**Phase -2 Design Document**

**Implementation of RDT 1.0 over UDP**

**Pavan Kumar Revanuru – 01578776**

**Source Codes:**

**Server\_Phase2.py:**

import socket

name = "localhost" # DNS lookup for the IP

port = 15008 # Interger variable

ss = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) # Server socket

ss.bind((name, port)) # Binds port number to the socket

print "server started"

buf = 1024 # Buffer size = 1024

addr = (name, port) # Address

f = open("newimage.jpg", "wb") # Opens newimage.jpg

def rdt\_recv(): # Function definition

data, addr = ss.recvfrom(buf) # Receives data of size 1024 bytes

print "received packet"

f.write(data) # Writes the data to the image file

for x in range(0,1500): # Loop executes for 2000 times

rdt\_recv() # Function call to receive another packet

f.close() # File closes

print "file downloaded"

**Client\_Phase2.py**

import socket

name = "localhost" # DNS lookup for the IP address

port = 15008 # Integer variable

cs = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) # Client socket

buf = 1024 # Buffer size

addr = (name, port) # Address

f = open("image.jpg", "rb") # Opens the image

def rdt\_send(): # function definition

print "sending 1024 bytes"

data = f.read(buf) # Reads 1024 bytes of data

cs.sendto(data, addr) # Sends data of size 1024 bytes

for x in range(0,1500): # Loop executes for 2000 times

rdt\_send() # Function call to send another packet.

f.close() # Closes file

cs.close() # Closes socket

**Code Execution:**

1. Server\_phase2.py is made to run (F5) so that the receiver/server is started prior to the sender.
2. Client\_Phase2.py is run so that the image is transferred from client to server.
3. The image from the client is sent to the server in the form of many packets and the receiver saves the packets in the same order in a file “newimage.jpg”. If there is no such file, it creates one or overwrites the existing data for the new incoming stream of packets.

Explanation:

1. As the aim is to send the data in the form of many packets, buffer size, buf is initialized to 1024 (1KB) in both client and server.
2. In the client side, a file named “image.jpg” is made to open in the binary format by using, f=open(“image.jpg”, “rb”)
3. A function with name “rdt\_send()” is created for sending the packets of size 1024 bytes.
4. data, is a variable which reads only 1024 bytes as the buf was initialized to that particular value.
5. cs.sendto(data, addr) is the command which sends the data ( 1024 bytes) to the particular address and port number.
6. For sending the data in the form of packets, a for loop is used [for x in range ( 0 to 1500)], which makes sure that 1500 packets of size 1024 bytes each is transmitted from client to server. (Assuming the file is less than 1500 KB. If the size is more than 1500 KB, the number should be appropriately given.)
7. Whenever the number is in the given range, a new packet with new data is sent.
8. If there is no data to be sent, assuming the size of image is less than 1500 KB, then an empty packet is sent to the server.
9. After all the transmission is done, file and socket are closed.
10. In the receiving side, which is the server, port number, name and socket are initialized.
11. A command, ss.bind((name,port)) binds the port number and name to the socket.
12. f=open(“newimage.jpg”, “wb”) opens the file if it’s already in the directory. Else, a new file with that name is created.
13. rdt\_recv() is a function defined for receiving the packets from client.
14. data, addr = ss.recvfrom(buf) saves the received packet information in the temporary variable ‘data’.
15. f.write(data) writes the contents in ‘data’ to the file “newimage.jpg” in the binary format.
16. For repeating the above process, a ‘for’ loop is used for the range 0 to 1500, that is the loop is executed for 1500 times.
17. rdt\_recv() function is called for saving the other packets in order.
18. Once every packet is received, the file is closed
19. The received image can be found with a name ‘newimage’ in the directory C:\Python27

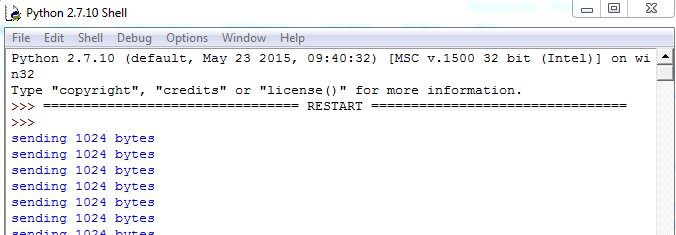
**Images used in the execution:**

image.jpg (Sending) newimage.jpg (Receiving)

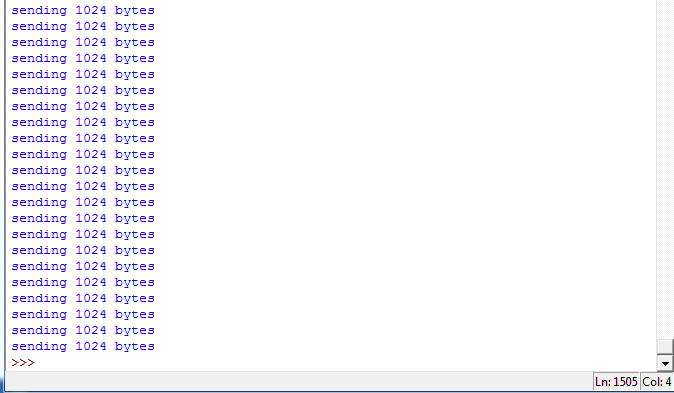
 

**Screenshots:**

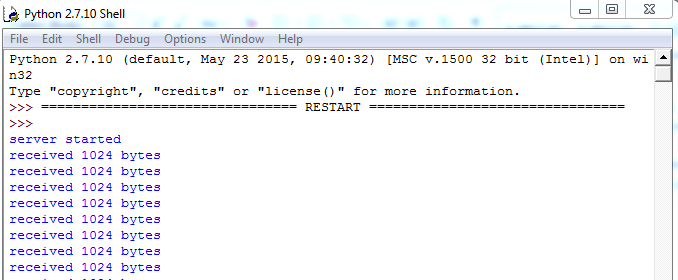
**Client output picture 1:**



Client output picture 2:



**Server output picture 1:**



**Server output picture 2:**

