Phase 2 Design Document

**Title and Authors**

J.M. Gallagher, James Whitney, Boban Pallathucharry

31-Jan-2020

**Environment**

**OS Used:** Ubuntu Linux

**Programming Language:** Python 3

**Code Explanation**

RDTClientPhase3.py

# Client portion of code

# First we define two methods to implement the

# client portion of the RDT protocol

# Method make\_pkt requires open file "bytes"

# and integer packetlength. It returns a series of bytes

# with length specified by packetlength.

# Import statement for content needed for the checksum function

from bitstring import \*

import time #imports time function

# 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

def recv\_pkt(Socket, packetlength):

message, Address = serverSocket.recvfrom(packetlength)

return message, Address

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 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):

data=bytearray(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

def is\_ACK(data, seq):

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

f = open("f22.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)

udt\_send(file\_data, clientSocket, serverAddress)

# Wait for ACK 0

while True:

content = clientSocket.recvfrom(1024)

rec\_pkt = bytearray(content[0])

if (is\_corrupt(rec\_pkt) or is\_ACK(rec\_pkt, b'1')):

udt\_send(file\_data, clientSocket, serverAddress)

else:

print("I received ack for seq\_0")

break

file\_bytes = f.read(1000)

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')):

udt\_send(file\_data, clientSocket, serverAddress)

else:

print("I just received ack for seq\_1")

break

i=i+1

file\_bytes = f.read(1000)

# In order to prompt the server to stop receiving packets,

# close the file, and shutoff, the client sends the empty binary packet

# b''

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

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

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[2:len(data)]

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

def udt\_send(packet, serverSocket, clientAddress):

serverSocket.sendto(packet, clientAddress)

def corrupting(data, corrupt\_chance): #function for corrupting packet

rand = random.randint(0,99) #random integer between 0-99 is generated

if rand<=corrupt\_chance: #checks if random number is less than percentage chance

data ^= 0xFF #if it is less, it inverts the bit

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.

while True:

print("Choose A Mode \n1: No Corruption\n2: Data Corruption\n3: ACK Corruption") #prompts user for mode selection

mode = int(input()) #puts input into a variable

if (mode != (1 or 2 or 3)):

print("Choose A Valid Number") #ensures that the number can only be

#a valid choice

else:

break #breaks out of loop if number is input correctly

if (mode != 1): #only asks user for input if mode with corruption is chosen

print("Enter A Number Between 0 and 100\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

while True:

#Analyze content with server logic

#Wait for 0 from below

if (mode==2): #performs only if packet corruption is chosen

corrupting(data[0], corrupt\_chance) #chance to corrupt packet

content = recv\_pkt(serverSocket, 1024) #receives next batch of data

data = bytearray(content[0])

checksum = data[1:3]

clientAddress = content[1]

if data == b'':

break

# ACK = b'0'

# seq = b'1'

# send\_pkt = make\_pkt(ACK, seq, checksum)

while True:

if (is\_corrupt(data) or not(has\_seq0(data))):

print("This packet doesn't have seq\_0")

if (onceThru == 1):

ACK = b'1'

seq = b'00'

send\_pkt = make\_pkt(ACK, seq, b'')

udt\_send(send\_pkt, serverSocket, clientAddress)

content = recv\_pkt(serverSocket, 1024) #receives next

#batch of data

data = bytearray(content[0])

clientAddress = content[1]

if (not(is\_corrupt(data)) and has\_seq0(data)):

f.write(extract\_bits(data))

ACK = b'0'

seq = b'00'

send\_pkt = make\_pkt(ACK, seq, b'')

udt\_send(send\_pkt, serverSocket, clientAddress)

print("I wrote seq\_0 packet to the file.")

onceThru = 1

break

# extract, write, send ack

#Wait for 1 from below

while True:

content = recv\_pkt(serverSocket, 1024) #receives next batch of data

data = bytearray(content[0])

checksum = data[1:3]

clientAddress = content[1]

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)

print("This packet is bad or the wrong one")

if (not(is\_corrupt(data)) and not(has\_seq0(data))):

f.write(extract\_bits(data))

ACK = b'1'

seq = b'10'

#print(data)

send\_pkt = make\_pkt(ACK, seq, b'')

udt\_send(send\_pkt, serverSocket, clientAddress)

print("I wrote seq\_1 packet to the file.")

onceThru = 0

break

# Tell the user the image is received and close the file so the user may go and

# open it.

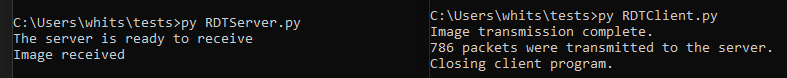
print("Image received")

f.close()

**Scenario/Walk Through**

Design Note: The code is for demonstration of the UDP 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.



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.

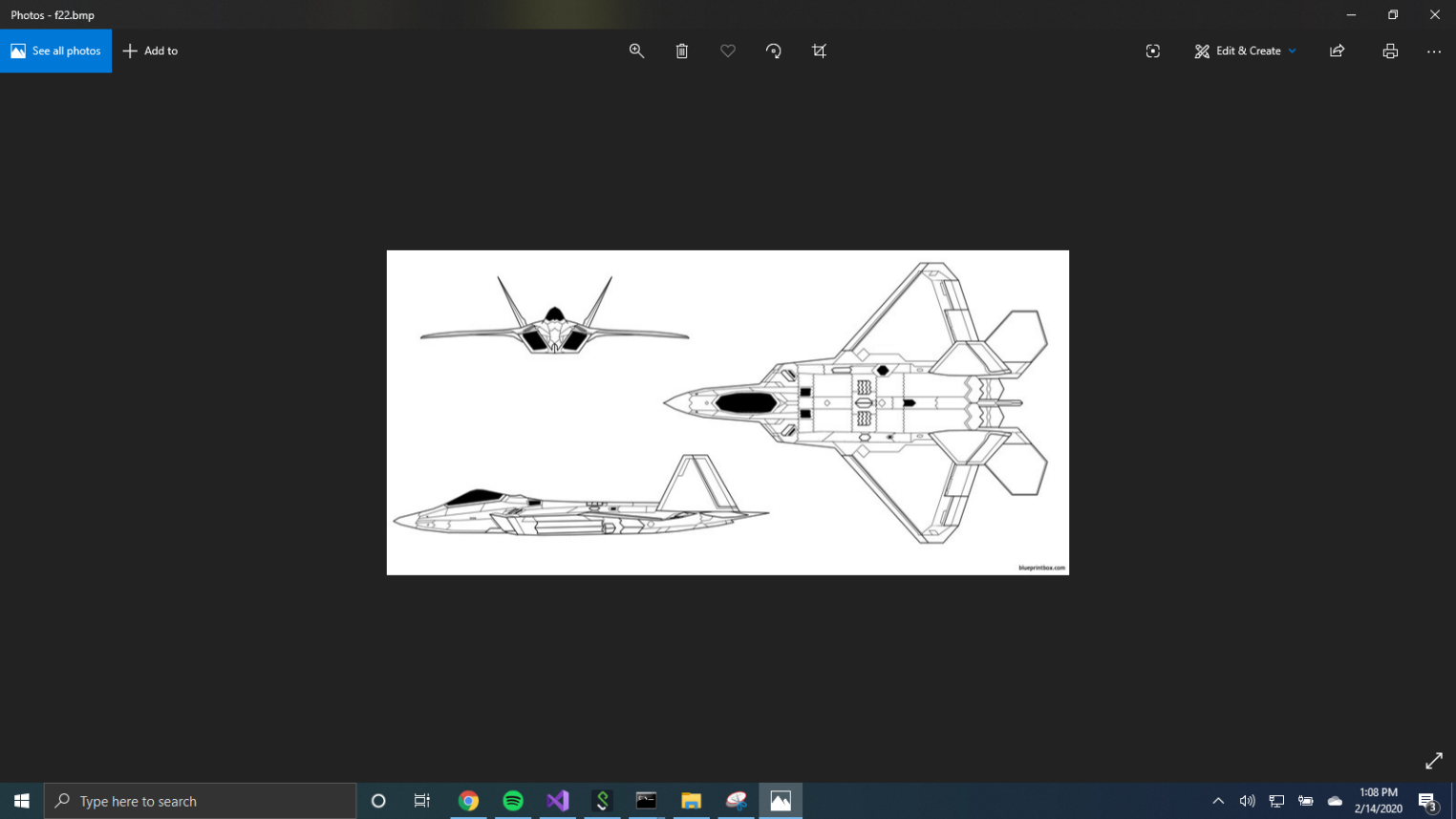
The image below is the first test image which was transmitted by the client.



The image below is the received image as seen from the server side.



The image below is the second test image transmitted by the client.



The image below shows the received image from the server.

