CHAPTER

25

NETWORKING IN JAVA

When we have several computers and each one's resources should be available to the other computers, then we should connect all the computers. This is called a network. A network represents interconnection of computers either by using a cable or a satellite where no cable is needed. In a network, there may be several computers—some of them receiving the services and some providing the services to others. The computer which receives service is called a 'client' and the computer which provides the service is called a 'server'. Remember, a client sometimes acts as a server and a server acts as a client.

There are 3 requirements to establish a network:

- Hardware: includes the computers, cables, modems, hubs, etc.
- Software: includes programs to communicate between servers and clients.
- Protocol: represents a way to establish connection and helps in sending and receiving data in a standard format.

Let us now discuss Protocol in some detail.

TCP/IP Protocol

A protocol represents a set of rules to be followed by every computer on the network. Protocol is useful to physically move data from one place to another place on a network. TCP (Transmission Control Protocol) / IP (Internet Protocol) is the standard protocol model used on any network, including Internet.

TCP/IP model has got the following 5 layers:

- Application layer
- TCP
- IP
- Data link layer
- Physical layer

Application layer is the topmost layer of the TCP/IP model that directly interacts with an application (or data). This layer receives data from the application and formats the data. Then it sends that data to the next layer called TCP in the form of continuous stream of bytes. The TCP, upon receiving the data from the Application layer, will divide it into small segments called 'packets'. A packet contains a group of bytes of data. These packets are then sent to the next IP layer. IP layer inserts the packets into envelopes called 'frames'. Each frame contains a packet, the IP address of destination computer, the IP address of source computer, and some additional bits useful in error detection and

correction. These frames are then sent to Data link layer which dispatches them to correc destination computer on the network. The last layer, which is called the Physical layer, is used t physically transmit data on the network using the appropriate hardware. See Figure 25.1.

Of course, to send data from one place to another, first of all the computers should be correctl identified on the network. This is done with the help of IP addresses. An IP address is a uniqu identification number given to every computer on the network. It contains four integer numbers i the range of 0 to 255 and separated by a dot as:

```
87.248.113.14
```

This IP address may represent, for example a website on a server machine on Internet as:

www.yahoo.com

Therefore, to open 'yahoo.com' site, we can type the site address as 'www.yahoo.com' or its I address as '87.248.113.14'. But when we type the IP address in numeric form, that number i mapped to the website automatically. This mapping service is available on Internet, which is calle 'DNS' (Domain Naming service).

Important Interview Question

What is IP address?

An IP address is a unique identification number allotted to every computer on a network or Internet. IP address contains some bytes which identify the network and the actual computer inside the network.

What is DNS?

Domain Naming Service is a service on Internet that maps the IP addresses with corresponding website names.

On Internet, IP addresses of 4 bytes are used and this version is called IP address version 4. Th next new version of IP address is version 6, which uses 16 bytes to identify a computer.

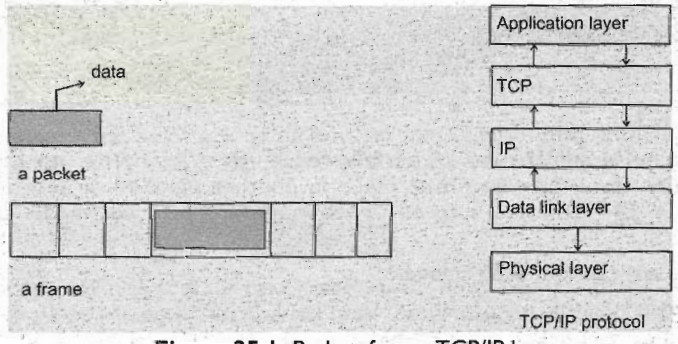


Figure 25.1 Packet, frame, TCP/IP layers

TCP/IP takes care of number of bits sent and whether all the bits are received duly by th destination computer. So it is called 'connection oriented reliable protocol'. Every transmitted bit accountable in this protocol. Hence, this protocol is highly suitable for transporting data reliably a network. Almost all the protocols on Internet use TCP/IP model internally.

HTTP (hyper text transfer protocol) is the most widely used protocol on Internet, which is used transfer web pages (.html files) from one computer to another computer on Internet. FTP (fi transfer protocol) is useful to download or upload files from and to the server. SMTP (simple ma transfer protocol) is useful to send mails on network. POP (post office protocol) is useful to rece mails into the mail boxes.

User Datagram Protocol (UDP)

UDP is another protocol that transfers data in a connection less and unreliable manner. It will not check how many bits are sent or how many bits are actually received at the other side. During transmission of data, there may be loss of some bits. Hence, UDP is used to send images, audio files, and video files. Even if some bits are lost, still the image or audio file can be composed with a slight variation that will not disturb the original image or audio.

Sockets

It is possible to establish a logical connecting point between a server and a client so that communication can be done through that point. This point is called 'socket'.

Important Interview Question

What is a socket?

A socket is a point of connection between a server and a client on a network.

Each socket is given an identification number, which is called 'port number'. Port number takes 2 bytes and can be from 0 to 65,535. Establishing communication between a server and a client using sockets is called 'socket programming'.

Important Interview Question

What is port number?

Port number is a 2 byte number which is used to identify a socket uniquely.

We should use a new port number for each new socket. Similarly, we should allot a new port number depending on the service provided on a socket. Every new service on the net should be assigned a new port number. Please have a look at some already allotted port numbers for the services shown in Table 25.1. Avoid using same port numbers, which are already used by applications running in your system.

Table 25.1

Port number	Application or service
13	Data and time services
21	FTP which transfers files
23	Telnet, which provides remote login
25	SMTP, which delivers mails
67	BOOTP, which provides configuration at boot time
80	HTTP, which transfers web pages
109	POP, which accesses mail boxes

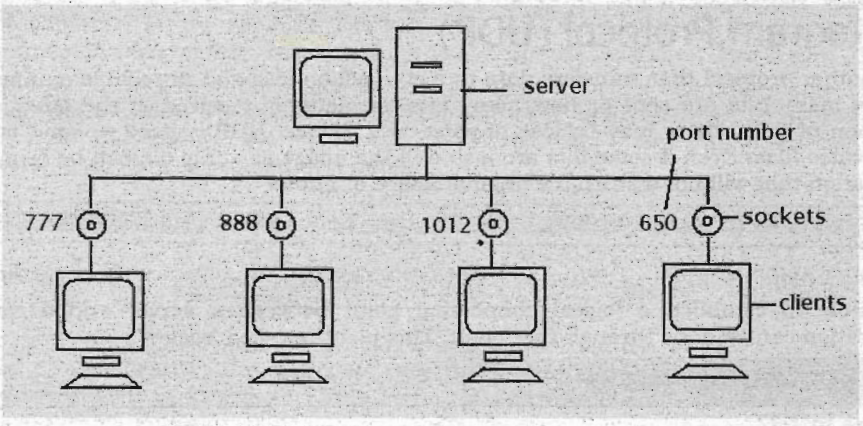


Figure 25.2 A server connected with clients using sockets

A socket, at server side is called 'server socket ' and is created using `ServerSocket` class in Java. socket, at client side is called 'Socket' and is created using `Socket` class. Both the `ServerSoc` and `Socket` classes are available in `java.net` package. Of course, a server socket may not necessarily at server side; it may be created at client side also, if the client acts as server. Simila a client socket may also exist at server side, if the server acts as client.

Knowing IP Address

It is possible to know the IP Address of a website on Internet with the help of `getByName()` meth of `InetAddress` class of `java.net` package. The `getByName()` method takes host name (ser name) and returns `InetAddress`, which is nothing but the IPAddress of that server. See t following program.

Program 1: Write a program to accept a website name and return its IPAddress, after checking it Internet.

Note

This program should be executed on a system which is connected to Internet.

```
//Knowing IPAddress of a website
import java.io.*;
import java.net.*;
class Address
{
      public static void main(String args[ ]) throws IOException
      {
            //accept name of website from keyboard
            BufferedReader br = new BufferedReader(new
             InputStreamReader(System.in));

            System.out.print("Enter a website name: ");
            String site = br.readLine();

            try{
                  //getByName() method accepts site name and returns its IP
                   Address
                  InetAddress ip = InetAddress.getByName(site);
                  System.out.println("The IP Address is: "+ ip);
            }catch(UnknownHostException ue)
            {
```

```
                    System.out.println("Website not found");
                }
            }
        }
```

Output:

```
C:\> javac Address.java
C:\> java Address
Enter a website name: www.yahoo.com
The IP Address is: www.yahoo.com/87.248.113.14
```

URL

URL (Uniform Resource Locator) represents the address that is specified to access some information or resource on Internet. Look at the example URL:

`http://www.dreamtechpress.com:80/index.html`

The URL contains 4 parts:

- The protocol to use (`http://`).
- The server name or IP address of the server (`www.dreamtechpress.com`).
- The third part represents port number, which is optional (`:80`).
- The last part is the file that is referred. This would be generally `index.html` or `home.html` file (`/index.html`).

URL is represented by a class 'URL' in `java.net` package. To create an object to URL, we can use the following formats:

```
URL obj = new URL(String protocol, String host, int port, String path);
```

Or,

```
URL obj = new URL(String protocol, String host, String path);
```

The following program accesses the different parts of the URL supplied to URL object and displays them.

Program 2: Write a program to retrieve different parts of a URL supplied to URL class object.

```
//URL
import java.net.*;
class MyURL
{
    public static void main(String args[ ]) throws Exception
    {
        URL obj = new URL("http://dreamtechpress.com/index.html");

        System.out.println("Protocol: "+ obj.getProtocol());
        System.out.println("Host: "+ obj.getHost());
        System.out.println("File: "+ obj.getFile());
        System.out.println("Port: "+ obj.getPort());
        System.out.println("Path: "+ obj.getPath());
        System.out.println("External form: "+ obj.toExternalForm());
    }
}
```

```
<input type="hidden" name="ei" value="UTF-8" />
<input type="hidden" name="fr" value="yfp-t-501" />
<input type="hidden" name="cop" value="mss" />
:
:
```

Creating a Server That Sends Data

We can create a socket that can be used to connect a server and a client. Once the socket created, the server can send data to the client and the client can receive it. All we have to do is just send the data from the server to the socket. The socket will take care of whom to send data the network. Let us follow these steps to create a server that sends some strings (messages) to client:

- At server side, create a server socket with some port number. This is done using `ServerSoc` class as:

```
ServerSocket ss = new ServerSocket(777);
```

- Now, we should make the server wait till a client accepts connection. This is done us `accept()` method.

```
Socket s = ss.accept();
```

- Attach output stream to the server socket using `getOutputStream()` method. This meth returns `OutputStream` object. This stream is used by the socket to send data to client.

```
OutputStream obj = s.getOutputStream();
```

- Take another stream like `PrintStream` to send data till the socket.

```
PrintStream ps = new PrintStream(obj);
```

- Finally, this `PrintStream` is used by the server to send data to the client. To send data, have `print()` or `println()` methods available in `PrintStream`.

```
ps.println(str);
```

- Then close the connection. This can be done by closing all the streams and sockets at se side as:

```
ss.close();  //close ServerSocket
s.close();   //close Socket
ps.close();  //close PrintStream
```

All these steps are shown in Figure 25.3 and also implemented in Program 4.

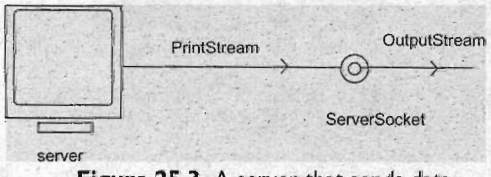


Figure 25.3 A server that sends data

Program 4: Write a program to create a server for the purpose of sending some strings to the client.

```
//Server1 - to send strings
import java.io.*;
import java.net.*;

class Server1
{
      public static void main(String args[ ])
      throws Exception
      {
            //Create a server socket with some port number
            ServerSocket ss = new ServerSocket(777);

            //let the server wait till a client accepts connection
            Socket s = ss.accept();
            System.out.println("Connection established");

            //attach output stream to the server socket
            OutputStream obj = s.getOutputStream();

            //attach print stream to send data to the socket
            PrintStream ps = new PrintStream(obj);

            //send 2 strings to the client
            String str = "Hello client";
            ps.println(str);
            ps.println("Bye");

            //close connection by closing the streams and sockets
            ps.close();
            ss.close();
            s.close();
      }
}
```

Ouput:

```
D:\rnr> javac Server1.java
D:\rnr>
DO NOT RUN THIS PROGRAM TILL CLIENT IS ALSO CREATED…
```

Creating a Client That Receives Data

We can write a client program that receives all the strings sent from the server. Let us follow these steps to do this:

- First, we should create a socket at client side using `Socket` class as:

```
Socket s = new Socket("IPAddress", port number);
```

Here, the `IPAddress` represents the IPAddress of the server machine where `Server1.java` program is running. To know the IPAddress, we can use DOS command, as:

```
C:\> ipconfig
```

This will display the IPAddress of the machine where the command is applied.

Or, we can follow the commands:

Start -> settings -> control panel

-> Network connections ->right click on this to see the 'local area connection' dialog box and th
double click on Internet protocol (TCP/IP).

It opens Internet protocol properties dialog box where we can see the IPAddress.
Figure 25.4.

It is possible to run the `Server1.java` and `Client1.java` programs on two different comput
connected in a network. But, at `Client1.java`, we should pass the server machine's IPAddre
Then the port number at `Client1.java` should be same as the port number with which the ser
socket has been created. In case, you do not have your computer in a network, you have to
both the server and client programs in the same system. In that case, you can use `localhost`
place of `IPAddress`. The word `localhost` represents that the server is also locally available in
same system.

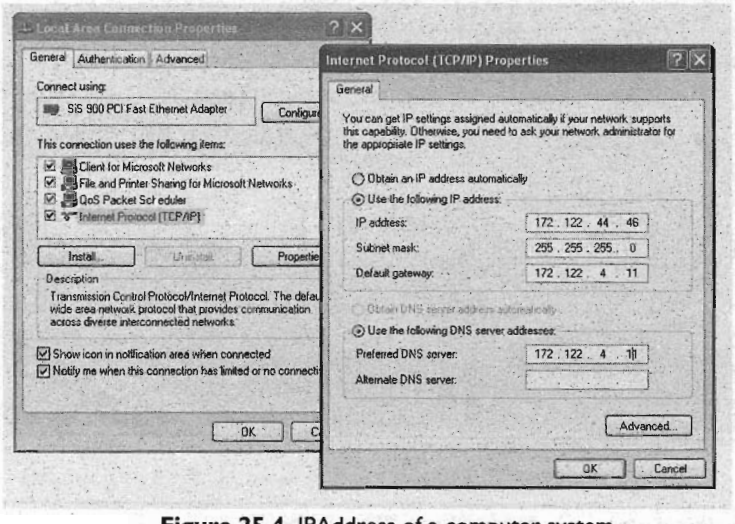


Figure 25.4 IPAddress of a computer system

- We should add `InputStream` to the socket so that the socket will be able to receive the data
the `InputStream`.

```
InputStream obj = s.getInputStream();
```

- To read the data from the socket into the client, we can take the help of `BufferedReader` as

```
BufferedReader br = new BufferedReader(new InputStreamReader(obj));
```

- Now we can read data from the `BufferedReader` object, using `read()` or `readLine()` meth
`Read()` method can read a single character at a time, where as `readLine()` can rea
string.

```
str = br.readLine();
```

- Close the connection by closing all the streams and sockets.

```
br.close();  //close the BufferedReader
s.close();  //close the Socket
```

All these steps are shown in Figure 25.4 and also implemented in Program 5.

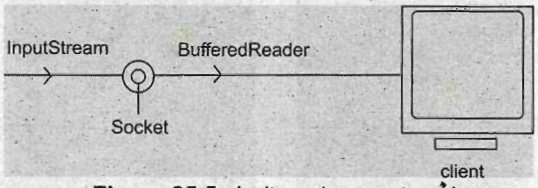


Figure 25.5 A client that receives data

Program 5: Write a program to create client side program, which accepts all the strings sent by the server.

```
//Client1 - to receive strings
import java.io.*;
import java.net.*;
class Client1
{
      public static void main(String args[ ])
      throws Exception
      {

            //create client socket with same port number
            Socket s = new Socket("localhost", 777);

            //to read data coming from server, attach InputStream to the socket
            InputStream obj = s.getInputStream();

            //to read data from the socket into the client, use BufferedReader
            BufferedReader br = new BufferedReader(new InputStreamReader(obj));

            //receive strings
            String str;
            while((str = br.readLine()) != null)
            System.out.println("From server: "+str);

            //close connection by closing the streams and sockets
            br.close();
            s.close();
      }
}
```

Output:

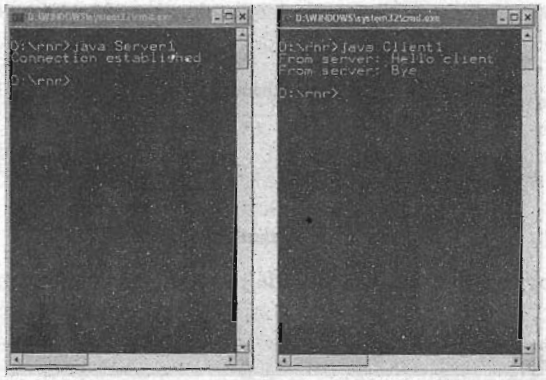
```
D:\rnr> javac Client1.java
D:\rnr>
```

To run the server and client in the same system:

After compiling server1.java and client1.java, run these programs in two separate dos windows, as shown in the output.

To run in different systems:

Run the server1.java in a computer and client1.java in another. They should have been connected in a network.



Run the `Server1.java` in a DOS window, the server would be in waiting state, expectin connection from a client. Then run `Client1.java` in another DOS window. Immediately connection is established, and at server side it displays "Connection established". Then it sends strings "Hello client" and "Bye" to the client, which are received and displayed at client term When the server disconnects, it sends a null string to the client. When client receives null, it disconnects.

Two-way Communication Between Server and Client

It is possible to send data from the server and receive the response from the client. Similarly, client can also send and receive the data to-and-fro. For this purpose, we need additional stre both at server and client. For example, to receive data into the server, it is a better idea to `BufferedReader` as:

```
InputStream obj = s.getInputStream();
BufferedReader br = new BufferedReader(new InputStreamReader(obj);
```

Then `read()` or `readLine()` methods of `BufferedReader` class can be used to read data.

To send data from the client, we can take the help of `DataOutputStream` as:

```
OutputStream obj = s.getOutputStream();
DataOutputStream dos = new DataOutputStream(obj);
```

Then `writeBytes()` method of `DataOutputStream` can be used to send strings in the form of g of bytes.

Program 6: Write a program to create a server such that the server receives data from the c using `BufferedReader` and then sends reply to the client using `PrintStream`.

```
//Server2 - A server that receives data and sends data
import java.io.*;
import java.net.*;

class Server2
{
     public static void main(String args[ ])
     throws Exception
     {
          //Create server socket
          ServerSocket ss = new ServerSocket(888);
```

```
                //connect it to client socket
                Socket s = ss.accept();
                System.out.println("Connection established");

                //to send data to the client
                PrintStream ps = new PrintStream(s.getOutputStream());

                //to read data coming from the client
                BufferedReader br = new BufferedReader(new
                 InputStreamReader(s.getInputStream()));

                //to read data from the key board
                BufferedReader kb = new BufferedReader(new
                 InputStreamReader(System.in));

                while(true)  //server executes continuously
                {
                        String str,str1;

                        //repeat as long as client does not send null string
                        while((str = br.readLine()) != null)  //read from client
                        {
                                System.out.println(str);
                                str1 = kb.readLine();
                                ps.println(str1); //send to client
                        }

                        //close connection
                        ps.close();
                        br.close();
                        kb.close();
                        ss.close();
                        s.close();
                        System.exit(0); //terminate application
                } //end of while
        }
}
```

Output:

```
D:\rnr> javac Sever2.java
D:\rnr>
```

Program 7: Write a program to create a client which first connects to a server, then starts the communication by sending a string to the server. The server sends response to the client. When 'exit' is typed at client side, the program terminates.

```
//Client2 - a client that sends data and receives also
import java.io.*;
import java.net.*;

class Client2
{
      public static void main(String args[ ])
      throws Exception
      {

              //Create client socket
              Socket s = new Socket("localhost", 888);

              //to send data to the server
              DataOutputStream dos = new DataOutputStream(s.getOutputStream());
              //to read data coming from the server
              BufferedReader  br = new BufferedReader(new
               InputStreamReader(s.getInputStream()));
```

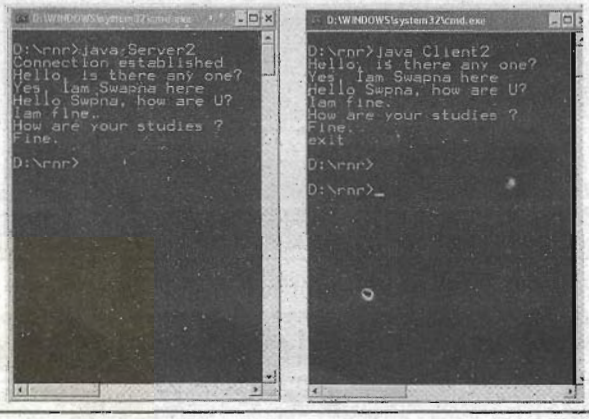
```
            //to read data from the key board
            BufferedReader kb = new BufferedReader(new
             InputStreamReader(System.in));

            String str,str1;
            //repeat as long as exit is not typed at client
            while(!(str = kb.readLine()).equals("exit"))
            {
                    dos.writeBytes(str+"\n");  //send to server
                    str1 = br.readLine(); //receive from server
                    System.out.println(str1);
            }
            //close connection.
            dos.close();
            br.close();
            kb.close();
            s.close();
        }
}
```

Output:

```
D:\rnr> javac Client2.java
D:\rnr>
```

Run the `server2.java` and `client2.java` in two dos windows.

See the output while running these programs.



Retrieving a file at server

Let us write client and server programs, such that the client sends the name of a file to the server. After receiving the filename, the server searches for the file to know if it exists or not. If the file exists, the server sends the file contents to the client. This is shown in Programs 8 and 9, which are self-explanatory.

Program 8: Write a program that accepts the filename and checks for its existence. When the file exists at server side, it sends its contents to the client.

```
//A server that sends a file content to the client
import java.io.*;
import java.net.*;
```

```
class FileServer
{
      public static void main(String args[ ]) throws Exception
      {
            //create server socket
            ServerSocket ss = new ServerSocket(8888);

            //make the server wait till a client accepts connection
            Socket s = ss.accept();
            System.out.println("Connection established");

            //to accept file name from client
            BufferedReader in = new BufferedReader(new
             InputStreamReader(s.getInputStream()));

            //to send file contents to client
            DataOutputStream out = new DataOutputStream(s.getOutputStream());

            //read the filename from the client
            String fname = in.readLine();

            FileReader fr = null;
            BufferedReader file = null;
            boolean flag;

            //create File class object with filename
            File f = new File(fname);

            //test if file exists or not
            if(f.exists()) flag = true;
            else flag = false;

            //if file exists, send "Yes" to client, else send "No"
            if(flag == true) out.writeBytes("Yes"+"\n");
            else out.writeBytes("No"+"\n");

            if(flag == true)
            {
                  //attach file to the FileReader to read data
                  fr = new FileReader(fname);

                  //attach FileReader to BufferedReader
                  file = new BufferedReader(fr);

                  String str;

                  //read from BufferedReader and write to DataOutputStream
                  while((str = file.readLine()) != null)
                  {
                        out.writeBytes(str+"\n");

                  }

                  file.close();
                  out.close();
                  in.close();
                  fr.close();
                  s.close();
                  ss.close();
            }

      }
}
```

Output:

```
D:\rnr> javac FileServer.java
```