Portable Client-Based Communication Server

Gurdeep Singh, Hans Singh, Aayoob Min, Sandeep Singh
COMP1549: Advanced Programming
University of Greenwich
Old Royal Naval College
United Kingdom

Abs*tract*—
Social media platform usage is increasing overtime with this threats and data breaches are evolving and increasing. Our group decided to make this problem ours and have created a program to allow users to safely communicate without worrying about data breaches.

Key words: communication, private, portable, reliable

I. Introduction

What is 'GangChat'?

In this group project we will highlight the use of 'GangChat' a client-based communication server with its main feature allowing it to be portable.

In this project we as a team created an application using Python as our programming language.

As modern adults we have come across a lot of communication applications that have taken the world by surprise, even though these new chat applications look appealing to its user the messages you send are still at risk from being leaked not only by the company but also third-party hackers. Therefore, we wanted something unique to be our main feature hence, we decided to make this chat extremely secretive allowing our users to feel at ease as their data will be in their own hands and once the chat is finished so does all traces of it. This is 'GangChat' unique selling point, it is extremely private and portable so wherever and whenever.

Our source of motivations derives from chatgroups such as 'Facebook' and 'WhatsApp', as devoted users of this app we love it, however, you can always get spammed randomly by some unknown bot due to promotion or even a scam, which is annoying when it reaches a certain point therefore, we decided to create something that prevents this.

For example, a key factor on creating our chat program was because of the incident on 'WhatsApp' that occurred in '2019' when a bug appeared allowing random users to search and gain access to group chats. '4,700,000' group invite links were compromised which indicated millions of data that can be sensitive was out to the world to be seen (1).

This gave our group a motive to allow us to create a server where we can communicate with each other without the worry of our data being leaked.

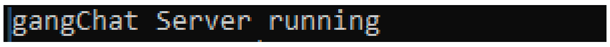
This report will focus on the design of our program so that we can explain how it works and show its aspects. We will also be analysing what we have investigated during this project and lastly, conclude the work done and show what can be done in the future.

II. Design/implementation

Gang Chat- is a client-based communication server which is run on CLI (Command line interface), it comes with many components.

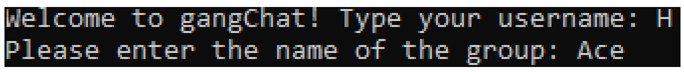
The first step to run the code is to launch the server.bat file which automatically starts the server; however, the file must be configured to the users setting to work.

Once the server has started the following messaging is shown:



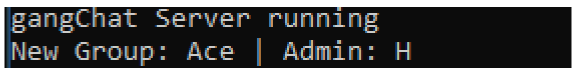
```
gangChat Server running
```

This message confirms that the server is running, and the next stage will be to run the client.bat file which is also the main file. Once that is running the message will be:



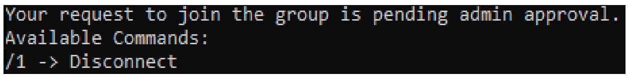
```
Welcome to gangChat! Type your username: H
Please enter the name of the group: Ace
```

The client will ask the user to enter their username which is also their ID and they are then asked to create the group's name Once this stage is completed the server is successfully created and the server client will display the following message:



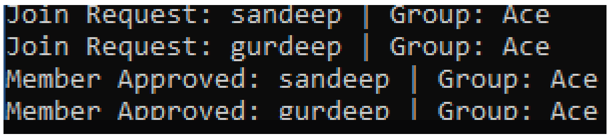
```
gangChat Server running
New Group: Ace | Admin: H
```

The next stage will be connecting the server, the second user will be allowed to choose their ID and then send a request to join the server which the admin must approve to join:



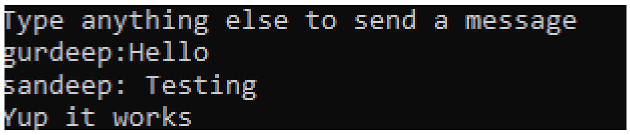
```
Your request to join the group is pending admin approval.
Available Commands:
/1 -> Disconnect
```

The command '/1' is the disconnect command which allows you to terminate your client and leave the chatroom.



```
Join Request: sandeep | Group: Ace
Join Request: gurdeep | Group: Ace
Member Approved: sandeep | Group: Ace
Member Approved: gurdeep | Group: Ace
```

Once all members are approved and confirmed they can start communicating:



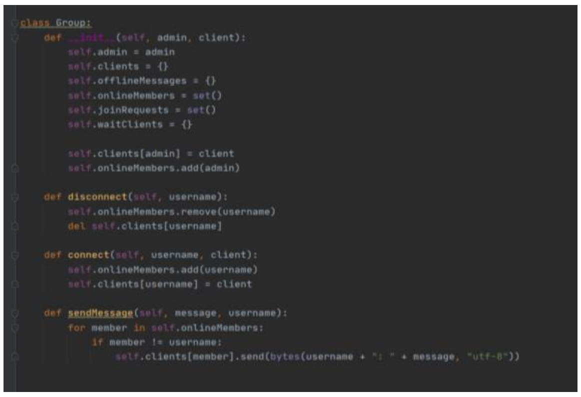
```
Type anything else to send a message
gurdeep:Hello
sandeep: Testing
Yup it works
```

The communication is successful.

MODULAR DESIGN PATTERN

Module design pattern is a vital pattern used to combine variables and functions in a sole scope so that only one case exists. This allows objects to get defined, variables and functions that are opened from outside of the scope are specific (5).

The image below, is part of our coded program and this part is a small part to show where we have implemented modularity by using design patterns. In our code we have used variables like 'class' and 'def' so that the variables and functions can be in a sole scope. In essence module pattern allows us to create a code with less classes and subclasses in comparison to having one class wrapped with another class. This is very useful to have as it stops the exposure of the methods that are wrapped (4).

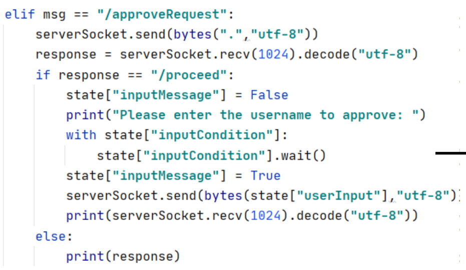


```
class Group:
    def __init__(self, admin, client):
        self.admin = admin
        self.clients = {}
        self.offlineMessages = {}
        self.onlineMembers = set()
        self.joinRequests = set()
        self.waitClients = {}

        self.clients[admin] = client
        self.onlineMembers.add(admin)

    def disconnect(self, username):
        self.onlineMembers.remove(username)
        del self.clients[username]

    def connect(self, username, client):
        self.onlineMembers.add(username)
        self.clients[username] = client

    def sendMessage(self, message, username):
        for member in self.onlineMembers:
            if member != username:
                self.clients[member].send(bytes(username + ": " + message, "utf-8"))
```

Afterwards, we focused on component-based development, this assisted us with making a fully functioning server and client-based chat. Most of our component-based development was linked through the client class as we made built in classes which helped having everything linked in just two classes instead of creating a class for each component-based development.

COMPONENT BASED DEVELOPMENT

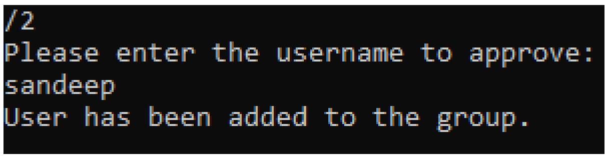
First component was 'Approve Request', when the server is running, you then need to run the client when you have run the client you become the admin. Furthermore, when you have become the admin you now have the power to approve the join requests and with the help of 'View request' you can accept and deny the join request if you do not want a certain person to join.



```
elif msg == "/approveRequest":
    serverSocket.send(bytes(".","utf-8"))
    response = serverSocket.recv(1024).decode("utf-8")
    if response == "/proceed":
        state["inputMessage"] = False
        print("Please enter the username to approve: ")
        with state["inputCondition"]:
            state["inputCondition"].wait()
        state["inputMessage"] = True
        serverSocket.send(bytes(state["userInput"],"utf-8"))
        print(serverSocket.recv(1024).decode("utf-8"))
    else:
        print(response)
```

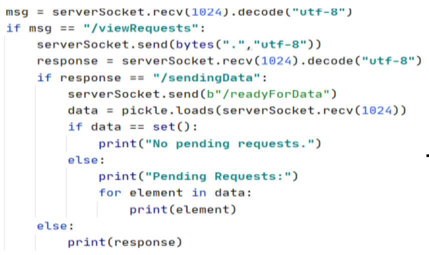
On the left is the code for the first (fig1) component. 'Approve Request' code

Fig1



```
/2
Please enter the username to approve:
sandeep
User has been added to the group.
```

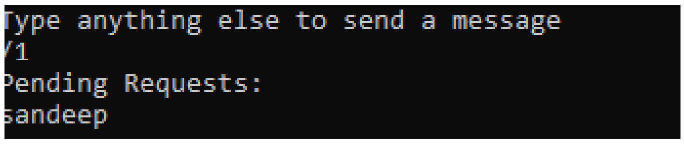
Second component was 'View Requests', as we ran the server then ran the client, the first person who joined would have needed to create a room to chat in plus once they made the room, they would be assigned the admin ship for the whole server if you are the first person joining. After you have made the room and chat the administrator would need a command to see who would like to join his/her chatroom, thus why having 'View Requests' would allow him/her to see who wants to join.



```
msg = serverSocket.recv(1024).decode("utf-8")
if msg == "/viewRequests":
    serverSocket.send(bytes(".","utf-8"))
    response = serverSocket.recv(1024).decode("utf-8")
    if response == "/sendingData":
        serverSocket.send(b"/readyForData")
        data = pickle.loads(serverSocket.recv(1024))
        if data == set():
            print("No pending requests.")
        else:
            print("Pending Requests:")
            for element in data:
                print(element)
    else:
        print(response)
```

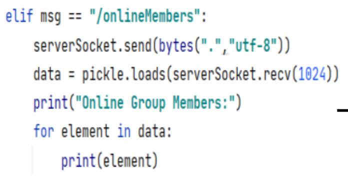
This is our Second Component 'View requests'. (fig 2)

Fig 2



```
Type anything else to send a message
/1
Pending Requests:
sandeep
```

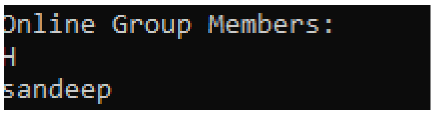
Following onto the next component which was 'online Members', is a very good component because if someone in that chat wants to speak to their friend you can use 'View Online Members' by using that you would know if that person were online or not. Plus, it shows you every member online.



```
elif msg == "/onlineMembers":
    serverSocket.send(bytes(".","utf-8"))
    data = pickle.loads(serverSocket.recv(1024))
    print("Online Group Members:")
    for element in data:
        print(element)
```

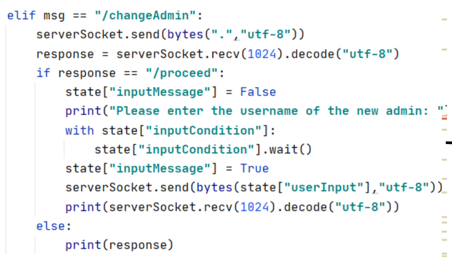
In this picture we have the code 'online members' (fig 3)

Fig 3



```
Online Group Members:
H
sandeep
```

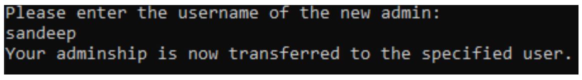
The fourth component 'Change Admin' is another important component, for instance if the current admin is busy or about to go offline, he/she simply give the admin to someone appropriate for the role. This was more appropriate because if the admin goes inactive and the admin goes to someone not suitable for the role can mess up the chatroom. However, this component that admin can make the judgement to give the role to someone suitable and he/she can do this by 'Online Members' and select you person from the list.



```
elif msg == "/changeAdmin":
    serverSocket.send(bytes(".","utf-8"))
    response = serverSocket.recv(1024).decode("utf-8")
    if response == "/proceed":
        state["inputMessage"] = False
        print("Please enter the username of the new admin: "
        with state["inputCondition"]:
            state["inputCondition"].wait()
        state["inputMessage"] = True
        serverSocket.send(bytes(state["userInput"],"utf-8"))
        print(serverSocket.recv(1024).decode("utf-8"))
    else:
        print(response)
```

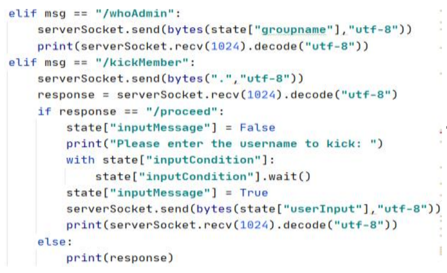
Fig 4 Contains the code for 'change admin'.

Fig 4



```
Please enter the username of the new admin:
sandeep
Your adminship is now transferred to the specified user.
```

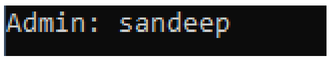
Continuing onto the fifth component, we have 'who Admin', this component is very beneficial for the people in the chats because if you have a person in the chat rooms that you do not want to be there, you can run the who you can make a request to the admin to give him/her a warning or kick/remove the member from the Chat.



```
elif msg == "/whoAdmin":
    serverSocket.send(bytes(state["groupname"],"utf-8"))
    print(serverSocket.recv(1024).decode("utf-8"))
elif msg == "/kickMember":
    serverSocket.send(bytes(".","utf-8"))
    response = serverSocket.recv(1024).decode("utf-8")
    if response == "/proceed":
        state["inputMessage"] = False
        print("Please enter the username to kick: ")
        with state["inputCondition"]:
            state["inputCondition"].wait()
        state["inputMessage"] = True
        serverSocket.send(bytes(state["userInput"],"utf-8"))
        print(serverSocket.recv(1024).decode("utf-8"))
    else:
        print(response)
```

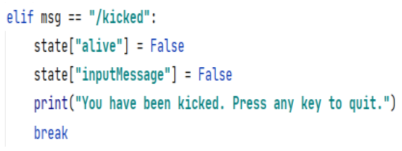
Fig 5 contains the code for 'who Admin'.

Fig 5



```
Admin: sandeep
```

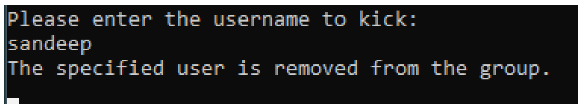
Finally, the last component we used is 'Kicked, previously talking about 'who Admin', this component would come hand in hand to help you remove a person from the chat or group, if you think this person is not acting accordingly.



```
elif msg == "/kicked":
    state["alive"] = False
    state["inputMessage"] = False
    print("You have been kicked. Press any key to quit.")
    break
```

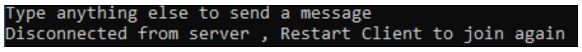
Code contains the 'kicked' command. Fig 6

Fig 6



```
Please enter the username to kick:
sandeep
The specified user is removed from the group.
```

Fig 7

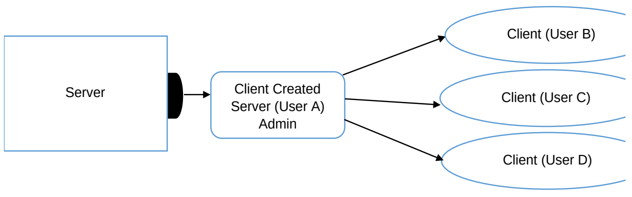


```
Type anything else to send a message
Disconnected from server , Restart Client to join again
```

Moreover, having spoken about all the components which is inside just two classes and in our case the client and server class, this was very organised for us because we did not need to make classes for each component and then linking it back to the server and client class. These were interlinked and each component has the capability to work without the other components, but the other components worked well together, whilst these components worked together so well, we got little to no errors whilst running the server and client with multiple clients. In our project we decided to make a command line interface instead of, a Server and Client with a Graphical User Interface. In addition, in our project we had two classes server and client and in the class server, we had multiple components, such as

- CTRL + C -Disconnect from server
- /1 – Approve Request.
- /2 – View Requests
- /3 – online members
- /4 – Change admin
- /5 – who admin
- /6 – Kicked.

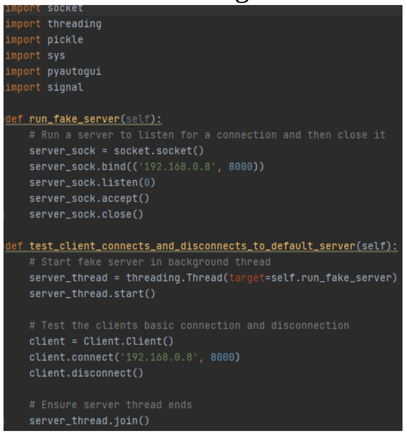
As you can see above these are the components we used for our code and these components are inside the server and client classes. Furthermore, we have a function when you click 'ctrl+c' this disconnects you from the server and this is in both the server and client class.



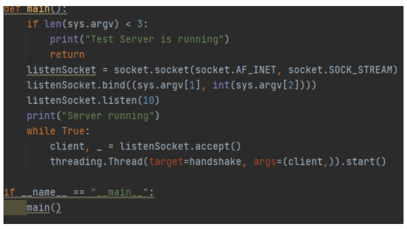
UNIT TESTING

To ensure that our server runs correctly we will be conducting test. The test that we are running is unit testing which check a part of our program. To ensure that our server runs correctly we will be conducting test. The test that we are running is unit testing which check a part of our program- Unit Testing is a type of software testing where the source code is tested, to ensure that the server and client are running and functioning correctly. Unit Testing is crucial as it will help us fix bugs and will allow us to see what part of the code is not functioning correctly and we can use that to tweak the code and rerun the test to see if it functions correctly (3).
We tested the client by first creating a fake server. The fake server will test if the real server is running by listening for a connection and will close the connection after listening to it (2).

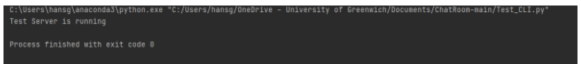
We had to first import multiple code to allow us to use the function. To test the fake server, we have inputted our host address ‘192.168.0.8’ and our port ‘8000’, The client will connect to the host and port and will first run a server to listen for a connection and close it. Then we started a fake server in background thread to test the client’s basic connection and disconnection. The server thread will then end, and it showed that test server is running. -



```python
import socket
import threading
import pickle
import sys
import pyautogui
import signal

def run_fake_server(self):
    # Run a server to listen for a connection and then close it
    server_sock = socket.socket()
    server_sock.bind(('192.168.0.8', 8000))
    server_sock.listen(0)
    server_sock.accept()
    server_sock.close()

def test_client_connects_and_disconnects_to_default_server(self):
    # Start fake server in background thread
    server_thread = threading.Thread(target=self.run_fake_server)
    server_thread.start()

    # Test the clients basic connection and disconnection
    client = Client.Client()
    client.connect('192.168.0.8', 8000)
    client.disconnect()

    # Ensure server thread ends
    server_thread.join()
```



```python
def main():
    if len(sys.argv) < 3:
        print("Test Server is running")
        return
    listenSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    listenSocket.bind((sys.argv[1], int(sys.argv[2])))
    listenSocket.listen(10)
    print("Server running")
    while True:
        client, _ = listenSocket.accept()
        threading.Thread(target=handshake, args=(client,)).start()

if __name__ == "__main__":
    main()
```



```
C:\Users\hansg\anaconda3\python.exe "C:/Users/hansg/OneDrive - University of Greenwich/Documents/ChatRoom-main/Test_CLI.py"
Test Server is running

Process finished with exit code 0
```

FAULT TOLERANCE-

Furthermore, to validate our systems integrity we need to ensure that in the case of system failure or any events that prevent the system from running successfully, counter measures must be set in place. Therefore, we are creating a fault-tolerant system to prevent issues ensuring high availability of our servers.

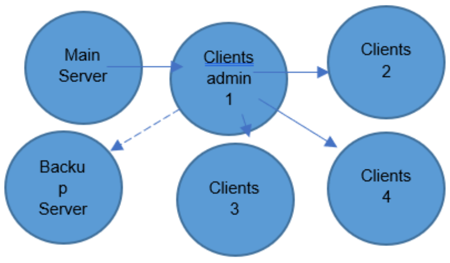
These measures will only take place in the case of a failure of a component ensuring no loss of service-

The components that we are going to be looking at are:

Hardware systems: To ensure that our server is efficiently running we are going to create an identical server in a backup device such as a USB as it is a portable server, this is adequate and extremely cost efficient therefore, we have created a backup server in a separate device.

Power Source: This is a big issue that is generally more expensive to cover however, our design for our system counters this problem as we do not need one power source to power up the system, the system can be powered portable on any device that contains the server program.

The next issue that we as a group must tackle is high availability this minimizes the downtime to ensure that server keeps running.



The above diagrams depict the mains server connection however, as you can see the backup server can also be running in case of any event so that the 1st user can connect to the backup server so they can join back and continue the conversation.

III. Analysis and Critical Discussion

Thousands of data are at risk of getting leaked daily through malware and other sources of data breaches. In addition to this, there are a lot of scams that occur on popular chatting apps like 'WhatsApp' and 'Facebook' which increases the risk their users must face. On the other hand, our program prevents this from happening as it provides reliability and assurance through its features.

A major international corporation 'BBC', (British Broadcasting Corporation) confirmed that 'Whatsapp', was used to install spyware on their clients' phones. '400 million' users in India had their data compromised at was at risk. (6).

Our program ensures that no spyware can affect it due to the user having easy access and full control to the program whenever they desire. The data that is created throughout the communication will be private that will not leak and not be compromised. Furthermore, one of the features makes it so that when the first user joins, they get administrator ensuring full control to their communication. This will allow for the admin to control who joins the chat and what can be added into it. This is very vital to our program as it prevents from bots to join along with spam and scam messages to be being prohibited.

In addition, the admin has the most authority to keep the chat in check and decide what goes on in the chat. We also have a feature 'kick member' so that if the server is compromised, the admin can kick that member as they do not want them in the chat.

Majority of the Client/Servers are online, connected to the internet where you are vulnerable. However, in our case this will not happen because our client/server is available to people who have our server/client. In addition, we have created a command 'online members', and if we use this command and see someone in our 'gangChat', we can make a request to the admin to kick the intruder, but the admin should be monitoring the chatrooms.

Furthermore, our potential users will be officials who cannot afford their data leaking as this can put countries to jeopardy therefore, they will require a system built like us. However, why cannot ordinary people who also use similar communication to talk our main aim is to privatise and ensure no data leaks no matter who the user participants are.

Why is Social Network targeted?

One of the major reasons Social Networks is attacked because people share their personal and private pictures and talk about their own secrets to their close friends and family. This alone motivates hackers to carry out attacks, there is sensitive information that can be used to blackmail people and due to that risk only people will end their lives.

However, comparing our server to other server, our is on a private server address which reduces the risk of being hacked into.

Why settle for risk when you can be risk-free using 'gangChat'?

IV. Conclusions

Data breaches will finally not be a problem when communicating with one another. Our newly developed form of communication allows its user to communicate without the sense of unease. In the future, once this form of communication is widely popular more features will be developed to ensure its in peak form however we will never abandon its best feature which is to always remain private and can be used wherever and whenever. The future of communication can finally be risk-free.

References

1. DH Web Desk (2021) WhatsApp group chats details leak on Google once again, resolved Deccan Herald - https://www.deccanherald.com/specials/whatsapp-group-chats-details-leak-on-google-once-again-resolved-937724.html
2. NanoDano (2015) Unit Testing TCP Server & Client with Python Dev Dungeon – https://www.devdungeon.com/
3. Anthony Shaw (2018) Getting Started with Testing in Python Real Python - https://realpython.com/python-testing/#automated-vs-manual-testing
4. Mayank Gupta (2017) Data Hiding with JavaScript Module Pattern Medium https://medium.com/technofunnel/data-hiding-with-javascript-module-pattern-62b71520bddd
5. Cory Rylan (2015) JavaScript Module Pattern Basics - https://coryrylan.com/blog/javascript-module-pattern-basics
6. BBC News (2019) Pegasus breach: Will quitting WhatsApp make your phone safer? - https://www.bbc.com/news/world-asia-india-50285350
7. Harsh S (2015) Implement TCP Server and Client Using Python Socket Class Learn Programming and Software - https://www.techbeamers.com/python-tutorial-write-tcp-server/