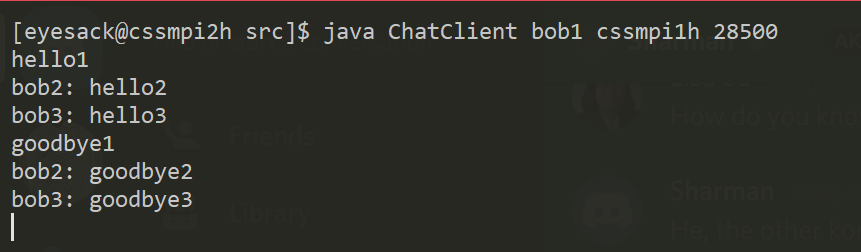
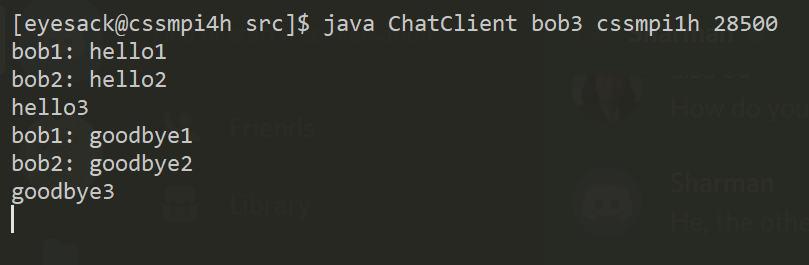
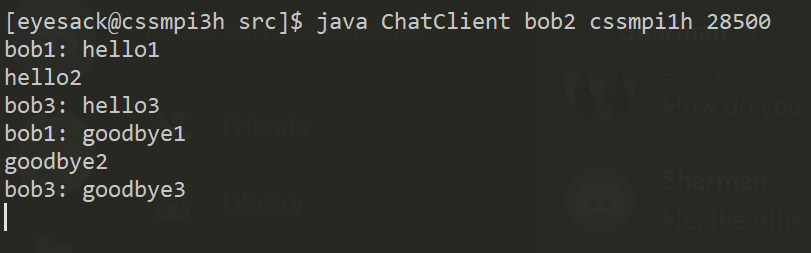
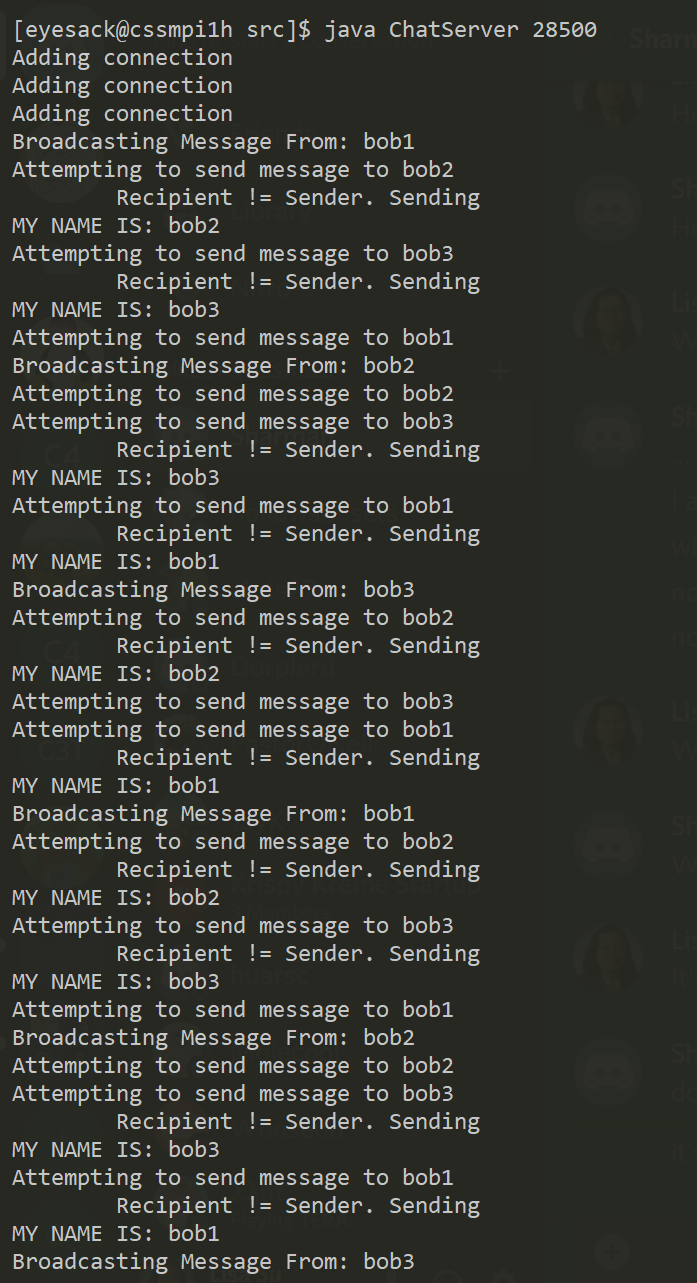
Part 1: Consistent Ordering

ChatServer.java + Connection.java

import java.net.\*;  
import java.io.\*;  
import java.util.\*;  
  
public class ChatServer {  
 private ServerSocket svr;  
 private Socket socket;  
 private static Set<Connection> connections;  
  
  
 public ChatServer(int port) {  
 connections = new HashSet<>();  
  
 try {  
 *// initialize server socket* svr = new ServerSocket(port);  
 *// set non blocking timeout* svr.setSoTimeout(500);  
  
 while (true) {  
 try {  
 socket = svr.accept();  
 } catch (SocketTimeoutException e) {  
 socket = null;  
 }  
  
 *// if a user connects* if (socket != null) {  
 System.out.println("Adding connection");  
 Connection connection = new Connection(socket);  
 *// add them to the connection list* connections.add(connection);  
 }  
  
 for (Connection connection : connections) {  
 *// if a connection wants to broadcast a message* if (connection.isReadAvailable()) {  
 String message = connection.readMessage();  
 *// if the message broadcast was null, an error occurred* if (message == null) {  
 System.out.println("Deleting connection1");  
 connection.deleteSocket();  
 connections.remove(connection);  
 } else  
 broadcast(message, connection);  
 }  
 }  
  
 System.out.println("----------");  
 }  
  
 } catch (Exception e) {  
*// e.printStackTrace();* }  
 }  
  
 *// broadcasts message to every other connection in connection list* private static void broadcast(String message, Connection sender) {  
 System.out.println("Broadcasting Message From: " + sender.getUserName());  
  
 boolean success = false;  
  
 for (Connection connection : connections) {  
 System.out.println("Attempting to send message to " + connection.getUserName());  
  
 if (!connection.equals(sender)) {  
 System.out.println("\tRecipient != Sender. Sending");  
  
 success = connection.writeMessage(message);  
 *// if the message was unable to be broadcast to a user  
 // an error occurred or they disconnected. either way, delete them* if (!success) {  
 System.out.println("Deleting connection2");  
 connection.deleteSocket();  
 connections.remove(connection);  
 }  
 }  
 }  
  
 }  
  
 public static void main(String args[]) {  
 if (args.length != 1) {  
 System.err.println("Syntax: java ServerClient <port>");  
 System.exit(1);  
 }  
  
 int port = Integer.parseInt(args[0]);  
 new ChatServer(port);  
 }  
}

import java.net.\*;  
import java.io.\*;  
  
public class Connection {  
 private Socket socket;  
 private String userName;  
 private String hostName;  
  
 private *InputStream* rawIn;  
 private DataInputStream in;  
 private DataOutputStream out;  
  
 public Connection(Socket \_socket) {  
  
 try {  
 socket = \_socket;  
 rawIn = socket.getInputStream();  
 in = new DataInputStream(socket.getInputStream());  
 out = new DataOutputStream(socket.getOutputStream());  
 userName = in.readUTF();  
 hostName = socket.getInetAddress().getHostName();  
  
 } catch(Exception e) {  
 e.printStackTrace();  
 }  
  
 }  
 public String getUserName() {  
 return userName;  
 }  
  
 public boolean isReadAvailable() {  
 try { return rawIn.available() > 0; }  
 catch (IOException e) { return false; }  
 }  
  
 public Boolean writeMessage(String message) {  
 try {  
 System.out.println("MY NAME IS: " + userName);  
 out.writeUTF(message);  
 return true;  
 } catch (IOException e) {  
 return false;  
 }  
 }  
  
 public String readMessage() {  
 try {  
 if (rawIn.available() > 0)  
  
 return userName + ": " + in.readUTF();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 @Override  
 public boolean equals(Object obj) {  
 if (!(obj instanceof Connection)) {  
 return false;  
 }  
 Connection otherChatConnection = (Connection) obj;  
 return userName.equals(otherChatConnection.userName)  
 && hostName.equals(otherChatConnection.hostName);  
 }  
 public void deleteSocket() {  
 try {  
 socket.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 public static void main(String[] args) {  
  
 }  
}



**Documentation**

My ChatServer class behaves similarly to lab1b’s TcpServerDouble class. It begins by initializing a ServerSocket and then enters a loop that continually accepts client sockets.

New client’s sockets are first passed into a “Connection” object. Connection objects store the socket itself, extract the client’s username, hostname, and generate input and output streams. Connection objects are then added to the set of Connections

For each Connection in Connections, chcek if a Connection desires to broadcast a message (determined by isReadAvailable()). If so, read the message using readMessage(). If the message is null, indicating some error, remove the Connection. The destructor for the Connection simply closes the socket. Otherwise broadcast() the message using the sender’s username.

Broadcast() loops through each Connection again. If the Connection is the sender, skip it to prevent sending the message back to the sender. Otherwise, send the message to the Connection using writeMessage(). If writeMessage() fails, delete the connection

**Algorithm Efficiency**

The main areas of my algorithm that impact performance are 1) Reading a client’s message broadcast requests and 2) actually broadcasting the message to all other clients.

Checking if a client wants to broadcast a message is O(# of clients) and broadcasting that message to every other client is also O(# of clients)

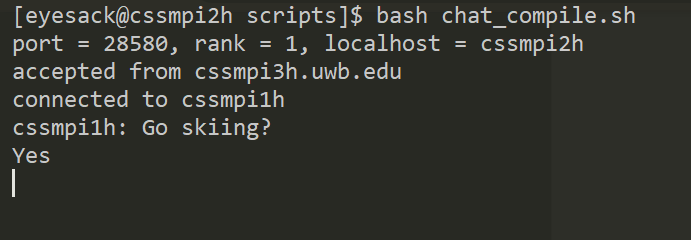
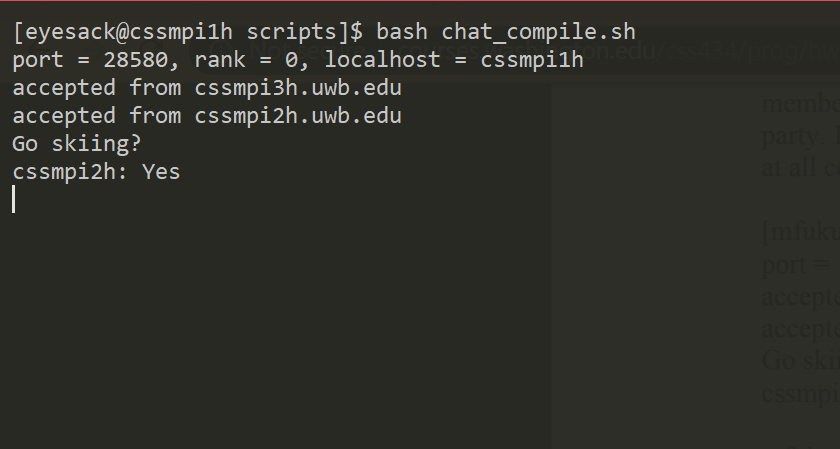
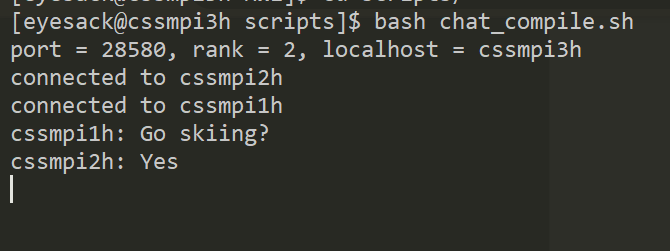
**Performance and functional improvement**

One possible change would be to replace the current “polling” system with an “interrupt” system. However further analysis would have to be performed to determine whether this would increase performance (reduce latency and memory usage). Currently, a single error when attempting to read a connection’s message results or write a message to that connection results in the deletion of the connection. One functional improvement would be to give connections three chances (errors) before deleting them.

Part 2: Causal Ordering

Chat.java

import java.net.\*; *// ServerSocket, Socket*import java.io.\*; *// InputStream, ObjectInputStream, ObjectOutputStream*import java.util.\*; *// Arraylist*public class Chat {  
 *// Each element i of the following arrays represent a chat member[i]* private Socket[] sockets; *// connection to i* private int[] stamps;  
 private ArrayList<String> msgQueue;  
 private ArrayList<int[]> stampQueue;  
 private ArrayList<Integer> srcQueue;  
  
 private *InputStream*[] indata; *// used to check data from i* private ObjectInputStream[] inputs; *// a message from i* private ObjectOutputStream[] outputs; *// a message to i* public Chat(int port, int rank, String[] hosts) throws IOException {  
 *// print out my port, rank and local hostname* System.out.println("port = " + port + ", rank = " + rank +  
 ", localhost = " + hosts[rank]);  
  
 *// create sockets, inputs, outputs, and vector arrays* sockets = new Socket[hosts.length];  
 stamps = new int[hosts.length];  
 msgQueue = new ArrayList<String>();  
 stampQueue = new ArrayList<int[]>();  
 srcQueue = new ArrayList<Integer>();  
  
  
 indata = new InputStream[hosts.length];  
 inputs = new ObjectInputStream[hosts.length];  
 outputs = new ObjectOutputStream[hosts.length];  
  
 *// establish a complete network* ServerSocket server = new ServerSocket(port);  
 for (int i = hosts.length - 1; i >= 0; i--) {  
 if (i > rank) {  
 *// accept a connection from others with a higher rank* Socket socket = server.accept();  
 String src\_host = socket.getInetAddress().getHostName();  
  
 *// find this source host's rank* for (int j = 0; j < hosts.length; j++) {  
 if (src\_host.startsWith(hosts[j])) {  
 *// j is this source host's rank* System.out.println("accepted from " + src\_host);  
  
 *// store this source host j's connection, input stream  
 // and object intput/output streams.* sockets[j] = socket;  
 indata[j] = socket.getInputStream();  
 inputs[j] = new ObjectInputStream(indata[j]);  
 outputs[j] = new ObjectOutputStream(socket.getOutputStream());  
 }  
 }  
 }  
  
 if (i < rank) {  
 *// establish a connection to others with a lower rank* sockets[i] = new Socket(hosts[i], port);  
 System.out.println("connected to " + hosts[i]);  
  
 *// store this destination host j's connection, input stream  
 // and object intput/output streams.* outputs[i] = new ObjectOutputStream(sockets[i].getOutputStream());  
 indata[i] = sockets[i].getInputStream();  
 inputs[i] = new ObjectInputStream(indata[i]);  
 }  
 }  
  
 *// create a keyboard stream* BufferedReader keyboard  
 = new BufferedReader(new InputStreamReader(System.in));  
  
 *// now goes into a chat* while (true) {  
 *// read a message from keyboard and broadcast it to all the others.* if (keyboard.ready()) {  
 *// since keyboard is ready, read one line.* String message = keyboard.readLine();  
 if (message == null) {  
 *// keyboard was closed by "^d"* break; *// terminate the program* }  
 *// broadcast a message to each of the chat members.* stamps[rank]++;  
 for (int i = 0; i < hosts.length; i++) {  
 if (i != rank) {  
 *// of course I should not send a message to myself* outputs[i].writeObject(stamps);  
 outputs[i].writeObject(message);  
 outputs[i].flush(); *// make sure the message was sent* }  
 }  
 }  
  
 *// read a message from each of the chat members* for (int i = 0; i < hosts.length; i++) {  
 *// to intentionally create a misordered message deliveray,  
 // let's slow down the chat member #2.* try {  
 if (rank == 2)  
 Thread.currentThread().sleep(5000); *// sleep 5 sec.* } catch (InterruptedException e) {}  
  
 *// check if chat member #i has something* if (i != rank && indata[i].available() > 0) {  
 *// read a message from chat member #i and print it out  
 // to the monitor* try {  
 int[] srcStamps = (int[]) inputs[i].readObject();  
 String message = (String) inputs[i].readObject();  
  
 if (stampsTooDifferent(i, srcStamps)) {  
 *// defer message and add to queue* msgQueue.add(message);  
 stampQueue.add(srcStamps);  
 srcQueue.add(i);  
 } else {  
 System.out.println(hosts[i] + ": " + message);  
 stamps[i]++;  
 }  
  
 } catch (ClassNotFoundException e) {}  
 }  
 }  
  
 *// for each message in the msgQueue* for (int i = 0; i < stampQueue.size(); i ++) {  
 int[] pendingStamp = stampQueue.get(i);  
 int pendingSrc = srcQueue.get(i);  
 String pendingMessage = msgQueue.get(i);  
  
 *// once again, check if it's similar enough to print* if (!stampsTooDifferent(pendingSrc, pendingStamp)) {  
 *// if so, remove message info from all three queues* stampQueue.remove(i);  
 srcQueue.remove(i);  
 msgQueue.remove(i);  
  
 stamps[pendingSrc]++;  
 System.out.println(hosts[pendingSrc] + ": " + pendingMessage);  
 }  
 }  
 }  
 }  
  
 *// checks whether a message can be received based on  
 // the receiver's stamp and the sender's stamp* public boolean stampsTooDifferent(int srcRank, int[] srcStamps) {  
 for (int i = 0; i < stamps.length; i++) {  
  
 *// if the sender's sent count in our stamps is not exactly  
 // one less than the sender's sent count in their own stamp* if (i == srcRank && srcStamps[i] + 1 != stamps[i])  
 return false;  
 else {  
 if (srcStamps[i] <= stamps[i])  
 return false;  
 }  
 }  
 return true;  
 }  
  
  
 public static void main(String[] args) {  
  
 *// verify #args.* if (args.length < 2) {  
 System.err.println("Syntax: java Chat <port> <ip1> <ip2> ...");  
 System.exit(-1);  
 }  
  
 *// retrieve the port* int port = 0;  
 try {  
 port = Integer.parseInt(args[0]);  
 } catch (NumberFormatException e) {  
 e.printStackTrace();  
 System.exit(-1);  
 }  
 if (port <= 5000) {  
 System.err.println("port should be 5001 or larger");  
 System.exit(-1);  
 }  
  
 *// retireve my local hostname* String localhost = null;  
 try {  
 localhost = InetAddress.getLocalHost().getHostName();  
 } catch (UnknownHostException e) {  
 e.printStackTrace();  
 System.exit(-1);  
 }  
  
 *// store a list of computing nodes in hosts[] and check my rank* int rank = -1;  
 String[] hosts = new String[args.length - 1];  
 for (int i = 0; i < args.length - 1; i++) {  
 hosts[i] = args[i + 1];  
 if (localhost.startsWith(hosts[i]))  
 *// found myself in the i-th member of hosts* rank = i;  
 }  
  
 *// now start the Chat application* try {  
 new Chat(port, rank, hosts);  
 } catch (IOException e) {  
 e.printStackTrace();  
 System.exit(-1);  
 }  
 }  
}



**Documentation**

The int array stamps keeps track of how many messages every client in the chat group has sent. If a client wants to send a message, they first have to increment their sent count in their personal stamp writeObject() their personal stamp and then the message.

When receiving a message, the sender’s stamp is first used to verify whether it can be printed using the stampsTooDifferent function. If the sender's sent count in their stamps is not exactly one ahead of the sender's sent count in own own stamp, then this message is too far ahead for us. Or if any other sent count in the sender’s stamps is less than our own sent count, than we are too far ahead for them.

If the message is not too different print the message and increment that sender’s sent count in our stamp. If it’s too different, store the message, the stamp, and the sender in msgQueue, stampQueue, and srcQueue respectively.

Then iterate through the stampQueue to check if recent changes have made any acceptable. If queued stamps are no longer too different, pop the corresponding message, stamp, and src. Send message as normal.

**Algorithm Efficiency**

The main areas of my algorithm that impact performance are 1) Reading a client’s message broadcast requests and 2) actually broadcasting the message to all other clients, 3) reading messages broadcast from other clients 4) checking if two stamps are too different, and 5) checking if any messages in the msgQueue can be printed

Checking if a client wants to broadcast a message, broadcasting that message, checking if two stamps are too different, and reading broadcasted messages from other clients are all O(# of clients). Checking if any messages in msgQueue can be printed is O(len(msgQueue))

**Performance and functional improvement**

Similar to Part 1, one possible change would be to replace the current “polling” system with an “interrupt” system. Similar to Part 1, allowing multiple errors before deleting the connection would be a functional improvement as well.