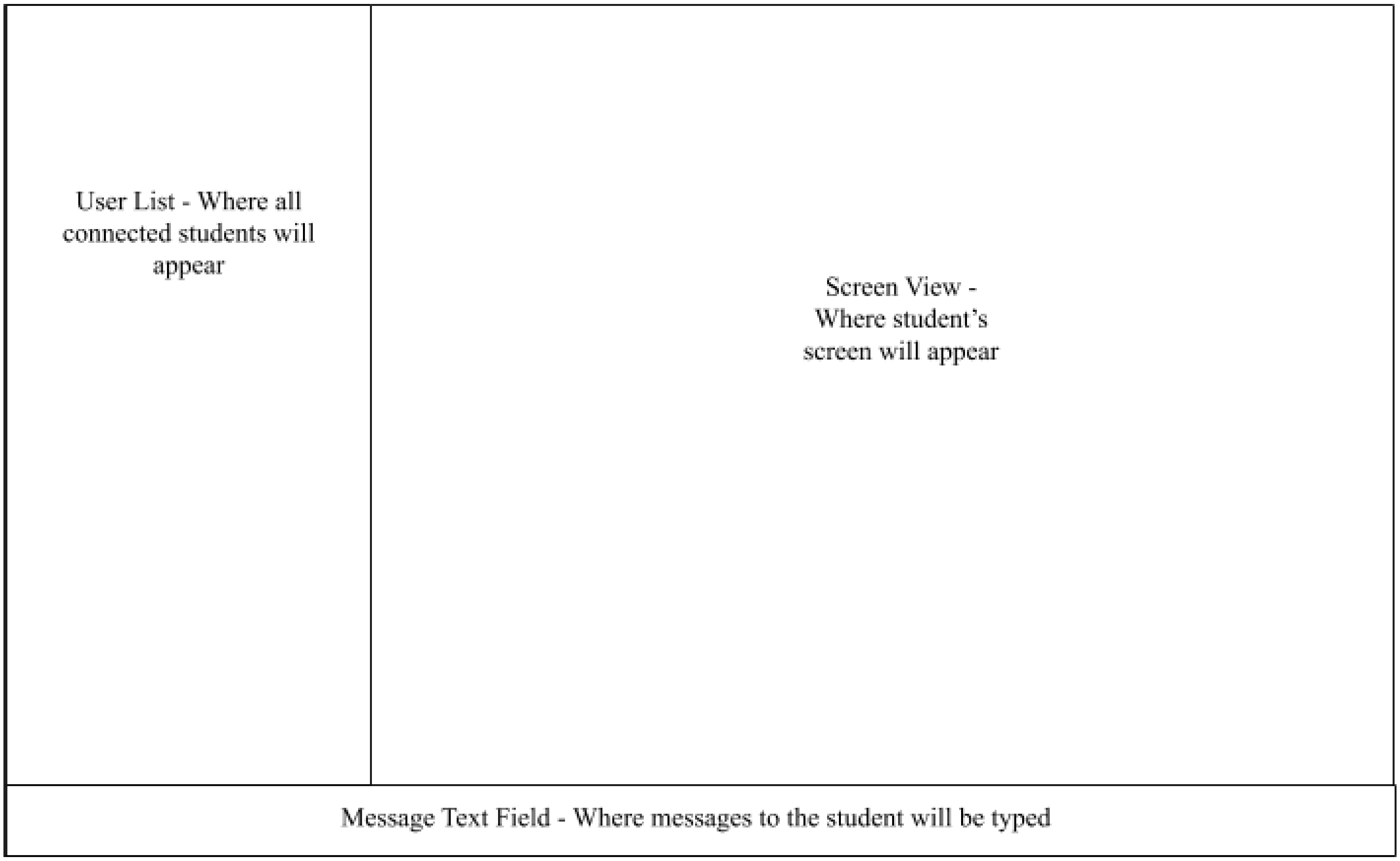
Criterion B: Design

Words: 481

GUI Design

This drawing was shown to Mr. Devry to explain the server GUI. The flowcharts helped outline the processes in server-client communication and server-side user input. After discussion, all were approved.[1]

Server:

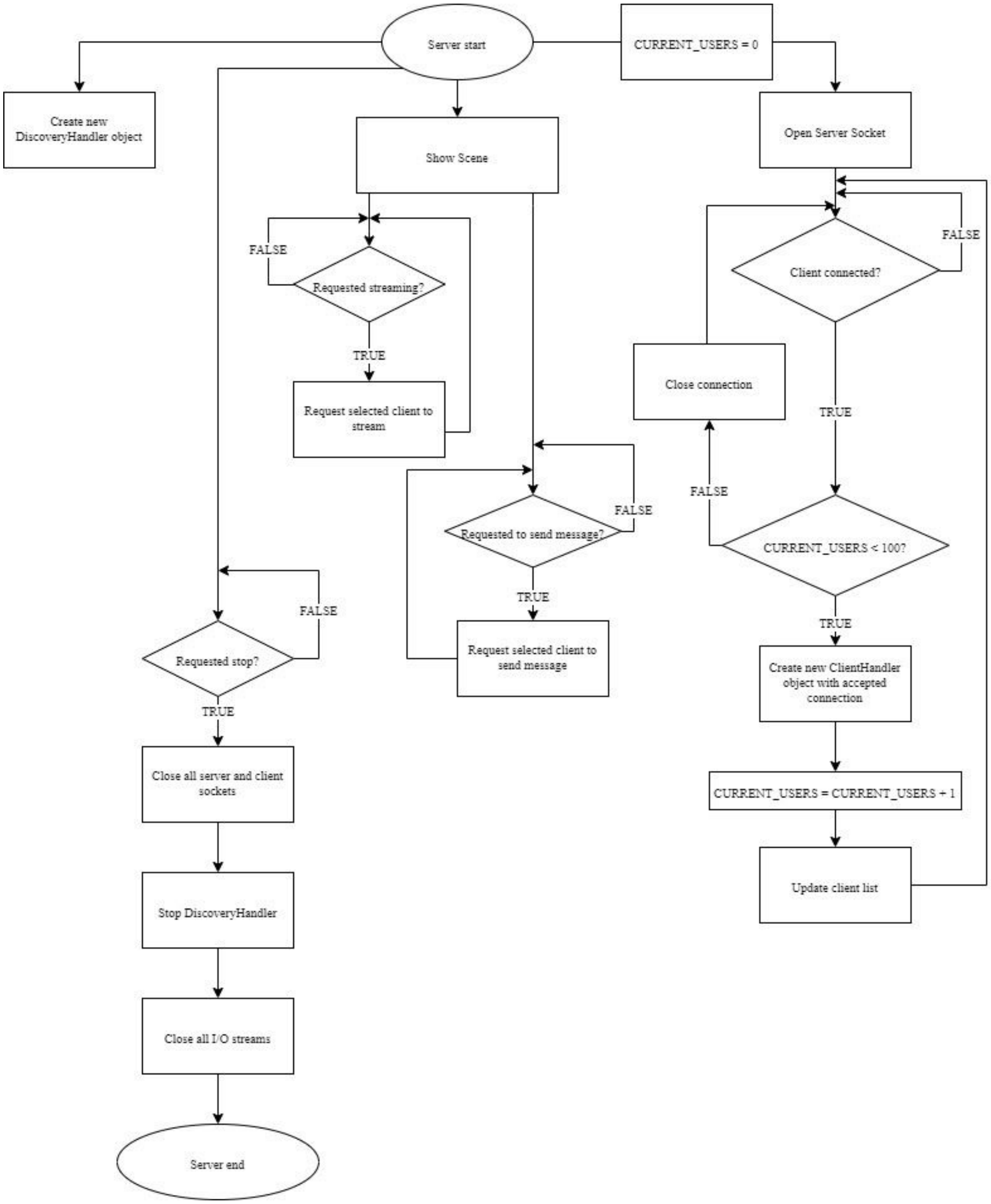

User List - Where all connected students will appear
Screen View - Where student's screen will appear
Message Text Field - Where messages to the student will be typed

[1] Refer to appendix, 1.3

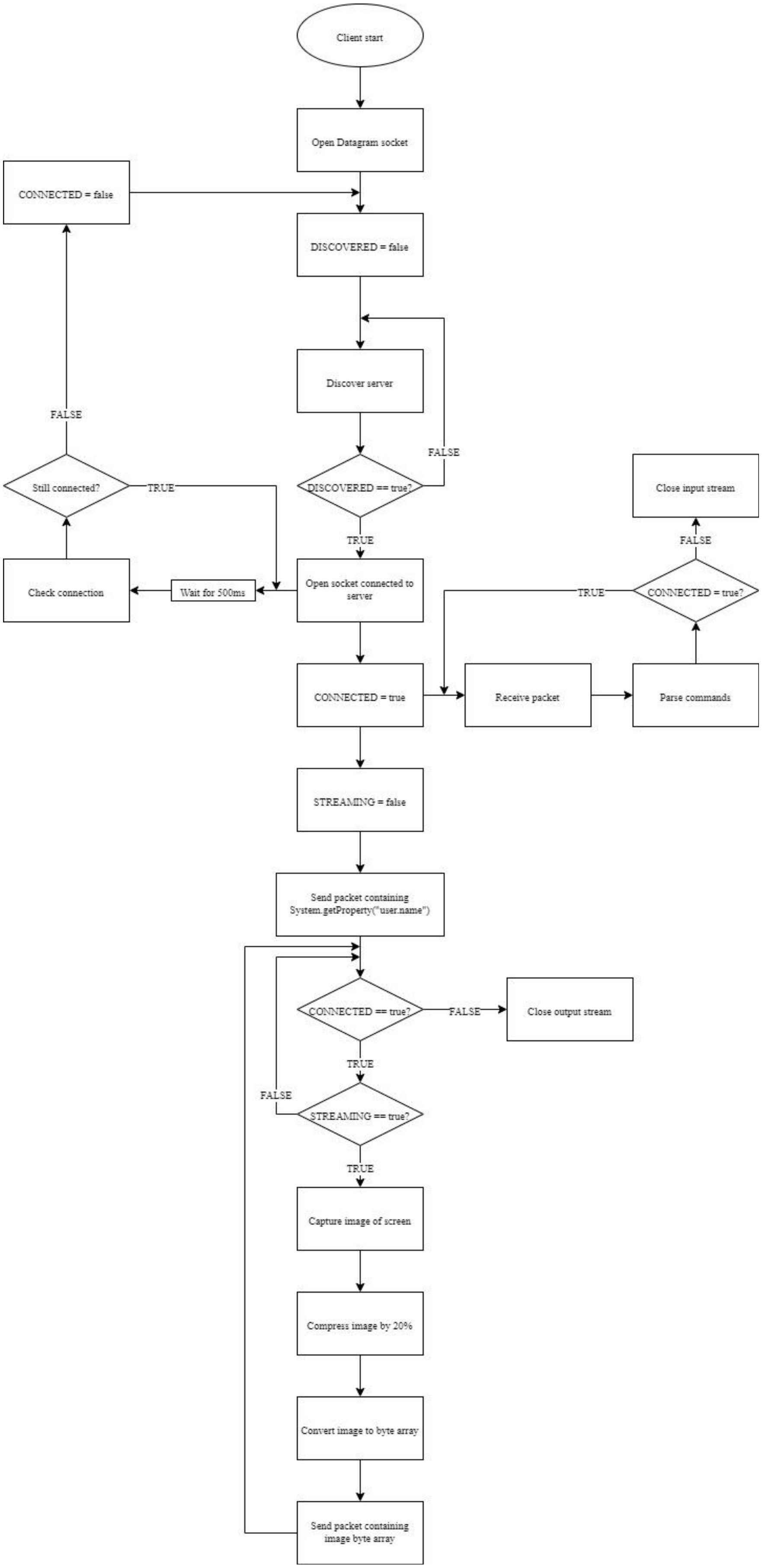
Process Flowcharts

The following flowcharts outline the main processes in the programs and objects.

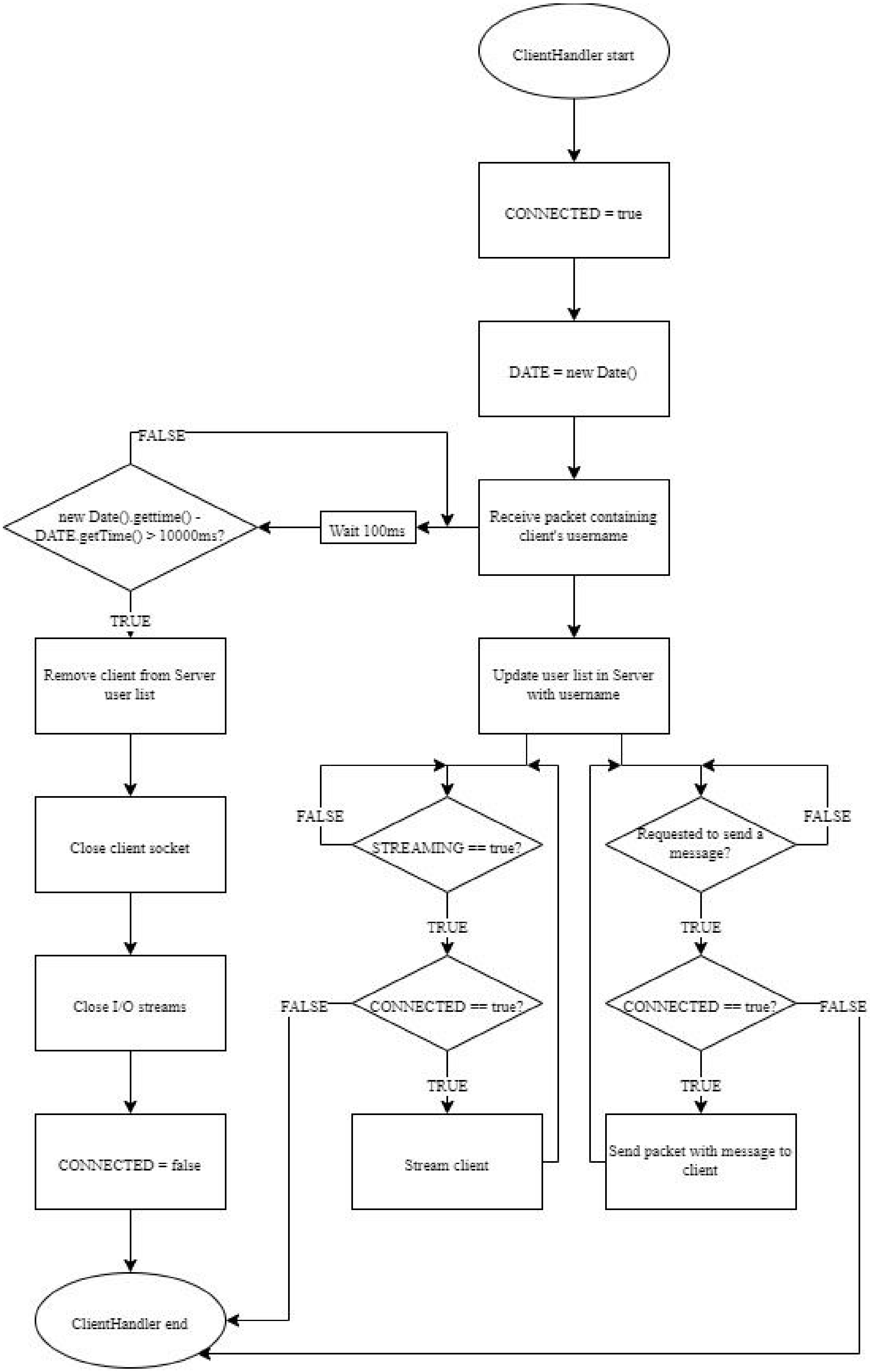
Server:



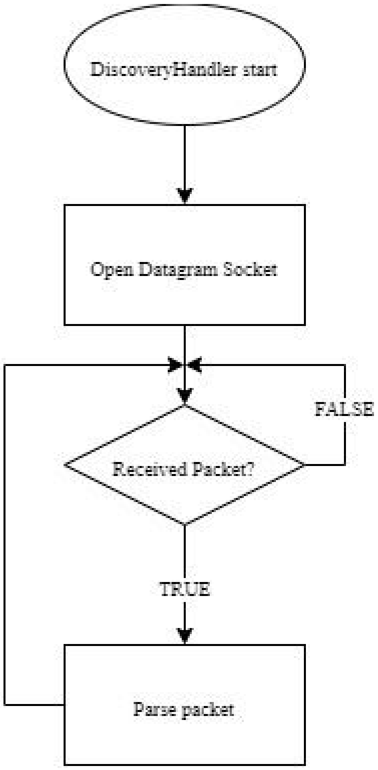
Client:


Client start
Open Datagram socket
CONNECTED = false
DISCOVERED = false
Discover server
FALSE
DISCOVERED == true?
TRUE
Still connected?
TRUE
FALSE
Check connection
Wait for 500ms
Open socket connected to server
Close input stream
FALSE
CONNECTED = true?
TRUE
CONNECTED = true
Receive packet
Parse commands
STREAMING = false
Send packet containing System.getProperty("user.name")
CONNECTED == true?
FALSE
Close output stream
TRUE
FALSE
STREAMING == true?
TRUE
Capture image of screen
Compress image by 20%
Convert image to byte array
Send packet containing image byte array

ClientHandler Object:



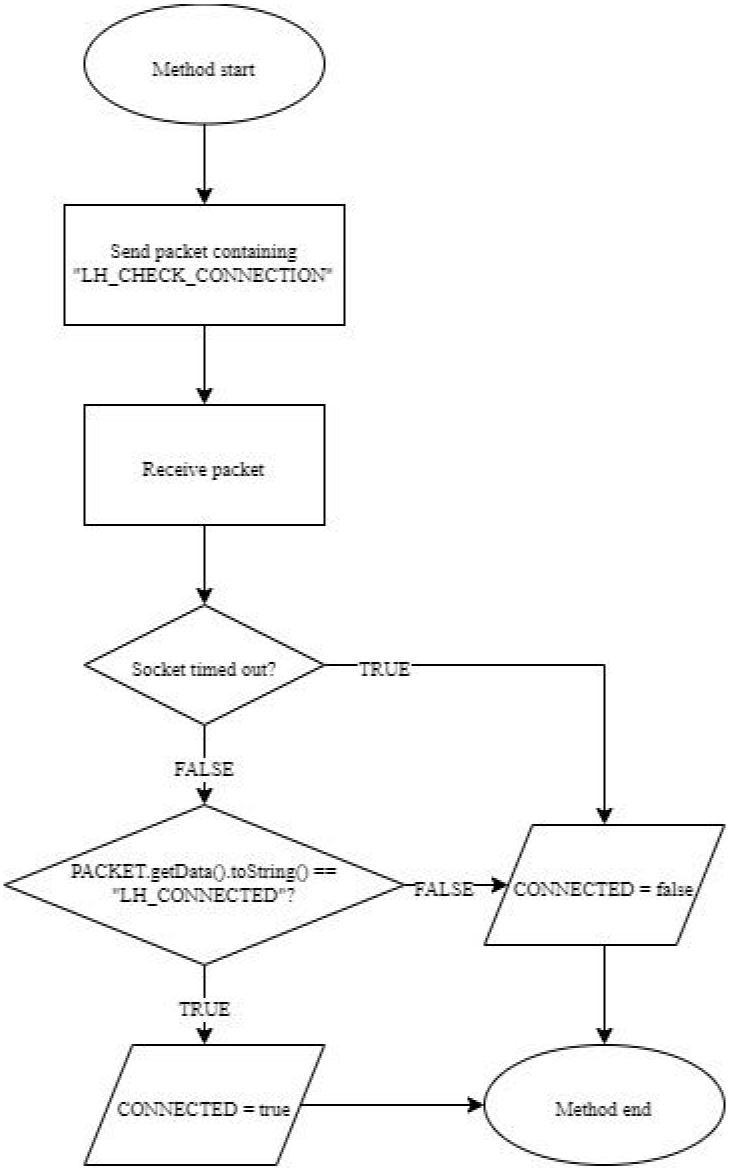
DiscoveryHandler Object:



Algorithms

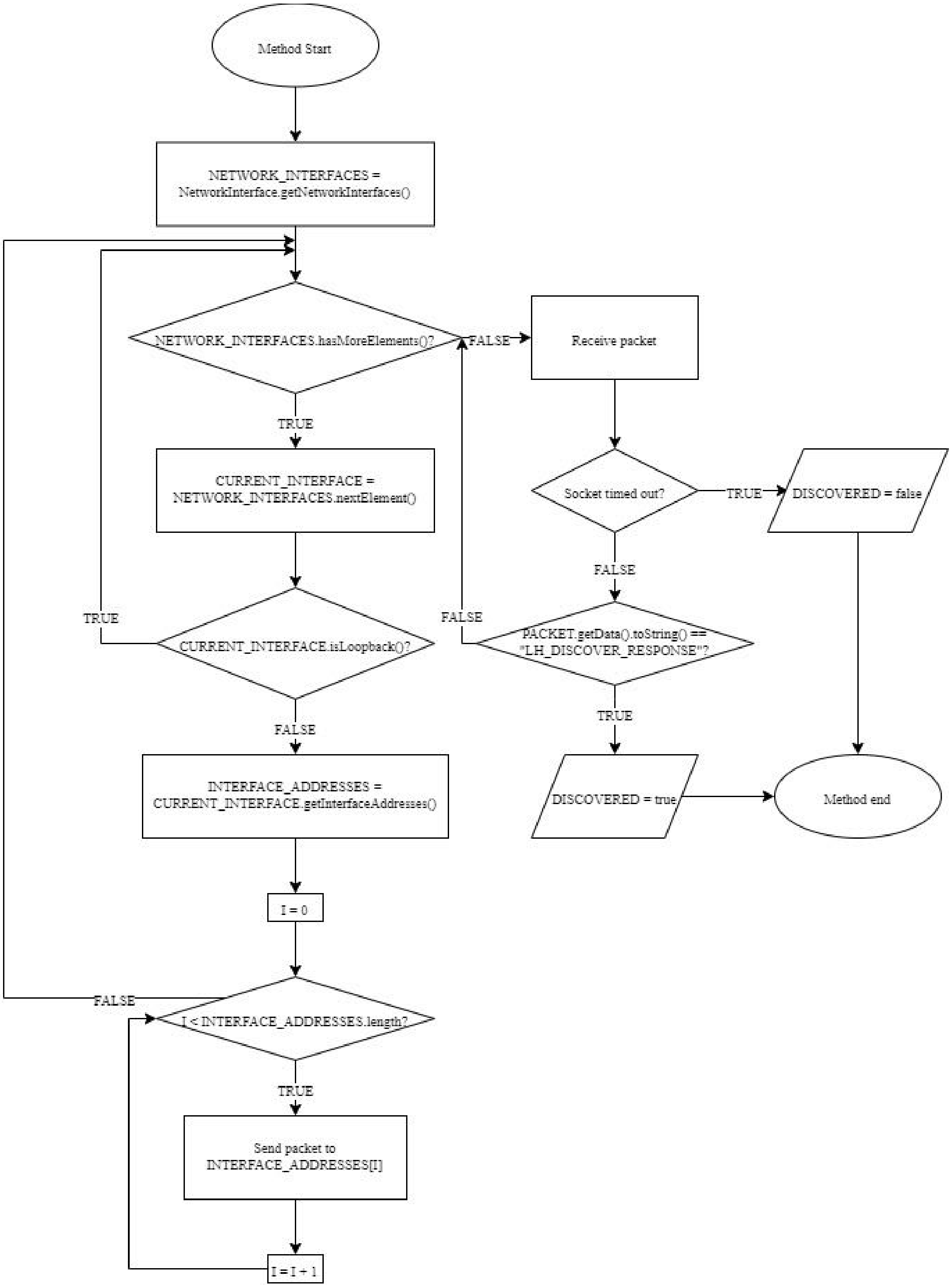
These flowcharts outline the important algorithms that handle communication between the server and client.

Check connection in Client:



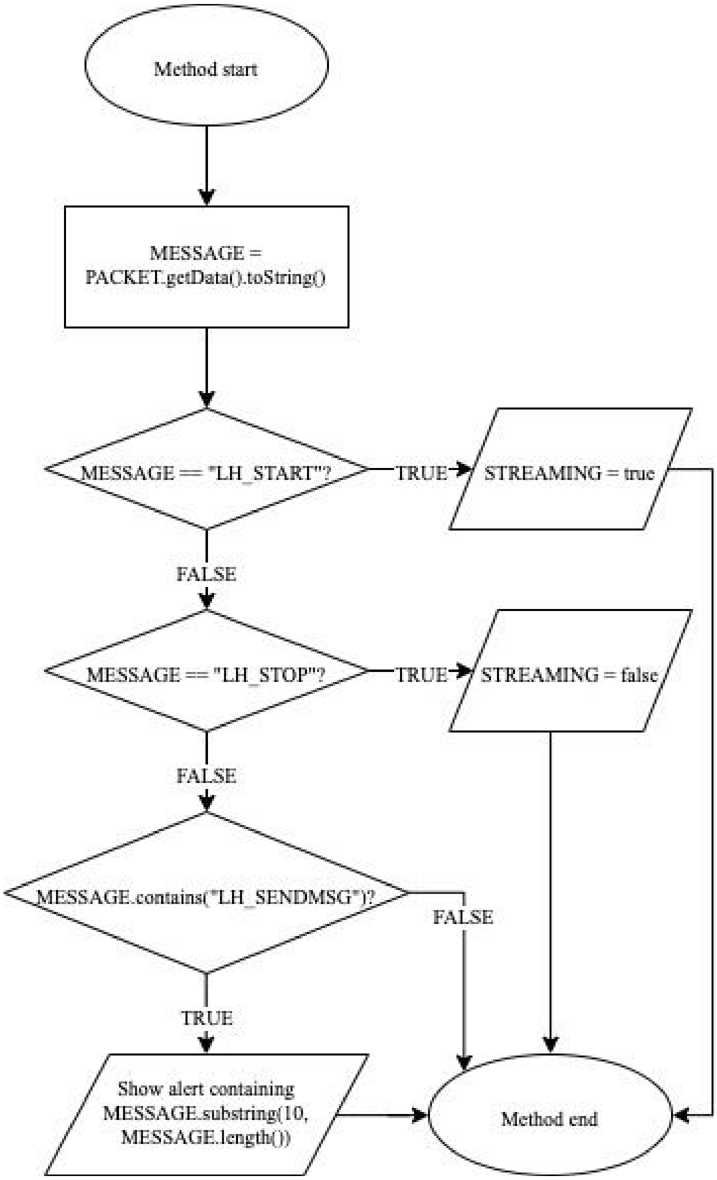
This method works by checking whether the client is able to send and receive validation packets.

Discover server in Client:


Method Start
NETWORK_INTERFACES = NetworkInterface.getNetworkInterfaces()
NETWORK_INTERFACES.hasMoreElements()?
FALSE
Receive packet
TRUE
CURRENT_INTERFACE = NETWORK_INTERFACES.nextElement()
Socket timed out?
TRUE
DISCOVERED = false
FALSE
TRUE
FALSE
CURRENT_INTERFACE.isLoopback()?
PACKET.getData().toString() == "LH_DISCOVER_RESPONSE"?
TRUE
FALSE
INTERFACE_ADDRESSES = CURRENT_INTERFACE.getInterfaceAddresses()
DISCOVERED = true
Method end
I = 0
FALSE
I < INTERFACE_ADDRESSES.length?
TRUE
Send packet to INTERFACE_ADDRESSES[I]
I = I + 1

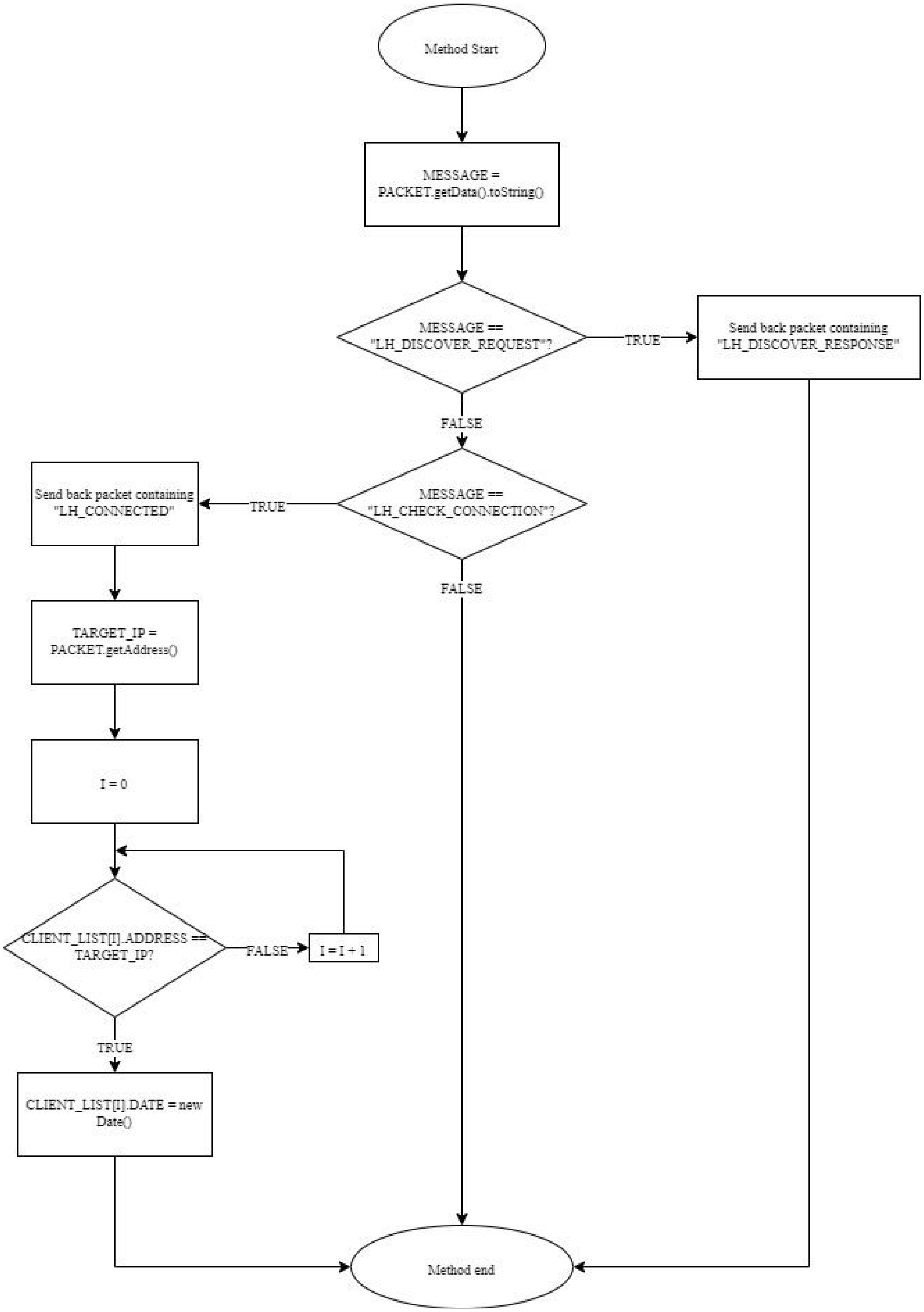
This method works by sending a packet to all devices on the network. The server sends back the correct response. It allows the address of the server to be discovered.

Parse commands in Client:


Method start
MESSAGE = PACKET.getData().toString()
MESSAGE == "LH_START"?
TRUE
STREAMING = true
FALSE
MESSAGE == "LH_STOP"?
TRUE
STREAMING = false
FALSE
MESSAGE.contains("LH_SENDMSG")?
FALSE
TRUE
Show alert containing MESSAGE.substring(10, MESSAGE.length())
Method end

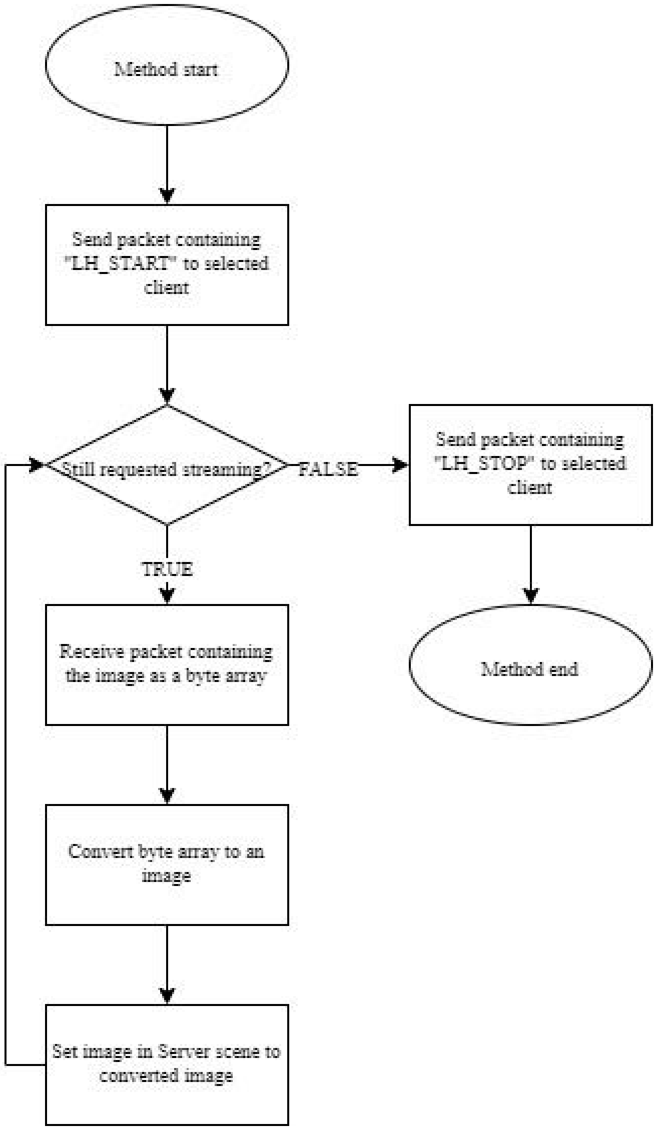
This method works by comparing the packet data to different commands, then executing the appropriate action.

Parse packets in DiscoveryHandler:


Method Start
MESSAGE = PACKET.getData().toString()
MESSAGE == "LH_DISCOVER_REQUEST"?
TRUE
Send back packet containing "LH_DISCOVER_RESPONSE"
FALSE
MESSAGE == "LH_CHECK_CONNECTION"?
TRUE
Send back packet containing "LH_CONNECTED"
FALSE
TARGET_IP = PACKET.getAddress()
I = 0
CLIENT_LIST[I].ADDRESS == TARGET_IP?
FALSE
I = I + 1
TRUE
CLIENT_LIST[I].DATE = new Date()
Method end

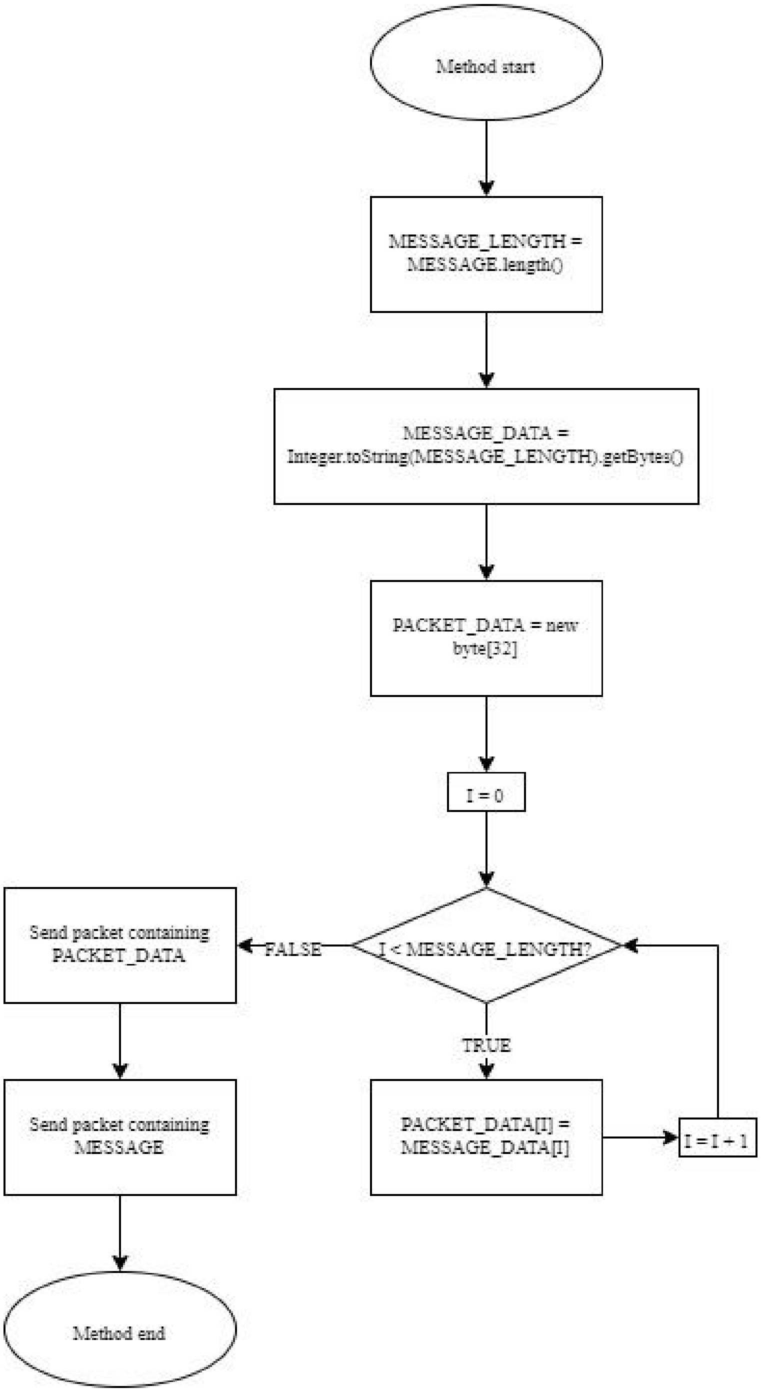
This method works by comparing the packet data to several different messages, then executing the proper action.

Stream client in Server:


Method start
Send packet containing "LH_START" to selected client
Still requested streaming?
FALSE
TRUE
Send packet containing "LH_STOP" to selected client
Method end
Receive packet containing the image as a byte array
Convert byte array to an image
Set image in Server scene to converted image

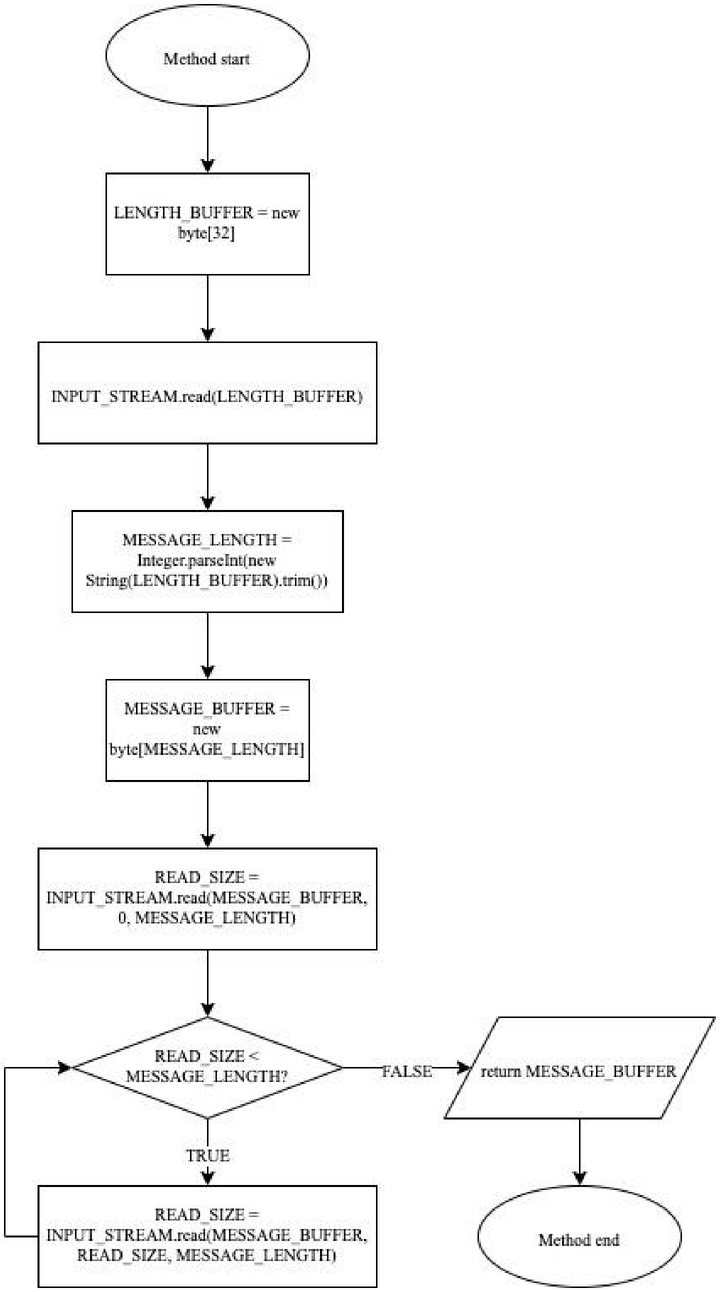
This method works by sending packets to tell the Client when to start or stop streaming. While streaming, it converts the byte array from the stream into a displayed image.

Send packets in Client:


Method start
MESSAGE_LENGTH = MESSAGE.length()
MESSAGE_DATA = Integer.toString(MESSAGE_LENGTH).getBytes()
PACKET_DATA = new byte[32]
I = 0
I < MESSAGE_LENGTH?
FALSE
TRUE
Send packet containing PACKET_DATA
PACKET_DATA[I] = MESSAGE_DATA[I]
I = I + 1
Send packet containing MESSAGE
Method end

This method works by first sending the length of the message in a 32-byte array before sending the message.

Receive packets in Server:


Method start
LENGTH_BUFFER = new byte[32]
INPUT_STREAM.read(LENGTH_BUFFER)
MESSAGE_LENGTH = Integer.parseInt(new String(LENGTH_BUFFER).trim())
MESSAGE_BUFFER = new byte[MESSAGE_LENGTH]
READ_SIZE = INPUT_STREAM.read(MESSAGE_BUFFER, 0, MESSAGE_LENGTH)
READ_SIZE < MESSAGE_LENGTH?
FALSE
return MESSAGE_BUFFER
TRUE
READ_SIZE = INPUT_STREAM.read(MESSAGE_BUFFER, READ_SIZE, MESSAGE_LENGTH)
Method end

This method works by converting the first 32 bytes from the stream into an integer, then reading that many bytes from the stream. This must be implemented since the input stream only allows around 2^{15} bytes to be read at a time, while the packets containing screen captures are approximately 2^{18} bytes large.

Test Plan

Nature of Test	Expected Results	Example
Test screen streaming	Screen of a client shows up in GUI	User requests a client to stream Server displays screen
Test button functionality	Clicking Start/Stop starts/stops streaming	User clicks on start Server displays screen until stop button is clicked
Test automatic client list updating	Username of client shows up in client list upon connection	Client connects to the server Server displays client’s username in table
Test switching between clients	Screen of selected client shows up	User clicks on another client Server displays selected client’s screen
Test sending messages to client	Alert pops up on client’s screen containing message	User inputs message Message displayed on the client’s screen
Test automatic client connection to the server	Client username shows up without intervention	Server starts up Client automatically connects Server updates client list
Test client clean-up upon client disconnection	Client disappears from client list, streaming is halted	Client disconnects from server Server program clears disconnected client’s username from client list
Test connection validation and automatic clean-up	After 2 seconds of unresponsiveness, client is removed from the client list	Client disconnects from the internet Server clears disconnected client’s username from client list after ~2 seconds