**Design File**

**Title and Authors:**

Title: Phase 2  
Michael Burton, Jocelyn Frechette, Jesse Haynes-Lewis

**Purpose of This Phase:**

In this phase of the project, we learned about the basic principles used to provide non-pipelined reliable data transfer over a perfectly reliable channel.

**Code Explanation:**

**UDPClient.py**

A screenshot of a computer program

Description automatically generated

UDPClient.py, Make\_Packet function, this function will take in the name of the .bmp file and the desired since of the packet. The file is then open using the open() function and read as a binary. Once the file is opened, the file is then read into a byte array according the packet size. With each read, each pack is appended into the packet\_list array in a while loop. Once the end of the file is read, the file is then closed and the variable packet\_list is returned.

A computer screen shot of a program

Description automatically generated

UDPClient.py, UDPClient class, two functions are defined in this class. Once is a static function that defines the name and port of the receiver and sets up the socket for the client. The function called send() is another static function that sends the data to designated server with a tuple of the server IP address and port number.

A screenshot of a computer screen

Description automatically generated

UDPClient.py, main script. This part of the UDPClient code is where the code is physical run. We first provide an instantiation of the client. The Make\_Packet function is then called to packetize the .bmp file known as LAND2.bmp with a defined packet size of 2048 bytes. The client sends over the number of packets that the server should expect and sends over the packets one by one over the UDP sockets. Once completed, the client will then send over the number of packets again to tell the server that the transmission of packets is complete.

**UDPServer.py**

A screenshot of a computer

Description automatically generated

UDPServer.py, deliver\_data function. This function takes in the bytearray of packets that were sent over from the client and the came of the output file. The output file is then opened using the open() function and is then written as a binary. Within the loop, the packets are written one by one into the file. Once the packets have been written to file, the file is closed.

A computer screen shot of a black screen

Description automatically generated

UDPServer.py, UDPserver class. Within in this class are two static functions that define the functionality of the UDPServer. This code block is split into two parts. The first, as shown above is a static function that defines the server IP address and port number. Next the socket for server is defined.

A computer screen shot of a computer code

Description automatically generatedA computer screen shot of a program code

Description automatically generatedUDPserver.py, receive() function within the UDPServer class. This function takes in the instantiation of the UDP Server and size of the buffer receiving data. First the server binds to the defined socket and then prints to console. From there the UDP server listens for an incoming message. The first received is the number of anticipated packets that was sent from the client. Once that is received, the second loop works to receive the packets until the desired number of packets is received. As the packets are received, the packet is appended to a buffer which is of the type of bytearray which is the received\_packets variable. Once completed, the last entry is deleted, as that is the end of message from the server. The received\_packets byte array is returned from the function.

A screen shot of a computer

Description automatically generated

UDPServer.py, main script. The script above is where the UDPServer is ran. As shown shown above, the IP Address and port of the server is defined. From there, an instantiation of UDPServer is initialized and received the packets via the receive() function. Once all packets are received, the deliver\_data() function is called and the received packets are written to a copy.bmp file. This file can be opened to check if packets were lost during a run of the code.

**Execution Example:**

**A screenshot of a computer

Description automatically generated**

UDPServer is started with the execution of the line python3 .\UDPServer.py, the port number is then defined to be 15024, with the IP address defined as 127.0.0.1, which is for loopback execution only.

A screenshot of a computer

Description automatically generated

UDPClient, prior to execution. The UDP client is initialized with the line python3 UDPClient.py, the port number is then defined to define the server port number.

A black and white screen

Description automatically generated

Once port was defined, the “Enter” was hit, this is the result from the UDPClient, once the packets have been sent.



Result from the UDPServer side, this screen shows some of the packet received as well as the number of packets received.

A screenshot of a computer game

Description automatically generated

Resulting copy.bmp file that was written from the UDPServer side.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated