COL334 Assignment 2

Rahul Chhabra
2019CS11016

September 2021

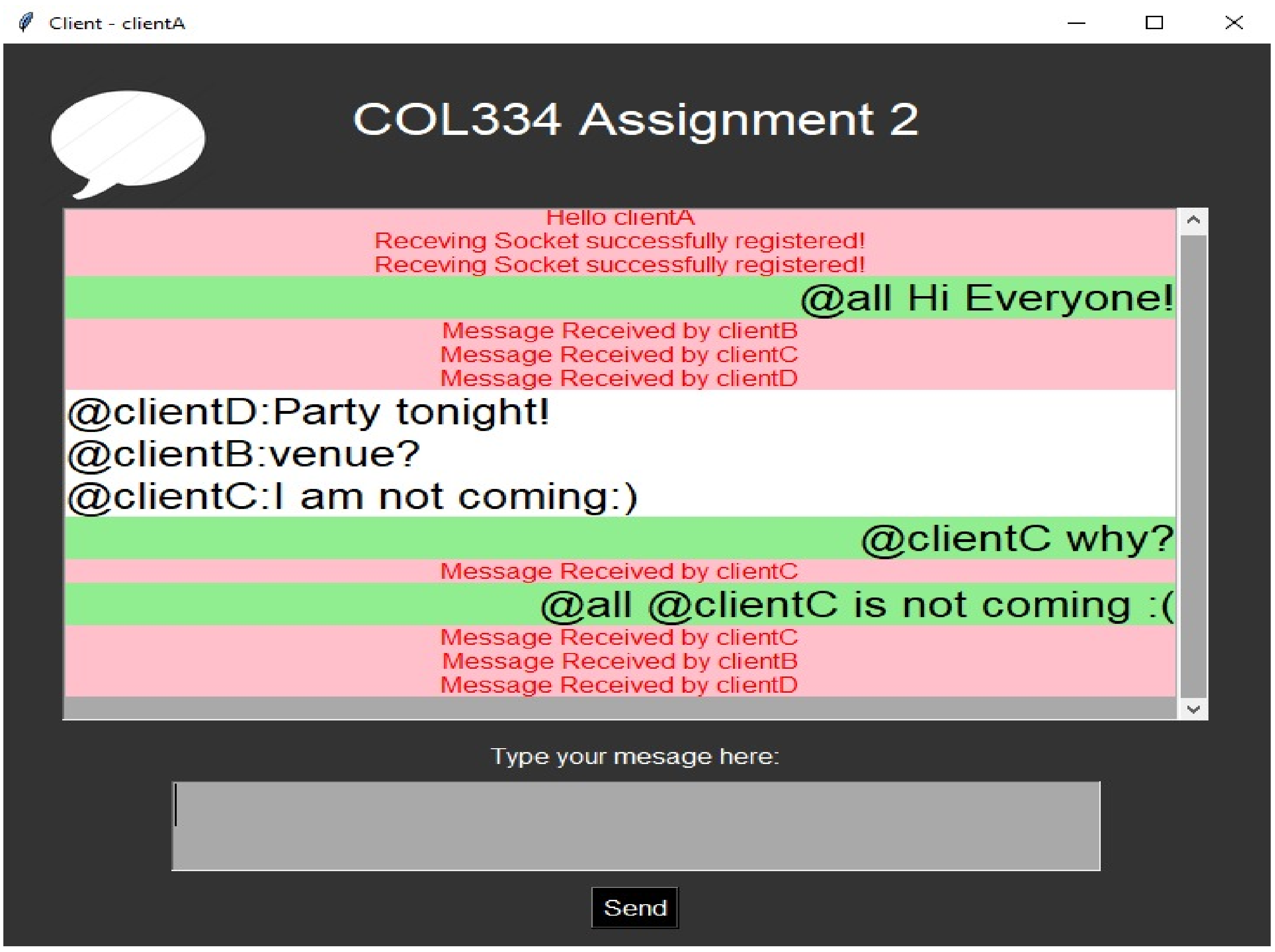
1 Specifications

TCP is used as the carrier of the messages exchanged due to its reliability. I have used python's socket and threading library to create the architecture and tkinter library for GUI . Here is a list of additional features :

1.1 Graphical user interface

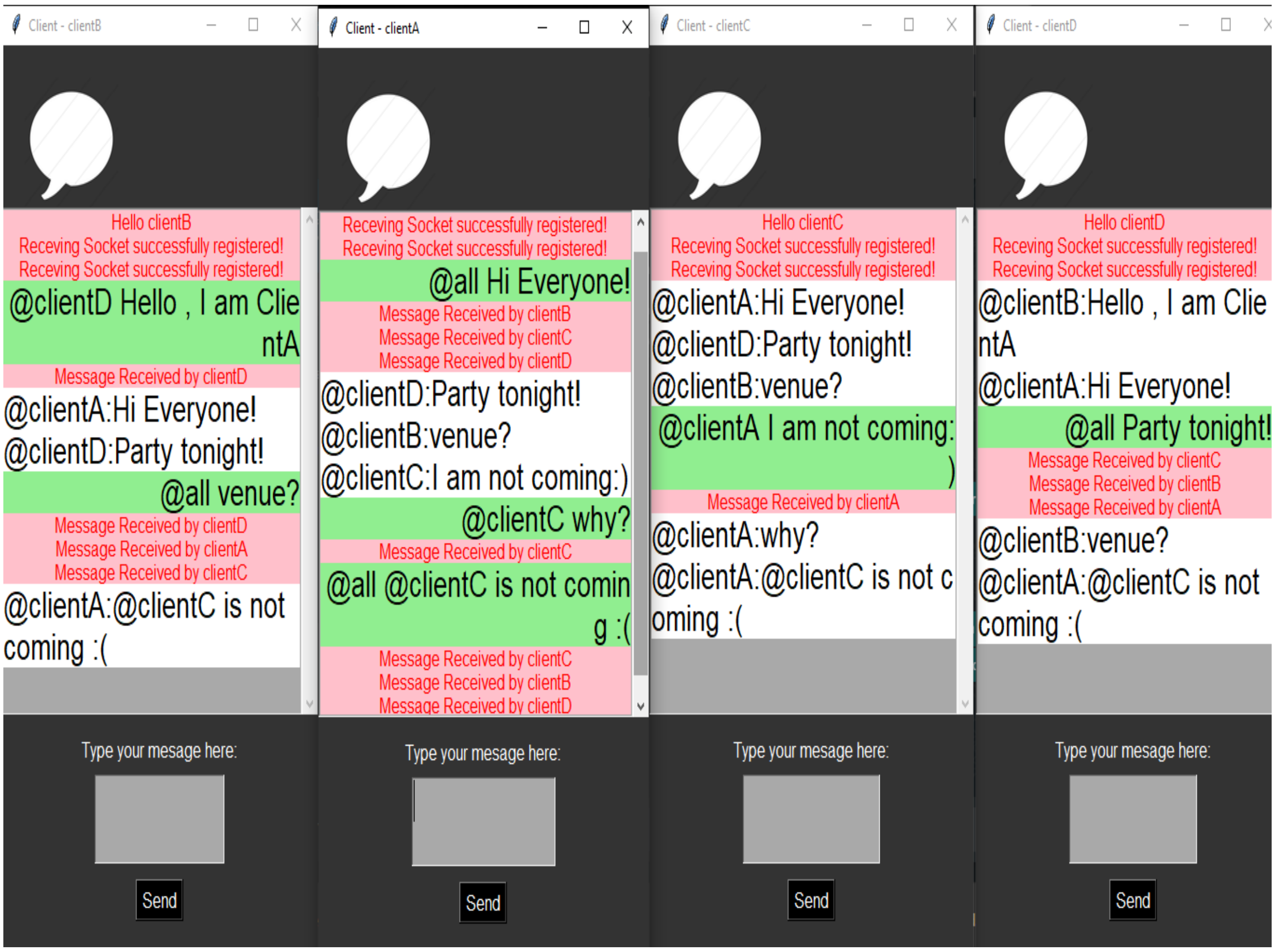
I have used Tkinter library to add a nice GUI to our chat app which has many similiarities with whatsapp . Some features include:

- The sent messages are shown in green colour and are right aligned
- The received messages are shown in white colour and left aligned
- Errors and notifications from server are shown in red colour and small font , they are center aligned.



Hello clientA
Receving Socket successfully registered!
Receving Socket successfully registered!
@all Hi Everyone!
Message Received by clientB
Message Received by clientC
Message Received by clientD
@clientD:Party tonight!
@clientB:venue?
@clientC:I am not coming:)
@clientC why?
Message Received by clientC
@all @clientC is not coming :(
Message Received by clientC
Message Received by clientB
Message Received by clientD

Figure 1: Demonstration of chat application



Client - clientB

Hello clientB
Receving Socket successfully registered!
Receving Socket successfully registered!

@clientD Hello , I am Clie
ntA

Message Received by clientD

@clientA:Hi Everyone!
@clientD:Party tonight!

@all venue?

Message Received by clientD
Message Received by clientA
Message Received by clientC

@clientA:@clientC is not
coming :(

Type your mesage here:

Send

Client - clientA

Receving Socket successfully registered!
Receving Socket successfully registered!

@all Hi Everyone!

Message Received by clientB
Message Received by clientC
Message Received by clientD

@clientD:Party tonight!
@clientB:venue?
@clientC:I am not coming:)

@clientC why?

Message Received by clientC

@all @clientC is not comin
g :(

Message Received by clientC
Message Received by clientB
Message Received by clientD

Type your mesage here:

Send

Client - clientC

Hello clientC
Receving Socket successfully registered!
Receving Socket successfully registered!

@clientA:Hi Everyone!
@clientD:Party tonight!
@clientB:venue?

@clientA I am not coming:
)

Message Received by clientA

@clientA:why?
@clientA:@clientC is not c
oming :(

Type your mesage here:

Send

Client - clientD

Hello clientD
Receving Socket successfully registered!
Receving Socket successfully registered!

@clientB:Hello , I am Clie
ntA
@clientA:Hi Everyone!

@all Party tonight!

Message Received by clientC
Message Received by clientB
Message Received by clientA

@clientB:venue?
@clientA:@clientC is not
coming :(

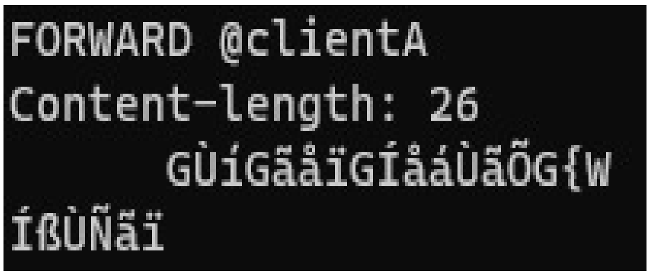
Type your mesage here:

Send

Figure 2: Working application with 4 clients

1.2 Encryption and Decryption of message contents

The message contents are end-to-end-encrypted so that no-one within the network (even server) cannot read our message content. I have used very basic hashing function to demonstrate the encryption decryption feature . The encryption function has two variables which can be used as key for encryption(See figure). This encryption can hide only the contents but there can be one more implementation also . In this encryption there is a end-to-end-encryption between all clients . Anyone in the network analysing the network through tools like wireshark can actually see that who has sent message to whom but cannot tell the message contents wihout knowing the encryption keys but wee can also have a different key for encryption of messages between client and server , this will lead to packet completely encrypted and noone in the network can tell whether who has sent message to whom which can improve privacy application.



```
FORWARD @clientA
Content-length: 26
      GÙíGããïGÍååÙãÕG{W
ÍßÙÑãï
```

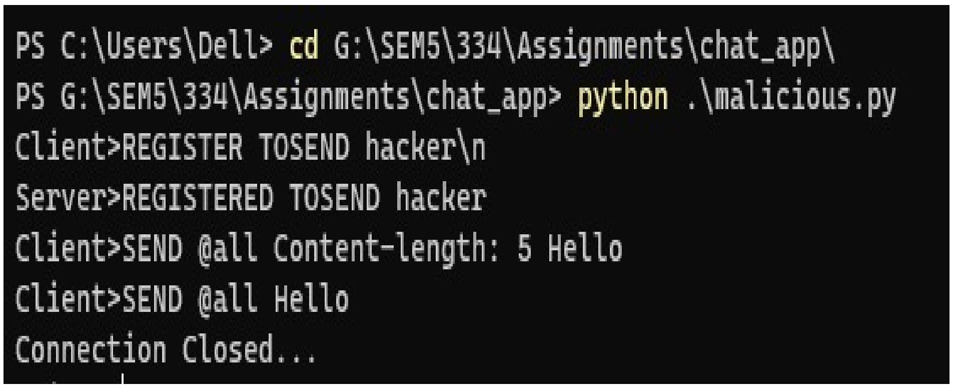
Figure 3: Message contents as observed at server due to encryption : Original message is clientC is not coming:(

1.3 Using minimum resources without blocking the messages

I have used a hybrid approach for broadcasting . In stop and wait approach , the messages are blocked in case some client doesn't send the acknowledgement maybe due to a network failure , the whole server is blocked for a particular client , he cannot receive other data so this is not a very good approach . On the other hand , in case of using parallel threads , it creates problem if there are so many clients , and suppose there is a network failure , many clients don't respond or there is a loss of data due to unexpected connection error , many threads will be running at a time uneccesarily and may cause CPU overheating , So this implementation may case issue in case of large number of clients , I have assumed the client to be a whole system with two sockets where sockets can share data among themselves within the program itself . Instead of waiting or creating uneccesarily new threads , we can make use of the already created receiving thread at the server for each client , I have used this socket to send the acknowledgements and saved resources without blocking the messages. Although I have used two sockets just for simplicty purpose but we can also have single socket implementation with this.

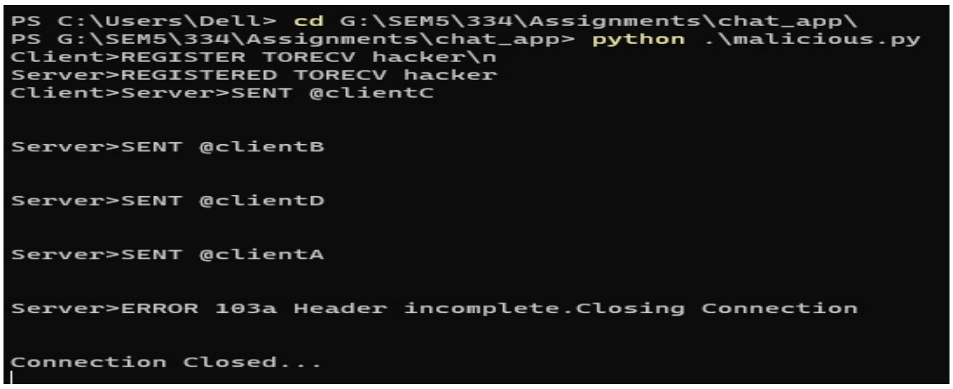
1.4 Malicious clients

I have also included custom client file which can be used to check for the message transfer and acknowledgements in the application . This is developed just for demonstration purpose. Here is a picture of working with malicious client. In malicious client file we can send the actual messages which are transferred in the internal system of this application . Here is a demonstration for that.



```
PS C:\Users\Dell> cd G:\SEM5\334\Assignments\chat_app\
PS G:\SEM5\334\Assignments\chat_app> python .\malicious.py
Client>REGISTER TOSEND hacker\n
Server>REGISTERED TOSEND hacker
Client>SEND @all Content-length: 5 Hello
Client>SEND @all Hello
Connection Closed...
```

Figure 4: Malicious sending socket sending malformed packets



```
PS C:\Users\Dell> cd G:\SEM5\334\Assignments\chat_app\
PS G:\SEM5\334\Assignments\chat_app> python .\malicious.py
Client>REGISTER TORECV hacker\n
Server>REGISTERED TORECV hacker
Client>Server>SENT @clientC

Server>SENT @clientB

Server>SENT @clientD

Server>SENT @clientA

Server>ERROR 103a Header incomplete.Closing Connection

Connection Closed...
```

Figure 5: Responses received at malicious sender's receiving port

1.5 Instructions to run

- To start the server type python ./main_server.py [IP ADDRESS] [PORT NUMBER]
- To start the client type python ./main_client.py [IP ADDRESS] [PORT NUMBER] [USERNAME]
- To start the custom client type python ./malicious.py [IP ADDRESS] [PORT NUMBER]