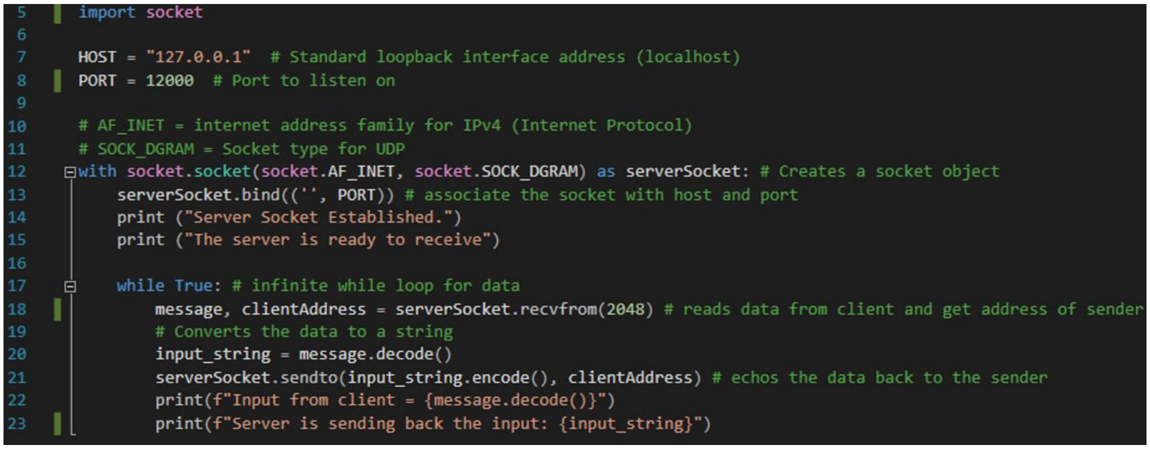
UDP Message and File Transfer with RDT 1.0

- **Author:** Parthaw Goswami 02196346

- **Introduction**
  In Phase 1, the objective is to develop a UDP ECHO Client-Server application that facilitates the exchange of messages between a client and a server. This initial phase serves as an introduction to the User Datagram Protocol (UDP), enabling us to gain hands-on experience in setting up and managing UDP-based client-server communication. The project also evolves to support the transfer of .bmp image files over the UDP protocol with RDT 1.0. Despite UDP's inherent lack of reliability, this phase focuses on implementing mechanisms to ensure reliable data transfer, assuming that the underlying layers function flawlessly. Through this process, we gained a deeper understanding of both network programming concepts and the challenges associated with ensuring data integrity over a connectionless protocol like UDP.

- **Code Explanation of Phase 1(a)**
  NOTE: The word "Hello" shown in the program screenshots does not align with the specifications outlined in the project description. Although the program files have been updated to reflect "HELLO" as required, the screenshots still display the previous version with "Hello".

  UDP Server



```
5   import socket
6
7   HOST = "127.0.0.1"  # Standard loopback interface address (localhost)
8   PORT = 12000  # Port to listen on
9
10  # AF_INET = internet address family for IPv4 (Internet Protocol)
11  # SOCK_DGRAM = Socket type for UDP
12  with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as serverSocket: # Creates a socket object
13      serverSocket.bind(('', PORT)) # associate the socket with host and port
14      print ("Server Socket Established.")
15      print ("The server is ready to receive")
16
17      while True: # infinite while loop for data
18          message, clientAddress = serverSocket.recvfrom(2048) # reads data from client and get address of sender
19          # Converts the data to a string
20          input_string = message.decode()
21          serverSocket.sendto(input_string.encode(), clientAddress) # echos the data back to the sender
22          print(f"Input from client = {message.decode()}")
23          print(f"Server is sending back the input: {input_string}")
```

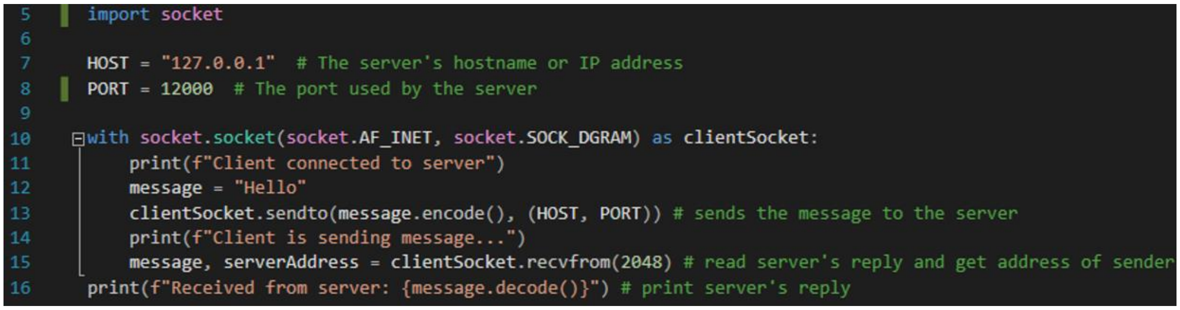
  The code implements a UDP Server designed to facilitate an echo communication with a client. In this setup, the server listens for incoming messages from the client and sends them back unchanged, demonstrating the fundamental principles of UDP (User Datagram Protocol) communication.
  At the beginning of the code, line 3 defines the variable HOST, which represents the server's IP address. The address is set to "127.0.0.1", which is the standard loopback

address. Line 4 defines the PORT variable, specifying the port number on which the server will listen for incoming messages from the client. In case of UDP, the socket type is of AF_INET which as provided in the comments on line 6 is the internet address family for IPv4. UDP also uses a socket type called SOCK_DGRAM to operate in UDP.

Line 8 is the start of the “with” statement that uses the “socket.socket()” to create a socket object which is set to “serverSocket”. The “serverSocket.bind((‘ ’, Port))” binds the socket to the IP address of the HOST as well as the port number as shown in line 9 of the program code. Once the socket is bound, the server prints confirmation messages to indicate successful initialization.

The next functionality of the server is implemented within a while loop, allowing it to continuously listen for incoming messages. In line 14, the server waits to receive data from a client using the serverSocket.recvfrom(2048) function. This function retrieves both the incoming message and the address of the client that sent it, storing them in the variables message and clientAddress. After receiving the data, the server decodes the message from bytes to a string using the message.decode() method in line 16, storing the result in a new variable for further processing. To complete the echo functionality, the server sends the received message back to the client using the serverSocket.sendto(input_string.encode(), clientAddress) function in line 17. Finally, the last two print statements print the received data “HELLO” and the data that is being sent back to the client which is still “HELLO” since this is an echo server. This server is always open to connect with new clients until the program is closed.

UDP Client

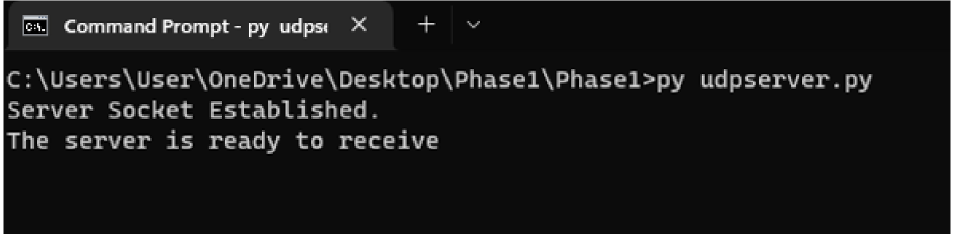


```
5    import socket
6
7    HOST = "127.0.0.1"  # The server's hostname or IP address
8    PORT = 12000  # The port used by the server
9
10   with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as clientSocket:
11       print(f"Client connected to server")
12       message = "Hello"
13       clientSocket.sendto(message.encode(), (HOST, PORT)) # sends the message to the server
14       print(f"Client is sending message...")
15       message, serverAddress = clientSocket.recvfrom(2048) # read server's reply and get address of sender
16   print(f"Received from server: {message.decode()}") # print server's reply
```

The image above displays the code for the UDP Client program. In this code, lines 7 and 8 define the HOST and PORT variables, which specify the server's IP address and the port number that the client will connect to. The program utilizes a with statement to create and manage a socket object, which binds to the host (server IP address) and port number. “AF_INET” is needed as it is the internet address family for IPv4. The socket type for UDP, “SOCK_DGAM” is also needed for both client and server and is used in both programs.

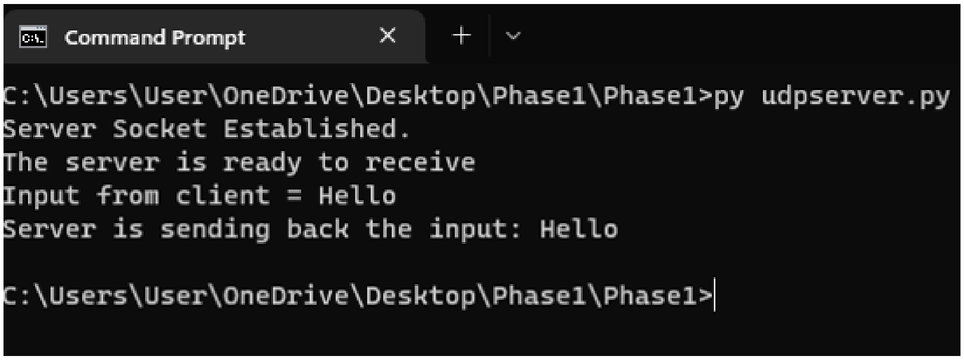
Using the with statement block, the client sends a message, "HELLO", to the server using the clientSocket.sendto() function. This function transmits the message to the server by referencing the address and port stored in the HOST and PORT variables. After sending the message, the client waits to receive an echoed response from the server. This is accomplished using the clientSocket.recvfrom() function, which retrieves the server's response.

- **Execution Example of Phase 1(a)**
  UDP Server Start Up



```
C:\Users\User\OneDrive\Desktop\Phase1\Phase1>py udpserver.py
Server Socket Established.
The server is ready to receive
```

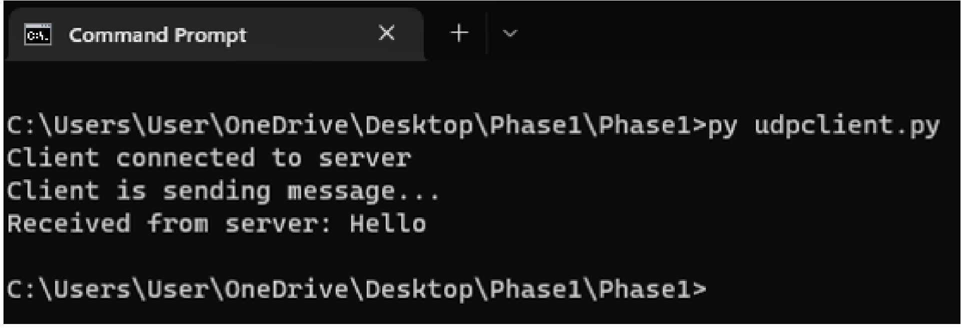
UDP Server After Echo



```
C:\Users\User\OneDrive\Desktop\Phase1\Phase1>py udpserver.py
Server Socket Established.
The server is ready to receive
Input from client = Hello
Server is sending back the input: Hello

C:\Users\User\OneDrive\Desktop\Phase1\Phase1>
```

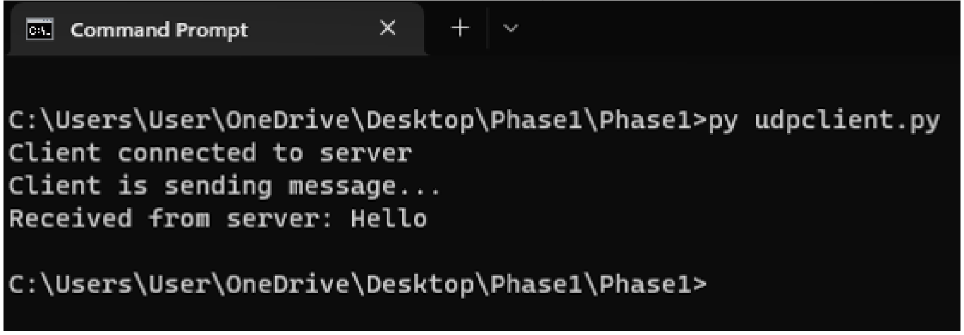
UDP Client Connection/ Start Up



```
C:\Users\User\OneDrive\Desktop\Phase1\Phase1>py udpclient.py
Client connected to server
Client is sending message...
Received from server: Hello

C:\Users\User\OneDrive\Desktop\Phase1\Phase1>
```

UDP Client Connection/ Message Sending

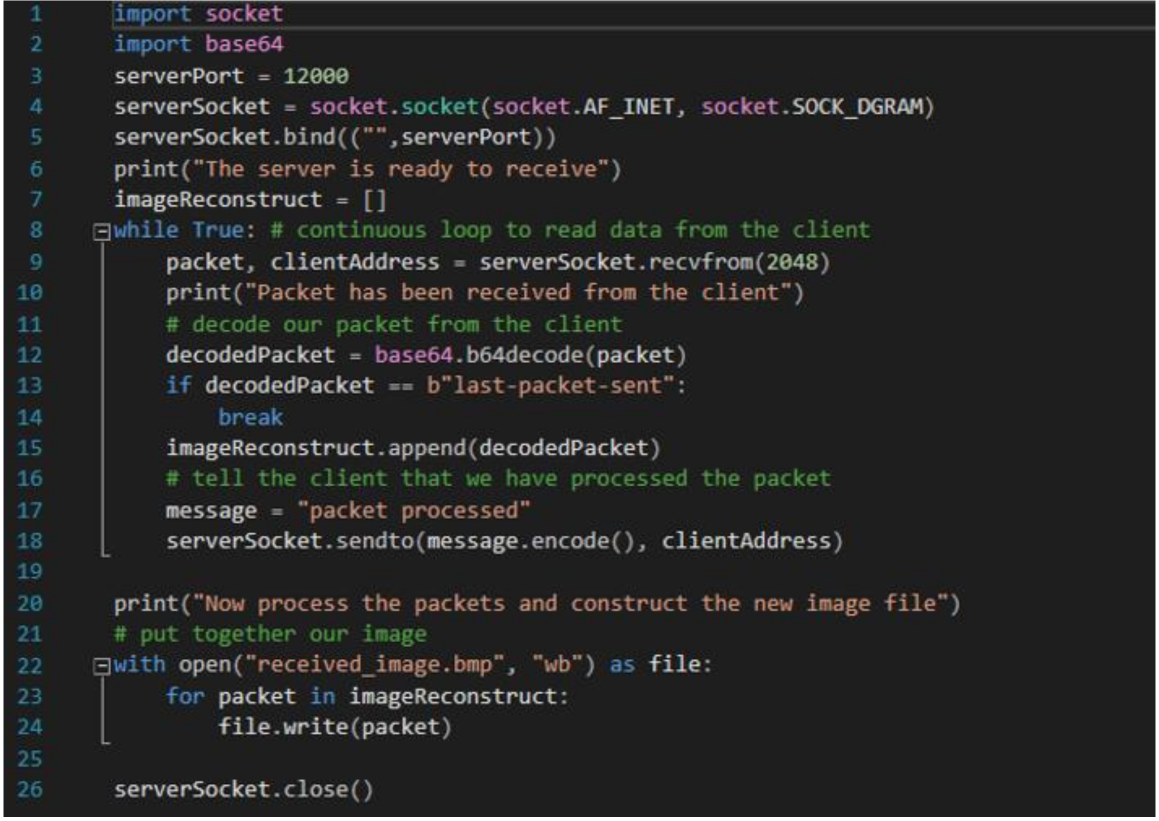


```
C:\Users\User\OneDrive\Desktop\Phase1\Phase1>py udpclient.py
Client connected to server
Client is sending message...
Received from server: Hello

C:\Users\User\OneDrive\Desktop\Phase1\Phase1>
```

- **Code Explanation of Phase 1(b)**

  ImageServer.py



```
1   import socket
2   import base64
3   serverPort = 12000
4   serverSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
5   serverSocket.bind(("",serverPort))
6   print("The server is ready to receive")
7   imageReconstruct = []
8   while True: # continuous loop to read data from the client
9       packet, clientAddress = serverSocket.recvfrom(2048)
10      print("Packet has been received from the client")
11      # decode our packet from the client
12      decodedPacket = base64.b64decode(packet)
13      if decodedPacket == b"last-packet-sent":
14          break
15      imageReconstruct.append(decodedPacket)
16      # tell the client that we have processed the packet
17      message = "packet processed"
18      serverSocket.sendto(message.encode(), clientAddress)
19
20  print("Now process the packets and construct the new image file")
21  # put together our image
22  with open("received_image.bmp", "wb") as file:
23      for packet in imageReconstruct:
24          file.write(packet)
25
26  serverSocket.close()
```

Lines 1-7 first imports two modules, “socket” and “base64”. These modules are used to create a socket connection and encode/decode binary data in base 64 format respectively. The server is then set to listen on port 12,000 as well as specifying the server to use the IPv4 address in addition to providing the specific parameter “SOCK_DGRAM” for UDP protocol. The server then uses the bind method to listen on available network interfaces and port 12000. Once the server socket is established, it prints out a validation message.

Lines 7-18 initializes an empty list “imageReconstruct” to be used to reconstruct the .bmp file (image) sent by the client. The while loop uses the “recvfrom” method to continuously receive packets from the client. The received packet is then decoded and appended to the empty list for reconstruction. Lastly once the decoding is done, a validation message is sent to the client.

In Lines 20-26, the with statement is used to open a new binary file named "received_image.bmp". The "wb" argument to the open function specifies that the file should be opened in write binary mode. The for loop then iterates over the packets stored in the “imageReconstruct” list, and the write method of the file object is called in each iteration to write the packet to the file. Finally, the close method of the server socket is called to close the socket.

ClientServer.py

```
1
2   import socket
3   import os
4   import base64
5
6   # ************ function definitions ************
7   def MakeOutputPacketArray(fileToRead):
8       #all of the packets that we are going to send out
9       packetsToSend = []
10      #open our file that is in the same directory of our python script
11      with open(fileToRead, "rb") as file:
12          #indefinitely loop our file
13          while True:
14              packet = file.read(1024)
15              #once we have reached the end of our .bmp file, we need to exit our while loop
16              if not packet:
17                  break
18              #create our array of packets to output
19              packetsToSend.append(packet)
20
21      return packetsToSend
22
23
24
25  #setup the client UDP
26  #we will use a generic establishment, thus will work on any pc
27  serverName = socket.gethostname()
28  print("Host client name: " + serverName)
29  serverHostIP = socket.gethostbyname(serverName)
30  print("Host server IP: " + serverHostIP)
31  HOST = serverHostIP
32  clientPort = 12000
33  clientSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
34  print(f"Client is now connected to the server")
35
36
37  #server connection has now been established. Now that we have our server working
38  #we should now work on splitting up our image
39
40  #source of image: https://people.math.sc.edu/Burkardt/data/bmp/bmp.html
41
42  #fileDir = os.environ.get("PythonApplication1")
43  fileName = "tricolor.bmp"
44  #call our make packet function
45  outputImage = MakeOutputPacketArray(fileName)
46  count = 0
47  #all of our packets are now in our outputImage array, we now need to parse this array and send each packet one by one.
48  for packet in outputImage:
49      #base64 must be used to encode each packt before sending them over, base64 will be used to decode each packet on the server side as well
50      #we need to send packets 1 by 1. We will wait for a message back from the server stating that the packet has been processed, this will let us know we can move on to the next packet to send
51      #if this message does not state the packet has been processed, we will continue to wait and listen to the server to hear back that the packet that was sent has been processed
52      #start off by sending 1 packet over to the server
53      encodePacket = base64.b64encode(packet)
54      clientSocket.sendto(encodePacket, (serverName, clientPort))
55      messageFromServer, serverAddress = clientSocket.recvfrom(2048)
56      messageFromServer = messageFromServer.decode()
57      #we now must wait for our server to tell us that it has processed our packet and then we can move on to our next packet
58      while messageFromServer != "packet processed":
59          messageFromServer = ""
60          messageFromServer, serverAddress = clientSocket.recvfrom(2048)
61          messageFromServer = messageFromServer.decode()
62      count += 1
63
64  finalMessage = b"last-packet-sent"
65  encodeFinalMessage = base64.b64encode(finalMessage)
66  clientSocket.sendto(encodeFinalMessage, (serverName, clientPort))
67  print("packets sent {count}")
68  print("all packets sent, shutting down client")
69  clientSocket.close()
```

Lines 7-21 is a function called "MakeOutputPacketArray". This function takes a file name as input and returns a list of packets that can be sent over a network. It first opens the file that is in the same directory as the script, then it reads the file in 1024-byte chunks and appends each chunk into an array or list called packetsToSend. The function returns the array afterwards.

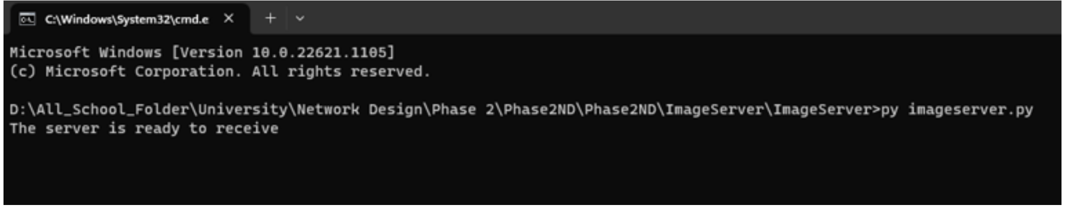
Lines 25-34 sets up a UDP client by retrieving the host name and IP address of the client, setting the client's port number to 12000, and creating a socket using the IPv4 address family and UDP protocol. The client is then connected to the server, as indicated by a message printed at the end.

Lines 43-62 sends a .bmp image file to the server. First, the image is broken down into smaller chunks using the “MakeOutputPacketArray” function. Each packet is then encoded using the base 64 encoding before being sent to the server. The client waits for a confirmation response from the server that each packet is received and processed before sending the next chunk of packet. The number of packets is recorded in the “count” variable.

Lines 64-69 sends the final message once the last packet is sent and closes the client socket soon after. The number of packets sent is also printed as well as a print statement to show that the client is closing.

- **Execution Example of Phase 1(b)**

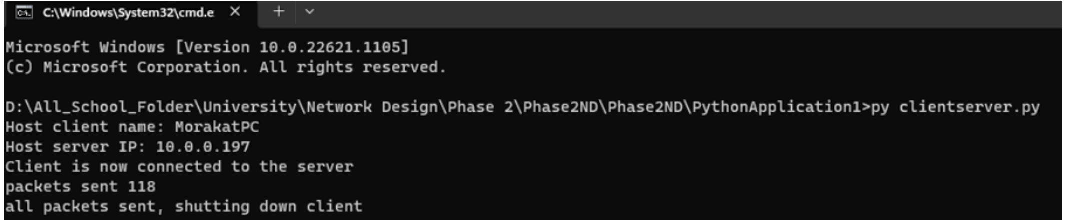
Server Startup: the startup of the server to establish the server socket.



```
Microsoft Windows [Version 10.0.22621.1105]
(c) Microsoft Corporation. All rights reserved.

D:\All_School_Folder\University\Network Design\Phase 2\Phase2ND\Phase2ND\ImageServer\ImageServer>py imageserver.py
The server is ready to receive
```

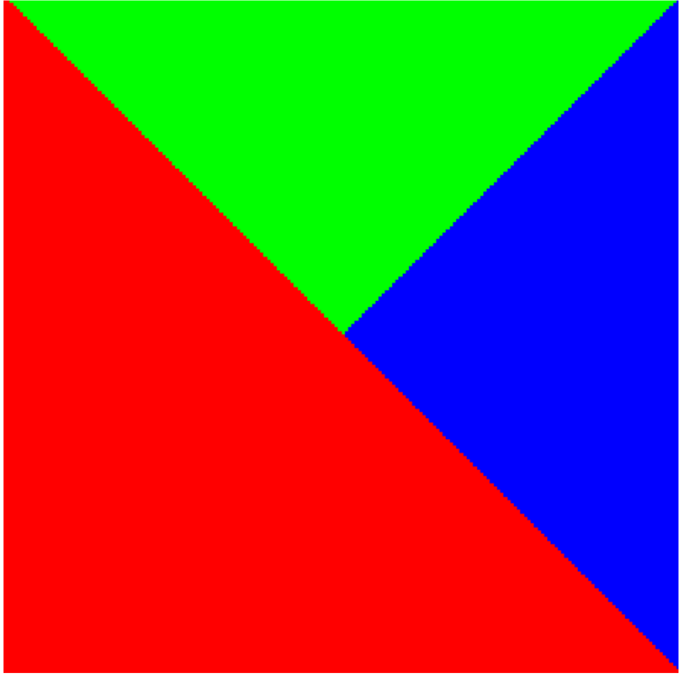
Client Startup/ Packet Transfer



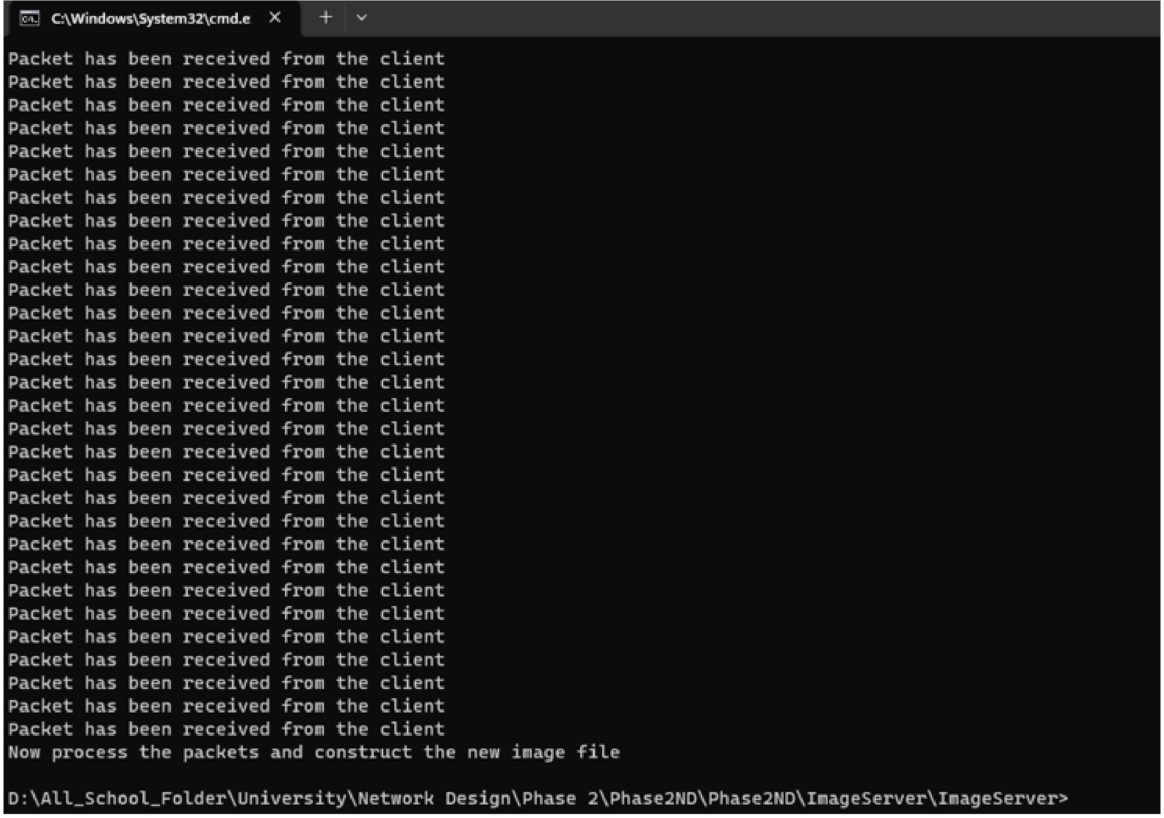
```
Microsoft Windows [Version 10.0.22621.1105]
(c) Microsoft Corporation. All rights reserved.

D:\All_School_Folder\University\Network Design\Phase 2\Phase2ND\Phase2ND\PythonApplication1>py clientserver.py
Host client name: MorakatPC
Host server IP: 10.0.0.197
Client is now connected to the server
packets sent 118
all packets sent, shutting down client
```

Original .bmp Image file “tricolor.bmp” for Transfer



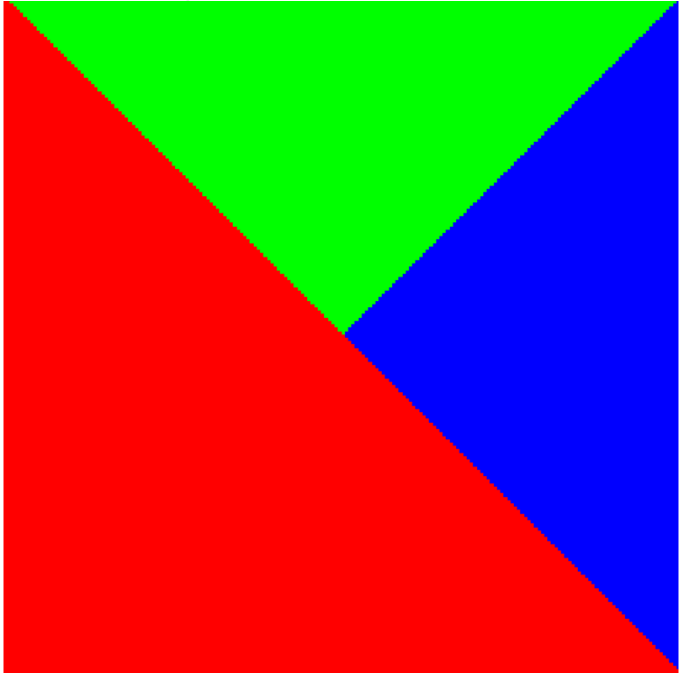
Packet Transfer Acknowledgement



```
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Packet has been received from the client
Now process the packets and construct the new image file

D:\All_School_Folder\University\Network Design\Phase 2\Phase2ND\Phase2ND\ImageServer\ImageServer>
```

Received Image after Transfer



Above is the reconstruction of the original image label received_image.bmp. There is no packet loss as the entire image has been reconstructed perfectly.

- **Block diagram for implementing the RDT 1.0 protocol for reliable BMP file transfer**