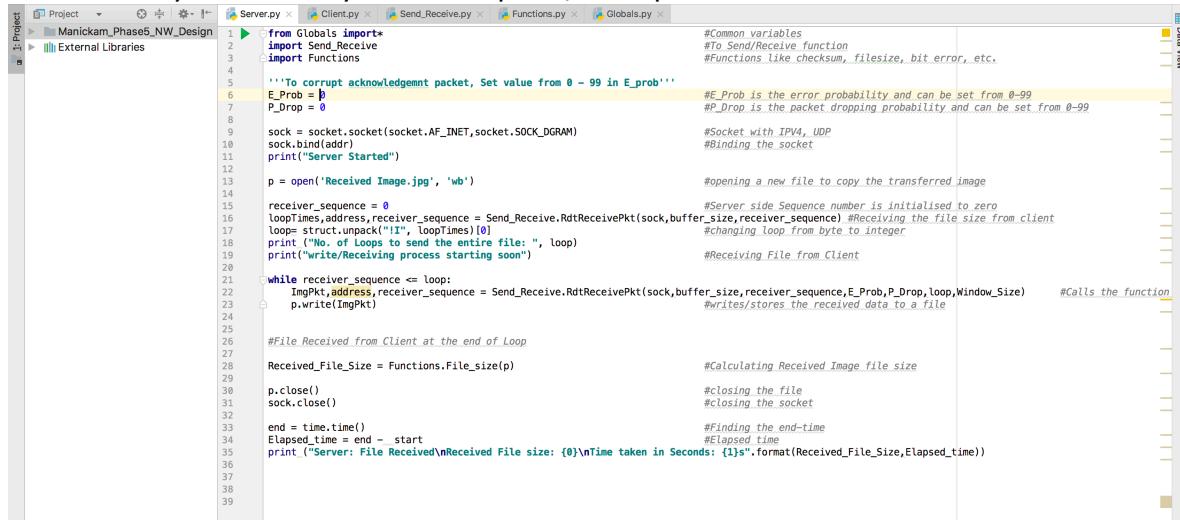


EECE 5830: NETWORK DESIGN
PROJECT PHASE 5: GO-BACK-N PROTOCOL OVER AN UNRELIABLE UDP CHANNEL
AUTHOR: ASWIN KUMAR MANICKAM, 01624359.

PROGRAM:

- Server.py has the main program for the server to receive a file, Set E_Prob OR P_Drop in server side alone if you need only data corruption/ Data packet loss.

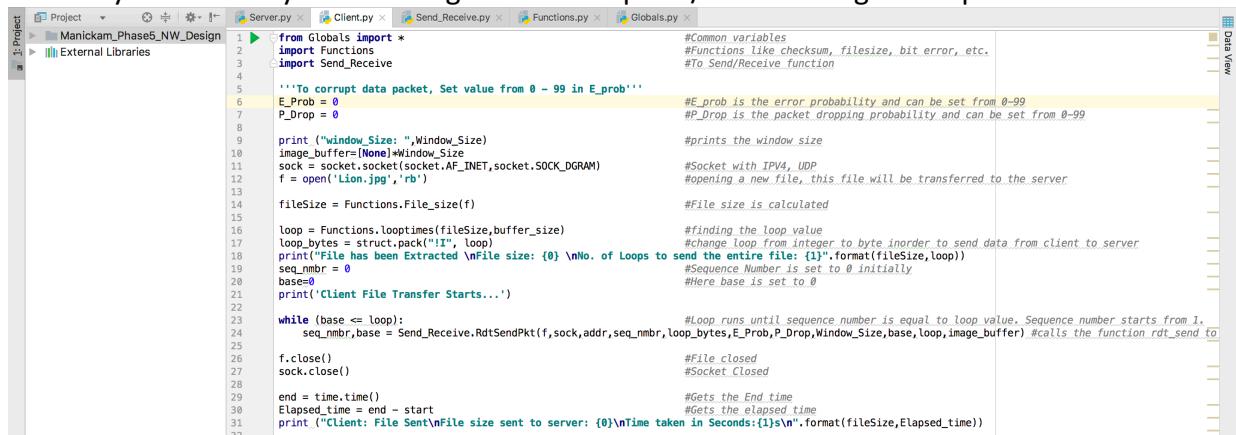


```

1  from Globals import *
2  import Send_Receive
3  import Functions
4
5  '''To corrupt acknowledgement packet, Set value from 0 - 99 in E_Prob'''
6  E_Prob = 0
7  P_Drop = 0
8
9  sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
10 sock.bind(addr)
11 print("Server Started")
12
13 p = open('Received Image.jpg', 'wb')
14
15 receiver_sequence = 0
16 loopTimes,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence)
17 loop= struct.unpack("I",loopTimes)[0]
18 print("No. of Loops to send the entire file: ", loop)
19 print("Receiving process starting soon")
20
21 while receiver_sequence <= loop:
22     ImgPkt,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence,E_Prob,P_Drop,loop,Window_Size)
23     p.write(ImgPkt)
24
25 #File Received from Client at the end of Loop
26 Received_File_Size = Functions.File_size(p)
27
28 p.close()
29 sock.close()
30
31 end = time.time()
32 Elapsed_time = end - start
33 print ("Server: File Received\nReceived File size: {} \nTime taken in Seconds: {}".format(Received_File_Size,Elapsed_time))
34
35
36
37
38
39

```

- Client.py has the main program for the client to send a file, Set E_Prob or P_Drop in client side alone if you need only Acknowledgement corruption/ Acknowledgement packet loss.



```

1  from Globals import *
2  import Functions
3  import send_Receive
4
5  '''To corrupt data packet, Set value from 0 - 99 in E_Prob'''
6  E_Prob = 0
7  P_Drop = 0
8
9  print ("window Size: ",Window_Size)
10 image_buffers=[None]*Window_Size
11 sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
12 f = open('Lion.jpg','rb')
13
14 fileSize = Functions.File_size(f)
15
16 loop = Functions.looptimes(fileSize,buffer_size)
17 loop_bytes = struct.pack("I",loop)
18 print("file has been Extracted \nfile size: {} \nNo. of Loops to send the entire file: {} \n".format(fileSize,loop))
19 seq_nbr = 0
20 base=0
21 print('Client File Transfer Starts...')
22
23 while (base <= loop):
24     seq_nbr,base = Send_Receive.RdtSendPkt(f,sock,addr,seq_nbr,loop_bytes,E_Prob,P_Drop,Window_Size,base,loop,image_buffer) #calls the function rdt_send to
25
26 f.close()
27 sock.close()
28
29 end = time.time()
30 Elapsed_time = end - start
31 print ("Client: File Sent\nFile size sent to server: {} \nTime taken in Seconds: {}".format(fileSize,Elapsed_time))
32

```

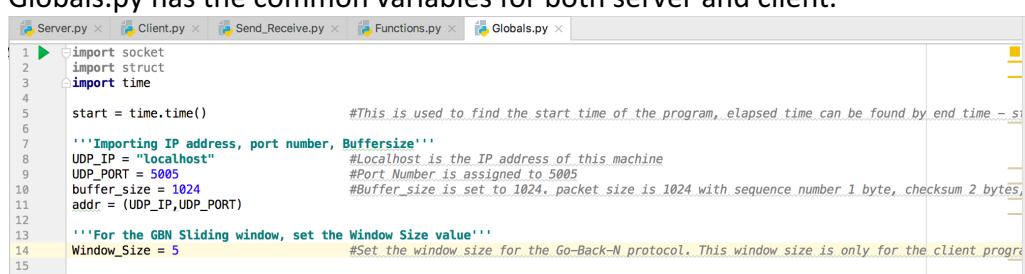
- Set E_Prob or P_Drop value from 0 to 99.

```

'''To corrupt data packet, Set value from 0 - 99 in E_Prob'''
E_Prob = 0
P_Drop = 0

```

- Globals.py has the common variables for both server and client.



```

1  import socket
2  import struct
3  import time
4
5  start = time.time() #This is used to find the start time of the program, elapsed time can be found by end_time - start
6
7  '''Importing IP address, port number, BufferSize'''
8  UDP_IP = "localhost" #localhost is the IP address of this machine
9  UDP_PORT = 5005 #Port Number is assigned to 5005
10 buffer_size = 1024 #Buffer_size is set to 1024, packet size is 1024 with sequence number 1 byte, checksum 2 bytes,
11 addr = (UDP_IP,UDP_PORT)
12
13 '''For the GBN Sliding window, set the Window Size value'''
14 Window_Size = 5 #Set the window size for the Go-Back-N protocol. This window size is only for the client program
15

```

- Also, we can set the Window Size in the Global.py. The window size refers to the sliding window size for the client side. The receiver side window size is always 1.

```
'''For the GBN Sliding window, set the Window Size value'''
Window_Size = 5 #Set the window size for the Go-Back-N protocol. This window size is only for the client program
```

- Functions.py has the functions for checksum, file size, number of time loop has to run, Bit-error condition, data corruption, make packet, extracting packet.

```

1  import math
2  import random
3  import struct
4  from Globals import *
5
6  '''This function is used to find the file size with the help of seek function.'''
7  def file_size(file):...
12
13
14  '''This function is used to find how many loops the program has to run to transfer the file.'''
15  def lopptimes(fileSize, buffer_size):...
19
20
21  '''This function updates the sequence number'''
22  def Update_Seq_num(seq_num):...
24
25
26  '''This function is used to find the Checksum for the data'''
27  def checksum(data):...
39
40
41  '''This function is used to find the Bit_Error has to happen or not'''
42  def Error_Condition(E_Prob=0):...
48
49
50  '''This Function is used to corrupt the data'''
51  def Data_Corrupt(data):...
53
54
55  '''This Function is used to Extract Data (Sequence number,checksum, data) from packet'''
56  def ExtractData(packet):...
60
61
62  '''This Function is used to make packet (Sequence number + checksum + data -> together forms a packet)'''
63  def MakePkt(seqNums,chksums,data):...
67
68
69

```

*Functions used in this project are simple and code is well commented.

- Send_Receive.py has the RDT protocol for server and client to send/receive file with sequence number, checksum along with the data.

```

1  from Globals import *
2  import Functions
3  import time
4
5
6  '''This function is the basically the client's RDT function to send the file.'''
7
8  def RdtSendPkt(udp_sock,addr,seq_nmbr,data,Data_buffer,E_Prob=0,P_Drop=0,window_size=0,base=0,loop=0):...
70
71
72
73  '''This function is the basically the Server's RDT function to receive the file.'''
74  def RdtReceivePkt (sock,buffer_size,Rx_seq_num,E_Prob=0,P_Drop=0,loop=0,Window_Size = 0):...
113
114

```

*RDT protocol is designed as per the project/protocol requirements.

PROGRAM EXECUTION:

- Set window Size, (Above screenshots) I took window size as 5.
- Run Server.py first.

```

1 from Globals import*
2 import Send_Receive
3 import Functions
4
5 """To corrupt acknowledgement packet, Set value from 0 - 99 in E_prob"""
6 E_Prob = 0
7 P_Drop = 0
8
9 sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
10 sock.bind(addr)
11 print("Server Started")
12
13 p = open('Received Image.jpg', 'wb')
14
15 receiver_sequence = 0
16 loopTimes,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence,E_Prob,P_Drop,loop,Window_Size)
17 loopTimes,address,receiver_sequence = struct.unpack("!I", loop)
18 print ("No. of Loops to send the entire file: ", loop)
19 print("Receiving process starting soon")
20
21 while receiver_sequence <= loop:
22     ImgPkt,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence,E_Prob,P_Drop,loop,Window_Size)
23     p.write(ImgPkt)
24
25
26 #File Received from Client at the end of Loop
27
28 Received_File_Size = Functions.File_size(p)
29
30 p.close()
31 sock.close()
32
33 end = time.time()
34 Elapsed_time = end - start
35 while receiver...

```

The screenshot shows the PyCharm IDE interface with the Server.py file open. The status bar at the bottom indicates "Server Started". The code implements a UDP server that receives files from a client. It uses a window size of 5 and handles packet corruption and dropping based on configuration variables E_Prob and P_Drop.

- Run Client.py next and you can see file transfer starts.

```

1 from Globals import *
2 import Functions
3 import Send_Receive
4
5 """To corrupt data packet, Set value from 0 - 99 in E_prob"""
6 E_Prob = 0
7 P_Drop = 0
8
9 print("Window Size: ",Window_Size)
10 image_buffer=(None)Window_Size
11 sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
12 f = open('Lion.jpg', 'rb')
13
14 fileSize = Functions.File_size(f)
15
16 loop = Functions.looptimes(fileSize,buffer_size)
17 loop_bytes = struct.pack("!I", loop)
18 print("File has been Extracted \nfile size: {} \nNo. of Loops to send the entire file: {} \nSequence Number is set to 0 initially".format(fileSize,loop))
19 seq_nbr = 0
20 base=0
21 print("Client File Transfer Starts...")
22
23 while (base <= loop):
24     seq_nbr,base = Send_Receive.RdtSendPkt(f,sock,addr,seq_nbr,loop_bytes,E_Prob,P_Drop,Window_Size,base,loop,image_buffer)
25
26 f.close()
27 sock.close()
28
29 end = time.time()
30 Elapsed_time = end - start
31 print ("Client: File Sent\nFile size sent to server: {} \nTime taken in Seconds:{}\n".format(fileSize,Elapsed_time))
32
33

```

The screenshot shows the PyCharm IDE interface with the Client.py file open. The status bar at the bottom indicates "Client File Transfer Starts...". The code implements a UDP client that sends a file to a server. It uses a window size of 5 and handles packet corruption and dropping based on configuration variables E_Prob and P_Drop. The client prints the file size and the time taken to send it.

- Check Server.py console again to make sure data has been transferred successfully.

```

1  from Globals import*
2  import Send_Receive
3  import Functions
4
5  '''To corrupt acknowledgement packet, Set value from 0 - 99 in E_prob'''
6  E_Prob = 0
7  P_Drop = 0
8
9  sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
10 sock.bind(addr)
11 print("Server Started")
12
13 p = open('Received Image.jpg', 'wb')
14
15 receiver_sequence = 0
16 loop,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence) #Receiving the file size from client
17 loop= struct.unpack("!I", loop)[0]
18 print ("No. of Loops to send the entire file: ", loop)
19 print("write/Receiving process starting soon") #Receiving File from Client
20
21 while receiver_sequence <= loop:
22     ImgPkt,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence,E_Prob,P_Drop,loop,Window_Size) #Calls the function
23     p.write(ImgPkt) #writes/stores the received data to a file
24
25 #File Received from Client at the end of Loop
26
27 Received_File_Size = Functions.File_size(p) #Calculating Received Image file size
28
29 p.close() #closing the file
30 sock.close() #closing the socket
31
32 end = time.time() #Finding the end-time
33 Elapsed_time = end - start #Elapsed time
34
35 while receiver...

```

Run: Server Client

Sequence Number: 632,Receiver_sequence: 632, Checksum from Client: 12648, Checksum for Received File: 12648
Sequence Number: 633,Receiver_sequence: 633, Checksum from Client: 30250, Checksum for Received File: 30250
Sequence Number: 634,Receiver_sequence: 634, Checksum from Client: 45778, Checksum for Received File: 45778
Sequence Number: 635,Receiver_sequence: 635, Checksum from Client: 3113, Checksum for Received File: 3113
Sequence Number: 636,Receiver_sequence: 636, Checksum from Client: 14365, Checksum for Received File: 14365
Sequence Number: 637,Receiver_sequence: 637, Checksum from Client: 50814, Checksum for Received File: 50814
Server: File Received
Received File size: 649413
Time taken in Seconds: 7.642054080963135s
Process finished with exit code 0

4: Run Python Console

CORRUPTION (E_Prob is set to 10, that is 10% chance of corrupting the data [Sample Output]):

Data Corruption:

```

1  from Globals import*
2  import Send_Receive
3  import Functions
4
5  '''To corrupt acknowledgement packet, Set value from 0 - 99 in E_prob'''
6  E_Prob = 10
7  P_Drop = 0
8
9  sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
10 sock.bind(addr)
11 print("Server Started")
12
13 p = open('Received Image.jpg', 'wb')
14
15 receiver_sequence = 0
16 loop,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence) #Receiving the file size from client
17 loop= struct.unpack("!I", loop)[0]
18 print ("No. of Loops to send the entire file: ", loop)
19 print("write/Receiving process starting soon") #Receiving File from Client
20
21 while receiver_sequence <= loop:
22     ImgPkt,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence,E_Prob,P_Drop,loop,Window_Size) #Calls the function
23     p.write(ImgPkt) #writes/stores the received data to a file
24
25 #File Received from Client at the end of Loop
26
27 Received_File_Size = Functions.File_size(p) #Calculating Received Image file size
28
29 p.close() #closing the file
30 sock.close() #closing the socket
31
32 end = time.time() #finding the end-time
33 Elapsed_time = end - start #Elapsed time
34
35 while receiver...

```

Run: Server Client

Sequence Number: 617,Receiver_sequence: 617, Checksum from Client: 43058, Checksum for Received File: 43058
Sequence Number: 618,Receiver_sequence: 618, Checksum from Client: 17436, Checksum for Received File: 17436
Sequence Number: 619,Receiver_sequence: 619, Checksum from Client: 57483, Checksum for Received File: 57483
Data Corrupted #####
Sequence Number: 620,Receiver_sequence: 620, Checksum from Client: 32952, Checksum for Received File: 32952
Sequence Number: 621,Receiver_sequence: 621, Checksum from Client: 10675, Checksum for Received File: 10675
Sequence Number: 622,Receiver_sequence: 622, Checksum from Client: 34861, Checksum for Received File: 34861
Sequence Number: 623,Receiver_sequence: 623, Checksum from Client: 831, Checksum for Received File: 831
Sequence Number: 624,Receiver_sequence: 624, Checksum from Client: 23506, Checksum for Received File: 23506

4: Run Python Console

You can see Data Corrupted intentionally in the Server

Client resends the packet and file has been transferred

```

from Globals import *
import Functions
import Send_Receive
import Send_Receive

'''To corrupt data packet, Set value from 0 - 99 in E_prob'''
E_Prob = 10
P_Drop = 0

print ("Window Size : " + Window_Size)
image_buffer=ImageWindow_Size
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
f = open('Lion.jpg','rb')

fileSize = Functions.File_size(f)

loop = Functions.loopsizes(fileSize,buffer_size)
loop_bytes = struct.pack("!I", loop)
print("File has been Extracted \nFile size: {} \nNo. of Loops to send the entire file: {}".format(fileSize,loop))
seq_nbr = 0
base=0
print("Client File Transfer Starts...")

while(base <= loop):
    seq_nbr,base = Send_Receive.RdtSendPkt(f,sock,addr,seq_nbr,loop_bytes,E_Prob,P_Drop,Window_Size,base,loop,image_buffer) #calls the function rdt_send to
    f.close()
    sock.close()
    end = time.time()
    Elapsed_time = end - start
    print("Client: File Sent\nFile size sent to server: {} \nTime taken in Seconds:{}\n".format(fileSize,Elapsed_time))

    Process finished with exit code 0

```

ACK CORRUPTION:

```

from Globals import *
import Functions
import Send_Receive
import Send_Receive

'''To corrupt data packet, Set value from 0 - 99 in E_prob'''
E_Prob = 10
P_Drop = 0

print ("Window Size : " + Window_Size)
image_buffer=ImageWindow_Size
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
f = open('Lion.jpg','rb')

fileSize = Functions.File_size(f)

loop = Functions.loopsizes(fileSize,buffer_size)
loop_bytes = struct.pack("!I", loop)
print("File has been Extracted \nFile size: {} \nNo. of Loops to send the entire file: {}".format(fileSize,loop))
seq_nbr = 0
base=0
print("Client File Transfer Starts...")

while(base <= loop):
    seq_nbr,base = Send_Receive.RdtSendPkt(f,sock,addr,seq_nbr,loop_bytes,E_Prob,P_Drop,Window_Size,base,loop,image_buffer) #calls the function rdt_send to
    f.close()
    sock.close()
    end = time.time()
    Elapsed_time = end - start
    print("Client: File Sent\nFile size sent to server: {} \nTime taken in Seconds:{}\n".format(fileSize,Elapsed_time))

    Process finished with exit code 0

```

You can see ACK Corrupted intentionally in the client (above screenshot)
File has been successfully transferred later.

```

from Globals import *
import Functions
import Send_Receive
import Send_Receive

'''To corrupt data packet, Set value from 0 - 99 in E_prob'''
E_Prob = 10
P_Drop = 0

print ("Window Size : " + Window_Size)
image_buffer=ImageWindow_Size
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
f = open('Lion.jpg','rb')

fileSize = Functions.File_size(f)

loop = Functions.loopsizes(fileSize,buffer_size)
loop_bytes = struct.pack("!I", loop)
print("File has been Extracted \nFile size: {} \nNo. of Loops to send the entire file: {}".format(fileSize,loop))
seq_nbr = 0
base=0
print("Client File Transfer Starts...")

while(base <= loop):
    seq_nbr,base = Send_Receive.RdtSendPkt(f,sock,addr,seq_nbr,loop_bytes,E_Prob,P_Drop,Window_Size,base,loop,image_buffer) #calls the function rdt_send to
    f.close()
    sock.close()
    end = time.time()
    Elapsed_time = end - start
    print("Client: File Sent\nFile size sent to server: {} \nTime taken in Seconds:{}\n".format(fileSize,Elapsed_time))

    Process finished with exit code 0

```

PACKET LOSS (P_Drop is set to 10, that is 10% chance of dropping the packet [Sample Output]):

Data Packet Loss:

```

from Globals import*
import Send_Receive
import Functions
...
'''To corrupt acknowledgement packet, Set value from 0 - 99 in E_prob'''
E_Prob = 0
P_Drop = 10

sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.bind(addr)
print("Server Started")

p = open('Received_Image.jpg', 'wb')

receiver_sequence = 0
loopTimes,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence,E_Prob,P_Drop,loop,Window_Size)
#Socket with IPv4, UDP
#Binding the socket
#opening a new file to copy the transferred image
#Server side Sequence number is initialised to zero
#Loop struct.unpack("!I", loopTimes)[0]
print ("No. of Loops to send the entire file: ", loop)
print("write/Receiving process starting soon")
#Receiving File from Client

while receiver_sequence <= loop:
    ImgPkt,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence,E_Prob,P_Drop,loop,Window_Size)
    p.write(ImgPkt)
    #Writes/stores the received data to a file

#File Received from Client at the end of Loop
Received_File_Size = Functions.File_size(p)
#Calculating Received Image file size

p.close()
sock.close()
#closing the file
#closing the socket
#Finding the end-time
end = time.time()
#Elapsed time = end - start
#Elapsed time

```

Run tab output:

```

Sequence Number: b09,Receiver_sequence: b09, Checksum from Client: 49982, Checksum for Received File: 49982
Sequence Number: b10,Receiver_sequence: b10, Checksum from Client: 64238, Checksum for Received File: 64238
Sequence Number: b11,Receiver_sequence: b11, Checksum from Client: 10283, Checksum for Received File: 10283
#####
Sequence Number: b12,Receiver_sequence: b12, Checksum from Client: 20616, Checksum for Received File: 20616
Sequence Number: b13,Receiver_sequence: b13, Checksum from Client: 31274, Checksum for Received File: 31274
Sequence Number: b14,Receiver_sequence: b14, Checksum from Client: 11385, Checksum for Received File: 11385
Sequence Number: b15,Receiver_sequence: b15, Checksum from Client: 3790, Checksum for Received File: 3790
Sequence Number: b16,Receiver_sequence: b16, Checksum from Client: 12751, Checksum for Received File: 12751
Sequence Number: b17,Receiver_sequence: b17, Checksum from Client: 43058, Checksum for Received File: 43058

```

You can see Data packet lost intentionally in the Server

Client resends the packet and data transfers successfully later in the server.

```

from Globals import*
import Send_Receive
import Functions
...
'''To corrupt acknowledgement packet, Set value from 0 - 99 in E_prob'''
E_Prob = 0
P_Drop = 10

sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
sock.bind(addr)
print("Server Started")

p = open('Received_Image.jpg', 'wb')

receiver_sequence = 0
loopTimes,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence,E_Prob,P_Drop,loop,Window_Size)
#Socket with IPv4, UDP
#Binding the socket
#opening a new file to copy the transferred image
#Server side Sequence number is initialised to zero
#Loop struct.unpack("!I", loopTimes)[0]
print ("No. of Loops to send the entire file: ", loop)
print("write/Receiving process starting soon")
#Receiving File from Client

while receiver_sequence <= loop:
    ImgPkt,address,receiver_sequence = Send_Receive.RdtReceivePkt(sock,buffer_size,receiver_sequence,E_Prob,P_Drop,loop,Window_Size)
    p.write(ImgPkt)
    #Writes/stores the received data to a file

#File Received from Client at the end of Loop
Received_File_Size = Functions.File_size(p)
#Calculating Received Image file size

p.close()
sock.close()
#closing the file
#closing the socket
#Finding the end-time
end = time.time()
#Elapsed time = end - start
#Elapsed time

```

Run tab output:

```

Sequence Number: d32,Receiver_sequence: d32, Checksum from Client: 12648, Checksum for Received File: 12648
Sequence Number: d33,Receiver_sequence: d33, Checksum from Client: 30250, Checksum for Received File: 30250
Sequence Number: d34,Receiver_sequence: d34, Checksum from Client: 45778, Checksum for Received File: 45778
Sequence Number: d35,Receiver_sequence: d35, Checksum from Client: 3113, Checksum for Received File: 3113
Sequence Number: d36,Receiver_sequence: d36, Checksum from Client: 14365, Checksum for Received File: 14365
Sequence Number: d37,Receiver_sequence: d37, Checksum from Client: 50814, Checksum for Received File: 50814
Server: File Received
Received File size: 649413
Time taken In Seconds: 4.8006851673126225
Process finished with exit code 0

```

ACK Packet Loss:

The screenshot shows the PyCharm IDE interface with the following details:

- Code Editor:** The main window displays Python code for a client-side file transfer. It includes imports for `socket` and `struct`, and defines variables like `E_Prob` and `P_Drop`. The code uses a loop to send data packets, with logic to handle sequence numbers and a sliding window. A comment indicates that an ACK packet was intentionally dropped.
- Run Tab:** The "Run" tab is active, showing the execution environment. It lists several log entries from the client's perspective, including sending multiple packets (e.g., "Sending the packet: 622", "623", "624", "625") and receiving an ACK (e.g., "ACK OKAY: ACK624"). One entry specifically notes an "ACK packet DROPPED !!!".
- Python Console:** The bottom pane shows the command-line output of the application. It includes the same log entries as the Run tab, along with the final message "Client: File Sent" and the time taken for the transfer.

You can see ACK packet dropped intentionally in the client, but here cumulative ack works.

You can see Ack packet lost intentionally in the Server

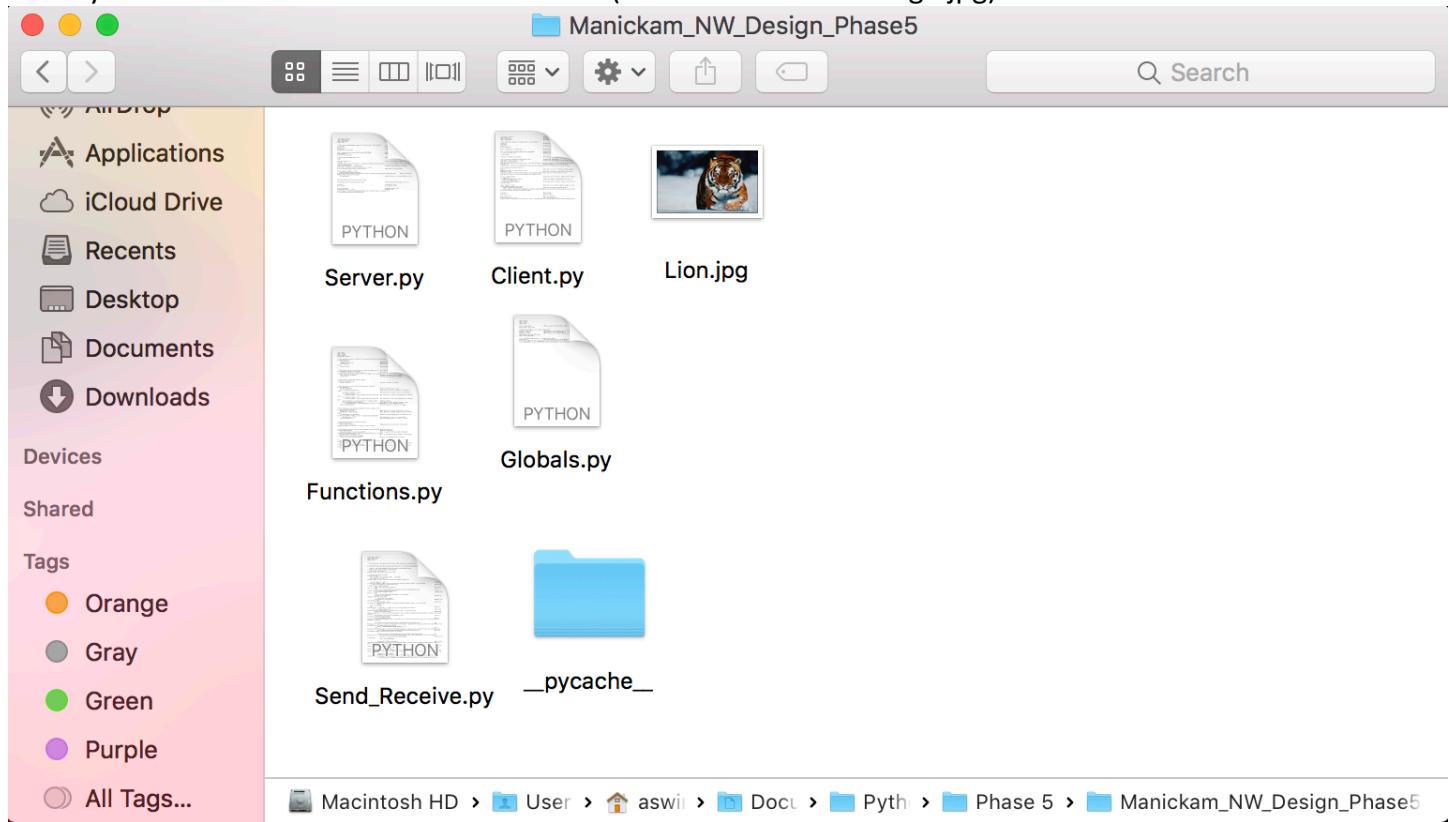
Client resends the packet and data transfers successfully later in the server.

The screenshot shows the PyCharm IDE interface with the following details:

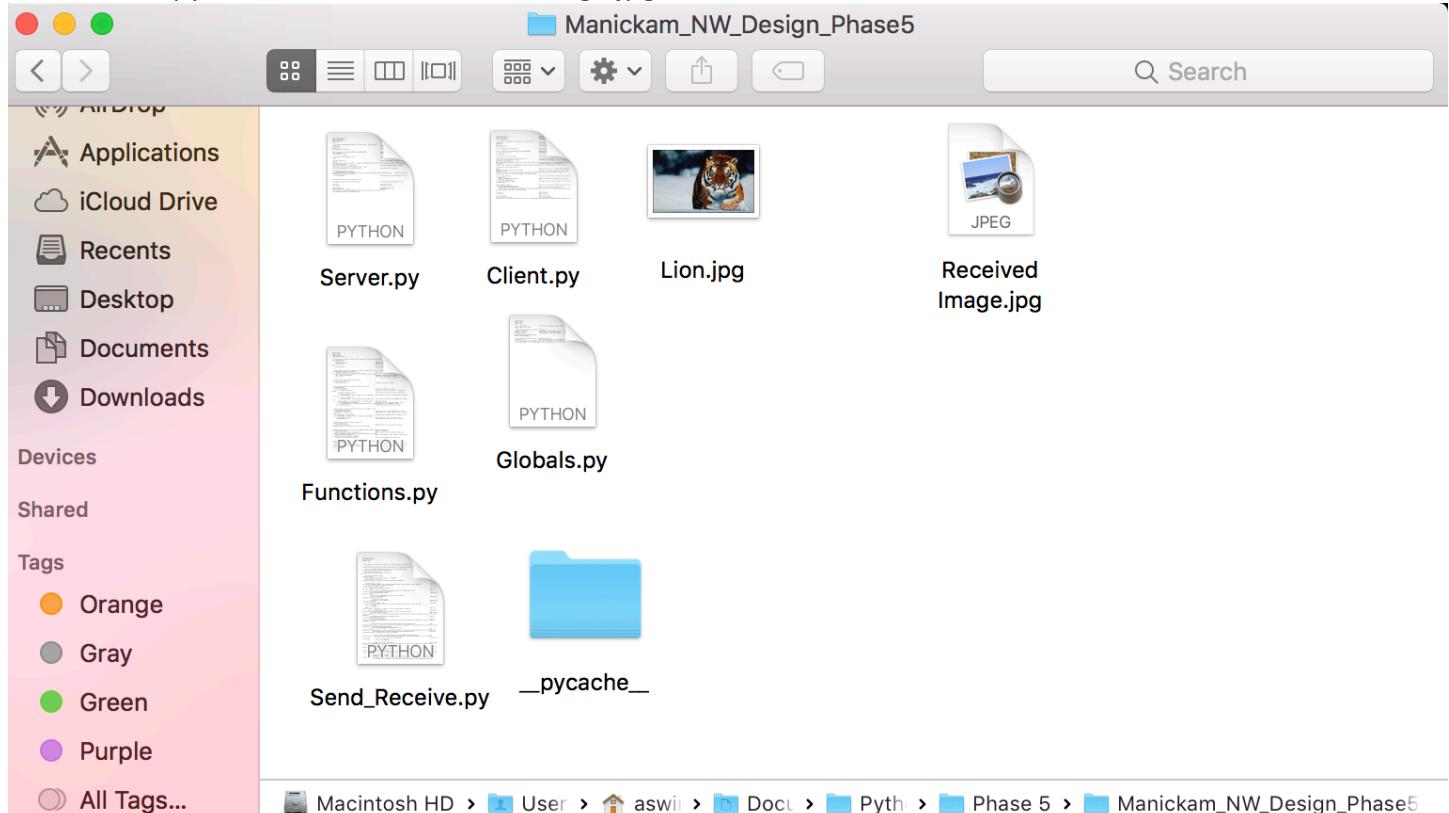
- Code Editor:** The main window displays Python code for a client-side file transfer, similar to the first example but with different sequence numbers (e.g., 622, 623, 624, 625).
- Run Tab:** The "Run" tab is active, showing the execution environment. It lists log entries from the client, including sending multiple packets and receiving an ACK (e.g., "ACK OKAY: ACK624"). One entry notes an "ACK packet DROPPED !!!".
- Python Console:** The bottom pane shows the command-line output. It includes the same log entries as the Run tab, along with the final message "Client: File Sent" and the time taken for the transfer.

FILE TRANSFER OUTPUT:

Initially the folder looks like below screenshot (Without Received Image.jpg)



After Server.py executed first, Received Image.jpg is created



After Running the Program, New file called Received Image.jpg will be in the folder which shows successful File transfer

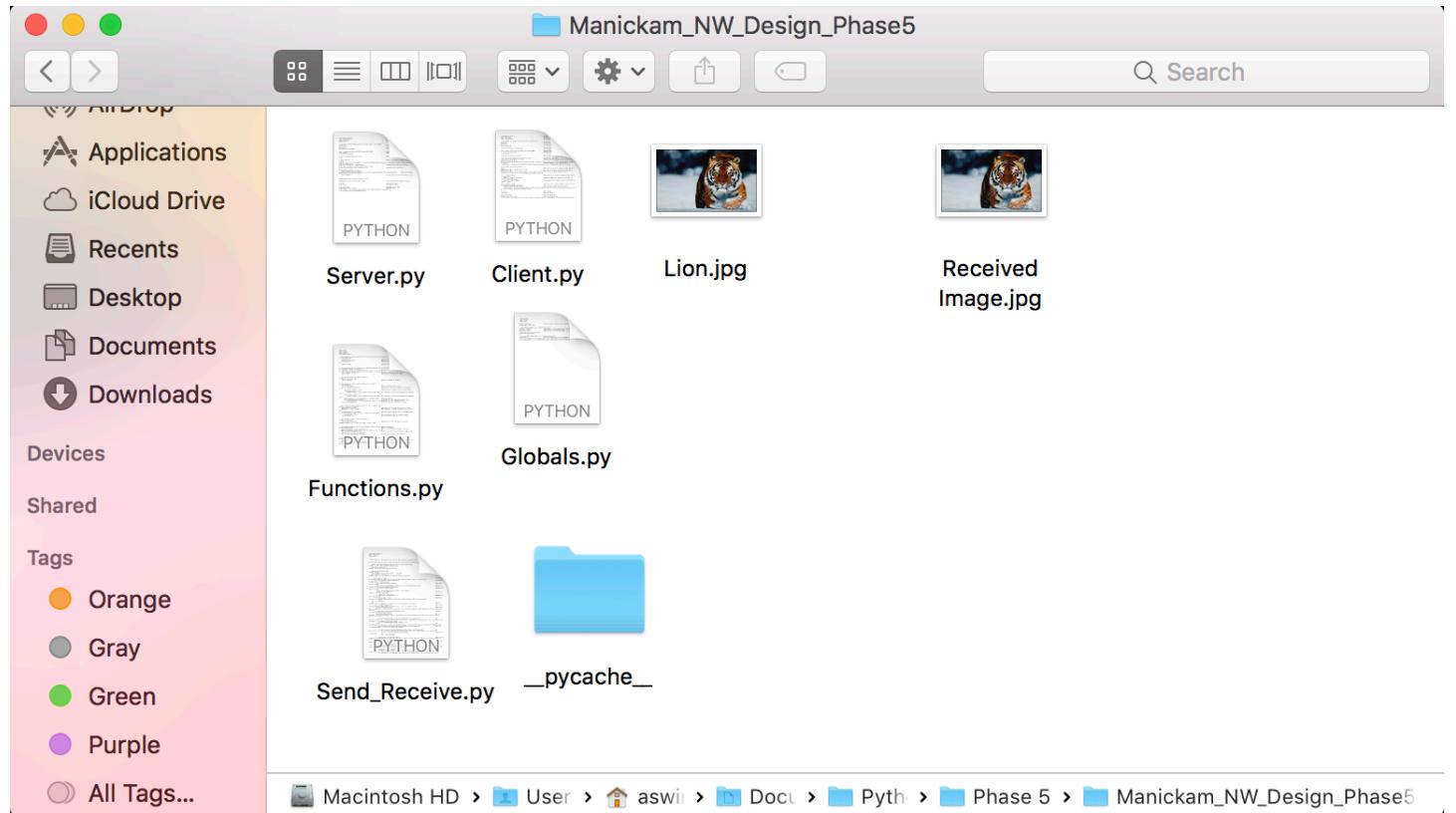


Image file and Received Image is of same size (Reliable Data Transfer)

Lion.jpg Info		Received Image.jpg Info	
	Lion.jpg		Received Image.jpg
	Modified: Friday, September 22, 2017 at 1:54 PM		Modified: Today, 12:12 AM
<input type="button" value="Add Tags..."/>		<input type="button" value="Add Tags..."/>	
<p>▼ General:</p> <p>Kind: JPEG image Size: 649,413 bytes (651 KB on disk) Where: Macintosh HD ▶ Users ▶ aswinkumarmanickam ▶ Documents ▶ Python ▶ Phase 5 ▶ Manickam_NW_Design_Phase5 Created: Friday, September 22, 2017 at 1:54 PM Modified: Friday, September 22, 2017 at 1:54 PM</p> <p><input type="checkbox"/> Stationery pad <input type="checkbox"/> Locked</p>			
<p>▼ More Info:</p> <p>Last opened: Today at 12:12 AM Dimensions: 1680 × 1050 Color space: RGB Alpha channel: No</p>			
<p>► Name & Extension: ► Comments: ► Open with: ▼ Preview:</p> 			
<p>► Sharing & Permissions:</p>			
<p>▼ General:</p> <p>Kind: JPEG image Size: 649,413 bytes (651 KB on disk) Where: Macintosh HD ▶ Users ▶ aswinkumarmanickam ▶ Documents ▶ Python ▶ Phase 5 ▶ Manickam_NW_Design_Phase5 Created: Today, 12:12 AM Modified: Today, 12:12 AM</p> <p><input type="checkbox"/> Stationery pad <input type="checkbox"/> Locked</p>			
<p>▼ More Info:</p> <p>Last opened: Today at 12:13 AM Dimensions: 1680 × 1050 Color space: RGB Alpha channel: No</p>			
<p>► Name & Extension: ► Comments: ► Open with: ▼ Preview:</p> 			
<p>► Sharing & Permissions:</p>			

Result:

Thus Go-Back-N protocol over an unreliable UDP channel has been implemented.