Computer and Communication

Networks

EE-357

BEE-9

Submitted by:

Sardar Hassan (223488)

Abdul Rehman (237532)

Table of contents:

| Page Number | Contents |
| --- | --- |
| 3 | Overview & Implementation |
| 4 | Implementation |
| 5 | Diagram of working principle |
| 6 | Receiver side functions & Sender  side functions |
| 7-13 | Receiver side code |
| 14-16 | Sender side code |

Overview:

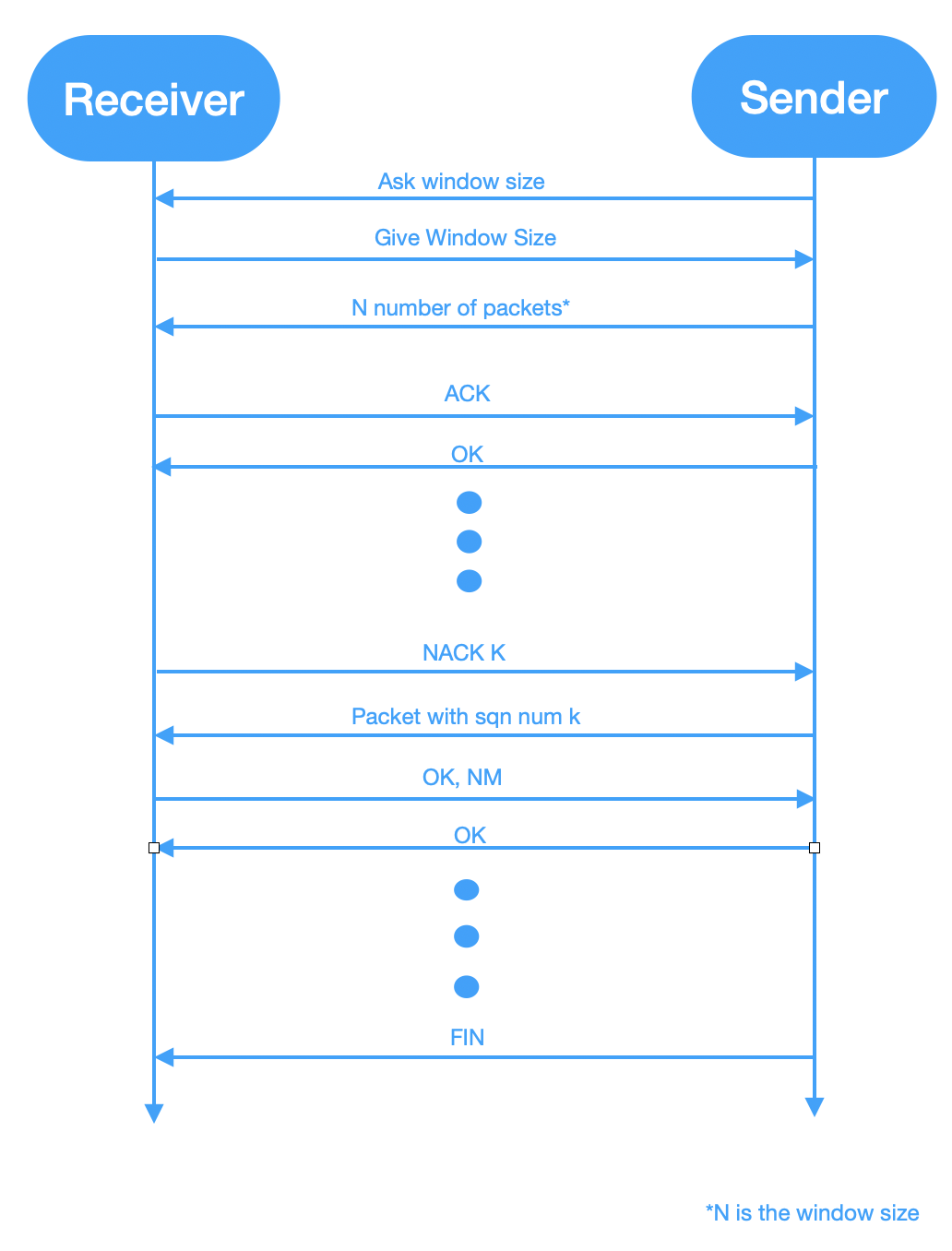
The purpose of this project is to make UDP more reliable. This is done by introducing Acknowledgement (ACK) and Negative Acknowledgement (NACK) messages to make certain that our packets are delivered. We also introduce packet number in order to reorder the packets once they arrive at the receiver. Furthermore, we have also restricted our packet payload size to 500 bytes and have introduced a window size, which is to be between 5 and 10.

Implementation:

1. Window size:
   * We defined a variable in the receiver side code which defines the desired window size as a string.
   * At the start of transferring the data file, the sender asks the user what the window size is.
   * The receiver then replies with the window size which is stored as a string on the sender side.
2. ACK and NACK messages:
   * We adjusted the payload by giving 495 bytes to the audio file data and the remaining 5 bytes of data to the packet number being sent.
   * At the receiver side, we started a master counter that would start from 0 and would increment after every packet arrived.
   * We then made our own buffer to hold the packet/sequence numbers and we defined a function that would run through the buffer and check if any packet number was missing by comparing them to the master counter.
3. Retransmission:
   * Retransmission, if required, is accomplished after the ACK and NACK messages check.
   * If there is a need for retransmission, the receiver sends the a NACK message, which is stored at both sides, along with the required packet number.
   * The sender then retransmits the required packet and waits for any other retransmission requests. If there aren’t any, the receiver transmits a ‘No More’ (NM) message (which is stored on both sides).
4. Reordering of packet:
   * The reordering of the received packets was done after the ACK and NACK message check and retransmission (if any).
   * We wrote a function that would do a bubble sort of the packet/sequence number buffer.
   * If there are any changes, the same changes were made in the data buffer.
   * We realize that bubble sort is not an elegant and efficient sorting algorithm, but it will do for the purpose of our project since the buffer size in small (the buffer size the same as the window size, between 5 and 10).
5. Restriction of packet payload to 500:
   * The packet payload is restricted by simply setting the buffer size to 500 while transmitting the packet.
6. Indicating no more data left to be transferred:
   * We have programmed the sender to send a ‘FIN’ message to the receiver when there is no more data to be sent.
   * The receiver, upon receiving the ‘FIN’ message, closes the file and ends the transmission.

**Note: Do not forget to add the audio files to path, and check that you have the correct file names.**

**Note: The sender side code contains a NACK message check case that intentionally misses out packet number 12 and packet number 100 when sending data to the receiver. This to confirm that the NACK message is sent, and the required packets are correctly received.**

Diagram of working principle:

Receiver side functions:

| Function Name | Purpose |
| --- | --- |
| give\_window\_size | To see if the sender has asked for the window size, and if yes, send it to them. |
| bubble\_sort | Sort the contents of the packet number buffer and make the same changes in the data buffer. |
| store\_buffer | Store the contents of the incoming packet payload to the associated buffers and set the sequence number. |
| send\_ack | Send the stored acknowledgement buffer to the sender. |
| empty\_buffers | Empty the contents of the data and packet number buffers by using the pop function. |
| send\_correct\_message | Check if the sender requires the ACK, NACK, OK, NM message to be sent again. |
| check\_missing | To check if there is a missing packet, by comparing the contents of the packet number buffer and the master counter. If yes, it calls store\_buffer and bubble\_sort. |

Sender side functions:

| Function Name | Purpose |
| --- | --- |
| ask\_window\_size | Get the window size from the receiver. |
| send\_data | Sends the contents of the audio file along with the packet number to the receiver. |
| check\_message | Check the incoming messages from the receiver. If it is a NACK message, send the required packet again. If it is an ACK message, continue on. This function responds to other messages accordingly. |
| empty\_buffers | Empty the contents of the data and packet number buffers by using the pop function. |

Receiver side code:

#created by sardar hassan and abdul rehman qamar aftab

from socket import \*

import select

import sys

import time

window\_size="10" #set window size

ip\_addr="127.0.0.1" #server ip address

port\_num=55000 #server port number

client\_port=50000

timeout=3 #no of seconds to wait till timeout

file\_to\_save=“from\_sender.mp3" #<---------------------------file name where we save data

open\_mode="ab" #<--------------------------------------------file open mode

buffer\_size=500 #set buffer length to to 500 bytes of packet

remaining=5

master\_counter=0

packet\_lost=False

fin\_recieved=False

ack\_message="ACK"

nack\_message="NACK"

send\_again\_message="Send again"

ack\_message="ACK"

nack\_message="NACK"

ok\_message="OK"

nm\_message="NM"

fin\_message="FIN"

from\_us=ack\_message #default from us message

sqn\_num\_prefix= "0" #set prefix digits of syn number

ack\_num="0" #set ack number to zero

sqn\_num="0"

data\_buffer=[""]

sqn\_num\_buffer=[""]

nack\_buffer=[""]

fsqn\_num\_buffer=[""]

temp4=[""]

temp5=[""]

counter=0

#temp 2 for storing

#temp 3 is len of data buffer

#temp 4 is for sqn num buffer, not used

#temp 5 is for data buffer

#temp 6 for nack buffer

#bind socket

udp\_sock=socket(AF\_INET,SOCK\_DGRAM)

udp\_sock.bind((ip\_addr,port\_num))

#custom define bubble sort

def bubble\_sort(sqn\_arr,data\_arr):

temp3=len(data\_arr)

print "recieved sqn nums are:", sqn\_num\_buffer[1:temp3],

index = len(sqn\_arr) - 1

while index >= 1:

for j in range(1,index):

if int(sqn\_arr[j]) > int(sqn\_arr[j+1]):

sqn\_arr[j], sqn\_arr[j+1] = sqn\_arr[j+1], sqn\_arr[j]

data\_arr[j], data\_arr[j+1] = data\_arr[j+1], data\_arr[j]

index -= 1

print "reordered sqn nums are:", sqn\_num\_buffer[1:temp3],

print ""

return sqn\_arr,data\_arr

#storring function

def store\_buffer(sqn\_num\_buffer,data\_buffer,message,fin\_recieved,sqn\_num,master\_counter):

if message != fin\_message: #if not FIN recieved

temp1=len(message) #length of message

temp2=buffer\_size-len(message)

if temp2==4: #for single digit sqn num

sqn\_num=message[temp1-1]

print "Sqn num is",sqn\_num

sqn\_num\_buffer.append(sqn\_num)

data\_buffer.append(message[0:temp1-1])

elif temp2==3: #for double digit sqn num

sqn\_num=message[temp1-2:temp1]

sqn\_num\_buffer.append(sqn\_num)

print "Sqn num is",sqn\_num

data\_buffer.append(message[0:temp1-2])

elif temp2==2: #for three digit sqn number

sqn\_num=message[temp1-3:temp1]

print "Sqn num is",sqn\_num

sqn\_num\_buffer.append(sqn\_num)

data\_buffer.append(message[0:temp1-3])

elif temp2==1: #for four digit sqn number

sqn\_num=message[temp1-4:temp1]

print "Sqn num is",sqn\_num

sqn\_num\_buffer.append(sqn\_num)

data\_buffer.append(message[0:temp1-3])

elif master\_counter>=10000: #for five digit sqn number

sqn\_num=message[temp1-5:temp1]

print "Sqn num is",sqn\_num

sqn\_num\_buffer.append(sqn\_num)

data\_buffer.append(message[0:temp1-3])

else: #for file data less than buffer\_size - remaining

if master\_counter<10: #for single digit sqn num

sqn\_num=message[temp1-1]

print "Sqn num is",sqn\_num

sqn\_num\_buffer.append(sqn\_num)

data\_buffer.append(message[0:temp1-1])

elif master\_counter<100: #for double digit sqn num

sqn\_num=message[temp1-2:temp1]

print "Sqn num is",sqn\_num

sqn\_num\_buffer.append(sqn\_num)

data\_buffer.append(message[0:temp1-2])

elif master\_counter<1000: #for triple digit sqn num

sqn\_num=message[temp1-3:temp1]

print "Sqn num is",sqn\_num

sqn\_num\_buffer.append(sqn\_num)

data\_buffer.append(message[0:temp1-3])

elif master\_counter<10000: #for four digit sqn num

sqn\_num=message[temp1-4:temp1]

print "Sqn num is",sqn\_num

sqn\_num\_buffer.append(sqn\_num)

data\_buffer.append(message[0:temp1-4])

elif master\_counter<100000: #for five digit sqn num

sqn\_num=message[temp1-5:temp1]

print "Sqn num is",sqn\_num

sqn\_num\_buffer.append(sqn\_num)

data\_buffer.append(message[0:temp1-5])

else:

fin\_recieved=True

return sqn\_num,sqn\_num\_buffer,data\_buffer,fin\_recieved

def send\_ack(): #send ack

udp\_sock.sendto(ack\_message,(ip\_addr,client\_port)) #send ack

time.sleep(0.05)

print ack\_message

def empty\_buffers(data\_buffer,sqn\_num\_buffer,nack\_buffer): #empty buffers

temp3=len(data\_buffer)

for k in range(temp3-1,0,-1):

data\_buffer.pop(k)

sqn\_num\_buffer.pop(k)

#empty nack buffer

if len(nack\_buffer)>1:

temp6=len(nack\_buffer)

for k in range(temp6-1,0,-1):

nack\_buffer.pop(k)

return data\_buffer,sqn\_num\_buffer,nack\_buffer

def send\_correct\_message(from\_us):

while(1): #check for getting correct message

message, address = udp\_sock.recvfrom(buffer\_size) #get message

print message

if message==send\_again\_message: #if asked to send again

udp\_sock.sendto(ack\_message,(ip\_addr,client\_port)) #send message

time.sleep(0.05)

elif message==ok\_message:

break

def give\_window\_size(message,counter,master\_counter):

if message=="Window size?": #if window size asked

print "Sender:",address[0],", message:",message #print the in comming message

udp\_sock.sendto(window\_size,(ip\_addr,client\_port)) #send the window size

time.sleep(0.05)

counter=-1 #reinitialise counter

master\_counter=-1

else:

for i in range(buffer\_size-3):

f1.writelines(message[i]) #write incomming data

print "Data coming from ip address:",address[0]

sqn\_num=message[buffer\_size-3:len(message)]

print "sqn num is ", sqn\_num #print single digit sqn number to shell

counter=counter+1

master\_counter=master\_counter+1

return master\_counter,counter

def check\_missing(master\_counter,counter,sqn\_num\_buffer,data\_buffer,packet\_lost,from\_us,fin\_recieved):

temp3=len(data\_buffer)

t=master\_counter-counter

i=1

while i<temp3:

if sqn\_num\_buffer[i]!=str(t+i-1): #if sqn num missing

packet\_lost=True

#ask for missing packet

while(1):

udp\_sock.sendto(nack\_message+str(t+i-1),(ip\_addr,client\_port)) #send nack

time.sleep(0.05)

message, address = udp\_sock.recvfrom(buffer\_size) #get packet

master\_counter +=1 #increment master counter

counter += 1 #increment counter

sqn\_num,sqn\_num\_buffer,data\_buffer,fin\_recieved= store\_buffer(sqn\_num\_buffer,data\_buffer,message,fin\_recieved,sqn\_num\_buffer[i],master\_counter)

sqn\_num\_buffer,data\_buffer=bubble\_sort(sqn\_num\_buffer,data\_buffer)

if sqn\_num\_buffer[i]==str(t+i-1): #if correct recieved

#send ok

udp\_sock.sendto(ok\_message,(ip\_addr,client\_port))

print ok\_message

if temp3 <= int(window\_size)+1: #while temp3 is in range

temp3 +=1 #increment temp3

break

i+=1 #increment i

if len(data\_buffer) != int(window\_size)+1: #buffer not full

if fin\_recieved==False:

while(len(data\_buffer) != int(window\_size)+1) and fin\_recieved==False:

#send nack

udp\_sock.sendto(nack\_message+str(t+i-1),(ip\_addr,client\_port)) #send nack

print str(t+i-1)

time.sleep(0.05)

message, address = udp\_sock.recvfrom(buffer\_size) #get packet

packet\_lost=True

master\_counter +=1 #increment master counter

counter += 1 #increment counter

sqn\_num,sqn\_num\_buffer,data\_buffer,fin\_recieved= store\_buffer(sqn\_num\_buffer,data\_buffer,message,fin\_recieved,sqn\_num\_buffer[i-1],master\_counter)

sqn\_num\_buffer,data\_buffer=bubble\_sort(sqn\_num\_buffer,data\_buffer)

if sqn\_num\_buffer[i]==str(t+i-1): #if correct recieved

#send ok

udp\_sock.sendto(ok\_message,(ip\_addr,client\_port))

print ok\_message

if temp3 <= int(window\_size)+1: #while temp3 is in range

temp3 +=1 #increment temp3

i +=1

if packet\_lost==False:

#print ack

send\_ack()

from\_us=ack\_message

#send No more messgae

if packet\_lost==True:

udp\_sock.sendto(nm\_message,(ip\_addr,client\_port))

print nm\_message

return sqn\_num\_buffer,data\_buffer,from\_us,packet\_lost,master\_counter,counter,fin\_recieved

print "ready to serve"

#main while loop

while True:

f1=open(file\_to\_save,open\_mode) #open file for appending

message, address = udp\_sock.recvfrom(buffer\_size) #recieve message

if message: #if message recieved

master\_counter,counter=give\_window\_size(message,counter,master\_counter)

while True: #secondary loop

temp=select.select([udp\_sock],[],[],timeout)

if temp[0]: #if messages still coming

packet\_lost=False

master\_counter += 1

counter += 1

message, address = udp\_sock.recvfrom(buffer\_size)

#storring function

sqn\_num,sqn\_num\_buffer,data\_buffer,fin\_recieved=store\_buffer(sqn\_num\_buffer,data\_buffer,message,fin\_recieved,sqn\_num,master\_counter)

#print "time difference=",time\_difference

if (counter==int(window\_size)-1 ): #if window size reached

temp3=len(data\_buffer) #length of data buffer

#reordering of packets

sqn\_num\_buffer,data\_buffer=bubble\_sort(sqn\_num\_buffer,data\_buffer)

#check for missed packets

sqn\_num\_buffer,data\_buffer,from\_us,packet\_lost,master\_counter,counter,fin\_recieved= check\_missing(master\_counter,counter,sqn\_num\_buffer,data\_buffer,packet\_lost,from\_us,fin\_recieved)

send\_correct\_message(from\_us)

packet\_lost=False

f1.writelines(data\_buffer[1:temp3])

#empty buffers

data\_buffer,sqn\_num\_buffer,nack\_buffer=empty\_buffers(data\_buffer,sqn\_num\_buffer,nack\_buffer)

counter=-1 #reinitiase counter

elif not temp[0]: #no incoming data

#check for packet loss

sqn\_num\_buffer,data\_buffer,from\_us,packet\_lost,master\_counter,counter,fin\_recieved= check\_missing(master\_counter,counter,sqn\_num\_buffer,data\_buffer,packet\_lost,from\_us,fin\_recieved)

#to see if correct message sent

if fin\_recieved==False or (master\_counter+1)%int(window\_size)!=0: #if not end of transfer

send\_correct\_message(from\_us)

if packet\_lost==False:

#reordering of packets

sqn\_num\_buffer,data\_buffer=bubble\_sort(sqn\_num\_buffer,data\_buffer)

temp3=len(data\_buffer) #length of data buffer

print "remaining buffer size is",temp3

#write data in to file

if counter >=0:

f1.writelines(data\_buffer[1:temp3])

#clean up buffers

data\_buffer,sqn\_num\_buffer,nack\_buffer=empty\_buffers(data\_buffer,sqn\_num\_buffer,nack\_buffer)

counter=-1 #reinitialise counter

#close file

if packet\_lost==False or fin\_recieved==True:

f1.close()

print "file closed"

if fin\_recieved==True:

print fin\_message

fin\_recieved==False #reset fin recieved

break #break secondary loop

udp\_sock.close() #close socke

Sender side code:

#created by sardar hassan and abdul rehman qamar aftab

from socket import \*

import time

import sys

#set server IP, port number

serverName="127.0.0.1"

serverPort=55000

my\_port=50000

file\_name="to\_send.mp3" #🡨-----------------------------------------------file to transfer

open\_mode="rb" #<----------------------------------------------------------file open mode

buffer\_length=495 #set packet payload length to 495 bytes of packet

remaining=5

sqn\_num="0"

counter=0

ack\_message="ACK"

nack\_message="NACK"

send\_again\_message="Send again"

ok\_message="OK"

nm\_message="NM"

fin\_message="FIN"

temp=0

test=False

data\_buffer=[""]

sqn\_num\_buffer=[""]

#temp for sqn num

#temp 3 for data buffer

#temp 4 for nack sqn

clientSocket = socket(AF\_INET,SOCK\_DGRAM)

clientSocket.bind((serverName,my\_port))

def send\_data(sqn\_num\_buffer,data\_buffer): #to send data to reciever

temp3=len(data\_buffer)

for k in range(1,temp3):

if sqn\_num\_buffer[k]!="12" and sqn\_num\_buffer[k]!="100":

clientSocket.sendto(data\_buffer[k] + sqn\_num\_buffer[k],(serverName,serverPort)) #send data one by one

time.sleep(0.05) #wait

#clientSocket.sendto(data\_buffer[k] + sqn\_num\_buffer[k],(serverName,serverPort)) #send data one by one

#time.sleep(0.05) #wait

print "sqn num is",sqn\_num\_buffer[k] #print syn number

def check\_message(sqn\_num\_buffer,data\_buffer): #if incomming message is correct

message,address=clientSocket.recvfrom(buffer\_length)

print "entered check message"

while(1): #check for getting correct message

if message[0:len(nack\_message)]==nack\_message: #if nack recieved

temp3=len(data\_buffer)

print message

while(1):

req\_sqn=message[len(nack\_message):len(message)]

for i in range(1,temp3):

if sqn\_num\_buffer[i]==req\_sqn:

while(1):

clientSocket.sendto(data\_buffer[i] + sqn\_num\_buffer[i],(serverName,serverPort))

time.sleep(0.05) #wait

message,address=clientSocket.recvfrom(buffer\_length)

if message==ok\_message:

break

message,address=clientSocket.recvfrom(buffer\_length)

if message==nm\_message: #if nm message recieved

clientSocket.sendto(ok\_message,(serverName,serverPort))

time.sleep(0.05) #wait

break

break

elif message==ack\_message: #if ack recieved

print message

clientSocket.sendto(ok\_message,(serverName,serverPort))

time.sleep(0.05)

print ok\_message

break

def empty\_buffers(sqn\_num\_buffer,data\_buffer):

temp3=len(data\_buffer)

for j in range(temp3-1,0,-1):

data\_buffer.pop(j)

sqn\_num\_buffer.pop(j)

print "remaining sqn num buffer:",sqn\_num\_buffer

def ask\_window\_size():

#ask for window size

print"Asking for window size..."

prompt="Window size?"

clientSocket.sendto(prompt,(serverName,serverPort))

time.sleep(0.05)

#get and store window size

message,address=clientSocket.recvfrom(buffer\_length)

print "From reciever window size is",message

return message

window\_size=ask\_window\_size()

f = open(file\_name, open\_mode) #open file for reading

data=f.read(buffer\_length) #get data to send

while data: #while data remaining to send

#store data and sqn numbers in buffer

data\_buffer.append(data)

sqn\_num\_buffer.append(sqn\_num)

if counter==int(window\_size)-1:

temp3=len(data\_buffer)

send\_data(sqn\_num\_buffer,data\_buffer) #send data

check\_message(sqn\_num\_buffer,data\_buffer) #check for incominng message

#pop sent sqn number and data

empty\_buffers(sqn\_num\_buffer,data\_buffer) #empty the buffers

counter=-1 #reinitialise counter

temp=int(sqn\_num)+1 #increment syn num

sqn\_num=str(temp)

data=f.read(buffer\_length) #get remaining data

counter += 1

#for last few packets

print "remaining counter is:", counter

if counter>0: #if any remaining packets

temp3=len(data\_buffer) #data buffer length

#send data and sqn number one by one from buffer when window size reached

send\_data(sqn\_num\_buffer,data\_buffer)

#pop sent sqn number and data

empty\_buffers(sqn\_num\_buffer,data\_buffer)

#send fin

clientSocket.sendto(fin\_message,(serverName,serverPort))

time.sleep(0.05)

#recieve messages

check\_message(sqn\_num\_buffer,data\_buffer) #check for incominng message

clientSocket.close() #close socket

print "File closed”