Computer Networks and Distributed Systems — Assessed Coursework: RMI and UDP

An RMI (Remote Method Invocation) and a UDP (User Datagram Protocol) Client and Server were successfully written and ran in JAVA. The findings are outlined. Messages were sent between the client and the server, over a local area network on two different computers which were not physically close to each other. Messages were sent in increments of 100, starting from 10, up until 1000 (the initial increment was from 10 to 100).

UDP mechanism:

Sending 10-300 messages resulted in no message loss. Any more than 300 had some message loss, which increased at each 100 message increment.

Possible causes include: ***network congestion*** *which causes the 30ms timeout to be reached,* ***attenuation*** *of the packet signals due to interference,* ***network hardware failure*** *or* ***network drivers failure****. UDP does not guarantee receipt of the message being sent (no compensation for lost packets) and the ordering of message receipt is not defined.*

Furthermore, we observed that messages after message 303 have a very high chance of not arriving to the server. This might be due to some buffer in the network since by "slowing down" the client through a wait statement, we were able to increase the messages sent successfully by a considerable amount (very few messages if any were being lost even when sending thousands of messages).

RMI mechanism:

There was no packet loss at any number of messages sent from 10-1000.

Reason: *The RMI uses the TCP/IP protocol. Therefore a connection is established between client and server with a 3-way handshake. If a message is not received by the server, a request is made to the client to be re-sent. Therefore no messages were lost.*

RMI vs. UDP mechanism:

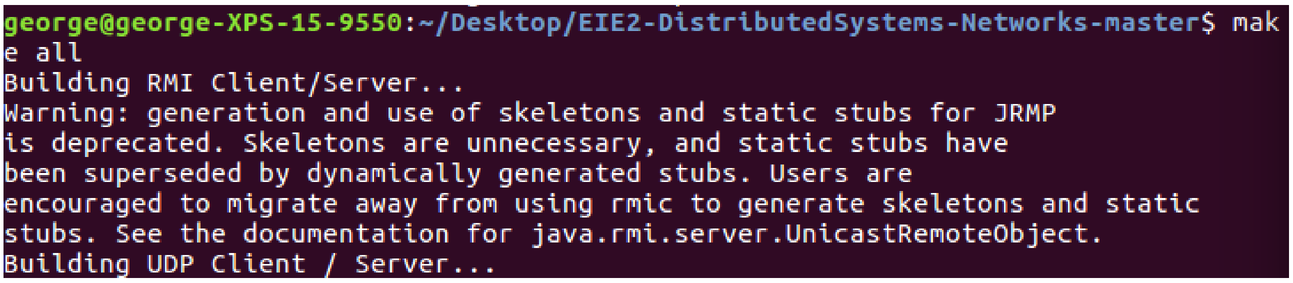
RMI: Guaranteed receipt of all messages, messages are received in the same order they were sent. Dropped packets are re-transmitted. It uses the security manager defined to protect systems from hostile applets. It is multi-threaded, allowing the server to exploit Java threads for better concurrent processing of client requests. It can pass full objects as arguments and return values, not just predefined data types. It is more reliable than UDP when data loss cannot be tolerated and in-order delivery is required (eg. establishing an SSH connection).

UDP: It does not need to retransmit lost packets and does not do any connection establishment, therefore sending data has less delay than RMI. Much better than RMI for multi-casting since it does not have to keep track of retransmissions/sending rate for multiple receivers. It is more reliable than RMI when data loss can be tolerated and in-order delivery of messages is not needed (eg. Domain Name Server Lookups).

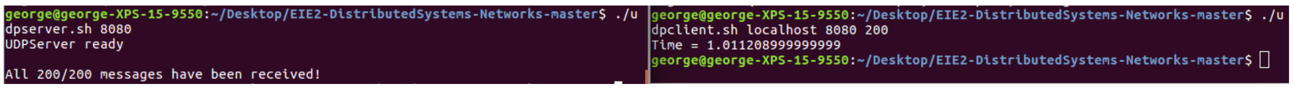
Programming Competence:

The easiest to program was the RMI mechanism. RMI offers a higher level of abstraction than UDP through the use of the readily available code in the JAVA libraries. On the server side, the programmer simply has to create a new instance of the RMIServer and bind it to the name. On the client, side the programmer simply, has to initialize the security manager, lookup the NameServer and invoke the available method. However, in UDP, the programmer must create a DatagramSocket at an agreed port, create a new DatagramPacket which will contain the information that will be transmitted as a byte array and send the DatagramPacket through the DatagramSocket. The issue here lies in the low-level of abstraction of the UDP packet (I.e. marshalling a complicated data set into an array of bytes will get quite cumbersome).

Proof that both RMI and UDP programs actually ran:

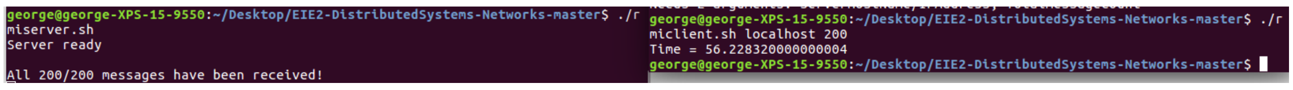


```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ mak
e all
Building RMI Client/Server...
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
Building UDP Client / Server...
```



```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ ./u
dpserver.sh 8080
UDPServer ready

All 200/200 messages have been received!
```

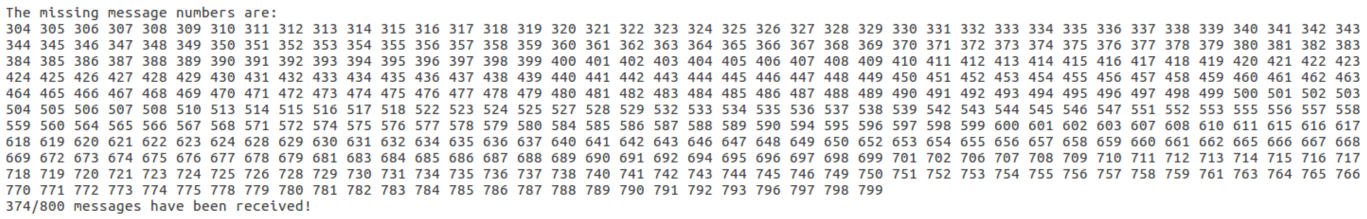
```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ ./u
dpclient.sh localhost 8080 200
Time = 1.011208999999999
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$
```



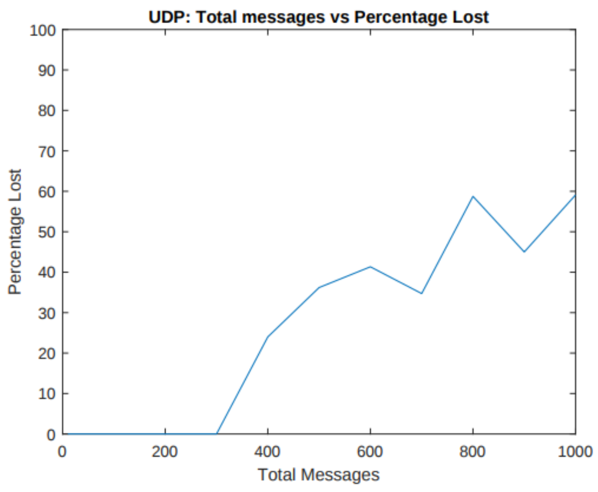
```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ ./r
miserver.sh
Server ready

All 200/200 messages have been received!
```

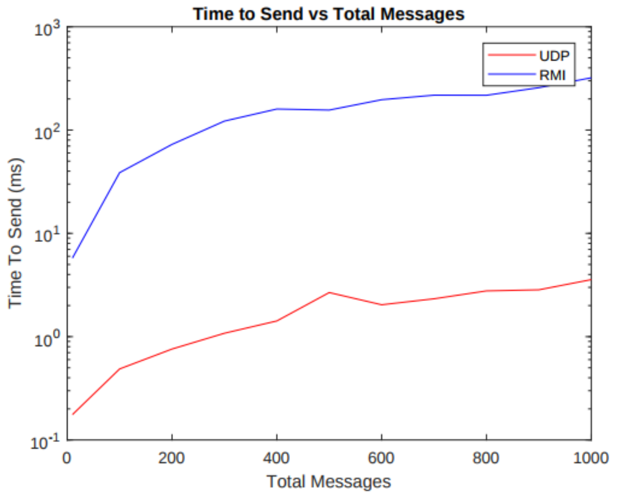
```
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$ ./r
miclient.sh localhost 200
Time = 56.228320000000004
george@george-XPS-15-9550:~/Desktop/EIE2-DistributedSystems-Networks-master$
```

Indication of which message number were lost when 1000 messages were transmitted using UDP mechanism:



```
The missing message numbers are:
304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343
344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383
384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423
424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463
464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503
504 505 506 507 508 510 513 514 515 516 517 518 522 523 524 525 527 528 529 532 533 534 535 536 537 538 539 542 543 544 545 546 547 551 552 553 555 556 557 558
559 560 564 565 566 567 568 571 572 574 575 576 577 578 579 580 584 585 586 587 588 589 590 594 595 596 597 598 599 600 601 602 603 607 608 610 611 615 616 617
618 619 620 621 622 623 624 628 629 630 631 632 634 635 636 637 640 641 642 643 646 647 648 649 650 652 653 654 655 656 657 658 659 660 661 662 665 666 667 668
669 672 673 674 675 676 677 678 679 681 683 684 685 686 687 688 689 690 691 692 694 695 696 697 698 699 701 702 706 707 708 709 710 711 712 713 714 715 716 717
718 719 720 721 723 724 725 726 728 729 730 731 734 735 736 737 738 740 741 742 743 744 745 746 749 750 751 752 753 754 755 756 757 758 759 761 763 764 765 766
770 771 772 773 774 775 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 796 797 798 799
374/800 messages have been received!
```





```java
package udp;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;

import common.MessageInfo;

public class UDPClient {

    private DatagramSocket sendSoc;

    //Calculate the amount of time spent sending messages
    private static double   execution = 0;

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

        // 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]);

        try{
            UDPClient client = new UDPClient();
            client.testLoop(serverAddr, recvPort, countTo);
            //Print the amount of time spent sending
            System.out.println("Time = " + execution);
        }catch(Exception e){
            e.printStackTrace();
        }
    }


    public UDPClient(){
        try{
            //Initialize UDP socket for sending data
            sendSoc = new DatagramSocket();

        }catch(SocketException e){
            e.printStackTrace();
        }
    }

    private void testLoop(InetAddress serverAddr, int recvPort, int countTo){
        //Loop for sending messages to server
        for(int tries=0; tries < countTo; tries++){
            String message = new String((Integer.toString(countTo)) + ";" + (Integer.toString(tries)));
            send(message,serverAddr,recvPort);
        }
    }

    private void send(String payload, InetAddress destAddr, int destPort){
        int payloadSize = payload.length();
        byte[] pktData =  new byte[128];

        pktData =  payload.getBytes();

        try{
            //Start timer
            long startTime = System.nanoTime();
            //Construct the DatagramPacket
            DatagramPacket  pkt = new DatagramPacket(pktData, payloadSize, destAddr, destPort);
            //Send the Datagram Packet to the Server
            sendSoc.send(pkt);
            //Stop timer
            long endTime = System.nanoTime();
            //Convert to milliseconds
            execution += ((endTime - startTime) / 1000000.0);
        }catch (Exception e){
            e.printStackTrace();
        }

    }
}
```

```java
package udp;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.util.Arrays;

import common.MessageInfo;

public class UDPServer {

    private DatagramSocket recvSoc;
    private static int totalMessages = -1;
    private static int[] receivedMessages;
    private static int messagesReceived = 0;
    private boolean close = false;

    public static void main(String args[]){

        if (args.length < 1) {
            System.err.println("Arguments required: recv port");
            System.exit(-1);

        }

        //Construct UDPServer
        UDPServer myServer = new UDPServer(Integer.parseInt(args[0]));

        try{
            //Run Server
            myServer.run();
        }catch(SocketTimeoutException e){
            //Check that we got at least one message
            if(totalMessages != -1){
                msg_log();
            }
            e.printStackTrace();
        }
    }


    private void run() throws SocketTimeoutException{

        byte[] pacData = new byte[128];
        int pacSize = pacData.length;
        DatagramPacket pac;

        try{
            //Loop until to close flag is set to true
            while(!close){
                //Construct new DatagramPacket
                pac = new DatagramPacket(pacData,pacSize);
                //Receive packet from client
                recvSoc.receive(pac);
                //Convert the data in the packet to a string
                String message = new String(pac.getData(),0,pac.getLength());
                processMessage(message);
            }
        }
        catch (IOException e){
            e.printStackTrace();
        }
    }


    public void processMessage(String data) {

        MessageInfo message = null;

        try {
            //Reconstruct the MessageInfo that was sent
            message = new MessageInfo(data);
        }
        catch(Exception e){
            e.printStackTrace();
        }

        if (totalMessages == -1){
            //Initialize the receive buffer
            totalMessages = message.totalMessages;
            receivedMessages = new int[totalMessages];
        }

        //Log receipt of the message
        messagesReceived++;
        receivedMessages[message.messageNum] = 1;

        //If we received all the messages then close
        //Because messages might not arrive in order, it is not sufficient if messageNum = totalMessages
        if(messagesReceived == totalMessages){
            msg_log();
            close = true;
        }
    }

    public static void msg_log(){
        int lost = totalMessages - messagesReceived;

        if(lost > 0){
            System.out.println("The missing message numbers are: ");

            //Print which messages have been lost
            for (int i=0; i < receivedMessages.length; ++i) {
                if(receivedMessages[i] == 0) {
                    System.out.print(i + " ");
                }

            }
            System.out.println();
            System.out.println(messagesReceived + "/" + totalMessages + " messages have been received!");
        }
        else{
            System.out.println();
            System.out.println("All " + totalMessages + "/" + totalMessages + " messages have been received!");
        }
    }

    public UDPServer(int rp) {

        try {
            //Construct new socket
            recvSoc = new DatagramSocket(rp);
            //Set socket timeout to 30 seconds
            recvSoc.setSoTimeout(30000);
            System.out.println("UDPServer ready");
        }
        catch(Exception msg){
            msg.printStackTrace();
        }
        // Done Initialisation
    }
}
```

```java
package rmi;
import java.io.IOException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

import common.MessageInfo;

public class RMIClient {

    //Calculate the ammount of time spent sending messages
    //Because this is a TCP/IP connection this will also include the time
    //for the server the receive the message and aknowledge the receipt of the message
    private static double execution = 0;

    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]);

        //Initialize the security manager
        if(System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }

        try {
            //Get reference ot RMIServer stub from the remote registry
            //Note the cast to in interface since the client interacts with the interface
            iRMIServer = (RMIServerI) Naming.lookup(urlServer);

            for(int i = 0; i < numMessages; i++) {
                //String message = new String( (Integer.toString(numMessages)) + ";" + (Integer.toString(i)) );
                MessageInfo msg = new MessageInfo(numMessages,i);

                //Start timer
                long startTime = System.nanoTime();
                //Send message
                iRMIServer.receiveMessage(msg);
                //Stop timer
                long endTime = System.nanoTime();
                //Convert to milliseconds
                execution += ((endTime - startTime) / 1000000.0);
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("Time = " + execution);
    }
}
```

```java
package rmi;

import java.net.MalformedURLException;
import java.rmi.AlreadyBoundException;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Arrays;

import common.*;

public class RMIServer extends UnicastRemoteObject implements RMIServerI {

    private int totalMessages = -1;
    private int[] receivedMessages;
    private int messagesReceived = 0;

    //Since RMIServer implements a remote interface, it's constructor must throw a RemoteException
    public RMIServer() throws RemoteException{}

    public static void main(String[] args){

        RMIServer rmis = null;

        //Initialize new security manager
        if(System.getSecurityManager() == null){
            System.setSecurityManager(new SecurityManager());
        }

        String serverURL = new String("rmi://localhost/RMIServer");

        try {
            //Construct the server
            rmis = new RMIServer();

            //Bind the rerver to RMI registry
            rebindServer(serverURL, rmis);
            System.out.println("Server ready");
        }
        catch (RemoteException e) {
            e.printStackTrace();
        }
    }


    public void receiveMessage(MessageInfo message) throws RemoteException {

        //On receipt of first message, initialise the receive buffer
        if (totalMessages == -1){
            totalMessages = message.totalMessages;
            receivedMessages = new int[totalMessages];
        }

        //Log the receipt of the message
        messagesReceived++;
        receivedMessages[message.messageNum] = 1;

        //Messages are in order (TCP/IP). If the last message is received, all the messages have been received
        if(message.messageNum == totalMessages - 1){
            msg_log();
        }
    }

    public void msg_log() {
        int lost = totalMessages - messagesReceived;

        //Print any lost messages
        if(lost > 0){
            System.out.println("The missing message numbers are: ");

            for (int i=0; i < receivedMessages.length; ++i) {
                if(receivedMessages[i] == 0) {
                    System.out.print(i + " ");
                }

            }
            System.out.println();
            System.out.println(messagesReceived + "/" + totalMessages + " messages have been received!");
        }
        else{
            System.out.println();
            System.out.println("All " + totalMessages + "/" + totalMessages + " messages have been received!");
        }
    }



    protected static void rebindServer(String serverURL, RMIServer server) {

        try {
            //Construct a registry on the localhost the listens to the specific port
            LocateRegistry.createRegistry(1099);

            //Rebing the serverURL to the remote object
            Naming.rebind(serverURL, server);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```