**Phase 2 DesignFile**

**Author and Title:**

**Title:** Design File for Phase 2

**Group 6 Members:**  
Victor  
Samonorom  
Adarsh  
Nishchay

**Purpose of The Phase:**

Purpose is to provide reliable data transfer (RDT) service assuming that the underlying layer is perfectly reliable using UDP connection developed or not. An image file will be broken down into packets and be sent from a client to a server.

**Code Explanation:**

**# serverUDP.py**

1. **serverUDP.py Library and Parameter Setup:**

*# Import socket and time library*import socket  
import time  
  
*# Configuration parameters for server*UDP\_IP = "localhost" *# Could use "127.0.0.1"*UDP\_PORT = 8080 *# Assigned port # to communicate*PACKET\_SIZE = 2048 *# Size of each packet to send (can be adjusted)*SLEEP\_TIME = 1 *# Intervals to prevent overloading*

1. On “serverUDP.py”, first, two libraries are imported (socket and time). Socket is to allow for communications through the network, and time allows us to use intervals to prevent bottleneck. This is done using “import socket” and “import time”.
2. Next, parameters are setup. First, using “localhost” allow us to get the machine’s local IP address, but using “127.0.0.1” would also work due to it being the home IP address. Either one is used to assign the machine’s IP address to UDP\_IP.
3. UDP\_PORT is used to assign a port number to allow direct communication between the client and server.
4. PACKET\_SIZE is used to determine how many bits will each packet contain. The size can be changed to the user preference, but the larger the packet size, the slower the transfer speed will be.
5. SLEEP\_TIME is used as a 1 second interval for another function to prevent bottleneck on the server while transferring the packets.
6. **serverUDP.py Socket Setup:**

*# Create a UDP socket for the server*Serversock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
  
*# Bind the socket to a specific address and port*server\_address = (UDP\_IP, UDP\_PORT)  
Serversock.bind(server\_address)

1. “Serversock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)”uses the socket function to create a UDP socket for the client, allowing it to communicate with clients/servers.
2. “server\_address = (UDP\_IP, UDP\_PORT)” and “Serversock.bind(server\_address)are used to bind the socket to a specific address and port for communication.
3. A print statement will notify the user that the server is ready, indicating that clientUDP.py should be ran next.
4. **serverUDP.py File Receiving Loop:**

*# Print message indicating server is ready, notifying the user to run the client*print("Server is ready to receive the image")  
  
*# Open the image file for writing*with open("received\_image.jpg", "wb") as image\_file:  
 while True:  
 *# Receive data from the client* data, client\_address = Serversock.recvfrom(PACKET\_SIZE)  
  
 *# Write the received data to the image file* received\_image = image\_file.write(data)  
  
 *# Print message for each packet received* print("A chunk of the image was received as a packet")  
  
 *# Wait for the next Packet of data* time.sleep(SLEEP\_TIME)

1. Using “with open("received\_image.jpg", "wb") as image\_file:”, the packets being sent from the client to the server will be used to recreate the image into another file named “received\_image.jpg”. The command “wb” will write the bits into the new jpg file.
2. “data, client\_address = Serversock.recvfrom(PACKET\_SIZE)” is used to receive the packet from the client.
3. “received\_image = image\_file.write(data)” will write the data to the received\_image.jpg file.
4. Each time a packet is received, a print statement will notify the user. Indicating more of the image has been sent.
5. “time.sleep(SLEEP\_TIME)”, will use a 1 second interval to prevent bottleneck.

**# clientUDP.py**

1. **clientUDP.py Library and Parameter Setup:**

*# Import socket and time library*import socket  
import time  
  
*# Configuration parameters for client*UDP\_IP = "localhost" *# Could use "127.0.0.1"*UDP\_PORT = 8080 *# Assigned port # to communicate*PACKET\_SIZE = 2048 *# Size of each packet to send (can be adjusted)*SLEEP\_TIME = 1 *# Intervals to prevent overloading*

1. The setup for “clientUDP.py’s library and parameter is similar to “serverUDP.py”.
2. On “clientUDP.py”, first, two libraries are imported (socket and time). Socket is to allow for communications through the network, and time allows us to use intervals to prevent bottleneck. This is done using “import socket” and “import time”.
3. Next parameters are setup. First, using “localhost” allow us to get the machine’s local IP address, but using “127.0.0.1” would also work. Either one is used to assign the machine’s IP address to UDP\_IP.
4. UDP\_PORT is used to assign a port number to allow direct communication between the client and server.
5. PACKET\_SIZE is used to determine how many bits will each packet contain. The size can be changed to the user preference, but the larger the packet size the slower the transfer speed will be.
6. SLEEP\_TIME is used as a 1 second interval to prevent bottleneck on the server while transferring the packets.
7. **clientUDP.py Socket Setup:**

*# Create a UDP socket for the client*Clientsock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
  
*# Uses timeout function from time library to prevent overload*Clientsock.settimeout(SLEEP\_TIME)  
  
*# Connect to the server*server\_address = (UDP\_IP, UDP\_PORT)  
Clientsock.connect(server\_address)

1. “Clientsock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)”uses the socket function to create a UDP socket for the client, allowing it to communicate with others.
2. “Clientsock.settimeout(SLEEP\_TIME)” sets up the 1 second interval between each packet transfer.
3. “server\_address = (UDP\_IP, UDP\_PORT)” and “Clientsock.connect(server\_address)” are used to connect the client to the server.
4. **clientUDP.py Image Transfer:**

*# Open the image file for reading  
# Replace "bubbles.jpg" for another file image if desired.*with open("bubbles.jpg", "rb") as image\_file:  
 *# Read the file in Packets* Packet = image\_file.read(PACKET\_SIZE)  
  
 while Packet:  
 *# Send the Packet to the server* sent = Clientsock.send(Packet)  
  
 *# Wait for the next Packet* print("A chunk of the image was sent as a packet ")  
 time.sleep(SLEEP\_TIME)  
  
 *# Read the next Packet from the file* Packet = image\_file.read(PACKET\_SIZE)  
  
print("image was sent") *# Validate the send operation on screen  
# Close the socket when complete*Clientsock.close()

1. Using “with open("bubbles.jpg", "rb") as image\_file:” opens up the image file “bubbles.jpg” from the same local folder as “clientUDP.py”. using the command “rb” (read bits), will allows us to break down the image into bits.
2. “Packet = image\_file.read(PACKET\_SIZE)” set the bits into packets.
3. The while loop “While Packet:” will continuously send the packets to the server using “sent = Clientsock.send(Packet)”. Meanwhile, a print statement will tell the user whenever a packet is sent to the server.
4. “time.sleep(SLEEP\_TIME)” will make use of the 1 second interval to prevent bottleneck.
5. Then, the next set of bits from the image is sent to “Packet” using “Packet = image\_file.read(PACKET\_SIZE)” again and, it will cycle till all the bits from the image is sent.
6. When the image has been completely sent, a print state will notify the user that it has been complete.
7. Lastly, the client’s socket is closed using “Clientsock.close()”.

**Execution Example:**

1. **Default “bubble.jpg” in Local Folder:**

A default image will be placed in the same local folder as clientUDP.py and serverUDP.py. in this case, we are using “bubbles.jpg”  
  
A picture containing bubble

Description automatically generated

1. **Server Running:**

The Server is running in PyCharm, and the print statement “Server is ready to receive the image” notify the user that the server is ready. Indicating to run “clientUDP.py”

Graphical user interface, text, application, email

Description automatically generated

1. **Client Running:**

The client is now running and sending packets to the server.

Graphical user interface, text, application

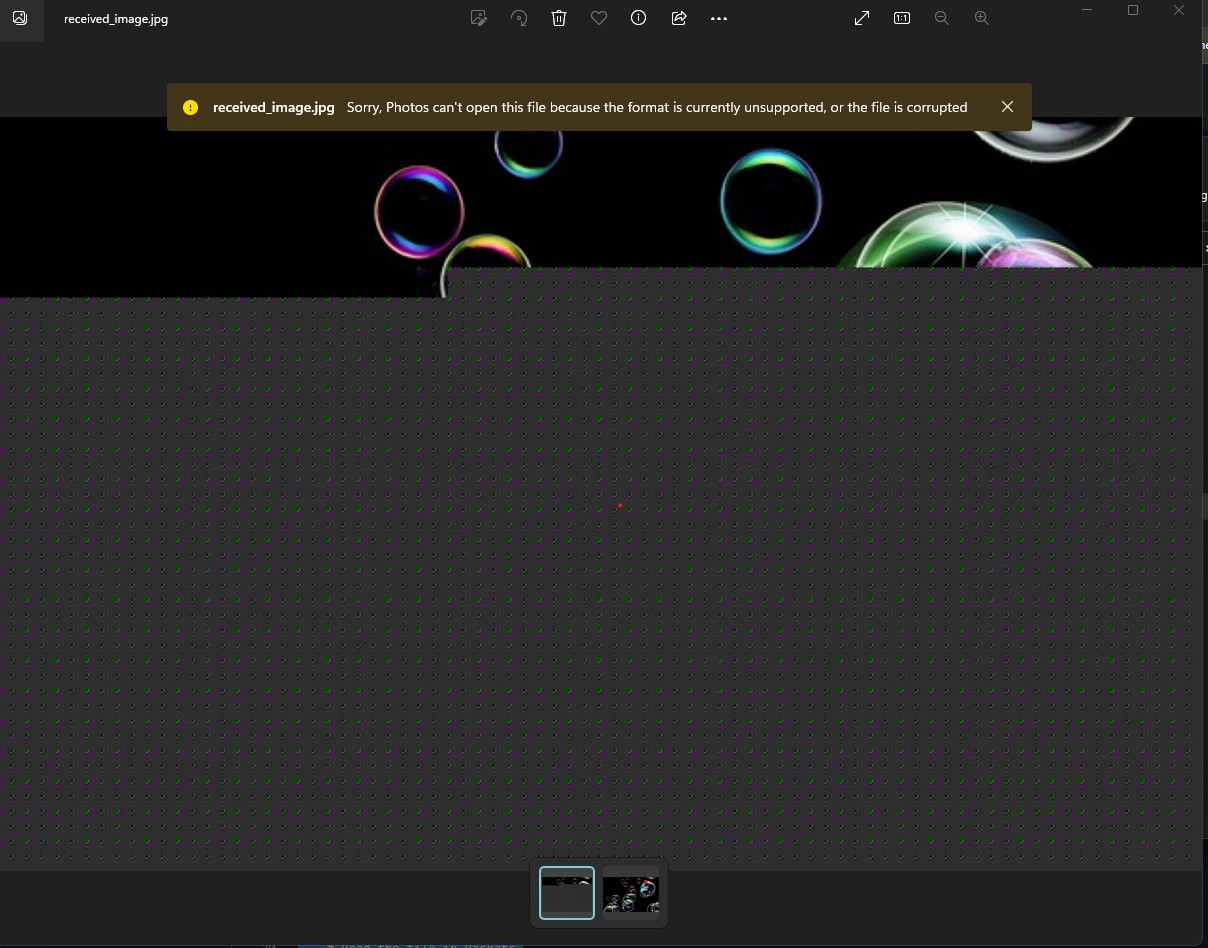
Description automatically generated

**4. Server Receiving Packets:**

The Server print statements for each packet received.

Graphical user interface, text, application

Description automatically generated

1. Example of Imaging being Re-created:  
   This is a screenshot of the image being partially uploaded to the new file “received\_image.jpg”  
   
2. **Completed Image:**

The Image has now been recreated as “received\_image.jpg”

A picture containing bubble

Description automatically generated