

# **ECGR 3123 Project 1**

**Grayson Bass**

**Landon Gibson**

## **Basic Requirements:**

### **Introduction:**

The goal of this project was to design an application that would allow a client and server to continuously exchange messages. The programming language of choice is go. Go was chosen because it is relatively straightforward to implement these sockets. It also has easy thread support allowing the client to wait for an incoming message as well as write a message simultaneously. This was achieved using both TCP and UDP sockets. For the base requirements, the client application accepts user input and the server echos the message back to the client. This demonstrates that the client and server have established a connection and messages are sent and received. Included in the zip file, there are two gifs demonstrating TCP and UDP group chats. Additionally, the go files are included in their respective folders.

### **TCP:**

For TCP, the connection between the server and client requires establishment, unlike UDP that acts as an open door. The TCP portion of the project was done in a single file (tcpChat.go) using goroutines and go channels for threading. There are four functions, main (line 28), server (line 46), handleConn (line 83), and clientWriter (line 110). The main function establishes a connection to the localhost at port 8000 and calls the goroutines: server and handleConn. The server holds the map with all of the connected clients as well as protocol for broadcasting messages to all of the currently connected users. The server is split into three sections using cases which are “messages” to broadcast messages, “entering” which adds a user to the map and broadcasts currently connected users, and the last case is “leaving” which removes the user from the map and closes the connection to the client. handleConn gets passed a connection value and uses that value to collect information on who the user is and broadcast a welcome message to other users. Additionally, handleConn facilitates the messages that users create and sends them to the clientWriter. clientWriter then takes a connection, and a string, which will then send the message to all of the client’s servers using go channels. In figure 1 below, there is a client

connecting to the server and informing the server of the client's name. The server confirms the client's name, then tells the client how many people are connected to the server. The server runs in one terminal while the client connects and uses Netcat in another terminal to connect to the server.

```
landon@london-VirtualBox:~/DataCommunications$ netcat localhost 8000
Who are you?
Landon Gibson
You are Landon Gibson

Present Clients: You are currently alone
Landon Gibson
```

*Figure 1: Landon joins the server*

In figure 2 you can see that the server is responding to the client with the message that the client sent. This confirms that the server has received the client's message, and is able to respond to the tcp connection.

```
landon@london-VirtualBox:~/DataCommunications$ netcat localhost 8000
Who are you?
Landon Gibson
You are Landon Gibson

Present Clients: You are currently alone
Landon Gibson

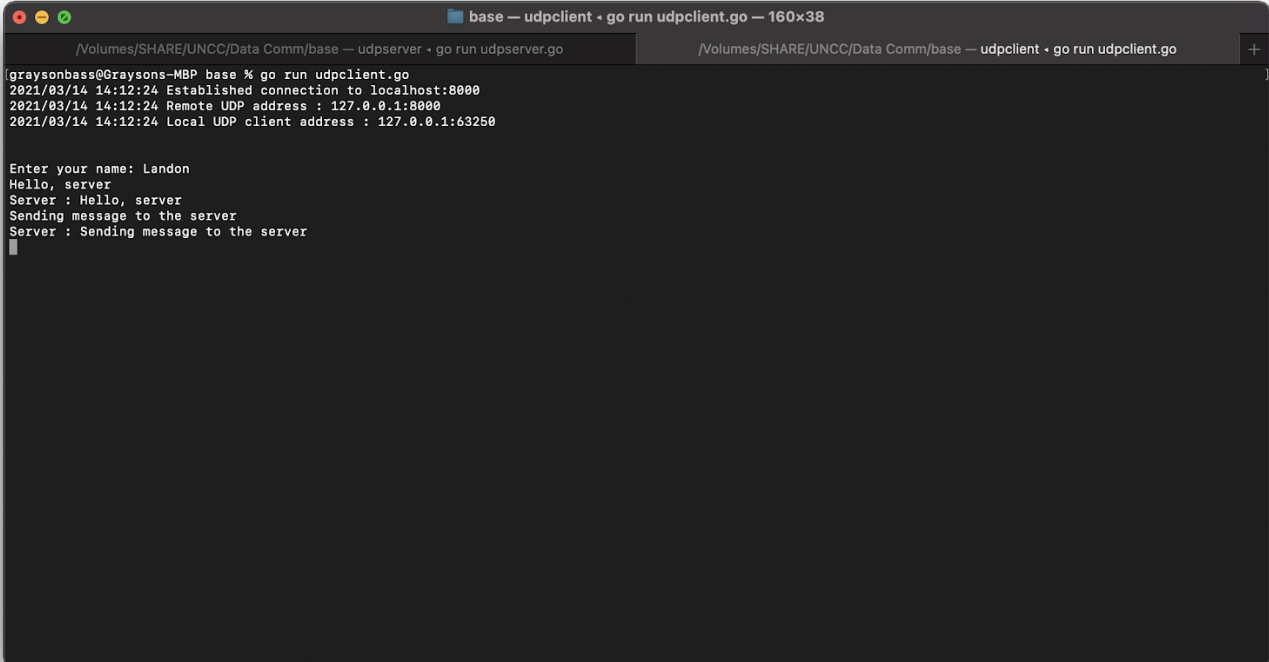
Hello, Server
Landon Gibson: Hello, Server
```

*Figure 2: Server receives a message and responds to client*

## UDP:

For the UDP, the connection between the client and server does not require establishment and the server listens for any incoming message. The client and server are handled using separate files and were run using two separate terminal windows on the same computer. The net library was included to give support for establishing a connection to the client as well as listening for incoming messages. In the main function of the server (line 60), `ResolveUDPAddr` was used from the net library and it returns a local IP address and specified port and allows for the specification of the UDP protocol. After the address is found and the protocol is specified, the server uses the `listenUDP` function to listen for incoming messages on the local address and port.

Once a message has been seen by the server, the server echos the message back to the client. This is achieved inside the readWrite function (line 8). This is done in an infinite while loop to continuously poll for incoming messages. The message is also displayed on the server with the name of the client that has joined the server. For the main function in the client (line 12), the same ResolveUDPAddr was used to receive the local IP address and port of the client. After that, DialUDP is used to establish a connection with the server. The client accepts user input and then encodes it using a byte array then sends it to the server. The client then displays the echoed message from the server. The readFromServer function(line 45) handles the incoming message from the server and displays it. The entirety of the line is displayed from the server including the server and colon, “server : message.” Figure 3 shows the client exchanging messages with the server:

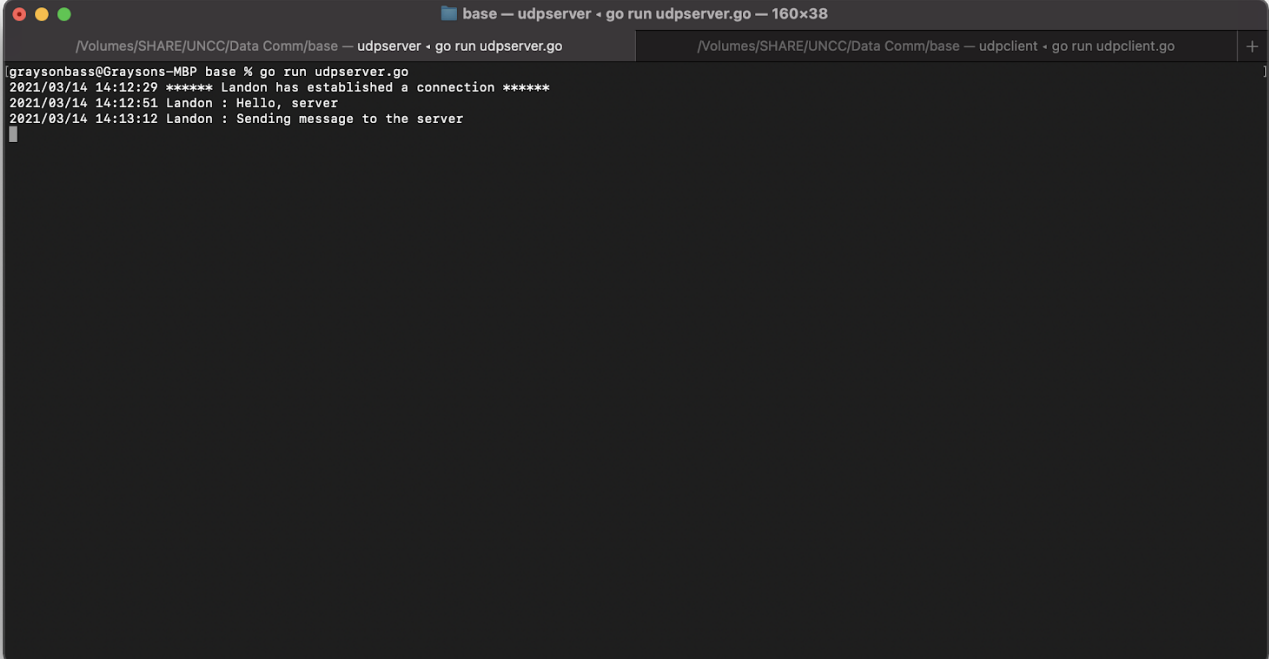
A terminal window titled "base — udpclient · go run udpclient.go — 160x38" is shown. The window has two tabs: "/Volumes/SHARE/UNCC/Data Comm/base — udpserver · go run udpserver.go" and "/Volumes/SHARE/UNCC/Data Comm/base — udpclient · go run udpclient.go". The active tab is the client. The terminal output shows the client running "go run udpclient.go", establishing a connection to localhost:8000, and displaying the remote and local UDP addresses. It then prompts the user to enter a name, which is "Landon". The client sends "Hello, server" to the server, and the server responds with "Server : Hello, server". The client then sends "Sending message to the server" to the server, and the server responds with "Server : Sending message to the server".

```
graysonbass@Graysons-MBP base % go run udpclient.go
2021/03/14 14:12:24 Established connection to localhost:8000
2021/03/14 14:12:24 Remote UDP address : 127.0.0.1:8000
2021/03/14 14:12:24 Local UDP client address : 127.0.0.1:63250

Enter your name: Landon
Hello, server
Server : Hello, server
Sending message to the server
Server : Sending message to the server
```

*Figure 3: The client-side of UDP*

Figure 4 shows the server receiving the messages and printing them:

A screenshot of a macOS terminal window with a dark background. The title bar at the top reads "base — udpserver • go run udpserver.go — 160x38". There are two tabs: the active one is "/Volumes/SHARE/UNCC/Data Comm/base — udpserver • go run udpserver.go" and the other is "/Volumes/SHARE/UNCC/Data Comm/base — udpclient • go run udpclient.go". The terminal output shows the following lines:

```
graysonbass@Graysons-MBP base % go run udpserver.go
2021/03/14 14:12:29 ***** Landon has established a connection *****
2021/03/14 14:12:51 Landon : Hello, server
2021/03/14 14:13:12 Landon : Sending message to the server
```

## Client - Server - Client Communication:

### TCP:

For client-to-client communication through the server, goroutines and channels really come into play. Threading in golang is made very easy as it is a programming language designed for networking and multithreading. The goroutines allowed us to use the same code for the server and client. Channels allowed the messages to be exchanged between goroutines that would be created every time a client joins the server. Figure 5 shows a second user connecting to the server and broadcasting their connection status. When a client joins, the server informs the user how many people are in chat and informs other users that a new client has been added. Figure 6 shows the communication between two users through the server.

```

Landon@Landon-VirtualBox:~/DataCommunications$ netcat localhost 8000
Who are you?
Grayson Bass
You are Grayson Bass

Present Clients: There are currently 1 other clients
Landon Gibson
Grayson Bass

```

*Figure 5: The second user joins the server*

```

Landon@Landon-VirtualBox:~/DataCommunications$ netcat localhost 8000
Who are you?
Landon Gibson
You are Landon Gibson

Present Clients: You are currently alone
Landon Gibson

Hello, Server
Landon Gibson: Hello, Server
Grayson Bass has arrived

Grayson Bass: Hello, Landon

```

*Figure 6: Server passes Grayson's message to Landon*

## UDP:

For the client-side, there has not been much of a change at all compared to the base project. The only change that was made is the user's message will not be sent back to them from the server. It will only display the other user's name and message. This prevents the message from displaying twice. Threads were also implemented on the client side to simultaneously poll for a message from the server and also accept user input to send to the server. Also the user can now type "quit" anytime to leave the chat. Most of the change was made to the server. A map and slice were both used to store the ip/port of the client as well as their name. This allowed the server to hand two clients. The ip/port upon the client sending a message is checked with the map containing the clients that have already been established and the message is sent with the client's name to the other clients on the server. The message is also displayed on the server side. Figure 7 shows the output of client 1:

```
extra credit — udpclient • go run udpclient.go — 160x38
...ta Comm/extra credit — udpserver • go run udpserver.go
...a Comm/extra credit — udpclient • go run udpclient.go
...omm/extra credit — udpclient • go run udpclient.go
]
graysonbass@Graysons-MBP extra credit % go run udpclient.go
2021/03/14 14:57:05 Established connection to localhost:8000
2021/03/14 14:57:05 Remote UDP address : 127.0.0.1:8000
2021/03/14 14:57:05 Local UDP client address : 127.0.0.1:51399

Welcome to the server
Enter 'quit' to leave the server

What is your name : Grayson
Hello Landon
Landon : Hello Grayson
😊
█
```

*Figure 7: Client 1 side UDP*

Figure 8 shows client 2 connected to the server and exchanging messages with client 1:

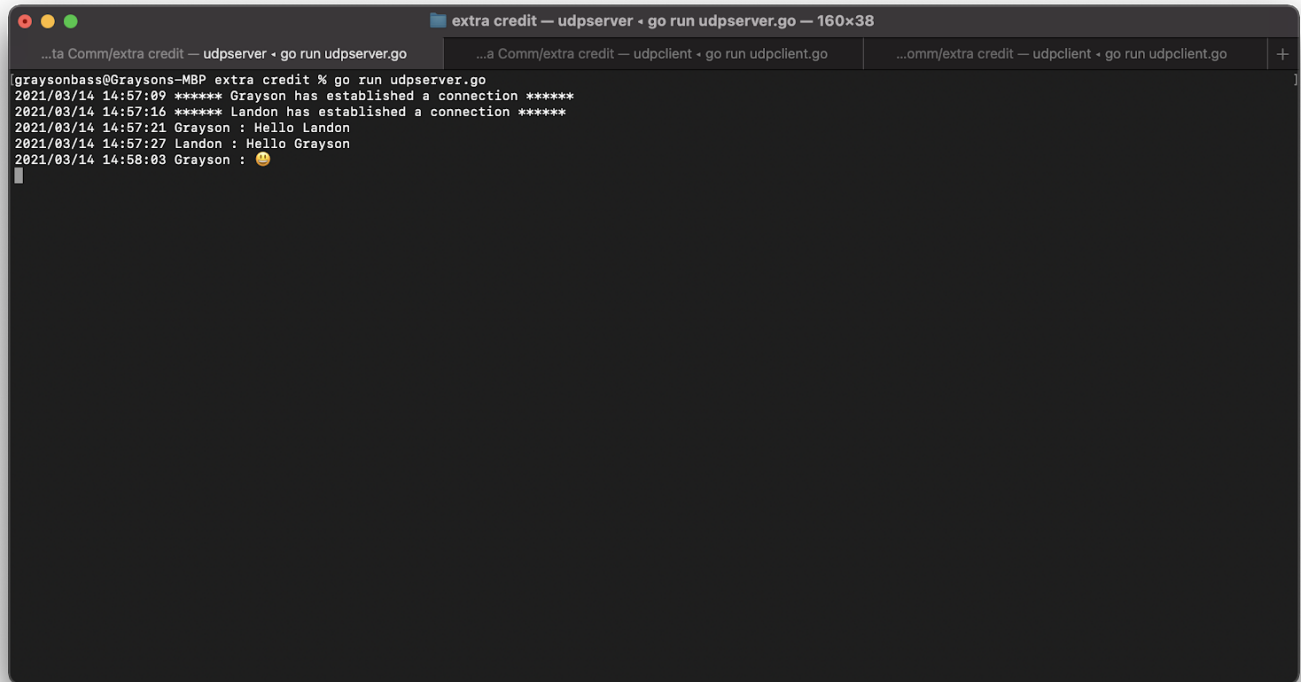
```
extra credit — udpclient • go run udpclient.go — 160x38
...ta Comm/extra credit — udpserver • go run udpserver.go
...a Comm/extra credit — udpclient • go run udpclient.go
...omm/extra credit — udpclient • go run udpclient.go
]
Last login: Sun Mar 14 13:38:59 on ttys003
graysonbass@Graysons-MBP extra credit % go run udpclient.go
2021/03/14 14:57:13 Established connection to localhost:8000
2021/03/14 14:57:13 Remote UDP address : 127.0.0.1:8000
2021/03/14 14:57:13 Local UDP client address : 127.0.0.1:57057

Welcome to the server
Enter 'quit' to leave the server

What is your name : Landon
Grayson : Hello Landon
Hello Grayson
Grayson : 😊
█
```



Figure 9 shows the server side with client 1 and 2 both connected and exchanging messages:

A terminal window titled "extra credit - udpserver - go run udpserver.go - 160x38" is shown. It contains the following text:

```
graysonbass@Graysons-MBP extra credit % go run udpserver.go
2021/03/14 14:57:09 ***** Grayson has established a connection *****
2021/03/14 14:57:16 ***** Landon has established a connection *****
2021/03/14 14:57:21 Grayson : Hello Landon
2021/03/14 14:57:27 Landon : Hello Grayson
2021/03/14 14:58:03 Grayson : 😊
```

## Client - Server - Group Chat:

### TCP:

Our TCP implementation of the group chat is highly scalable and in figure 10, 5 users will join the chat and communicate with one another. In the figure, only the first client's perspective is shown for simplicity's sake.

```
london@london-VirtualBox:~/DataCommunications$ netcat localhost 8000
Who are you?
Client 1
You are Client 1

Present Clients: You are currently alone
Client 1

Client 2 has arrived
Client 3 has arrived
Client 4 has arrived
Client 5 has arrived

Hello, everyone!
Client 1: Hello, everyone!
Client 2: Hey Client 1!
Client 3: Hey Client 1! How are you?
Client 4: Hey Client 1! What's up?
Client 5: Hey Client 1! How has your day been?
```

*Figure 10: Client 1 sends a message to the group chat, and receives a unique message from all clients.*

## UDP:

Our UDP implementation of the group chat is highly scalable and in figure 11, 5 users will join the chat and communicate with one another. Figure 11 shows the perspective of the server to cut down on the number of images. Also in figure 11, clients can join and leave at any point



```
extra credit — udpserver · go run udpserver.go — 160x38
...go run udpserver.go  .../extra credit — -zsh  .../extra credit — -zsh  .../extra credit — -zsh  .../extra credit — -zsh  .../extra credit — -zsh  +
graysonbass@Graysons-MBP extra credit % go run udpserver.go
2021/03/14 14:57:09 ***** Grayson has established a connection *****
2021/03/14 14:57:16 ***** Landon has established a connection *****
2021/03/14 14:57:21 Grayson : Hello Landon
2021/03/14 14:57:27 Landon : Hello Grayson
2021/03/14 14:58:03 Grayson : 😊
2021/03/14 15:05:56 ***** Jesse has established a connection *****
2021/03/14 15:06:13 Jesse : Hello Grayson and Landon
2021/03/14 15:06:31 ***** Bob has established a connection *****
2021/03/14 15:07:06 Bob : Hello everyone in the server 😊
2021/03/14 15:08:43 Jesse : Hello
2021/03/14 15:08:46 Landon : Hello
2021/03/14 15:08:48 Grayson : Hello
2021/03/14 15:09:08 ***** jim has established a connection *****
2021/03/14 15:09:13 jim : Hello everyone
2021/03/14 15:09:23 Grayson : See y'all later
2021/03/14 15:09:25 ***** Grayson has disconnected *****
2021/03/14 15:09:37 Landon : bye
2021/03/14 15:09:39 ***** Landon has disconnected *****
2021/03/14 15:09:45 Jesse : bye
2021/03/14 15:09:46 ***** Jesse has disconnected *****
2021/03/14 15:09:49 Bob : bye
2021/03/14 15:09:50 ***** Bob has disconnected *****
2021/03/14 15:09:57 jim : bye
2021/03/14 15:09:59 ***** jim has disconnected *****
```

Figure 11: Server side of a group chat of 5 people using UDP