YNGFYN001
GMLMOG016
FLDCLA001

CSC3002F

NETWORKS ASSIGNMENT 1

INTRODUCTION

Network applications and their application-layer protocols reside in the application layer. The goal of this assignment is to design an application-layer protocol to support a client-server chat application. The application-layer protocol designed took into consideration use of the UDP protocol at the transport layer to transmit segments to the network layer.

COMMUNICATION MODEL

CLIENT-SERVER ARCHITECTURE

The architectural paradigm made use of in this chat application is the client-server architecture. This allows for provision of an "always-on" host to accept communications from many clients. In the context of a communication session between a pair of processes, the process that initiates the communication is the client.

SERVER

Although this server is not "always on", it is required that the server program be running before any client program can run. It is the process that waits to be contacted to begin the session. The server has a fixed IP address, thus the client can always contact the server by sending a packet to its IP address. As there are not a large number of clients making use of the server, only one host is needed.

CLIENT

The client in this case has its own IP address and port number that it makes use of in order to connect to the server. Clients communicate with each other through processes that reside in the application layer. In this network application, a client browser process exchanges messages with a Web server process. It is only through the server that one client may communicate with another.

SOCKET

This application makes use of a socket, which is the interface between the application layer and the transport layer within each host running the application. It is the API between the application and the network since the socket is the programming interface with which this network application is built. Messages are pushed through the socket at both the sending and receiving side of the network layer.

PROTOCOLS

APPLICATION LAYER PROTOCOL

The application layer protocol present in this assignment is distributed over multiple end systems, with the application in one end system exchanging packets of information with the application in another end system. The packet, in this case, contains the message being sent/received. A client must first connect to the server. This is simply completed by having the client enter their IP address and port number. A socket is created so that the client program can send messages through the network. At this point, the client is considered "online". Once the connection is established, the client and the server processes access UDP through their socket interfaces. The socket interface is the door between the client/server process and the UDP connection. Despite having the ability to store client information, this application is considered a stateless protocol as each request can be understood in isolation.

TRANSPORT LAYER

The UDP transport protocol is connectionless, so data transfer services are known to be unreliable. Messages that do arrive at the receiving process may arrive out of order—if they arrive at all. Additionally, there are no congestion-control mechanisms, the sending side of UDP can pump data into the network layer at any rate. As developers, it has been accepted that some loss may occur due to the risk of message loss, and thus acknowledgement passages between the client and server have been instantiated.

NETWORK LAYER

At the network layer, a segment as well as its destination address is received from UDP. This datagram is then delivered to the transport layer in the destination host. As this chat application is largely based off the internet, it makes use of the Routing Protocol. This protocol specifies the routes between the nodes in a computer network, allowing data packets to be forwarded through the internet and to their destination.

PATTERN OF COMMUNICATION

1. Once a client and server have started, the client must first connect to the server using the command (/connect <ip> <port>).
2. The server will prompt the client with ("/n" for new user, "/e" for existing user), /n will create a new user where the next string is the username and the string after that would be the password of the user.

3. The new user is checked to see if it already exists. If it does, the server will tell the client that the user information is in use and the client will have to try again.
4. Upon successfully creating a new user, the server will then say (/e <username> <password>), prompting a log in. If the client inputs these details and they don't exist in the server's database, the server will tell the client that the user does not exist. Alternatively, the server will send the (Type "/help" for help) and ([CONNECT]) messages if the client logs in successfully.

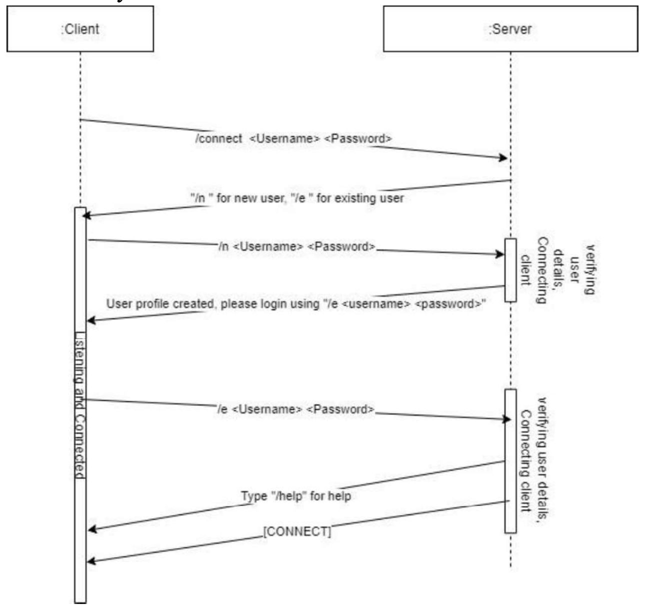

:Client
:Server
/connect <Username> <Password>
"/n " for new user, "/e " for existing user
/n <Username> <Password>
verifying user details, Connecting client
User profile created, please login using "/e <username> <password>"
Listening and Connected
/e <Username> <Password>
verifying user details, Connecting client
Type "/help" for help
[CONNECT]

Figure 1 Creating a new user

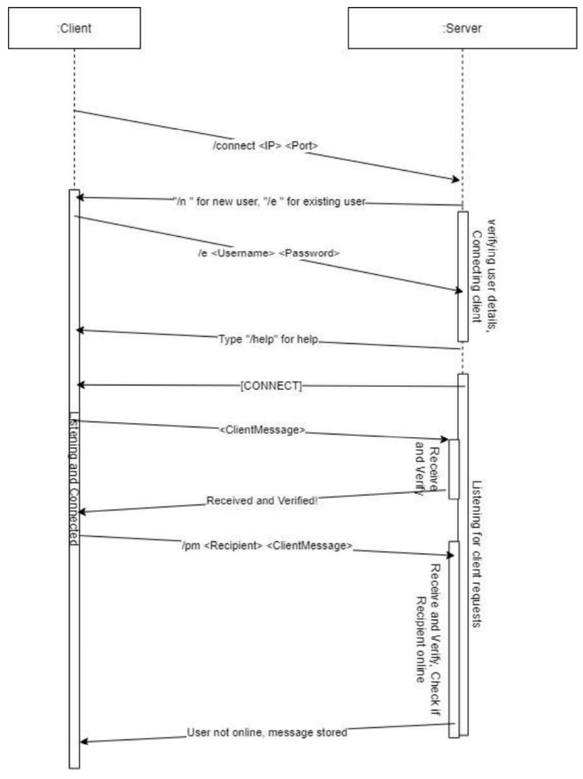

:Client
:Server
/connect <IP> <Port>
"/n " for new user, "/e " for existing user
/e <Username> <Password>
verifying user details, Connecting client
Type "/help" for help
[CONNECT]
<ClientMessage>
Receive and Verify
Received and Verified!
/pm <Recipient> <ClientMessage>
Receive and Verify, Check if Recipient online
Listening for client requests
Listening and Connected
User not online, message stored

Figure 2 Personal Message Sequence Diagram

MESSAGE STRUCTURE

CLIENT

REQUEST MESSAGES

The client sends messages to the server through the socket. All packets being pushed onto the transport layer by the client will have the following structure:

bytesToSend	serverAddress

In which the serverAddress consists of the following:

IP Address of Server	Port Number of Sever

There are many cases in which a message may be sent to the server.

<u>CASE ONE: USER CONNECTION</u>

HEADER	DESCRIPTION
CONNECT	The header is encoded into bytes and sent to the server in order to indicate the user has come online.
DISCONNECT	Upon the user pressing quit, the disconnect header will be sent to the server. In addition to this, the username is included in the 'bytesToSend' in the form of the message body. This username will indicate to the server which user to remove from the "online users" file.

<u>CASE TWO: MESSAGES</u>

COMPONENT	DESCRIPTION
HEADER	The header/instruction will always be "SEND" when the client is requesting to send a message.
USERNAME	The username of the client.

RECIPIENT	There are two options for this field in the body of the message format: • "global" - when the user has not indicated they would like to send a personal message, the recipient is set to global and the server will broadcast this message to all users. • The username of the client the user is trying to message personally - the server will then pass this message directly to that user if their username is existent on the profiles database.
HASH	The hash function that has been used on the message being sent. This allows for the server to decrypt and ensure that the integrity of the message being sent before forwarding to the recipient.
MESSAGE	The message the client has inputted via the user interface.

CASE THREE: USER LOG IN

COMPONENT	DESCRIPTION
HEADER	The header acts as an instruction, which will tell the server to do one of the following when received: • Create a profile - "CREATE" • Login to a profile - "LOGIN"
USERNAME	The username the client has chosen, stored on the server.
PASSWORD	The password the client has chosen, stored on the server.

RESPONSE MESSAGES

The messages received via the client's socket will always be incoming from the server. The message displayed to the client is dependent on the header. The following outlines the reactions based on the header of the incoming packet:

HEADER	DESCRIPTION
BYE	This means the user has disconnected from the server and the threads no longer need to listen for incoming messages from the socket.
OK	This means that the message passed verification and is being sent to the recipient.
ERROR	The message was lost somewhere in the transfer and needs to be resent.
If the header does not match any of the above	The application will output to the user the ASCII text received from the socket. This is most commonly the "Received and Verified" message when a message has been successfully delivered to the server.

SERVER

REQUEST MESSAGES

The packet structure received from the socket on the server side is as follows:

clientMsg	senderAddress

The senderAddress is used to log the user in at the initial connection, as well as to provide further information on the origin of the packet. It consists of the following components:

Local IP Address of Sender	Local Port Number of Sender

The clientMsg first needs to be decoded. Once the clientMsg has been converted from bytes to ASCII, the clientMsg can be seen to consist of the following components:

CASE ONE: CLIENT COMING ONLINE

COMPONENT	DESCRIPTION
INSTRUCTION	Instructs the server to do one of the following: • Create a profile - "CREATE" • Login to a profile - "LOGIN" • Process a user as being 'online' - "CONNECT"

	• Process user as being 'offline' - "DISCONNECT"
USERNAME	The username the client has chosen, stored on the server
PASSWORD	The password the client has chosen, stored on the server

CASE TWO: CLIENT SENDING A MESSAGE

COMPONENT	DESCRIPTION
INSTRUCTION	Instructs the server send a message - "SEND"
MESSAGE	The ASCII message that must be sent, still with the hash function encrypting it.
RECIPIENT	The username of the client the message is directed toward.

RESPONSE MESSAGES

All response messages are sent to either:

a) The intended recipient, containing:

bytesMsg	recipAddress

In this case, the the bytesMsg consists of the ASCII message that will be delivered to the recipient, with its contents hashed and encoded into bytes. The recipAddress is the IP and port number injected into the socket, indicating the destination address.

Local IP Address of Recipient	Local Port Number of Recipient

Before sending the message, however, the ASCII message hash function must be verified. If the message has incurred no damage, the message is sent to the recipient and the server informs the sender of this.

b) The original sender. The server follows the same criteria as sending a message to any other client, as indicated in a) above.

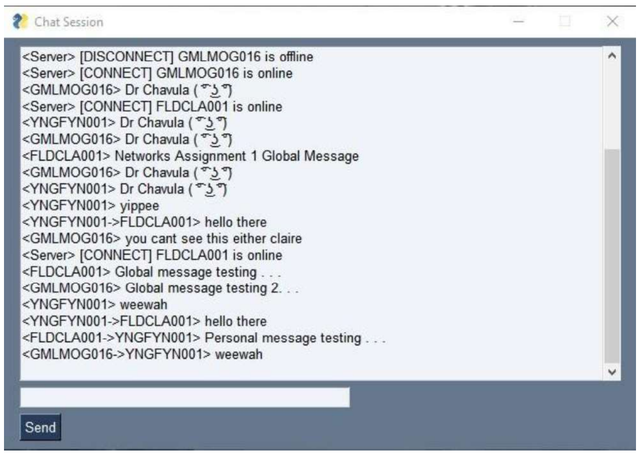
Message to Socket (bytesMsg)	DESCRIPTION
"You're all alone..."	An indication if the user is the only user online and that client is trying to send a global message.
"User not online, message stored"	The personal message trying to be sent is to an offline user, the client is informed that the message is stored.
"User does not exist"	The personal message trying to be sent is to a user that does not exist, the client is informed.
"Messages received while offline:"	When the user logs in, the messages they received whilst offline are sent to the socket. These will be displayed along with the above message automatically.
"Received and Verified!"	Whether or not a message was received by the server in a correct state.
"Incorrect user details"	If the user trying to send the message is making use of the correct profile details.
"User profile created, please login using "/e <username> <password>"	A user profile has been created successfully.
"Username in use, please try another"	The username the client is trying to make use of is already taken.
"Type "/help" for help"	The ability to press '/help' for a menu option, which will subsequently display further prompts a user can adhere to.
"Incorrect user details"	If the user cannot be verified as per the server's list of users, a message is sent to the socket indicating that the user details are incorrect.

c) The user interface, indicating:
- "Server is online": An indication of when the server is able to perform functions for the clients.
- "<username> is online"/ "<username> is offline": When a user comes online/offline

FUNCTIONALITY

SECURITY: A hash function is applied to each message, ensuring data integrity.
SPEED: This application-layer protocol encodes each message into bytes before being sent. This allows messages to be made as small as possible, not only reducing network traffic but further avoiding packet losses.
RELIABILITY: When a message is received by the server, a confirmation message is sent to the socket of the sender. This ensures that messages sent are received. If not, a loss detection mechanism allows for the client to be informed that their message has been lost.
VERIFICATION: After the message has been decrypted into ASCII by the server, the hash function is checked to ensure the integrity of the message.
REALTIME INTERACTION: Threads are used to ensure the client and server are continuously listening for incoming packets from their respective sockets.

FEATURES



```
<Server> [DISCONNECT] GMLMOG016 is offline
<Server> [CONNECT] GMLMOG016 is online
<GMLMOG016> Dr Chavula ( ͡° ͜ʖ ͡°)
<Server> [CONNECT] FLDCLA001 is online
<YNGFYN001> Dr Chavula ( ͡° ͜ʖ ͡°)
<GMLMOG016> Dr Chavula ( ͡° ͜ʖ ͡°)
<FLDCLA001> Networks Assignment 1 Global Message
<GMLMOG016> Dr Chavula ( ͡° ͜ʖ ͡°)
<YNGFYN001> Dr Chavula ( ͡° ͜ʖ ͡°)
<YNGFYN001> yippee
<YNGFYN001->FLDCLA001> hello there
<GMLMOG016> you cant see this either claire
<Server> [CONNECT] FLDCLA001 is online
<FLDCLA001> Global message testing . . .
<GMLMOG016> Global message testing 2. . .
<YNGFYN001> weewah
<YNGFYN001->FLDCLA001> hello there
<FLDCLA001->YNGFYN001> Personal message testing . . .
<GMLMOG016->YNGFYN001> weewah
```

USER INTERFACE
The GUI allows for a more user-friendly experience with this chat application. This includes a menu when /help is entered, which will enable to user to either sent a personal message, broadcast message, disconnect from the server or quit the application.
USER AUTHENTICATION
After connecting with one's IP address and port number, they are prompted to log in. This login allows for user authentication, a key process in keeping unauthorized users from gaining sensitive information.
RETRIEVAL OF HISTORY
Upon login, the user will receive all the messages that were sent to their username whilst offline.
DIALOGUE MANAGEMENT
The user is constantly informed of their online/offline status, as well as the status of the message they have sent.
TRANSMISSION PATTERNS
The server retains the username and passwords of its previous clients so that they are able to log in. This allows the server to maintain a list of users that are currently online. This is a necessity because if the client wants to send a personal message to another client who is offline, the server is able to save the message and direct it to that user when they do eventually come online. However, this application is still considered a stateless protocol as each request can be understood in isolation.

INDIVIDUAL CONTRIBUTION

STUDENT NUMBER		
YNGFYN001	100%	
GMLOG016	100%	
FLDCLA001	100%	