SECURE DATA TRANSFER BETWEEN CLIENT AND SERVER

Esha Ahuja	071444
Varun Sareen	071607
Malvika Chauhan	071426
Ashish Yadav	071281

Project head: **S.P Ghrera**
Internal head: **P.K Tripathi**

SUBMITTED IN PARTIAL FULFILLMENT OF THE DEGREE OF BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY (JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT)

INDEX

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT, SOLAN-173215 CERTIFICATE

This is to certify that the preliminary project report entitled

"SECURE DATA TRANSFER BETWEEN CLIENT AND SERVER"

Submitted By:

Esha Ahuja	**071444**
Varun Sareen	**071607**
Malvika Chauhan	**071426**
Ashish Yadav	**071281**

This is a bonafide work carried out by us under the supervision of our project Head **Prof H.O.D CSE S.P Ghrera** and internal head **P.K Tripathi** and it is approved for the partial fulfillment of the requirement of University of Jaypee University of Information Technology for the award of the degree of Bachelor of Engineering (Computer Engineering and Information Technology)

Prof. : S.P Ghrera
Internal Guide: P.K Tripathi
Department of Computer Engineering

Head: S.P Ghrera
Department of Computer Engineering

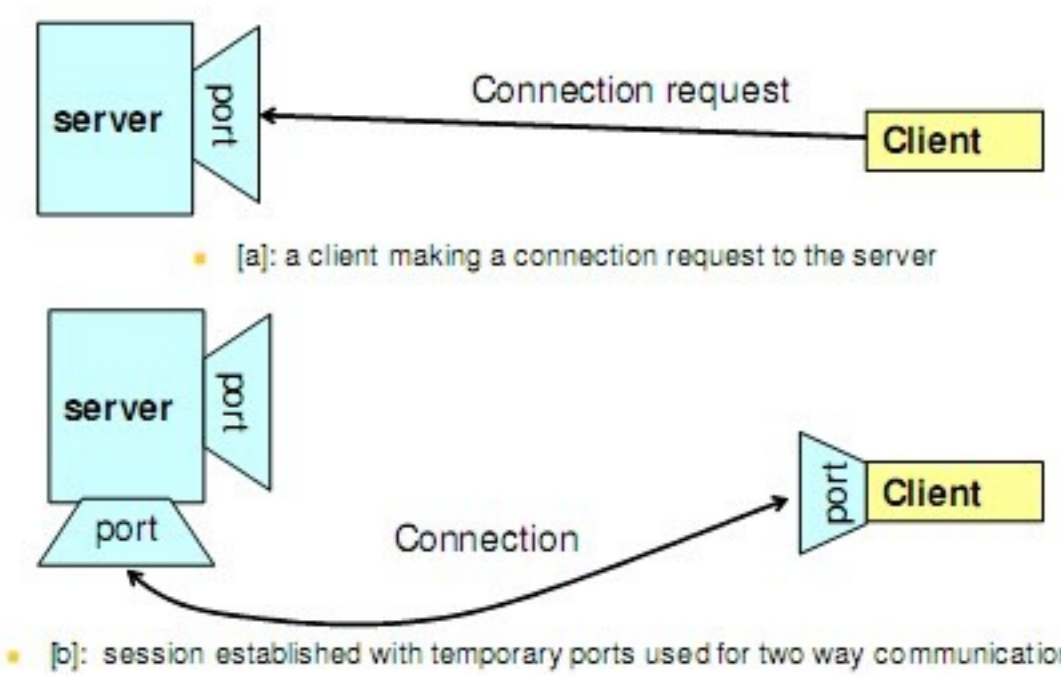
Name: _____
Signature :_____
External Examiner: _____
Place : Waknaghat
Date :_____

ACKNOWLEDGEMENT

It gives us great pleasure in presenting the project report for our project on ‘SECURE DATA TRANSFER BETWEEN CLIENT AND SERVER’. We would like to take this opportunity to thank our internal guide Prof. P.K Tripathi for giving us all the help and guidance we needed. We are really grateful to him for him kind support throughout the analysis and design phase. We are also grateful to Project head, Prof S.P. Ghrera Head of Computer science Department, Jaypee University of Information Technology and other staff members for giving important suggestions.

ABSTRACT

Disclosed are a method, a client/server system and a computer program for the secure data transfer from a server system to a client system that runs remotely an application on the server. The data transfer control is affected by defining a trigger event in the client system. The client system and the server are arranged so that an occurrence of a first trigger event initiates the data transfer of from client computer to the server system.



1.INTRODUCTION

The TCP/IP protocol suite as we know it today was developed in the late 1970s and early 1980s, with the watershed event probably the publishing of the version 4 standards of the Internet Protocol and Transmission Control Protocol in 1980. Modern TCP/IP was the result of experimentation and development work that had been underway since the 1960s. This work included both the design and implementation of the protocols that would implement internetworks, and also the creation of the first networking applications to allow users to perform different tasks.

What is a file transfer?

A file transfer is a procedure that allows you to move information from one computer system to another. When you use Procomm Plus to connect one computer to another, one way of exchanging information between the two computers is by typing on your keyboard. A file transfer takes the concept of communicating with another computer system a step further.

A file transfer will allow you to move files stored on the hard drive of a computer (or on floppy disks, CD ROM drives, for example,) to the system to which you are connected. Any file you can store on your disk drive can be sent to the other computer, including text documents, graphics, and actual programs. You can also retrieve the same types of files from another computer and save them on your disk drive.

Socket Communication

In computer networking, an Internet socket or network socket is an endpoint of a bidirectional inter-process communication flow across an Internet Protocol-based computer network, such as the Internet.

The term Internet sockets is also used as a name for an application-programming interface (API) for the TCP/IP protocol stack, usually provided by the operating system.

Internet sockets constitute a mechanism for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote IP addresses and port numbers. Each socket is mapped by the operating system to a communicating application process or thread.

A socket address is the combination of an IP address (the location of the computer) and a port (which is mapped to the application program process) into a single identity, much like one end of a telephone connection is the combination of a phone number and a particular extension.

6.1 Communicating through Sockets

In this application of file transfer we are using point-to-point communication therefore we don't need a dedicated FTP server, as we will be receiving file from a single source. Everything else works the same except for the use of headers we use hand shaking with the server. In the given scenario we create listening socket on the server side application and when a client connects to it, the file is transferred over the socket is created.

6.2 Internet socket types

Datagram sockets, also known as connectionless sockets, which use User Datagram Protocol (UDP)

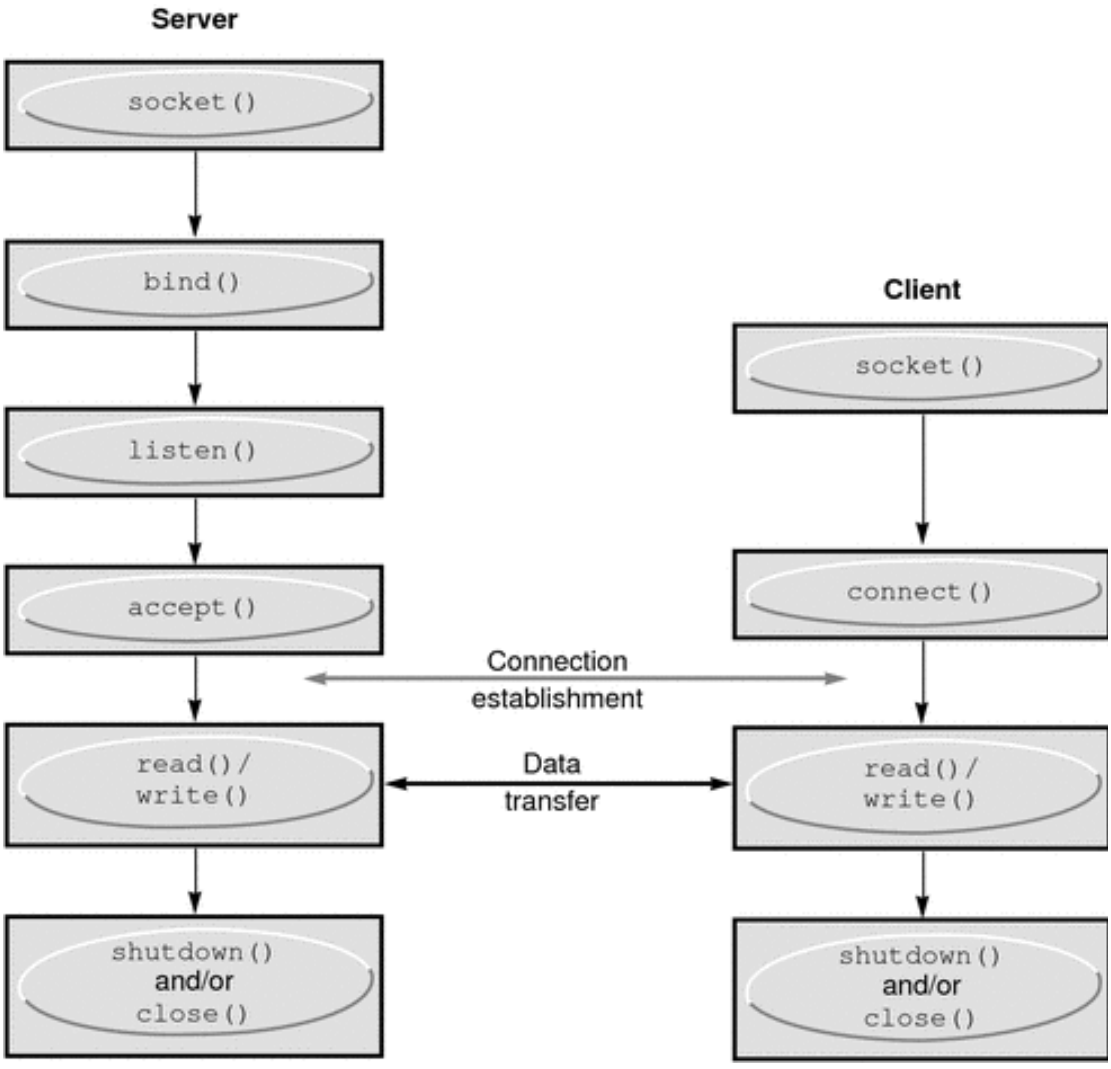
Stream sockets, also known as connection-oriented sockets, which use Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP).

Raw socket, Here the transport layer is bypassed, and the packet headers are not stripped off, but are accessible to the application. Application examples are

Internet Control Message Protocol (ICMP, best known for the Ping sub operation), Internet Group Management Protocol (IGMP), and Open Shortest Path First (OSPF).

In this application we are using Stream Sockets because they are connection-oriented sockets and make connection secure and much more reliable.

SOCKET COMMMUNICATION DIAGRAM



1.1 Project Idea:

Using Socket Communication we will be simulating data transfer from one remote machine to another remote machine based on concept of Client- Server model. In which one machine will act as client machine who will be connecting to another computer which act as Server listening on a defined port

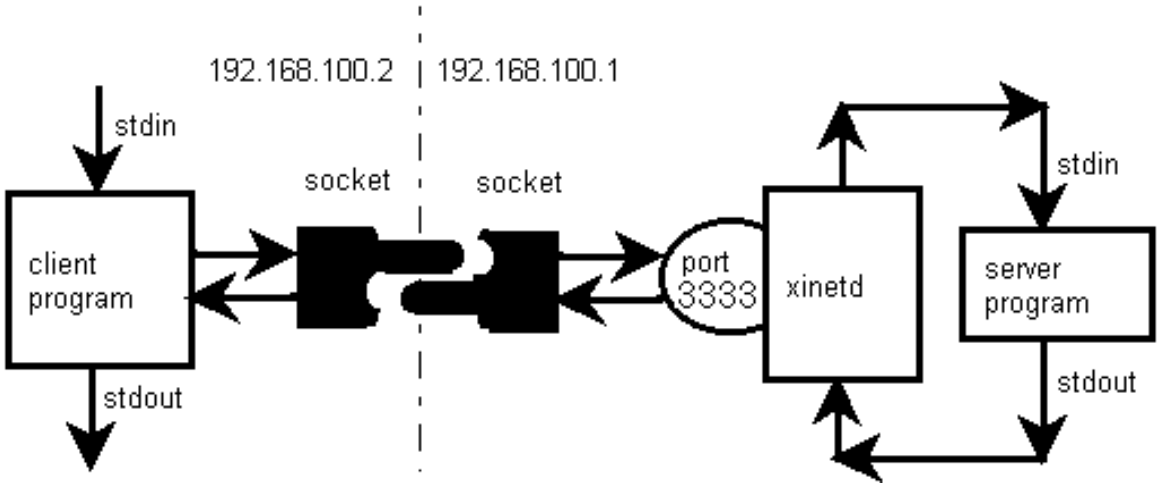
1.2 Need of the project:

When you want to copy files between two computers that are on the same local network, often you can simply "share" a drive or folder, and copy the files the same way you would copy files from one place to another on your own PC.

What if you want to copy files from one computer to another that is halfway around the world? You would probably use your Internet connection. However, for security reasons, it is very uncommon to share folders over the Internet. File transfers over the Internet use special techniques, of which one of the oldest and most widely used is FTP. FTP, short for "File Transfer Protocol, "or Socket Communication ,can transfer files between any computers that have an Internet connection, and also works between computers using totally different operating systems.

Transferring files from a client computer to a server computer is called "uploading" and transferring from a server to a client is "downloading"

.



1.3 Background

Socket Connections

Scoket uses one connection for commands and the other for sending and receiving data. Socket communication has no standard port number so it can utilize any available port. A port is a "logical connection point" for communicating using the Internet Protocol (IP).

Communicating local and remote sockets are called socket pairs. Each socket pair is described by a unique 4-tuple consisting of source and destination IP addresses and port numbers, i.e. of local and remote socket addresses.[5][6] As seen in the discussion above, in the TCP case, each unique socket pair 4-tuple is assigned a socket number, while in the UDP case, each unique local socket address is assigned a socket number.

Addresses and Ports

Today, sockets are typically used in conjunction with the Internet protocols -- Internet Protocol, Transmission Control Protocol, and User Datagram Protocol (UDP). Libraries implementing sockets for Internet Protocol use TCP for streams, UDP for datagrams, and IP itself for raw sockets.

To communicate over the Internet, IP socket libraries use the IP address to identify specific computers. Many parts of the Internet work with naming services, so that the users and socket programmers can work with computers by

name (e.g., "thiscomputer.compnetworking.about.com") instead of by address (e.g., 208.185.127.40). Stream and datagram sockets also use IP port numbers to distinguish multiple applications from each other. For example, Web browsers on the Internet know to use port 80 as the default for socket communications with Web servers.

Socket Implementation Issues

These are examples of functions or methods typically provided by the API library[7]:

socket() creates a new socket of a certain socket type, identified by an integer number, and allocates system resources to it.

bind() is typically used on the server side, and associates a socket with a socket address structure, i.e. a specified local port number and IP address.

listen() is used on the server side, and causes a bound TCP socket to enter listening state.

connect() is used on the client side, and assigns a free local port number to a socket. In case of a TCP socket, it causes an attempt to establish a new TCP connection.

accept() is used on the server side. It accepts a received incoming attempt to create a new TCP connection from the remote client, and creates a new socket associated with the socket address pair of this connection.

send() and recv(), or write() and read(), or recvfrom() and sendto(), are used for sending and receiving data to/from a remote socket.

close() causes the system to release resources allocated to a socket. In case of TCP, the connection is terminated.

gethostbyname() and gethostbyaddr() are used to resolve host names and addresses.

select() is used to prune a provided list of sockets for those that are ready to read, ready to write or have errors

poll() is used to check on the state of a socket. The socket can be tested to see if it can be written to, read from or has errors.

Point-to-Point Communication

In a nutshell, a socket represents a single connection between exactly two pieces of software. More than two pieces of software can communicate in client/server or distributed systems (for example, many Web browsers can simultaneously communicate with a single Web server) but multiple sockets are required to do this. Socket-based software usually runs on two separate computers on the network, but sockets can also be used to communicate locally (interprocess) on a single computer.

Sockets are bidirectional, meaning that either side of the connection is capable of both sending and receiving data. Sometimes the one application that initiates communication is termed the client and the other application the server, but this terminology leads to confusion in non-client/server systems and should generally be avoided.

2. PROBLEM DEFINITION AND SCOPE

2.1 Problem Statement

Data transfer between a client and a server is a very common and useful aspect in a networking system. Although there is no guarantee about the security of the data that is being transferred. The project will provide the security angle to the data transfer between the client and the server. The project will secure the data that is being transferred between the client and the server.

2.1.1 Goals

Initially any data transfer method like FTP or Socks communication was not designed to encrypt its traffic; all transmissions are in clear text, and user names, passwords, commands and data can be easily read by anyone able to perform packet capture (sniffing) on the network. This problem is common to many Internet Protocol specifications (such as SMTP, Telnet, POP and IMAP) designed prior to the creation of encryption mechanisms such as TLS or SSL[2]. A common solution to this problem is use of the "secure", TLS-protected versions of the insecure protocols (e.g. FTPS for FTP, Telnets for Telnet, etc.) or selection of a different, more secure protocol that can handle the job, such as the SFTP/SCP tools included with most implementations of the Secure Shell protocol.

2.1.2 Objectives

- Infrastructure setup for Client/ Server Model
- Client initiates the connection with the server
- Server Authenticates the client and accepts the connection request
- Once connection is maintained initiate the data transfer

2.2 Statement of scope

A description of the software with processing functionality described without regard to implementation detail. The functionality of the project is divided into 3 parts:-

1. Connection between Client and Server
2. Authentication between Client and Server
3. Data Transfer between remote and host

2.3 Major constraints:

1. Internet bandwidth can act as a bottleneck.
2. Design:
   - The system is not perfectly accurate for huge file transfer and it is not been designed for very high confidential data
3. Implementation:
   - The major constraint will be to achieve maximum security.
4. Testing:
   - Whether the system will be able to transfer all the different types file between remote and host machine accurately in a secure mode.

2.4 Hardware Resources Required

1. Pentium Processor PIII/IV

2. 2 GB RAM.
3. 10 GB HDD.
4. Power Supply (5V).

Minimum System Requirements:-

1. Main processor: Pentium 3
2. RAM: 1 GB
3. DISK capacity: 2.5 GB HDD
4. DISK drives: 1.44 Mb FDD.52 X CD-ROM drive

For Best Performance:-

1. Main processor: Pentium 4
2. RAM: 2 GB
3. DISK capacity: 10 GB HDD
4. DISK drives: 1.44 Mb FDD.52 X CD-ROM drive

2.5 Software Resources Required

1. Windows operating system (windows 98/2000/XP)

2. Java, J2EE

Front End:

Eclipse refers to both a platform for the development of applications for the network (using Java, JavaScript, PHP, Python, Ruby, Groovy, C, and C++), and an integrated development environment (IDE) developed using the Eclipse Platform.

The Eclipse Platform allows applications to be developed from a set of modular software components called modules. A module is a Java archive file that contains Java classes written to interact with the Eclipse Open APIs and a manifest file that identifies it as a module. Applications built on modules can be extended by adding

new modules Eclipse Helios IDE extended the existing Java EE features (including Java Persistence support, EJB 3 and JAX-WS). Additionally, the Eclipse Enterprise Pack supports development of Java EE 5 enterprise applications, including SOA visual design tools, XML schema tools, web services orchestration and UML modeling.

3. Project Plan

3.1 Introduction

3.1.1 Problem Definition

To simulate the data transfer between a client and a server is a very common and useful aspect in a networking system. Although there is no guarantee about the security of the data that is being transferred. The project will provide the security angle to the data transfer between the client and the server. The project will secure the data that is being transferred between the client and the server.

3.1.2 Management and technical constraints

1. Internet bandwidth can act as a bottleneck.

2. Design:

   - The system is not perfectly accurate for huge file transfer and it is not been designed for very high confidential data

3. Implementation:

   - The major constraint will be to achieve maximum security.

4. Testing:

   - Whether the system will be able to transfer all the different types file between remote and host machine accurately in a secure mode

3.1.3 Purpose of the Document

- To simulate the data transfer between remote and host in secure mode

3.2 Project Resources

People, hardware, software, tools, and other resources required to build the software.

People: 4 Persons.

Hardware: One IBM Compatible PC, 1 GB RAM, 10 GB hard disk

Software: Windows 98/2000/XP Operating System, Java.

3.3 Risk Management

Risk is a possibility of loss or injury. The definition of risk in the software engineering environment that we will use is exposure to harm or loss, as this includes not only the possibility of risks, but their impact as well. Using risk management techniques, we alleviate the harm or loss in a software project. All risks cannot be avoided, but by performing risk management, we can attempt to ensure that the right risks are taken at the right time. "...Risk taking is essential to progress and, and failure is often a key part of learning".

Software Risk Management is an iterative process. About two iterations are both feasible and useful. We use the risk table to identify risks and briefly describe them.

We have classified the risks into different categories. They are :

1. Technical

2. Requirement
3. Design
4. Implementation
5. Post Release
6. Business Risks

3.3.1 Risk Table

No.	Risk	Category	Probability
1	Changes in Design regarding choice of underlying data structure.	Technical – Design	30%
2	Lack of communication among team members due to clashes in schedule	Project	5%
3	Ambiguous Requirements that may change	Project	60%
4	Performance. (Something taking too long or too heavy on resources)	Technical	40%
5	Release of a similar product by another team	Business	10%
6	Experience with development language/ platform/ tools	Project	20%

3.3.2 Overview of Risk Mitigation, Monitoring, Management

Project scope is vast with limited time, and disaster can be taken care of by risk avoidance using a proactive approach. This can be done by developing a risk mitigation plan.

Small staff size and staff inexperience can be taken care of in risk monitoring stage by the project group members having a good relationship with one another. The members jell with each other well and there must be proper co-ordination among the team members. Also by arranging meetings with people who are experienced in the field and have complete knowledge about the field will help us.

3.4 Project Schedule

Sr.	Name of	Subtask	
1	**Problem Identification and Information Gathering**	**1. Problem Definition:** • Collecting detailed problem definition of the system to be implemented	21/07/2010 To 02/08/2010
		2. Literature Survey: • Visiting different websites. • Studying existing system with it's limitation • Going through Journals, magazines • Studying the reference books	03/08/2010 To 31/08/2010
2	**Analysis**	**1. Project Plan:** • Preparing for complete project plan	1/09/2010 To 15/09/2010

		2. Requirement Analysis • Software requirements • Hardware requirements • Databases	16/09/2010 To 30/09/2010
3	**Design**	**1. Architectural design:** • Describing relationships between modules and sub modules	1/10/2010 To 05/10/2010
		2. UML documentation: • Use case diagrams • Sequence diagrams	06/10/2010 To 15/10/2010
		3. Form Designs: • Showing relationship among different menus and submenus	16/10/2010 To 30/10/2010
4	**Output Screen formats**	**Output Screens:** • Preparing for detailed output screens describing output formats	1/11/2010 To 4/11/2010

		Report submission: • Submitting reports of Analysis and Design	5/11/2010 To 6/11/2010
5	**Development**	**1. Coding:** • Implementing design details Using programming language Java.	7/11/2010 To 22/11/2010
7	**Testing**	Testing the system for expected results	22/11/2010 To 25/11/2010
6	**Deployment of system**	**System deployment:** • Delivery of project • Support • Feedback	25/4/2010
7	**Final Document preparation and Submission**	**Project submission:** • Working on data encryption using AES • Working on SSL sockets and Java ciphers. • Preparing final project report • Submitting final project report	8th Semester

4. Software Requirement Specification

4.1 Purpose and Scope of Document

This system can authenticate the server and initiates the secure data transfer.

4.2 Usage scenario

This section provides a usage scenario for the software. It is the organized information collected during requirements elicitation into use-cases.

4.2.1 User profiles

Client- Anybody using the system.

Server- Listening on a particular port and authenticate the client request

4.2.2 Use-cases

1. Connect to the Server

2. Authenticate the Client Connection

3. Initiate the File Transfer

4. Accept the file data and Store the server share drive

4.2.3 Use Case View

A use case diagram is a type of behavioral diagram defined by the Unified Modeling Language (UML) and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

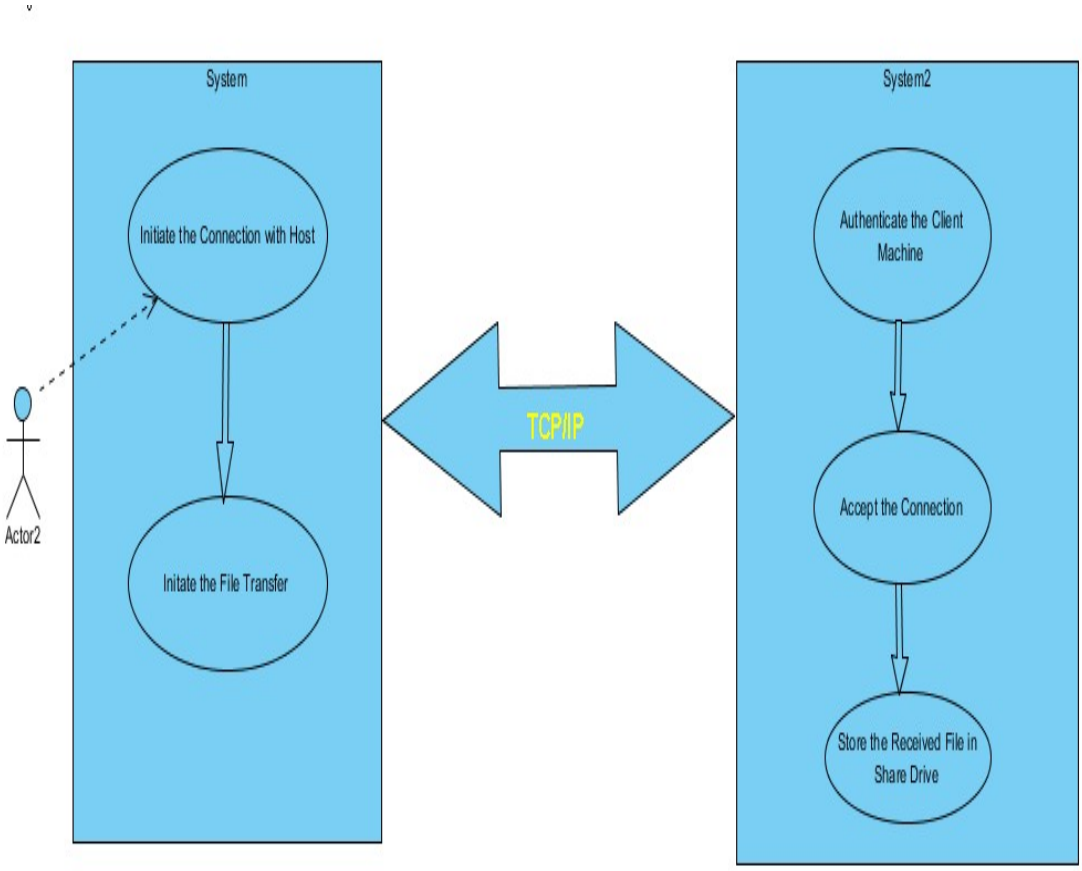


Figure 4.1: Use Case Diagram

4.3 Functional Model and Description

4.3.1 Physical aspects of Socket Communication

Java I/O Overview

Input and Output - Source and Destination

The terms "input" and "output" can sometimes be a bit confusing. The input of one part of an application is often the output of another. Is an OutputStream a stream where output is written to or output comes out from. After all, an Input Stream outputs its data to the reading program.

Java's IO package mostly concerns itself with the reading of raw data from a source and writing of raw data to a destination. The most typical sources and destinations of data are these:

- Files
- Pipes
- Network Connections
- In-memory Buffers (e.g. arrays)
- System.in, System.out, System.error

Java I/O Purposes and Features

The Java I/O classes, which mostly consists of streams and readers / writers, are addressing various purposes. That is why there are so many different classes. The purposes addressed are summarized below:

- File Access
- Network Access
- Internal Memory Buffer Access
- Inter-Thread Communication (Pipes)
- Buffering
- Filtering
- Parsing
- Reading and Writing Text (Readers / Writers)
- Reading and Writing Primitive Data (long, int etc.)
- Reading and Writing Objects

These purposes are nice to know about when reading through the Java IO classes. They make it somewhat easier to understand what the classes are targeting.

Java I/O Class Overview Table

Having discussed sources, destinations, input, output and the various IO purposes targeted by the Java IO classes, this text will finish off with a table of most (if not all) Java IO classes divided by input, output, being byte based or character based, and any more specific purpose they may be addressing, like buffering, parsing etc.

	Byte Based		Character Based	
	Input	Output	Input	Output
Basic	InputStream	OutputStream	Reader InputStreamReader	Writer OutputStream
Arrays	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
Files	FileInputStream RandomAccessFile	FileOutputStream RandomAccessFile	FileReader	FileWriter
Pipes	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
Buffering	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
Filtering	FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
Parsing	PushbackInputStream StreamTokenizer		PushbackReader LineNumberReader	
Strings			StringReader	StringWriter

Data	DataInputStream	DataOutputStream		
Data – Formatted		PrintStream		PrintWriter
Objects	ObjectInputStream	ObjectOutputStream		
Utilities	SequenceInputStream			

Java.io.InputStreamReader

• This class provides a character interface to input stream (also converts to unicode characters)

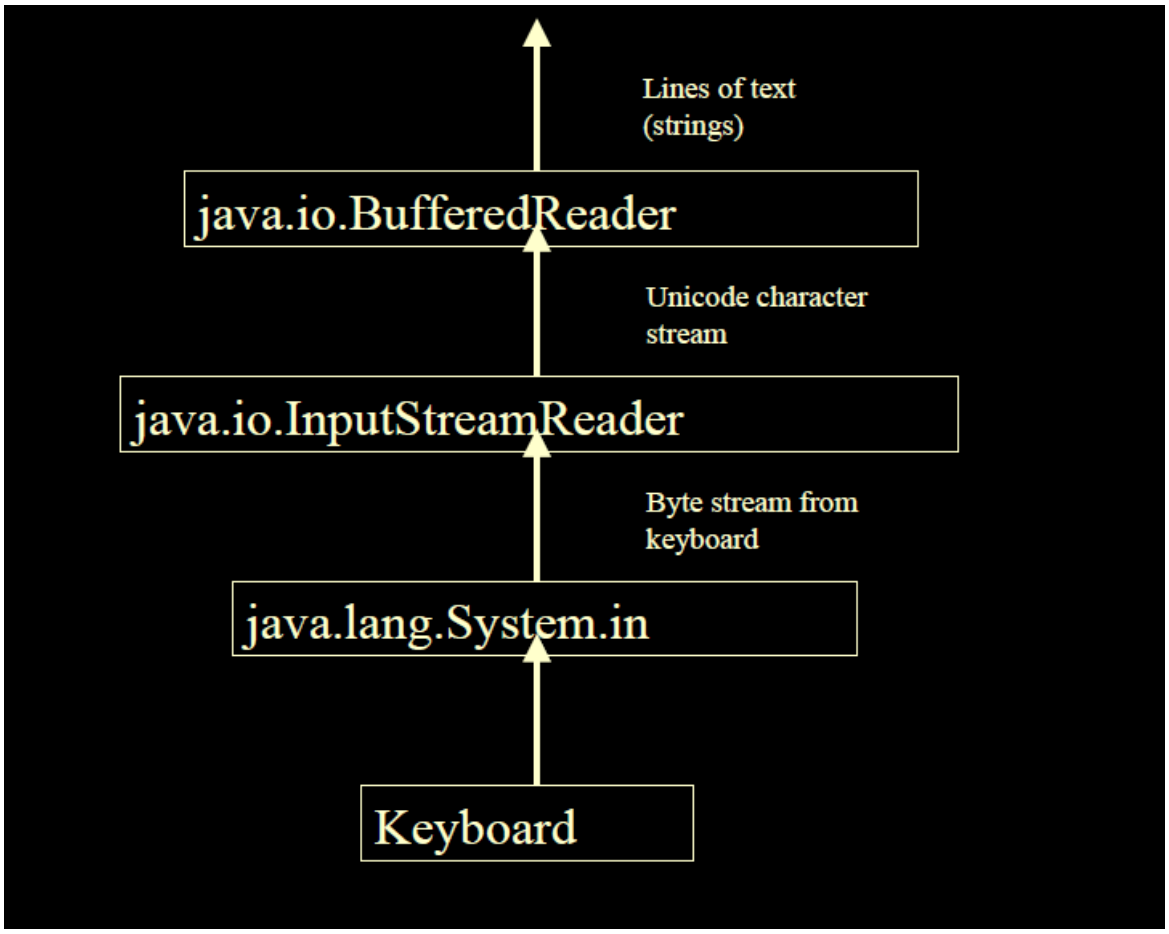
```
int cc;
InputStreamReader inkey;
Inkey=new InputStreamReader(System.in);
While(true) {
try {
cc = inkey.read();
} catch (IOException ex) {}
System.out.print(cc);
}
```

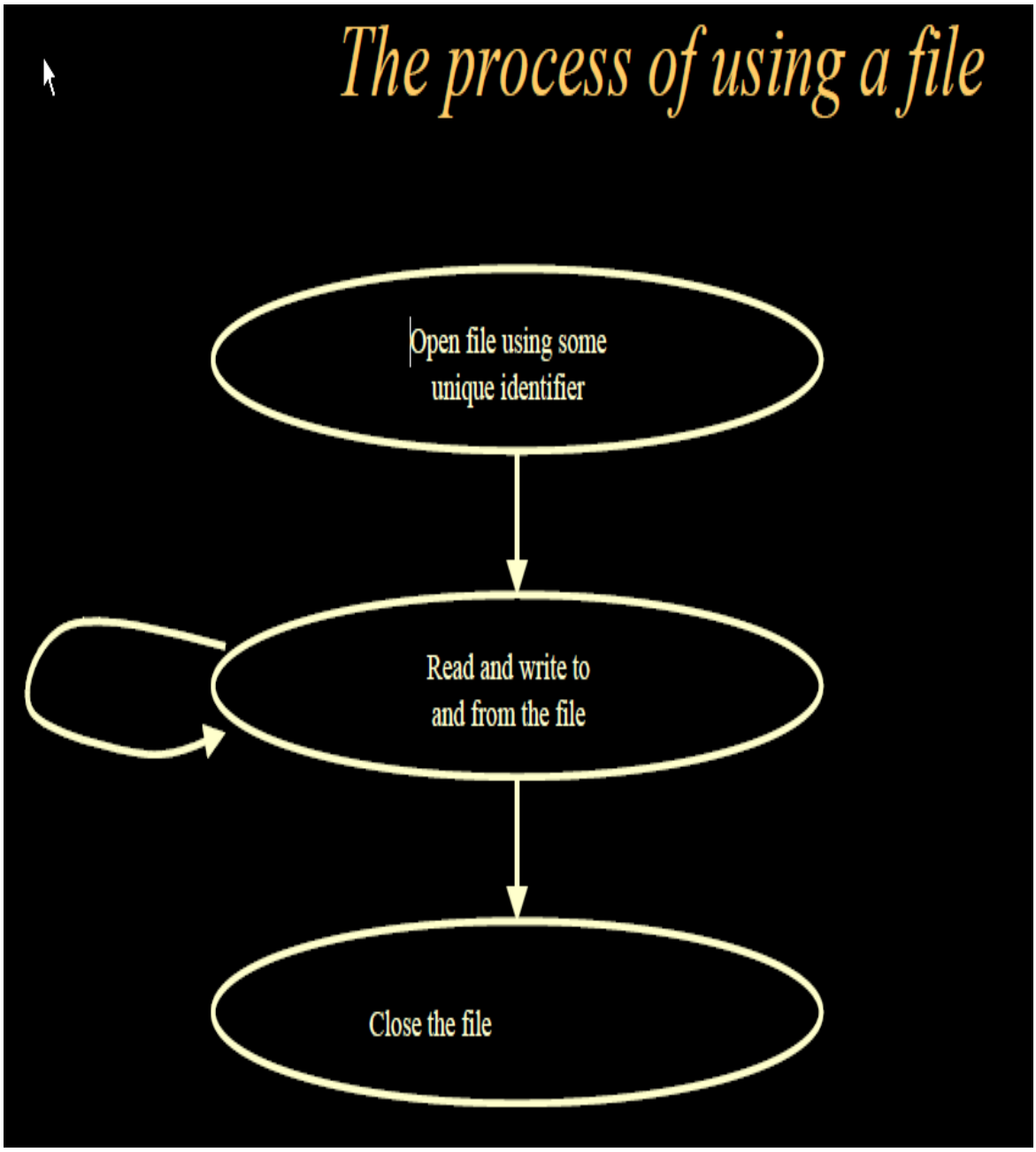
Java.Io.BufferedReader

• Provides a buffer for the input stream can be accessed on a line by line basis

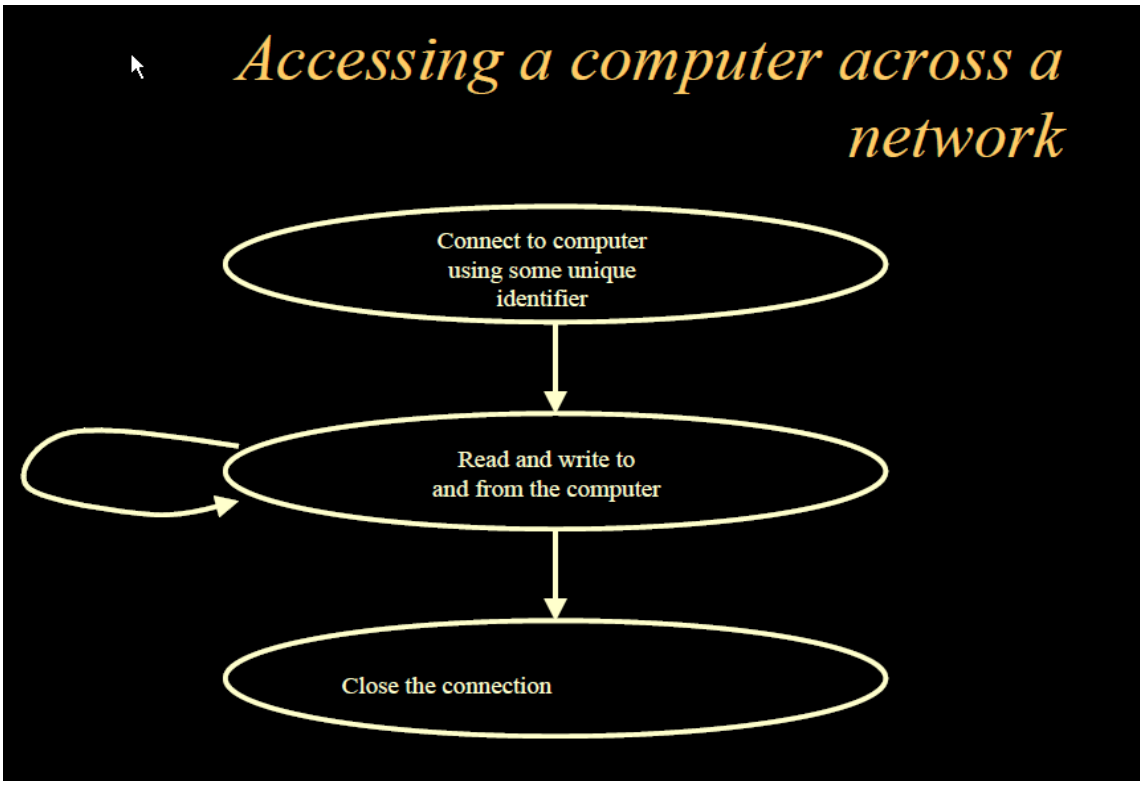
```
String s;
InputStreamReader inkey;
BufferedReader bufkey;
```

```
inkey=new InputStreamReader(System.in);
bufkey=new BufferedReader(inkey);
While(true) {
s=bufkey.readLine();
System.out.println(s);
}
```





The Process of using a File



Sockets – Connecting to other Computers

When connecting to another computer you use a ‘socket’.

Analogous to a ‘file handle’ used when accessing files

When opening a file you uniquely identify it by its file name. When connecting to a computer you uniquely identify it with its IP number

Addressing Computers

An IP number is four numbers (each between 0 and 255) separated by a '.' eg. Goblin's IP is 130.217.208.41However, because numbers are difficult to remember,the network provides a service that associates names with numbers. (www.yahoo.com:130.217.208.41)

Ports — connecting to programs on

Other computers over a network ysing a unique number we can identify a computer to connect to we identify which program to communicate with by using a port numberCommon networking programs (such as telnet, ftp and WWW services) are always on the same port. These ports are called "well known" like Telnet is on port 23, FTP on port 21, WWW services are on port 80, etc.

File /etc/services

- tcpmux 1/tcp # TCP port service multiplexer
- echo 7/tcp
- echo 7/udp
- discard 9/tcp sink null
- discard 9/udp sink null
- systat 11/tcp users
- daytime 13/tcp
- daytime 13/udp
- netstat 15/tcp
- qotd 17/tcp quote

- msp 18/tcp # message send protocol
- msp 18/udp # message send protocol
- chargen 19/tcp ttytst source
- chargen 19/udp ttytst source
- ftp-data 20/tcp # File Transfer [Default Data]
- ftp-data 20/udp # File Transfer [Default Data]
- ftp 21/tcp # File Transfer [Control]
- ftp 21/udp # File Transfer [Control]
- ssh 22/tcp # Secure Shell Login
- ssh 22/udp # Secure Shell Login
- telnet 23/tcp
- telnet 23/udp

Implementation of Socket Communication

Java's networking facilities are provided in the java.net package To make a connection to another machine we must first know its IP number

• InetAddress goblinsIP = InetAddress.getByName("???");

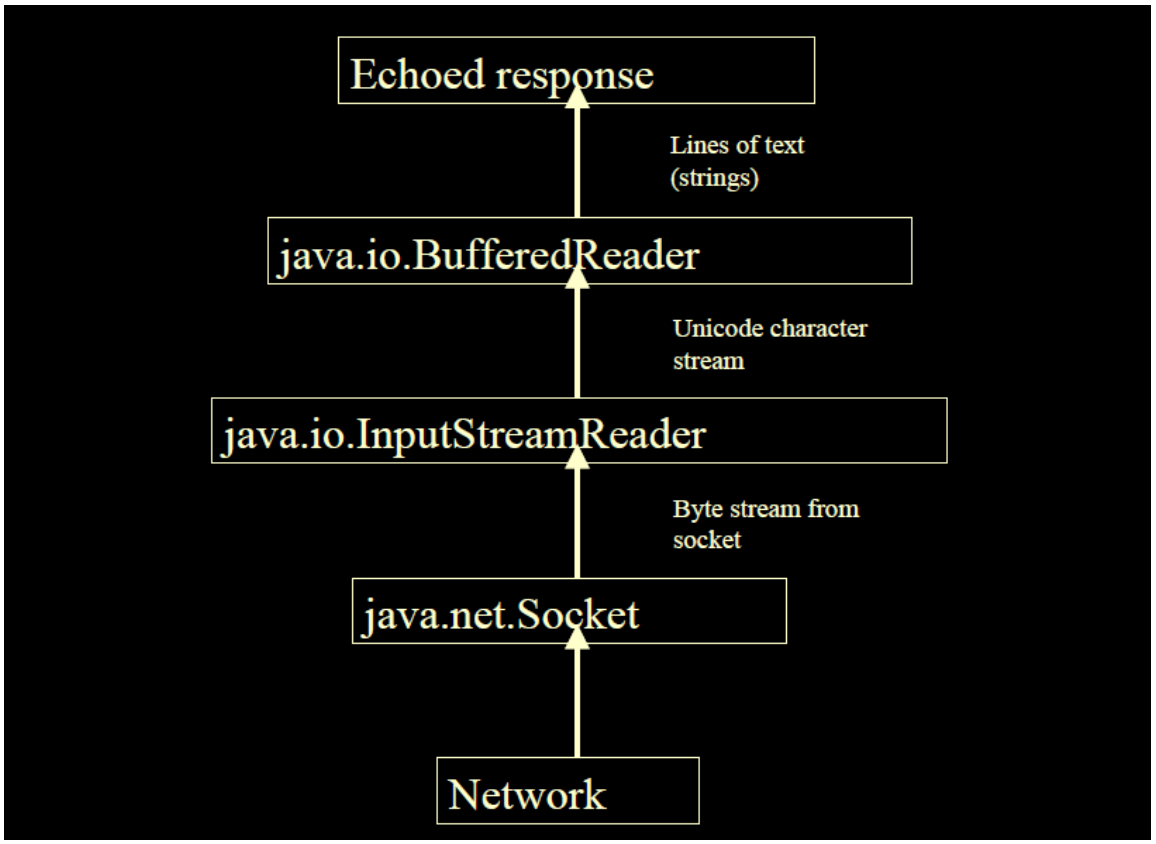
Now we can open a socket

• Socket mysocket = new Socket(????, 23);

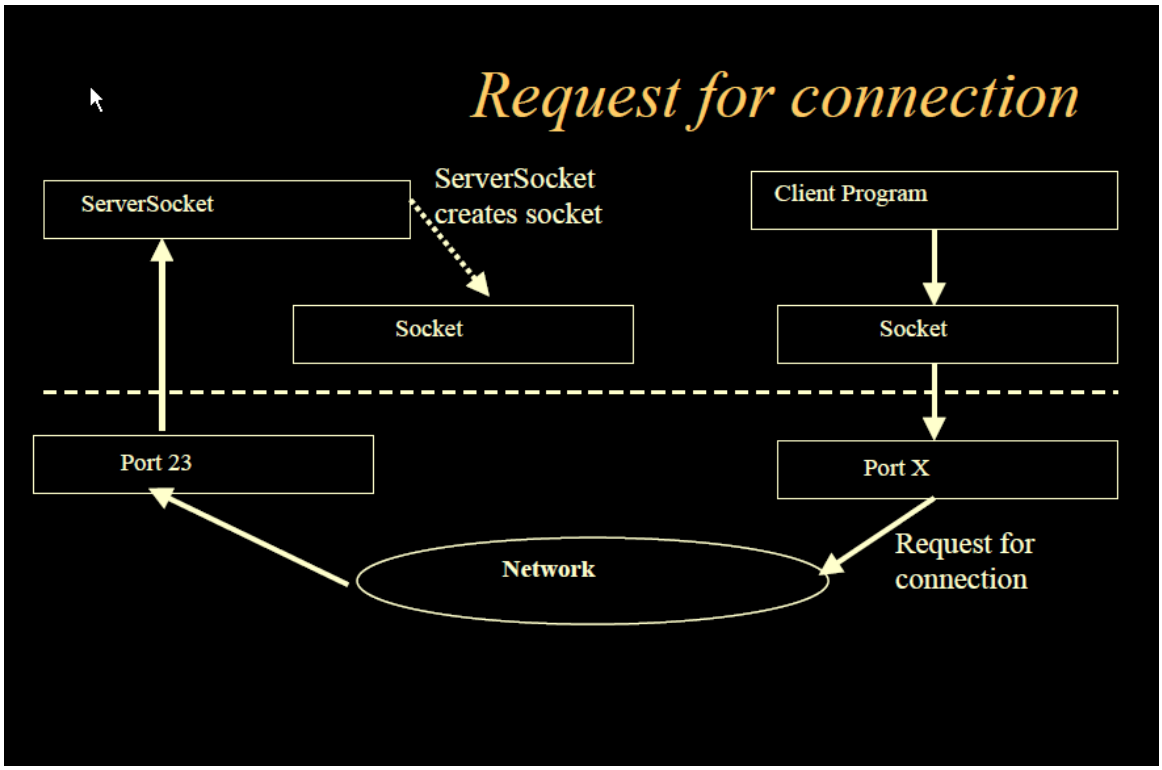
This would connect to the telnet port on and then following methods return input and output streams that you can use to read and write to the socket, just like a file

- mysocket.getInputStream();
- Mysocket.getOutputStream();
- InputStreamReader in = new InputStreamReader(mysocket.getInputStream());

• OutputStreamWriter out = new OutputStreamWriter(mysocket.getOutputStream());



Client Side Implementation



SENDER SIDE CODE:

```
package com.filetransfer.core;

import java.io.*;

import java.net.Socket;

public class FileSender extends FileTransfer

    implements Runnable

{

    public FileSender(Socket s, File f, StatusListener listener)

    {

        socket = s;

        file = f;

        this.listener = listener;

    }

    public void cancel()

    {

        cancel = true;

    }



    protected void setStatus(String status, int percent)

    {
```

```
    if(listener != null)
        listener.setStatus(status, percent);
}
public void send()
{
    (new Thread(this)).start();
}
public void run()
{
    try
    {
        OutputStream os = socket.getOutputStream();
        DataOutputStream dos = new DataOutputStream(os);
        dos.writeUTF(file.getName());
        dos.writeLong(file.length());
        FileInputStream fis = new FileInputStream(file);
        byte buffer[] = new byte[socket.getSendBufferSize()];
        int res = fis.read(buffer);
        long fileSize = file.length();
```

```
            long pos = 0L;
            for(; res > 0 && !cancel; res = fis.read(buffer))
            {
                dos.write(buffer, 0, res);
                incBytes(res);
                pos += res;
                 setStatus((newStringBuilder("Sending(")).append(getBytesString(pos)).
append("/").append(getBytesString(fileSize)).append(") ").append((pos * 100L) /
fileSize).append("% ").append(getTransferSpeed()).toString(), (int)((pos * 100L) /
fileSize));
}
dos.close();
            os.close();
          //  break MISSING_BLOCK_LABEL_255;
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        try
```

```
        {
            socket.close();
        }
        catch(IOException ioexception) { }
       //  break MISSING_BLOCK_LABEL_267;
        // Exception exception;
       // exception;
        try
        {
            socket.close();
        }
        catch(IOException ioexception1) { }
        //throw exception;
        try
        {
            socket.close();
        }
        catch(IOException ioexception2) { }
    }
```

```
public static void main(String args[])
   throws Exception
{
   if(args.length < 2)
   {
      System.out.println("Use <host> <file>");
      return;
   } else
   {
      Socket s = new Socket(args[0], 1923);
      File f = new File(args[1]);
      FileSender fileSender = new FileSender(s, f, null);
      fileSender.send();
      return;
   }
}

private Socket socket;
```

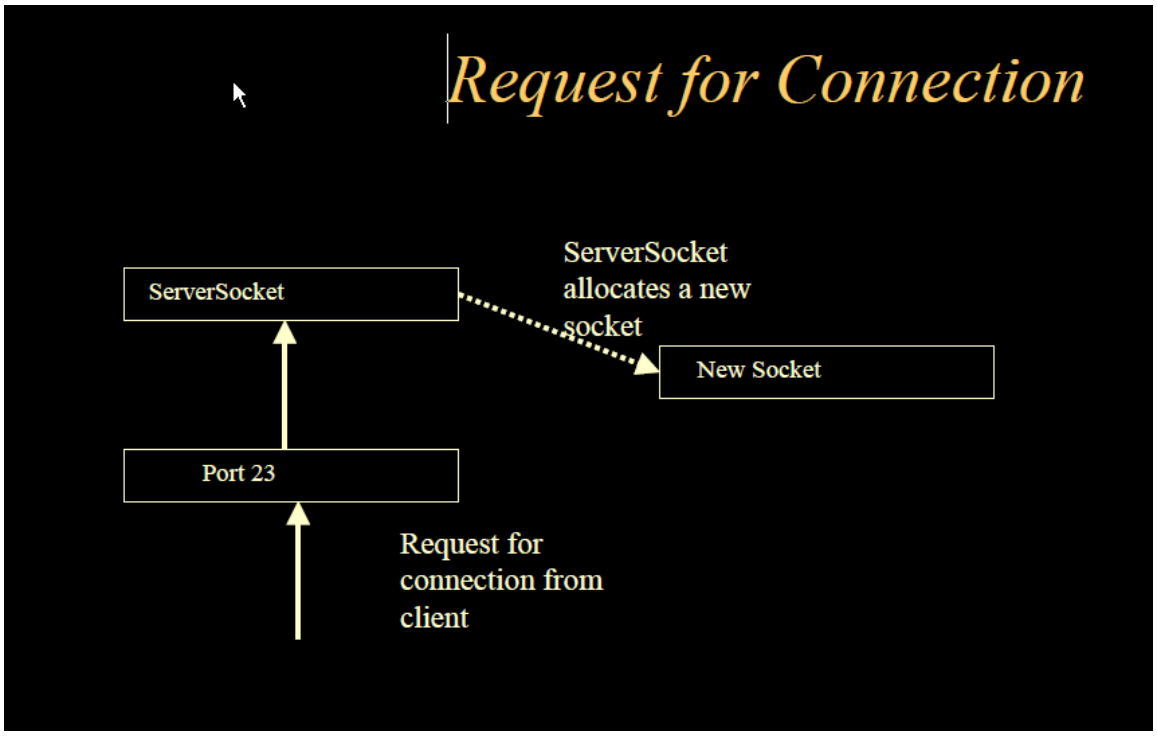
```
    private File file;

    private boolean cancel;

    private StatusListener listener;

}
```

Server Side Implementation


Request for Connection
ServerSocket
ServerSocket allocates a new socket
New Socket
Port 23
Request for connection from client

RECEIVER SIDE CODE:

```
package com.filetransfer.core;

import java.io.*;

import java.net.*;

import java.util.Date;

public class FileReceiver extends FileTransfer

 implements Runnable

{

 public FileReceiver(Socket s, StatusListener listener)

   {

this.s = s;

this.listener = listener;

   }   public void receive()

   {

      (new Thread(this)).start();

   }



  public void cancel()

   {
```

```
        cancel = true;
    }
    protected void setStatus(String status, int percent)
    {
        if(listener != null)
            listener.setStatus(status, percent);
    }
  public void run()
    {
        try
        {
            InputStream is = s.getInputStream();
            DataInputStream dis = new DataInputStream(is);
            String fileName = dis.readUTF();
            long fileSize = dis.readLong();
                System.out.println((new StringBuilder()).append(new Date()).append("
Receiving file: ").append(fileName).toString());
            FileOutputStream fout = new FileOutputStream(new File(fileName));
            byte buffer[] = new byte[s.getReceiveBufferSize()];
```

```
            int res = dis.read(buffer);
            long pos = 0L;
            for(; res > 0 && !cancel; res = dis.read(buffer))
            {
                incBytes(res);
                fout.write(buffer, 0, res);
                pos += res;
                                                setStatus((new StringBuilder("Receiving
("")).append(getBytesString(pos)).append("/").append(getBytesString(fileSize)).ap
pend(")  ").append((pos  *  100L)  /  fileSize).append("%
").append(getTransferSpeed()).toString(), (int)((pos * 100L) / fileSize));
            }
            fout.close();
            dis.close();
            is.close();
         //   break MISSING_BLOCK_LABEL_281;
        }
        catch(Exception e)
        {
            e.printStackTrace();
```

```
        }
        try
        {
           s.close();
        }
        catch(IOException ioexception) { }
            try
        {
           s.close();
        }
        catch(IOException ioexception1) { }
         try
        {
           s.close();
        }
        catch(IOException ioexception2) { }
    }
public static void main(String args[])
        throws Exception
```

```
    {
        ServerSocket ss = new ServerSocket(1923);
                        System.out.println((new StringBuilder("Listening on
")).append(Inet4Address.getLocalHost().getHostAddress()).append(":1923").toStr
ing());
        System.out.println("Waiting for connection...");
        Socket s = ss.accept();
            System.out.println((new StringBuilder()).append(new Date()).append("
Connected with ").append(s.getInetAddress().getHostAddress()).toString());
        FileReceiver fileReceiver = new FileReceiver(s, null);
        fileReceiver.receive();
    }

    private Socket s;
    private boolean cancel;
    private StatusListener listener;
}
```

STATUS LISTENER CODE:

```
package com.filetransfer.core;

public interface StatusListener

{

   public abstract void setStatus(String s, int i);

}
```

FILE TRANSFER CODE:

```
package com.filetransfer.core;

public class FileTransfer

{

   class Eraser extends Thread

   {



      public void run()

      {

         do
```

```
        {
            long beforeSleeping = 0L;
            try
            {
                synchronized(this)
                {
                    if(beforeSleeping == 0L)
                        history[pos] = bytes;
                    else
                        history[pos] = (bytes * (System.currentTimeMillis() -
beforeSleeping)) / 1000L;
                    bytes = 0L;
                    pos = (pos + 1) % history.length;
                    if(historySlots < history.length)
                        historySlots++;
                    int sum = 0;
                    for(int x = 0; x < historySlots; x++)
                        sum = (int)((long)sum + history[x]);
```

```
                    int average = sum / historySlots;
                    transferSpeed = (new
StringBuilder(String.valueOf(getBytesString(average)))).append("/s").toString();
                }
                beforeSleeping = System.currentTimeMillis();
                sleep(1000L);
            }
            catch(InterruptedException e)
            {
                e.printStackTrace();
            }
        } while(true);
    }

    final FileTransfer this$0;

    Eraser()
    {
        super();
```

```
        this$0 = FileTransfer.this;


    }
}



protected synchronized void incBytes(long bytes)
{
    this.bytes += bytes;
}


public synchronized String getTransferSpeed()
{
    return transferSpeed;
}
    protected String getBytesString(long bytes)
{
    int unit;
    for(unit = 0; bytes > 1024L; unit++)
```

```
        bytes /= 1024L;


    switch(unit)
    {
    case 0: // '\0'
        return (new StringBuilder(String.valueOf(bytes))).append(" B").toString();


    case 1: // '\001'
                return (new StringBuilder(String.valueOf(bytes))).append("
KB").toString();


    case 2: // '\002'
                return (new StringBuilder(String.valueOf(bytes))).append("
MB").toString();


    case 3: // '\003'
                return (new StringBuilder(String.valueOf(bytes))).append("
GB").toString();
    }
```

```
                return (new StringBuilder(String.valueOf(bytes))).append("
TooMuch").toString();
    }
     public FileTransfer()
    {
        history = new long[5];
        historySlots = 0;
        transferSpeed = "0 B/s";
        (new Eraser()).start();
    }
    private long bytes;
    private long history[];
    private int pos;
    private int historySlots;
    private String transferSpeed;
}
```

5. SOFTWARE DESIGN SPECIFICATION

This document describes the high level design for data, architecture, interface and component for the software.

5.1 Architectural Design

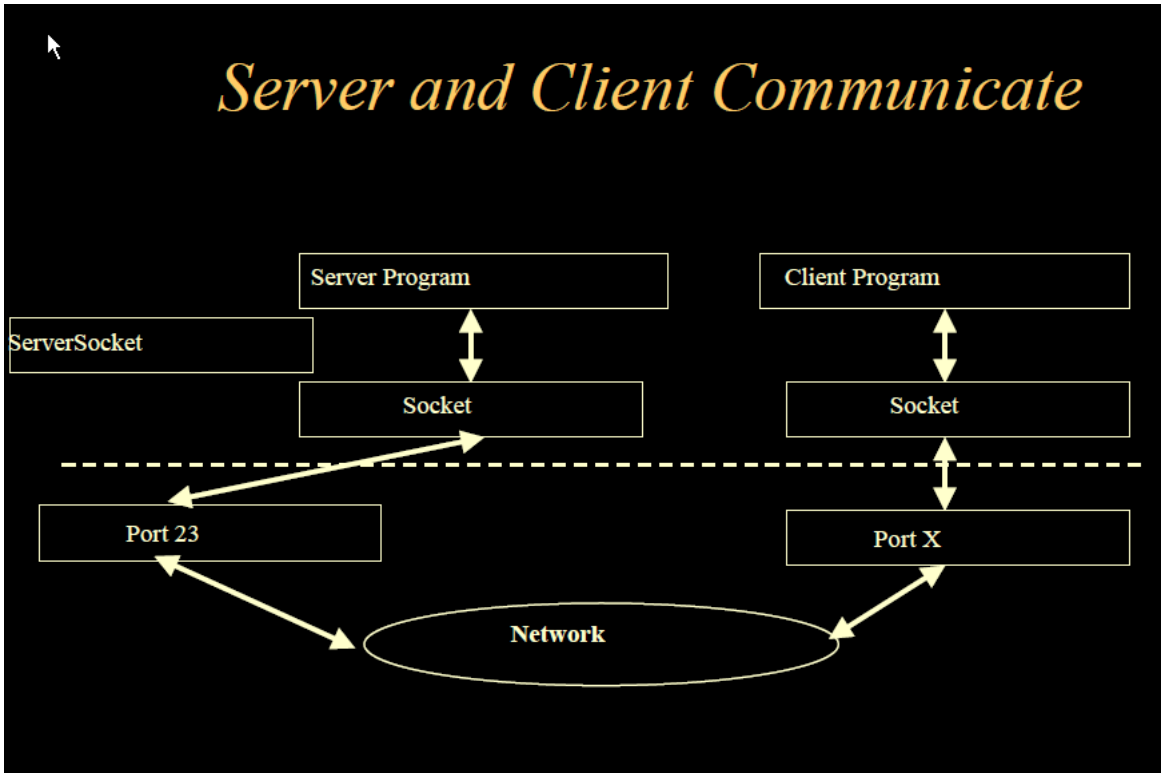


Figure 5.2:Block Diagram

\

5.2 Class Diagram

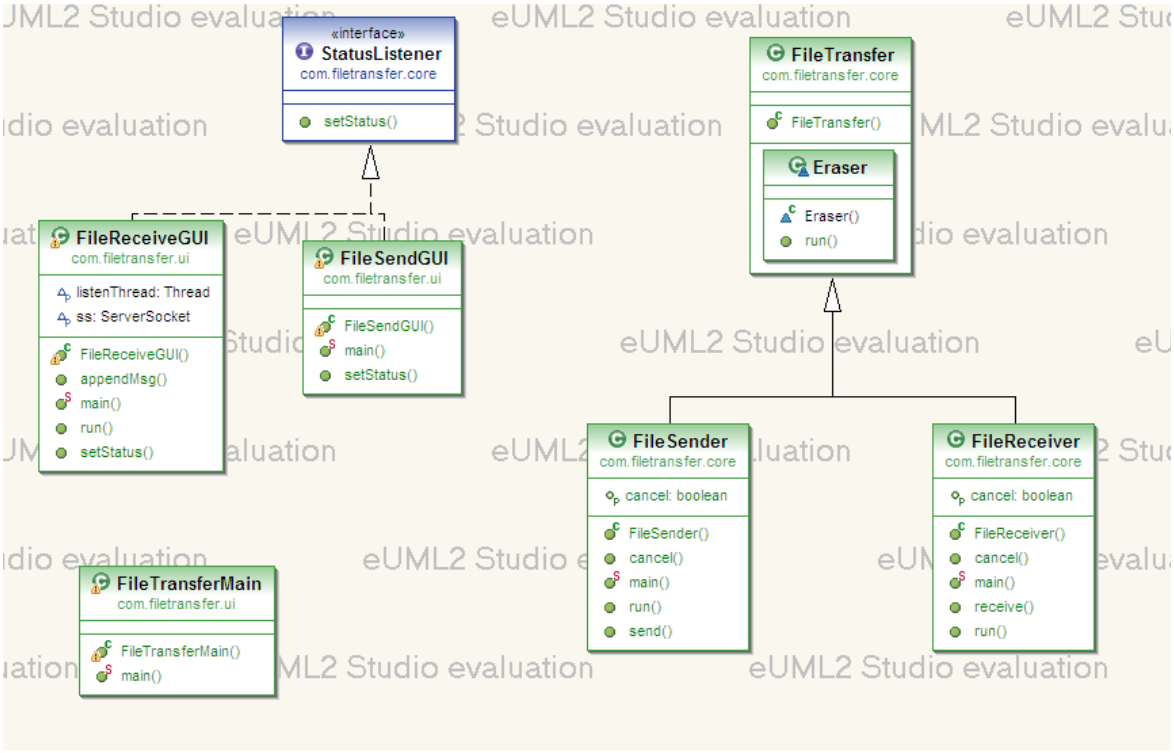


Figure 5.3:Class Diagram

5.3 Activity Diagram

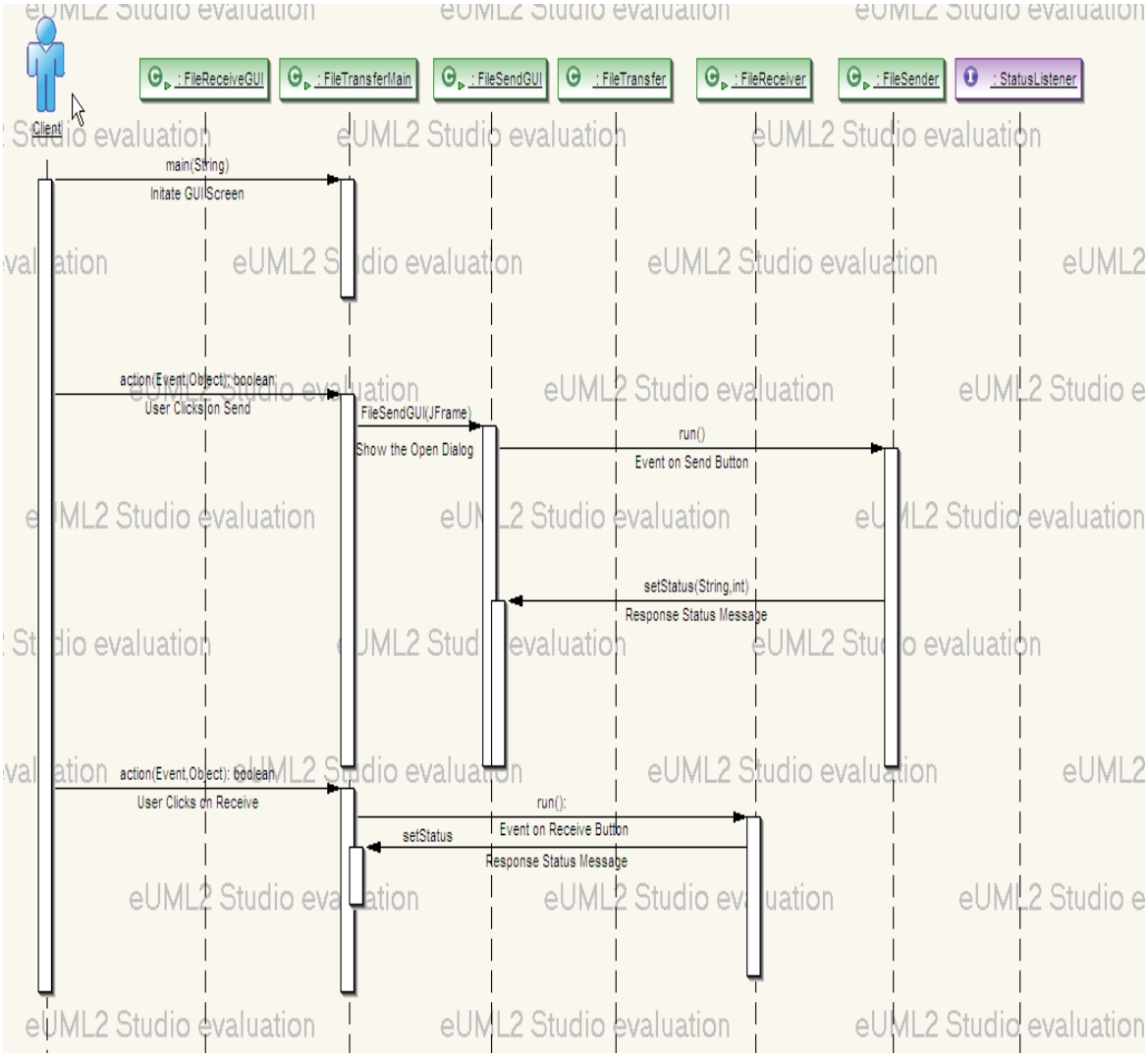


Figure 5.4:Activity Diagram

\

5.6 Deployment Diagram

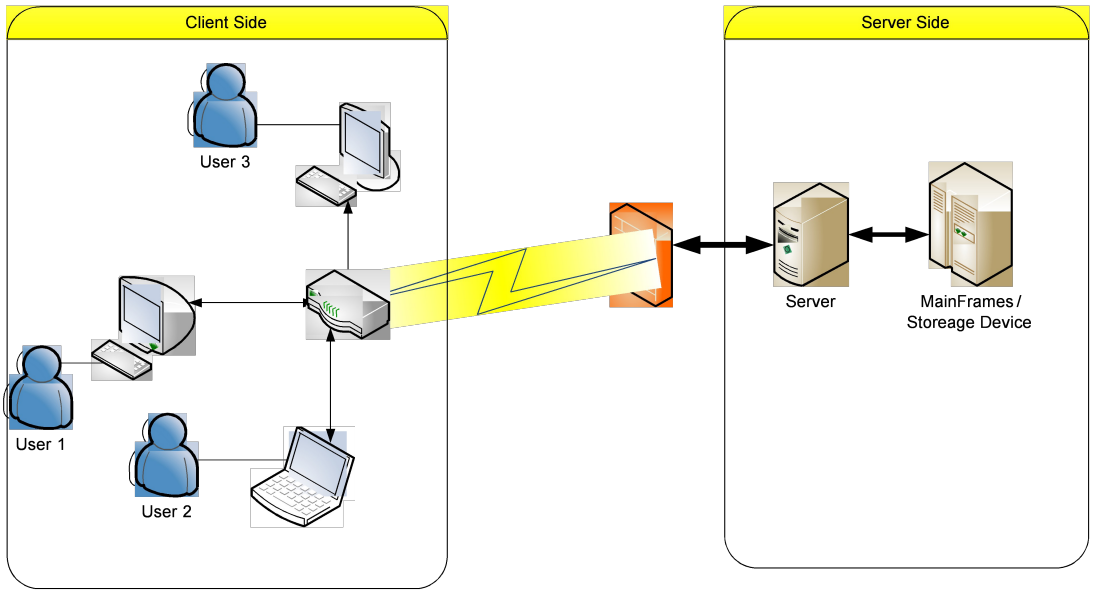


Figure 5.6: Deployment Diagram

5.7 INTERFACE DESCRIPTION

A detailed description of the input and output interfaces for the function is presented.

The software's interface(s) to the outside world are described.

5.7.1 External Machine Interfaces

Interfaces to other machines (computers or devices) are described. The application is

Interfaced with a Internet Router.

5.7.2 User Interface design

Interfaces the user will use to interact with the system are described. The user can use the GUI portal

5.7.3 Restrictions, limitations, and constraints

1. Internet bandwidth is required to transfer the images.

2. The project is not designed for highly secure data.

6.SSL

The Secure Sockets Layer (SSL) is a commonly used protocol for managing the security of a message transmission on the Internet. SSL has recently been succeeded by Transport Layer Security (TLS), which is based on SSL. SSL uses a program layer located between the Internet's Hypertext Transfer Protocol (HTTP) and Transport Control Protocol (TCP) layers. SSL is included as part of both the Microsoft and Netscape browsers and most Web server products. Developed by Netscape, SSL also gained the support of Microsoft and other Internet client/server developers as well and became the de facto standard until evolving into Transport Layer Security. The "sockets" part of the term refers to the sockets method of passing data back and forth between a client and a server program in a network or between program layers in the same computer. SSL uses the public-and-private key encryption system from RSA, which also includes the use of a digital certificate.

TLS and SSL are an integral part of most Web browsers (clients) and Web servers. If a Web site is on a server that supports SSL, SSL can be enabled and specific Web pages can be identified as requiring SSL access. Any Web server can be enabled by using Netscape's SSLRef program library which can be downloaded for noncommercial use or licensed for commercial use.

TLS and SSL are not interoperable. However, a message sent with TLS can be handled by a client that handles SSL but not TLS.

For receiving and sending data through an encrypted channel we can use two strategies, the first is that we encrypt some data at the server end and send it over

traditional sockets where even if the data if leaked in between somewhere the data is not stolen, but the only problem is that the key has also to be given with the data interleaved in the file to be transmitted. This can be dangerous, therefore the better method is to use SSL sockets. With SSL sockets we can create a secured channel and exchange keys before hand and then send data. To decrypt this data by brute force will take a lot of time making it virtually impossible.

Java provides services for both data encryption and SSL sockets.

```
KeyGeneratorkgen = KeyGenerator.getInstance("AES");

kgen.init(128);

SecretKeyskey = kgen.generateKey();

byte[] raw = skey.getEncoded();

SecretKeySpecskeySpec = newSecretKeySpec(raw, "AES");

Cipher cipher = Cipher.getInstance("AES");

cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
```

Here we have used some of the default services provided by the java(java.security).

When an SSL client socket connects to an SSL server, it receives a certificate of authentication from the server. The client socket then validates the certificate against a set of certificates in its \meta{trust store}.

The default truststore is <java-home>/lib/security/cacerts. If the server's certificate cannot be validated with the certificates in the truststore, the server's

certificate must be added to the truststore before the connection can be established.

Class SSL Socket

java.lang.Object

java.net.Socket

javax.net.ssl.SSLSocket

public abstract class SSLSocket

extends Socket

This class extends Sockets and provides secure socket using protocols such as the "Secure Sockets Layer" (SSL) or IETF "Transport Layer Security" (TLS) protocols.

Such sockets are normal stream sockets, but they add a layer of security protections over the underlying network transport protocol, such as TCP. Those protections include:

Integrity Protection. SSL protects against modification of messages by an active wiretapper.

Authentication. In most modes, SSL provides peer authentication. Servers are usually authenticated, and clients may be authenticated as requested by servers.

Confidentiality (Privacy Protection). In most modes, SSL encrypts data being sent between client and server. This protects the confidentiality of data, so that

passive wiretappers won't see sensitive data such as financial information or personal information of many kinds.

These kinds of protection are specified by a "cipher suite", which is a combination of cryptographic algorithms used by a given SSL connection. During the negotiation process, the two endpoints must agree on a ciphersuite that is available in both environments. If there is no such suite in common, no SSL connection can be established, and no data can be exchanged.

The cipher suite used is established by a negotiation process called "handshaking". The goal of this process is to create or rejoin a "session", which may protect many connections over time. After handshaking has completed, you can access session attributes by using the getSession method. The initial handshake on this connection can be initiated in one of three ways:

- calling startHandshake which explicitly begins handshakes, or
- any attempt to read or write application data on this socket causes an implicit handshake, or
- a call to getSession tries to set up a session if there is no currently valid session, and an implicit handshake is done.

If handshaking fails for any reason, the SSLSocket is closed, and no futher communications can be done.

There are two groups of cipher suites which you will need to know about when managing cipher suites:

Supported cipher suites: all the suites which are supported by the SSL implementation. This list is reported using getSupportedCipherSuites.

Enabled cipher suites, which may be fewer than the full set of supported suites. This group is set using the setEnabledCipherSuites method, and queried using the getEnabledCipherSuites method. Initially, a default set of cipher suites will be enabled on a new socket that represents the minimum suggested configuration.

Implementation defaults require that only cipher suites which authenticate servers and provide confidentiality be enabled by default. Only if both sides explicitly agree to unauthenticated and/or non-private (unencrypted) communications will such a ciphersuite be selected.

When SSLSockets are first created, no handshaking is done so that applications may first set their communication preferences: what cipher suites to use, whether the socket should be in client or server mode, etc. However, security is always provided by the time that application data is sent over the connection.

You may register to receive event notification of handshake completion. This involves the use of two additional classes. HandshakeCompletedEvent objects are passed to HandshakeCompletedListener instances, which are registered by users of this API. SSLSockets are created by SSLSocketFactorys, or by accepting a connection from a SSLServerSocket.

A SSL socket may choose to operate in the client or server mode. This will determine who begins the handshaking process, as well as which messages should be sent by each party. However, each connection must have one client and one server, or handshaking will not progress properly.

CRYPTOGRAPHY

In cryptography, a Caesar cipher, also known as a Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a shift of 3, A would be replaced by D, B would become E, and so on. The method is named after Julius Caesar, who used it to communicate with his generals.

The encryption step performed by a Caesar cipher is often incorporated as part of more complex schemes, such as the Vigenère cipher, and still has modern application in the ROT13 system. As with all single alphabet substitution ciphers, the Caesar cipher is easily broken and in practice offers essentially no communication security.

Encryption and Decryption Concept

The original information is known as plaintext, and the encrypted form as cipher text. The cipher text message contains all the information of the plaintext message, but is not in a format readable by a human or computer without the proper mechanism to decrypt it; it should resemble random gibberish to those not intended to read it. The encrypting procedure is varied depending on the key which changes the detailed operation of the algorithm. Without the key, the cipher cannot be used to encrypt or decrypt.

7.AES

AES is a block cipher. This means that it operates on fixed-length chunks of data (for example, blocks), applying the same transformation to each block. The transformation is controlled by use of the encryption key. Block ciphers (and thus AES) use symmetric keys, which means that the same key used to encrypt data is also used to decrypt it (or in some cases, a key only trivially different). In operation, a user inputs 128 bytes of plaintext, along with a key, and receives as output 128 bytes of ciphertext. To decrypt the ciphertext, the user inputs it and the key to the algorithm to retrieve the original 128 bytes of plaintext. Encryption proceeds via a number of rounds. For 128-bit keys, AES prescribes 1ten rounds; for 192-bit keys, it uses 12 rounds; and for 256-bit keys, it uses 14 rounds.

AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits. In contrast, the parent Rijndael algorithm can have both key and block sizes of 128, 160, 192, 224, or 256 bits. The 128 bits in a block are arranged in a grid of 4 x 4 bytes (also known as the state). Each round of encryption consists of four steps to generate a new state:

1. AddRoundKey

2. SubBytes

3. ShiftRows

4. MixColumn

In the final round of encryption, the MixColumn step is replaced with another AddRoundKey step.

In Step 1, AddRoundKey, a subkey is combined with the state. The subkey is derived from the main key using a key schedule, which generates an endless supply of subkeys using a well defined set of rotations, exponents, and multiplications. The subkey is the same size as the state, and the two are combined using the logical exclusive OR operation (XOR). This state obscures the original state and provides a new encrypted state.

In Step 2, SubBytes, each byte in the state is substituted using an S-Box. The S-Box (or substitution box) is another transformation, this time achieved by finding the multiplicative inverse of the byte in Rijndael's finite field, then transforming that result using binary linear algebra (an affine transform). Choosing good S-Box transforms is critical to the security of an encryption algorithm. Again, the result of this step is to obscure the original state and provide a new, encrypted state.

In Step 3, ShiftRows, the bytes in the rows of the 4 x 4 state are shifted within the row.The first row is left unchanged, the second row is shifted left one byte, and the third and fourth rows are shifted left two and three bytes, respectively.

Finally, in Step 4, MixColumns, the four bytes of each column are combined using an invertible linear transform. Four input bytes generate four output bytes, with each input byte influencing each output byte. You can view this as a matrix multiply within a finite field. An equivalent view is that it is a modulo multiply of a pair of polynomials. This operation provides diffusion, meaning that it spreads the input of a single character of plaintext across several characters. Repetition of the ShiftRows and MixColumns steps ensures that changing a single letter of the plaintext changes every character in the output block of ciphertext.

Performance is an issue with any piece of software; however, it is especially important in cryptography. AES was designed with 32-bit processors in mind, and is extremely efficient. In particular, the transforms in Steps 2, 3, and 4 can all be achieved using lookup tables, so that a particular state or part of a state is used as an index into the pre calculated table, and the result is read from the table rather than calculated. This ability to "pre-calculate" most of the transformations makes AES computationally efficient. Four tables of 256 32-bit entries, for a total of 4096 bytes, can be used, so that the computation required for a round is a series of table lookups and XOR operations. In addition, this can be implemented in hardware.

The Security of AES

The entire point of cryptography is keeping secrets safe. An algorithm is useful only to the extent that breaking the encryption is difficult and expensive. In other words, if someone can easily break the encryption, what is the point of protecting the data in the first place? AES has been reviewed by many of the world's best code breakers, and no significant flaws have been reported. The National Security Agency of the U.S. government reviewed all the AES finalists, including Rijndael, and pronounced all of them adequate for federal non-classified data. In 2003, the U.S. Government announced that AES was appropriate for encrypting classified data. This is the first time in history that the general public has had access to a NSA-approved cipher for top secret information.

To date, the known attacks against AES have been side-channel attacks. A side-channel attack uses information about the implementation of the algorithm, rather

than a theoretical weakness of the algorithm. Side-channel attacks use things, such as audio information, power consumption, radiation leaks, and timing information, to deduce whole or partial solutions. Side-channel attacks require significant technical skill. An "open" algorithm, such as AES, is vulnerable to these attacks because the algorithm is available to attackers and to legitimate users.

The only known successful attacks against AES to date are side-channel attacks relying on the precise timing of an AES system. These attacks are against specific implementations of the algorithm. Attention to timing security in the design phase of an implementation can negate or greatly reduce the chances of a successful timing attack.

Some cryptographers still have concerns about AES. A common attack on block ciphers is to attack the algorithm with a reduced number of rounds. At the time of this writing, attacks on AES exist for seven rounds with 128-bit keys, eight rounds with 192-bit keys, and nine rounds with 256-bit keys. Recall that the full implementation of AES uses ten, 12, and 14 rounds with 128-, 192-, and 256-bit keys, respectively. There is concern that there is not enough distance between the attack for a seven-round encryption and the actual ten-round implementation and that there is a risk these attacks could be improved to break the cipher. Another worry results from the mathematical structure of AES. In contrast to most ciphers, AES has a concise and elegant algebraic structure. There is concern among some cryptographers that an attack based on new insights into this formulation could be successful.

AES appears to be secure as of this writing in late 2006. The largest well-known brute force attack occurred in 2002 against a 64-bit RC5 key. With a key size of at least 128 bits, AES is well out of reach of brute force attacks by normal adversaries for years if not decades.

Applications

Vendors of both hardware and software have enthusiastically adopted AES. Because AES uses a simple and efficient algorithm, using it as an encryption specification decreases system complexity, lowers costs, and promotes interoperability.

There are many areas where AES is now in commercial use. Most high-end VPN software contains implementations of AES, including offerings from Checkpoint, Cisco, and Symantec. AES is now commonly found in network appliances. Voice Over IP vendors are using AES for telephone security. Vendors now use AES to provide security for process control (SCADA) systems. AES has even been added to common file compression programs, such as WinZip. Dozens of hardware implementations are available that use both FPGAs and ASICs. There are multiple implementations in software in the public domain.

AES is one of the newest and most well known encryption standards. It was developed and analyzed in a thorough, lengthy, and widely respected process by NIST. AES is approved by the U.S. government for classified data, and numerous hardware and software vendors have implemented it.

AES uses 128-, 192-, or 256-bit keys. Encryption consists of ten, 12, or 14 rounds, where each round consists of four steps: AddRoundKey, SubBytes, ShiftRows, and MixColumns. The known attacks against AES to date have involved timing, where keys are guessed by analyzing how long particular steps require. Because AES has a well defined algebraic structure, some cryptographers worry that there might be attacks on the algorithm itself possible, but none have publicly emerged to date.

AES is efficient, elegant, and secure. It will be a top choice for data security in the next decade and beyond.

8. Summary and Conclusion

- Used Socket Communication for file transfer between Client and server
- Used JUI for transfer
- Used J2EE for implemtation of coding successful to transfer data between client and server using file transfer protocol.
- Addition of the security aspect by using AES encryption.
- Used SSL sockets and Java Cipher.
- Thus complete successful implementation of the secure data transfer using AES encryption.

9. References

Books and Papers:

- [1] Rod Johnson PhD Expert One-on-One J2EE Design and Development (Programmer to Programmer)
- [2] Roger S. Pressman,"Software engineering: a practitioner's approach", McGraw Hill, 5th edition,2001.
- [3] O'Reiliy Java Network Programming,
- Cryptography and Network Security- William Stallings
- Applied Cryptography and Data Security -Prof. Christof Paar, Ruhr-Universitat Bochum, Germany