

95-702 Distributed Systems

Project 2

Assigned: Friday, January 31, 2014

Due: Midnight, Friday, Feb 21, 11:59pm

Principles

In this project, we attempt to illustrate several non-functional characteristics of distributed systems. One such characteristic is reliability. UDP is considered an unreliable protocol since it does nothing to ensure that packets arrive at their destination. In Task 1, we build a small client and server that use UDP to communicate. In Task 2, we add a measure of reliability with positive acknowledgements and retransmission.

Interoperability and performance (speed) are also important characteristics of distributed systems. In Task 3, we write a client and a server that exchange binary data. It is essential that both the client and the server know how to properly interpret the binary data sent and received. The binary transmission is fast – there is no need to convert the data coming off the network (or going to the network) into any special format.

Along the way, we introduce the proxy design pattern. This is a common pattern in distributed systems. The proxy design pattern is one example of the very sound engineering principle that we should strive to separate concerns. We see the same proxy pattern used in web services, RPC, and RMI. These topics will be visited carefully later in the course.

In Task 4, we build a secure system using symmetric key cryptography. The encryption occurs just above the TCP layer and is provided by the Tiny Encryption Algorithm (TEA). User authentication with ID's and passwords is done in the application layer. This is a common theme in modern systems. User names and passwords are passed over an encrypted, but otherwise insecure, channel. Thus, in Task 4, our main concern is with security – another important characteristic of many distributed systems.

For each task below, you must submit screenshots that demonstrate your programs running. These screenshots will aid the grader in evaluating your project.

Review

In Project 1 we worked with J2EE servlets and Java Server Pages using the Glassfish application server.

In this project we will be working at a lower level. That is, we will not have the Glassfish runtime environment to rely on. You may, however, continue to use Netbeans for all of this work. In this project, we will be programming with UDP and TCP sockets.

Discussion of UDP and TCP

In this project you will work with both UDP and TCP over IP. UDP is a simpler protocol than TCP and therefore usually requires more work from the application programmer. TCP presents the application programmer with a stream abstraction and is busier behind the scenes. TCP, unlike UDP, tries its best to make sure packets are delivered to the recipient. The underlying network we are using may, on occasion, drop packets. So, how can TCP provide for reliable delivery of packets? TCP uses the fundamental principal of "positive acknowledgement with retransmission".

A simplified example of "positive acknowledgement with retransmission" looks like the following. In this scenario, no packets are lost. These notes are adapted from "Internetworking with TCP/IP, Volume I: Principles, Protocols and Architecture" by Douglas E. Comer

Cool example: Positive acknowledgement with retransmission

Sender	Service
=====	
Send packet 1	
Start timer	
	Receive packet 1
	Send ACK for packet 1
Receive ACK 1	
Cancel timer	
Send packet 2	
Start timer	
	Receive packet 2
	Send ACK for packet 2
Receive ACK 2	
Cancel timer	

Here is an example where the first packet is lost.

Sender	Service
=====	
Send packet 1	
Start timer	
	Packet lost
Time expires	
Send packet 1	
Start timer	
	Receive packet 1
	Send ACK for packet 1
Receive ACK 1	
Cancel timer	
Send packet 2	
Start timer	
	Receive packet 2
	Send ACK for packet 2
Receive ACK 2	
Cancel timer	

Here is an example where the first ACK is lost.

Sender	Service
=====	
Send packet 1	
Start timer	
	Receive packet 1
	Send ACK 1
ACK 1 lost	
Time expires	
Send packet 1	
Start timer	
	Receive packet 1 a second time
	Send ACK for packet 1
Receive ACK 1	
Cancel timer	
Send packet 2	
Start timer	
	Receive packet 2
	Send ACK for packet 2
Receive ACK 2	
Cancel timer	

The acknowledgement may be replaced with the result of the service. Here is another example.

Sender	Service
=====	
Send packet 1	
Start timer	
	Receive packet 1
	Send response
	Response
	Lost
Time expires	
Send packet 1	
Start timer	
	Receive packet 1 a second time
	Send response again
Receive response	
Cancel timer	
Send packet 2	
Start timer	
	Receive packet 2
	Send response for packet 2
Receive response	
Cancel timer	

Task 1A

In Figures 4.3 and 4.4 of the Coulouris text, two short programs are presented. UDPClient.java and UDPServer.java communicate by sending and receiving UDP packets. The code for these programs can be found at:

<http://www.cdk5.net/wp/extra-material/source-code-for-programs-in-the-book>

Make modifications to UDPClient.java and UDPServer.java so that the client may ask the server to perform simple integer arithmetic. You need to implement addition, subtraction, multiplication, exponentiation and division of integers. You may assume that you have a well-behaved user and all input is correct. You may also assume that the user uses spaces to separate arguments. For the mathematical operators, use '+', '-', '/', '^', 'X' for exponentiation and the letter 'X' for multiplication.

The execution of the client program will prompt the user for input and will look like the following:

```
java UDPClient
Enter simple expression to be computed by the server:
2 ^ 3
2 ^ 3 = 8
```

UDPClientArithmetic.java and UDPServerArithmetic.java will be placed in a project called Project2Task1 and submitted to Blackboard.

Task 1B

Using the same UDP server that you wrote for Task 1A, write a new UDP client named UDPClientWithProxy.java that has a main routine that prompts the user for an integer $k > 0$ and then computes and displays the sum of the integers $1+2+3+\dots+k$. The main routine of the client must be very clean and contain no socket level programming. All of the socket work will be done within a single method with the following signature:

```
public static int add(int x,int y);
```

The main routine will make k calls on the add method. For example, if the user enters a 1 for k , the main routing will call the add method once: `add(0,1)`. If the user enters a 3 for k , the main routine will make three calls on add: `add(0,1)`, `add(1,2)`, `add(3,3)`. $1+2+3 = 6$.

Task 1B Continued

The add method on the client, however, will not perform any addition. Instead, it will send a UDP message to the server and receive the server's response. It will return that response as a simple integer. This is an example of the proxy design. The add method is simply being used as a proxy for the server and the server is where the addition is actually being carried out.

This is a design we will visit several times in the course. It's a simple example of the principle of separation of concerns. The main routine programmer need not be aware of how the add method is working. You may assume that your user is friendly and will enter a reasonable integer > 0 . No data validation is required.

Here is a sample execution:

```
java UDPClientWithProxy
```

```
Enter an integer > 0
```

```
100
```

```
1 + 2 + ... + 100 = 5050
```

UDPClientWithProxy.java will be placed in the same project (Project2Task1) as Task 1A.

Task 2

Build a new project called Project2Task2. Modify the UDPServer.java code and create a new Java class called UDPServerThatIgnoresYou.java. Write the new server so that it randomly ignores 80% of requests – a very unreliable server. In other words, the new UDPServerThatIgnoresYou will contain code close to this:

```
// rnd is an object of the Random class
aSocket.receive(request);
if(rnd.nextInt(10) < 8) {
    System.out.println("Got request " +
        new String(request.getData()) +
        " but ignoring it.");
    continue;
}
else {
    System.out.println("Got request" +
        new String(request.getData()));
    System.out.println("And making a reply");
}
```

Create a new client called UDPClientWithReliability.java. This new client is a modification of UDPClientWithProxy. After a request, it waits only 2 seconds for a reply. If the reply does not arrive after two seconds, the client tries again. Unlike many such systems, this one never gives up. The client side UDP receive logic will look something like this.

```
aSocket.setSoTimeout(2000);
aSocket.receive(reply);
```

See the above discussion on "positive acknowledgement with retransmission".

UDPClientWithReliability.java will have a main routine that prompts the user for $k > 0$ and then computes and displays the sum of the integers $1+2+3+\dots+k$. The main routine of the client must be very clean and contain no socket level programming. All of the socket work (and retry code) will be done by a single method with the following signature:

```
public static int add(int x, int y);
```

All files for this Task should be in a project named Project2Task2. You only need to implement an add method. This is an example of adding reliability to UDP.

Task 3

Build a new project called Project2Task3. Write a UDP server called `UDPServerWithDoubleArithmetic.java` that receives two double operands and an operator from a UDP client and returns the double result after applying the operator ('+', 'X', '-', or '/') to the two double operands.

Create a new client called `UDPClientWithDoubleArithmetic.java`. This new client will use the server and will compute the sum $1.75 + 2.75 + 3.75 + \dots + 100.75$. The main routine of the client should be free of any networking. This program does not prompt the user. The main routine simply makes calls on a proxy to compute the sum.

The point of this task is to work with marshaling and un-marshalling of parameters other than simple integers or character text. In this task, you will be reading and writing binary data rather than string data. In other words, we will not be taking doubles and converting them to strings before writing bytes to the wire. Instead, we will be writing the binary data (doubles) to the wire directly. The binary data is read by the server with no conversion from a string representation. In my solution, I made good use of the following Java methods:

```
double Double.longBitsToDouble(long);
```

```
long Double.doubleToLongBits(double);
```

In addition, I made good use of the following method (from StackOverflow):

```
public static long byteArrayToLong(byte bytes[]) {  
    long v = 0;  
    for (int i = 0; i < bytes.length; i++)  
    {  
        v = (v << 8) + (bytes[i] & 0xff);  
    }  
    return v;  
}
```

You may use `byteArrayToLong()` in your solution, but please take some time and study how it works. You will be required to understand this kind of code (bit manipulation using `&` and `<<`) on exams. `0xff` is a hexadecimal value that represents `11111111` in binary.

As a thinking exercise, how would this system perform when compared with one that converted doubles to strings before transmission? Which would be faster? Which would be more interoperable?

Task 4

This project will be named Project2Task4.

In this task we will make use of the Tiny Encryption Algorithm (TEA). You are not required to understand the underlying mechanics of TEA. You will need to be able to use it in your code. Later in the course, we will discuss TEA and describe it as one of many symmetric key encryption schemes. TEA is well known because of its small size and speed.

In Figure 4.5 and 4.6 of the Coulouris text, two additional short programs are presented: TCPClient and TCPServer. You can also find these programs at:

<http://www.cdk5.net/wp/extra-material/source-code-for-programs-in-the-book>

The TCPClient program takes two string arguments: the first is a message to pass and the second is an IP address of the server (e.g. localhost). The server will echo back the message to the client. Before running this example, look closely at how the command line argument list is used. You will need to include localhost on the command line. In Netbeans, command line arguments can be set by choosing Run/Set Project Configuration/Customize.

Make modifications to the TCPClient and TCPServer programs so that spies in the field are able to securely transmit their current location (longitude and latitude) to Intelligence Headquarters. Intelligence Headquarters is run by Sean Beggs. There are three spies and one spy commander as listed here:

Unique User-id	password	title	location
jamesb	james	spy	long, lat, alt
joem	joe	spy	long, lat, alt
mikem	mike	spy	long, lat, alt
seanb	sean	spy commander	

The spies are required to inform Sean of their locations as they move about the world (on super secret missions). Sean uses Google Earth to view the locations of his spies. The spies communicate over a channel encrypted using TEA. TEA is a symmetric key encryption algorithm and so Sean has provided his spies with the symmetric key before they left Hamburg Hall on their missions. Of course, Sean's software knows the

ID and password of each spy. So, while TEA is used for encryption, authentication is provided by the user name and password.

Name these two new programs `TCPSpyUsingTEAandPasswords.java` and `TCPSpyCommanderUsingTEAandPasswords.java`. The first is a TCP client used by each spy in the field. The second is a TCP server used by Spy Commander Beggs.

Here is an example execution of the client on Mike's machine.

```
java TCPSpyUsingTEAandPasswords
Enter symmetric key for TEA (taking first sixteen bytes):
thisissecretson'ttellanyone
Enter your ID: mikem
Enter your Password: mike
Enter your location: -79.956264,40.441068,0.00000
Data securely transmitted to Intelligence Headquarters
```

Here is an example execution of the client on Joe's machine.

```
java TCPSpyUsingTEAandPasswords
Enter symmetric key for TEA (taking first sixteen bytes):
thisissecretson'ttellanyone
Enter your ID: joem
Enter your Password: joe
Enter your location: -79.945389,40.444216,0.00000
Data securely transmitted to Intelligence Headquarters
```

Here is an example execution of the client on James Bond's machine.

```
java TCPSpyUsingTEAandPasswords
Enter symmetric key for TEA (taking first sixteen bytes):
thisissecretson'ttellanyone
Enter your ID: jamesb
Enter your Password: james
Enter your location: -79.940450,40.437394,0.000000
Data securely transmitted to Intelligence Headquarters
```

Here is an example execution of the client on Mallory's machine (Mallory is not an approved spy!!).

```
java TCPSpyUsingTEAandPasswords
Enter symmetric key for TEA (taking first sixteen bytes):
thisissecretson'ttellanyone
Enter your ID: joem
Enter your Password: sesame
Enter your location: -80.0,40.4417,0.000000
Illegal user name or password or symmetric key.
```

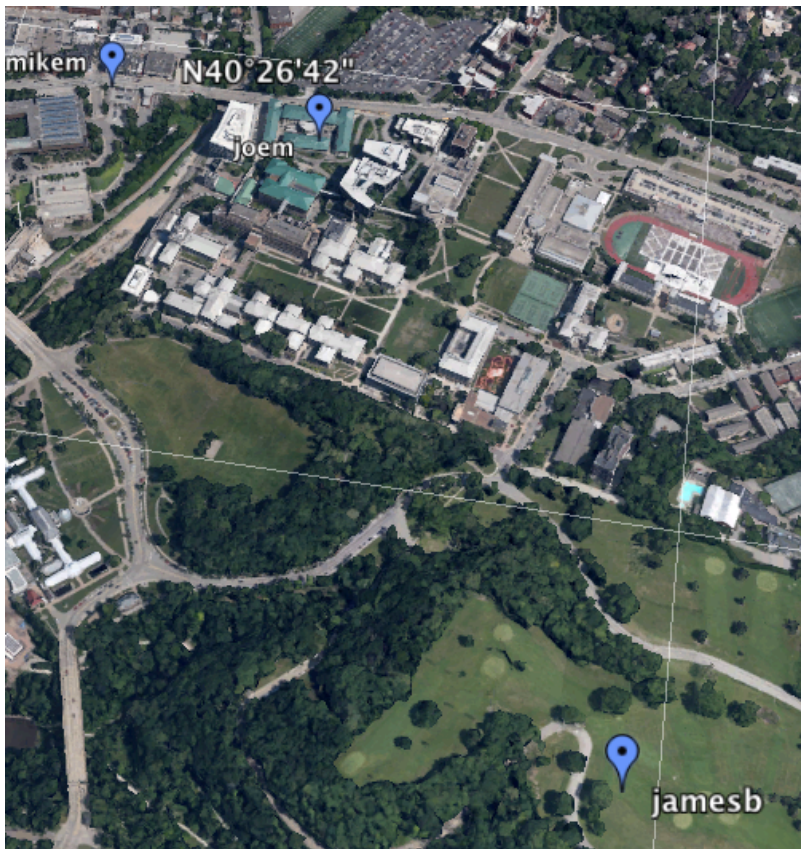
After each visit, the server writes a file called `SecretAgents.kml`. Here is a copy of a typical KML file that may be loaded into Google Earth.

SecretAgents.kml

```
<?xml version="1.0" encoding="UTF-8" ?>
<kml xmlns="http://earth.google.com/kml/2.2"
><Document>
  <Style id="style1">
    <IconStyle>
      <Icon>
        <href>http://maps.gstatic.com/intl/en_ALL/mapfiles/ms/micons/blue-
dot.png</href>
      </Icon>
    </IconStyle>
  </Style><Placemark>
    <name>jamesb</name>
    <description>Spy</description>
    <styleUrl>#style1 </styleUrl>
    <Point>
      <coordinates>-79.940450,40.437394,0.0000</coordinates>
    </Point>
  </Placemark>
  <Placemark>
    <name>joem</name>
    <description>Spy</description>
    <styleUrl>#style1 </styleUrl>
    <Point>
      <coordinates>-79.945389,40.444216,0.00000</coordinates>
    </Point>
  </Placemark>
  <Placemark>
```

```
<name>mikem</name>
<description>Spy</description>
<styleUrl>#style1 </styleUrl>
<Point>
<coordinates>-79.948460,40.444501,0.00000</coordinates>
</Point>
</Placemark>
</Document>
</kml>
```

When loaded, SecretAgents.kml looks like this in Google Earth.



Note that Joe is at work in Hamburg Hall, Mike is hanging out at Starbucks and James Bond is golfing.

You are required to rewrite the entire file (SecretAgents.kml) after each visit from a spy. The file always contains data on all three spies. This means that you need to maintain the state on the server for each spy. This would include location data, ID, password, and a title for display on Google Earth (see the Description element in the KML file).

If a visitor does not have the correct ID or password, no change will be made to the SecretAgents.kml file. See the error message on the client side shown for Mallory's attempt to enter a new location for Joe.

You may assume that the location data is accurate and well formed. That is, you do not have to validate the longitude, latitude, or altitude. The spies are always careful to enter the correct data.

Initially, before any spy has communicated with the server using TEA and TCP, all of the spies have their initial state stored in memory objects on the server. How you do this is of your own design. Each spy is initially located in Hamburg Hall (see Joe's coordinates in the KML file).

As soon as the first spy visits using TEA and TCP, the SecretAgents.kml file is re-written with that spy's new location. The other values (for the other spies still located in Hamburg Hall) are also re-written to the file. Thus, the file should always have data for all three spies (in Hamburg Hall or not).

From the Spy Commanders point of view, he runs the server and leaves it running all day and all night. On occasion, perhaps every few hours, he loads the SecretAgents.kml file into Google Earth to see where his spies are located. We are not writing an automatic refresh into Google Earth (maybe next term).

See Wikipedia and see the course schedule for a copy of TEA.java (which you may use.) Name this project Project2Task4. It will contain the files:

TCPSpyUsingTEAandPasswords.java and

TCPSpyCommanderUsingTEAandPasswords.java

TEA.java. Other files may be included as needed.

In my solution, since I am reading and writing streams of bytes, I did not use writeUTF() and readUTF(). Instead, I used these methods in DataInputStream and DataOutputStream:

```
public final int read(byte[] b)
public void write(byte[] b)
```

The return value of the read method came in very handy.

Project 2 Summary

Be sure to review the grading rubric on the schedule. We will use that rubric in evaluating this project. Documentation is always required.

There will be 4 projects in Netbeans.

- Project2Task1
- Project2Task2
- Project2Task3
- Project2Task4

You should also have four screen shot folders:

- Project2Task1ScreenShots
- Project2Task2ScreenShots
- Project2Task3ScreenShots
- Project2Task4ScreenShots

Copy all of your Netbeans project folders and screenshot folders into a folder named with your id.

Zip that folder and submit it to Blackboard.

The submission should be a single zip file.