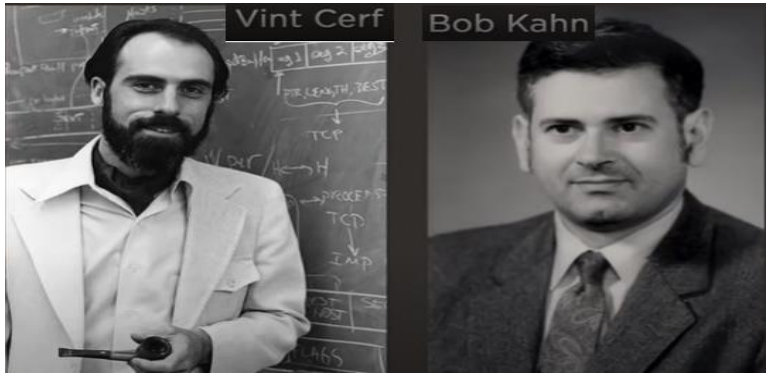
CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

What is the Internet?

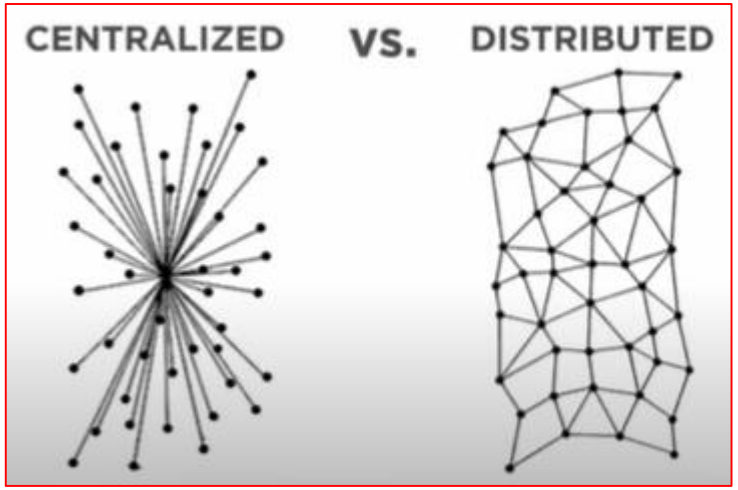
- Connection of devices on a network to share the information in worldwide.
- Who invented the Internet?
  - Vint Cerf
  - Bob khan

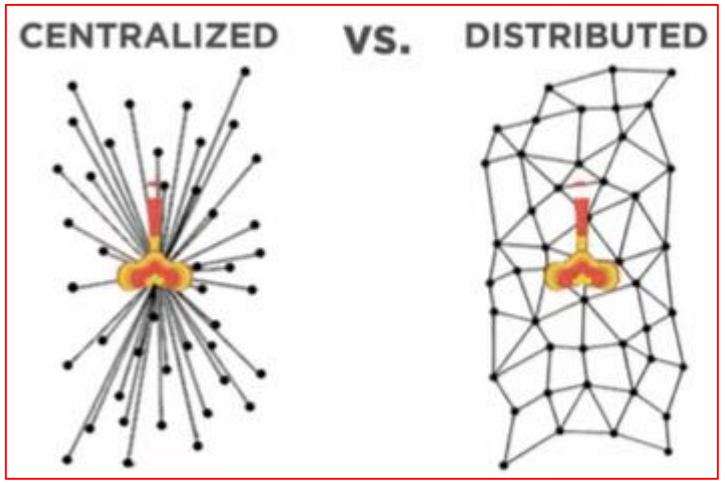


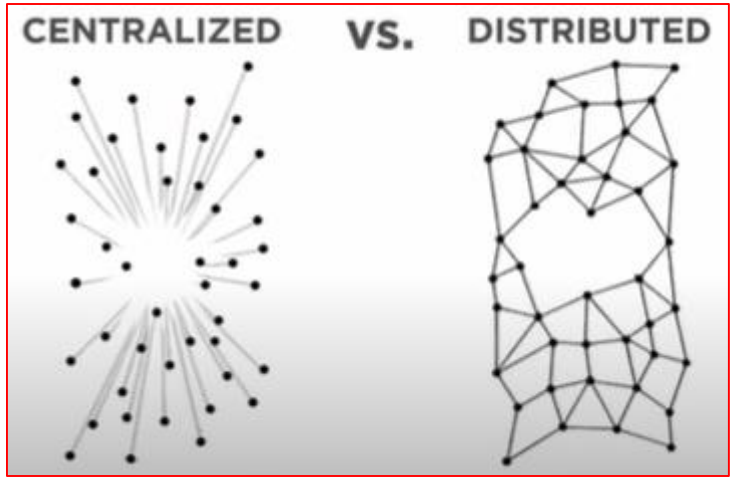
- The **Internet** is a global network of computing devices communicating with each other in some way, whether they're sending emails, downloading files, or sharing websites.

Introduction to Internet

- Internet is the result of the Defence research project ARPANET
- ARPANET
  - Advanced Research project Agency Network
  - To build a communication system that will survive a nuclear attack

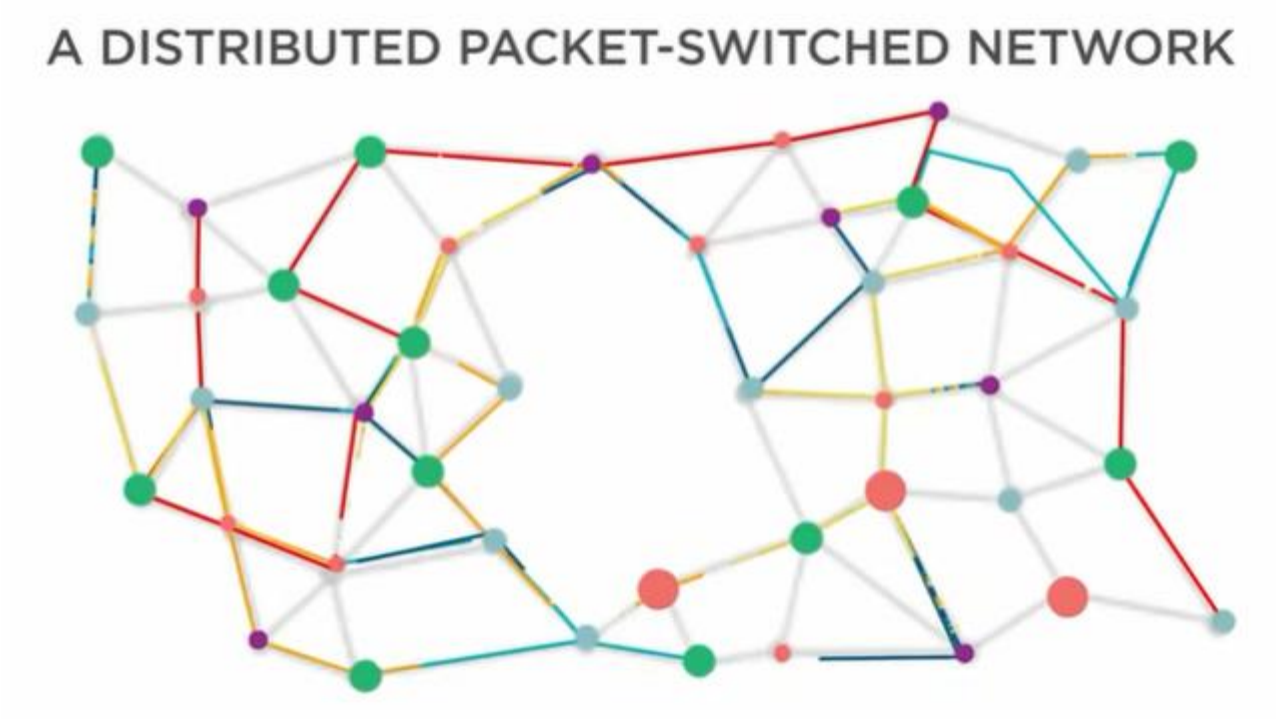






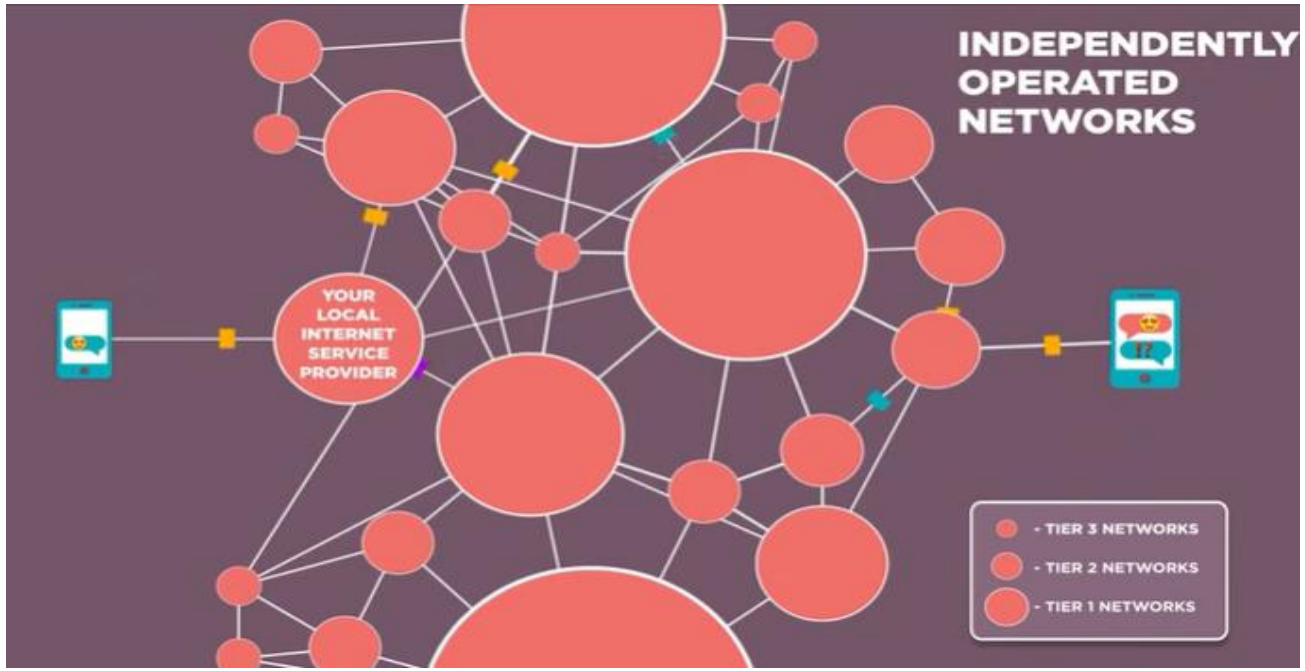
Introduction to Internet

- Break up the messages into blocks and sending them fast in possible direction through the mesh network.



Introduction to Internet

- Who is the in-charge of the Internet?
  - Nobody or everybody-i.e Internet is made up of large number independently operated networks.

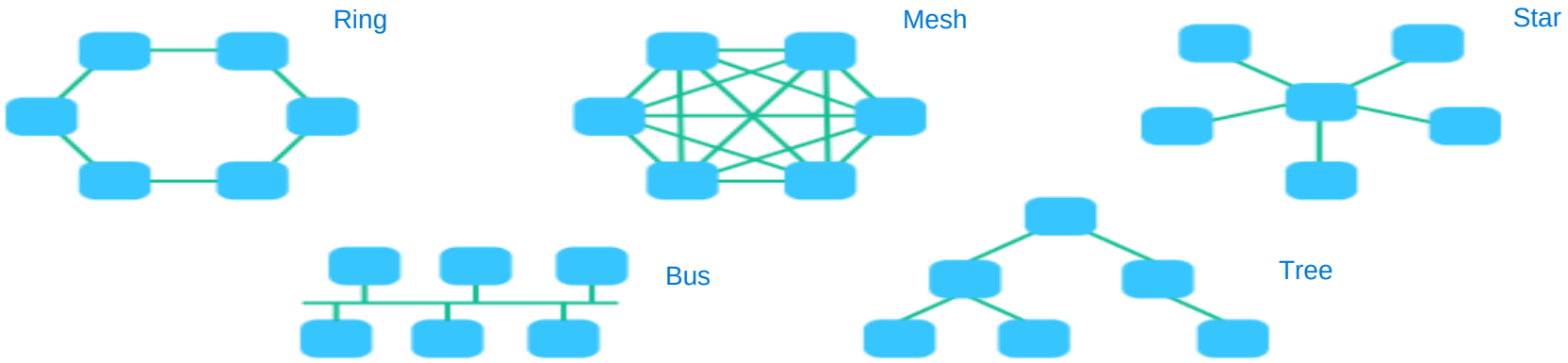


Introduction to Internet

- The Internet is an **open** network: any computing device can connect in network by following the rules of the Internet, the rules are known as **protocols** and they define how each device must communicate with each other.
- The Internet is powered by many layers of protocols.
  - **Wires & wireless**: The mode of physical connections between devices and the protocols for converting electromagnetic signals into binary data.
  - **IP**: A protocol that uniquely identify devices using IP addresses and provides a routing strategy to send data to a destination IP address.
  - **TCP/UDP**: Protocols that can transport packets of data from one device to another and check for errors along the way.
  - **TLS**: A secure protocol for sending encrypted data so that attackers can't view private information.
  - **HTTP & DNS**: The protocols powering the World Wide Web, what the browser uses every time you load a webpage.

Introduction to Internet

- A **computer network** is any group of interconnected computing devices capable of sending or receiving data.
- The different kinds of network topology of how a computing device(computer, phone, tablet) can connect .



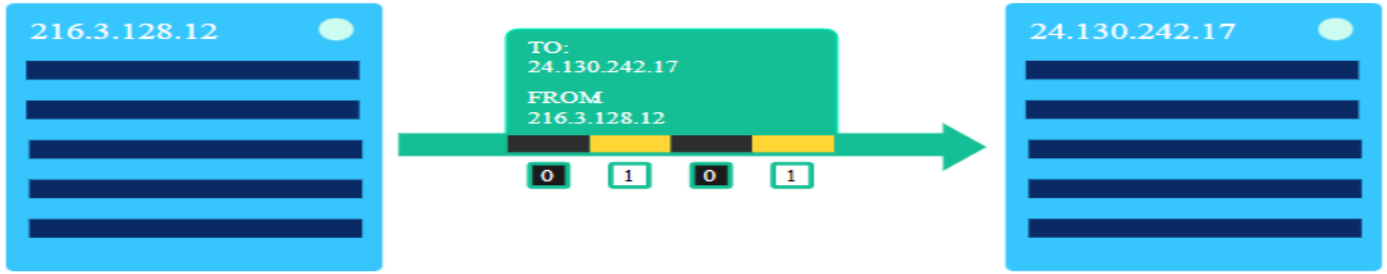
The top row shows the ring, mesh, and star topologies. The bottom row shows the bus and tree topologies.

Introduction to Internet

- **Bit rate**
  - Network connections can send bits very fast. We measure that speed using the **bit rate**, the number of bits of data that are sent each second. i.e 1Gbps
- **Bandwidth**
  - **Bandwidth** describes the maximum bit rate of a system. If a network connection has a bandwidth of 100 Mbps, that means it can't transfer more than 100 megabits per second.
- **Latency**
  - Another way to measure the speed of a computer network is **latency**. latency is the time between the sending of a data message and the receiving of that message, measured in milliseconds. INTERNET SPEED TEST website! Latency=24 ms
- **Internet speed**
  - Speed is a combination of bandwidth and latency. Computers split up messages into packets, and they can't send another message until the first packet is received. Even if a computer is on a connection with high bandwidth, its speed of sending and receiving messages will still be limited by the latency of the connection.

IP ADDRESS

- The **Internet Protocol (IP)** is used in all Internet communication to handle both addressing and routing.
- The protocol describes the use of **IP addresses** to uniquely identify Internet-connected devices, Internet-connected devices need an IP address to receive messages.
- When a computer sends a message to another computer, it must specify the recipient's IP address and also include its own IP address so that the second computer can reply.



IP ADDRESS

- There are actually two versions of the Internet Protocol in use today:
  - IPv4, the first version ever used on the Internet
  - IPv6, a backwards-compatible successor
- In the IPv4 protocol, IP addresses look like this:Each IP address is split into 4 numbers, and each of those numbers can range from 0 to 255.
  - IPV4
    - [0-255].[0-255].[0-255].[0-255]
    - IP v4address is represented by 32 bits, so there are 2^{32} possible IP v4 addresses(2^32 devices can connect to the Internet)
  - IPV6
    - FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF
    - 16^3+16^2+16^1+16^0=65535 16 bit each
    - IP v6 address is represented by 128 bits, so there are 2^{128} possible IP v6 addresses.

IP Protocol

- The Internet Protocol (IP) describes how to split messages into multiple IP packets and route packets to their destination by hopping from router to router.
- Each IP packet contains both a header (20 or 24 bytes long) and data (variable length). The header includes the IP addresses of the source and destination, plus other fields that help to route the packet. The data is the actual content, such as a string of letters or part of a webpage.
- IP does not handle all the consequences of packets, however. For example:
  - Receive **multiple messages** ,Packets can arrive **out of order,** Packets can be **corrupted**, Packets can be **lost** ,packets might be **duplicated.**

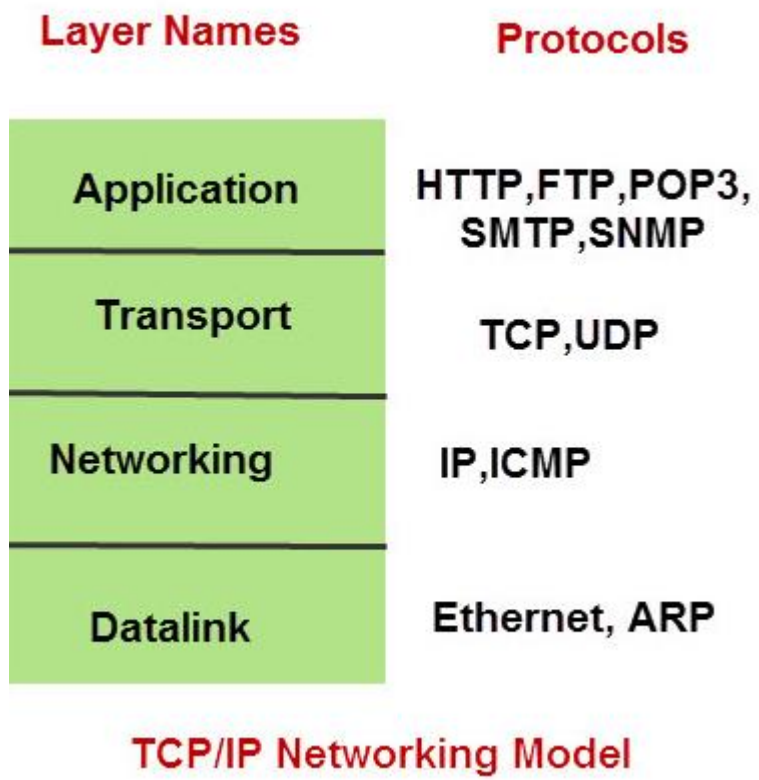
Transmission Control Protocol (TCP)

- The **Transmission Control Protocol (TCP)** is a transport protocol that is used on top of IP to ensure reliable transmission of packets.
- TCP includes mechanisms to solve many of the problems that arise from packet-based messaging, such as lost packets, out of order packets, duplicate packets, and corrupted packets.
- Since TCP is the protocol used most commonly on top of IP, the Internet protocol stack is sometimes referred to as **TCP/IP**.

User Datagram Protocol (UDP)

- The **User Datagram Protocol (UDP)** is a lightweight data transport protocol that works on top of IP.
- UDP is simple but fast, at least in comparison to other protocols that work over IP. It's often used for time-sensitive applications (such as real-time video streaming) where speed is more important than accuracy.
- UDP provides a mechanism to detect corrupt data in packets, but it does *not* attempt to solve other problems that arise with packets, such as lost or out of order packets.
- UDP is sometimes known as the ***Unreliable*** **Data Protocol**.
- Advantage: No connection establishment(stateless), broadcast or multicast,
- Disadvantage: No error correction, no flow control, no congestion control,

TCP/IP NETWORKING MODEL



TCP/IP Networking Model

TCP vs UDP

TCP

- Connected
- Byte stream
- Ordered data delivery
- Flow control
- Reliable
- Error free
- Relatively slow
- Guaranteed transmission
- secure
- Used by critical applications
- Acknowledgement
- DNS, FTP, SNMP, HTTP, HTTPS,SMTP

UDP

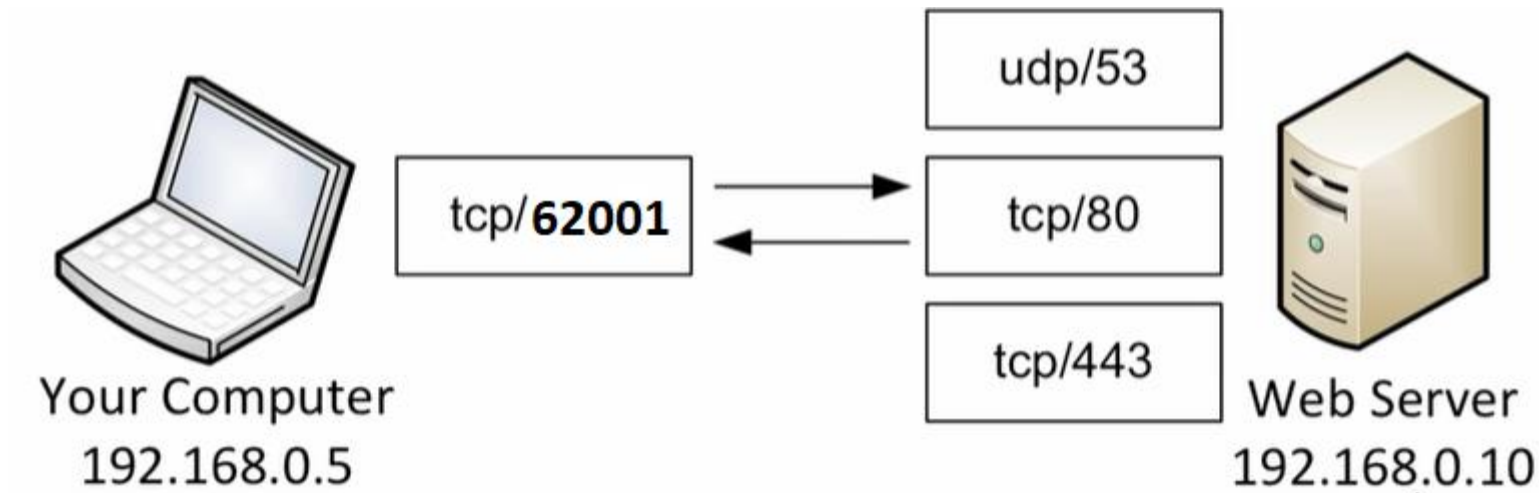
- Connectionless
- Datagram
- No sequence quarantee
- No flow control
- Unreliable
- Error packets discarded
- Relatively fast
- No guarantee
- Unsecure
- Used by real time applications
- No acknowledgement
- VoIP, DNS, RIP, SNMP

TCP/IP Ports and Sockets

- Every device on a TCP/IP network have an IP address.
    - The IP address identifies the device (e.g. computer) on the network.
- A computer can run multiple applications and/or services.
    - The network port identifies the application or service running on the computer.
- The use of ports allow computers/devices to run multiple services/applications.
    - A port number uses 16 bits and have a value from **0** to **65535** decimal
    - Port numbers are divided into ranges as follows:
    - **Port numbers 0-1023 – Well known ports.**
        - Allocated to **server services** by the **Internet Assigned Numbers Authority** (IANA).
        - e.g Web servers normally use **port 80** and SMTP servers use **port 25** (see diagram above).
    - **Ports 1024-49151- Registered Port** -
        - Registered for services with the **IANA** and should be treated as **semi-reserved.** User written programs should not use these ports.
    - **Ports 49152-65535**
        - Used by **client programs and are known as ephemeral ports.**
        - When a Web browser connects to a web server the browser will allocate itself a port in this range.

20-21-23-25-53-80-110-143-443

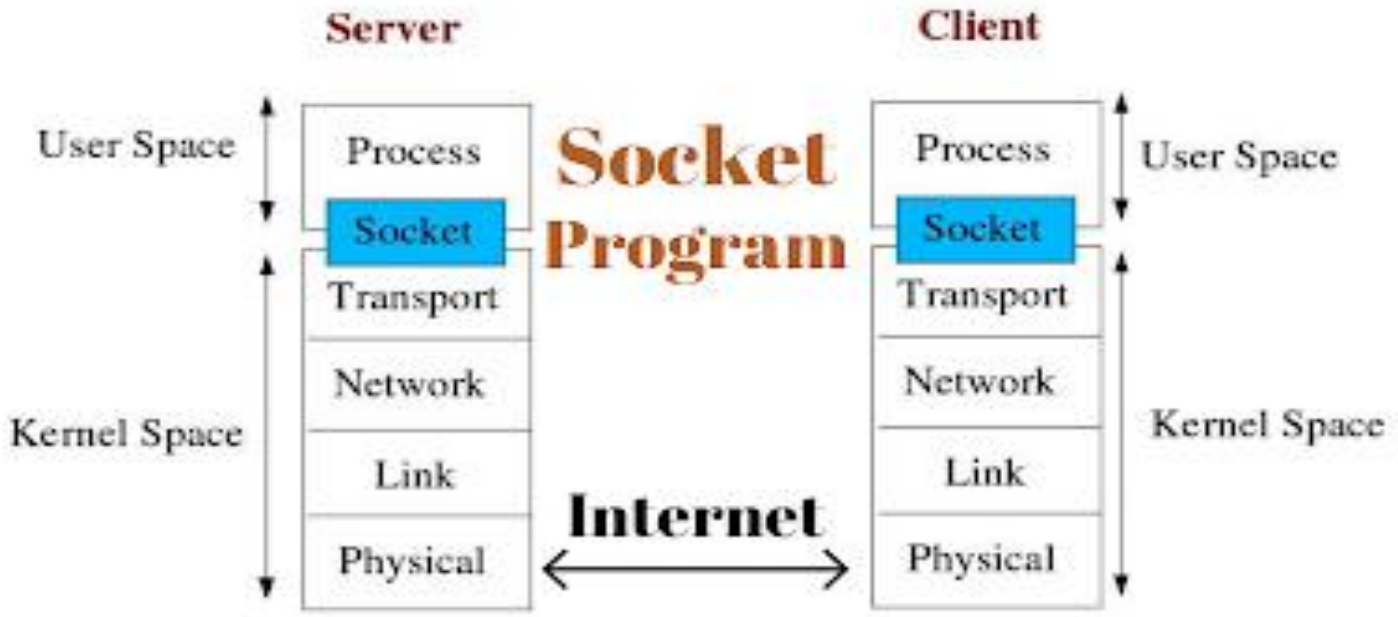
- FTP – File Transfer Protocol – tcp/20 (data), tcp/21 (control)
- Telnet – tcp/23
- SMTP – Simple Mail Transfer Protocol – tcp/25
- DNS – Domain Name Services – tcp/53 (zone transfers)
- HTTP – Hypertext Transfer Protocol – tcp/80
- POP3 – Post Office Protocol version 3 – tcp/110
- IMAP – Internet Message Access Protocol v4 – tcp/143
- HTTPS – Hypertext Transfer Protocol Secure – tcp/443



computer to computer connection with the IP addresses and ports.

Sockets

- A *socket* is a communications connection point (endpoint) in a network.



Source:codetextpro

Sockets

- A *socket* is a communications connection point (endpoint) in a network.
- A connection between two computers uses a **socket.**
- A socket is the combination of **IP address plus port**

The connection to Google would be:
Your PC IP Address + port 60400 ⟷ Google IP Address +port **80** (standard port)
The combination **IP address +60400** = the socket on the client computer
The combination **IP address + port 80** = the socket on the Google server.
The connection to Yahoo would be:
your PC IP Address + port 60801 ⟷ Yahoo IP Address + port **80** (standard port)
The combination **IP address +60801** = the socket on the client computer
The combination **IP address + port 80** = the socket on the yahoo server.
Client port numbers are dynamically assigned and can be reused once the session is closed.
Server port numbers and IP address are statically assigned.

Socket Types

- Two widely used socket types
  1. **Stream sockets**
     1. *Connection-oriented* communication implies that a connection is established, and a dialog between the programs follows. (much like a telephone call).
     2. server can assign a name to the service that it supplies, which allows clients to identify where to obtain and how to connect to that service.
     3. The client of the service (the client program) must request the service of the server program.
     4. The client does this by connecting to the distinct name or to the attributes associated with the distinct name that the server program has designated.
  2. **Datagram sockets**.
     1. *Connectionless* communication implies that no connection is established, over which a dialog or data transfer can take place.
     2. Instead, the server program designates a name that identifies where to reach it (much like a post-office box).
     - Stream sockets treat communications as a continuous stream of characters, while datagram sockets have to read entire messages at once.
- Each socket uses its own communications protocol.
  - Stream sockets use TCP (Transmission Control Protocol), a reliable, stream-oriented protocol
  - Datagram sockets use UDP (User Datagram Protocol), unreliable and message-oriented.

Server socket programming

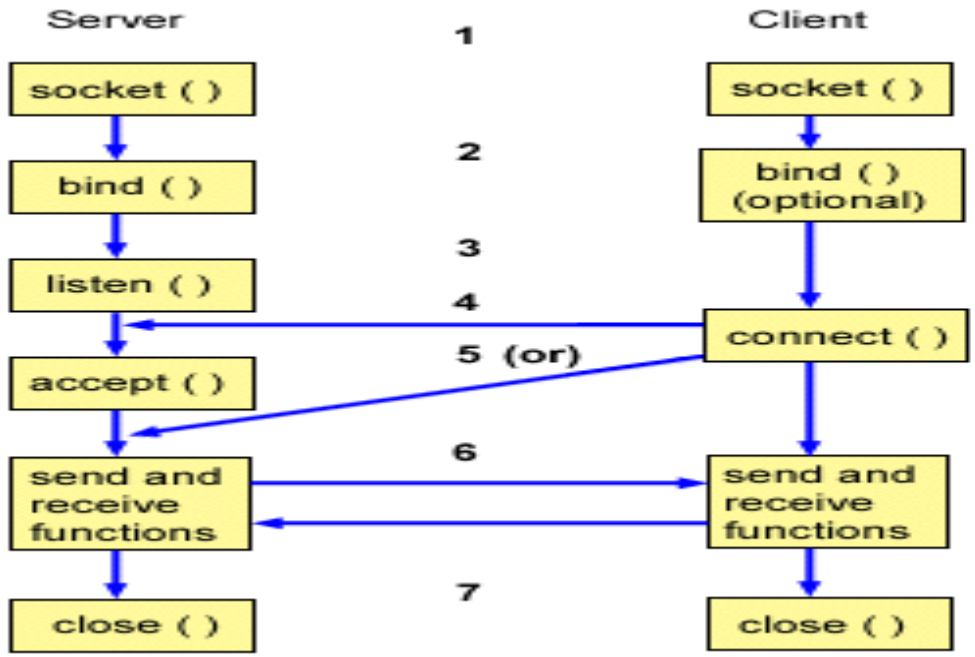
- **The steps in establishing a socket on the server-side :**
  - Create a socket with the **socket()** system call
  - Bind the socket to an address using the bind() system call.
  - For a server socket on the Internet, an address consists of a port number on the host machine.
  - Listen for connections with the listen() system call
  - Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.
  - Send and receive data, use the read() and write() system calls.

Client socket programming

- **The steps in establishing a Socket on the client side :**
  - Create a socket with the socket() system call.
  - Connect the socket to the address of the server using the connect() system call.
  - Send and receive data, use the read() and write() system calls.

Connection-oriented sockets

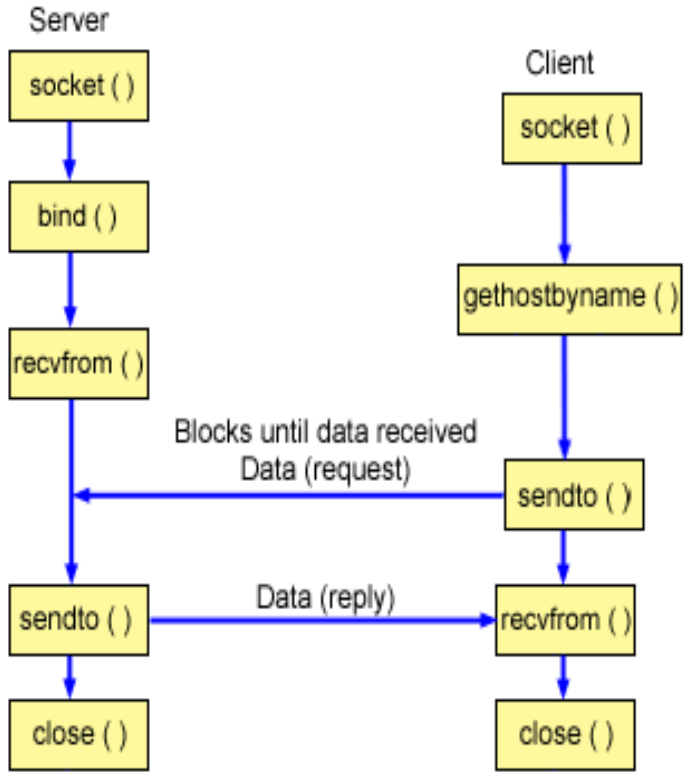
- *Connection-oriented* communication implies that a connection is established, and a dialog between the programs follows. (much like a telephone call).
- The server can assign a name to the service that it supplies, which allows clients to identify where to obtain and how to connect to that service.
- The client of the service (the client program) must request the service of the server program.
- The client does this by connecting to the distinct name or to the attributes associated with the distinct name that the server program has designated.



Source:IBM

Connectionless sockets

- Connectionless sockets do not establish a connection over which data is transferred. Instead, the server application specifies its name where a client can send requests.
- Connectionless sockets use User Datagram Protocol (UDP) instead of TCP/IP.
- The figure illustrates the client/server relationship of the socket APIs used in the examples for a connectionless socket design.



Source:IBM

Create a Server Socket:

```
ServerSocket serverSocket = new ServerSocket(6868);
```

Listen for a connection: Listen+Accept!

```
Socket socket = serverSocket.accept();
```

Read data from the client:

```
InputStream input = socket.getInputStream();
```

Send data to the client:

```
OutputStream output = socket.getOutputStream();
```

Close the client connection:

```
socket.close();
```

SERVER PROGRAM!
TCP!

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
public class SimpleServer {
    public static void main(String a[]){
        //listen to a port
        try {
            ServerSocket server = new ServerSocket(9999);
            System.out.println("server started");
            System.out.println("server waiting for client connection request");
            while(true) {
                try {
                    Socket socket = server.accept();
                    System.out.println("client request accepted");
                    DataInputStream in = new DataInputStream(socket.getInputStream());
                    DataOutputStream out = new DataOutputStream(socket.getOutputStream());
                    String clientMessage = in.readUTF();   //A message recieved from client!
                    out.writeUTF("Echo from server  " + clientMessage);   //sending message back to client!
                    out.close();
                    in.close();
                    socket.close();                server.close();
                }           catch(Exception e){              System.out.println(e);           }        }      }
    catch(Exception e){         System.out.println(e);      }   }}
```

CLIENT PROGRAM
TCP!

```
import java.io.DataOutputStream;
import java.net.Socket;
import java.io.DataInputStream;
import java.util.Scanner;
public class SimpleClient {
   public static void main(String a[]) throws Exception{
      try {
         //establish connection to a server
         Socket socket = new Socket("localhost", 9999);    //just  a socket!! not a SERVER SOCKET!
         //create streams to read and write from and to sockets
         DataInputStream in = new DataInputStream(socket.getInputStream());
         DataOutputStream out = new DataOutputStream(socket.getOutputStream());
         String clientInfo;
         Scanner input = new Scanner(System.in);
         clientInfo = input.nextLine();
         // write data to server
         out.writeUTF(clientInfo);
         out.flush();
         //read data from server
         String serverInfo = in.readUTF();
         System.out.println(serverInfo);
         in.close();
         out.close();
         socket.close();
      }       catch(Exception e){           System.out.println(e);       }    }}
```

CLIENT PROGRAM UDP!

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
public class UDPClient {
   public static void main(String a[]) throws Exception{
      DatagramSocket socket=new DatagramSocket();

      byte[] toServer=new byte[1024];
      byte[] fromServer=new byte[1024];
      int i=8;
      String str=new String(String.valueOf(i));      //Integer to string!
      toServer=str.getBytes();   //String to bytes!

      InetAddress ip=InetAddress.getByName("localhost");
      DatagramPacket ds=new DatagramPacket(toServer, toServer.length,ip,9999 );
      socket.send(ds);
      System.out.println("toServer"+ ds.getData().toString());

      DatagramPacket dr=new DatagramPacket(fromServer, fromServer.length);
      socket.receive(dr);
      String str1=new String(dr.getData(),0, dr.getLength());
      System.out.println("from server"+ str1);
      socket.close();
   }
}
```

Sending:

DatagramPacket(byte[] data, int length, InetAddress address, int port)

Recieving:

DatagramPacket(byte[] data, int length)

----------> The String constructor with these parameters (byte[] data, int offset, int length) creates a string using a portion of the byte array starting from the specified offset and using the specified length. The bytes are interpreted using the default platform character encoding (typically UTF-8 or UTF-16).

To convert a String to an int in Java, you can use the Integer.parseInt() method

UDP server!

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.nio.charset.StandardCharsets;
public class UDPServer {
    public static void main(String a[]) throws Exception{
        DatagramSocket socket=new DatagramSocket(9999);
        byte[] toClient=new byte[1024];
        byte[] fromClient=new byte[1024];
        DatagramPacket dr=new DatagramPacket(fromClient, fromClient.length);
        socket.receive(dr);
        String str=new String(dr.getData(),0, dr.getLength());
        System.out.println("from client"+ str);
        int i=Integer.parseInt(str);
        int result=i*i;
        System.out.println("result computed for " +i+"is " + result);
        toClient=String.valueOf(result).getBytes();
        DatagramPacket ds=new DatagramPacket(toClient, toClient.length,dr.getAddress(),dr.getPort());
        socket.send(ds);
        System.out.println("toClient"+ ds.getData());
        socket.close();
    }
}
```

NOTE IT:
-getAddress()
-getPort()

Email validation -SERVER(tcp)!

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
public class TCPServer {
public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(12345)) {
            System.out.println("Server is listening on port " + 12345);
            while (true) {
                try (Socket clientSocket = serverSocket.accept();
                     DataInputStream in = new DataInputStream(clientSocket.getInputStream());
                     DataOutputStream out = new DataOutputStream(clientSocket.getOutputStream())) {
                    // Read the email from the client
                    String email = in.readUTF();
                    // Validate the email
                    boolean isValidEmail = isValidEmail(email);
                    // Send the validation result back to the client
                    out.writeBoolean(isValidEmail);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        } catch (IOException e) {
            e.printStackTrace();     }   }
```

Port!

validation function!!

```
private static boolean isValidEmail(String email) {
        // Basic email validation without using regular expressions
        if (email == null || email.isEmpty()) {
            return false;
        }
        int atIndex = email.indexOf('@');
        int dotIndex = email.lastIndexOf('.');
// Check if "@" and "." are present, and "@" appears before "."
        if (atIndex > 0 && dotIndex > atIndex) {
            // Check the local part and domain part
            String localPart = email.substring(0, atIndex);
            String domainPart = email.substring(atIndex + 1);
            // Validate the local part and domain part
            boolean isValidLocal = isValidLocalPart(localPart);
            boolean isValidDomain =
isValidDomainPart(domainPart);
            // Close the streams and socket before returning
            return isValidLocal && isValidDomain;
        }
        return false;
    }
```

start-end!(substring)

string--->toChar array!

```
private static boolean isValidLocalPart(String localPart) {
        // Allow alphanumeric characters, dots, and
underscores
        for (char c : localPart.toCharArray()) {
            if (!(Character.isLetterOrDigit(c) || c == '.' || c ==
'_')) {
                return false;
            }
        }
        return true;
    }
    private static boolean isValidDomainPart(String
domainPart) {
        // Allow alphanumeric characters, dots, and hyphens
        for (char c : domainPart.toCharArray()) {
            if (!(Character.isLetterOrDigit(c) || c == '.' || c == '-
')) {
                return false;
            }
        }
        return !domainPart.startsWith(".") &&
!domainPart.endsWith(".");
    }
}
```

TCP CLIENT!!

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
public class TCPClient {
public static void main(String[] args) {
      try (Socket socket = new Socket(“localhost, 12345);
          DataInputStream in = new DataInputStream(socket.getInputStream());
          DataOutputStream out = new DataOutputStream(socket.getOutputStream())) {
         // Get the email from the user
         String email = "user@example.com";
         // Send the email to the server
         out.writeUTF(email);
         // Receive the validation result from the server
         boolean isValidEmail = in.readBoolean();
         // Display the result
         if (isValidEmail) {
            System.out.println("The email is valid.");
         } else {
            System.out.println("The email is not valid.");
         }
      } catch (IOException e) {
         e.printStackTrace();       }    }}
```

MULTI THREADING+TCP!!!

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
public class TCPServer {
public static void main(String[] args) {
      ExecutorService executorService =
Executors.newFixedThreadPool(2);
      try (ServerSocket serverSocket = new ServerSocket(12345)) {
         System.out.println("Server is listening on port " + 12345);
         while (true) {
            try {
               Socket clientSocket = serverSocket.accept();
               executorService.submit(new
ClientHandler(clientSocket));
            } catch (IOException e) {
               e.printStackTrace();
            }
         }
      } catch (IOException e) {
         e.printStackTrace();
      } finally {          executorService.shutdown();       }   }}
```

```
class ClientHandler implements Runnable {
   private final Socket clientSocket;

   public ClientHandler(Socket clientSocket) {
      this.clientSocket = clientSocket;
   }

   @Override
   public void run() {
      try (DataInputStream in = new
DataInputStream(clientSocket.getInputStream());
           DataOutputStream out = new
DataOutputStream(clientSocket.getOutputStream())) {

         // Read the number from the client
         double number = in.readDouble();

         // Calculate the square root
         double result = Math.sqrt(number);

         // Send the result back to the client
         out.writeDouble(result);
      } catch (IOException e) {
         e.printStackTrace();       }   }}
```

InetAddress Class

- The InetAddress class is used to encapsulate both the numerical IP address and the domain name for that address.
- The InetAddress class hides the number inside.
- InetAddress can handle both IPv4 and IPv6 addresses.

```
static InetAddress getLocalHost( ) throws UnknownHostException
static InetAddress getByName(String hostName) throws UnknownHostException
static InetAddress[ ] getAllByName(String hostName) throws UnknownHostException
```

  - The getLocalHost( ) method simply returns the InetAddress object that represents the local host.
  - The getByName( ) method returns an InetAddress for a host name passed to it.
  - The getAllByName( ) factory method returns an array of InetAddresses that represent all of the addresses that a particular name resolves to.
  - If these methods are unable to resolve the host name, they throw an UnknownHostException.

UDP

- Java implements datagrams on top of the UDP protocol by using two classes:
- **DatagramPacket** object is the data container
- **DatagramSocke**t is the mechanism used to send or receive the DatagramPackets.

Datagram Socket

- No handshaking between server and client
- Sender explicitly attaches IP address and port of destination to the packet!!
- Receiver(server) must extract IP address, port of sender from received datagram.

DatagramSocket( ) throws SocketException
DatagramSocket(int port) throws SocketException
DatagramSocket(int port, InetAddress ipAddress) throws SocketException
DatagramSocket(SocketAddress address) throws SocketException

- The first creates a DatagramSocket bound to any unused port on the local computer.
- The second creates a DatagramSocket bound to the port specified by port.
- The third constructs a DatagramSocket bound to the specified port and InetAddress.
- The fourth constructs a DatagramSocket bound to the specified SocketAddress.
- SocketAddress is an abstract class that is implemented by the concrete class InetSocketAddress. InetSocketAddress encapsulates an IP address with a port number.
- All can throw a SocketException if an error occurs while creating the socket.

Datagram Socket

- DatagramSocket defines many methods.
- Two of the most important are send( ) and receive()

void send(DatagramPacket packet) throws IOException
void receive(DatagramPacket packet) throws IOException

- The send( ) method sends a packet to the port specified by packet.
- The receive( ) method waits for a packet to be received and returns the result.
- DatagramSocket also defines the close( )method, which closes the socket.
- DatagramSocket also implements AutoCloseable, which means that a DatagramSocket can be managed by a try-with-resources block.

Datagram Socket

- Methods give you access to various attributes associated with a DatagramSocket.

InetAddress getInetAddress( )	If the socket is connected, then the address is returned. Otherwise, **null** is returned.
int getLocalPort( )	Returns the number of the local port.
int getPort( )	Returns the number of the port connected to the socket. It returns –1 if the socket is not connected to a port.
boolean isBound( )	Returns **true** if the socket is bound to an address. Returns **false** otherwise.
boolean isConnected( )	Returns **true** if the socket is connected to a server. Returns **false** otherwise.
void setSoTimeout(int *millis*) throws SocketException	Sets the time-out period to the number of milliseconds passed in *millis*.

Datagram Sockets: UDP

- DatagramPacket

```
DatagramPacket(byte data [ ], int size) ;
DatagramPacket(byte data [ ], int size, InetAddress ipAddress, int port);
```

- The first constructor specifies a buffer that will receive data and the size of a packet.
  - used for receiving data over a DatagramSocket.
- The second form specifies a target address and port
  - used by a DatagramSocket to determine where the data in the packet to be sent.

Methods of DatagramPacket

InetAddress getAddress( )	Returns the address of the source (for datagrams being received) or destination (for datagrams being sent).
byte[ ] getData( )	Returns the byte array of data contained in the datagram. Mostly used to retrieve data from the datagram after it has been received.
int getLength( )	Returns the length of the valid data contained in the byte array that would be returned from the **getData( )** method. This may not equal the length of the whole byte array.
int getOffset( )	Returns the starting index of the data.
int getPort( )	Returns the port number.
void setAddress(InetAddress *ipAddress*)	Sets the address to which a packet will be sent. The address is specified by *ipAddress*.
void setData(byte[ ] *data*)	Sets the data to *data*, the offset to zero, and the length to number of bytes in *data*.
void setData(byte[ ] *data*, int *idx*, int *size*)	Sets the data to *data*, the offset to *idx*, and the length to *size*.
void setLength(int *size*)	Sets the length of the packet to *size*.
void setPort(int *port*)	Sets the port to *port*.

```java
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Scanner;

public class UDPClient {
    public static StringBuilder data(byte[] a)
    {
        if (a == null)
            return null;
        StringBuilder ret = new StringBuilder();
        int i = 0;
        while (a[i] != 0)
        {
            ret.append((char) a[i]);
            i++;
        }
        return ret;
    }
```

```java
    public static void main(String args[]) throws IOException, NullPointerException {
        DatagramSocket cs=new DatagramSocket();
        DatagramPacket sendPacket=null;
        DatagramPacket receivePacket=null;
        byte[] receiveData=new byte[1024];
        Scanner in = new Scanner(System.in);
        System.out.println("Enter bye to Quit");
        while(in.hasNextLine()){
            InetAddress IPAddress=InetAddress.getByName("localhost");
            int port=4000;
            String message=in.nextLine();
            byte[] sendData=message.getBytes();
            sendPacket=new DatagramPacket(sendData, sendData.length, IPAddress, port);
            cs.send(sendPacket);
            receivePacket=new DatagramPacket(receiveData,receiveData.length);
            cs.receive(receivePacket);
            System.out.println("from Server:"+ data(receiveData));
            receiveData=new byte[1024];
        }

    }
}
```

output

```
Enter bye to Quit
jayachitra v p
from Server:JAYACHITRA V P
hello
from Server:HELLO
hi
from Server:HI
bye
```

UDP Client!!

```java
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
public class UDPServer {
    public static StringBuilder data(byte[] a)
    {
        if (a == null)
            return null;
        StringBuilder ret = new StringBuilder();
        int i = 0;
        while (a[i] != 0)
        {
            ret.append((char) a[i]);
            i++;
        }
        return ret;
    }
```

```java
    public static void main(String args[]) throws IOException {
        byte[] dataReceive=new byte[1024];
        DatagramSocket ss=new DatagramSocket( port: 4000);
        DatagramPacket receivePacket=null;
        DatagramPacket sendPacket=null;
        while(true) {
            receivePacket = new DatagramPacket(dataReceive, dataReceive.length);
            ss.receive(receivePacket);
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String message = new String(receivePacket.getData());
            String modifiedMessage = message.toUpperCase();
            byte[] dataSend = modifiedMessage.getBytes();
            System.out.println("from Client:"+data(receivePacket.getData()));
            sendPacket = new DatagramPacket(dataSend, dataSend.length, IPAddress, port);

            if (data(dataReceive).toString().equals("bye"))
            {
                System.out.println("Client sent bye.....EXITING");
                break;
            }
            else
                ss.send(sendPacket);
            dataReceive=new byte[1024];
        }
    }
}
```

output

```
from Client:jayachitra v p
from Client:hello
from Client:hi
from Client:bye
Client sent bye.....EXITING

Process finished with exit code 0
```

UDP server!

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;
import java.util.concurrent.Executors;
public class CapitalizeServer {
public static void main(String[] args) throws Exception {
      try (var listener = new ServerSocket(50000)) {
         System.out.println(“Server is running...");
         ExecutorService pool =
               Executors.newFixedThreadPool(20);
         while (true) {
            pool.execute(new Capitalizer(listener.accept()));
         }
      }
   }
```

execute/submit!

```
class Capitalizer implements Runnable {
      private Socket socket;
      Capitalizer(Socket socket) {
         this.socket = socket;  }
        public void run() {
         System.out.println("Connected: " + socket);
         try {
            var in = new Scanner(socket.getInputStream());
            var out = new PrintWriter(socket.getOutputStream(), true);
            while (in.hasNextLine()) {
               out.println(in.nextLine().toUpperCase());
            }
         } catch (Exception e) {
            System.out.println("Error:" + socket);
         } finally {
               socket.close();
            }
       }
      }
```

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class CapitalizeClient {
    public static void main(String[] args) throws Exception {
        try (var socket = new Socket(“localhost”, 50000)) {
         System.out.println("Enter lines of text then Ctrl+D or Ctrl+C to quit");
         var scanner = new Scanner(System.in);
         var in = new Scanner(socket.getInputStream());
         var out = new PrintWriter(socket.getOutputStream(), true);
         while (scanner.hasNextLine()) {
            out.println(scanner.nextLine());
            System.out.println(in.nextLine());
         }
      }
   }
}
```