Secure Net Messenger

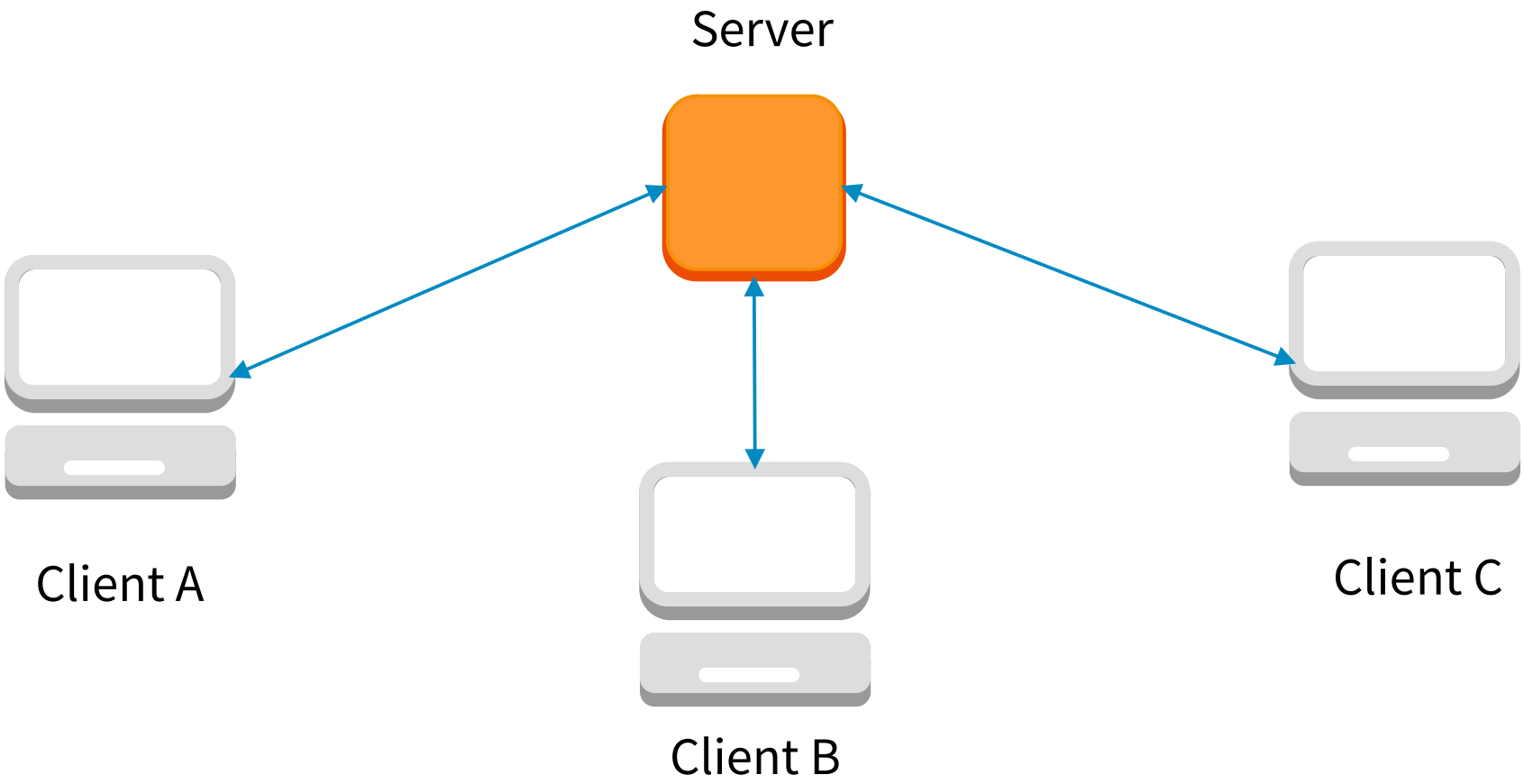
A TCP based Desktop Chatting Application

Project Members & Guide

- Ankit Makwana (206400307085)
- Harmik Sarvaliya (206400307087)
- Vivek Gokhale (206400307090)
- Darshan Patel (206400307103)

- Project Guide: H.V. Katpara

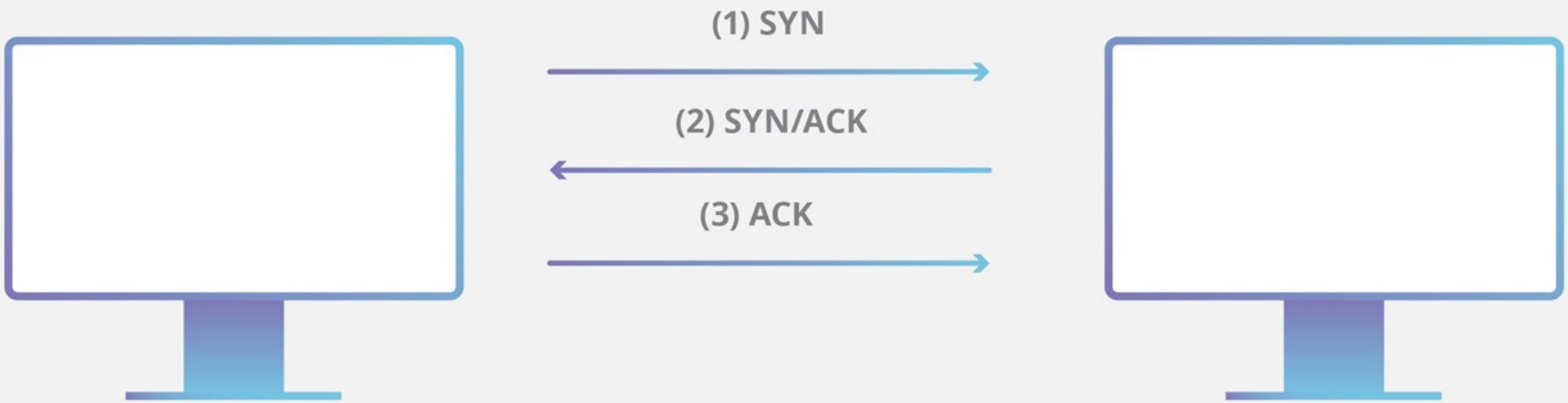
Concept of a group chat based on TCP



What is TCP?

- The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite, often simply referred to as TCP/IP.
- Using TCP, applications on networked hosts can create connections to one another, over which they can exchange streams of data using Stream Sockets.

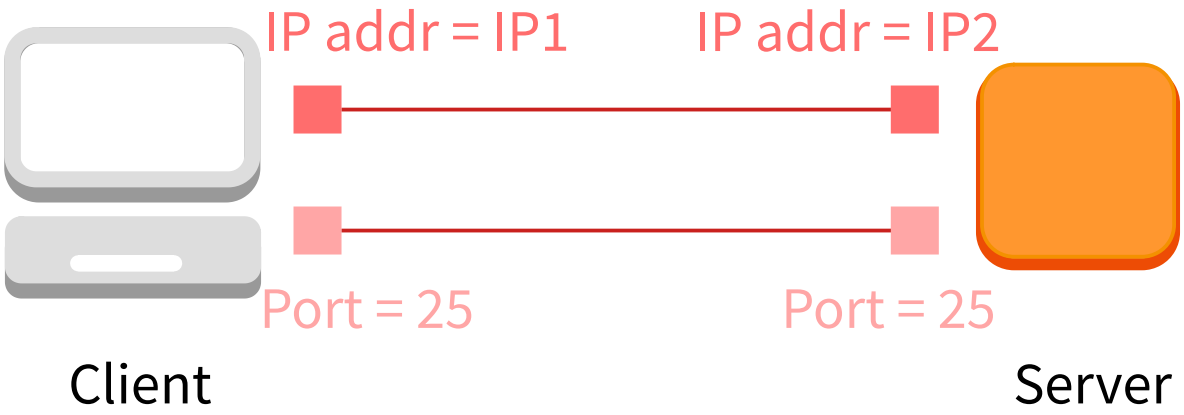
THREE - WAY HANDSHAKE (TCP)



SYN = SYNCHRONIZATION

ACK = ACKNOWLEDGEMENT

Socket-based TCP communication



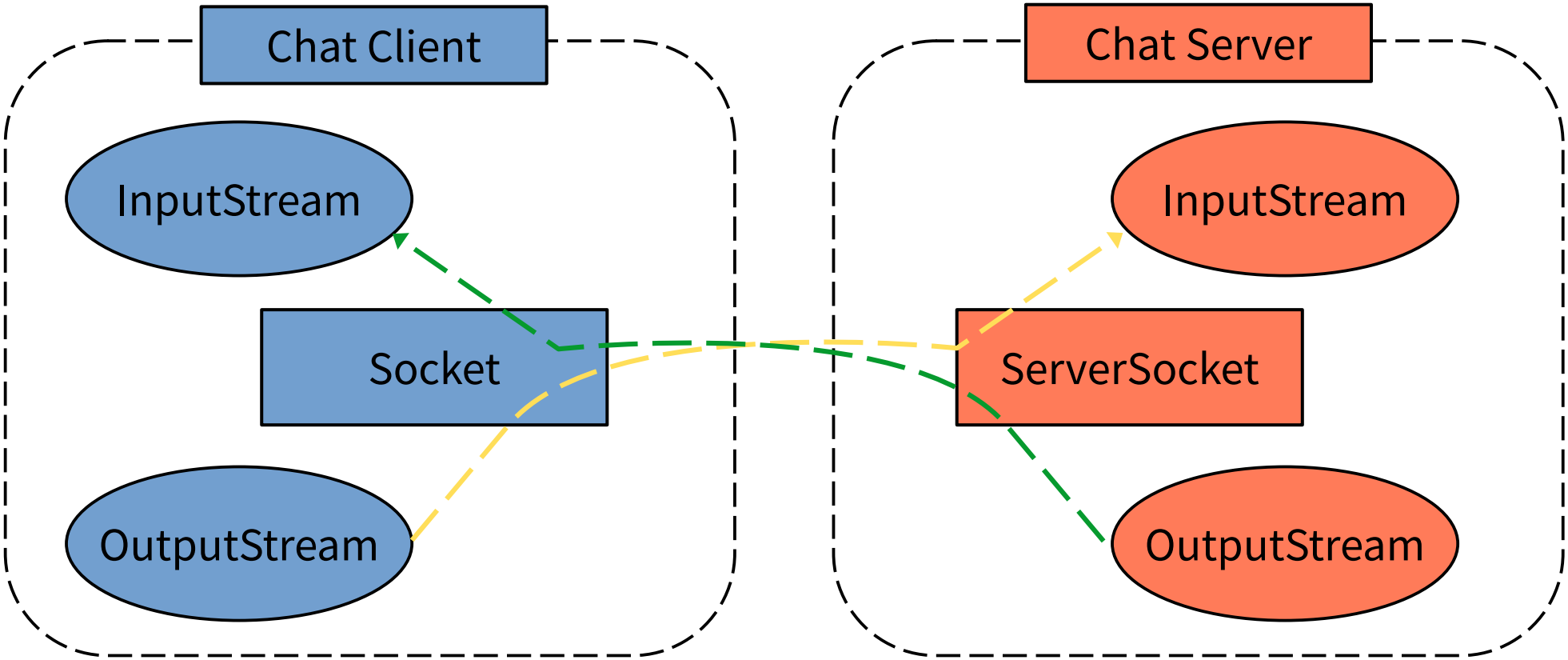
IP Address + Port Number = Socket

```
Socket socket = new Socket("127.0.0.1", 60001)
```

Client & Server implementation

- In the proposed system, Those who create a chatroom (admin to that specific room) hosts chat server (server logic).
- Those who joins chatroom as member runs client logic on thier machine.
- Chat server actively listenes to incoming join requests and creates a ServerSocket.
- Chat client request to join chatroom through Socket communication.

Communication by Socket Input Stream and Output Stream




```
Socket sock = new Socket("127.0.0.1", 61000);
```

```
ServerSocket server = new ServerSocket(61000);
Socket sock = server.accept();
```

Java I/O Package

- java.io.InputStream
- java.io.OutputStream
- java.io.PrintWriter
- java.io.BufferedReader

Java Network Package

- java.net.Socket
- java.net.ServerSocket

Class ServerSocket

Constructors

Constructor and Description
`ServerSocket()` Creates an unbound server socket.
`ServerSocket(int port)` Creates a server socket, bound to the specified port.
`ServerSocket(int port, int backlog)` Creates a server socket and binds it to the specified local port number, with the specified backlog.
`ServerSocket(int port, int backlog, InetAddress bindAddr)` Create a server with the specified port, listen backlog, and local IP address to bind to.

Class ServerSocket

All Methods	Static Methods	Instance Methods	Concrete Methods

Modifier and Type	Method and Description
Socket	accept() Listens for a connection to be made to this socket and accepts it.
void	bind(SocketAddress endpoint) Binds the ServerSocket to a specific address (IP address and port number).
void	bind(SocketAddress endpoint, int backlog) Binds the ServerSocket to a specific address (IP address and port number).
void	close() Closes this socket.

Class Socket

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	**bind**(SocketAddress bindpoint) Binds the socket to a local address.
void	**close**() Closes this socket.
void	**connect**(SocketAddress endpoint) Connects this socket to the server.
void	**connect**(SocketAddress endpoint, int timeout) Connects this socket to the server with a specified timeout value.
SocketChannel	**getChannel**() Returns the unique **SocketChannel** object associated with this socket, if any.
InetAddress	**getInetAddress**() Returns the address to which the socket is connected.
InputStream	**getInputStream**() Returns an input stream for this socket.

Class Socket

All Methods	Static Methods	Instance Methods	Concrete Methods

boolean	**getKeepAlive()** Tests if **SO_KEEPALIVE** is enabled.
InetAddress	**getLocalAddress()** Gets the local address to which the socket is bound.
int	**getLocalPort()** Returns the local port number to which this socket is bound.
SocketAddress	**getLocalSocketAddress()** Returns the address of the endpoint this socket is bound to.
boolean	**getOOBInline()** Tests if **SO_OOBINLINE** is enabled.
OutputStream	**getOutputStream()** Returns an output stream for this socket.

Sample Server Socket Implementation

```
try  {
   ServerSocket server = new ServerSocket(1728);
   while (true) {
      Socket connection = server.accept();
      provideService(connection);
   }
}
catch (IOException e) {
   System.out.println("Server shut down with error: " + e);
}

//provideService() method defines logic to communicate with
client and providing them service. ex.PrintWriter,
BufferReader, etc.
```

Sample Client Socket Implementation

```
try {
    Socket connection = new Socket(computerName,serverPort);
    InputStream in = connection.getInputStream();
    OutputStream out = connection.getOutputStream();
}
catch (IOException e) {
    System.out.println("Attempt to create connection failed
                        with error: " + e);
}

// Use the streams, in and out, to communicate with the
server. InputStream to read, OutputStram to write to.
```

Thread Java programming

- A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.
- A Thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Thread Constructors and methods

Constructors

Constructor and Description
`Thread()` Allocates a new Thread object.
`Thread(Runnable target)` Allocates a new Thread object.
`Thread(Runnable target, String name)` Allocates a new Thread object.
`Thread(String name)` Allocates a new Thread object.
`Thread(ThreadGroup group, Runnable target)` Allocates a new Thread object.

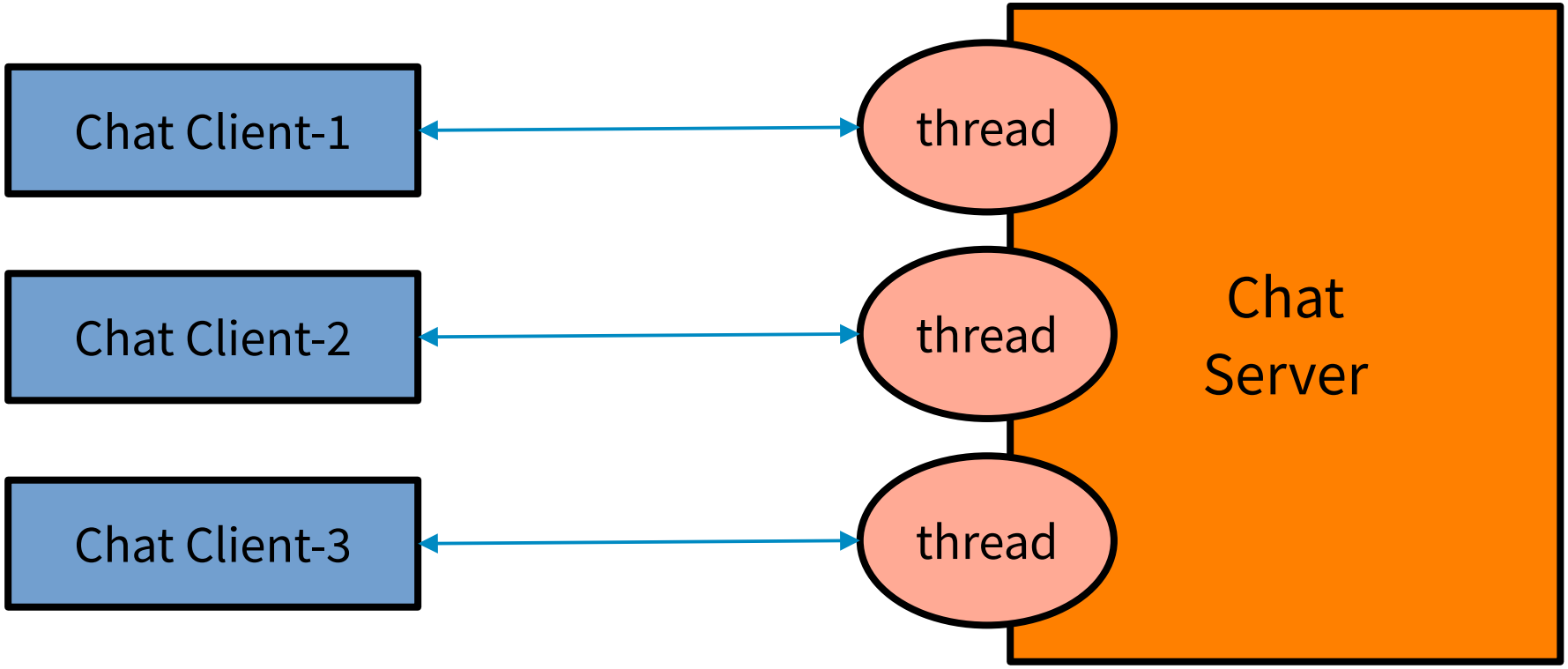
Thread Constructors and methods

All Methods | Static Methods | Instance Methods | Concrete Methods

void	**start()** Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread.

void	**run()** If this thread was constructed using a separate Runnable run object, then that Runnable object's run method is called; otherwise, this method does nothing and returns.

Thread chat server concept of multi-client access

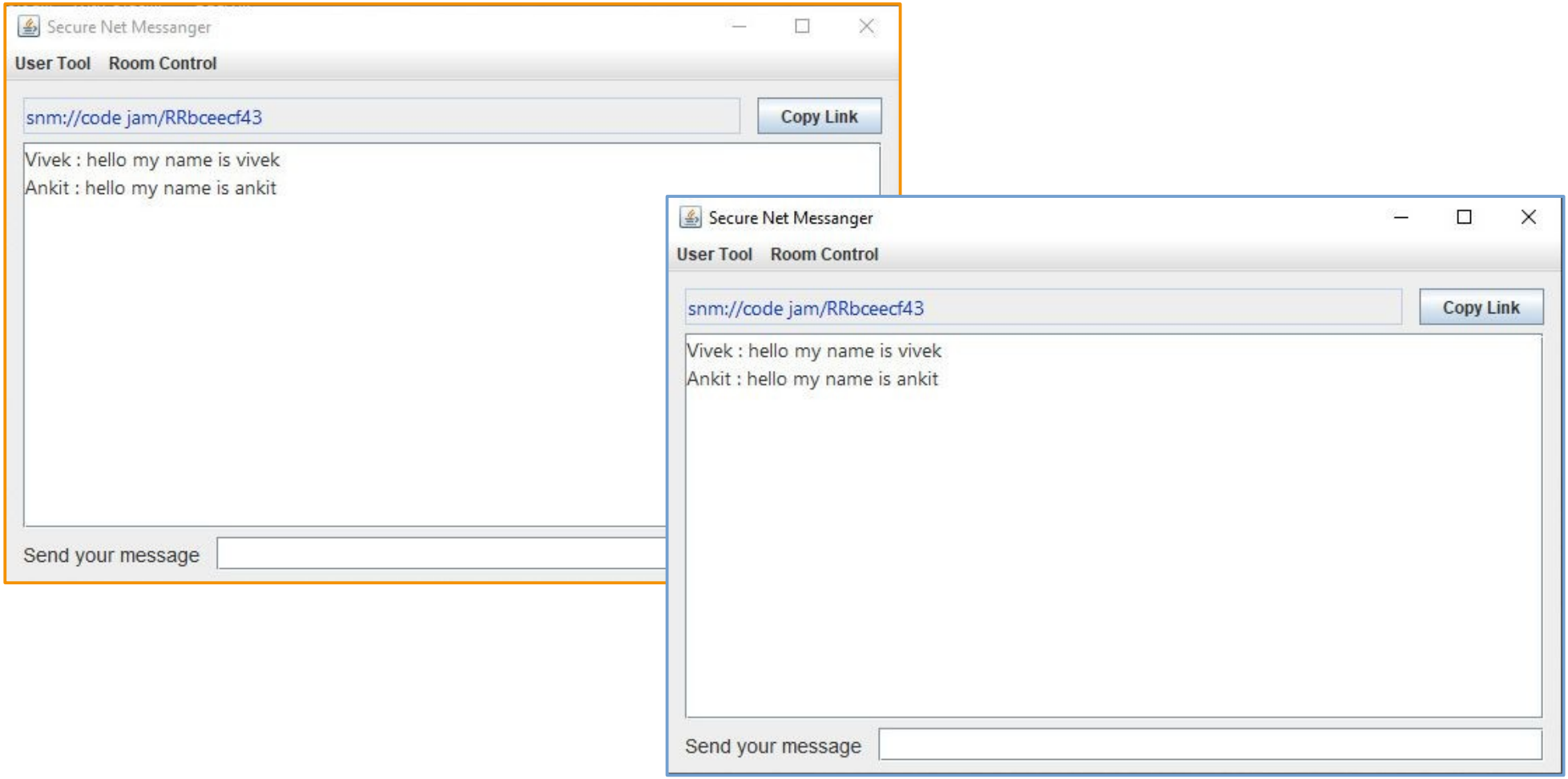


N - Clients

TCP based Network

Multi-thread Service

Chat Window Simulation



END