

Implementation of a chat-based file transfer system using socket programming



Group Members:

Shubhang Gopal, 2K18/IT/115

Sandesh Jain, 2K18/IT/109

Subject: Computer Networks (IT303)

Submitted to: Professor Anamika Chauhan

Date: 24 November 2020

Department of Information Technology, DTU

INDEX

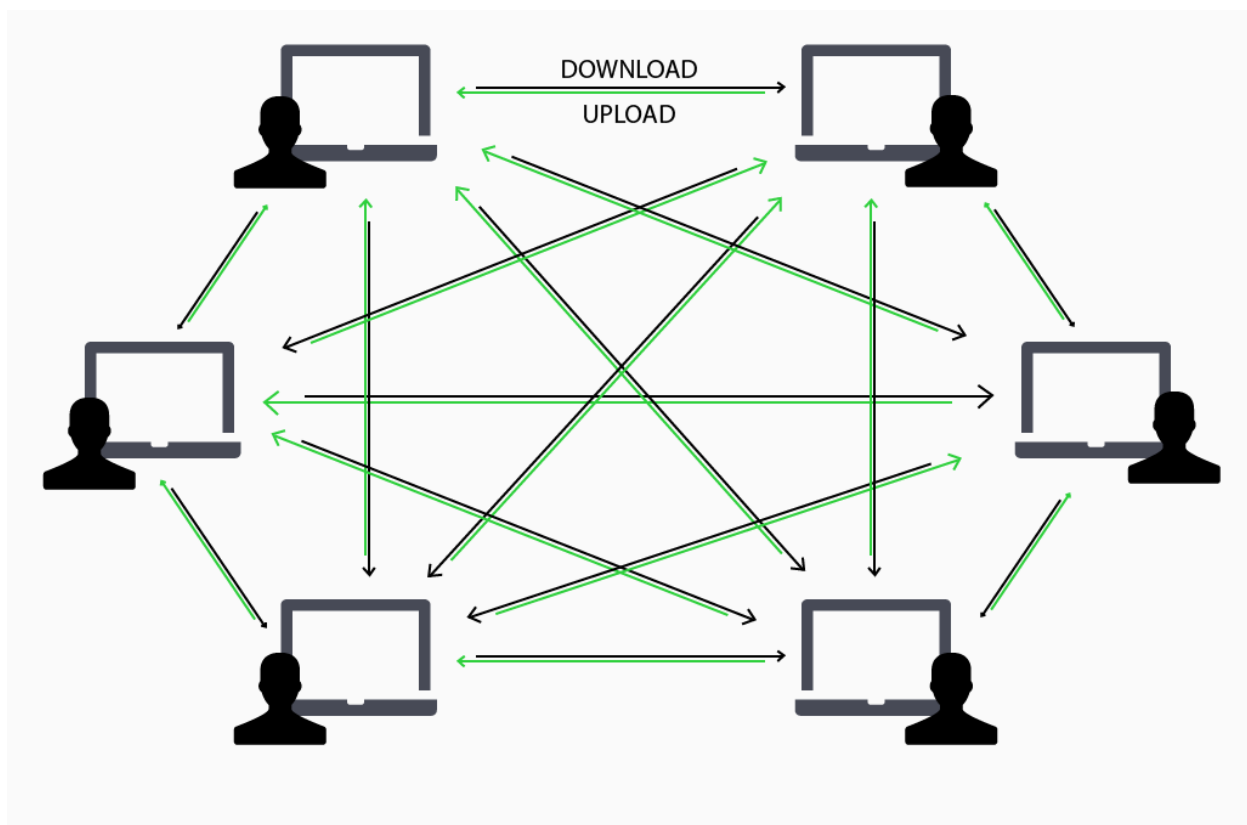
S. No.	Topic	Page No.
I	Introduction	3
II	Theory of Socket Programming	4
III	Project Objectives	5
IV	Proposed Approach	5
V	Implementation Steps	6
VI	Description of Program Files	7
VII	Results and Observations	9
VIII	Unique Features	12
IX	Conclusion	13
X	References	14

I. Introduction

We often come across chatrooms where peers come together in a virtual chat environment and exchange messages with each other. Chatrooms provide an easy-to-use interface for peers to communicate with each other. However, the functionality of a chatroom would be much more if the chat can be used by the peers to send and receive files from other peers. Thus, a chatroom-based communication system which provides file transfer functionality is the ideal utilization of the chatroom environment.

We use several servers and the peer connects to the closest server which allows it to communicate to the peers connected to the same server or to other servers (by hopping). In addition, all peers use a chat room identification code while connecting which is visible only to other peers in the same chat room. User Datagram Protocol (UDP), a transport layer protocol, is used to transfer files so that unnecessary latency during hopping and traffic is avoided. In addition, data link layer protocols such as Selective Repeat and Go Back N are used to ensure reliability.

General Representation of a Peer-to-Peer System



II. Theory of Socket Programming

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols. Sockets are the endpoints of a bidirectional communications channel and they may communicate within a process, between processes on the same machine, or between processes on different continents. The *socket* library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server. They are the real backbones behind web browsing.

A simple socket can be created by importing the socket library and executing the following command-

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Here, AF_INET refers to the address family IPv4. The SOCK_STREAM means connection-oriented TCP protocol.

There are several common socket methods which are as follows-

1. s.recv()- Receives TCP message
2. s.send()- Transmits TCP message
3. s.recvfrom()- Receives UDP message
4. s.sendto()- Transmits UDP message
5. s.close()- Closes socket
6. s.gethostname()- Returns host name
7. s.connect()- Initiates TCP server connection [CLIENT ONLY]
8. s.bind()- Binds address (hostname, port number etc.) to socket [SERVER ONLY]
9. s.listen()- Sets up and starts TCP listener [SERVER ONLY]
10. s.accept()- Accepts TCP client connection, waits until connection arrives [SERVER ONLY]

Sockets may be implemented over a number of different channel types such as unix domain sockets, TCP, UDP etc.

III. Objectives

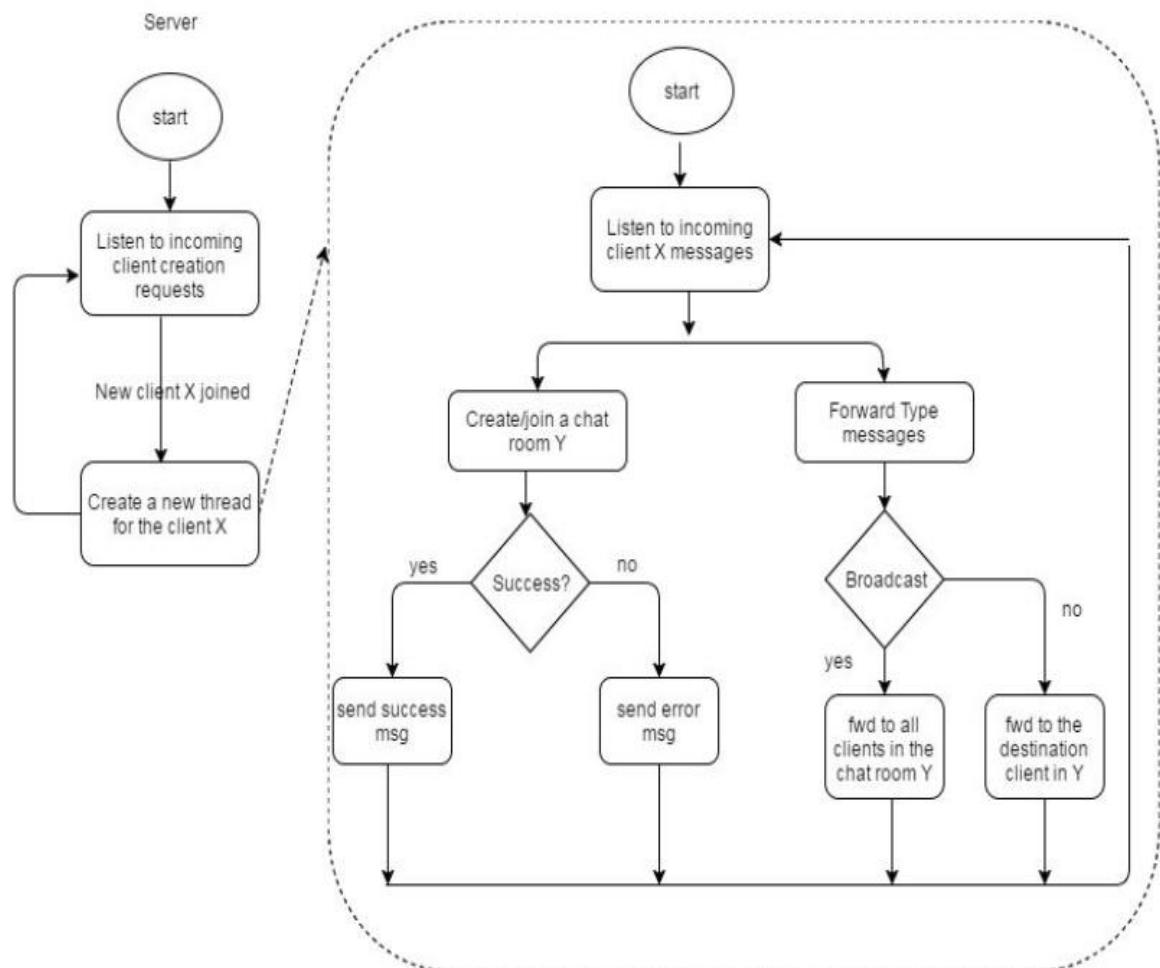
The primary objective of the project is to create a chatroom-based file transfer system, where peers can come together and pass messages as well as transfer files to each other. A client would connect to a server and then decide whether to enter an existing chatroom or a new one. The server would provide information to the client, such as the peers already in the room, clients who join or leave the chatroom, and the messages being sent in the chatroom. In order to leverage the file transfer functionality, a client would ask his/ her peers about who has a particular file, and the clients with the requested file can respond so that the requester can select a peer to obtain the file from. File transfer is done by UDP that does not involve the server, and Go Back N is used at the same time for reliability. Thus, the objectives include-

- Simulating a chatroom with multiple servers and multiple clients.
- Informing every peer whenever another peer enters or leaves the chatroom.
- Allow peers to take requests to transfer files to another peer
- Allow peers to refuse connections when it is already uploading a file at its maximum capacity.
- Allow peers to choose the next peer when a client refuses to entertain a request.

IV. Approach

Socket programming libraries in python are used to implement the project. Separate programs are created for the client and the server and different situations are executed from the client end. The server is always running on pre-assigned TCP ports and multiple clients can connect to the server which handles all the clients in parallel. For file transfer, each client has a folder in his/ her home directory which represents the set of files of the client. When a client wants a file, he/ she send out a broadcast message and the name of the required file, and the peers which have sharing for that file enabled and are willing to do so, can respond. A UDP connection is established and file is shared. The project also tests file sharing for .txt, .pdf, .jpg files.

The interactions between server and clients and the various functionalities available to the client can be depicted by the following flowchart-



V. Implementation Steps

The successful implementation of the project requires the use of inbuilt socket programming libraries in python. After importing the requisite functionalities, several individual programs are created for the working of client, server, udpclient, udpserver, chatroom etc. The implementation can be described in the following steps-

- Ensure that python is installed on all the participating devices, i.e., both the server and the clients, and copy all the python program files into the same directory of the device on which we want to implement file transfer.

- Connect all the participating devices on the same local area network (LAN).
- Use ipconfig command in the command prompt to obtain the IP address of the device which we wish to configure as the server in our working demo. Run server.py program on this device to start the server application.
- After this, run the client by using the following commands-

```
sudo ./client.py -h --display help
```

```
sudo ./client.py -w --run with window size 16
```

The code uses several different subprogram python files. In order to understand the working of the application, it is important to understand about the programs individually.

VI. Description of Program Files

The different program files being used are essential to the smooth working of the project. The functionalities of the different programs are described below. Every program plays an important role in the overall implementation.

server.py

Our program starts with this program as it is used to implement the TCP server to which clients connect to enter/ create a chatroom and start their operations. The Server class is used to represent all server related information including a list of client objects instantiated by the server. On deleting objects, their thread attribute is suspended so that they can reconnect at some point, if required. Python's dictionary data structure is used, and the program contains the functions to remove clients, list chatrooms, listen and broadcast information, and get inputs from users. The main thread of the server is execute(), as when a new client connects, it starts execution from a new thread , thereby allowing parallel clients. This multithreading capability improves the performance of the server, and the server binds to ports. Binding to a port can fail if the port is already open or in use by another connection).

client.py

This program implements the peer-to-peer file transfer on the client side, and is run on the device on which we wish to implement the file transfer. The program contains the Client class which has all the information associated with the peer, and also contains several functions such as checking folder for file, acknowledging file request queries, and checking the transfer of file from the sender to the receiver (via the execute() thread). The UDP connection is initiated by calling the udpclient.py function.

udpclient.py

This program contains the functions used by the client requesting the file in order to respond to the UDP file transfer connection coming from the udpserver program. It has three function- udp_send(), udp_recv() and execute().

udpserver.py

It contains the functions used by the peer which sends the file to the requester after the UDP connection has been initiated. It has nine functions, and udp_send(), udp_recv() and execute() are common to both udpclient.py and udpserver.py. Other functions include get_index(), connect() etc.

chatroom.py

This program holds the information associated with the chatroom. Chatrooms are identified by an ID, which is generated based on the server's room count history and is random in nature. The client which creates the room is automatically a member of that chatroom. This client, who creates the chatroom, can also specify the nature of the chatroom (public or private) by setting a password. This program holds functions relevant to the chatroom, such as removing clients, getting client list, getting peer information etc.

clientnode.py

This python file contains the client related information in a clientnode class, and is initialized with the IP address of the client and the socket used to communicate with the client. The program has several important functionalities, such as connecting to a server, creating/ joining a chatroom, sending messages to server etc. A central menu helps select between various options which allows the user to customize the configurations of the implementation. These options include help, share, ip, port etc.

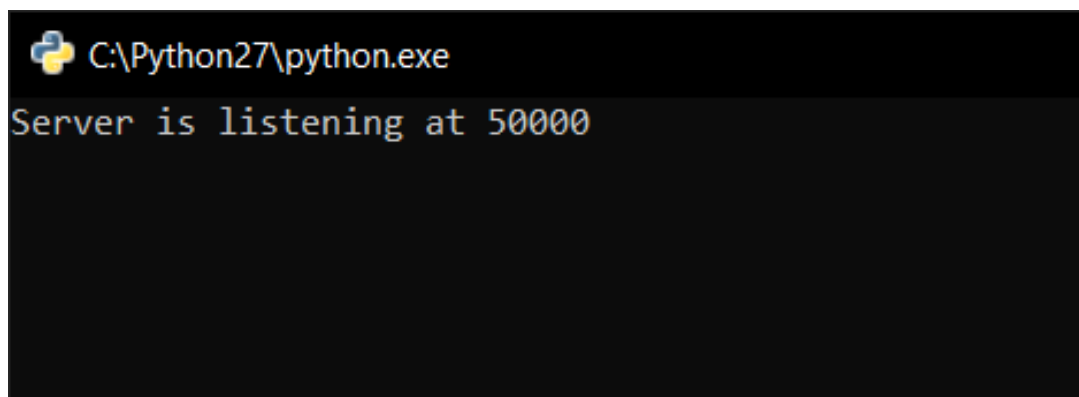
library.py

This is the most general program file which contains functions such as `client_send`, `client_receive`, `send_data`, `send_err` etc. Common library program is created to allow the implementation of these general functions by reusing the code and not having separate files for the client and the server. Different function calls perform different functions when called.

In addition, the server also has the ability to act as the admin and manage clients by removing misbehaving clients etc. Server can also broadcast messages to peers. The creator of the chatroom can include a password, as discussed before, to increase the security of the chatroom. Restrictions can also be applied on the file sharing to disable sharing of illicit content. Variable window size can also be configured and the number of threads and parallel connections can also be controlled.

VII. Results

Following the steps mentioned above, we successfully execute our chat-based file transfer system. We run the server at port number 50000 by binding it. The server enters listen mode as shown below.

A screenshot of a Windows command prompt window. The title bar at the top shows a yellow icon and the text "C:\Python27\python.exe". The command prompt itself has a dark background with light blue text. The first line of text is "Server is listening at 50000".

```
C:\Python27\python.exe
Server is listening at 50000
```

After this, on the client system, we run the `client.py` code and get it connected to the server. The client is then asked to enter their username and then given the option to either join or create a chatroom. In the snapshots below, we implement a model with two clients and one server. Client Shubhang enters the chatroom named 'whatsapp', in which peer Sandesh is already present.

Peer Shubhang's view

```
C:\Python27\python.exe
Connected to 192.168.1.10 at port 50000
Please enter your username
shubhang
OK|Username shubhang accepted. Send create/join a chatroom
join
OK|Here is a list of chatrooms you can join.
whatsapp
whatsapp
OK|This chatroom is private. Please enter password.
1234
OK|Password accepted.
OK|You have joined chatroom - whatsapp
Here is a list of peers in the room:
sandesh
Welcome to whatsapp. You are all set to pass messages
```

Peer Sandesh's view

```
C:\Python27\python.exe
Connected to 192.168.1.10 at port 50000
Please enter your username
sandesh
OK|Username sandesh accepted. Send create/join a chatroom
create
OK|Specify a chatroom name to create.
whatsapp
OK|Chatroom whatsapp created. Do you want to password protect the room? [yes/no]
yes
OK|Please enter password.
1234
OK|Password 1234 accepted.
Welcome to whatsapp. You are all set to pass messages
INFO| New user shubhang has joined
-
```

Once both the peers are in the chatroom, we test our program by sending some messages and requesting a file. The file transfer functionality is verified by the successful transfer of a file to our folder. The snapshot for this is as follows-

```
C:\Python27\python.exe
Connected to 192.168.1.10 at port 50000
Please enter your username
shubhang
OK|Username shubhang accepted. Send create/join a chatroom
join
OK|Here is a list of chatrooms you can join.
whatsapp
whatsapp
OK|This chatroom is private. Please enter password.
1234
OK|Password accepted.
OK|You have joined chatroom - whatsapp
Here is a list of peers in the room:
sandesh
Welcome to whatsapp. You are all set to pass messages
@sandesh|hi
@shubhang|hello!
#me|hello!
@sandesh|hello!
@sandesh|getfile|text
OK| Sending file on port 49325 from |192.168.1.10|42042
192.168.1.10 42042
1605431561.47 1605431561.43
Completed file transfer in 0.0329999923706 seconds.
```

Several commands are available to the peers in the chatroom. The list of commands available to the clients are-

Command	Description
@username chat	Sends a message 'chat' to 'username'
@all chat	Sends a message 'chat' to all users in chatroom
@server chat	Sends a message 'chat' to server (admin)
@all whohas file	Sends broadcast message to see who has file with filename 'file'
@user getfile file	Sends a message to 'user' to start UDP peer-to-peer file transfer
@server get_rooms	Get a list of chat rooms available with server
@server get_peers	Get a list of connected peers in chatroom
@server exit	Clean exit chat room
@me setshare file	Enable sharing of filename: 'file'

@me clrshare file	Disable sharing of filename: 'file'
@me setglobalshare	Enable sharing all files
@me clrglobalshare	Disable sharing any files
@me getsharestatus	Get global share status, as well as individual file share status

Similarly, several commands are available to the server as well. These commands include-

Command	Description
@username chat	Sends a message 'chat' to 'username'
@all chat	Sends a message 'chat' to all users in all chatroom
@server chat	Sends a message 'chat' to self
exit	Kill all client connections and exit application gracefully

VIII. Unique Features

In today's world, there exist several chatbots which perform the basic functionality of connecting peers and allowing file transfer. In order to differentiate from the market, we tried to focus more on a key aspect of chatbots- security. There have been several instances of security breaches in chatrooms where information has reached unwanted hands. With an aim to integrate cryptography and network security with socket programming, we created a modified version of RSA algorithm to work along with the project.

The Rivest-Shamir-Adleman algorithm is a cryptosystem technique used to ensure secure data transmission. In usual cryptosystem techniques, the encryption key is kept public, while the decryption key is kept private, so that only the intended recipient can decode the message. In RSA algorithm, the public encryption key is based on two large prime numbers and an auxiliary value, such that the prime numbers are kept secret or hidden. This

allows anyone to encrypt the messages, but only those with the knowledge of the prime numbers can decrypt it. This can help improve the security of the files transferred.

After reading some general research papers, we made some changes in RSA algorithm from our side to increase the security.

- For derivation of public key, we choose a number which is coprime with ϕ .
- For derivation of private key, we took multiplicative inverse of our public key.
- Now, in our improved version we broke the public key in two parts as $y=e*x$ and we will send them differently in the file as $\{n,y\}$ and $\{x\}$.

Comparison

RSA	Improved RSA
It uses one public key.	It uses two public keys.
More vulnerable to brute force attack.	Less vulnerable to brute force attack.
It is less secure	It is more secure.

In addition to this, we also aim to potentially innovate our chatroom system, by attempting to include features such as multiple language support by language recognition, adding customization options for peers and allowing even greater file types.

IX. Conclusion

The aim of the project was to build a chatroom-based communication and file transfer application using socket programming, in which multiple peers can connect with each other, send messages and share files. UDP protocol was used to achieve file transfer capability and a small set of commands was used to request for and send files. Several small python programs were written to achieve modularity and segregate the functioning of different aspects of the system. Finally, after writing the code and following the steps

of implementation, the application was tested for two clients and one server by sending messages and transferring a file. The positive results of the operations carried out show the success of the system and prove its working ability.

X. References

<https://www.geeksforgeeks.org/socket-programming-python/>

https://www.tutorialspoint.com/python/python_networking.htm

<https://www.geeksforgeeks.org/simple-chat-room-using-python/>

<https://pythonprogramming.net/server-chatroom-sockets-tutorial-python-3/>

<https://stackoverflow.com/questions/7749341/basic-python-client-socket-example>