UMass Dartmouth ECE 369: Computer Networks

Network Programming Framework Project

We certify that this work is original
and not a product of anyone's work but our own

X______________ X______________ X______________

Cameron Whittle James McCarthy Peter McGrory

Experiment: March - April, 2019
Submitted: April 24, 2019

Graded by:______________________ Date:______________________

Contents

List of Figures

Listings

Abstract

Socket programming is a core functionality within phones, computers, and networks. By testing different computer networking frameworks and learning the difference between protocols, teams can integrate and design their own application to feature a core functionality of socket programming. **Amity Communications**, a simple yet effective group chat system, allows users to organize their daily communication within a nice GUI package. The project taught teams the basics of socket programming and was valuable in our careers as engineers.

1 Introduction

Computer Networks are a massive invaluable part of our everyday lives whether we know it or not. From something as small as messaging your friend on social media to the entirety of the internet, Networks are all around us. For computers to communicate back and forth, there are a few options regarding the established protocols available. TCP and UDP are common used protocols that can be implemented within Server/Client or Peer-to-peer frameworks.

This laboratory experiment features a series of tests that need to be reformed in Linux to learn the basic functionalities of a small computer network. Once the necessary testing is complete, teams must attempt writing code on their own that exhibits these core functionality, eventually leading up to a final open ended design project. Projects will test the limits of teams understanding while at the same time encouraging students to learn through hands on applications with a real world design to fix an identified problem.

2 Methods and Procedures

2.1 Materials

- PC with Virtual Machine(VM) capabilities and necessary compilers

2.2 Procedure

1. Experiment with client and server communication within a TCP and UDP based system in Python. Document findings on the ways a client and server behave within the given programs from the professor.

2. Experiment with a multithreaded TCP server in Python and document the differences between the base TCP server and a multithreaded one provided by the professor.

3. Experiment with a UDP client timer and document the features of a UDP system with a built in timer in Python, once again provided by the professor.

4. Write your own Peer2Peer framework, one with UDP and one with TCP in Python and test the programs.

5. Recreate the functionality of the UDP client timer program in a language of your choice. Teams may use sample codes in C, Java, or Perl as an example.

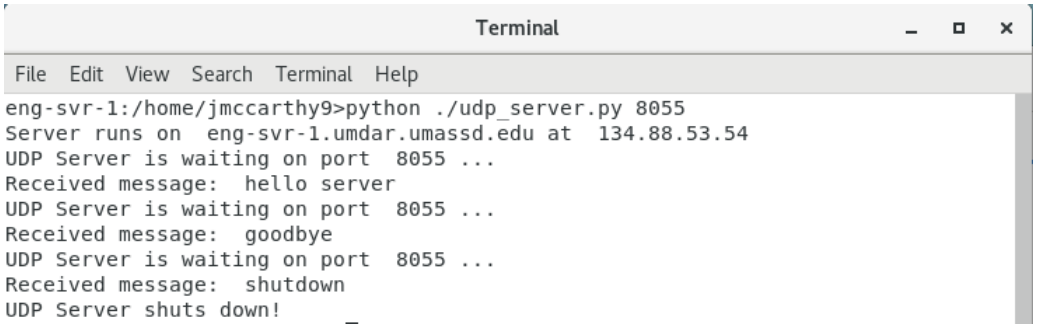
6. Design and implement your own Networking Application in any language. Feature a simple but effective graphical user interface and justify your teams choice on a client/server vs a Peer2Peer model.

7. Thoroughly test your PC application and prepare a class demo to showcase the systems functionality.

3 Experimental Results

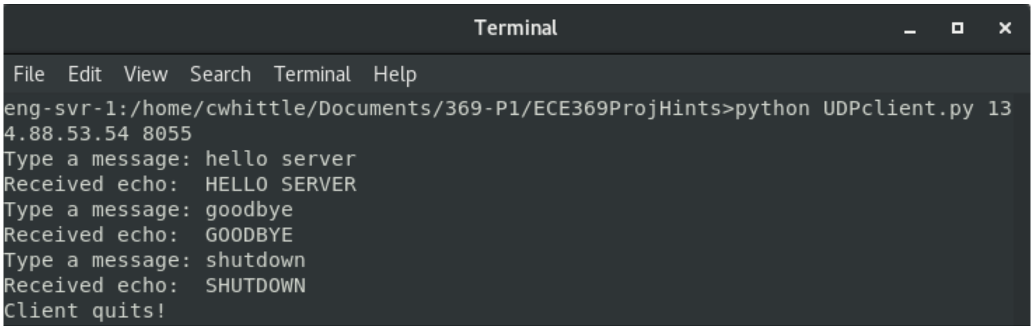
3.1 Experiment 1: Client/Server Framework

The first attempt into understanding computer networks in python was to run tests on a UDP and TCP server/client. Using code provided by the professor, a simple UDP program was tested. The program is designed for a Server user and a Client user to communicate using User Datagram Protocol. The first machine sets up a port to listen from and the second machine then can send data to the server. This data will be echoed back and then printed back out in the client's window. Lastly, the second machine can close it's port using the 'shutdown' command. A simple test showing this test being implemented is shown below in Figures 1 and 2.



```
eng-svr-1:/home/jmccarthy9>python ./udp_server.py 8055
Server runs on  eng-svr-1.umdar.umassd.edu at  134.88.53.54
UDP Server is waiting on port  8055 ...
Received message:  hello server
UDP Server is waiting on port  8055 ...
Received message:  goodbye
UDP Server is waiting on port  8055 ...
Received message:  shutdown
UDP Server shuts down!
```

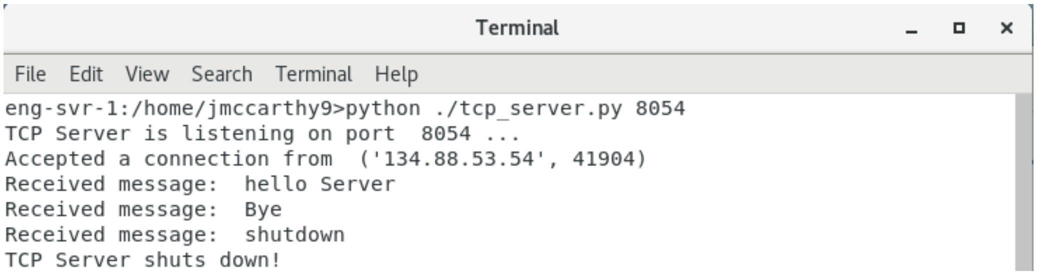
Figure 1: UDP Server Test Terminal



```
eng-svr-1:/home/cwhittle/Documents/369-P1/ECE369ProjHints>python UDPclient.py 13
4.88.53.54 8055
Type a message: hello server
Received echo:  HELLO SERVER
Type a message: goodbye
Received echo:  GOODBYE
Type a message: shutdown
Received echo:  SHUTDOWN
Client quits!
```

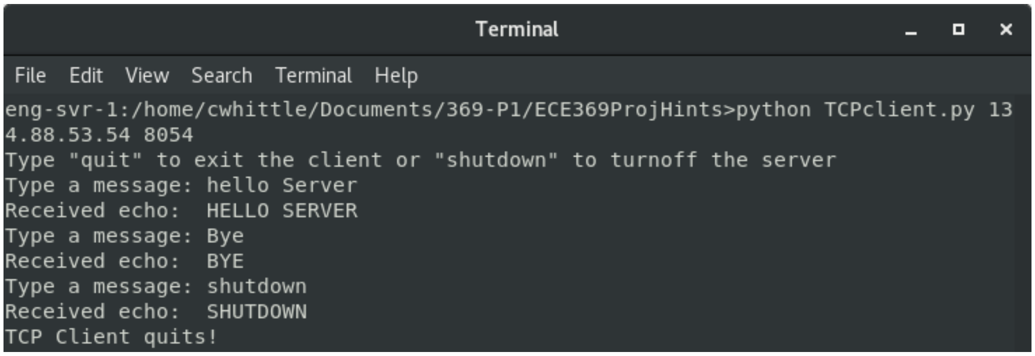
Figure 2: UDP Client Test Terminal

UDP is a connection-less protocol, as there is no solid connection between the two processes. Instead, the users send data back and forth using their port numbers and IP addresses to locate where to send the data to. A TCP connection on the other hand, or Transmission Control Protocol, creates a reliable connection between the two ports where data can be sent. In an identical fashion to the UDP testing, a TCP program was run in a similar fashion to test TCP connections. These tests are shown below in Figures 3 and 4. Between the two, UDP is slightly faster but TCP is more secure due to its ACK/NACK system and established connection.



```
eng-svr-1:/home/jmccarthy9>python ./tcp_server.py 8054
TCP Server is listening on port  8054 ...
Accepted a connection from  ('134.88.53.54', 41904)
Received message:  hello Server
Received message:  Bye
Received message:  shutdown
TCP Server shuts down!
```

Figure 3: TCP Server Test Terminal

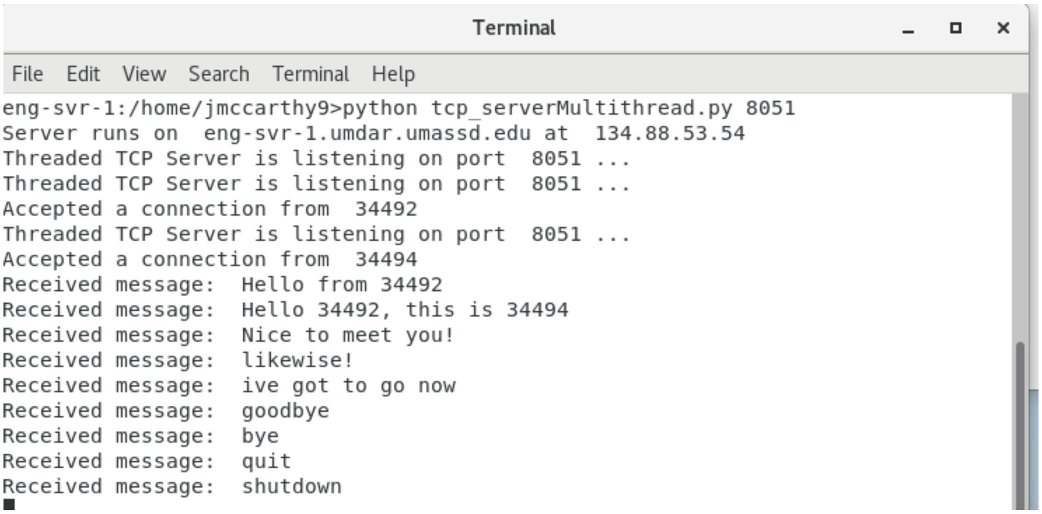


```
eng-svr-1:/home/cwhittle/Documents/369-P1/ECE369ProjHints>python TCPclient.py 13
4.88.53.54 8054
Type "quit" to exit the client or "shutdown" to turnoff the server
Type a message: hello Server
Received echo:  HELLO SERVER
Type a message: Bye
Received echo:  BYE
Type a message: shutdown
Received echo:  SHUTDOWN
TCP Client quits!
```

Figure 4: TCP Client Test Terminal

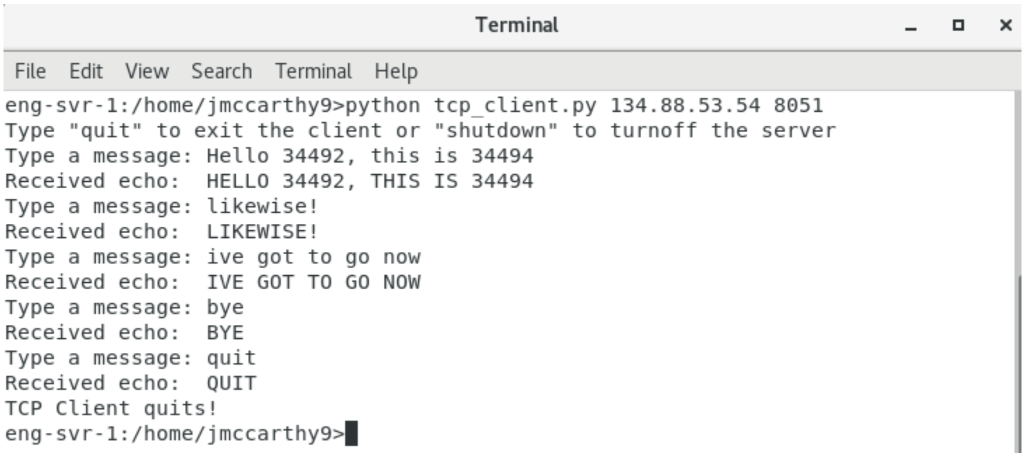
3.2 Experiment 2: Multithreaded Server

One of the biggest issues regarding a simple TCP server and client is that only one client can connect to the server at a time. That obviously is an issue when regarding a group chat scenario where you want more than 2 people to be able to communicate over the same connection. This is where TCP Multithreading comes into play, allowing multiple clients onto the same server simultaneously. Shown below in Figures 5, 6, and 7 is an example of the functionality between two clients and a single server. This is identical to the TCP system shown prior except it adds the utility of being able to have multiple clients echoing back over the same server, a key feature of a group chat system.



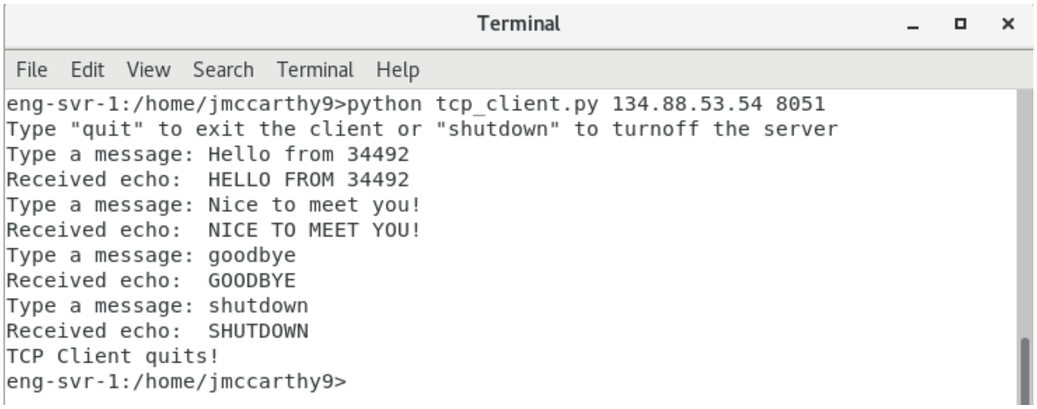
```
eng-svr-1:/home/jmccarthy9>python tcp_serverMultithread.py 8051
Server runs on  eng-svr-1.umdar.umassd.edu at  134.88.53.54
Threaded TCP Server is listening on port  8051 ...
Threaded TCP Server is listening on port  8051 ...
Accepted a connection from  34492
Threaded TCP Server is listening on port  8051 ...
Accepted a connection from  34494
Received message:  Hello from 34492
Received message:  Hello 34492, this is 34494
Received message:  Nice to meet you!
Received message:  likewise!
Received message:  ive got to go now
Received message:  goodbye
Received message:  bye
Received message:  quit
Received message:  shutdown
```

Figure 5: TCP Multithreaded Server Test Terminal



```
eng-svr-1:/home/jmccarthy9>python tcp_client.py 134.88.53.54 8051
Type "quit" to exit the client or "shutdown" to turnoff the server
Type a message: Hello 34492, this is 34494
Received echo:  HELLO 34492, THIS IS 34494
Type a message: likewise!
Received echo:  LIKEWISE!
Type a message: ive got to go now
Received echo:  IVE GOT TO GO NOW
Type a message: bye
Received echo:  BYE
Type a message: quit
Received echo:  QUIT
TCP Client quits!
eng-svr-1:/home/jmccarthy9>
```

Figure 6: TCP Multithreaded Client1 Test Terminal

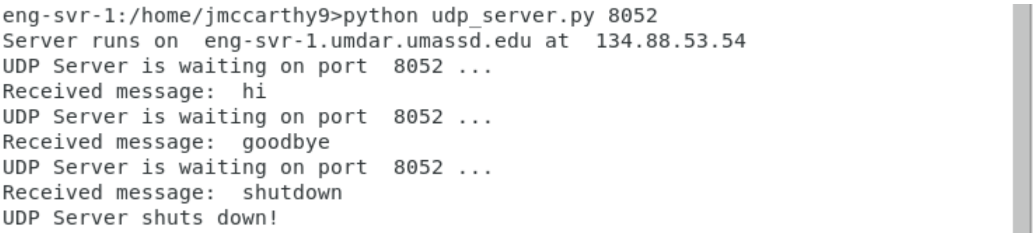


```
eng-svr-1:/home/jmccarthy9>python tcp_client.py 134.88.53.54 8051
Type "quit" to exit the client or "shutdown" to turnoff the server
Type a message: Hello from 34492
Received echo:  HELLO FROM 34492
Type a message: Nice to meet you!
Received echo:  NICE TO MEET YOU!
Type a message: goodbye
Received echo:  GOODBYE
Type a message: shutdown
Received echo:  SHUTDOWN
TCP Client quits!
eng-svr-1:/home/jmccarthy9>
```

Figure 7: TCP Multithreaded Client2 Test Terminal

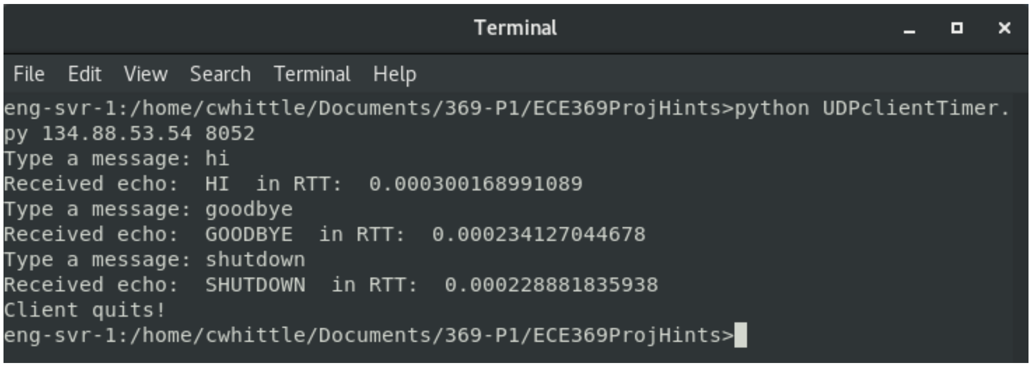
3.3 Experiment 3: Timer Client

With prior experiments, when a UDP message is sent, the client must wait indefinitely for the message to be echoed back, or an ACK from the server. In the case that the packet gets lost or the server isn't running, the client waits forever as the message will never come back. This is why a timer/timeout implementation is necessary. Shown below in Figures 7 and 8 are the successful implementation testing of this UDP timer implementation. The server echoes back the message successfully and records the Round Trip Time of the packet. This is the scenario that occurs if the packet is successfully received after it has been sent within the time window.



```
eng-svr-1:/home/jmccarthy9>python udp_server.py 8052
Server runs on  eng-svr-1.umdar.umassd.edu at  134.88.53.54
UDP Server is waiting on port  8052 ...
Received message:  hi
UDP Server is waiting on port  8052 ...
Received message:  goodbye
UDP Server is waiting on port  8052 ...
Received message:  shutdown
UDP Server shuts down!
```

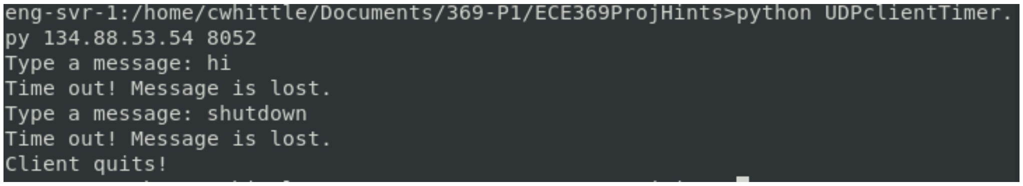
Figure 8: UDP Server With Timer Terminal [TestA]



```
eng-svr-1:/home/cwhittle/Documents/369-P1/ECE369ProjHints>python UDPclientTimer.
py 134.88.53.54 8052
Type a message: hi
Received echo:  HI  in RTT:  0.000300168991089
Type a message: goodbye
Received echo:  GOODBYE  in RTT:  0.000234127044678
Type a message: shutdown
Received echo:  SHUTDOWN  in RTT:  0.000228881835938
Client quits!
eng-svr-1:/home/cwhittle/Documents/369-P1/ECE369ProjHints>
```

Figure 9: UDP Client With Timer Terminal [TestA]

The other scenario is that the packet is lost, or in this case, the server is nit running. Showcased in Figure 9 is the client running independently of the server. In this scenario, the message is lost and instead of waiting indefinitely, the client receives a timeout message after a set time interval and then can continue to send another message and function properly.

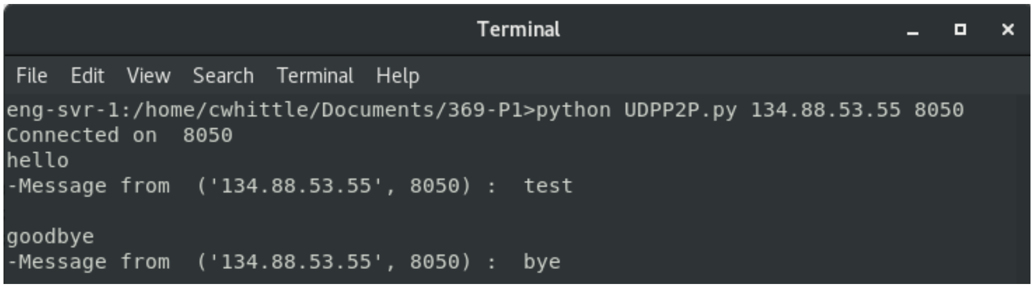


```
eng-svr-1:/home/cwhittle/Documents/369-P1/ECE369ProjHints>python UDPclientTimer.
py 134.88.53.54 8052
Type a message: hi
Time out! Message is lost.
Type a message: shutdown
Time out! Message is lost.
Client quits!
```

Figure 10: UDP Client With Timer Terminal [TestB]

3.4 Extension: Peer2Peer Framework

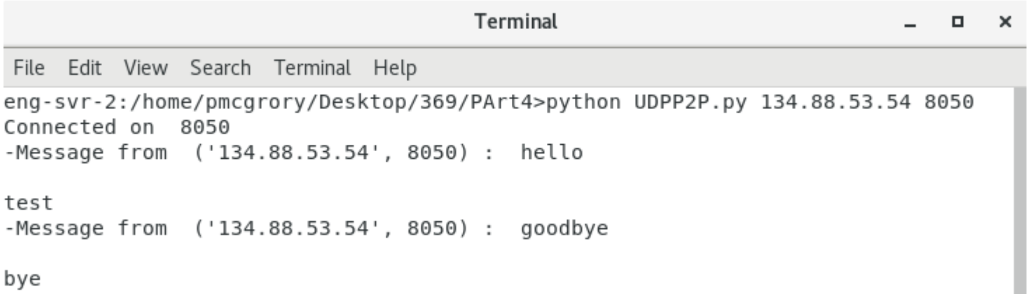
This part requested both a UDP and TCP peer to peer connection. Peer to peer connections are useful because it is easy to add new peers to them and because they make file sharing easy. This is because files can be shared simultaneously. This was done by taking the the client and server files used in experiment 1 and combining them into one single program. This program can essentially be run on two separate computers and allows for messages to be sent back and forth without the use of a server. In order to accomplish this using a TCP connection, multiple sockets were required. One socket handled the sending and receiving for one computer while a different socket handled the sending and receiving for the other computer. This functionality for the UDP system is showcased in Figures 11 and 12 below.



```
eng-svr-1:/home/cwhittle/Documents/369-P1>python UDPP2P.py 134.88.53.55 8050
Connected on  8050
hello
-Message from  ('134.88.53.55', 8050) :  test

goodbye
-Message from  ('134.88.53.55', 8050) :  bye
```

Figure 11: UDP Peer2Peer Framework Client 1 Terminal



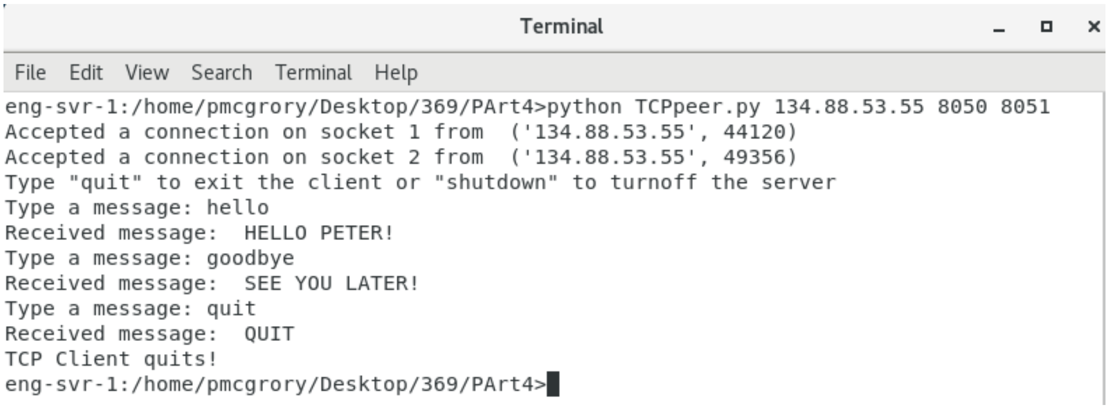
```
eng-svr-2:/home/pmcgrory/Desktop/369/PArt4>python UDPP2P.py 134.88.53.54 8050
Connected on  8050
-Message from  ('134.88.53.54', 8050) :  hello

test
-Message from  ('134.88.53.54', 8050) :  goodbye

bye
```

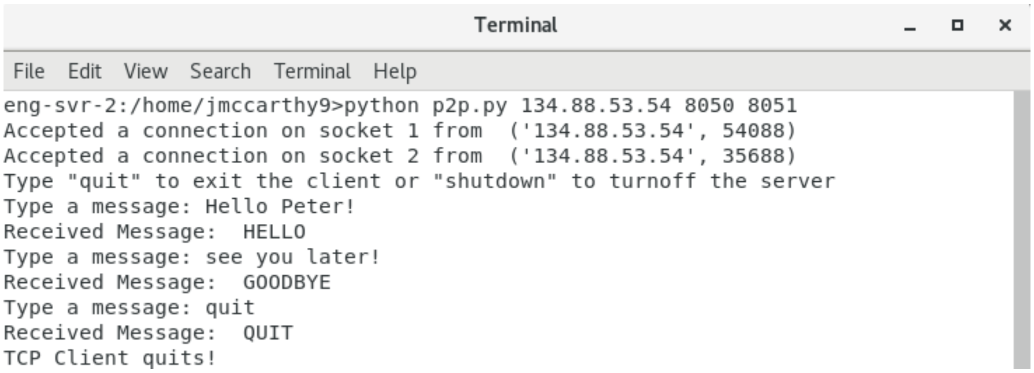
Figure 12: UDP Peer2Peer Framework Client 2 Terminal

To establish a peer-to-peer connection, we used two computers each running the same program connecting to each others respective IP's, as well as using two ports utilized for sending and receiving on either end. Once the program is ran, it waits until the partner program is detected and creates the connection. When this occurs, both users now have the ability to send messages to the other person using the program. Once a user types "quit" or "shutdown", the program cuts the connection on that users end, as seen in figures 13 and 14.



```
eng-svr-1:/home/pmcgrory/Desktop/369/PArt4>python TCPpeer.py 134.88.53.55 8050 8051
Accepted a connection on socket 1 from  ('134.88.53.55', 44120)
Accepted a connection on socket 2 from  ('134.88.53.55', 49356)
Type "quit" to exit the client or "shutdown" to turnoff the server
Type a message: hello
Received message:  HELLO PETER!
Type a message: goodbye
Received message:  SEE YOU LATER!
Type a message: quit
Received message:  QUIT
TCP Client quits!
eng-svr-1:/home/pmcgrory/Desktop/369/PArt4>
```

Figure 13: TCP Peer2Peer Framework Client 1 Terminal



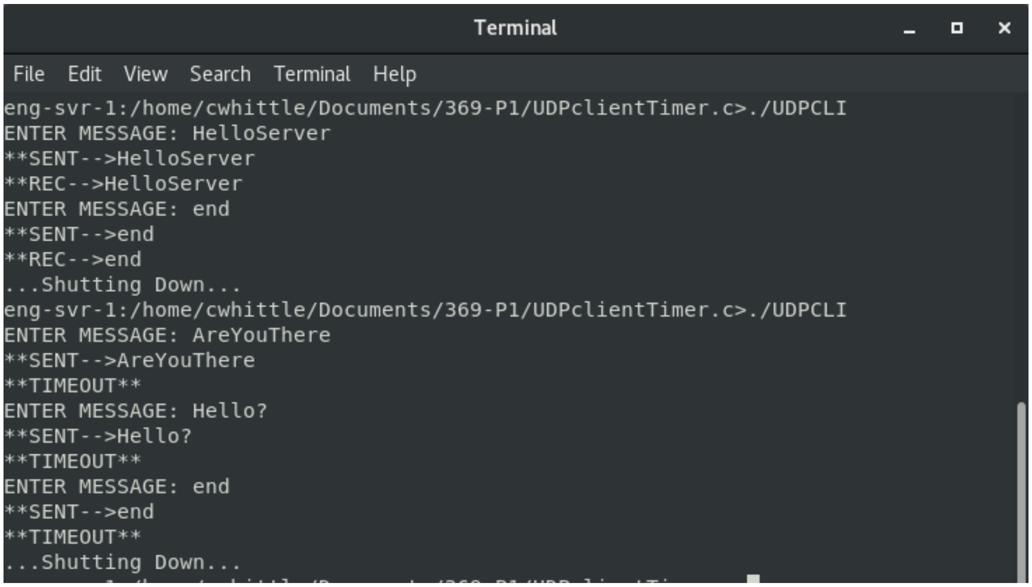
```
eng-svr-2:/home/jmccarthy9>python p2p.py 134.88.53.54 8050 8051
Accepted a connection on socket 1 from  ('134.88.53.54', 54088)
Accepted a connection on socket 2 from  ('134.88.53.54', 35688)
Type "quit" to exit the client or "shutdown" to turnoff the server
Type a message: Hello Peter!
Received Message:  HELLO
Type a message: see you later!
Received Message:  GOODBYE
Type a message: quit
Received Message:  QUIT
TCP Client quits!
```

Figure 14: TCP Peer2Peer Framework Client 2 Terminal

3.5 Integration: UDPclientTimer in Another Language

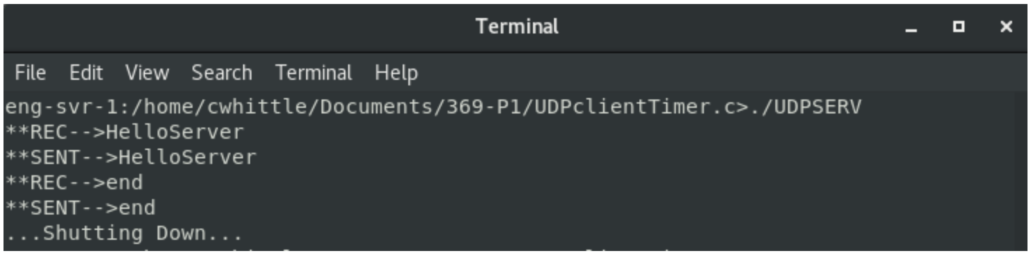
As discussed in section 3.3, the UDP timer implementation is crucial in preventing the system from not being able to handle packet loss and recovery of the systems utility. In this section, teams were asked to write their own UDP timer implementation is another language, not provided by the professor. The team decided on the language of C. The final code that was written can be shown in the corresponding Appendix entry under section 8.2 UDPclientTimer in Another Language Code where both the server and client codes are shown.

Figures 14 and 15 showcase this C code in action as it displays the full functionality of the timer implementation. As shown, when the message is echoed correctly the system functions perfectly but when the server is shutdown and the client continues to send data, the message times out and the user can send another message following this message. Since the messages are not going through, the client can then close itself so that it can try to send data elsewhere.



```
eng-svr-1:/home/cwhittle/Documents/369-P1/UDPclientTimer.c>./UDPCLI
ENTER MESSAGE: HelloServer
**SENT-->HelloServer
**REC-->HelloServer
ENTER MESSAGE: end
**SENT-->end
**REC-->end
...Shutting Down...
eng-svr-1:/home/cwhittle/Documents/369-P1/UDPclientTimer.c>./UDPCLI
ENTER MESSAGE: AreYouThere
**SENT-->AreYouThere
**TIMEOUT**
ENTER MESSAGE: Hello?
**SENT-->Hello?
**TIMEOUT**
ENTER MESSAGE: end
**SENT-->end
**TIMEOUT**
...Shutting Down...
```

Figure 15: UDP Client with Timer in C Terminal [TestA] and [TestB]



```
eng-svr-1:/home/cwhittle/Documents/369-P1/UDPclientTimer.c>./UDPSERV
**REC-->HelloServer
**SENT-->HelloServer
**REC-->end
**SENT-->end
...Shutting Down...
```

Figure 16: UDP Server in C Terminal [TestA] and [TestB]

3.6 Creation: Propose Your Own App in Any Language

For the open ended design, the team decided on attempting to fix an issue in the everyday lives of students. When in a group chat environment, chats can often become cluttered where critical information or questions can get lost among other responses or questions by other people. The proposed system, called **Amity Communications**, is a basic group text chat application that separates the common chat window into three distinct chat boxes. This allows the users to allocate these three boxes however they choose in order to organize the chats. An example of this systems utilization would be for the users to allocate one chat for school questions, one for work, and the other for leisure activities. Although the full control is in the hands of the users of the chat application.

Amity Communications was designed utilizing Visual Studio on Windows. The program was written in the C# Object Oriented language with the utilization of Visual Studio's "Windows

Forms" GUI designer. This was decided because the C# language was familiar to the team and the GUI creation aspect would allow the team to create a visually appealing and effective interface for users to work with. The framework chosen is a Server/Client system with focus on a TCP connection. This was chosen so that a user can launch a server in the background and act as a host where other users can connect to and send to. The GUI of the application on start-up is shown below in Figure 17.

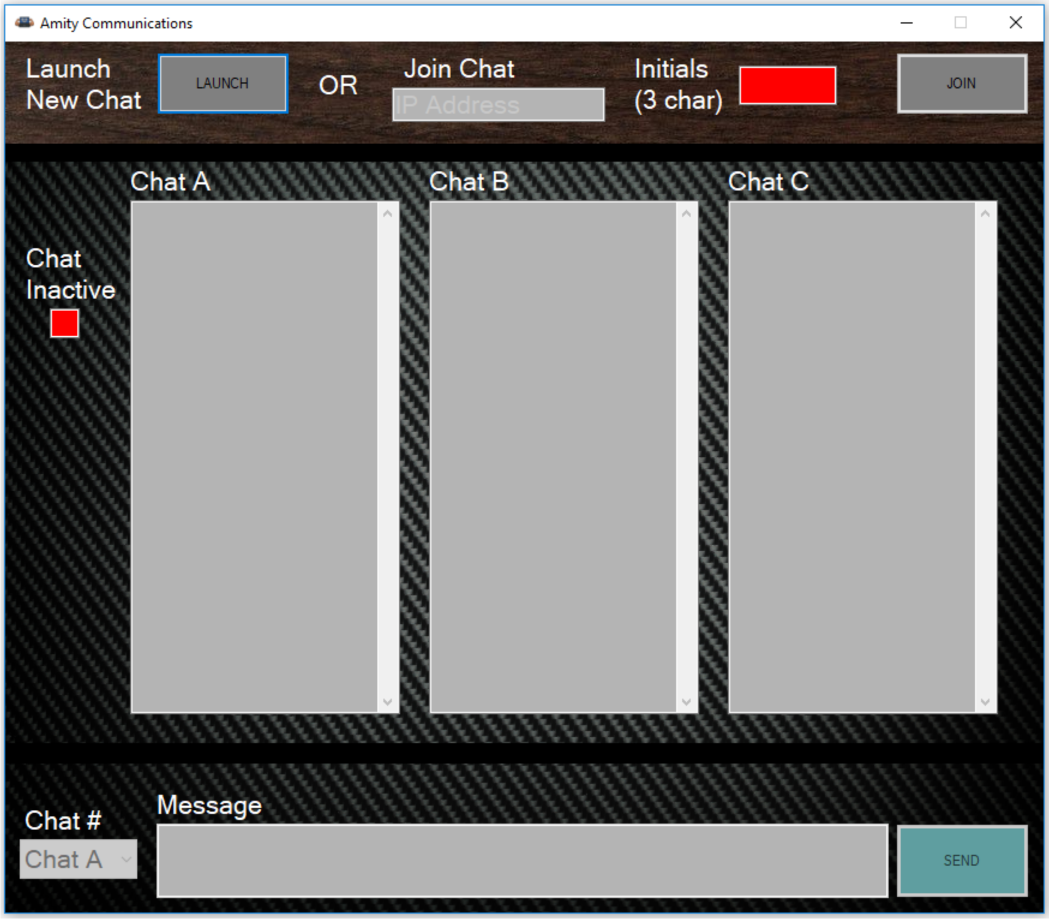
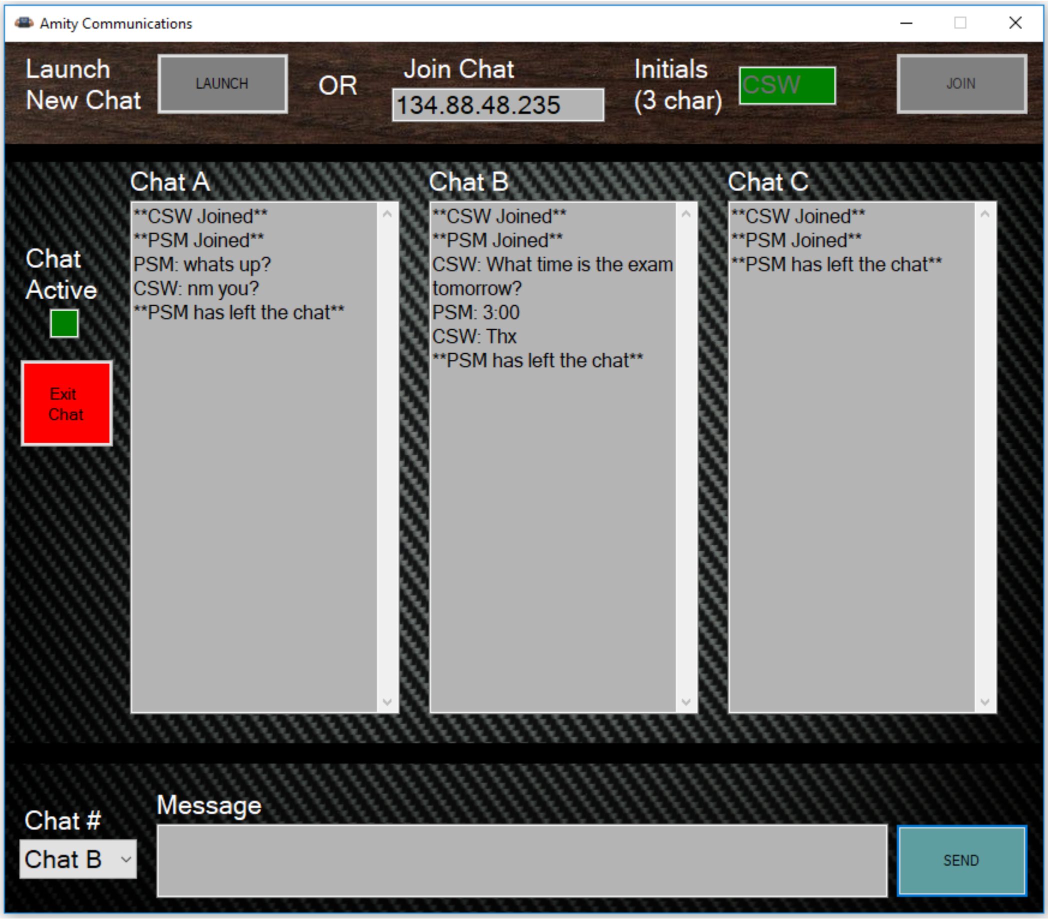


Figure 17: Amity Communications GUI on Start-up

Amity Communications is an applications that is focused on simplicity and utility. It's an easy interface that solves the cluttered classic group chat scenario. When the App is launched, users have the option to LAUNCH a new server, or JOIN an existing one. If the LAUNCH button is pressed, then a server console application will be launched in the background under the users IP. This Server is an outside Server.exe that is run on the event of the LAUNCH button click. To join that chat, the user then can type their IP into the JOIN box and press join. At this point, the light on the left hand side of the screen will indicate that they have joined a chat and they can now send data to one of the 3 chat boxes. Users can only join a chat if they have entered their 3-digit initials as a screen name that will be sent along with their messages. Messages can be typed and sent at the bottom of the screen and will appear in the corresponding chat, chosen in the drop down menu. Other users can enter the chat by typing the IP address of the server into their join chat text box and choosing a unique 3-digit screen name.



Join Chat: 134.88.48.235

Initials (3 char): CSW

Chat A

CSW Joined
PSM Joined
PSM: whats up?
CSW: nm you?
PSM has left the chat

Chat B

CSW Joined
PSM Joined
CSW: What time is the exam tomorrow?
PSM: 3:00
CSW: Thx
PSM has left the chat

Chat C

CSW Joined
PSM Joined
PSM has left the chat

Chat #: Chat B

Figure 18: Amity Communications GUI in-chat

As shown above, the users can all communicate in the 3 chats independently and their initials will appear next to their messages. The messages will all be sent to the corresponding multithreaded TCP server that they are connected to, where the message will then be echoed to all users in the chat. On the left hand side of the screen is a Leave Chat button that appears once the user is in a chat. Once this button is pressed, the user leaves the chat and their app closes. A message is sent to all users saying that you have left the chat and the chat can resume without you.

4 Discussion

Overall, our final application works within its own executable file, although it requires source files, such as the message sound effect and the server executable. The server executable runs once any user presses the LAUNCH button, although it can be forced to run separately in the file explorer. To increase the multimedia usage, we added a sound effect every time a message is received. During testing, we concluded that users can join open chats at any given time, increasing the ease of access. Also, once someone exits from a chat, all other active users still have the ability to communicate.

5 Conclusion

Over the course of the semester, the project was built piece by piece using knowledge learned in class. First, knowledge of TCP and UDP connections was improved after completing the experiments. Additionally, working with sockets to send and receive data by creating servers and clients helped to understand how messaging systems work in the real world. Creating an application in windows forms increased our ability to create GUI's that were easy to use and clearly defined. Thinking about how to refine the GUI to make it more user friendly helped us to think about the people that a product is designed for and how to make it optimized for their needs.

6 References

[1] J. Kurose, K. Ross, *Computer Networking: A Top Down-Approach 6th Ed.*, Pearson, 2013

7 Laboratory Reflection

Our team effectively completed the project that we set out to achieve. We worked through obstacles efficiently and worked on most elements together as a team to produce a finished project that we are proud of. There were a few minor design changes throughout the course of the semester but the final product functions as it should and still holds on the proposed design promises. We hope to have continued success in future projects like this one in our careers.

8 Appendix

8.1 Peer2Peer Framework Code

```
import socket
import sys
import select
running = 1
def getInput():
    i,j,k = select.select([sys.stdin], [], [], 0.0001)
    for l in i:
        if l == sys.stdin:
            input = sys.stdin.readline()
            return input
    return False
hostIP = sys.argv[1];
portVal = int(sys.argv[2]);
sendAddr = (hostIP, portVal)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
sock.setblocking(False)
sock.bind(('', portVal))
print "Connected on ", portVal
while running:
    try:
        msg, addr = sock.recvfrom(8192)
        if msg:
            print "-Message from ", addr, ": ", msg
    except:
        pass
    input = getInput();
    if input != False:
        sock.sendto(input, sendAddr)
sock.close()
sys.exit(0)
```

Listing 1: UDP Peer2Peer Framework Code

```
# Import socket module and system module
from socket import *
import sys
import time

```

```
running = 1
if len(sys.argv) <= 3:
  print 'Usage: "python TCPclient.py server_address server_port"'
  print 'server_address = Visible Inside: "eng-svr-1" or 2 or "localhost" or "127.0.0.1"'
  print '                 Visible Outside: IP address or fully qualified doman name'
  print 'server_port = server welcome socket port: #80GX'
  sys.exit(2)
# Create a TCP client socket: (AF_INET for IPv4 protocols, SOCK_STREAM for TCP)
clientSocket = socket(AF_INET, SOCK_STREAM)
#create a server socket for the server
serverSocket = socket(AF_INET, SOCK_STREAM)
clientSocket1 = socket(AF_INET, SOCK_STREAM)
serverSocket1 = socket(AF_INET, SOCK_STREAM)
# Bind the welcome socket to server address = '' any address the machine has & port = 80GX
serverSocket.bind(('', int(sys.argv[2])))
serverSocket1.bind(('', int(sys.argv[3])))
# Become a server socket by listening to at most 1 connection at a time
serverSocket.listen(1)
serverSocket1.listen(1)
time.sleep(5)
# Request a TCP connection to the TCP server welcome socket: host = argv[1] & port = argv[2]
clientSocket.connect((sys.argv[1], int(sys.argv[2])))
clientSocket1.connect((sys.argv[1], int(sys.argv[3])))
connectionSocket, address = serverSocket.accept()
print 'Accepted a connection on socket 1 from ', address
connectionSocket1, address = serverSocket1.accept()
print 'Accepted a connection on socket 2 from ', address
# Client takes message from user input, sends it to the server, and receives its echo
print 'Type "quit" to exit the client or "shutdown" to turnoff the server'
while True:
  message = raw_input ("Type a message: ")
  clientSocket.send(message)
  connectionSocket.send(message.upper())
  clientSocket1.send(message)
  #connectionSocket1.send(message.upper())
  modifiedMessage = clientSocket.recv(1024)
  print 'Received message: ', modifiedMessage
  if message == 'quit' or message == 'shutdown':
    print 'TCP Client quits!'
    break
# Close the client socket
serverSocket.shutdown(0)
serverSocket1.shutdown(0)
clientSocket.close()
clientSocket1.close()
serverSocket.close()
serverSocket1.close()
```

Listing 2: TCP Peer2Peer Framework Code

8.2 UDPclientTimer in Another Language Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/time.h>

#define PORT 8055
#define MAXLINE 1024

int main()
{
  int sockfd;
  int En = 1;
  char buffer[MAXLINE];
  char *mess;
  int n, len;
  struct sockaddr_in servaddr;
  if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
      perror("socket creation failed");
      exit(EXIT_FAILURE);
    }
  memset(&servaddr, 0, sizeof(servaddr));

  servaddr.sin_family = AF_INET; //setting up server
  servaddr.sin_port = htons(PORT);
  servaddr.sin_addr.s_addr = INADDR_ANY;
  while(En) //main sender program
    {
      struct timeval timeout={2,0}; //2 sec timeout
      printf("ENTER MESSAGE: ");
      scanf("%s", mess); //store input in mess
      sendto(sockfd, (const char *)mess, strlen(mess), MSG_CONFIRM, (const struct sockaddr *) &servaddr, sizeof(servaddr));
      printf("**SENT-->%s\n", mess); //send mess
      setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(struct timeval)); //set timeout
      n = recvfrom(sockfd, buffer, sizeof(buffer), 0, (struct sockaddr *) &servaddr, &len);
      if(n >= 0) //check timeout
  {
          buffer[n] = '\0'; //message was recieved
          printf("**REC-->%s\n", buffer);
  }
      else
  {
    printf("**TIMEOUT**\n"); //message was not recieved
  }
      if(strcmp(mess, "end") == 0 || strcmp(mess, "end\0") == 0) //end enable to quit app
  {
          En = 0;
  }
    }
  close(sockfd); //close and shutdown
  printf("...Shutting Down...\n");
  return 0;
}
```

Listing 3: UDPclientTimer in C Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
```

```

#define PORT 8055
#define MAXLINE 1024

// Driver code
int main()
{
  int sockfd;
  int En = 1;
  int len, n;
  char buffer[MAXLINE];
  struct sockaddr_in servaddr, cliaddr;
  if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) //create socket
    {
      perror("socket creation failed");
      exit(EXIT_FAILURE);
    }
  memset(&servaddr, 0, sizeof(servaddr));
  memset(&cliaddr, 0, sizeof(cliaddr));
  servaddr.sin_family     = AF_INET; //set up server
  servaddr.sin_addr.s_addr = INADDR_ANY;
  servaddr.sin_port = htons(PORT);
  if(bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) //bind socket and address
    {
      perror("bind failed");
      exit(EXIT_FAILURE);
    }
  while(En) //wait for message and echo them back
    {
      n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, ( struct sockaddr *) &cliaddr, &len);
      buffer[n] = '\0';
      printf("**REC--->%s\n", buffer);  //message was recieved
      if(strcmp(buffer, "end") == 0 || strcmp(buffer, "end\0") == 0) //end and close if desired
      {
              En = 0;
      }
          sendto(sockfd, (const char *)buffer, strlen(buffer), MSG_CONFIRM, ( const struct sockaddr *) &cliaddr, len);
          printf("**SENT--->%s\n", buffer); //echo message
        }
      printf("...Shutting Down...\n");
      return 0;
  }
```

Listing 4: UDPserver in C Code

8.3 Amity Communications Windows Forms Program

```
/******************************
     Amity Communications
   ECE369 Computer Networks
******************************/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Threading;
using System.Net.Sockets;
using System.Net.NetworkInformation;
using System.Diagnostics;
using System.Media;

namespace GUI
{
    public partial class Form1 : Form
    {
        System.Net.Sockets.TcpClient clientSocket = new System.Net.Sockets.
   TcpClient();
        NetworkStream serverStream = default(NetworkStream);
        string readData = null;
        public static string IPz;
        public Form1()
        {
            InitializeComponent();
        }
        public static class Globals //Global variables stored in class
        {
            public static IPAddress ipAddress = IPAddress.Parse(IPz);
            public static IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());

            public static IPEndPoint remoteEP = new IPEndPoint(ipAddress, 8050);
            public static Socket Sender;

            public static bool running = false;
            public static string data = null;
        }
//*****************************************************
        private void Form1_Load(object sender, EventArgs e) //On Application startup
        {
            ChatCB.SelectedIndex = 0; //set to defaults
            ActiveTB.BackColor = Color.Red;
            ExitB.Hide();
            SendB.Enabled = false;
            InputTB.Enabled = false;
            ChatCB.Enabled = false;
        }
//*****************************************************
        private void JoinB_Click(object sender, EventArgs e) //Join chat with given IP
        {
            int NumInit = InitTB.Text.Length; //Initial length check
            if (NumInit == 3)
            {
                byte[] bytes = new byte[1024];
                string S = "";
                try
                {
                    clientSocket.Connect(IPz, 8050);
                    serverStream = clientSocket.GetStream();
                    byte[] outStream = System.Text.Encoding.ASCII.GetBytes("3" + InitTB.Text + "$");
                    serverStream.Write(outStream, 0, outStream.Length);
                    serverStream.Flush();
                    Thread ctThread = new Thread(getMessage);
                    ctThread.Start();
                    Globals.running = true;
                    JoinB.Enabled = false;
                    InitTB.Enabled = false;
                    SendB.Enabled = true;
                    InputTB.Enabled = true;
                    ChatCB.Enabled = true;
                    ActiveTB.BackColor = Color.Green;
                    ActiveL.Text = "Chat\nActive";
                    ExitB.Show();
                }
                catch (Exception) //Error message for no server
                {
                    System.Windows.Forms.MessageBox.Show("ERROR: CANT CONNECT");
                }
            }
            else //Error message for invalid Init
                System.Windows.Forms.MessageBox.Show("Initials must be\n3 characters");
        }
//*******************************************************
        private void ChooseChatCLB_ItemCheck(object sender, ItemCheckEventArgs e) //Chat selector
        {
            for (int ix = 0; ix < ChooseChatCLB.Items.Count; ++ix)
                if (ix != e.Index) ChooseChatCLB.SetItemChecked(ix, false);
        }
//*******************************************************
        private void ExitB_Click(object sender, EventArgs e) //Leaving a chat
        {
            byte[] outStream = System.Text.Encoding.ASCII.GetBytes(InitTB.Text + "_QUITTING$");
            serverStream.Write(outStream, 0, outStream.Length);
            serverStream.Flush();
            System.Windows.Forms.MessageBox.Show("You have left the chat...");
            ChatATB.Text = String.Empty; //Reset all to default
            ChatBTB.Text = String.Empty;
            ChatCTB.Text = String.Empty;
            InputTB.Text = String.Empty;
            InitTB.Text = String.Empty;
            ActiveL.Text = "Chat\nInactive";
            ActiveTB.BackColor = Color.Red;
            JoinB.Enabled = true;
            InitTB.Enabled = true;
            ExitB.Hide();
            clientSocket.GetStream().Close();
        }
//******************************************************
        private void JoinChatTB_TextChanged(object sender, EventArgs e) //Set IPz variable on text change
        {
            IPz = JoinChatTB.Text;
        }
//******************************************************
        private void JoinChatTB_Click(object sender, EventArgs e) //Default description in TB
        {
            if (JoinChatTB.Text == "IP Address")
            {
                JoinChatTB.Text = "";
                JoinChatTB.ForeColor = Color.Black;
            }
        }
//******************************************************
        private void InitTB_TextChanged_1(object sender, EventArgs e) //Color effects on Init TB
        {
            int NumInit = InitTB.Text.Length;
            if (NumInit != 3)
                InitTB.BackColor = Color.Red;
            else
                InitTB.BackColor = Color.Green;
        }
//******************************************************
        private void SendB_Click_1(object sender, EventArgs e) //Send a message
        {
            int NumInit = InitTB.Text.Length;
            if (NumInit == 3)
            {
                string S = "";
                S = ChatCB.SelectedIndex + InitTB.Text + InputTB.Text;

                byte[] outStream = System.Text.Encoding.ASCII.GetBytes(S + "$");
                serverStream.Write(outStream, 0, outStream.Length);
                serverStream.Flush();
                InputTB.Text = String.Empty;
            }
            else
                System.Windows.Forms.MessageBox.Show("Initials must be\n3 characters");
        }
//*******************************************************
        private void getMessage() //Recieve a message
        {
            while (true)
            {
                serverStream = clientSocket.GetStream();
                int buffSize = 0;
                byte[] inStream = new byte[10025];
                buffSize = clientSocket.ReceiveBufferSize;
                serverStream.Read(inStream, 0, inStream.Length);
                string returndata = System.Text.Encoding.ASCII.GetString(inStream);
                readData = "" + returndata;
                msg();
            }
        }
//*******************************************************
        private void msg() //Prints message to screen (called from getMessage)
        {
            string S = ChatCB.SelectedIndex + InitTB.Text + InputTB.Text;
            string outputText = readData.Substring(0, S.Length);
            string MessID = outputText.Substring(1, 3);
            outputText = readData.Substring(4);
            SoundPlayer simpleSound = new SoundPlayer(@"\\umdfs1.umdar.umassd.edu\studentshares$\pmcgrory\Desktop\FINAL SYSTEM2.0\GUI\msgRec.wav");
            if (this.InvokeRequired)
                this.Invoke(new MethodInvoker(msg));
            else
                switch ((int)Char.GetNumericValue(readData[0])) //Choose which chat
                {
                    case 0: //Chat A
                        readData = readData.Substring(1, readData.Length - 1);
                        ChatATB.AppendText(MessID + ": " + outputText);
                        ChatATB.Text += "\r\n";
                        simpleSound.Play(); //Play sound on message
                        break;
                    case 1: //Chat B
                        readData = readData.Substring(1, readData.Length - 1);
                        ChatBTB.AppendText(MessID + ": " + outputText);
                        ChatBTB.Text += "\r\n";
                        simpleSound.Play(); //Play sound on message
                        break;
                    case 2: //Chat C
                        readData = readData.Substring(1, readData.Length - 1);
                        ChatCTB.AppendText(MessID + ": " + outputText);
                        ChatCTB.Text += "\r\n";
                        simpleSound.Play(); //Play sound on message
                        break;
                    case 3: //Chat ALL
                        readData = readData.Substring(1, readData.Length - 1);
                        ChatATB.AppendText(readData + Environment.NewLine);
                        ChatBTB.AppendText(readData + Environment.NewLine);
                        ChatCTB.AppendText(readData + Environment.NewLine);
                        ChatATB.Text += "\r\n";
                        ChatBTB.Text += "\r\n";
                        ChatCTB.Text += "\r\n";
                        break;
                }
        }
//********************************************************
        private void LaunchB_Click(object sender, EventArgs e) //Launch Server EXE
        {
            ProcessStartInfo startInfo = new ProcessStartInfo();
            startInfo.FileName = @"\\umdfs1.umdar.umassd.edu\studentshares$\pmcgrory\Desktop\FINAL SYSTEM2.0\ServerTest\ServerTest\bin\Debug\netcoreapp2.0\win10-x64\ServerTest.exe";
            startInfo.WindowStyle = System.Diagnostics.ProcessWindowStyle.Minimized;
        }
//********************************************************
        private void ChatATB_TextChanged(object sender, EventArgs e) //Scrolling funct A
        {
            ChatATB.SelectionStart = ChatATB.TextLength;
            ChatATB.ScrollToCaret();
        }
//********************************************************
        private void ChatBTB_TextChanged(object sender, EventArgs e) //Scrolling funct B
        {
            ChatBTB.SelectionStart = ChatBTB.TextLength;
            ChatBTB.ScrollToCaret();
        }
//*******************************************************
        private void ChatCTB_TextChanged(object sender, EventArgs e) //
    Scrolling funct C
        {
            ChatCTB.SelectionStart = ChatCTB.TextLength;
            ChatCTB.ScrollToCaret();
        }
//*******************************************************
    }
}
```

Listing 5: Amity Communications in C#

```
/******************************
          Server EXE
   ECE369 Computer Networks
******************************/
using System;
using System.Threading;
using System.Net.Sockets;
using System.Text;
using System.Collections;

namespace ConsoleApplication1
{
    class Program
    {
        public static Hashtable clientsList = new Hashtable();
        static void Main(string[] args)
        {
            System.Diagnostics.ProcessStartInfo start = new System.Diagnostics
    .ProcessStartInfo();
            start.FileName = @"\ServerTest.exe";
            start.WindowStyle = System.Diagnostics.ProcessWindowStyle.Hidden;
    //Server doesnt pop up when created in GUI
            TcpListener serverSocket = new TcpListener(8050);
            TcpClient clientSocket = default(TcpClient);
            int counter = 0;
            serverSocket.Start();
            Console.WriteLine("Chat Server Started ....");
            counter = 0;
            while ((true)) //Server accepts new users
            {
                counter += 1;
                clientSocket = serverSocket.AcceptTcpClient();
                byte[] bytesFrom = new byte[10025];
                string dataFromClient = null;
                NetworkStream networkStream = clientSocket.GetStream();
                networkStream.Read(bytesFrom, 0, bytesFrom.Length);
                dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
                dataFromClient = dataFromClient.Substring(0, dataFromClient.IndexOf("$"));
                clientsList.Add(dataFromClient, clientSocket); //Adds new client to list of all clients
                broadcast("3**" + dataFromClient.Substring(1) + " Joined**", dataFromClient, false);
                Console.WriteLine(dataFromClient.Substring(1) + " Joined chat room ");
                handleClient client = new handleClient();
                client.startClient(clientSocket, dataFromClient.Substring(1), clientsList);
            }
            clientSocket.Close();
            serverSocket.Stop();
            Console.WriteLine("exit");
            Console.ReadLine();
        }
        public static void broadcast(string msg, string uName, bool flag) //Sends message to clients
        {
            foreach (DictionaryEntry Item in clientsList)
            {
                TcpClient broadcastSocket;
                broadcastSocket = (TcpClient)Item.Value;
                NetworkStream broadcastStream = broadcastSocket.GetStream();
                Byte[] broadcastBytes = null;
                if (flag == true)
                {
                    broadcastBytes = Encoding.ASCII.GetBytes(msg);
                }
                else
                {
                    broadcastBytes = Encoding.ASCII.GetBytes(msg);
                }
                broadcastStream.Write(broadcastBytes, 0, broadcastBytes.Length); //Sends the encoded bytes through the stream
                broadcastStream.Flush();
            }
        }
    }
    public class handleClient //Class holding client information
    {
        TcpClient clientSocket;
        string clNo;
        Hashtable clientsList;
        public void startClient(TcpClient inClientSocket, string clineNo, Hashtable cList)
        {
            this.clientSocket = inClientSocket;
            this.clNo = clineNo;
            this.clientsList = cList;
            Thread ctThread = new Thread(doChat);
            ctThread.Start();
        }
        private void doChat() //A message was sent from a client
        {
            int requestCount = 0;
            byte[] bytesFrom = new byte[10025];
            string dataFromClient = null;
            Byte[] sendBytes = null;
            string serverResponse = null;
            string rCount = null;
            requestCount = 0;
            while ((true))
            {
                try
                {
                    requestCount = requestCount + 1;
                    NetworkStream networkStream = clientSocket.GetStream();
                    networkStream.Read(bytesFrom, 0, bytesFrom.Length);
                    dataFromClient = System.Text.Encoding.ASCII.GetString(bytesFrom);
                    dataFromClient = dataFromClient.Substring(0, dataFromClient.IndexOf("$")); //Gets users full message

                    if (dataFromClient.Substring(3) == "_QUITTING") //If user
```

```
pressed the exit button
                    {
                        clientsList.Remove(dataFromClient.Substring(0, 3)); //Removed from hashtable
                        Console.WriteLine("CLIENT LEAVING: " + dataFromClient.Substring(0, 3));
                        rCount = Convert.ToString(requestCount);
                        Program.broadcast("3**" + dataFromClient.Substring(0, 3) + " has left the chat**", clNo, true); //Sends message of who left
                        return;
                    }
                    else if (dataFromClient.Length == 4) //If user sent empty message
                    {
                        //Dont send empty message
                    }
                    else //User sent a normal message
                    {
                        Console.WriteLine("From client - " + clNo + " : " + dataFromClient);
                        rCount = Convert.ToString(requestCount);
                        Program.broadcast(dataFromClient, clNo, true); //Sends message to clients
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.ToString());
                }
            }
        }
    }
}
```

Listing 6: Server Executable in C#