connections. The server provides the information for a TCP connection to be formed between any two clients. Once other details have been shared securely, a new UDP connection is formed and used from then on to pass text messages as described below. A Chat class is used to save information relating to the messages passed between the two clients. Chat history is stored so that multiple chats can be switched to and from. Finally, the chat class has a thread that continuously checks the connection for new messages to update the chat in real time.

PROTOCOL SPECIFICATION

MESSAGES

Messages are bytestrings that are sent between the client and server in order to convey some information. All messages in our protocol have a specific layout as explained below. They are made up of headers and content, with commands not containing any content. The header always precedes the content and has a set size and layout.

The headers of all messages begin with a single byte that specifies whether the message is a command or data transfer message, 0b10000000 or 0b00000000 respectively. Following this, the layout of the two headers diverges.

COMMAND

The second byte specifies the type of command. No matter the type, the command header also contains two 8 byte parameter buffers, param_1 and param_2 that immediately follow the first two bytes of the header. Both parameters are padded by null bytes to the left. This makes 18 bytes in total for a command.

Usernames are unique so a request to sign up with an already existing username will be declined by the server. There are no restrictions on passwords however. The only constraint on both is that they must be 8 characters or less.

The user list is returned via a data transfer message from the server, with the user list making up the content of the text message.

The server transforms a REQUEST_PTP command into a relayed version that contains the same information, and passes on an ACCEPT_PTP command in the same manner. No information is changed between the requesting client and the accepting client.

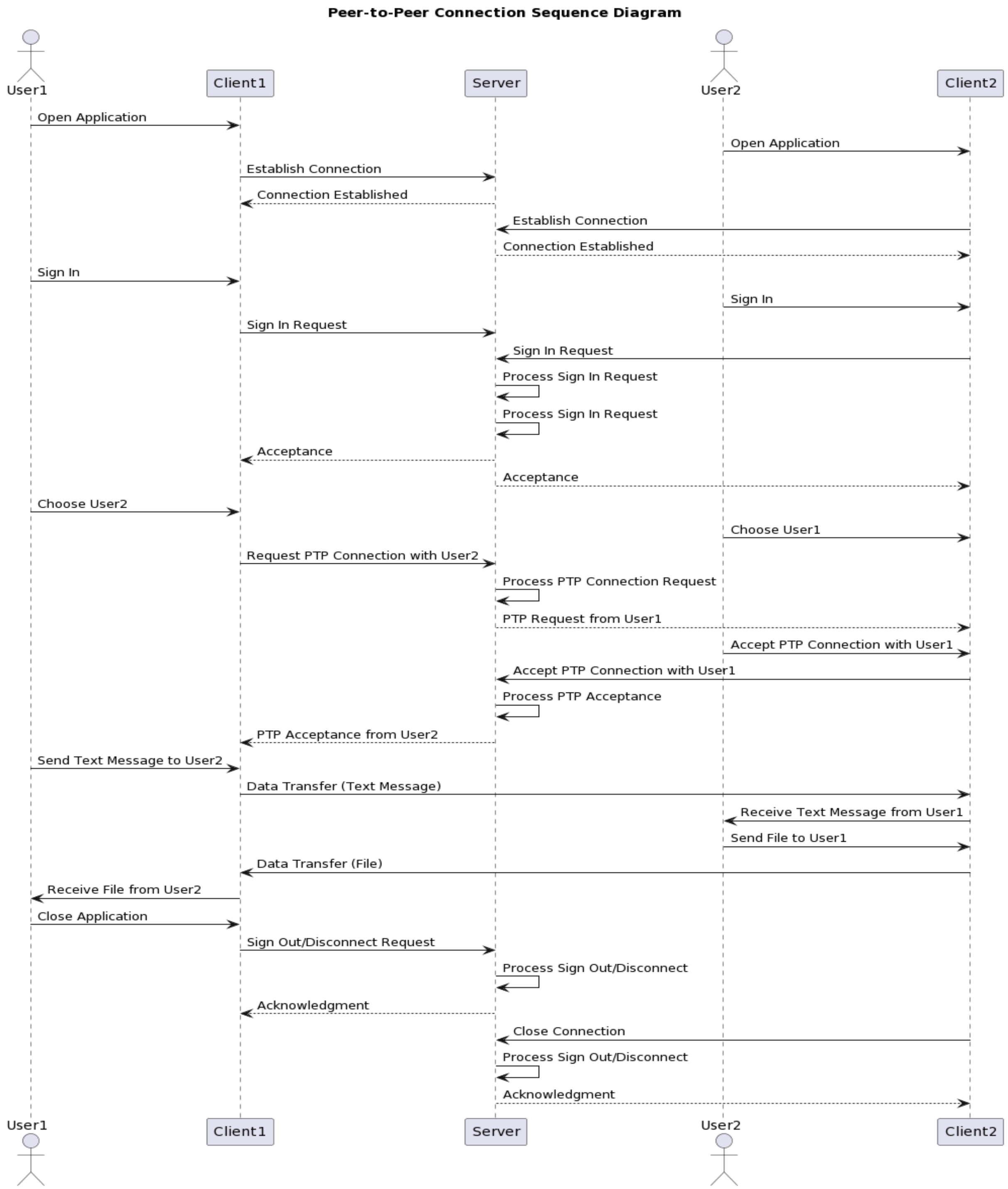
In ACCEPT_PTP, the ip address (having been converted into an integer) makes up the first four bytes of the second parameter, and the port number makes up the next two bytes.

NAME	DESCRIPTION	PARAMETERS
SIGN UP	The client requests to sign up to the server.	1: UTF-8 encoded username 2: UTF-8 encoded password
DECLINE_SIGN_UP	The client declines the user's sign up due to invalid username.	
SIGN IN	The client requests to sign in to the server.	1: UTF-8 encoded username 2: UTF-8 encoded password
ACCEPT_SIGN_IN	The client accepts the user's sign up / sign in request.	
DECLINE_SIGN_IN	The client declines the user's sign up due to invalid username/password.	
REQUEST_USER_LIST	The client requests the list of visible online users.	
REQUEST_PTP	The client requests a PTP connection with another client.	1: Accepting client's username
DECLINE_PTP	The client declines a PTP connection with another client.	
ACCEPT_PTP	The client accepts a PTP connection and sends connection details.	1: Requesting client's username 2: IP address and port number of TCP connection
SIGN_OUT	The client requests that the connection be closed.	1: UTF-8 encoded username 2: UTF-8 encoded password

DATA TRANSFER

Data transfer messages are either text or a file, with first byte 0b00000000 or 0b01000000 respectively. The next byte is the filename length, otherwise empty. The next four bytes encode the length of the message or file size in bytes. The filename precedes the content.

SEQUENCE DIAGRAM


Peer-to-Peer Connection Sequence Diagram
User1
Client1
Server
User2
Client2
Open Application
Open Application
Establish Connection
Connection Established
Establish Connection
Connection Established
Sign In
Sign In
Sign In Request
Sign In Request
Process Sign In Request
Process Sign In Request
Acceptance
Acceptance
Choose User2
Choose User1
Request PTP Connection with User2
Process PTP Connection Request
PTP Request from User1
Accept PTP Connection with User1
Accept PTP Connection with User1
Process PTP Acceptance
PTP Acceptance from User2
Send Text Message to User2
Data Transfer (Text Message)
Receive Text Message from User1
Send File to User1
Data Transfer (File)
Receive File from User2
Close Application
Sign Out/Disconnect Request
Process Sign Out/Disconnect
Acknowledgment
Close Connection
Process Sign Out/Disconnect
Acknowledgment
User1
Client1
Server
User2
Client2