A*

For each mechanism, what are possible causes, if any, of messages being lost?

UDP offers no guarantee of delivery, ordering, or duplicate protection**.** IP datagram packets may be dropped because of network congestion or error. Congestion can occur when too many messages are sent or when too many users are on the same network. Errors may occur when distances increase as there may be more servers in between, increasing the probability of hardware or software failure. If on a wireless networks, external interference can corrupt the data. Packets can also be dropped when their checksum fail (corrupted).

Our UDP implementation also has a fixed buffer and message size. When incoming transmissions exceed these allocations, the buffer overflows, and begins skipping packets (discussed further in the next section). - no mention to RMI working over TCP

RMI is very reliable. However, the target server can still malfunction (terminating early/breaking the connection). Programming errors can occur during interface implementation or parameter handling. There can also also be protocol disagreements between client and server. (Though these errors are not exclusive to RMI).

Are there any patterns in the way messages are lost?

Our RMI server implementation did not lose any of the 1000 messages (RMI 1) or even the 10,000 messages we sent from the client (RMI 2). This indicates a high level of reliability. Nevertheless, we did notice a significant drop in performance (higher latency) which was not present in UDP after the total message count exceeded 2000.

Our UDP server seemed to handle requests below 200 messages well (UDP 1). However, when we tested with 500 messages, missing packets began to surface at 260~ at an interval between 1 and 4, but most frequently between 1 and 2. (UDP 2). Multiple tests indicate that the start of and the number of missing messages vary randomly. Curious, we decided to tweak with the pack and buffer sizes through (pacData=**new byte**[], **recvSoc**.setReceiveBufferSize()), and all messages between 5 and 99 went missing! (UDP 3) This indicates that small buffer or large message sizes can cause significant packet drops until the buffer is flushed, reset after some time, or when the server is rebooted.

What is the relative reliability of the different communication mechanisms?

RMI is significantly more reliable than UDP for many reasons. The request-reply system, duplicate filtering, retransmission of results, and interface abstraction/checking significantly improves the reliability of RMI. On the other hand, beyond the reasons discussed in question one, we were able to send UDP messages without the UDP server even being up. Would that then be considered a 100% drop rate?

Which was easier to program and why?

RMI was easier to program for us mainly due to its high-level abstraction and Java programming syntax. The remote interface abstracts away many tasks like marshalling, garbage collection, registry, and security. The allowance of object reflection and method invocation, makes it easy to manipulate! UDP was longer and required a new understanding of datagram and sockets, which was educational, but more complex.

Figure RMI 1

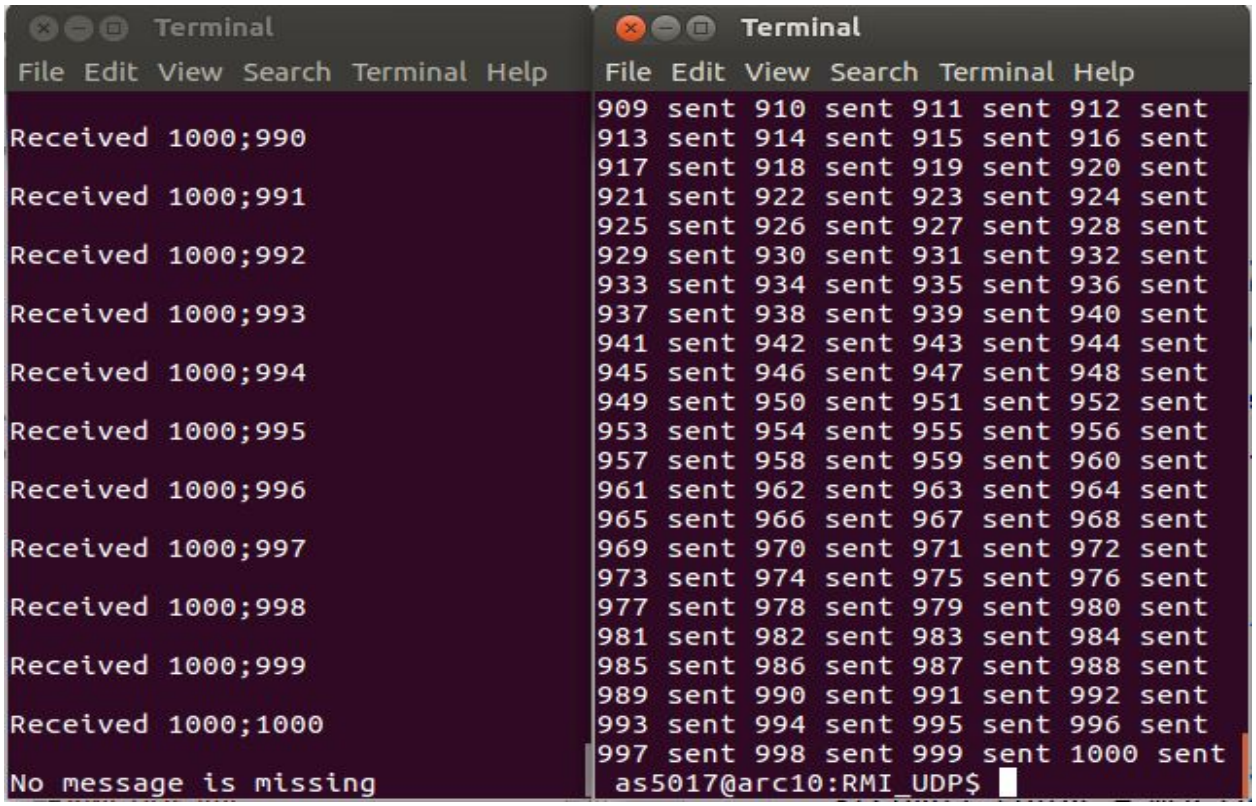


Figure RMI 2

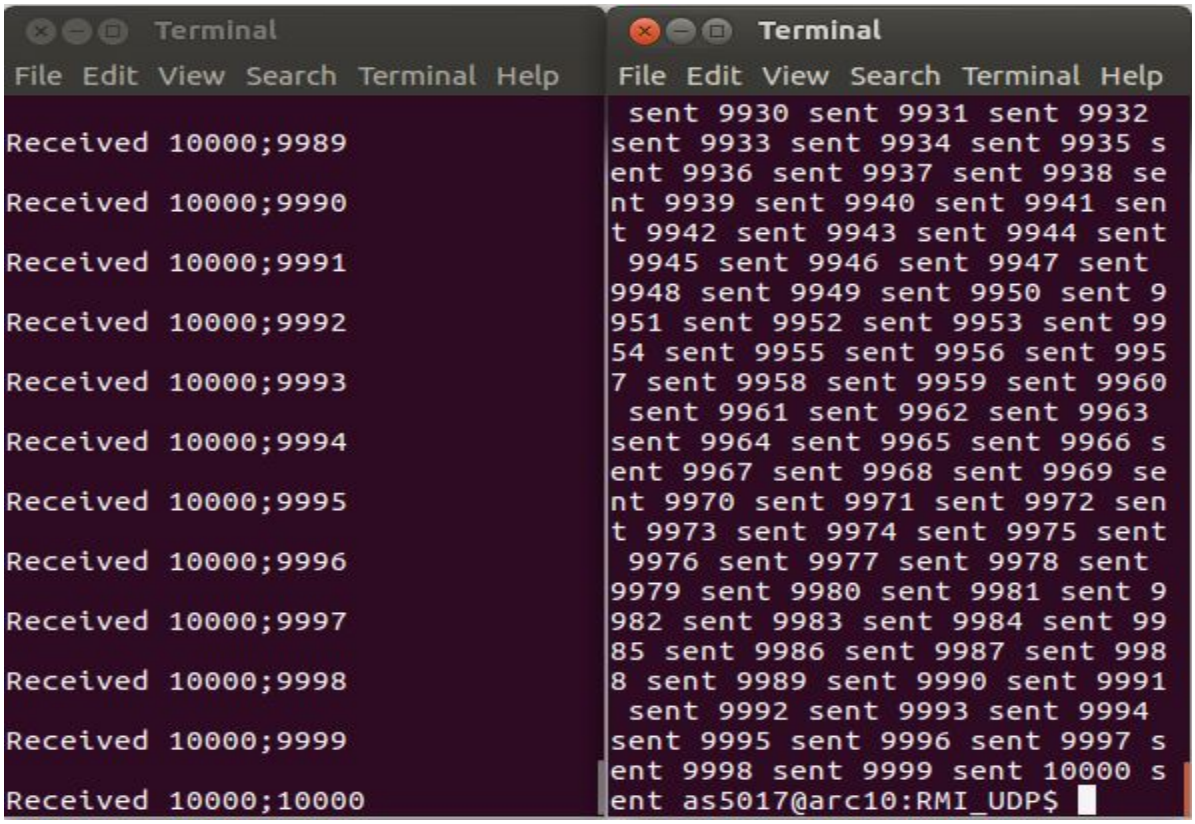


Figure UDP 1

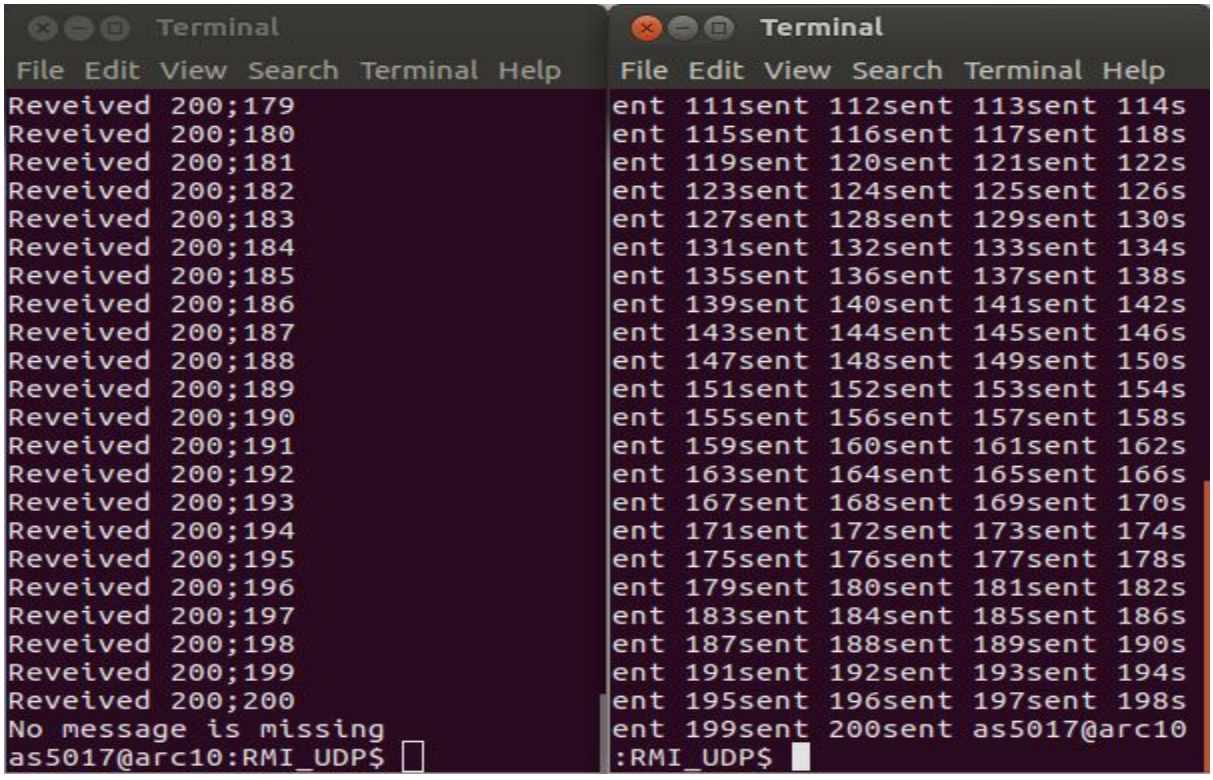


Figure UDP 2

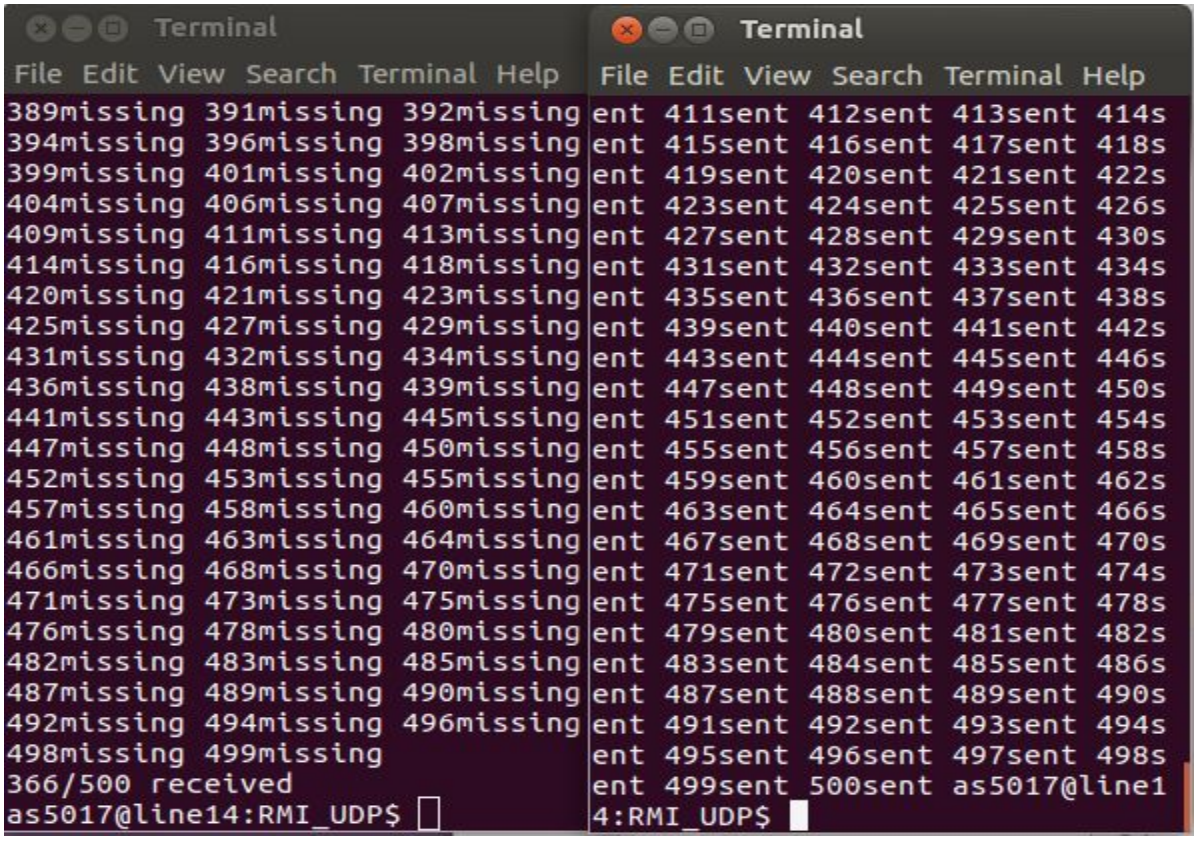
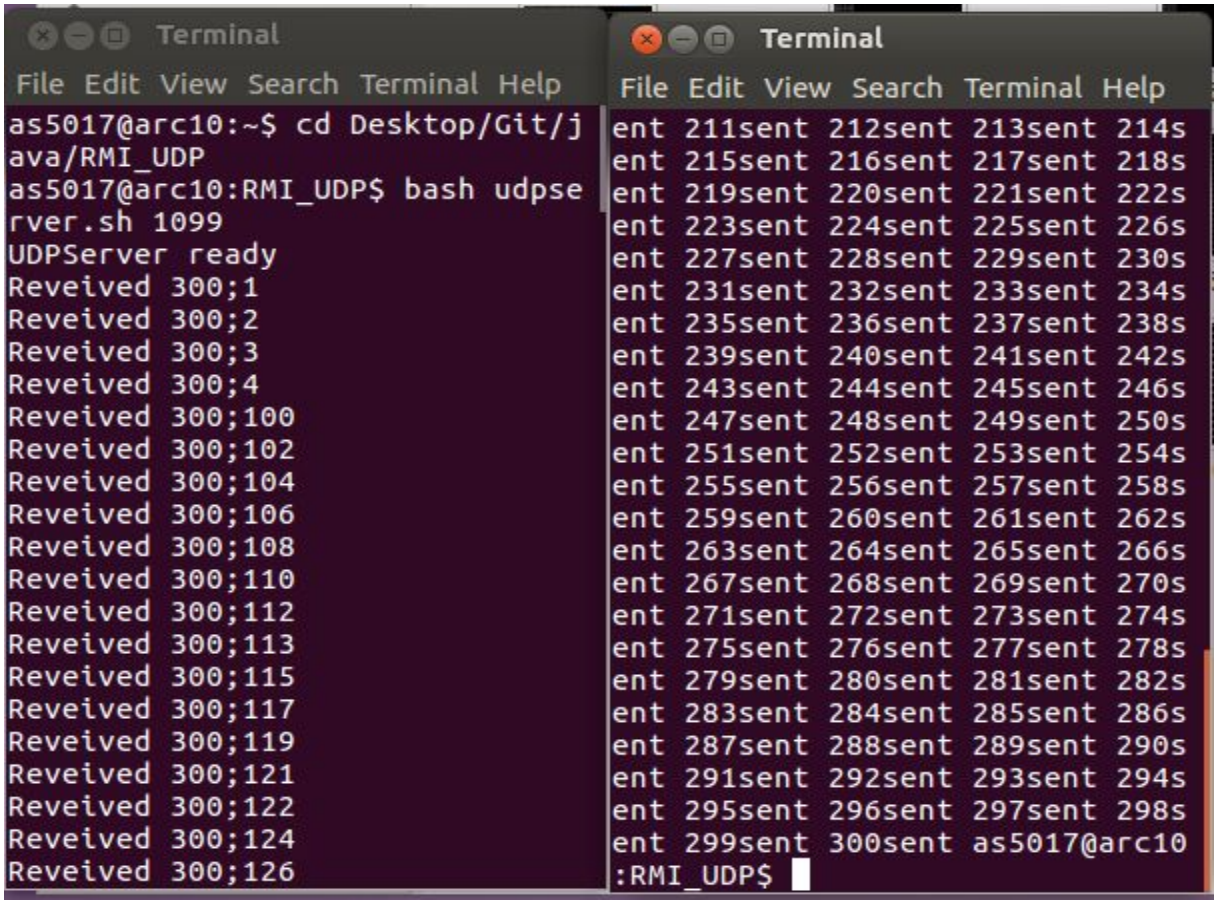


Figure UDP 3


Terminal
File Edit View Search Terminal Help
as5017@arc10:~$ cd Desktop/Git/java/RMI_UDP
as5017@arc10:RMI_UDP$ bash udpserver.sh 1099
UDPServer ready
Reveived 300;1
Reveived 300;2
Reveived 300;3
Reveived 300;4
Reveived 300;100
Reveived 300;102
Reveived 300;104
Reveived 300;106
Reveived 300;108
Reveived 300;110
Reveived 300;112
Reveived 300;113
Reveived 300;115
Reveived 300;117
Reveived 300;119
Reveived 300;121
Reveived 300;122
Reveived 300;124
Reveived 300;126
Terminal
File Edit View Search Terminal Help
ent 211sent 212sent 213sent 214s
ent 215sent 216sent 217sent 218s
ent 219sent 220sent 221sent 222s
ent 223sent 224sent 225sent 226s
ent 227sent 228sent 229sent 230s
ent 231sent 232sent 233sent 234s
ent 235sent 236sent 237sent 238s
ent 239sent 240sent 241sent 242s
ent 243sent 244sent 245sent 246s
ent 247sent 248sent 249sent 250s
ent 251sent 252sent 253sent 254s
ent 255sent 256sent 257sent 258s
ent 259sent 260sent 261sent 262s
ent 263sent 264sent 265sent 266s
ent 267sent 268sent 269sent 270s
ent 271sent 272sent 273sent 274s
ent 275sent 276sent 277sent 278s
ent 279sent 280sent 281sent 282s
ent 283sent 284sent 285sent 286s
ent 287sent 288sent 289sent 290s
ent 291sent 292sent 293sent 294s
ent 295sent 296sent 297sent 298s
ent 299sent 300sent as5017@arc10
:RMI_UDP$

```
public class RMIClient {

  public static void main(String[] args) {
    RMIServerI iRMIServer = null;

    // Check arguments for Server host and number of messages
    if (args.length < 2){
      System.out.println("Needs 2 arguments: ServerHostName/IPAddress,
TotalMessageCount");
      System.exit(-1);
    }

    String urlServer = new String("rmi://" + args[0] + "/RMIServer");
    int numMessages = Integer.parseInt(args[1]);

    // Initialise Security Manager
    if(System.getSecurityManager() == null)
      System.setSecurityManager(new SecurityManager());

    try{
      //Bind to RMIServer
      Registry registry = LocateRegistry.getRegistry(args[0]);
      RMIServerI server = (RMIServerI) registry.lookup("RMIServer");
      // Attempt to send messages the specified number of times
      for (int msgNum = 1; msgNum <= numMessages; msgNum++) {
        System.out.print(msgNum+" sent ");
        MessageInfo msg = new MessageInfo(numMessages, msgNum);
        server.receiveMessage(msg);
      }
    } catch (Exception e){          - should handle different exceptions separately
      e.printStackTrace();
      System.exit(1);
    }
  }
}
```

```
public class RMIServer extends UnicastRemoteObject implements
RMIServerI {

  private int totalMessages = -1;
  private int[] receivedMessages;
  private int msg_index = 0;
  private boolean[] miss_msg;

  public RMIServer() throws RemoteException {}

  public void receiveMessage(MessageInfo msg) throws RemoteException {
    // On receipt of first message, initialise the receive buffer
    if(totalMessages == -1) {
      totalMessages = msg.totalMessages;
      receivedMessages = new int[totalMessages];
      miss_msg = new boolean[totalMessages];
      for(int index=0; index<totalMessages; index++)
        miss_msg[index] = true;
    }

    // Log receipt of the message
    System.out.println("Received "+msg);
    miss_msg[msg.messageNum-1] = false;
    receivedMessages[msg_index++] = msg.messageNum;
    // If this is the last expected message, then identify any missing messages
    if (msg.messageNum == totalMessages){
      boolean any_miss = false;
      for (int index=0; index<totalMessages; index++)
        if (miss_msg[index] == true){
          any_miss = true;
          System.out.println("missing "+index+1);
        }
      if(!any_miss)
        System.out.println("No message is missing");
      totalMessages = -1;
      msg_index = 0;
```

```
    }
  }

  public static void main(String[] args) {
    // Initialise Security Manager
    if(System.getSecurityManager() == null)
      System.setSecurityManager(new SecurityManager());

    try{
      // Instantiate the server class
      String serverURL = "rmi://localhost/RMIServer";
      RMIServer rmis = new RMIServer();
      // Bind to RMI registry
      rebindServer(serverURL, rmis);
      System.out.println("RMIServer Bound");
    } catch ( Exception e){
      e.printStackTrace();
      System.exit(1);
    }
  }

  protected static void rebindServer(String serverURL, RMIServer server) {
    try {
      // Start / find the registry
      LocateRegistry.createRegistry(1099);
      // Rebind the server to the registry
      Naming.rebind(serverURL, server);
    } catch (Exception e){
      e.printStackTrace();              - should handle different exceptions separately
      System.exit(1);
    }
  }
}
```

```
public class UDPClient {

  private DatagramSocket sendSoc;

  public static void main(String[] args) {
    InetAddress    serverAddr = null;
    int recvPort;
    int countTo;
    String message;

    // Get the parameters
    if (args.length < 3) {
      System.err.println("Arguments required: server name/IP, recv port, message
count");
      System.exit(-1);
    }

    try {
      serverAddr = InetAddress.getByName(args[0]);
    } catch (UnknownHostException e) {
      System.out.println("Bad server address in UDPClient, " + args[0] + " caused an
unknown host exception " + e);
      System.exit(-1);
    }

    recvPort = Integer.parseInt(args[1]);
    countTo = Integer.parseInt(args[2]);
    // Construct UDP client class and try to send messages
    UDPClient udpClient = new UDPClient();
    udpClient.testLoop(serverAddr, recvPort, countTo);
  }

  // Initialise the UDP socket for sending data
  public UDPClient() {
    try {
      sendSoc = new DatagramSocket();
    } catch (Exception e) {
      e.printStackTrace();
```

```
      System.exit(1);
    }
  }

  // Send the messages to the server
  private void testLoop(InetAddress serverAddr, int recvPort, int countTo) {
    int    tries = 0;
    while(tries < countTo) {
      send(countTo+";"+(tries+1), serverAddr, recvPort);
      System.out.print((tries+1)+"sent ");
      tries++;
    }
  }

  // Build the datagram packet and send it to the server
  private void send(String payload, InetAddress destAddr, int destPort) {
    byte[] pktData = payload.getBytes();
    int    payloadSize = pktData.length;
    DatagramPacket pkt = new DatagramPacket(pktData, payloadSize, destAddr,
destPort);

    try {
      sendSoc.send(pkt);
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
  }
}
```

```
public class UDPServer {

  private DatagramSocket recvSoc;
  private int totalMessages = -1;
  private int[] receivedMessages;
  boolean[] miss_msg;
  private boolean close;
  private int msg_index = 0;
  private int count=0;

  // Use a timeout (e.g. 30 secs) to ensure the program doesn't block forever
  private void run() {
    int pacSize;
    byte[] pacData;
    pacData = new byte[32];
    DatagramPacket pac = new DatagramPacket(pacData, pacData.length);
    close = false;

    // Receive the messages and process them by calling processMessage(...)
    while(!close) {
      try {
        recvSoc.receive(pac);
        String data = new String(pac.getData(), 0, pac.getLength());
        processMessage(data);
      } catch (SocketTimeoutException tm_out_exc) {
        System.out.println("timed out!");
      } catch (Exception e){
        e.printStackTrace();
        System.exit(1);
      }
    }
  }

  public void processMessage(String data) {
    // Use the data to construct a new MessageInfo object
    MessageInfo msg = null;
    try{
      msg = new MessageInfo(data);
```

- not printing received/lost msgs in case of timeout

```
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }

    // Log receipt of the message
    if(totalMessages == -1) {
      close = false;
      totalMessages = msg.totalMessages;
      receivedMessages = new int[totalMessages];
      miss_msg = new boolean[totalMessages];
      for(int index = 0; index<totalMessages; index++)
        miss_msg[index] = true;
    }

    // If this is the last expected message, then identify any missing messages
    System.out.print("Received "+msg);
    miss_msg[msg.messageNum-1] = false;
    receivedMessages[msg_index++] = msg.messageNum;
    if (msg.messageNum == totalMessages){
      boolean any_miss = false;
      for (int index=0; index<totalMessages; index++)
        if (miss_msg[index]){
          any_miss = true;
          count+=1;
          System.out.print((index+1)+"missing ");
        }
      System.out.println("\n"+(totalMessages-count)+"/"+totalMessages+" received");
      if(!any_miss)
        System.out.println("No message is missing");
      close = true;
      System.exit(0);
    }
  }

  public UDPServer(int rp) {
    try {
      // Initialise UDP socket for receiving data
      recvSoc = new DatagramSocket(rp);
```

```
      // Use a timeout (e.g. 30 secs) to ensure the program doesn't block forever
      recvSoc.setSoTimeout(30000);
      // On receipt of first message, initialise the receive buffer
      recvSoc.setReceiveBufferSize(51200);
    } catch (Exception e) {
      e.printStackTrace();
      System.exit(1);
    }
    // Done Initialisation
    System.out.println("UDPServer ready");
  }

  public static void main(String args[]) {
    int    recvPort;

    // Get the parameters from command line
    if (args.length < 1) {
      System.err.println("Arguments required: recv port");
      System.exit(-1);
    }
    recvPort = Integer.parseInt(args[0]);
    // Construct Server object and start it by calling run().
    UDPServer udpServer = new UDPServer(recvPort);
    udpServer.run();
  }
}
```