ISSN:0975-9646

Limi Kalita / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 4802-4807

Socket Programming

Limi Kalita

M.Tech Student, Department of Computer Science and Engineering,
Assam Down Town University,
Guwahati, India.

Abstract: The aim of the paper is to introduce sockets, its deployment pertaining to network programming. Sockets play a vital role in client server applications. The client and server can communicate with each other by writing to or reading from these sockets. They were invented in Berkeley as part of the BSD flavor of UNIX operating systems. And they spread like wildfire with the Internet. This paper introduces elements of network programming and concepts involved in creating network applications using sockets. One of the most basic network programming tasks likely to be faced as a java programmer is performing the socket functions/methods because java has been preferred mostly for establishing client server communications using sockets.

Keywords: Socket, Port, Transport Layer, TCP, UDP

1. INTRODUCTION:

In the 1980s, the US government's Advanced Research Projects Agency (ARPA) provided funds to the University of California at Berkeley to implement TCP/IP protocols under the UNIX operating system. During this project, a group of Berkeley researchers developed an application program interface (API) for TCP/IP network communications called the socket interface. The socket interface is an API TCP/IP networks i.e. it defines a variety of software functions or routines for the development of applications for TCP/IP networks. The socket interface designers originally built their interface into the UNIX operating system. However, the other operating systems, environments, such as Microsoft Windows, implement the socket interface as software libraries. However, regardless of the environment in which we program, the program code will look much the same. Socket programming can be done in any language, which supports network communication, but java is preferred because it is platform independent, it has exception mechanisms for robust handling of common problems that occur during I/O and networking operations and its threading facilities provide a way to easily implement powerful servers One of java's major strong suits as a programming language for client-server application development is its wide range of network support. Java has this advantage because it was developed with the Internet in mind. Another advantage of java is that it provides security. The result is that we have a lot of options in regard to network programming in Java. Java performs all of its network communication through sockets.

1.1 Client-Server Communication

A network is composed of computers which is either a client or a server. A server is a program that is offering some service whereas a client is a program that is requesting some service. Servers are powerful computers or processes dedicated to managing disk drives (file servers), printers (print servers), or network traffic (network services) whereas clients are PCs or workstations on which users run applications. Clients rely on servers for resources, such as files, devices and even processing power. When these programs are executed, as a result, a client and a server process are created simultaneously and these two processes communicate with each other by reading from and writing to sockets as shown in figure: 1.1.

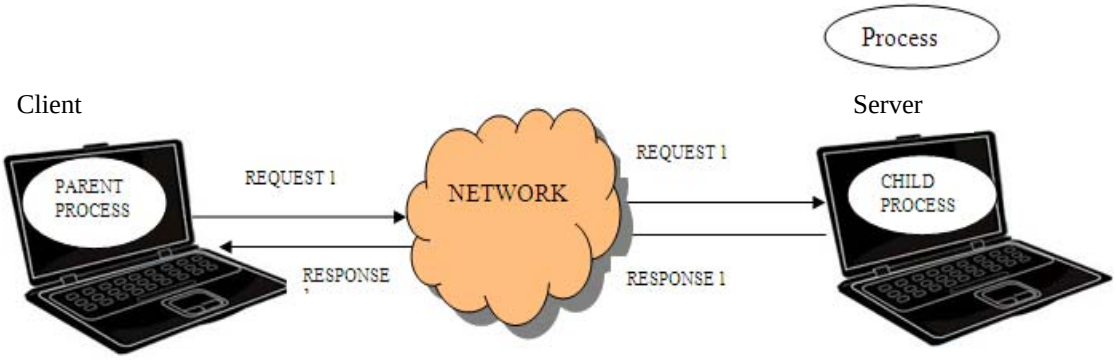

Figure: 1.1 Client-Server communications

These sockets are the programming interfaces provided by the TCP and UDP protocols for stream and datagram communication respectively of the transport layer which is a part of the TCP/IP stack. When creating a network application, the developer's main task is to write the code for both the client and server programs. The client/server application that is covered here is a proprietary client/server application. A single developer (or development team) creates both the client and server programs, and the developer has complete control over what goes in the code. But because the code does not implement a public-domain protocol, other independent developers will not be able to develop code that interoperates with the application. When developing a proprietary application, the developer must be careful not to use one of the well-known port numbers defined in the RFCs.

1.2 Sockets:

This term 'socket' has come from an electricity/phone socket metaphor where sockets acts as interfaces that plug into each other over a network.

Technically, sockets had been defined in many ways.

1. According to Wikipedia, a network socket is an endpoint of an inter-process communication flow across a computer network . A socket is composed of an IP address and a port number.
2. Sockets can be defined as the end-points of a connection between two computers identified by an IP Address and a port number.
3. Sockets can also be defined as a software abstraction used to represent the "terminals" of a connection between two machines.
4. It can also be defined as an abstraction that is provided to an application programmer to send or receive data to another process.
5. The socket is the door between the application process and TCP.
6. Socket is an interface between the application and the network.

One popular e.g. of Sockets are the Dixie cups game played where we took two paper cups and tied them together with the help of a string to form a telephone as shown in figure: 1.3.

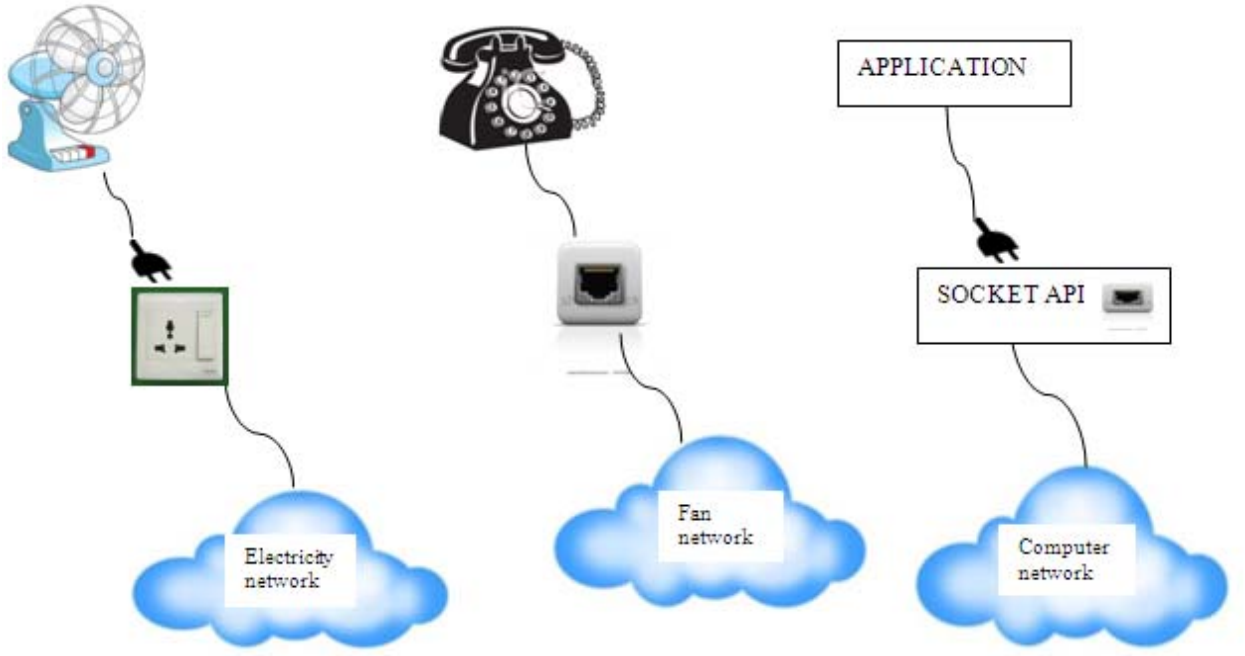

Figure: 1.2 Examples of a socket

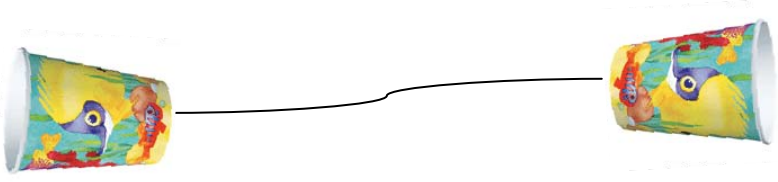

Figure: 1.3 Dixie cups

Your friend would take one of the cups and walk to the other side of the room and talk into the cup. You would put your ear up to the other cup and be able to hear your friend. The Dixie cups in this example represent Sockets. You communicate with your friend by talking into the cup (getting an output stream from the Socket and sending bytes) and by putting your ear up to the cup to hear your friend talk (getting an input stream from the Socket and reading data from it). A stream is a flowing sequence of characters that flow into or out of a process.

1.3 Operations of Socket:

A socket performs four fundamental operations:

1. To connect to a remote machine,
2. Send data,
3. Receive data and
4. Close the connection.

A socket may not be connected to more than one host at a time. However, a socket may both send data to and receive data from the host to which it's connected. The java.net.Socket class is Java's interface to a network socket and allows you to perform all four fundamental socket operations.

1.4 Port:

In computer networking, a port is an application-specific or process-specific software construct serving as a communications endpoint in a computer's host operating system. A port is associated with an IP address of the host, as well as the type of protocol used for communication. The purpose of ports is to uniquely identify different applications or processes running on a single computer. The protocols that primarily use ports are the Transport Layer protocols, such as the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) of the Internet software stack, often called TCP/IP (Transport Control Protocol/Internet Protocol)stack, as shown in figure :1.4. They use ports to map incoming data to a particular process running on a computer. So a port will identify a socket on a host.

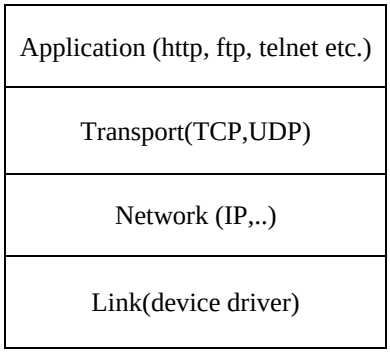

Figure: 1.4 TCP/IP software stack

Again, an IP Address is not enough to identify a unique server, because many server programs may exist on one machine. So we need a unique identification for each server. This unique identification is referred to as port.

When we are setting up client or a server we must choose a port to where both client and server agree to meet. This port is not a physical port, but a logical port specified by a 16-bit integer number. Some port nos. from 0 to 1024 has been reserved to support common/well known services and the developer must be careful not to use one of the well-known port numbers defined in the RFCs while developing a proprietary client-server application.

- ftp 21/tcp
- telnet 23/tcp
- smtp 25/tcp
- login 513/tcp
- http 80/tcp,udp
- https 443/tcp,udp

User-level process/services generally use port number value >= 1024

2. NETWORK PROGRAMMING WITH SOCKETS:

The Internet has been very popular in the past few years. With its popularity still growing, increased demand for Internet network software has grown as well. One of the greatest advantages to developing Internet software with Java is in its robust networking support built into the core language. The java.net package provides us with classes representing URLs, URL connections and sockets. Combined with the java.io package, we can quite easily write sophisticated platform-independent networking (Internet) applications. Network programming makes use of socket for Interprocess Communication. Due to which Network programming is also termed as socket programming. In Socket programming using Java, BSD style Socket to Interface with TCP/IP services is used. BSD Socket Interface provides facilities for Interprocess Communication. BSD Socket Interface supports different domain, the UNIX Domain, the Internet Domain and the NS Domain. Java Basically supports the Internet Domain to maintain cross platform. In Internet Domain, the BSD Socket Interface is built on the top of either TCP/IP or UDP/IP or the raw Socket. Socket Programming is important to understand how internet based interprocess communication work but not at the level program developed but at a higher level that is compiled to set of Socket Programs. Here sockets can also be termed as network socket or Internet socket since communication between computers is based on Internet protocol.

2.1 Socket Programming With TCP:

TCP provides a connection oriented service, since it is based on connections between clients and servers. Connection-oriented means that a connection is established before processes can exchange data. The Transmission Control Protocol is also reliable because when a TCP client sends data to the server, it requires an acknowledgement in return. If an acknowledgement is not received, TCP automatically retransmit the data and waits for a longer period of time.

The processes running on different machines communicate with each other by sending messages into sockets. Each process is analogous to a house and the process's socket is analogous to a door. As shown in Figure 2.1, the socket is

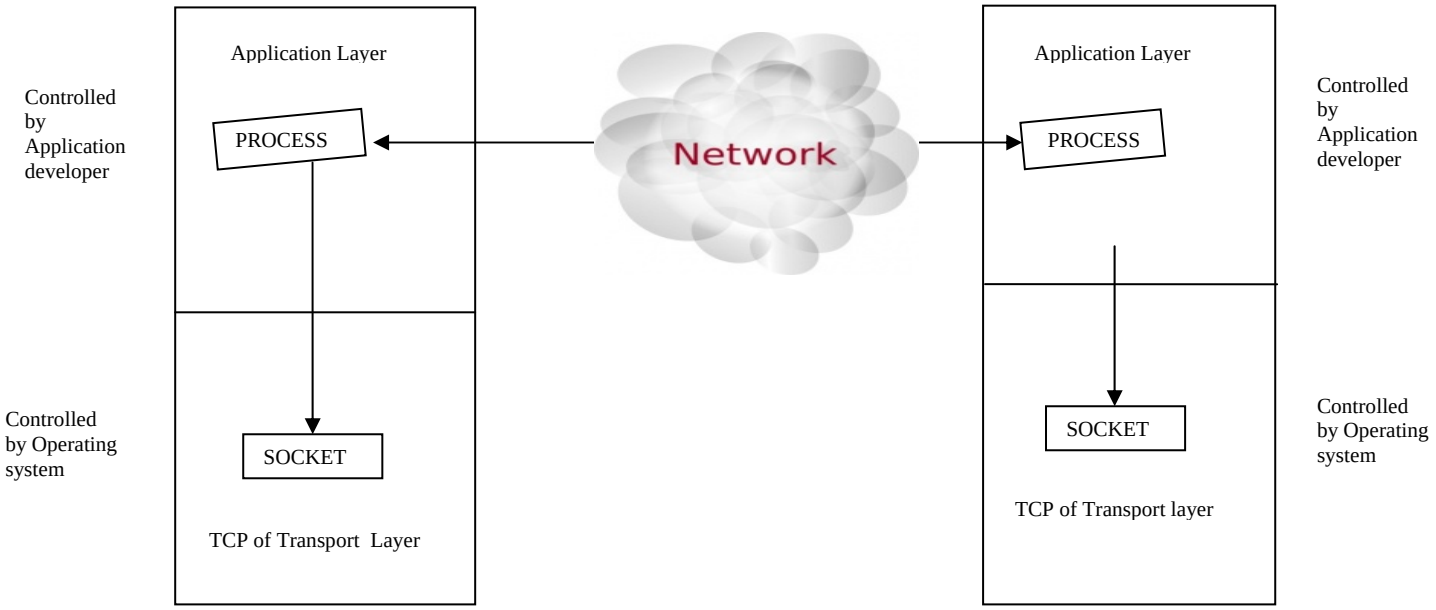

Figure: 2.1 Process communicating through TCP sockets

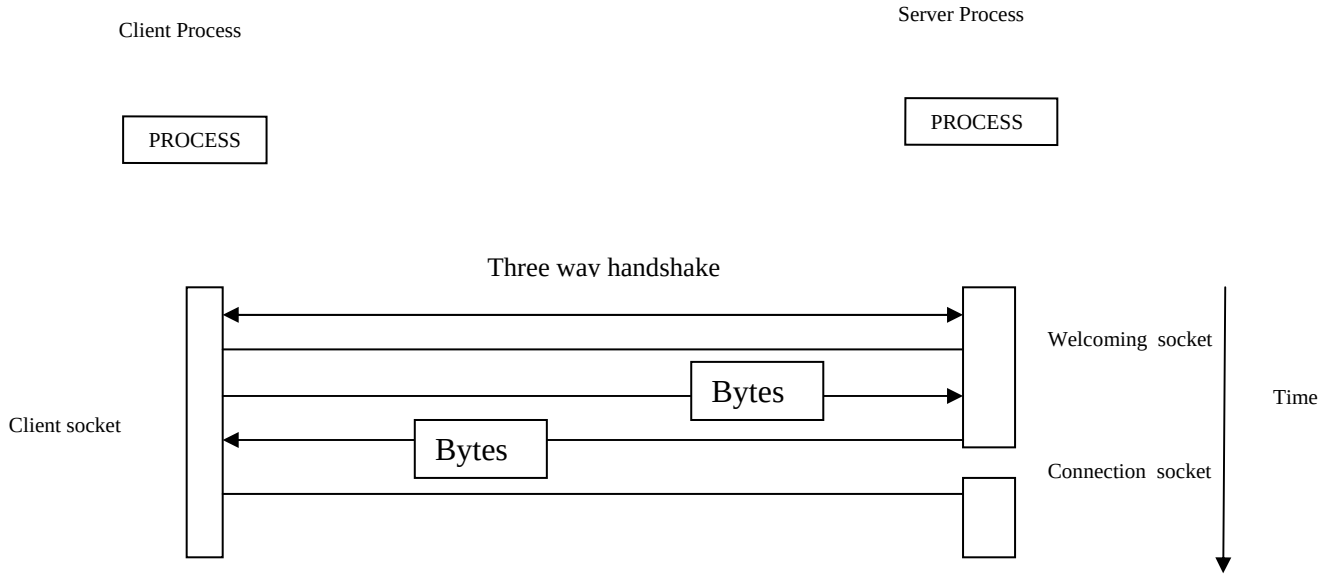

Figure 2.2: Client socket, welcoming socket, and connection socket

the door between the application process and TCP. The application developer has control of everything on the application-layer side of the socket; however, it has little control of the transport-layer side. (At the very most, the application developer has the ability to fix a few TCP parameters, such as maximum buffer size and maximum segment sizes.)

From the application's perspective, the TCP connection is a direct virtual pipe between the client's socket and the server's connection socket. The client process can send arbitrary bytes into its socket; TCP guarantees that the server process will receive (through the connection socket) each byte in the order sent. Furthermore, just as people can go in and out the same door, the client process can also receive bytes from its socket and the server process can also send bytes into its connection socket. This is illustrated in Figure 2.2.

Because sockets play a central role in client/server applications, client/server application development is also referred to as socket programming. Before providing our example client/server application, it is useful to discuss the notion of a stream. A stream is a sequence of characters that flow into or out of a process. Each stream is either an input stream for the process or an output stream for the process. If the stream is an input stream, then it is attached to some input source for the process, such as standard input (the keyboard) or a socket into which data flow from the Internet. If the stream is an output stream, then it is attached to some output source for the process, such as standard output (the monitor) or a socket out of which data flow into the Internet.

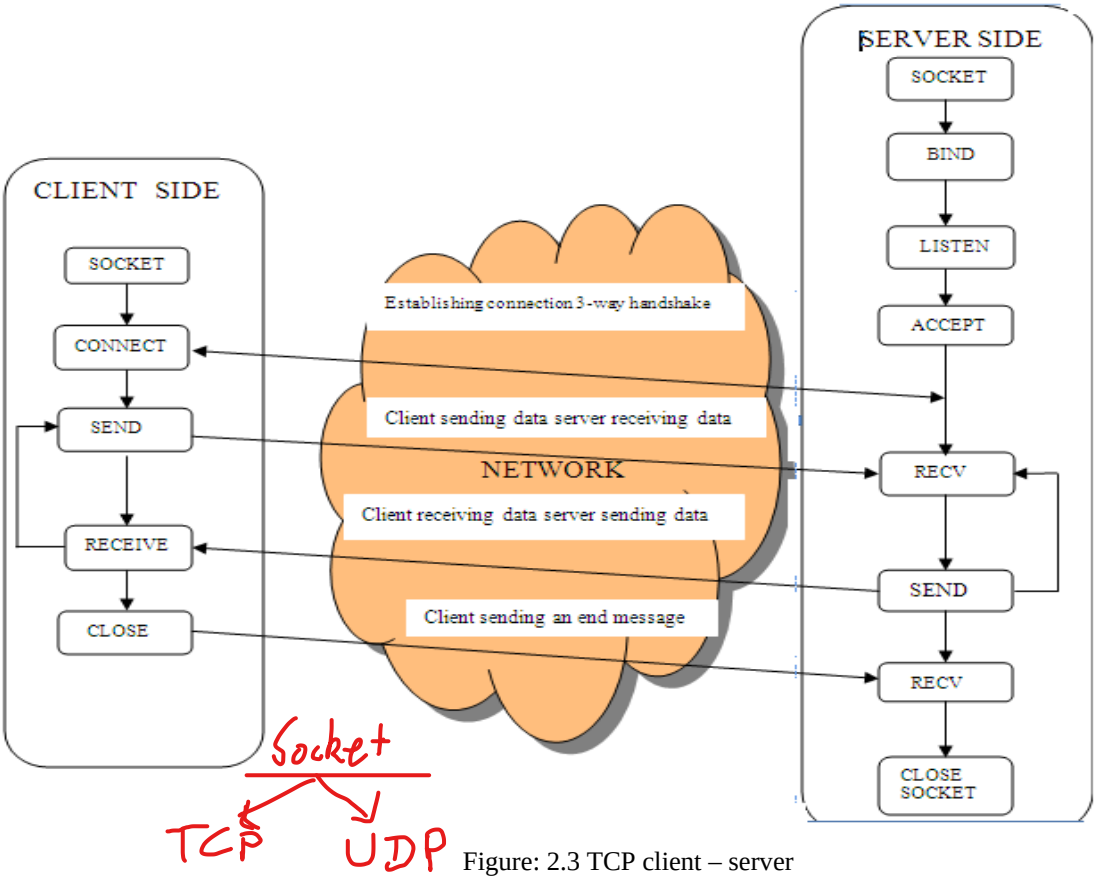

Figure: 2.3 TCP client – server

2.2 Socket programming over TCP in Java:

Java has provided the facility to create sockets for interprocess communication (IPC). So while programming for sockets in java, one has to make sure to import the java.net package. The java.net package in the Java platform provides a class, Socket that implements the client side connection. And a class ServerSocket that implements the server side connection. The Server Socket on the server performs the methods 'bind' which is to fix to a certain port no. and IP address, 'listen' to wait for incoming requests on the port and 'accept' for acceptance of connection from the client respectively. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint (i.e. socket) set to the name of the machine and port of the client. So the client initiates a three way handshake with the server and creates a TCP connection with the server. The client and server can now communicate by writing to or reading from their sockets. And when communication is done between the client and the server, the close method is called from both client and the server for closing the connection as shown in figure: 2.3 below:

The list is a summary of functions or methods provided by the Berkeley sockets API library:

- 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 sendto() and recvfrom(), 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.

2.3 Socket programming over UDP:

UDP is a connection-less, datagram protocol. The client does not establish a connection with the server like in case of TCP. Instead, the client just sends a datagram to the server using the sendto function which requires the address of the destination as a parameter. Similarly, the server does not accept a connection from a client. Instead, the server

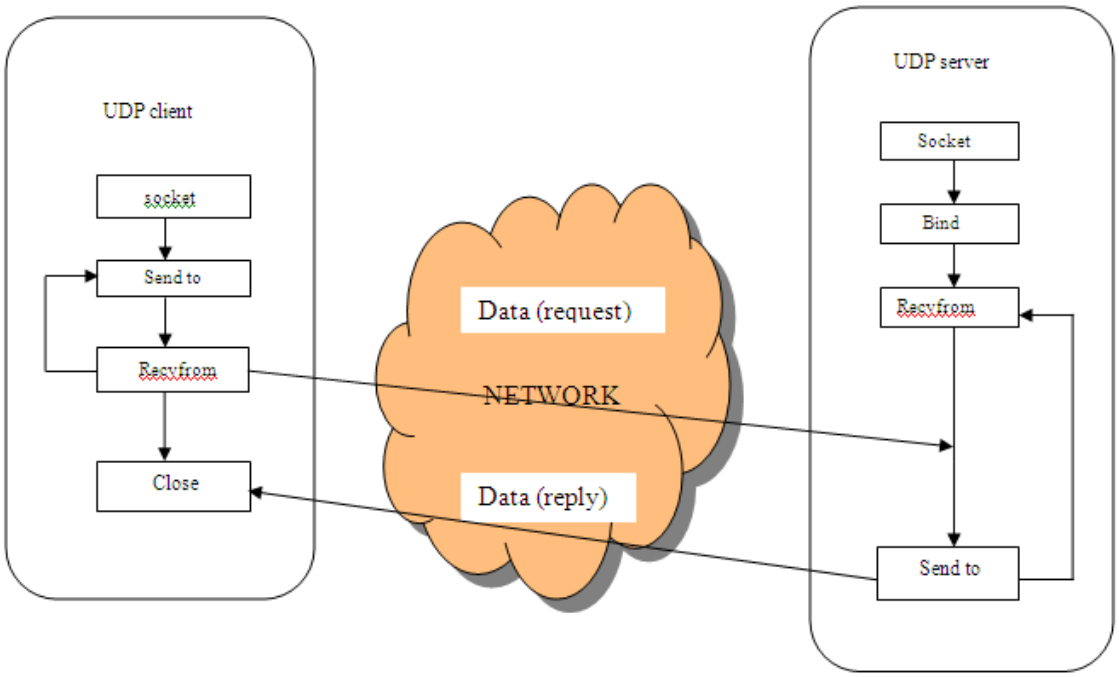

Figure: 2.4 UDP client - server

just calls the recvfrom function, which waits until data arrives from some client. recvfrom returns the IP address of the client, along with the datagram, so the server can send a response to the client as shown in figure:2.4. There isn't any initial handshaking phase. It lacks reliability because when you send a data or message, you don't know if it'll get there, it could get lost on the way. There may be corruption while transferring a message.

This list is a summary of functions or methods provided by the UDP Socket given below:

- socket():Both the client and the server creates the socket() function.
- bind():It is typically used on the server side, and bounds a socket with a socket address structure, i.e. a specified local port # and IP address
- listen():It is typically used on the server side passively waiting for incoming connections from the client.
- sendto():It is on both client side and server side .It is used to send a datagram to another UDP socket
- recvfrom(): It is on both client side and server side . It is used to receive a datagram from another UDP socket.
- close():close a socket(tear down the connection).

No Connect() on accept()

3. Conclusion:

This paper describes the details about sockets, ports, socket programming over TCP and a little bit of UDP. Network programming makes use of socket for interprocess communication between hosts where sockets act as the endpoint of the interprocess communication. Here sockets can also be termed as network socket or Internet socket since communication between computers is based on Internet protocol. So Network programming is also Socket Programming. The paper also describes about socket programming in java over TCP. Because java has been preferred more than any other language for establishing connections between clients and servers using sockets. Socket programming in java is easy.

References:

1. http://en.wikipedia.org/wiki/Berkeley_sockets.
2. James F. Kurose, Keith W. Ross, "Computer Networking: A Top-Down Approach featuring the Internet".
3. Joseph M. Dibella , "Socket Programming with Java"
4. http://www.tutorialspoint.com/java/java_networking.htm
5. The Java Tutorials, "Lesson: All about Sockets".
6. Rajkumar Buyya, "Socket Programming".