Toph Chat ©

Version: 0.0.1

Authors:

Abel Jimenez

Justin Lee

Aung Thu

Khoi Trinh

Jason Duong

Affiliated with: University of California, Irvine

Table of Contents

Glossary

Client: hardware or application that connects to a server.

Decryption: the process of converting encrypted cipher into data.

Encryption: the process of converting information into a cipher to prevent unauthorized access of it.

FIFO: first-in, first-out.

Server: the host application/hardware that clients connect to and communicate with.

Socket: endpoint of two-way communication between client and server, defined by port number and IP address.

1. Client Software Architecture Overview

1.1 Main Client Data Types and Structures

struct sockaddr_in

- Holds the data for the internet address used for the sockets for the clients

struct serverConnection

- Stores the socket number that the server and client connects to
- Cookies information for saving password and user information

struct room

- Stores serverConnection struct, room number of client, and FIFObuffer struct.
- The client-side rooms have the information of which clients are connected to which rooms and the messages stored in each room.

struct messageClient

- Stores the message sent by the server and the room number it is sent to.

struct FIFObuffer

- Whenever a client sends a message, it is sent to this buffer at the end of the buffer.
- A FIFO buffer is used to display the messages between the clients in chronological order.

1.2 Major Client Software Components

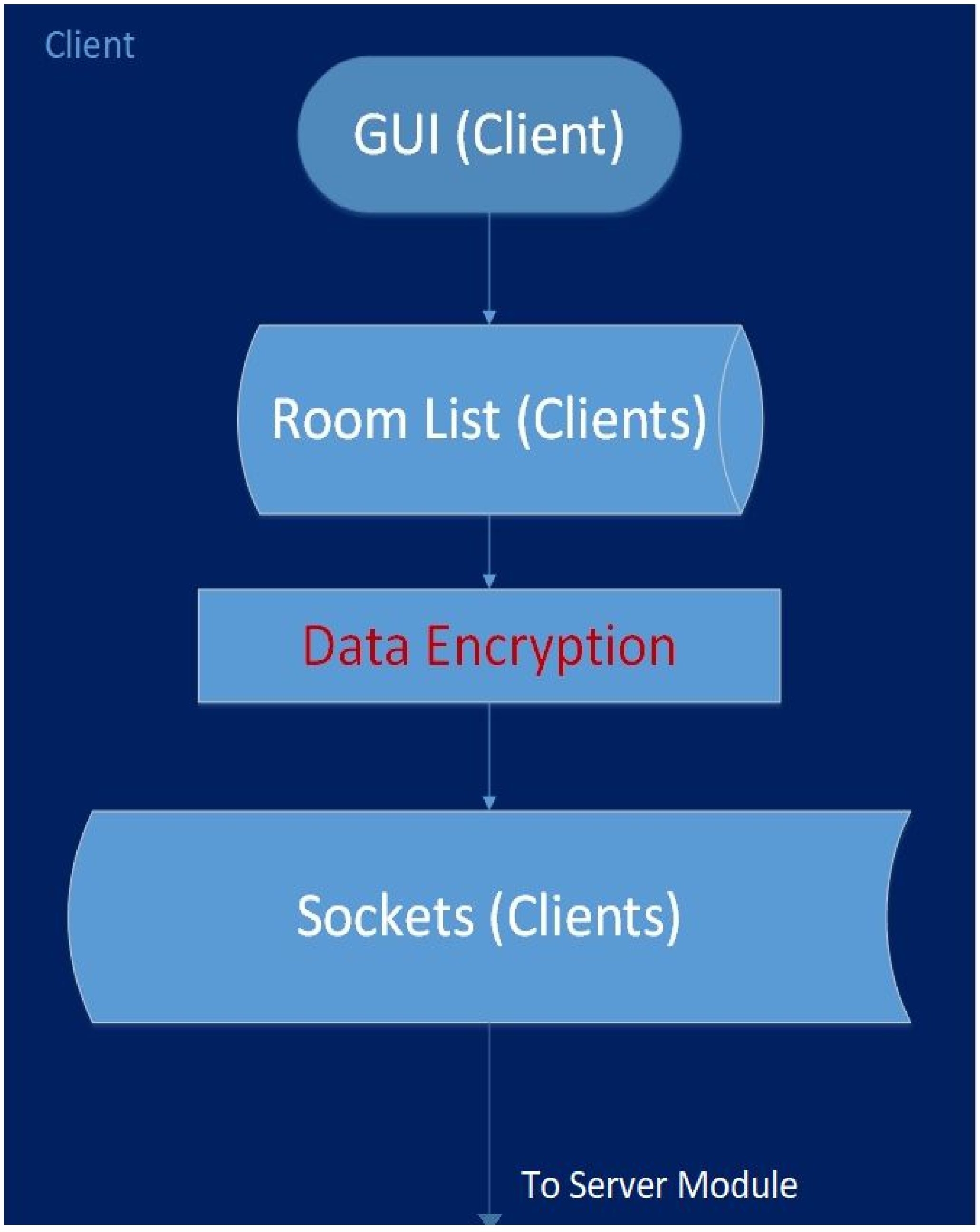

Figure 1 (Client Module Flow): The client is presented with the GUI and can interact with it. If the user sends a message, the data is sent to the Room List which contains all the information for each chat room. From there, the data is then encrypted and taken into the client side sockets to be sent to the server module.

1.3 Client Module Interfaces

Client GUI Module

Input: username, password, message, room selection, friend invites.
Output: messages sent and received in the room, friend list.
Description: A client first logs in with username and password. Each client has its own GUI and can connect to a room. Multiple clients can connect to a single room. Clients can send messages to the connected room.

Room List Module
Input: room number, input messages from clients.
Output: data packets to and from server sockets.
Description: Contains all the chat rooms and list of users that can access each. individual room. Handles the compilation of all user messages and passes it to the client GUI display or server depending on which direction the messages are going.

Encryption Module
Input: password, messages.
Output: encrypted data to protect user information.
Description: Encrypts the data before it is sent to server.

Sockets Module
Input: port number, host name, host IP address.
Output: socket descriptor.
Description: Creates a port for the client and is bound to the address of the server.

1.4 Overall Program Control Flow

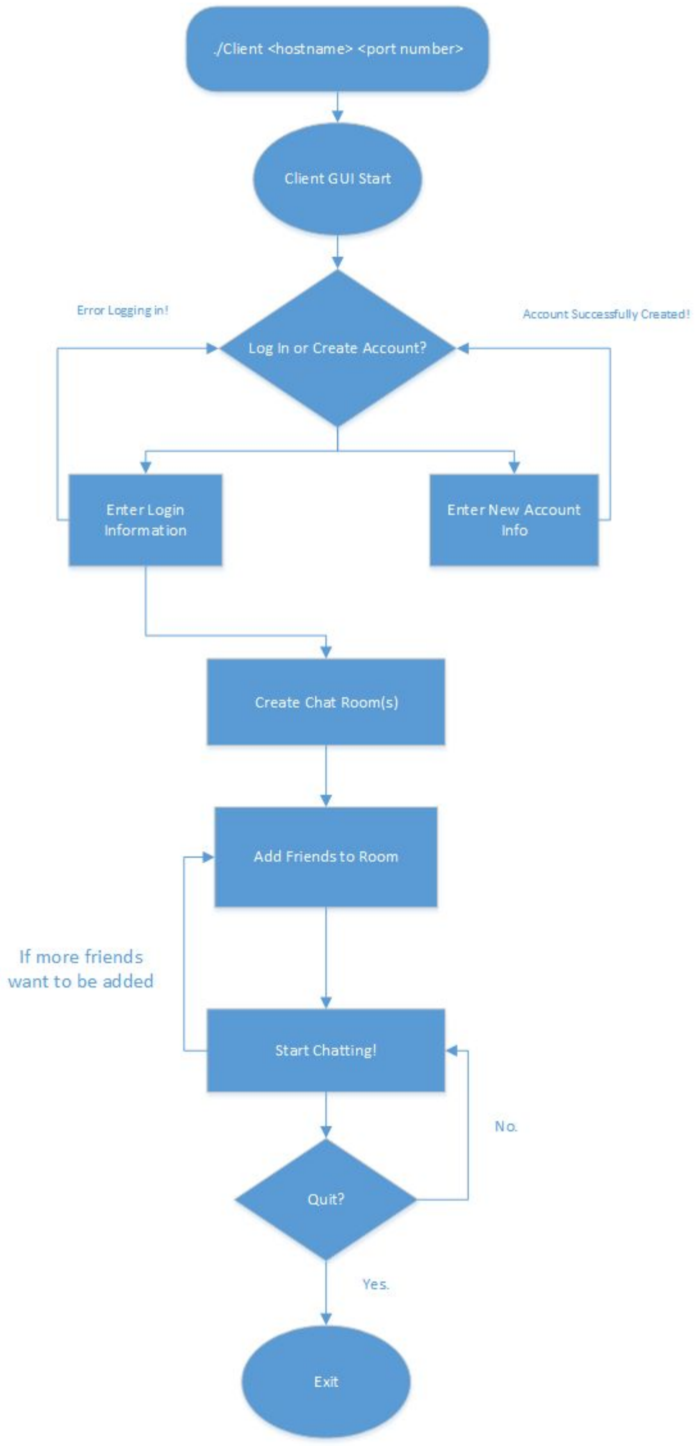

Figure 2 (Client-Side Control Flow): Basic usage of the chat application for the client.

2. Server Software Architecture Overview

2.1 Main Server Data Types and Structures

struct sockaddr_in
- Holds the data for the internet address used for the server socket

struct serverConnection
- Stores the socket number that both the server and client connects to

struct room
- Stores serverConnection struct, room number of client, and FIFObuffer struct.
- The client side room and server side room are not always in sync.
- After a client’s message is received by the server-side room, the server then looks at all of the clients in that room and updates their screen by sending the message to the each respective client.

struct messageClient
- Stores the message sent by the client and the room number it is sent to.

struct FIFObuffer
- Keeps track of messages that come from a room and adds it to the FIFO buffer
- The rooms are updated with the respective messages in the order that the messages first came in

2.2 Major Server Software Components

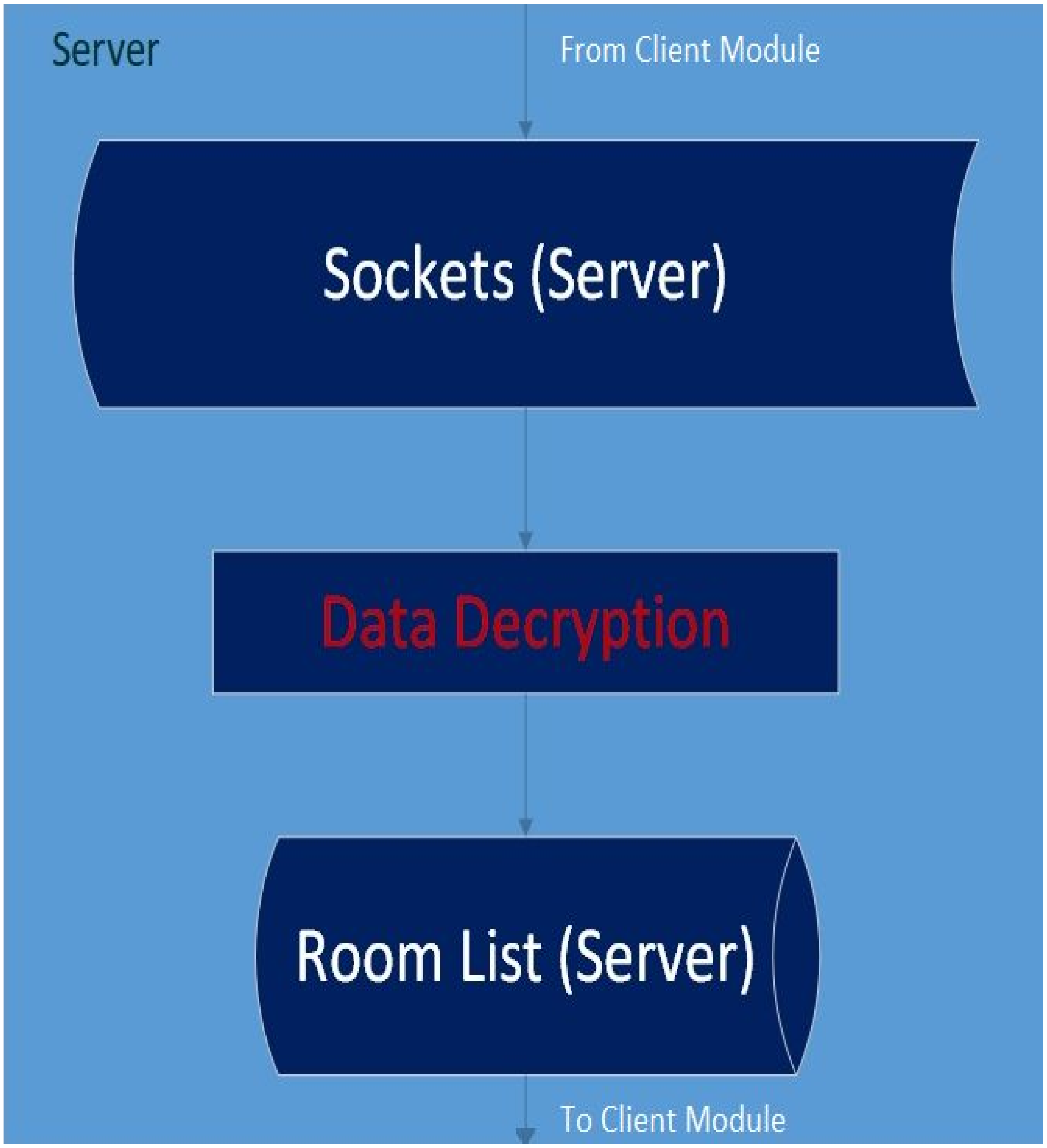

Figure 3 (Server Module Flow): the data from the client module is received by the server side sockets. From there, the data is decrypted and sent to the server side Room List. The server side room list updates the room that is being altered and sends the updated room data back to the client side room list to update all of the clients connected to each room.

2.3 Server Module Interfaces

Server GUI Module
Input: nothing
Output: room list
Description: Displays the clients connected to the server and the rooms they are connected to.

Decryption Module
Input: room number, encrypted account information, and messages.
Output: decrypted messages for parsing by server.
Description: Server takes in encrypted data. The account information is stored for later reference. The messages are stored in a message log for each respective room and also decrypted and sent back to the client rooms for display.

Server Module
Input: socket number
Output: data packages sent to clients
Description: binds to the input socket number and listens. Then it uses select to connect with different clients.

2.4 Overall Program Control Flow

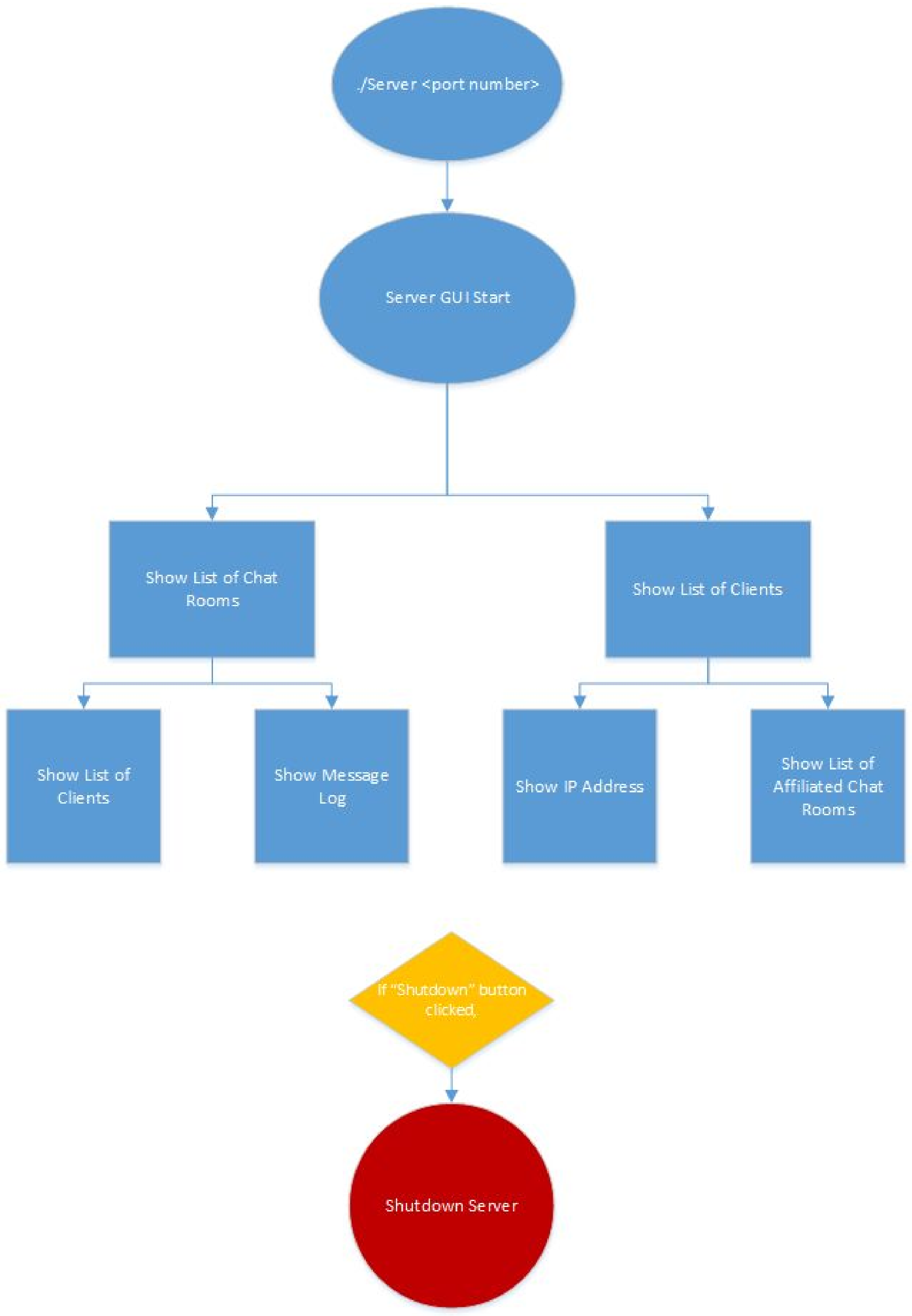

Figure 4 (Server-Side Control Flow): Shows the usage of the server-side GUI.

3. Installation

3.1 System Requirements/Compatibility

- Linux operating system (e.g. RHEL-6-x86_64)
- Gcc version 4.4.7 minimum
- Make utility

3.2 Setup and Configuration

1. Download the tarball
2. Move to *bin* directory
3. Ensure the chat server is online by running command "netstat -pant | grep ":<*port number*>" to see if the port is current being used.

3.3 Building, Compilation, Installation

1. Run the command "make ChatApp"
2. The client side program is now compiled and ready to run
3. Assuming the chat server is online, run "./ChatApp <hostname>< port number>"
4. Once connected, follow the on-screen directions to get to chatting!

4. Documentation of Packages, Modules, Interfaces

4.1 Detailed Description of Data Structures

```
/* holds data on the sockets */
struct sockaddr_in
{
 short     sin_family;          /* keeps track of what address type is used */
 u_short   sin_port;            /* port number for the socket */
 struct    in_addr sin_addr;    /* holds the IP address of the host */
 char      sin_zero[8];         /* Not used, must be zero */
};

/* stores the socket number of connection between client and server */
struct serverConnection
{
   int socket;                  /* socket number client and server both connects to */
```

```
    FILE *cookies_file; // file that stores a private key and undefined number of public key
    int privKey;
    int *pubKey;
    int pubKeyLength;
};

/* stores the messages waiting to be displayed in a room as well as room number */
struct room
{
    struct serverConnection server; /* contain information to connect to server*/
    int roomNum;                    /* room number */
    struct FIFObuffer inMessage;    /* struct that stores the messages sent into room*/
};

/* keeps track of all the rooms in a list */
struct allRoom
{
    int totalRoom;                              /* total number of rooms */
    struct room roomList[CHAT_ROOM_LIMIT];  /* list of rooms */
};

/* array of messages sent to a room in FIFO form */
 struct FIFObuffer
{
    // use union if needing more types of buffer
    char **buffer;      /* array of messages */
    int buffLength;     /* length of array */
    int readPos;        /* index for reading buffer */
    int writePos;       /* index for writing buffer */
};

/* contain information to be sent to the server */
struct PacketClient
{
    char message[MESS_LIMIT];
    uint8_t roomNum;
};
```

4.2 Detailed Description of Functions and Parameters

fifo.h

struct FIFObuffer *initBuffer(int length);

Input: the length of the intended buffer.

Output: the pointer to the FIFO struct.

Description: used to create buffer with customized length.

char *readBuffer(fifo *buffer);

Input: the buffer struct

Output: the character stored at the last position in the buffer

Description: this is used to retrieve the information stored at the end of the fifo and clear the that position for other storage.

int writeBuffer(fifo *buffer, char *writeData, int length);

Input: the data to write to the buffer as well as the target buffer

Output: 0 if successful, 1 otherwise

Description: write to the first position available on the buffer with the given data.

Server.h

// setups the server, with a given port number

int setupServer(int portNum);

Input: the port number.

Output: socket descriptor for listening server.

Description: setup the listening socket.

// close the server

int closeServer(void);

Input: nothing.

Output: 0 if successful, 1 otherwise

Description: close the server, return error code if needed.

tcpGUI.h

/* open connection to chat server, the address and port of the server is defined in macros */

int openConnection(connection *server);

Input: address of server, port number for server.

Output: 1 if successful, 0 if failed.

Description: Opens up communication between server and clients.

// close connection to chat server
int closeConnection(connection *server);
Input: server
Output: 1 if successful, 0 if failed
Description: Blocks off communication between server and the clients

// will need to request a room number from the server
int openRoom(chatRoom *room);
Input: room
Output: the first open room available
Description: Searches for an available room in the room list and returns an open room

// make available the room number so that the server
// can use it for other room
int closeRoom(chatRoom *room);
Input: the pointer to a room structure.
Output: closed room number.
Description: Flushes out the room to make it available to be reused.

// return the next message in the FIFO
char *fetchMessage(chatRoom *room);
Input: the pointer to a room structure to update.
Output: message at the top of the FIFO buffer
Description: Ensures that the messages displayed are in chronological order from when they are sent by using a FIFO buffer. Returns the oldest message that was sent.

// package and send the message both to the input FIFO of the room
// as well as to the output FIFO of the client
int sendMessage(chatRoom *room, char *message);
Input: user-inputted message, room number
Output: 1 for success, 0 for fail
Description: After a user sends a message, it is sent to the room for processing in the FIFO buffer

tcpPacket.h

int sendPacket(clientPacket *packet, struct serverConnection *server);
Input: packet that the client is sending, the destination server.
Output: 1 for success, 0 for failure.
Description: Sends the packet to the server, the packet contains messages and the room number. The port number for the destination server is in the serverConnection struct. It also contains public and private keys for encryption and decryption.

int fetchPacket(clientPacket *packet, struct serverConnection *server);
Input: packet that the client is receiving, and the source server.
Output: 1 for success, and 0 for failure.
Description: Fetches the packet from the source server, the packet contains messages sent to the room and the room number. The port number for the source server is in the serverConnection. It also contains public and private keys for encryption and decryption.

int manageIO(connection *server, fifo *inputFIFO, fifo *outputFIFO);
Input: the connected server, inputFIFO
Output: outputFIFO, first in first out message list
Description: This function process the packets sent to and received from the server. It acts as a mailman between the client and the server.

4.3 Detailed Description of the Communication Protocol

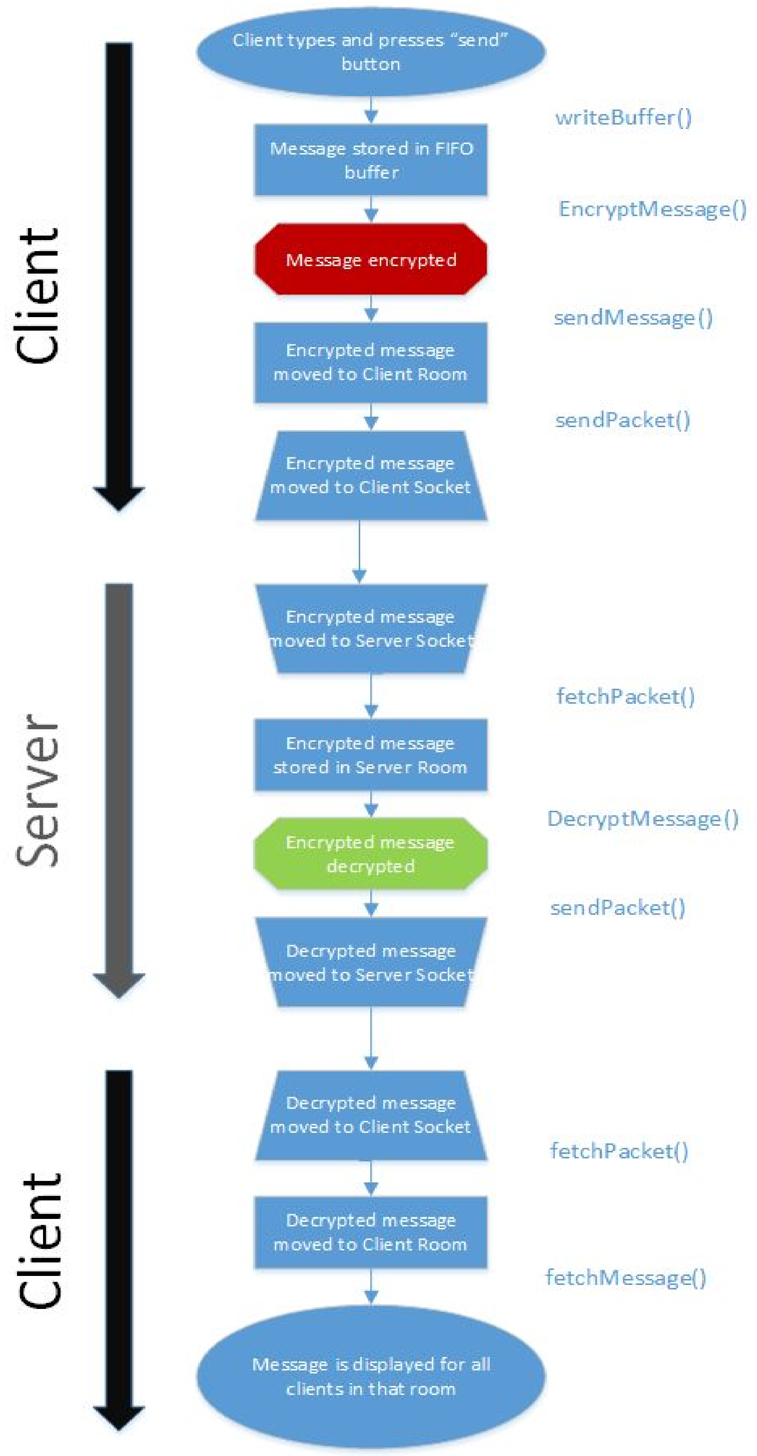
Client
Server
Client
Client types and presses "send" button
writeBuffer()
Message stored in FIFO buffer
EncryptMessage()
Message encrypted
sendMessage()
Encrypted message moved to Client Room
sendPacket()
Encrypted message moved to Client Socket
Encrypted message moved to Server Socket
fetchPacket()
Encrypted message stored in Server Room
DecryptMessage()
Encrypted message decrypted
sendPacket()
Decrypted message moved to Server Socket
Decrypted message moved to Client Socket
fetchPacket()
Decrypted message moved to Client Room
fetchMessage()
Message is displayed for all clients in that room

Figure 5 (Communication Protocol): Shows the Client-Server process of when a client sends a message.

Communication Keyword: Every message received by the server will be parsed and these strings will be looked for to decipher the purpose of the server.

```
#define ID_MESS "1TPM{}H" // identify that this his a message
#define ID_COMM "1TPC{}H" // identify that this is a command

#define FREQUEST "REQ"       // friend request
#define DEREQUEST "DEQ"      // deny friend request
#define DEFRIEND "DEFR"      // delete friends
#define CLOSECOM "CLOSECHAT" // close the communication
#define OPENCOM "OPENCHAT"   // open a new communication

// return value of parse function based on the the the of request
#define VAL_FREQUEST 1  // friend request
#define VAL_DEREQUEST 2 // deny friend request
#define VAL_DEFRIEND 3  // delete friends
#define VAL_CLOSECOM 4  // close the communication
#define VAL_OPENCOM 5   // open a new communication
#define VAL_MESSAGE 0   // just a regular message
```

5. Development Plan and Timeline

5.1 Partitioning of Tasks

- Client-Side Handling
- Server-Side Handling
- Client GUI
- Server GUI
- Data Encryption

5.2 Team Member Responsibilities

Abel Jimenez: Data Encryption
Khoi Trinh: Client-Side Handling
Justin Lee: Server-Side Handling
Aung Myat Thu: Server GUI, RSA-encryption
Jason Duong: Client GUI

Timeline

Week 3 (Alpha Release):
- Server and a single Client can connect
- Data can be sent back and forth between the Server and Client
- Basic GUI with minimum functionalities created

Week 4 (Beta Release):
- Multiple Clients can connect to the Server
- Security check for which Clients are able to connect to Server
- GUI with extra features added

Week 5 (Final Release):
- All extra features implemented
- GUI fully integrated with the chat application

Back Matter

References

Linux sockets: http://www.linuxhowtos.org/C_C++/socket.htm
GTK+ 2.0: https://developer.gnome.org/gtk-tutorial/stable/
RSA encryption: https://www.di-mgt.com.au/rsa_alg.html

Index