Phase 3 Design Document

Title and Authors

J.M. Gallagher, James Whitney, Boban Pallathucharry
18-March-2020

Environment

OS Used: Ubuntu Linux
Programming Language: Python 3

Code Explanation

RDTClientPhase3.py

```
# Client portion of code

# Import statement for content needed for the checksum function
from bitstring import *

import time #imports time function

import random #library needed for generating random numbers


# make_pkt expects all inputs to be in the form of bytes
# Example implementation make_pkt(b'1', b'4444', b'678432')
# Returns b'1444678432'
def make_pkt(ACK, checksum, data):
      ACK_b = bytearray(ACK)
      checksum_b = bytearray(checksum)
      data_b = bytearray(data)
      return ACK_b + checksum_b + data_b

# recv_pkt receives packets from the designated Socket with size up to the giv-
# en packetlength
def recv_pkt(Socket, packetlength):
      message, Address = serverSocket.recvfrom(packetlength)
      return message, Address

# comp_checksum calculates the UDP checksum for a set of input bytes
def comp_checksum(data):
      #declaring variables used in the checksum function
      carry = bytes(2)     # variable to store the actual carry bits from the
                           # 16-bit sum of a packet
```

```
        checksum = bytes(2)                          # variable to store the checksum
        value1 = bytes(2)                            # variable to store copies of
                                                     # the 16-bit sum during swapping
        sum2 = bytes(2)                              # variable to store copies of
                                                     # the 16-bit sum during swapping
        actual_sum = bytes(2)
        suminput = bytes(2)
        m = BitStream(bytearray(2))          # internediate variables to
                                             # convert a string into a stream of Bits
        n=BitStream(bytearray(2))            # internediate variables to convert a
                                             # string into a stream of Bits
        c = bytearray(data)
        k = BitStream(c)
        total_sum = sum(c)
        m.bin = bin(total_sum)[2:]
        n = BitStream(m)
        v = len(n)
        sum2 = total_sum
        while(v>8):                      # if length > 8 it means we have carry bits
              s = v-8                    # this value shows the number of carry bits
              #print(s)
              # Acquires the sum bits
              actual_sum = sum2 & 0x00FF
              # print(bin(value3)[2:])
              # Acquires carry bits
              carry = (sum2 & 0xFF00) >> 8
              #print(bin(carry)[2:])
              value3 = actual_sum + carry     # 1's compliment addition
              #print(bin(value2)[2:])
              v = len(bin(value3)[2:])           # checks the length of new sum
              #print(v)
              sum2 = value3              # swap values to make a copy for the next
                                         # iteration, sum2 is a bytes type variable

          #print('\n' + bin(~value2) + '\n' + bin(value2))

        return sum2                    # this final value will contain the checksum

# is_corrupt returns True or False based on analysis of the ACK packet
# acknowledgement and sequence number. It expects the packet to be input as
# bytes, e.g. b'2343x\00x\123'
def is_corrupt(packet):
        #return False
        data=bytearray(packet)
        #print("Packet data is: " + str(packet))
        ACK = data[0:1]
        seq = data[1:2]
```

```
      if ACK == seq:
            return False
      else:
            return True


# is_ACK requires data to be bytes and seq to be bytes as well
# Example implementation: is_ACK(b'12345', b'1') will return True
# The ACK corruption is also implemented in this function
# When the user sets corruption to a non-zero value, there is a chance that the
# is_ACK is corrupted and returns the wrong boolean.
# How often this happens is determined by how large a value the use inputs for
# corrupt_chance
def is_ACK(data, seq):
      rand = random.randint(1,100) #generates random number
      if rand<=corrupt_chance: #if random number is smaller than corrupt chance
            return True
      else:
            if data[0:1] == bytes(seq):
                  return True
            else:
                  return False

# Method udt_send takes a series of bytes and sends them
# to the specified socket using the given clientSocket
def udt_send(packet, Socket, Address):
        Socket.sendto(packet, Address)

# This line imports the socket library needed to execute this program
from socket import *
import time                         # imports time functions for timer

# The next two lines identify the server socket we will use for our communica
# tions
# The local machine will be running the server and it will have port 4356, as
# designated in the code written for the server. It is critical that the server
# code
# use the same port as designated here or the client will not be able to com
# municate
# with the server.

start_time = time.perf_counter() #starts the timer
serverName ='localhost'
serverPort = 4356
serverAddress = (serverName, serverPort)

# Create a UDP socket to use to send messages to a server
```

```
clientSocket = socket(AF_INET, SOCK_DGRAM)

# Open file for the make_pkt function
# The file needs to be located in the same directory as this .py file
# The "rb" flags specify options "read" and "binary"
# We want to read binary data so we can deliver it directly to the
# make_pkt function


print("Enter A Whole Number Between 0 and 100 for ACK Corruption Chance\n")
#enters number to be used as percentage of corruption
corrupt_chance = int(input()) # takes the number and places it in a variable to
                              # test against other random numbers

f = open("image1.bmp", "rb")
file_bytes = f.read(1000)

# Make the first packet to send to the server
# and a counter "i" so we can tell the user how many packets
# are sent.

i = 0
start_time = time.perf_counter()             # starts the timer
# The while loop runs the make_pkt method until
# it exhausts all of the bits in the specified transfer file.
# In Python, the end of the file is denoted as b'', so when this point
# is reached, the while loop ends.
while (file_bytes != b''):

      file_data = make_pkt(b'0', comp_checksum(file_bytes).to_bytes(2, 'big'),
file_bytes)
      #print(file_bytes[0])
      #print(file_data)
      #print(comp_checksum(file_bytes).to_bytes(2, 'big'))
      #print(is_corrupt(file_data))
      #print(file_bytes)
      udt_send(file_data, clientSocket, serverAddress)
      #print("I sent a seq_0 packet")
      # Wait for ACK 0
      while True:
            content = clientSocket.recvfrom(1024)
            rec_pkt = bytearray(content[0])
            #print(rec_pkt)
            #if (is_corrupt(rec_pkt)): print("ACK_0 is corrupt.")
            if (is_corrupt(rec_pkt) or is_ACK(rec_pkt, b'1')):
                  #if is_ACK(rec_pkt, b'1'): print("is wrong ack1")
                  udt_send(file_data, clientSocket, serverAddress)
```

```
            else:
                  #print("I received ack for seq_0")
                  break

      i = i+1
      file_bytes = f.read(1000)
      if (file_bytes== b''):
            break
      #print(file_bytes)
      file_data = make_pkt(b'1', comp_checksum(file_bytes).to_bytes(2, 'big'),
file_bytes)
      udt_send(file_data, clientSocket, serverAddress)
      #print("I sent a seq_1 packet.")
      # Wait for ACK 1
      while True:
            content = clientSocket.recvfrom(1024)
            rec_pkt = bytearray(content[0])
            if (is_corrupt(rec_pkt) or is_ACK(rec_pkt, b'0')):
                  #if is_ACK(rec_pkt, b'0'): print("is wrong ack2")
                  udt_send(file_data, clientSocket, serverAddress)
            else:
                  #print("I just received ack for seq_1")
                  break

      #print(packet)
      #print("I just sent packet # " + str(i))
      i=i+1
      file_bytes = f.read(1000)
      if (file_bytes== b''):
            break

# In order to prompt the server to stop receiving packets,
# close the file, and shutoff, the client sends the empty binary packet
# b''
#print("sending a blank")
udt_send(b'', clientSocket, serverAddress)

# We now tell the user the file has been transmitted to the server
# and how many packets were required for transmission.
stop_time = time.perf_counter() #stops timer
total_time = stop_time-start_time # subtracts start time from stop time to get
                                  # total time timer runs
print("Image transmission complete.")
print(str(i) + " packets were transmitted to the server.")
print("Transmission time: " + str(total_time) + " seconds") # prints the total
                                       # time taken to upload image in seconds
```

```
# Close file for the make_pkt function
# and close the client socket
f.close()
clientSocket.close()
print("Closing client program.")
```

RDTServerPhase3.py

```
# Server portion of code

# This line imports the socket library needed to execute this program
# We chose serverPort 4356 so the code doesn't interfere with any system
# processes
from socket import *
serverPort = 4356

# Import statement for content needed for the checksum function
from bitstring import *

import random                    # library needed for generating random numbers

# Create a UDP socket to receive messages from a client
serverSocket=socket(AF_INET,SOCK_DGRAM)

# This next line sets the server socket for our program as the current users'
# machine with the port specified in the earlier line of code
serverSocket.bind(('localhost', serverPort)) #sets up connection

# We now define a method recv_pkt to implement the RDT protocol
# This method requires a server socket and specified packet length
# It only returns the received message to the user in binary.
# It does not return the clientAddress as this code is not meant to send
# content back to the client.
def recv_pkt(serverSocket, packetlength):
      message, clientAddress = serverSocket.recvfrom(packetlength)
      return message, clientAddress

# comp_checksum calculates the UDP checksum for a set of input bytes
def comp_checksum(data):
      #declaring variables used in the checksum function
      carry = bytes(2)                # variable to store the actual carry bits
                                       # from the 16-bit sum of a packet
      checksum = bytes(2)             # variable to store the checksum
      value1 = bytes(2)               # variable to store copies of the 16-bit
                                       # sum during swapping
      sum2 = bytes(2)                 # variable to store copies of the 16-bit
                                      # sum during swapping
      actual_sum = bytes(2)
      suminput = bytes(2)
      m = BitStream(bytearray(2))     # internediate variables to convert a
                                       # string into a stream of Bits
      n=BitStream(bytearray(2))       # internediate variables to convert a
                                       # string into a stream of Bits
      c = bytearray(data)
```

```
      k = BitStream(c)
      total_sum = sum(c)
      m.bin = bin(total_sum)[2:]
      n = BitStream(m)
      v = len(n)
      sum2 = total_sum
      while(v>8):                    # if length > 8 it means we have carry bits
            s = v-8                  # this value shows the number of carry bits
            # Acquires the sum bits
            actual_sum = sum2 & 0x00FF
            # Acquires carry bits
            carry = (sum2 & 0xFF00) >> 8
            value3 = actual_sum + carry     # 1's compliment addition

            v = len(bin(value3)[2:])           # checks the length of new sum
            sum2 = value3              # swap values to make a copy for the next
                                       # iteration, sum2 is a bytes type variable

      return sum2                    # this final value will contain the checksum

# is_corrupt returns True or False based on analysis of the checksum
# of the input packet. It expects the packet to be input as bytes, e.g.
# b'2343x\00x\123'
def is_corrupt(packet):
      data=bytearray(packet)
      checksum = int.from_bytes(data[1:3], 'big')
      serverSum = ~comp_checksum(data[3:len(data)])
      if checksum + serverSum == -1:
            return False
      else:
            return True

# has_seq0 determines if the packet has sequence number 0
def has_seq0(data):
      if data[0:1] == b'0':
            return True
      else:
            return False

# Requires input of packet with 5 byte header
# The first byte is the ACK byte
# bytes 2-5 are the checksum, the remaining data is
# the packet content
def extract_bits(data):
      return data[3:len(data)]

# make_pkt takes 3 sets of bytes and concatenates them to be sent on a Socket
def make_pkt(ACK, checksum, data):
      ACK_b = bytearray(ACK)
      checksum_b = bytearray(checksum)
      data_b = bytearray(data)
      return ACK_b + checksum_b + data_b

# udt_send sends packets from the designated server socket to the given client
# address
def udt_send(packet, serverSocket, clientAddress):
        serverSocket.sendto(packet, clientAddress)
```

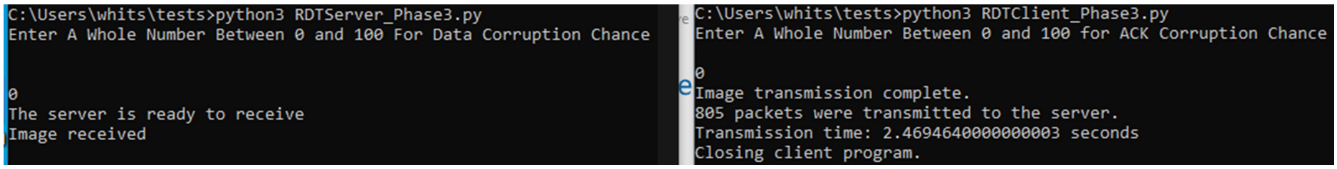
```
# The corrupting function simulates bit corruption in the packets received from
# a client.
# The amount of corruption is determined by the corrupt_chance parameter, which
equates to a percent chance of a bit
# being flipped in the packet.
# The function flips one bit based on whether a random number is less than the
# given corrupt_chance
def corrupting(data, corrupt_chance):           #function for corrupting packet
      rand = random.randint(1,100)    #random integer between 0-99 is generated
      if rand<=corrupt_chance: #checks if random number is less than percentage
                               # chance
            original_value = data[3]
            data[3] ^= 0x1F                 #if it is less, it corrupts the bit
            if original_value == data[3]: data[3] = 0xFF    # this line of code
                  # ensures some sort of corruption is introduced into the code
            return data #returns data

# Now we make a file in which the code will write binary
# information received in the form of packets from the socket.

print("Enter A Whole Number Between 0 and 100 For Data Corruption Chance\n")
#enters number to be used as percentage of corruption
corrupt_chance = int(input()) #takes the number and places it in a variable to
test against other random numbers

f=open('serverimage.bmp','wb')
print("The server is ready to receive")

# The while loop runs idle until the server begins to receive information from
# a client communicating with the server socket. The loop receives a packet,
# writes this packet to the open file and repeats until the packet contents
# is b''. When this final empty packet is received, the loop terminates.

onceThru = 0
send_pkt_cpy=b'0'
while True:
      while True:
            content = recv_pkt(serverSocket, 1024) #receives next batch of data
            data = bytearray(content[0])           #gets data from packet
            if data == b'':            #if empty, break out of inner while loop
                  break
            checksum = data[1:3]       #gets checksum
            clientAddress = content[1] #gets client address
            corrupting(data, corrupt_chance) #chance to corrupt packet

            if (is_corrupt(data) or not(has_seq0(data))):
                  ACK = b'1' #sets ack to opposite of what it should be
                  seq = b'00' #sets seq number
                  send_pkt = make_pkt(ACK, seq, b'')      #makes packet to send
                  udt_send(send_pkt, serverSocket, clientAddress) #sends packet
                                             # to tell client to resend data
                  break

            if (not(is_corrupt(data)) and has_seq0(data)):
                  ACK = b'0' #sets ack to proper value
                  seq = b'00' #sets sequence number
```

```
                send_pkt = make_pkt(ACK, seq, b'') # makes packet out of ack
                                                   # and sequence number
                if send_pkt == send_pkt_cpy: #if packet is equal to last
                            # packet saved, ack corruption, do not write data
                      udt_send(send_pkt, serverSocket, clientAddress)  # tells
                                                      # client to send new data
                      break
                else:           #if packet is different than last packet saved
                      f.write(extract_bits(data)) #write to file
                      send_pkt_cpy = send_pkt    # saves a copy of the ack and
                                                 # sequence number
                      udt_send(send_pkt, serverSocket, clientAddress)   # asks
                                                      # client to send new data
                      break
      if data == b'': break   # if data is blank, break out of outer while loop

      while True:

            content = recv_pkt(serverSocket, 1024) #receives next batch of data
            data = bytearray(content[0])
            if data == b'':
                  break
            checksum = data[1:3]
            clientAddress = content[1]
            corrupting(data, corrupt_chance)         # chance to corrupt packet

            if (is_corrupt(data) or has_seq0(data)):
                  ACK = b'0'
                  seq = b'10'
                  send_pkt = make_pkt(ACK, seq, b'')
                  udt_send(send_pkt, serverSocket, clientAddress)
                  break

            if (not(is_corrupt(data)) and not(has_seq0(data))):
                  ACK = b'1'
                  seq = b'10'
                  send_pkt = make_pkt(ACK, seq, b'')
                  if send_pkt == send_pkt_cpy:
                        udt_send(send_pkt, serverSocket, clientAddress)
                        break
                  else:
                        f.write(extract_bits(data))
                        send_pkt_cpy = send_pkt
                        udt_send(send_pkt, serverSocket, clientAddress)

                        break
      if data == b'': break

# Tell the user the image is received and close the file so the user may go and
# open it.
print("Image received")
f.close()
```

Scenario/Walk Through

Design Note: The code is for demonstration of Phase 3 of the RDT protocol and is meant to be run on one machine running a server and client instance. It is not designed to transmit information between two machines.

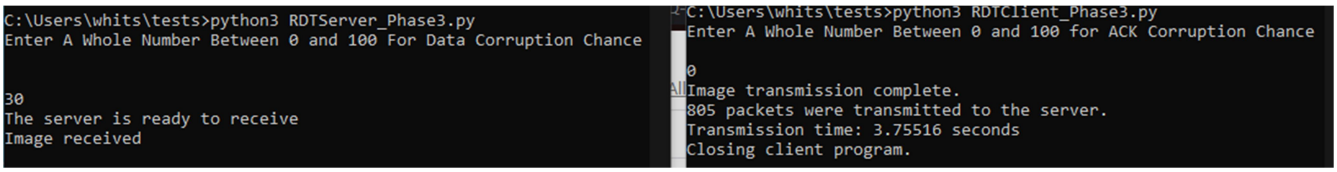
The following screenshot shows the communication between the server and the client instances on a Windows machine for a test image. The user would navigate to the folder containing the server and the client files. The server instance must run first so that its port is set to receive data from the client. This screenshot shows the interaction between client and server for **0% data corruption and 0% ACK corruption.**



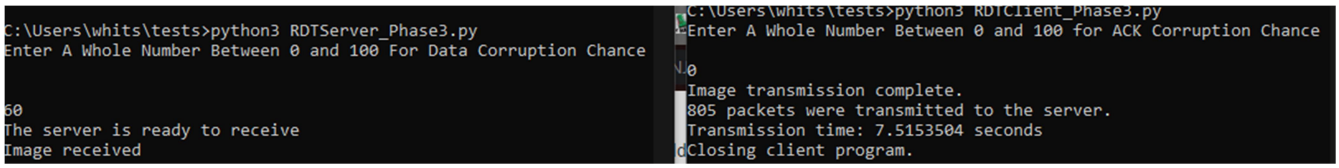
Once the server is ready, the message 'The server is read to receive' is displayed on the screen.

The client would then start to transmit packets. Once all packets are sent and the server receives them successfully, the server will say 'Image received' and close its socket while the client will say 'Image transmission complete' and report the number of packets which were transmitted to the server.

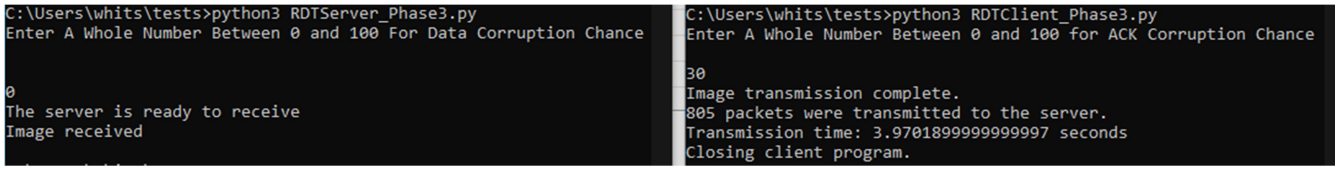
The following screenshot shows the interaction between client and server for **30% data corruption and 0% ACK corruption.**



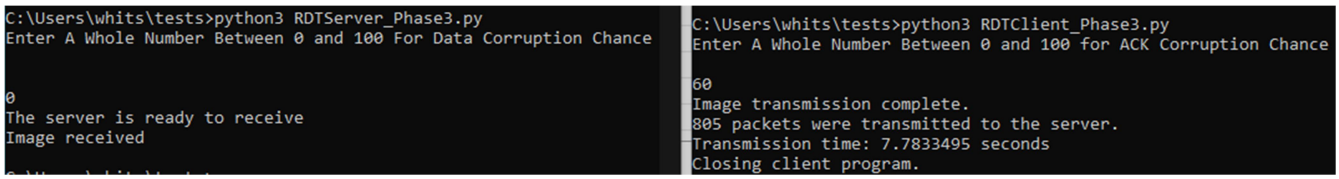
The following screenshot shows the interaction between client and server for **60% data corruption and 0% ACK corruption.**



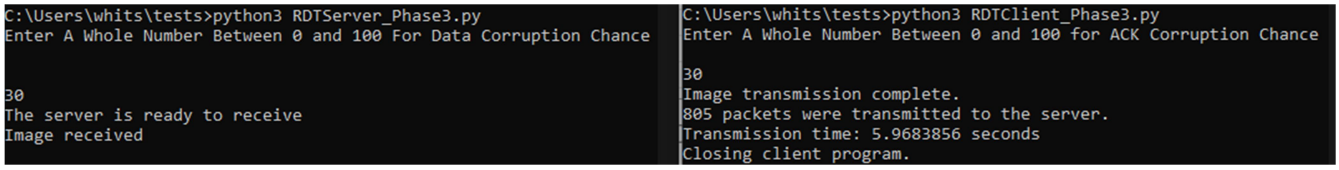
The following screenshot shows the interaction between client and server for **0% data corruption and 30% ACK corruption.**



The following screenshot shows the interaction between client and server for **0% data corruption and 60% ACK corruption.**



The following screenshot shows the interaction between client and server for **30% data corruption and 30% ACK corruption.**

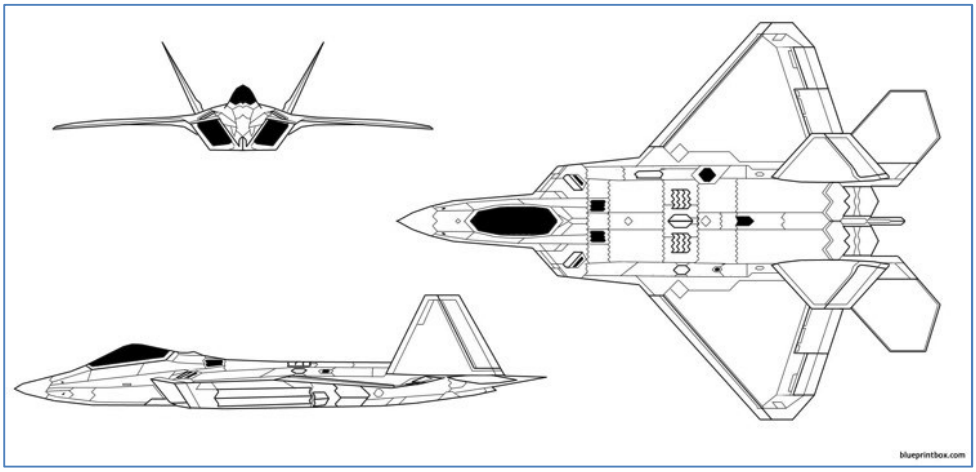


The following screenshot shows the interaction between client and server for **60% data corruption and 60% ACK corruption.**

```
C:\Users\whits\tests>python3 RDTServer_Phase3.py
Enter A Whole Number Between 0 and 100 For Data Corruption Chance


60
The server is ready to receive
Image received
```

```
C:\Users\whits\tests>python3 RDTClient_Phase3.py
Enter A Whole Number Between 0 and 100 for ACK Corruption Chance

60
Image transmission complete.
805 packets were transmitted to the server.
Transmission time: 21.7951137 seconds
Closing client program.
```

This is the test image on the client side



This is the received image on the server side