Encrypted Network Communication with Python

By
George S. Lansdown

Advanced Higher Computer Science Project

Candidate Number:
062727721
North Berwick High School:
Centre No. 5556031

Contents:

PROJECT PLAN

What is the issue that my solution hopes to solve?

During the 2016 american presidential election a series of high profile hacks were made to the email servers of the democrat national convention. These hacks were used to undermine some of the statements made by the democrat candidate and her cabinet and they are likely to have influenced the results of the election.
In the modern world, communication is required to encourage progress. These hacks are an example of the issues that arise from using a communication method which is insecure and easily breached. This is the problem which my solution will aim to help. My solution will be a chat client which allows communication between two computers with a network connection. It will feature an encryption algorithm to prevent the access to the messages being sent.

Time allocation

The most challenging parts of the development process will be the program design and the implementation. These will have more time dedicated than the rest of the parts of the process.

As intermittent testing is being conducted alongside the implementation, this requires less time and will just be gathering evidence that the solution meets the requirements.

Initial Outline project plan

Task	Time	Date to finish
Project proposal	6 hours	15/12/16
Requirement specification	4 hours	20/12/16
Interface design	4 hours	27/12/16
Test plan	4 hours	4/1/17
Program design	4 hours	16/1/17
implementation	10 hours	5/2/17
Final testing	2 hours	12/2/17
Evaluation	4 hours	15/2/17

Initial Gantt

	15/12/16	18/12/16	20/12/16	27/12/16	4/1/17	16/1/17	5/2/17	12/2/17	15/2/17
User survey									
Investigate technical requirements									
Write up requirement specification									
Draw interface wireframe									
Write test plan involving interface									
Write pseudocode for program									
Write program									
Final testing									
Evaluation									

Outline project plan (changed on 21/02/2017)

Task	Time	Date to finish
Project proposal	6 hours	15/12/16
Requirement specification	4 hours	20/12/16
Interface design	4 hours	27/12/16
Test plan	4 hours	4/1/17
Program design	8 hours	11/1/17
Implementation	15 hours	14/3/17
Intermittent testing	15 hours (same as implementation)	14/3/17
Final testing	2 hours	15/3/17
Evaluation	4 hours	21/3/17

Gantt Chart (changed on 21/02/2017**)**

	15/12/16	18/12/16	20/12/16	27/12/16	4/1/17	11/1/17	15/3/17	21/3/17
User survey								
Investigate technical requirements								
Write up requirement specification								
Draw interface wireframe								
Write test plan involving interface								
Write pseudocode for program								
Write program								
Test program								
Evaluation								
Documentation								

Intermediate targets (and what they involve):

Initial survey:
This involves asking potential users of my solution things about their ideal solution. I plan to ask them things like “Is encryption important to you?”. Several user surveys will be done through the development process to ensure that the solution is suitable for the users and meets the requirements.

Investigate technical requirements:
This involves doing research on my chosen environment and libraries. This will allow me to better estimate the requirements for the project.

Write up requirement specification:
Document the requirements of the final program as found in the initial user survey and the technical requirements.

Draw interface and wireframe:
Create interface based on user survey. Ideally it will meet all the requirements of the users however, if users have disagreeing feedback then this will be difficult to implement.

Write test plan involving interface:
Write a plan to test all available inputs for the interface.

Write pseudocode for program:
Write out sections of program in pseudocode before starting actual programming.

Write program:
Start to develop program in my chosen language.

Intermittent testing program:
Testing parts of program during development.

Final testing of program:
Using test plan, actually perform tests to ensure program works as expected and gather evidence. This means testing the requirements from the requirement specification and the test plan.

Evaluation:
What went well, what didn’t and how it could be better next time.

User survey:

- “Do you use chat clients?”
    - Ryan: “Yes, i use them regularly.”
    - Seb: “Yes.”

- “Is encryption important to you?”
    - Ryan: “Yes, i’d rather use a chat client with good encryption.”
    - Seb: “Encryption is very important to me.”

- “What features would you like to see in a chat client?”
    - Ryan: “Importing and exporting the chat log”
    - Seb: “Conversations with more than one person”

- “Do you prefer the interface to be functional or pretty?”
    - Ryan: “I like a pretty interface, but overall I’d prefer a functional one”
    - Seb: “I would much rather have a well designed and functional interface”

Initial Project Proposal

My project will be a peer-to-peer networked chat application with end to end encryption. The idea is that two people with copies of my project should be able to connect to each other and send messages via a network connection. My solution should encrypt the messages before they are sent and decrypt them afterwards to be displayed in plain text to the user.
The end user group should be people who have an interest in networking and encryption. And who want to communicate without having their messages intercepted and read.

I am planning to use Python to create my solution, this is because i have experience using Python and various libraries the solution requires. This reduces the time scale because otherwise I’d have to spend time learning python. However, some extra time is required to learn about encryption. To do this, i plan to read the book “Everyday Cryptography” by Keith Martin.

Initial Feasibility Study

My solution will require two computers that run Python and are networked together. It will not be an especially memory intensive program in RAM or backing storage.
The user interface for my project should be fairly simple to use and navigate. I will do this by planning the interface extensively.

Revised Project proposal (changed on 21/02/2017)

My project will be a networked chat application with end to end encryption. This involves creating a program with two different modes, client and server. The idea is that an copy of my project in client mode will be able to connect to a copy of my project in server mode and then they will be able to send messages to each other via a network connection. My solution should encrypt the messages before they are sent and decrypt them when received to be displayed in plain text to the user.
The end user group should be people who have an interest in networking and encryption. And who want to communicate without having their messages intercepted and read. Because the end user group would be one with a strong knowledge of networking terms. This means that the interface can be slightly more technical than would be accepted by the average computer user.

I am planning to use Python to create my solution, this is because I have experience using Python and various libraries the solution will require. This reduces the time scale because otherwise I'd have to spend time learning about the python language and the libraries needed. However, some extra time is required to learn about encryption. To do this, I will read the book "Everyday Cryptography" by Keith Martin. This will allow me to make an informed choice about what algorithm to use to prevent the messages sent between copies of the program being read.

Revised Feasibility Study (changed on 21/02/2017)

Equipment requirements:

- Two computers running Python (at least 2.7) with default libraries installed
- A reliable network connection between them (if the network has a firewall which blocks certain port, it is unlikely to allow for communication to work)
- Moderate hard drive space on computers (<1GB)
- "Everyday Cryptography" by Keith Martin (to reference)

It may be slightly RAM intensive as the program requires multiple threads running concurrently, however it is unlikely to use a lot of backing storage. It will however, also output the chat log after each conversation - this is likely to be a small file - but it depends upon the length of the conversation and size of messages.

From reading "Everyday Cryptography" I found that due to the complexity of modern encryption algorithms. They are usually very difficult to implement without a very strong knowledge of Maths. This means I have decided to use a modified version of the ROT13 algorithm. ROT13 is fairly simple algorithm to implement but it should

prevent the messages being read if intercepted by an attacker. It also has the added benefit of being it's own inverse. This means that it can encrypt and decrypt using the same algorithm, saving hard drive space.

The program will only allow communications with two people at once. This is because the Socket library in Python struggles when more than two people try and communicate.
Additionally, if the people wanting to communicate aren't on the same network then there are a series of other requirements. Things like neither of the communicating computers having a firewall which blocks ports on the network and a router with port forwarding set up on the port in use.

All of the languages and libraries I plan to use are available free of charge. I also have access to two networked computers in my school which are free for me to use. These both mean that my project plan is economically feasible.

As the people interviewed for the user survey suggested that a functional interface was better than an attractive one, no external images or designs will be used. This means that no content in my solution will have copyrights applied. This, coupled with the fact that I will not be selling my solution means that it is a legally feasible project.

The SQA requires that my program has the following:

- An interface appropriate for your users that validates all inputs
- Interfacing with stored data
- A minimum of two of the following:
    - 2-D arrays, arrays of records or linked lists
    - A binary search, a sort algorithm or other coding of similar complexity
    - recursion

My solution should have:

- An interface appropriate for your users that validates all inputs
- Interfacing with stored data by exporting the chat log after each conversation
- 2-D arrays for storing a timestamp and all the messages from my program
- My networkInterface class will have similar complexity to a search/sort algorithm
- Recursion to ensure that the user interface stays updated

REQUIREMENTS SPECIFICATION

From the research detailed in the feasibility study in the previous section. To solve this problem, my solution should have the following requirements:

- Send strings of text between two computers on a local network
- Encrypt and decrypt a message string using a suitable algorithm
- Export chat logs after every connection
- Have a clear, easy to use interface
- Take user input and ensure its validity
- Prevent intercepted communications being read

In scope:

- The users should be able to connect and send message to one another
- The messages will be encrypted after the message entered and decrypted by the other client after it is received.
- The users should be able to access a chat log text file after the chat has ended to read it.

Out of scope:

- They can only send text messages not things like pictures
- They cannot access the encrypted message, they can only access the messages when they've been decrypted.
- They cannot have conversations with more than one person
- Users can only export a chat log file after a conversation, they cannot import one.

Boundaries:

The encryption algorithm is not of a very high complexity. This is because it would take much too long to use a contemporary encryption algorithm. This means that intercepted messages would not be as secure.

The socket library I am using makes implementing communications with more than one person difficult. This means that my solution is limited to communications with just one other person at once.
After consideration however, I believe this aids the security of my program. Having the ability for other connections to be made would allow for an intruder to join the private conversation of two people with the ability to read their messages in plain text.

Communication between versions of my program is limited to the local network. Unless port forwarding has been set up on a network which would allow communication over the wider internet to occur. Port forwarding is when someone specifies a port on a network (for example 1234) which is allowed to receive and send packets to outside the network.

The users can only export a text file with their conversations in rather than import one. This would be functionally useless as they could read the chat log file in a text editor anyway and would take much more implementation.

End users:
The end user group should be people who have an interest in networking and encryption. And who want to communicate without having their messages intercepted and read. Possibly people who study computer science.

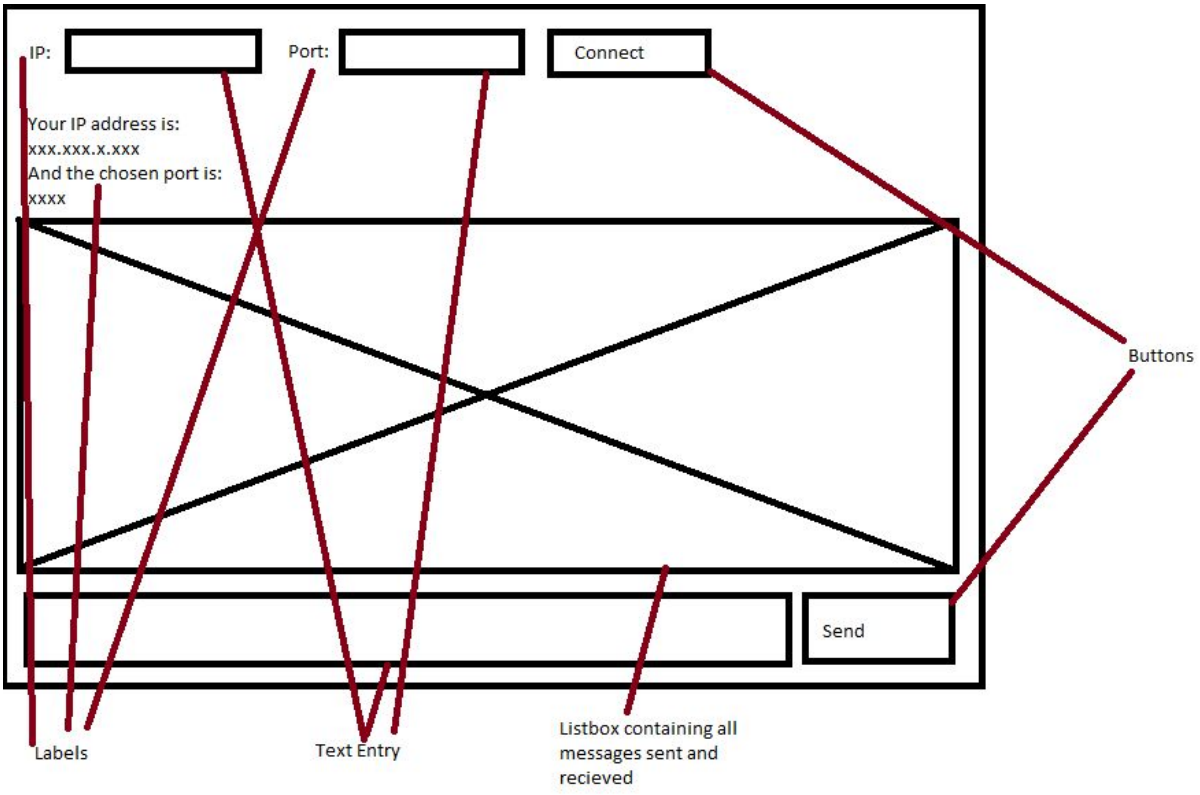
Having this end user group means that the interface can focus on being functional rather than necessarily attractive looking. This was mentioned in the user survey as well, the people queried said that they would much prefer a functional interface.

Inputs and outputs:

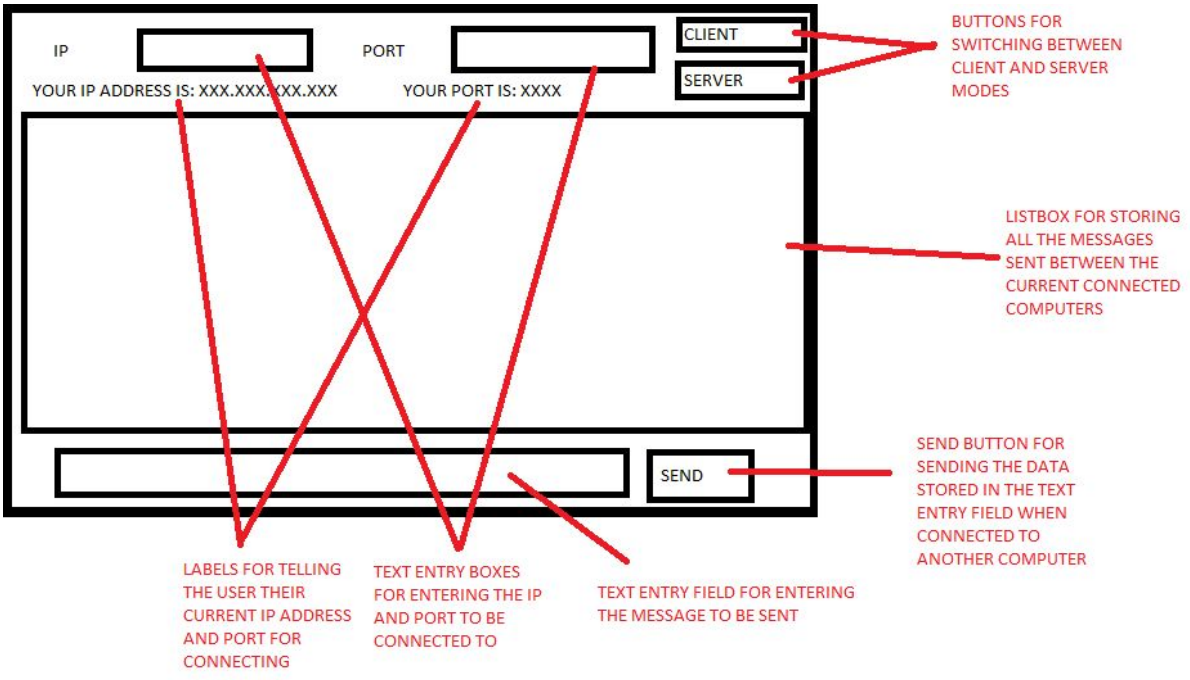
Input	**Datatype**	**Use**
Message entry box	String	For entering the messages to be sent between connected versions of my program
IP address entry box	Integers arranged in form XXX.XXX.XXX.XXX	For use when the program is in client mode, used for entering the address of the machine hosting the server
Port entry	Integer in range 0-65535	For use when program is in client mode, used for entering the specified port of the server

Output	**Datatype**	**Use**
Debug messages to user	String	Used to make the user aware of things occurring with the program. For example, when a connection is made.
Messages from other connections	String	Send to user when a message is received from a connection.
Messages sent to other connections	String	Messages that have been sent will be outputted to to the user as well to maintain a ‘chat’

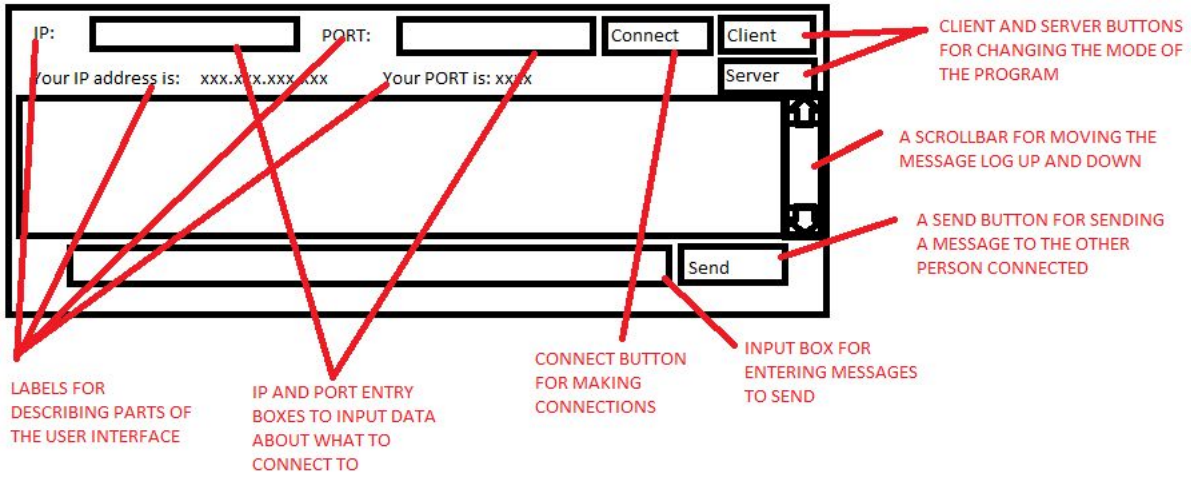
INTERFACE DESIGN



This was the initial design, it was a poor design because it featured large amounts of empty space. It also revolved around the idea that client and server modes could run concurrently without issues.



This is the second design for the interface, it features separate client and server buttons. These are required to change the mode of the program. This change was required after issues arose with the concurrency of the program.



This is the final interface design which matches the final program's interface.

I have redesigned this interface as the function of the program has changed due to the addition of concurrency. This addition allows for the interface to run separately to the network functions of the program, this allows for the mode of the network to be informed by the interface instantly rather than waiting for a timeout. This change in function requires client and server buttons added to the interface.

The IP and Port text boxes should only be accessible when in client mode as the server can only be hosted on the ip address allocated to the machine. This means that the ip and port text boxes should be 'disabled' when the program is in server mode.

The interface will all be on one window as there aren't enough options and having two windows might make the interface look cleaner, but overall I think it would confuse the users.

A more attractive interface may attract more users, however, this program was designed with technical people in mind who value functionality over visual appeal. This is something that could be changed later on, perhaps in perfective maintenance.

TEST PLAN

These are to test all of the entry elements in the interface.

IP entry textbox:

Entry Data	Type of data (Extreme/Exceptional/Normal)	Result
“192.168.1.5”	Normal	?
“yellow”	Exceptional	?
“192.168.1.1”	Extreme	?
10	Exceptional	?
[space]	Exceptional	?
“192.168.1.255”	Extreme	?

Port entry textbox:

Entry Data	Type of data (Extreme/Exceptional/Normal)	Result
6969	Normal	?
“brown”	Exceptional	?
“6666”	Exceptional	?
[space]	Exceptional	?

Message entry textbox:

Entry Data	Type of data (Extreme/Exceptional/Normal)	Result
“Hello”	Normal	?
“4565463626”	Normal	?
905432gfdijhgfdskjl	Extreme	?
[space]	Extreme	?

There are also the subjective requirements from earlier in the report:

- Send strings of text between two computers on a local network
- Encrypt and decrypt a message string using a suitable algorithm
- Export chat logs after every connection
- Have a clear, easy to use interface
- Take user input and ensure its validity
- Prevent intercepted communications being read

These will need to be tested in different ways, due to their nature.
For example, the “Have a clear, easy to use interface.” will need to be tested using a user survey because of its subjectivity.

PROGRAM DESIGN

__main__.py PSEUDOCODE:

```
userInterface thread
    Create instance of userInterface object
    Configure user interface
    start create instance of interactions thread with argument of userinterface
object and start thread

networkInterface thread
    Create instance of networkInterface thread
    Keep looping
        If mode variable from __init__ file is client
            Set mode of networkInterface to client
        If mode variable from __init__ file is server
            Set mode of networkInterface to server

Interactions thread
    If mode variable from __init__ isn't client
        Disable ip and port textboxes and connect/disconnect button
    else
        Enable ip and port textboxes and connect/disconnect button

    Update the messagebox in the userinterface with changes in the messages
    array in the __init__
    Update the chatlog file in the userinterface with changes in the messages
array in __init__

    Wait 0.5 seconds

    If userInterface still works then call the interactions thread again
    Otherwise close the chatlog file


If file is executed as the main file
    Create instance of userInterface thread and start
    Create instance of networkInterface thread and start
```

NetworkInterface.py PSEUDOCODE

```
Class NetworkInterface
    Method constructor
        currentMessage = “”
    Method client
        Create connection
        While mode = “client”
            If connected
                Connect to server
                If currentMessage is different to __init__.currentMessage
                    Send __init__.currentMessage to server
                    currentMessage = __init__.currentMessage
                Data = recieve messages from server
                If data exists
                    Add data to __init__.messages
        Close connection
        __init__.mode = None
    Method server
        Create connection
        Wait for clients
        While connected to a client
            If currentMessage is different to __init__.currentMessage
                        Send __init__.currentMessage to server
                        currentMessage = __init__.currentMessage
                    Data = recieve messages from server
                    If data exists
                        Add data to __init__.messages
        Close connection
        __init__.mode = None
```

Interface.py PSEUDOCODE

```
Class UserInterface
    Method constructor
        Create window
        Create buttons for connecting and sending a message
        Create textboxes for ip, port, and message
        Create listbox window for storing message log
        Create scrollbar for scrolling listbox
        Create labels for describing everything

        IPAddressLabel = __init__.IP
        PortLabel = __init__.Port
```

```
Method changeIPandPort
    If ip and port are valid
        __init__.ip = iptextbox
        __init__.port = porttextbox
        __init__.connected = True
    Else add error message to __init__.messages
Method changemessage
    __init__.currentMessage = messagetextbox
Method changeMode
    If client button pressed
        __init__.mode = “client”
    If server button pressed
        __init__.mode = “server”
```

__init__.py PSEUDOCODE

Create variables for IP(string), Port(int), mode(string), messages(2D array of strings), currentMessage(strings) and connected(boolean)

Create file to store chatlog with a timestamp in the title

```
Function for validating ip addresses
    Make list of integer parts of ip address between “.”s
        If any of the integer parts are out of range 0-255 then return false
        If if there are not 4 parts to the ip address then return false
        If port is out of range 1-65535 then return false
    Return true

Function for encryption(string x)
    For all letters in x
        if letter is not in alphabet
            Add letter to newstring
        else:
            Rotate letter by 13 characters in the alphabet
            If letter is lowercase
                Make letter uppercase
                Add letter to new string
            If letter is uppercase
                Make letter lowercase
                Add letter to new string
    Return newstring
```

IMPLEMENTATION

Highlighted lines are lines that meet the SQAs requirements of things like recursion, file handling, and 2D arrays.
Blue highlight is file operations
Green highlight is recursion
Yellow highlight is 2D arrays

__INIT__.py File:

```
'''_
__INIT__.PY
GEORGE LANSDOWN
CONTAINS VARIALES AND FUNCTIONS THAT NEED TO BE ACCESSED BY ALL OTHER PARTS
'''
import socket, time #DEPENDANCIES


IP = socket.gethostbyname(socket.gethostname())     #SETS IP ADDRESS TO MACHINES
CURRENT IP ADDRESS
PORT = 6969 #SETS PORT TO 6969
mode = None
messages = []   #USED FOR STORING ALL USER SIDE INFORMATION, GETS DUMPED TO
CHATLOG WHEN PROGRAM IS CLOSED
currentMessage = None   #USED FOR SENDING MESSAGES
connected = False   #VARIABLE FOR STATUS OF CLIENT CONNECTIONS

chatlog = open("CHATLOG "+time.strftime("%d-%b-%Y %H.%M.%S", time.gmtime())+".txt", "w")
#!!!!!!!!!FILE OPERATIONS!!!!!!!!    OPENS CHATLOG FILE WITH TIME STAMP


def validateIPandPort(ip, port):    #FUNCTION FOR VALIDATING IP AND PORT PAIR
    if ip != "" and port != "":
        try:
            IPsplit = ip.split(".") #SPLITS IP INTO LIST WITH INTEGER VALUES
            for each in IPsplit:
                if int(each) < 0 or int(each) > 255:    #ENSURES THAT INTEGER VALUES EXIST WITHIN
0 TO (2^8-1)
                    return False
            if len(IPsplit) == 4:   #ENSURES THAT ONLY 4 COMPONENTS EXIST x.x.x.x is valid but
x.x.x is not
                if int(port) > 0 and int(port) < 65536: #ENSURES PORT EXISTS IN RANGE
                    return True
        except ValueError:
            return False

def encrypt(x):     #FAIRLY UNIMPRESSIVE ENCRYPTION ALGORITHM - ROT13 + INVERTING
UPPERCASE AND LOWERCASE - BLUE BECOMES oyhr
```

```
    alphabet =
{"a":"n","b":"o","c":"p","d":"q","e":"r","f":"s","g":"t","h":"u","i":"v","j":"w","k":"x","l":"y","m":"z","n":"a","o":"b",
"p":"c","q":"d","r":"e","s":"f","t":"g","u":"h","v":"i","w":"j","x":"k","y":"l", "z":"m"} #DICTIONARY OF
LETTER SHIFTS TO ENSURE THAT CORRECT CHARACTERS ARE USED
    newstring = ""  #EMPTY NEW STRING
    for f in x:
        try:
            if f.isupper(): newstring += alphabet[f.lower()]
            else: newstring += alphabet[f].upper()
        except KeyError: #WHEN CHARACTER ISN'T IN DICTIONARY KEYERROR IS RAISED
            newstring += f #IGNORES SHIFT
    return newstring
```

__main__.py File:

```
'''
__MAIN__.PY
GEORGE LANSDOWN
COMBINES ALL OTHER ELEMENTS OF PROGRAM
'''

import __init__, userInterface, networkInterface, socket, threading, time       #DEPENDANCIES
from Tkinter import *


def network():              #NETWORK INTERFACE THREAD
        connection = networkInterface.connection()      #CREATES NETWORK INTERFACE OBJECT
        while True:
            if __init__.mode == "CLIENT":       #PUTS INTO CLIENT MODE
                connection.client()
            elif __init__.mode  == "SERVER":    #PUTS INTO SERVER MODE
                connection.server()


def user():             #USER INTERFACE THREAD - SETS UP INTERFACE
        root = Tk()
        root.wm_title("Network Chat Client")
        root.resizable(width=False, height=False)
        app = userInterface.Application(master=root)
        q = threading.Thread(target=interactions, args=[app])   #CREATE INTERACTIONS THREAD
WITH ACCESS TO USER INTERFACE ELEMENTS
        root.after(50, q.start) #STARTS INTERACTIONS THREAD AFTER 50ms
        app.mainloop() #STARTS USER INTERFACE

def interactions(x):            #DATA DISPLAY THREAD    USES RECURRSION

        if __init__.mode == None:                   #UPDATES STATUS OF MODE BUTTONS
                x.connect["state"] = DISABLED
                x.ipEntry["state"] = DISABLED
                x.portEntry["state"] = DISABLED
                x.clientButton.deselect()
                x.serverButton.deselect()
```

```
    elif __init__.mode == "CLIENT":
        x.connect["state"] = "normal"
        x.ipEntry["state"] = "normal"
        x.portEntry["state"] = "normal"
    elif __init__.mode == "SERVER":
        x.connect["state"] = DISABLED
        x.ipEntry["state"] = DISABLED
        x.portEntry["state"] = DISABLED
    else:
        x.connect["state"] = DISABLED
        x.ipEntry["state"] = DISABLED
        x.portEntry["state"] = DISABLED

    if __init__.connected:                    #CHANGES VALUE OF CONNECT BUTTON WHEN IT IS
CONNECTED TO A SERVER
        x.connect["text"] = "Disconnect"
    else:
        x.connect["text"] = "Connect"

    for f in range(x.messageLog.size(), len(__init__.messages)):          #UPDATES USER
INTERFACE MESSAGE LOG AND CHAT LOG FILE WHEN THE __INIT__.MESSAGES LIST
CHANGES
        x.messageLog.insert(END, str(__init__.messages[f][0])+"    "+__init__.messages[f][1])
        __init__.chatlog.write(str(__init__.messages[f][0])+"    "+__init__.messages[f][1]+"\n")

    time.sleep(0.5) #TIME DELAY SO FEWER PROCESSES ARE USED WHEN NOT REQUIRED


    if s.isAlive():        #IF USER INTERFACE THREAD STILL ALIVE
        interactions(x)        #RECURRSION TO MEET REQUIREMENTS
!!!!!!!!!!RECURSION!!!!!!!!!!
    else:
        __init__.chatlog.close()       #OTHERWISE CLOSE CHATLOG FILE AS NO CHANGES
WILL BE MADE



if __name__ == "__main__":      #IF BEING EXECUTED AS MAIN FILE
    t = threading.Thread(target=network)    #CREATE THREADS
    s = threading.Thread(target=user)

    s.start()      #ACTIVATE THREADS
    t.start()
```

networkInterface.py File:

```
'''
NETWORKINTERFACE.PY
GEORGE LANSDOWN
DEALS WITH COMMUNICATIONS BETWEEN TWO COMPUTERS
```

```
__init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Switched to client
mode"])
LINES WITH THIS ARE USED FOR APPENDING TIMESTAMPED MESSAGES IN
__INIT__.MESSAGES TO SHOW USER
!!!!!THIS USES 2D ARRAYS!!!!!

'''
import __init__, socket, threading, time        #DEPENDANCIES

class connection(object):       #CONNECTION CLASS WITH TWO MODES
    def __init__(self):     #CONSTRUCTOR
        self.currentMessage = None #CREATES LOCAL CURRENTMESSAGE VARIABLE TO
COMPARE TO __INIT__ ONE

    def client(self):       #CLIENT MODE
        self.connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #CREATES
SOCKET CONNECTION
        __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Switched
to client mode"]) #WRITES TIMESTAMPED MESSAGE ABOUT CLIENT MODE
        while __init__.mode == "CLIENT":
            if __init__.connected:
                try:    #ERROR HANDLING
                        self.connection.connect((__init__.IP, int(__init__.PORT)))
                        __init__.messages.append([(time.strftime("%H:%M:%S",
time.gmtime())),"[DEBUG] Connected to "+str(__init__.IP)])
                        __init__.connected = True
                        while __init__.connected:
                            if self.currentMessage != __init__.currentMessage and
__init__.currentMessage != "":
                                self.currentMessage = __init__.currentMessage
                                self.connection.send(__init__.encrypt(self.currentMessage))
#SENDS ENCRYPTED MESSAGE

__init__.messages.append([(socket.gethostbyname(socket.gethostname()),
time.strftime("%H:%M:%S", time.gmtime())),self.currentMessage])
                        else:
                            self.connection.send("@]32[/")  #RANDOM STRING SENT TO
SIGNIFY BLANK MESSAGE TO AVOID ISSUES WITH EMPTY PACKETS
                        self.data = self.connection.recv(1024)
                        if self.data and self.data.split("/")[0] != "@]32[":    #ENSURES
MESSAGE ISNT 'BLANK'
                            __init__.messages.append([(__init__.IP,
time.strftime("%H:%M:%S", time.gmtime())),__init__.encrypt(self.data)])
                except socket.error:    #ACCEPT ERROR ABOUT SERVER REFUSING
CONNECTION
                        __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),
"[DEBUG] Socket error, cannot connect"])
                finally:        #ENSURES CONNECTIONS ARE CLOSED PROPERLY
                        __init__.connected = False
                        __init__.mode = None
                        self.connection.close();
```

```
    def server(self):       #SERVER MODE
        self.connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Switched
to server mode"])
        self.connection.bind((socket.gethostbyname(socket.gethostname()), int(__init__.PORT)))
        __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Hosting
on "+socket.gethostbyname(socket.gethostname())])
        self.connection.listen(1)
        self.conn, self.addr = self.connection.accept()
        __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG]
Connection from "+str(self.addr[0])])
        try:
            while self.conn and __init__.mode == "SERVER":
                if self.currentMessage != __init__.currentMessage and __init__.currentMessage != "":
                    self.currentMessage=__init__.currentMessage
                    self.conn.send(__init__.encrypt(self.currentMessage))
                    __init__.messages.append([(__init__.IP, time.strftime("%H:%M:%S",
time.gmtime())),self.currentMessage])
                else:
                    self.conn.send("@]32[/")       #RANDOM STRING SENT TO SIGNIFY BLANK
MESSAGE TO AVOID ISSUES WITH EMPTY PACKETS
                self.data = self.conn.recv(1024)
                if self.data and self.data.split("/")[0] != "@]32[":    #ENSURES MESSAGE ISNT
'BLANK'
                    __init__.messages.append([(str(self.addr[0]), time.strftime("%H:%M:%S",
time.gmtime())),__init__.encrypt(self.data)])
        except socket.error:        #ACCEPT ERROR FOR BROKEN PIPE
            __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Server
lost connection"])
        finally:    #ENSURES CONNECTIONS ARE CLOSED PROPERLY
            self.connection.close()
            __init__.mode = None

    def __del__(self):      #DESTRUCTOR TO ENSURE CONNECTION IS CLOSED
        __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Shutting
down object"])
        self.connection.close()
```

userInterface.py File:

```
'''
tkinter class for user interface
George Lansdown
'''

from Tkinter import *           #DEPENDANCIES
import networkInterface, __init__, time

class Application(Frame):
    def __init__(self, master=None):        #CONSTRUCTOR FOR USER INTERFACE CLASS
```

```
            Frame.__init__(self, master)
            self.pack()
            self.createWidgets()

        def createWidgets(self):        #CREATES USER INTERFACE ELEMENTS
            #Labels
            self.ipEntryLabel = Label(master=self, text="IP").grid(row=0, column=0)
            self.portEntryLabel = Label(master=self, text="PORT").grid(row=0, column=2)
            self.ipLabel = Label(master=self, text="Your IP address is: "+__init__.IP)
            self.ipLabel.grid(row=1, column=0)
            self.portLabel = Label(master=self, text="Your port is: "+str(__init__.PORT))
            self.portLabel.grid(row=1, column=2)

            #Entry elements
            self.ipEntry = Entry(master=self)
            self.ipEntry.insert(0, __init__.IP)
            self.ipEntry.grid(row=0, column=1)
            self.portEntry = Entry(master=self)
            self.portEntry.insert(0, __init__.PORT)
            self.portEntry.grid(row=0, column=3)
            self.messageEntry = Entry(master=self, width=70)
            self.messageEntry.bind("<Return>", lambda event:
self.changeMessage(self.messageEntry.get()))
            self.messageEntry.grid(row=3, columnspan=4)

            #Listbox
            self.scrollbar = Scrollbar(master=self, orient="vertical")
            self.messageLog = Listbox(master=self, width=100, yscrollcommand=self.scrollbar.set)
            self.scrollbar.config(command=lambda *args: apply(self.messageLog.yview, args))
            self.scrollbar.grid(column=6, row=2, rowspan=1, sticky="NS")
            self.messageLog.grid(row=2, columnspan=6)

            #Buttons
            self.connect = Button(master=self, text="connect", command=lambda:
self.changeIPandPORT(self.ipEntry.get(), self.portEntry.get()))
            self.connect.grid(row=0, column=4)
            self.send= Button(master=self, text="send", command=lambda:
self.changeMessage(self.messageEntry.get()))
            self.send.grid(row=3, column=4)
            self.clientButton = Radiobutton(master=self, text="Client", value="CLIENT", indicatoron=0,
command=lambda: self.changeMode("CLIENT"))
            self.clientButton.grid(row=0, column=5)
            self.serverButton = Radiobutton(master=self, text="Server", value="SERVER",
indicatoron=0, command=lambda: self.changeMode("SERVER"))
            self.serverButton.grid(row=1, column=5)

        def changeIPandPORT(self, x, y):        #METHOD FOR CHANGING IP AND PORT VARAIBLES
IN __INIT__
            if __init__.connected:
                __init__.connected = False
                __init__.mode = None
```

```
                __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Disconnected from server"])
            else:
                if __init__.validateIPandPort(x, y):
                    __init__.IP = x
                    __init__.PORT = y
                    __init__.connected = True
                else:
                    __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Invalid form for IP and Port, must be in the form XXX.XXX.XXX.XXX XXXX where X is Integer"])
    def changeMessage(self, x):          #METHOD FOR CHANGING THE CURRENT MESSAGE VARIABLE IN __INIT__
        __init__.currentMessage = x
    def changeMode(self, x):             #METHOD FOR CHANGING THE MODE BETWEEN CLIENT AND SERVER
        __init__.mode = x
```

__main__.py File:

```
'''
__MAIN__.PY
GEORGE LANSDOWN
COMBINES ALL OTHER ELEMENTS OF PROGRAM
'''

import __init__, userInterface, networkInterface, socket, threading, time       #DEPENDANCIES
from Tkinter import *


def network():              #NETWORK INTERFACE THREAD
        connection = networkInterface.connection()      #CREATES NETWORK INTERFACE OBJECT
        while True:
            if __init__.mode == "CLIENT":       #PUTS INTO CLIENT MODE
                connection.client()
            elif __init__.mode  == "SERVER":    #PUTS INTO SERVER MODE
                connection.server()


def user():             #USER INTERFACE THREAD - SETS UP INTERFACE
        root = Tk()
        root.wm_title("Network Chat Client")
        root.resizable(width=False, height=False)
        app = userInterface.Application(master=root)
        q = threading.Thread(target=interactions, args=[app])   #CREATE INTERACTIONS THREAD WITH ACCESS TO USER INTERFACE
        root.after(50, q.start) #STARTS INTERACTIONS THREAD AFTER 50ms
        app.mainloop() #STARTS USER INTERFACE

def interactions(x):            #DATA DISPLAY THREAD    USES RECURRSION

        if __init__.mode == None:                       #UPDATES STATUS OF MODE BUTTONS
                x.connect["state"] = DISABLED
                x.ipEntry["state"] = DISABLED
                x.portEntry["state"] = DISABLED
                x.clientButton.deselect()
                x.serverButton.deselect()
        elif __init__.mode == "CLIENT":
                x.connect["state"] = "normal"
                x.ipEntry["state"] = "normal"
                x.portEntry["state"] = "normal"
        elif __init__.mode == "SERVER":
                x.connect["state"] = DISABLED
                x.ipEntry["state"] = DISABLED
                x.portEntry["state"] = DISABLED
        else:
                x.connect["state"] = DISABLED
                x.ipEntry["state"] = DISABLED
                x.portEntry["state"] = DISABLED

        if __init__.connected:                          #CHANGES VALUE OF CONNECT BUTTON WHEN IT IS CONNECTED TO A SERVER
                x.connect["text"] = "Disconnect"
        else:
                x.connect["text"] = "Connect"

        for f in range(x.messageLog.size(), len(__init__.messages)):            #UPDATES USER INTERFACE MESSAGE LOG AND C
                x.messageLog.insert(END, str(__init__.messages[f][0])+"    "+__init__.messages[f][1])
                __init__.chatlog.write(str(__init__.messages[f][0])+"    "+__init__.messages[f][1]+"\n")

        time.sleep(0.5) #TIME DELAY SO FEWER PROCESSES ARE USED WHEN NOT REQUIRED


        if s.isAlive():         #IF USER INTERFACE THREAD STILL ALIVE

                interactions(x)         #RECURRSION TO MEET REQUIREMENTS                !!!!!!!!!!RECURSION!!!!!!!!!!
        else:
                __init__.chatlog.close()        #OTHERWISE CLOSE CHATLOG FILE AS NO CHANGES WILL BE MADE


if __name__ == "__main__":      #IF BEING EXECUTED AS MAIN FILE
        t = threading.Thread(target=network)    #CREATE THREADS
        s = threading.Thread(target=user)

        s.start()       #ACTIVATE THREADS
        t.start()
```

__init__.py File:

```
'''
__INIT__.PY
GEORGE LANSDOWN
CONTAINS VARIALES AND FUNCTIONS THAT NEED TO BE ACCESSED BY ALL OTHER PARTS
'''
import socket, time #DEPENDANCIES


IP = socket.gethostbyname(socket.gethostname())     #SETS IP ADDRESS TO MACHINES CURRENT IP ADDRESS
PORT = 6969 #SETS PORT TO 6969
mode = None
messages = []   #USED FOR STORING ALL USER SIDE INFORMATION, GETS DUMPED TO CHATLOG WHEN PROGRAM IS CLOSED
currentMessage = None   #USED FOR SENDING MESSAGES
connected = False   #VARIABLE FOR STATUS OF CLIENT CONNECTIONS

chatlog = open("CHATLOG "+time.strftime("%d-%b-%Y %H.%M.%S", time.gmtime())+".txt", "w")    #!!!!!!!!!FILE OPERATIONS!!!!!!


def validateIPandPort(ip, port):    #FUNCTION FOR VALIDATING IP AND PORT PAIR
    if ip != "" and port != "":
        try:
            IPsplit = ip.split(".") #SPLITS IP INTO LIST WITH INTEGER VALUES
            for each in IPsplit:
                if int(each) < 0 or int(each) > 255:    #ENSURES THAT INTEGER VALUES EXIST WITHIN 0 TO (2^8-1)
                    return False
            if len(IPsplit) == 4:   #ENSURES THAT ONLY 4 COMPONENTS EXIST x.x.x.x is valid but x.x.x is not
                if int(port) > 0 and int(port) < 65536: #ENSURES PORT EXISTS IN RANGE
                    return True
        except ValueError:
            return False

def encrypt(x):     #FAIRLY UNIMPRESSIVE ENCRYPTION ALGORITHM - ROT13 + INVERTING UPPERCASE AND LOWERCASE - BLUE BECOMES oy
    alphabet = {"a":"n","b":"o","c":"p","d":"q","e":"r","f":"s","g":"t","h":"u","i":"v","j":"w","k":"x","l":"y","m":"z","n"
    newstring = ""  #EMPTY NEW STRING
    for f in x:
        try:
            if f.isupper(): newstring += alphabet[f.lower()]
            else: newstring += alphabet[f].upper()
        except KeyError: #WHEN CHARACTER ISN'T IN DICTIONARY KEYERROR IS RAISED
            newstring += f #IGNORES SHIFT
    return newstring
```

userInterface.py File:

networkInterface.py File:

```
'''
NETWORKINTERFACE.PY
GEORGE LANSDOWN
DEALS WITH COMMUNICATIONS BETWEEN TWO COMPUTERS


__init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Switched to client mode"])
LINES WITH THIS ARE USED FOR APPENDING TIMESTAMPED MESSAGES IN __INIT__.MESSAGES TO SHOW USER
!!!!!THIS USES 2D ARRAYS!!!!!

'''
import __init__, socket, threading, time        #DEPENDANCIES

class connection(object):       #CONNECTION CLASS WITH TWO MODES
        def __init__(self):     #CONSTRUCTOR
            self.currentMessage = None #CREATES LOCAL CURRENTMESSAGE VARIABLE TO COMPARE TO __INIT__ ONE

        def client(self):       #CLIENT MODE
            self.connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #CREATES SOCKET CONNECTION
            __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Switched to client mode"]) #WRITES TIMESTAMPED MESSAGE ABOUT CLIENT MODE
            while __init__.mode == "CLIENT":
                if __init__.connected:
                        try:    #ERROR HANDLING
                                        self.connection.connect((__init__.IP, int(__init__.PORT)))
                                        __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Connected to "+str(__init__.IP)])
                                        __init__.connected = True
                                        while __init__.connected:
                                            if self.currentMessage != __init__.currentMessage and __init__.currentMessage != "":
                                                    self.currentMessage = __init__.currentMessage
                                                    self.connection.send(__init__.encrypt(self.currentMessage))     #SENDS ENCRYPTED MESSAGE
                                                    __init__.messages.append([(socket.gethostbyname(socket.gethostname()), time.strftime("%H:%M:%S", time.gmtime())),self.currentMessage])
                                            else:
                                                    self.connection.send("@]32[/")  #RANDOM STRING SENT TO SIGNIFY BLANK MESSAGE TO AVOID ISSUES WITH EMPTY PACKETS
                                            self.data = self.connection.recv(1024)
                                            if self.data and self.data.split("/")[0] != "@]32[":    #ENSURES MESSAGE ISNT 'BLANK'
                                                    __init__.messages.append([(__init__.IP, time.strftime("%H:%M:%S", time.gmtime())),__init__.encrypt(self.data)])
                        except socket.error:    #ACCEPT ERROR ABOUT SERVER REFUSING CONNECTION
                                        __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())), "[DEBUG] Socket error, cannot connect"])
                        finally:        #ENSURES CONNECTIONS ARE CLOSED PROPERLY
                                        __init__.connected = False
                                        __init__.mode = None
                                        self.connection.close();


        def server(self):       #SERVER MODE
            self.connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Switched to server mode"])
            self.connection.bind((socket.gethostbyname(socket.gethostname()), int(__init__.PORT)))
            __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Hosting on "+socket.gethostbyname(socket.gethostname())])
            self.connection.listen(1)
            self.conn, self.addr = self.connection.accept()
            __init__.messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Connection from "+str(self.addr[0])])
            try:
                while self.conn and __init__.mode == "SERVER":
                        if self.currentMessage != __init__.currentMessage and __init__.currentMessage != "":
                                self.currentMessage=__init__.currentMessage
                                self.conn.send(__init__.encrypt(self.currentMessage))
                                __init__.messages.append([(__init__.IP, time.strftime("%H:%M:%S", time.gmtime())),self.currentMessage])
                        else:
                                self.conn.send("@]32[/")        #RANDOM STRING SENT TO SIGNIFY BLANK MESSAGE TO AVOID ISSUES WITH EMPTY PACKETS
                        self.data = self.conn.recv(1024)
                        if self.data and self.data.split("/")[0] != "@]32[":    #ENSURES MESSAGE ISNT 'BLANK'
                                __init__.messages.append([(str(self.addr[0]), time.strftime("%H:%M:%S", time.gmtime())),__init__.encrypt(self.data)])
            except socket.error:        #ACCEPT ERROR FOR BROKEN PIPE
                  init  .messages.append([(time.strftime("%H:%M:%S", time.gmtime())),"[DEBUG] Server lost connection"])
```

FINAL TESTING OF SOLUTION

IP entry textbox:

Entry Data	Type of data (Extreme/Exceptional/Normal)	Expected result	Result
“192.168.1.5”	Normal	Accepted however it may not connect if server isn’t running	“14:29:30 [DEBUG] Socket error, cannot connect” This is as there is no server operating on this ip address
“yellow”	Exceptional	Not accepted	“14:31:34 [DEBUG] Invalid form for IP and Port, must be in the form XXX.XXX.XXX.XXX XXXX where X is Integer”
“192.168.1.1”	Extreme	Accepted however it may not connect if server isn’t running	“14:33:36 [DEBUG] Socket error, cannot connect”
10	Exceptional	Not accepted	“14:34:38 [DEBUG] Invalid form for IP and Port, must be in the form XXX.XXX.XXX.XXX XXXX where X is Integer”
[space]	Exceptional	Not accepted	“14:36:16 [DEBUG] Invalid form for IP and Port, must be in the form XXX.XXX.XXX.XXX XXXX where X is Integer”
“192.168.1.255”	Extreme	Accepted however will not connect if server isn’t running	“14:37:20 [DEBUG] Invalid form for IP and Port, must be in the form XXX.XXX.XXX.XXX XXXX where X is Integer”

Testing done with the ip entry textbox was done with the accepted port of 6969

Port entry textbox:

Entry Data	Type of data (Extreme/Exceptional/Normal)	Expected Result	Result
6969	Normal	Accepted and connected to server	“14:44:50 [DEBUG] Connected to 127.0.1.1”
“brown”	Exceptional	Not accepted	“14:42:30 [DEBUG] Invalid form for IP and Port, must be in the form XXX.XXX.XXX.XXX XXXX where X is Integer”
“6666”	Exceptional	Accepted but no connection will be made	“14:46:04 [DEBUG] Socket error, cannot connect”
[space]	Exceptional	Not accepted	“14:47:13 [DEBUG] Invalid form for IP and Port, must be in the form XXX.XXX.XXX.XXX XXXX where X is Integer”

Testing done of port will be done with ip address 127.0.1.1 with a server running on 127.0.1.1:6969

Message entry textbox:

Entry Data	Type of data (Extreme/Exceptional/Normal)	Expected result	Result
“Hello”	Normal	Accepted and sent	“('127.0.1.1', '14:52:38') Hello” Sent to server
“4565463626”	Normal	Accepted and sent	“('127.0.1.1', '14:53:57') 4565463626” Sent to server
905432gfdijhgfdskjl	Extreme	Accepted and sent	“('127.0.1.1', '14:55:18') 905432gfdijhgfdskjl” Sent to server
[space]	Extreme	Accepted and sent	“('127.0.1.1', '14:56:49') “ Sent to server

This testing is done on an instance of the program in client mode while connected to a server.

My program features the following:

- Interfacing with stored data by exporting the chat log after each conversation

```
chatlog = open("CHATLOG "+time.strftime("%d-%b-%Y %H.%M.%S", time.gmtime())+".txt", "w")
```

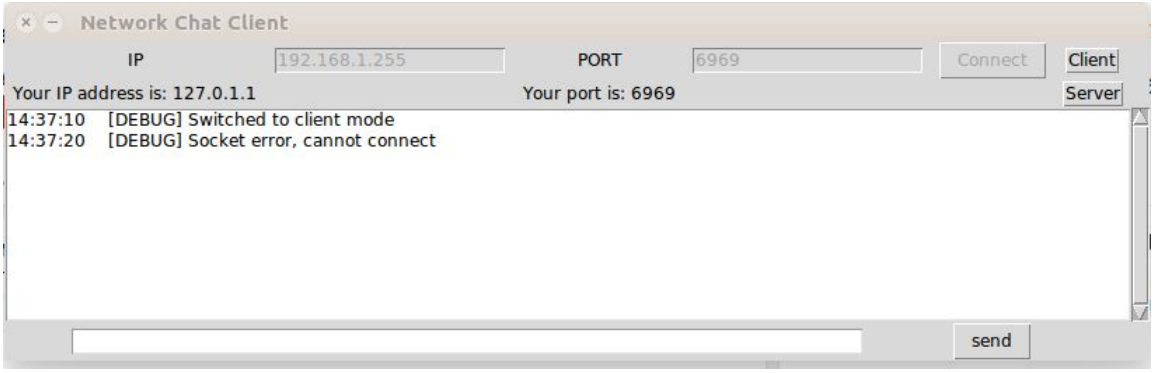
- 2-D arrays for storing a timestamp and all the messages from my program

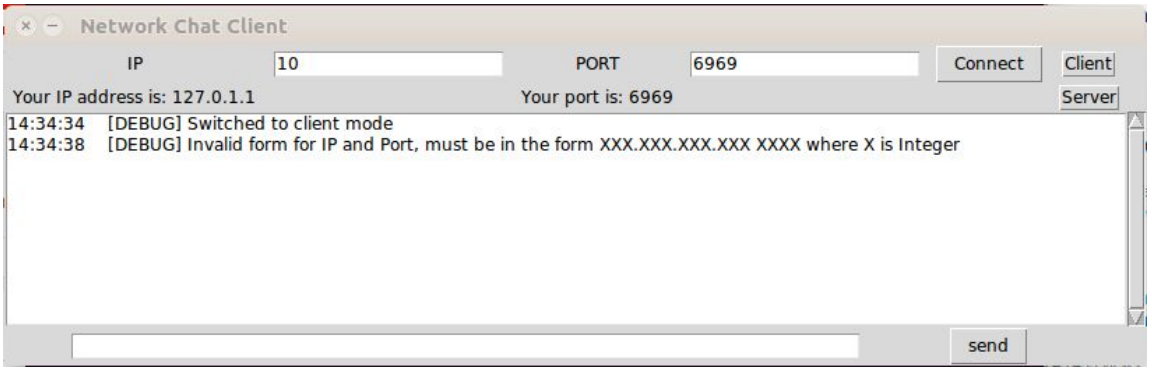
```
x.messageLog.insert(END, str(__init__.messages[f][0])+"    "+__init__.messages[f][1])
```

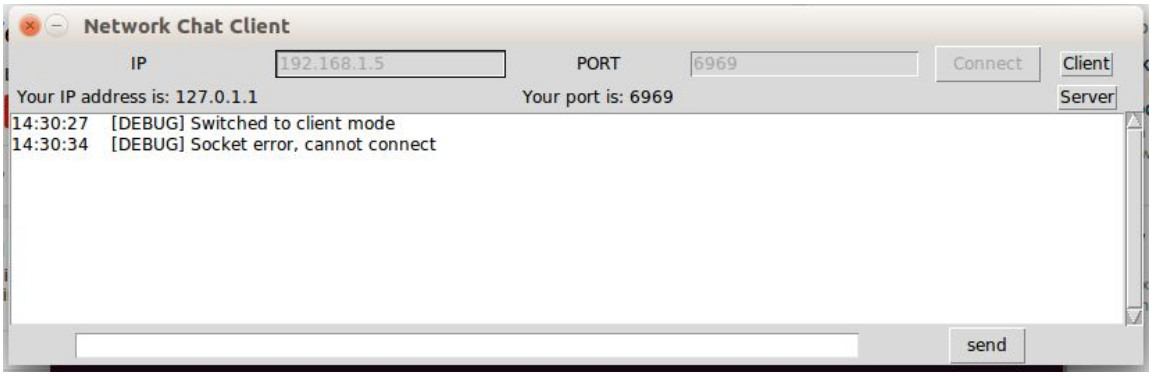
- Recursion to ensure that the user interface stays updated

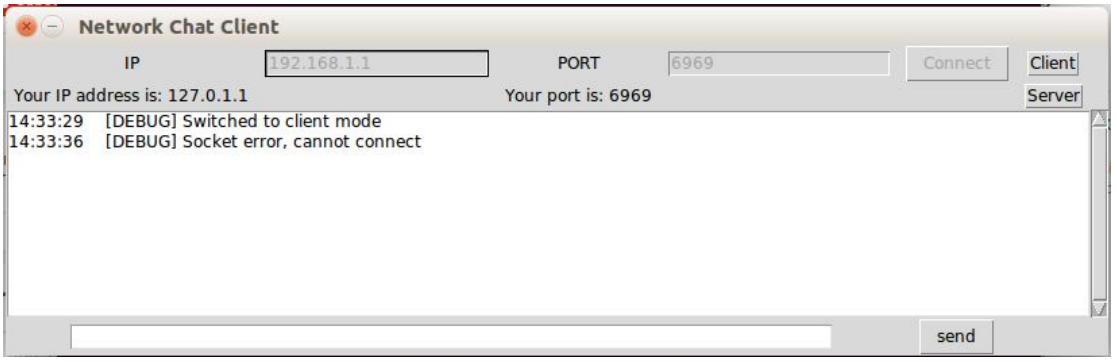
```
def interactions(x):            #DATA DISPLAY THREAD    USES RECURRSION

        if __init__.mode == None:                       #UPDATES STATUS OF MODE BUTTONS
                x.connect["state"] = DISABLED
                x.ipEntry["state"] = DISABLED
                x.portEntry["state"] = DISABLED
                x.clientButton.deselect()
                x.serverButton.deselect()
        elif __init__.mode == "CLIENT":
                x.connect["state"] = NORMAL
                x.ipEntry["state"] = NORMAL
                x.portEntry["state"] = NORMAL
        elif __init__.mode == "SERVER":
                x.connect["state"] = DISABLED
                x.ipEntry["state"] = DISABLED
                x.portEntry["state"] = DISABLED
        else:
                x.connect["state"] = DISABLED
                x.ipEntry["state"] = DISABLED
                x.portEntry["state"] = DISABLED

        if __init__.connected:                          #CHANGES VALUE OF CONNECT BUTTON WHEN IT IS CONNE
                x.connect["text"] = "Disconnect"
        else:
                x.connect["text"] = "Connect"

        for f in range(x.messageLog.size(), len(__init__.messages)):            #UPDATES USER INTERFACE
                x.messageLog.insert(END, str(__init__.messages[f][0])+"    "+__init__.messages[f][1])
                __init__.chatlog.write(str(__init__.messages[f][0])+"    "+__init__.messages[f][1]+"\n")

        time.sleep(0.5) #TIME DELAY SO FEWER PROCESSES ARE USED WHEN NOT REQUIRED


        if s.isAlive():         #IF USER INTERFACE THREAD STILL ALIVE
                interactions(x)         #RECURRSION TO MEET REQUIREMENTS                !!!!!!!!!!RECURSI
        else:
                __init__.chatlog.close()        #OTHERWISE CLOSE CHATLOG FILE AS NO CHANGES WILL BE MADE
```

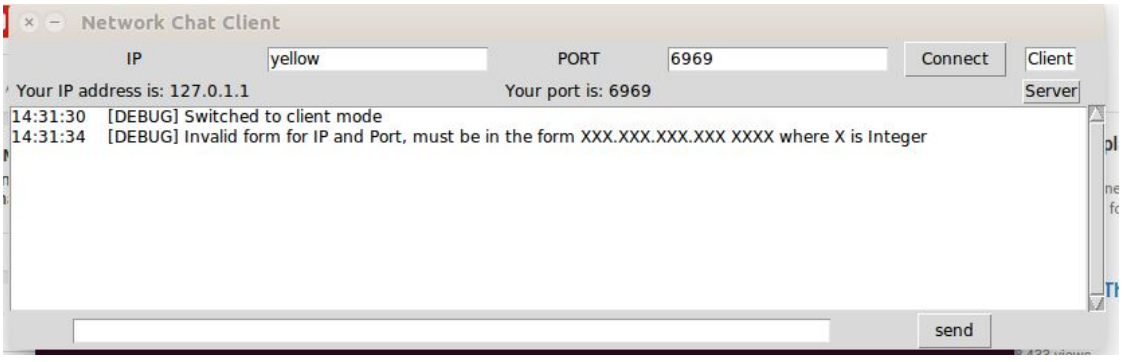
IP TEXTBOX TESTING EVIDENCE



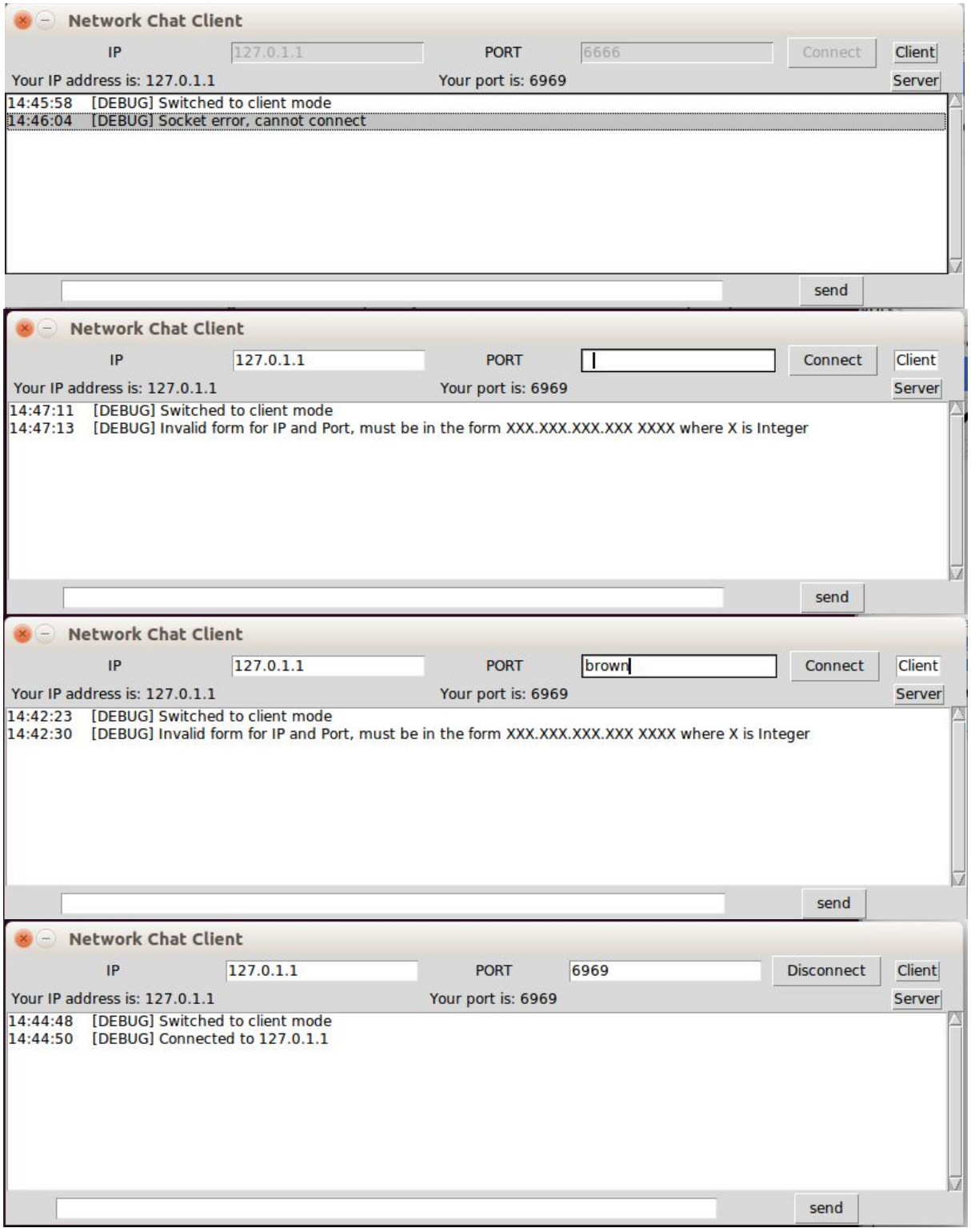




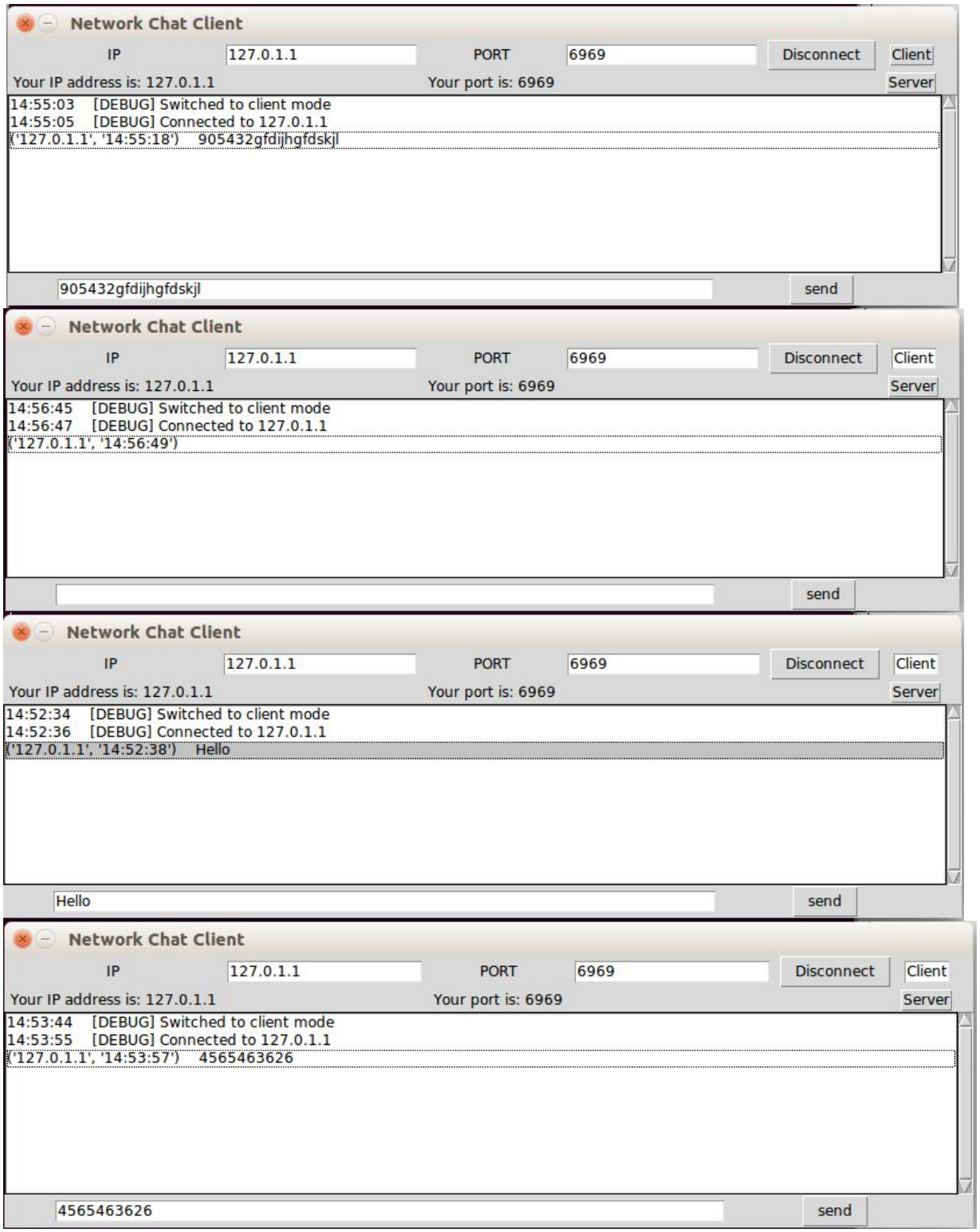




PORT TEXTBOX TESTING EVIDENCE



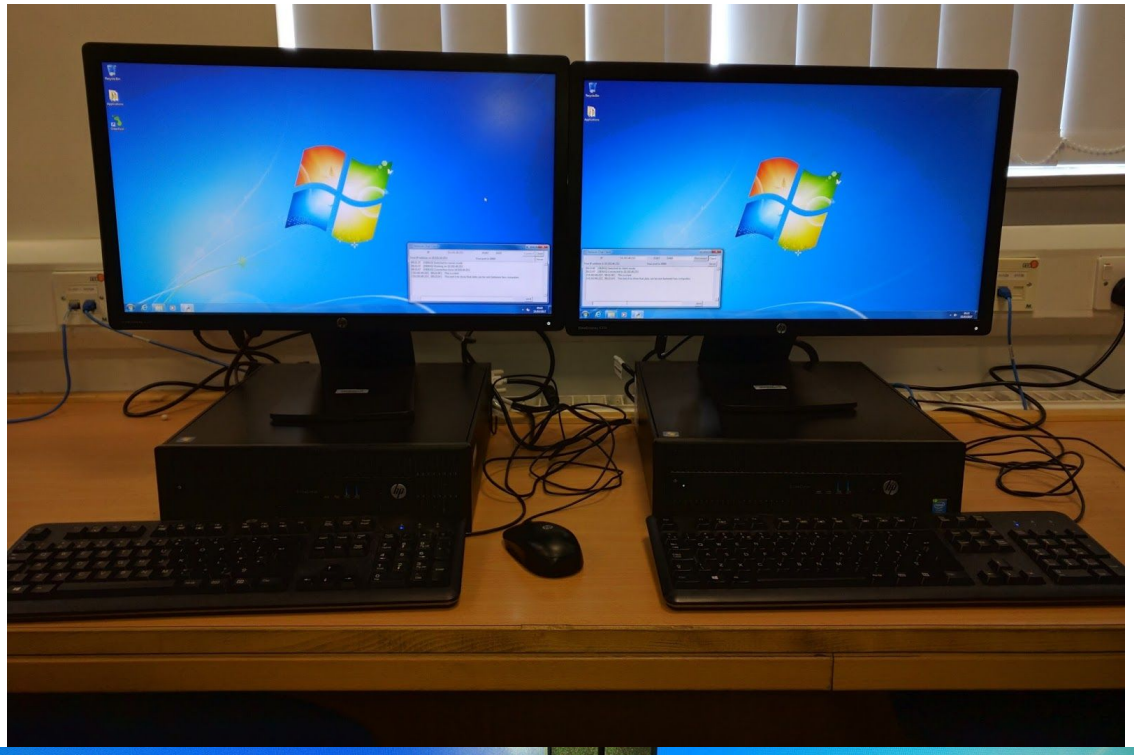
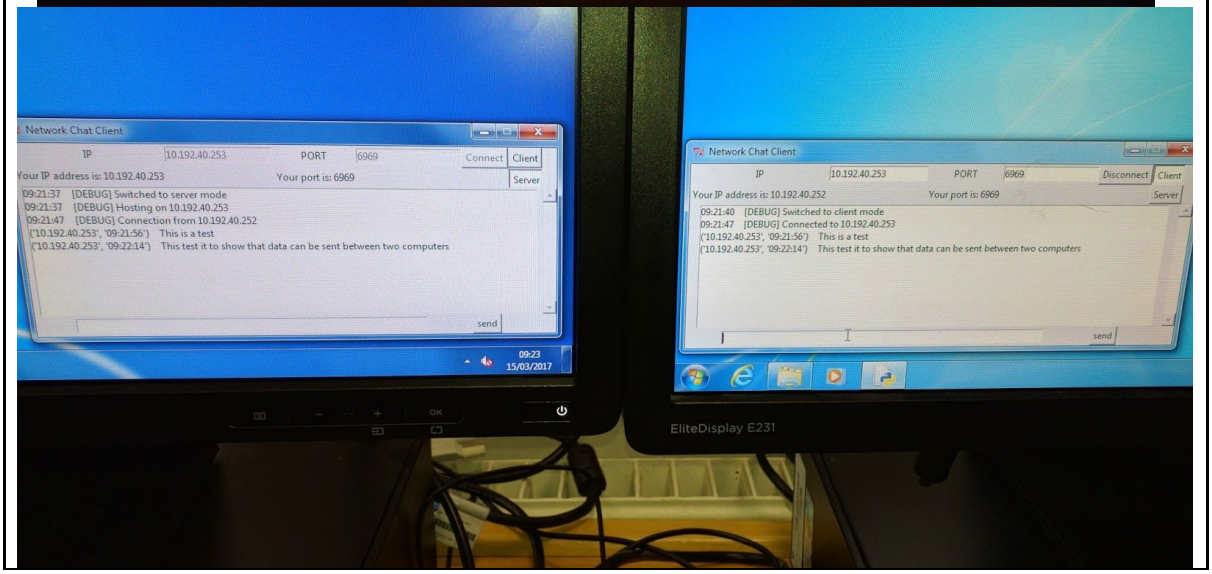
MESSAGE TEXTBOX TESTING EVIDENCE:



Functional Requirements:

The other testing should be done with my requirements:

- Send strings of text between two computers on a network
- Encrypt and decrypt a string using a suitable algorithm
- Export chat logs after every connection
- Have a clear, easy to use interface
- Prevent intercepted communications being read

“Send strings of text between two computers on a network”
To show that my solution can send data between two computers, the solution was installed on two computers on a local network.  

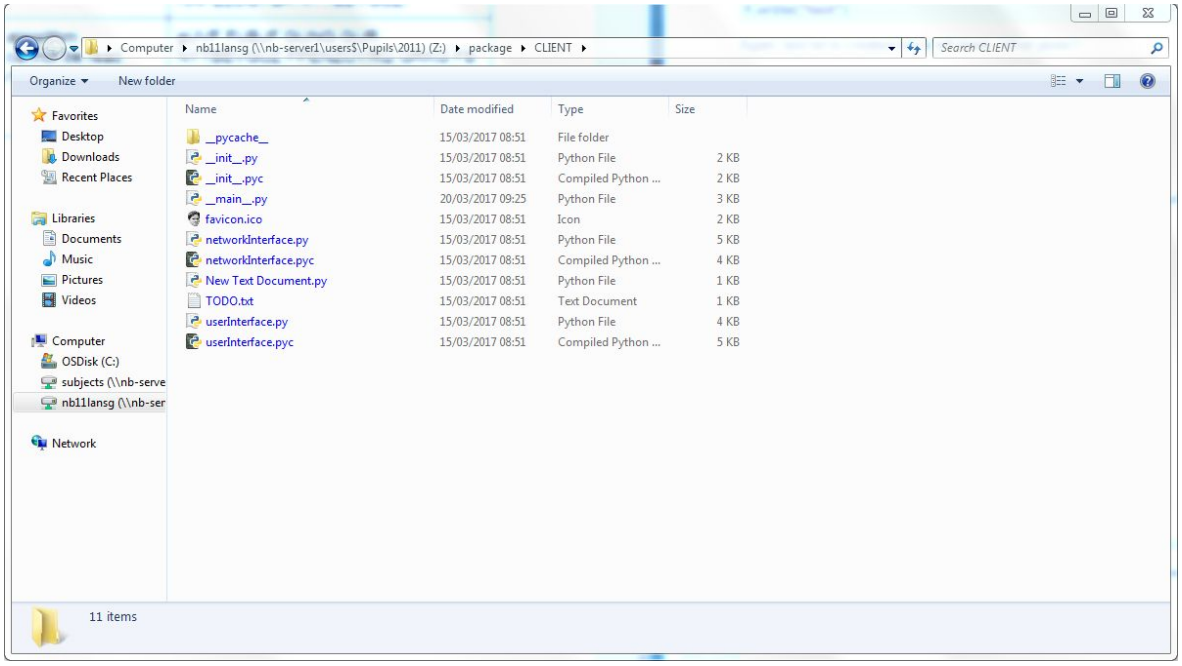
These photos show that the solution can communicate data between two computers on a local network.

“Encrypt and decrypt a string using a suitable algorithm”

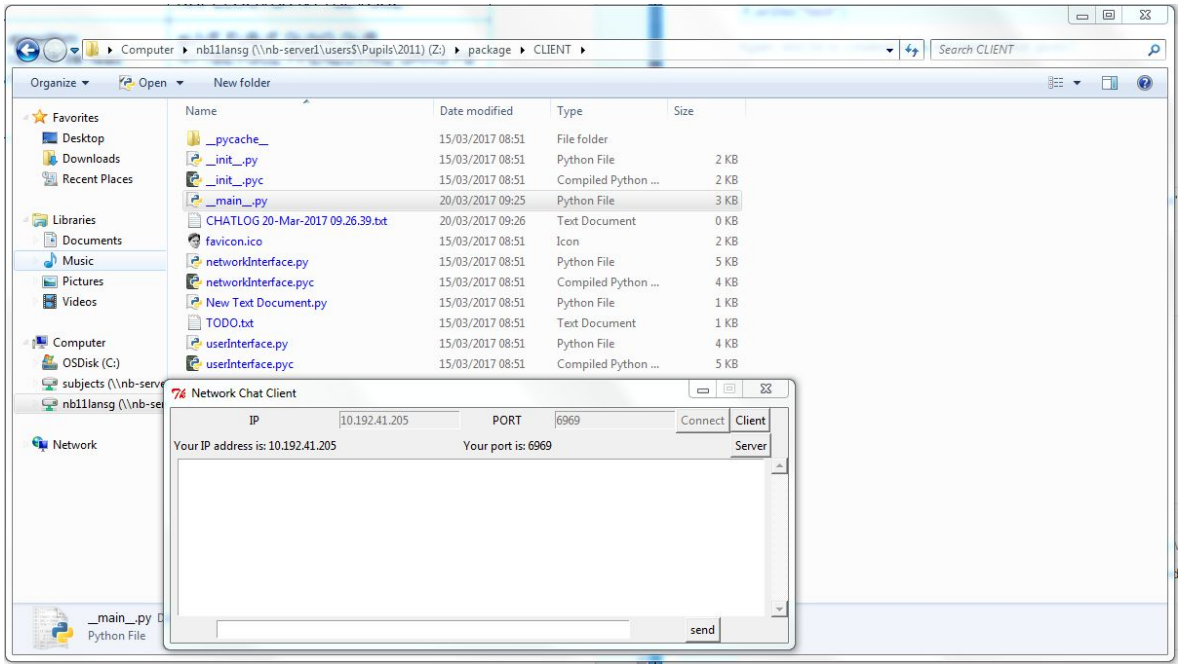
My solution uses a modified version of the ROT13 algorithm. Although this is an incredibly insecure algorithm by modern standards. It effectively scrambles the data and is fairly quick to process the messages. Additionally, it can also be used as its own inverse, this means that the same algorithm can be used to encrypt and decrypt the messages. These features make it suitable for the solution.

Input into encryption algorithm	**Output from encryption algorithm**
This is a test of the encryption algorithm.	gUVF VF N GRFG BS GUR RAPELCGVBA NYTBEVGUZ.
This shows that the algorithm scrambled text so it cannot be read directly. This makes the messages secure.	gUVF FUBJF GUNG GUR NYTBEVGUZ FPENZOYRQ GRKG FB VG PNAABG OR ERNQ QVERPGYL. gUVF ZNXRF GUR ZRFFNTRF FRPHER.

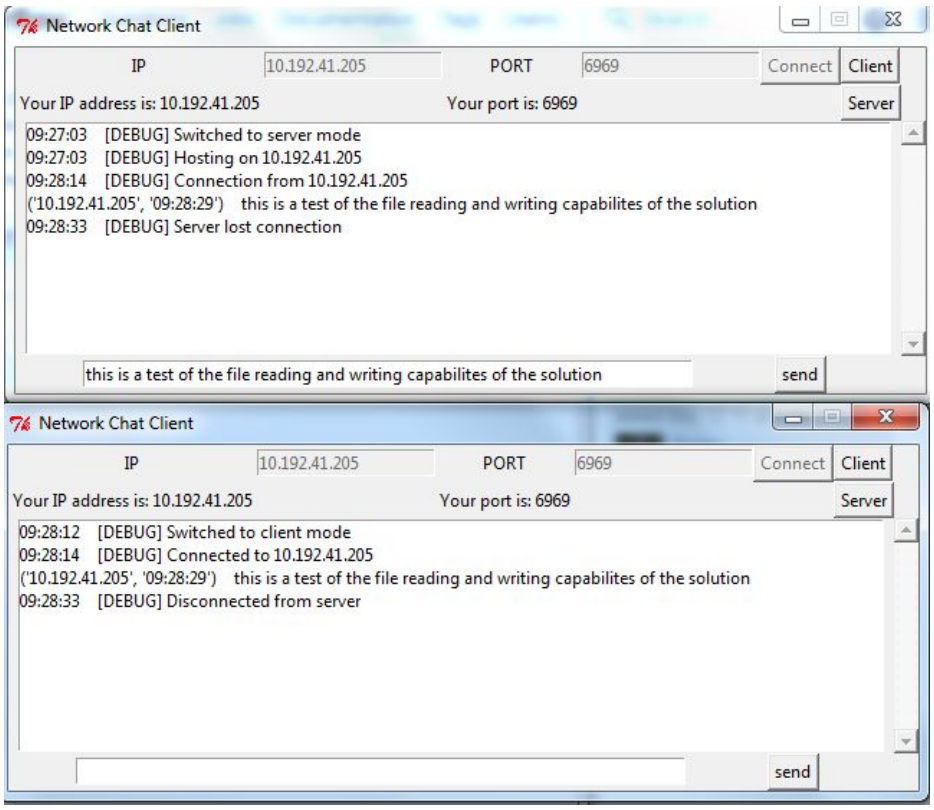
“Export chat logs after every connection”
The next requirement of my program is to create a file and dump the contents of the chat log into it after the program is closed.



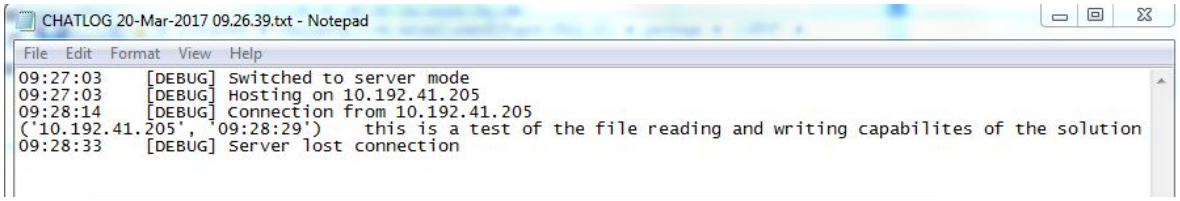
This screenshot shows the parent folder of the main file. This is prior to the file being created.



This shows that when an instance of the program is created, a file with the current date and time is created.

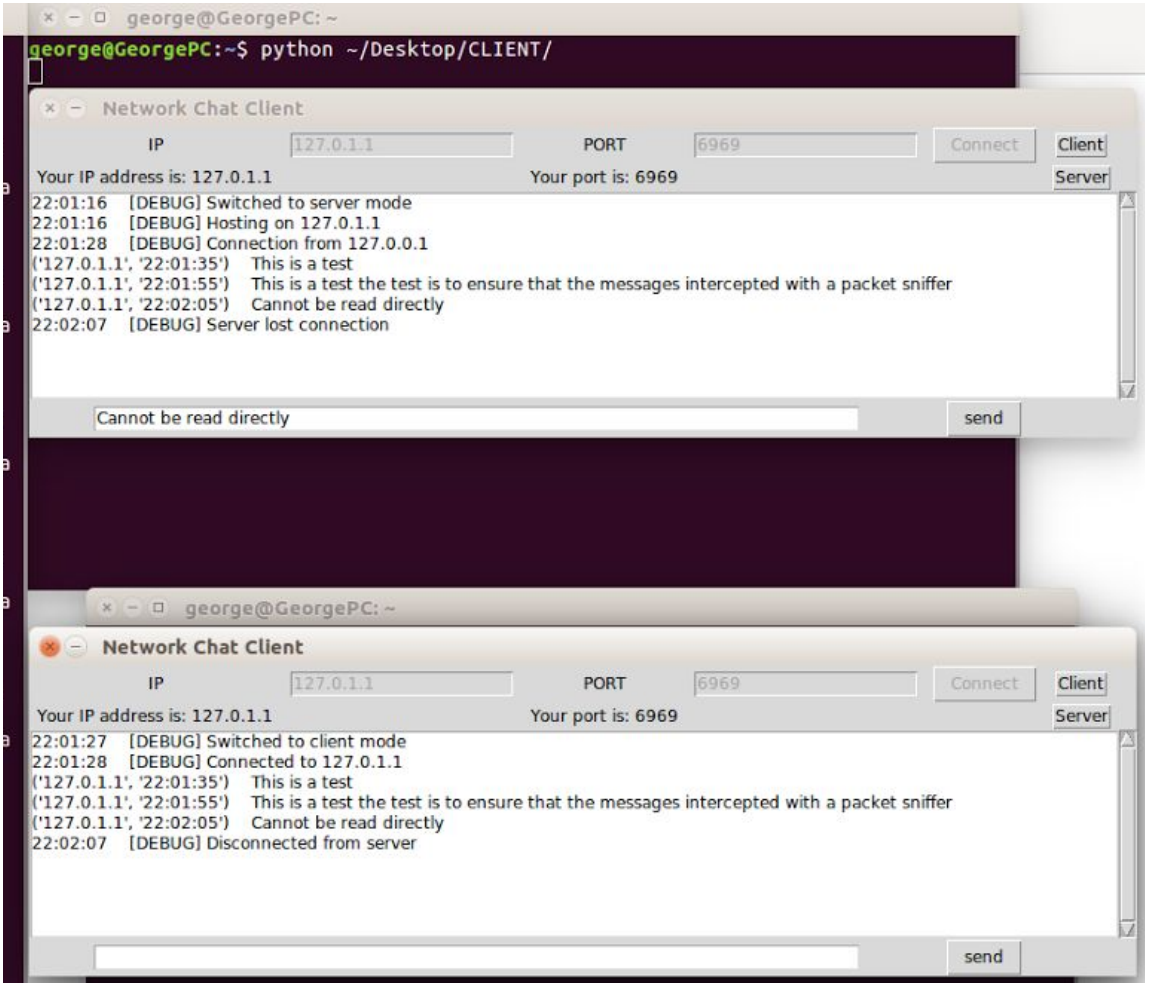


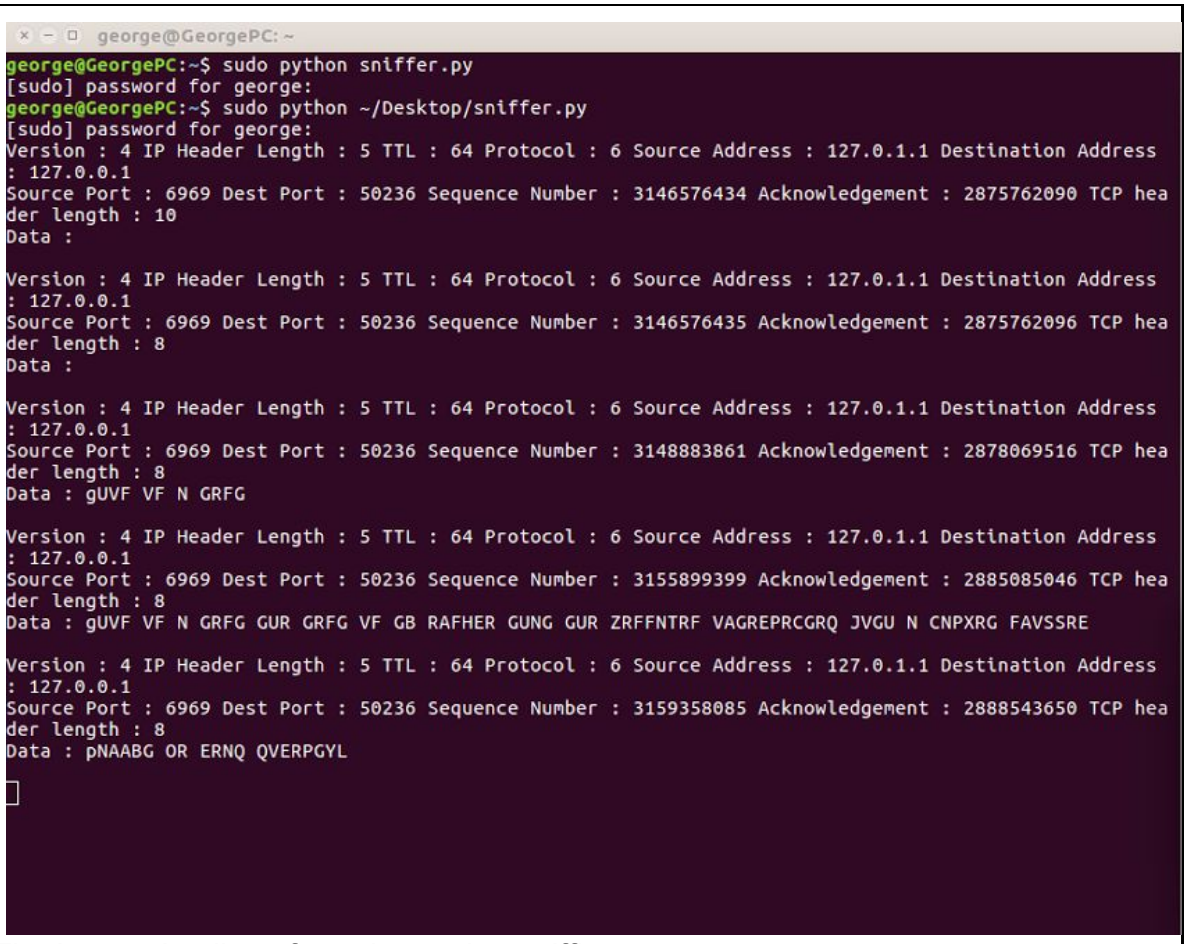
This is an example chat between two instances of the program. The instance at the top of the screenshot is the server instance which i will use to demonstrate the file creation of the program.



This is the file created after the program is closed. Notice how it's contents match that of the program in the screenshot above.
This proves that the program successfully creates a file and exports the chat log when the program is closed.

“The interface is clear and easy to use”
This is a fairly subjective requirement so i decided to use a user survey. I asked other advanced computer science students who would generally know more than the average person, however they would have a knowledge more similar to the end users detailed earlier in the report. Seb: “It’s fairly well laid out, and simple to understand.” Ryan: “Looks understandable, the send button could be more in line with the server, client buttons. The text box at the bottom could be across the width of the window. Generally could align things better.” These generally suggest that the user interface is clear, but could use some better layout. This partially completes the requirement.

“Prevent intercepted communications being read”
To demonstrate this, i will use a packet sniffer to intercept packets from communicating machines. This is an attack that someone wanting to steal data would likely use. Packet sniffing is an illegal activity on any network you don’t own. I used my completely private network and filtered the packets so that only the ones sent by a copy of my program appear on the terminal window. This is an example conversation had between two instances of my program.    Below is the output from the packet sniffer which was running in the background during the communication between the instances of my program. This emulates what an attacker would use to steal data from people.



The interesting lines from the packet sniffing are:

“Data : gUVF VF N GRFG”
“Data : gUVF VF N GRFG GUR GRFG VF GB RAFHER GUNG GUR ZRFFNTRF VAGREPRCGRQ JVGU N CNPXRG FAVSSRE”
“Data : pNAABG OR ERNQ QVERPGYL”

This data is impossible to read directly. However when decrypted with my algorithm:

“This is a test”
“This is a test the test is to ensure that the messages intercepted with a packet sniffer”
“Cannot be read directly”

These decrypted messages match the ones transmitted during the communication. This shows that although the messages were intercepted, they couldn’t be read by anyone without knowledge of the encryption algorithm.
I believe this satisfies “Prevent intercepted communications being read”

End user testing:
To test this, I provided other students with a copy of my solution and asked them to try and connect to a copy running in server mode on my machine.

Given the ip address of my machine, one student managed to connect to the server i had set up and send messages easily. They said “It was easy to see how to connect to other things as a client.”

One student struggled initially and said that they didn’t understand the distinction between client and server. They said however that they likely aren’t a part of the intended user group, this is still useful feedback though because it shows that the interface may need work

EVALUATION

The solution closely matches the requirements set out earlier in the report as all of the requirements have been proven in the final testing section

- Send strings of text between two computers on a network
- Encrypt and decrypt a string using a suitable algorithm
- Export chat logs after every connection
- Have a clear, easy to use interface
    - This was debated in the user survey
- Prevent intercepted communications being read

What went right:

- Researching the encryption early meant that it was simple to plan and implement.
- Knowing about the function of the socket library meant that there were no issues with the network communication.

What went wrong:

- Massive issues arose with time management, this is something that needs to be improved. Overestimating and underestimating the time requirements for my project meant that i struggled to keep up with the deadlines.
- The interface design likely could've been improved to look more attractive and appealing. I believe given the time constraints, the interface was better to be a
- My previous interface design didn't have a connect button for the client mode of the program. This was clumsy as changing the ip and port to connect to had to be done before choosing modes, this wasn't very intuitive and gave an error if misunderstood. Adding a connect button allows for a more intuitive user interface, however it has the drawback of adding a large amount of code to disable parts of the interface when not in use.
- I was updating the contents of the message box by deleting all the contents and replacing with its previous contents plus any changes every second. This was inefficient and meant that scrolling the box would only be applied for 1 second before it was deleted and replaced. I fixed this by finding the changes since the last update and adding those on top of the rest of the entries in the box.
- Major issues with the function of the program arose when trying to interact with the user interface and the network interface at the same time, this was because the user interface and the network interface both have loops which cannot run at the same time. This was solved by introducing threading to my program.
- Issues arose with the chat log files not being exported properly. This was because the file wasn't being closed from the right thread. This was fixed by

doing more research into the file operations in Python and moving the close command for the file to a different place.

How I could've improved the development process:

- Doing more research into smaller facets of my project like file operations in Python
- Giving more time to the implementation to account for issues like the concurrency issue.
- Planning the interface of my project for longer and possibly implement a prototype to allow for issues like, not being able to scroll the message log
- Allowing for more time in the gantt chart would've allowed for problematic circumstances.

More final testing whilst the program was in different modes (client, server, connected, disconnected, etc) would've allowed for a more comprehensive guarantee that the program works. However, this would've taken much more time to perform and document and likely wouldn't have shown that much more.

The end user testing could've been performed with more subjects to gain a better idea of how well my program performs with a new user. However, the user who performed the end user testing was extremely representative of the end users specified earlier in the report.

Due to the modular nature of my program it would be possible - with much more time - to better implement some of the shortcomings of my program. Things like a stronger encryption algorithm, or the use of user names rather than ip addresses.

RECORD OF PROGRESS

02/12/16
I began to try and estimate the time required for various parts of my project. I produced the project plan with this information in.

05/12/16:
Today I started write up my project proposal which outlines the purpose of my program.

07/12/16:
I began producing a gantt chart to display a more detailed version of the information from the outlined project plan. This means breaking down the tasks further and placing them in order of completion with date estimates for completion.

09/12/16:
I began producing my requirement specification, the requirements of my program are outlined from project proposal.

12/12/16:
Began investigation for requirement specification. Mostly around Encryption using the book “Everyday Encryption”. I found that many modern encryption algorithms are very difficult to code from scratch. This means that I will likely use a slightly weaker encryption algorithm for the ease of implementation.

15/12/16:
Documented findings from last time.
More encryption investigation lead me to encryption algorithms which are bidirectional, meaning they can be used to both encrypt and decrypt. I would like to use an algorithm like this as it saves space and implementation time. The simplest algorithm i have found of this type is the ROT13 algorithm. I believe i will use a modified version of this.

16/12/16:
Investigation into sockets and Tkinter was done today. This was quick as i am familiar with both libraries already. The research into sockets was done to ensure that there is no easy way to implement network communications with more than two parties.

Slightly missed deadline for finishing Requirement specification.

21/12/16:
Created interface design based upon user survey from project plan. Trying to make the interface clear and clean so that user can understand it more intuitively.

Overestimated time requirements for this, so project is slightly ahead of deadline.

23/12/16:
Using the interface design from earlier, I started to create a test plan using the input elements.
I've tried to test normal, extreme, and exceptional data where applicable for each interface element that allows entry.

Since only just back from christmas holidays slightly behind with schedule again this again was a result of poor time management.

8/1/16:
Finished the test plan for the interface. If there is time later, this may need refinement as it is fairly limited.

Began the pseudocode for my program. I have started with the layout and design of the files. I plan to use 4 files for my program.

1. __main__, this is the 'executable' file if you like. This is where the code goes that pieces together the rest of the program
2. __init__, this is where the variables that need to be accessed by all the other sections of the program will be stored along with the 'all encompassing' functions, like input validation functions which take something like a string and return true or false for it's validity.
3. networkInterface, this is the section of the program which deals with the connections between the computers. It will use Socket and should be object oriented to allow for a more modular design. This should help with testing and upgrading sections.
4. userInterface, this will be the program's user interface section. It will use Tkinter to create an interface object much in the same way as the object in networkInterface

I began writing the logic for the networkInterface.

Really far behind the gantt chart right now, underestimated the time requirements for writing pseudocode massively.

12/1/2017:
Continued with design from last time. Finished the majority of the networkInterface object.

The userInterface file is difficult to write pseudocode for as my knowledge of Tkinter is limited. I have written out the pseudocode for the object and will have to research more about the structure of Tkinter programs.

13/1/2017
Done more research into Tkinter and how it's laid out in a program. From the research it seems like a subroutine is usually used at the start of the program to initiate different user interface objects like buttons then they are organised using a method called grid that they all have.

I have updated some of the pseudocode in the userInterface object to reflect this research.

15/1/2017:
Today i finished the pseudocode for the networkInterface and began the __main__ file which instantiates a lot of objects at the moment. When the networkInterface and the userInterface are closer to being done, i will start the __init__ file which deals with communication between

the two.

20/1/2017:
Finished the userInterface pseudocode to the best I could as well as the __init__ file which pieces userInterface and networkInterface together. Hopefully the userInterface file looks like the interface design but this is something that will likely be changed during implementation.

23/1/2017:
Began implementation, wrote out most of userInterface pseudocode into python. This will take slightly longer than expected as my familiarity with Tkinter isn't very good.

When i manage to copy out the whole pseudocode i will be able to see if it matches the interface design.

24/1/2017:
Finished most of pseudocode for userInterface

26/1/2017:
Finished the pseudocode for userInterface file and created an instance of it. It doesn't look very much like the interface design. This arose because of the order of the project.
Tried to alter the code so that the userInterface instance looks more like the interface design.

27/1/2017:
Managed to make the userInterface look like the interface design. And began the networkInterface. These are the easiest parts to create because they can be tested independantly. The other sections are the difficult parts because they involve piecing those parts together.

3/2/2017:
The network interface is almost finished. This has been quick to implement as i am very familiar with the Socket library.

10/2/2017:
Finished the socket library, and __init__ file to store data. The implementation has been a lot faster than i thought it might.

15/2/2017:
Most of work on __main__ finished, this means that hopefully the implementation is almost done.

20/2/2017:
Large issues arose with the implementation now all sections are done. Trying to run the user interface whilst waiting for clients or trying to connect to a server leads to the interface freezing and occasionally crashing.

Might have been over ambitious to try a project of this scope. I might have to produce two versions of my program. A client version with a user interface and a separate server one without an interface. I don't like the idea of doing this however so will investigate other solutions.

21/2/2017:
After investigating my issue and possible solutions. I have discovered the threading library of python. It allows different subroutines to run concurrently at the same time.

I will need to do more experimentation with the module to investigate how it works.

I have updated parts of my project proposal, gantt chart, and feasibility study to reflect the changed circumstances. My implementation will require more time as i need to add threading to my program.

24/2/2017:
Today i experimented with the threading module of python. Initially i had some issues with actually getting things to run concurrently.
This will likely be difficult to implement with the rest of my program however.

27/2/2017:
I've decided i may need a third thread to communicate data between the two files. This is because the userInterface's message log needs to be updated whenever the networkInterface receives new data.

2/3/2017:
Implemented the third thread, however some issues have arisen with the threads ability with changing the value of variables. There are methods in the threading library used to lock the value of variables so there aren't issues with deadlocking. The variables may be locked by default.

3/3/2017:
Attempted to issue locking and releasing of the variables however the variable's values still don't change.

7/3/2017:
Solved the issue with the variables not changing. Due to the object oriented nature of the program, the threads couldn't access the variables needing to be changed because of the scope of the variables.
To fix this issue, I have moved the third thread definition to inside the userInterface thread and used the .after() function which i have used to start the function.

10/3/2017:
The program largely work correctly, although there are some slight issues with scrolling the message box. This is because the method of updating the messagebox involves deleting all the contents of the message box and replacing it every half a second. To fix this, i've instead found the things that haven't been changed every half a second and instead added that to the message box. I also plan to add a scrollbar and a connect button for the client mode of the program.
I solved the issue with the updating message box.

13/3/2017:
Added a scrollbar and connect button to make the interface look more like the updated interface design.

Started final testing.

16/3/2017:
Finished test plan for testing all inputs. Gathered all evidence of testing also.

17/3/2017:
Started testing functional requirements. Some are tricky to test, for example the “Prevent intercepted communications being read” this may involve the use of a packet sniffer to intercept communication.

20/3/2017:
Found an open source packet sniffer to test one of the functional requirements. Finished testing all the functional requirements.

21/03/2017:
Started evaluation, and piecing together different versions of report.

22/03/2017:
Continued writing up evaluation. Added the initial versions of the some sections of my program.

23/03/2017:
Finished evaluation just tidying sections of report and collating dates for progress diary.