

Bule Hora University
Faculty of Engineering and Technology
Department of Information technology

Year: III

Semester: II

Subject: Introduction to Distributed System
ITec3102

Course Code:

Lecture Notes Unit II - Issues and Architectures

- 1.1 Characterization of Distributed systems
- 1.2 Architectures
- 1.3 Networking and Internetworking
- 1.4 Inter-process Communication

1.1. Characterization of Distributed Systems

Difference between Computer network and Distributed system

- ⌘ *Computer Network*: the independent computers are explicitly visible (can be explicitly addressed).
- ⌘ *Distributed System*: existence of multiple independent computers is transparent.

Why Distributed System?

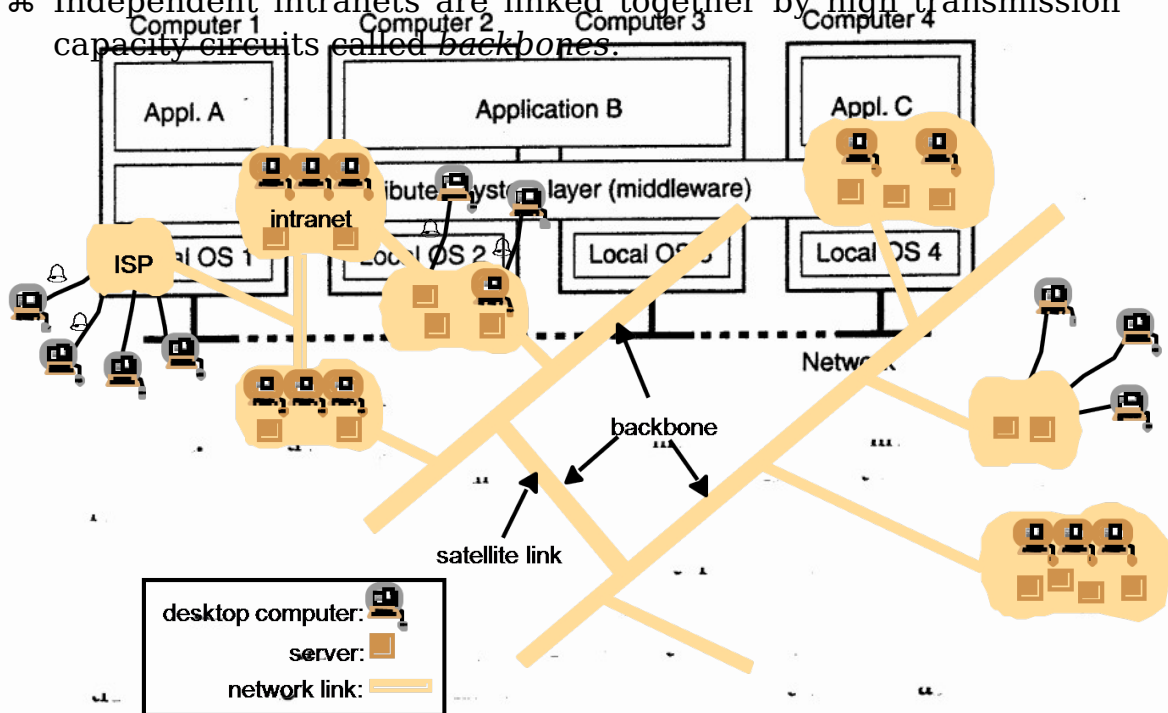
- ⌘ *Functional* distribution: computers have different functional capabilities (i.e., *sharing of resources* with specific functionalities).
- ⌘ *Load* distribution / balancing: assign tasks to processors such that the overall system performance is optimized.
- ⌘ Replication of *processing power*: independent processors working on the same task:
- ⌘ *Physical* separation: systems that rely on the fact that computers are physically separated
- ⌘ *Economics*: collections of microprocessors offer a better price/performance ratio than large mainframes.

Examples

1. Internet

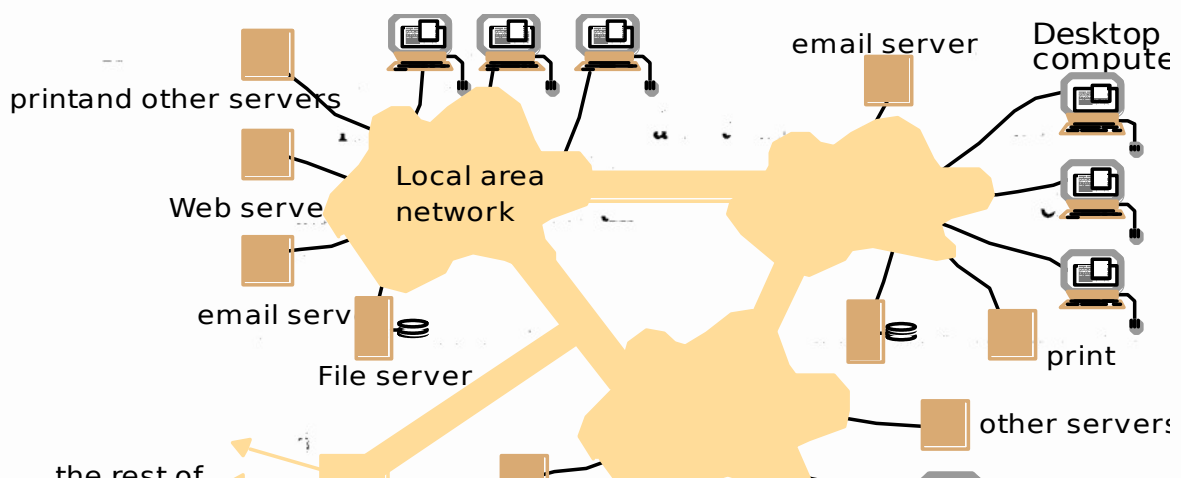
- ⌘ The Internet is a vast interconnected collection of computer networks of many types.

- ⌘ Its design enabling a program running anywhere to address messages to programs anywhere else.
- ⌘ Allowing its users to make use of many services as: WWW, E-Mail, Web hosting, and File transfer.
- ⌘ Its services can be extended by adding new types of service
- ⌘ Small organizations and individual users can access to internet services through *Internet Service Providers* (ISPs).
- ⌘ Independent intranets are linked together by high transmission capacity circuits called *backbones*.



2. Intranet

- ⌘ An Intranet is a portion of the internet that is administrated separately and its local security policies are enforced by a configured boundary.
- ⌘ Composed of several local area networks (LANs) linked by backbone connections to allow its users to access the provided services.
- ⌘ Connected to the Internet via a *router* which allows its users to make use of the internet services elsewhere.
- ⌘ Many organization protect their own services from unauthorized use by filtering incoming and outgoing messages using a *firewall*.



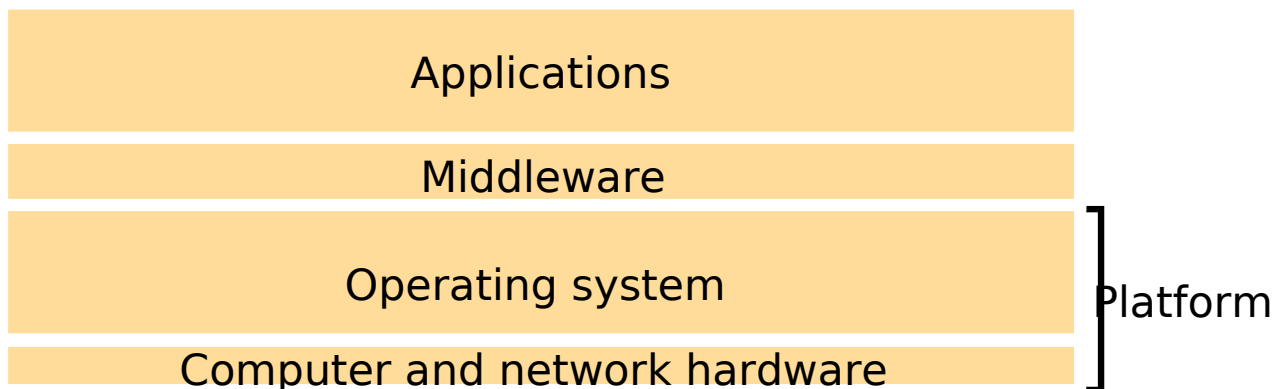
3. Mobile and Ubiquitous Computing

- ⌘ *Mobile computing* is the performance of computing tasks while the users are on the move and away from their residence intranet but still provided with access to resources via the devices they carry with them.
- ⌘ *Ubiquitous computing* is the harnessing of many small cheap computational devices that are present in user's physical environments.

1.2. Architectures

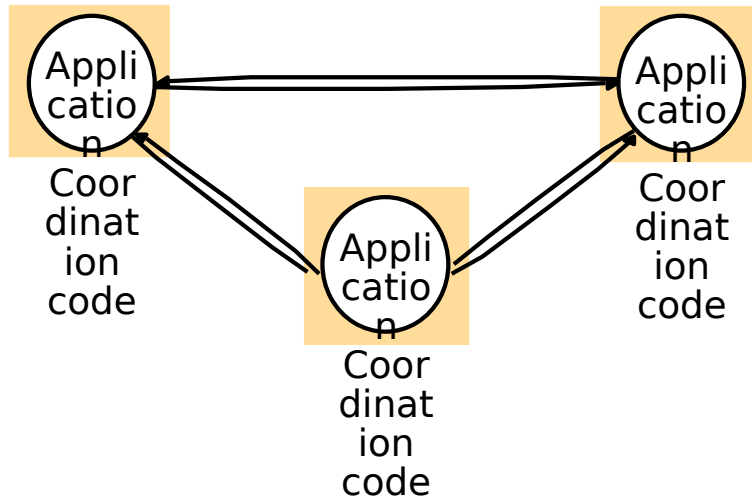
1. Layered Architecture

- ⌘ In the layered view of a system each layer offers its services to the level above and builds its own service on the services of the layer below.
- ⌘ Software architecture is the structuring of software in terms of layers (modules) or services that can be requested locally or remotely.



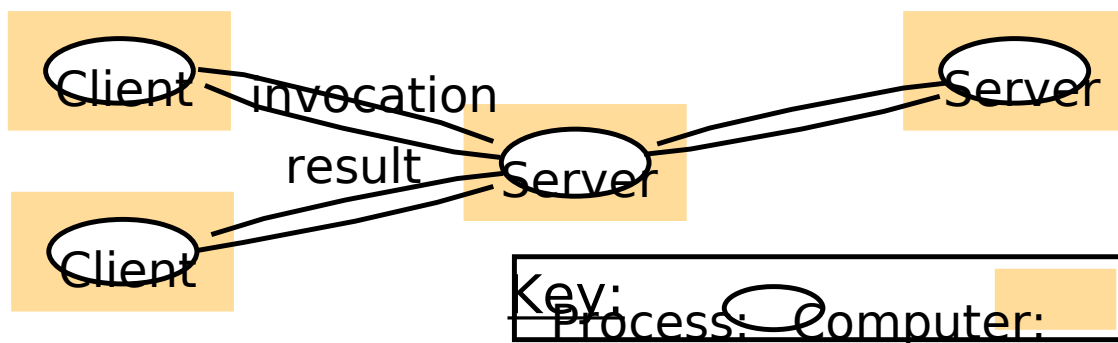
2. Peer -Peer Architecture:

- ⌘ All processes play similar roles without distinction as a client or a server.
- ⌘ Interacting cooperatively to perform a distributed activity.
- ⌘ Communications pattern will depend on application requirements.

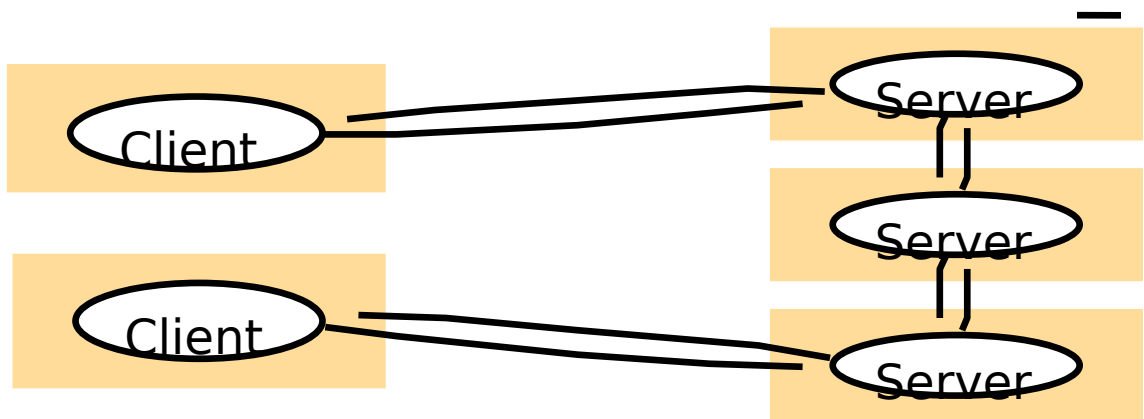


3. Client Server Architecture

- ⌘ Most important and most widely distributed system architecture.
- ⌘ Client and server roles are assigned and changeable.
 - Servers may in turn be clients of other servers.
- ⌘ Services may be implemented as several interacting processes in different host computers to provide a service to client processes:
 - Servers partition the set of objects on which the service is based and distribute them among themselves (e.g. Web data and web servers)
 - Servers maintain replicated copies of the service objects on several hosts for reliability and performance.

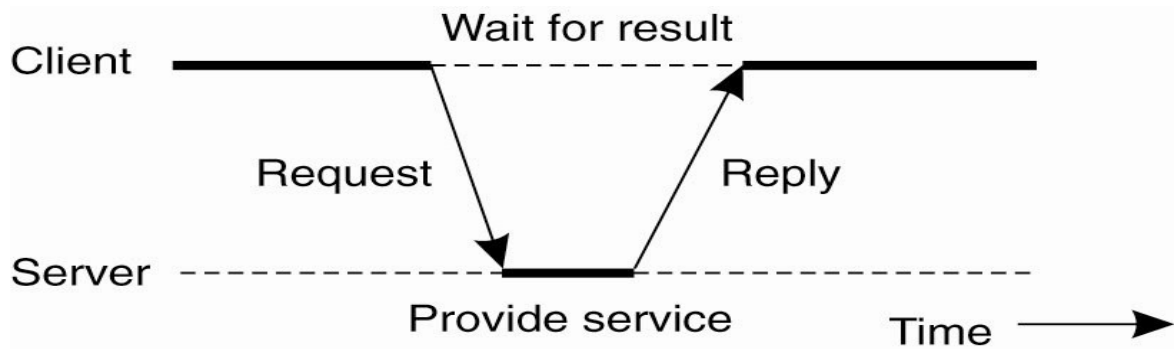


Clients invoke individual servers



A service provided by multiple servers

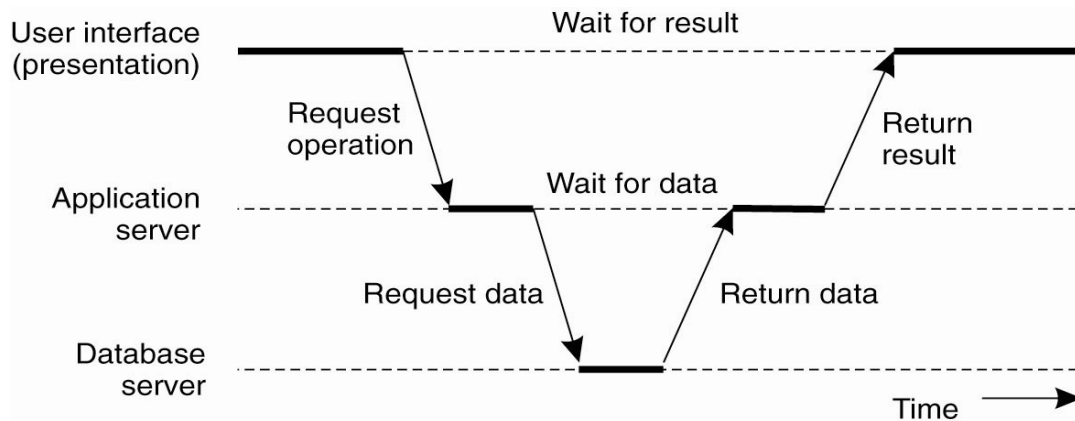
General interaction between a client and a server is shown below



4. Multitiered Architectures

⌘ It contains three tiers

1. Presentation
2. Application server
3. Database server



5. Object Oriented Architecture

- ⌘ In OO paradigm, distributed processes can be constructed as objects whose methods can be accessed by remote invocation (COBRA approach).
- ⌘ Many objects can be encapsulated in server or peer processes.
- ⌘ Number, types, and locations (in some implementation) of objects may change dynamically as system activates require.
- ⌘ Therefore, new services can be instantiated and immediately be made available for invocation.

1.3. Networking and Internetworking

Communication Subsystem

- ⌘ The hardware and software within a distributed system which provides the communication facilities is known as the *communication subsystem*.
- ⌘ Consists of:
 - Transmission media: providing the physical connectivity, e.g. wire, cable, fibre and wireless channels;
 - Hardware devices: providing the linkage, e.g. routers, bridges, hubs, repeaters, network interfaces and gateways;

- Software components: managing the communication, e.g. protocol stacks, communication handlers and drivers.

Impact on Distributed Systems

- ⌘ The communication impact on a distributed system will be one of the delay introduced by the message passing (Message passing is the most fundamental paradigm for distributed applications. A process sends a message representing a request. The message is delivered to a receiver, which processes the request, and sends a message in response).
- ⌘ The delay experienced by each individual message can be broken down into two factors:
 - Latency: is the time which is necessary to set up the communication, i.e. it is the delay incurred from the time the message is sent until it starts to arrive at the destination.
 - Transmission delay: determined by the length of the message and the *data transfer rate*, the speed of data transfer between two computers in the network, usually in bits per second.

Message transmission time = latency + length / data transfer rate

- ⌘ Above equation is valid only for messages shorter than the maximum allowable length by the underlying network.

Longer messages are segmented and the transmission time is the summation of segments transmission times.

Network Types

- ⌘ Local Area Networks (LANs)
 - High-speed communication on proprietary grounds (on-campus).
 - Based on twisted copper wire, coaxial cable or optical fibre.
 - Most typical solution: Ethernet with 100 Mbps
- ⌘ **Metropolitan Area Networks (MANs)**

- High-speed communication for nodes distributed over medium-range distances, usually belonging to one organization.
- Based on high bandwidth copper and optical fibre.
- Providing "back-bone" to interconnect LAN's.
- Technology often based on ATM, FDDI or DSL.

⌘ **Wide Area Networks**

- Communication over long distances (cities, countries, or continents).
- Covers computers of different organizations.
- Involves routers to manage network and route messages to their destinations.
- Speeds up to a few Mbps possible, but around 50-100 Kbps more typical.
- Most prominent example: the Internet.

⌘ **Wireless Networks**

- End user equipment accesses network through short or mid range radio or infrared signal transmission
- Wireless WANs:
 - ⌘ GSM (up to about 20 Kbps), UMTS (up to Mbps), PCS.
- Wireless LANs/MANs:
 - ⌘ WaveLAN (2-11 Mbps, radio up to 150 meters).
- Wireless Personal Area Networks:
 - ⌘ Bluetooth (up to 2 Mbps on low power radio signal, < 10 m distance).

Internetworks

- Several networks linked together to provide common data communication facilities.
- Needed for developing open distributed systems that contain very large numbers of computers.
- Interconnected by dedicated switching computers, *routers*, and general purpose computers, *gateways*.

Networking Principles

Packet transmission

- ⌘ A *packet* is a sequence of binary data with addressing information to identify the source and destination computers.
- ⌘ A network message with arbitrary length is divided before transmission into packets of restricted length.

Switching Schemes

- ⌘ A *switching* system is required to transmit information between two arbitrary nodes in the network using shared communications link.
- ⌘ Four types of switching are used in computer network:
 - Broadcast:
 - Requires no switches.
 - All messages are sent to all connected computers.
 - Each computer is responsible extracting messages addressed to itself.
 - Used approach in Ethernet and wireless networks.
 - Circuit switching:
 - Approach taken in the telephone system.
 - A physical link is established between the sender and the receiver.
 - Packet switching:
 - Otherwise known as store-and-forward (postal system).

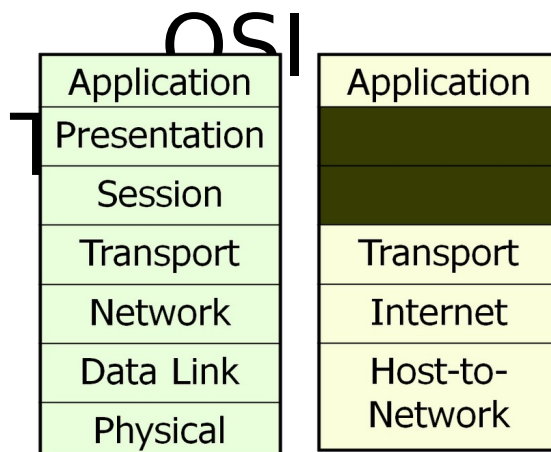
- At each switching node (connection point) a computer manages the packets by reading each one into memory, examining its destination, and choosing an outgoing circuit appropriately.

■ Frame relay:

- Reading in and storing the whole of each packet introduces a performance overhead which can become significant.
- In ATM networks a frame of fixed size is used in place of a packet and only its header needs to be examined.
- The remainder of the frame is simply relayed as a stream of bits.

Protocols

- ⌘ A well-known set of rules and formats used for communication between processes to perform a given task.
- ⌘ A complete set of protocol layers is referred to as a *protocol suite* or *protocol stack*.



Each of the Seven Layers of the OSI Model Represents Functions Required for Communication

Number	Name	Interpretation
1	Physical Layer	how devices are physically connected

2	Data Link Layer	how neighboring devices communicate
3	Network Layer	how non-neighboring devices communicate
4	Transport Layer	how to avoid communication errors
5	Session Layer	how devices managing ongoing communication
6	Presentation Layer	how devices agree on the format of data
7	Application Layer	how devices agree on the meaning of data

Internetworking:

Internetworking devices:

⌘ Routers:

- Responsible for forwarding the internetwork packets arrived on any connection to the correct outgoing connection and maintain routing tables for that purpose.

⌘ Bridges:

- Link networks of different types.

Internet Protocols

- ⌘ TCP is a reliable connection-oriented protocol used to transport streams of data.
- ⌘ Another transport protocol (User Datagram Protocol UDP) is used to meet traditional message-based communication.
- ⌘ IP is the underlying network protocol that provide the basic transmission mechanism for the Internet and other subnets.

IP Addressing

Used scheme for assigning addresses to networks and the computers connected to them.

- IPv4 (1984): 32-bit addresses for 2^{32} (~ 4 billion) addresses, but insufficient due to:

i) Unforeseen growth of internet. ii) Inefficient use of address space.

- IPv6 (1994): 128-bit addresses for 2^{128} ($\sim 3 \times 10^{38}$) addressable nodes.

1.4. Inter Process Communication

IPC refers Communication between two different processes by message passing.

1.4.1. Characteristics of IPC

Message passing between a pair of processes supported by two communication operations: *Send* and *Receive*.

⌘ A queue is associated with each message destination regarded as a buffer between the sender and receiver:

- When the buffer is empty the receiver must wait.
- When the buffer is full the sender must wait, or messages will be lost.

⌘ Communication is termed as synchronous or asynchronous depending on the degree of coordination imposed between sender and receiver:

- **Synchronous:** both send and receive are blocking operations:
 - ⌘ Sender blocks until a receive is issued
 - ⌘ Receiver blocks until a message arrives.
- **Asynchronous:** send operation is non-blocking
 - ⌘ Sender is non-blocking (copy goes to its local buffer)
 - ⌘ Receiver blocking (ALWAYS) or non-blocking (out of control flow).

1.4.2. Ports and Sockets

⌘ Communication between processes is made between ports.

- Each computer has 2^{16} possible ports represented by integer numbers:

- some ports have specific use but others are available for general use.

- Each port corresponds to a single receiving process, but each process may have more than one port at which it receives.

- Any number of senders can send messages to a port.

⌘ *Sockets* are software abstractions of ports used within running processes.

- Messages are sent between a pair of sockets.
- Sockets need to be bound to a port number and a host IP address in order to be useable for sending and receiving messages.
- Socket binding may be automatic (Java) or done explicitly (BSD UNIX).
- Each socket is associated with a particular transport protocol, i.e. UDP (datagrams) or TCP (streams).

1.4.3. **IPC Communication using UDP**

UDP is suitable for applications that need low overheads and accepting occasional omission failures.

- ⌘ A datagram is communicated between processes when one process sends it and the other receives it.
- ⌘ Datagrams sent from socket to socket without acknowledgement or reliability.
- ⌘ A server binds its socket to a specific *server port* known to clients but a client binds its socket to any free port.
- ⌘ Sender socket information is passed with the datagram content so that a reply can be sent.
- ⌘ The receiver must specify a buffer of a particular size (8 KBytes in most implementations) to place received message.
- ⌘ Datagram communication provides non-blocking sends and blocking receives although timeouts can be set.
- ⌘ Timeouts can be set on receiver socket to forbid indefinitely waiting in case of crashed sender or lost message.

Disadvantages of UDP transmission

- ⌘ Datagrams can be dropped because of
 - Full Buffers
 - Checksum Failure
- ⌘ Messages can arrive out of order

UDP Applications

- ⌘ Where failures are tolerable
 - DNS
 - VOIP
- ⌘ Where overheads are costly

1.4.3.1. Java API for Internet Addresses

- The java.net package is intended to provide an application programmer with a powerful infrastructure for networking:
- Java provides three basic socket classes:
 - DatagramSocket for datagram communication and
 - Both of ServerSocket and Socket for stream (TCP) communication.

Java includes classes for representing URLs (the URL class) and Internet addresses (the InetAddress class).

```
InetAddress aComputer =  
InetAddress.getByName("wassif.ccis.ksu.edu.sa");
```

Java API for UDP Datagrams

⌘ Java API provides datagram communication by two classes:

■ *DatagramPacket*:

- Has a constructor to create datagram packets comprising a message, its length, and destination socket information
- Has another constructor to put a received message into that packet.
- The message can be retrieved using *getData* method and the Internet address and the port are accessed by *getAddress* and *getPort* methods.

■ *DatagramSocket*:

- Supports sockets for sending and receiving UDP datagrams.
- Has a constructor with a port number as argument to tie that port, if a particular port is needed, or without argument to use any free local port.
- *send* and *receive* methods are used for transmitting datagrams between a pair of sockets.

- The method *setSoTimeout* allows a timeouts to be set and *connect* method is used for connecting a socket to a particular remote port and host address.

UDP client sends two numbers to the server and gets a sum of them

```
// import required java packages
import java.io.*;
import java.net.*;

public class udpclientadd
{
    public static void main(String args[])throws IOException
    {

        try {
            // Declare necessary variables
            byte[] num1=new byte[100];
            byte[] num2=new byte[100];
            byte[] sum=new byte[100];
            BufferedReader br=new BufferedReader(new
            InputStreamReader(System.in));

            // create a datagram socket
            DatagramSocket ds=new DatagramSocket();

            //get IP address of server
            InetAddress ipaddress=InetAddress.getByName("localhost");

            //Read first number
```

```

System.out.println("\nEnter first number");
String s1=br.readLine(); // read as string
num1=s1.getBytes(); // get bytes from the string


//Create a datagram packet of first number and send to the
server

DatagramPacket dp1=new DatagramPacket(num1,num1.length,
ipaddress, 9876);
ds.send(dp1);


//Read second number
System.out.println("\nEnter Second number");
String s2=br.readLine(); // read as string
num2=s2.getBytes(); //get bytes from the string


//Create a datagram packet of second number and send to the
server

DatagramPacket dp2=new DatagramPacket(num2,num2.length,
ipaddress, 9876);
ds.send(dp2);


//Create a datagram packet and receive the sum from the server
DatagramPacket    dpreceive=new    DatagramPacket(sum,
sum.length);
ds.receive(dpreceive);


String s=new String(dpreceive.getData()); // get the received
data as a string

int result=Integer.parseInt(s.trim()); // convert the string to
Integer

```

```

        System.out.println("\noutput is\t"+result); // print the result on
        console screen
    ds.close(); //close the socket
}

    catch (SocketException e)
    {
        System.out.println("Socket: " +
e.getMessage());
    }

}

}

```

UDP server receives two numbers from the client and sends the sum back to the client

```

// import required java packages
import java.io.*;
import java.net.*;

public class udpserveradd
{
    public static void main(String args[])throws IOException
    {
        try
        {
            // Declare necessary variables
            byte[] num1=new byte[100];

```

```
byte[] num2=new byte[100];
byte[] sum=new byte[100];

// create a datagram socket and wait for the client request
DatagramSocket ds=new DatagramSocket(9876);
System.out.println("\nWaiting for the client request");

//Create a datagram packet and receive the first number from the client
DatagramPacket dpreceive=new DatagramPacket(num1, num1.length);
ds.receive(dpreceive);

String s1=new String(dpreceive.getData()); // get the received data as a
string
int x=Integer.parseInt(s1.trim()); // convert the string to Integer

//Create a datagram packet and receive the second number from the client
DatagramPacket dpreceive1=new DatagramPacket(num2, num2.length);
ds.receive(dpreceive1);

String s2=new String(dpreceive1.getData()); // get the received data as a
string
int y=Integer.parseInt(s2.trim()); // convert the string to Integer

int z=x+y; // perform addition of two numbers

String s=Integer.toString(z); // convert the number to string
```

```

sum=s.getBytes(); // get bytes of the string

//Create a datagram packet of the result and send to the client
DatagramPacket dp1=new
DatagramPacket(sum,sum.length,dpreceive.getAddress

(),dpreceive.getPort());
ds.send(dp1);
System.out.println("\nResult sent to client");

ds.close(); //close the socket
}

catch (SocketException e)
{
    System.out.println("Socket: " +
e.getMessage());
}
}
}

```

1.4.4. IPC Communication using TCP

- TCP offers the following characteristics:
 - o No restriction about message sizes:
 - o Guarantee message delivery using acknowledgment scheme:
 - o Attempt to match the speeds of reader and writer processes:
 - o Guarantee unduplication and ordering of messages:
 - o Message identifiers enable the receiver to detect and reject duplicates or to reorder messages not arrived in sender order

- ⌘ A pair of communicating processes establish a connection before they can communicate over a stream:
 - The client role involves:
 - Creating a stream socket bound to any free port.
 - Making a *connect* request to a server at its server port.
 - The server role involves:
 - Creating a listening socket bound to a server port and waiting for clients to request connections.
 - Making *accept* request to reply a client request at its port.
 - Creating a new stream socket to communicate with this client.
 - Keeping the listening socket for new connecting requests.
 - Once the connection is established the processes write to and read from the stream without needing to use Internet addresses and ports.

Disadvantages

- ⌘ Timeout and retransmission
- ⌘ Connections can break down
- ⌘ No guarantee that connection will not break, so not fully reliable

Applications

Many services such as FTP, HTTP and SMTP use TCP

1.4.4.1. Java API for TCP Streams

- ⌘ Java API provides stream communication by two classes:

- *ServerSocket*:

- Intended for use by a server to create a socket at a server port for listening *connect* request from clients.
- Has an *accept* method that is blocked until arriving of any request to create an instance of stream *socket*.

- *Socket*:

- Used by a pair of processes with a stream connection.
- Has a constructor used by the client to create its stream socket and send a connect request to the specified remote server.

- Provides *getInputStream* and *getOutputStream* methods for accessing the two input and output streams associated with a socket.
- *read* and *write* methods are used for bytes reading from and writing to *InputStream* and *OutputStream* respectively.

TCP client sends two numbers to the server and gets a sum of them

```
// import required java packages
```

```
import java.io.*;
```

```
import java.net.*;
```

```
public class tcpclientadd
```

```
{
```

```
public static void main(String args[])throws IOException
```

```
{
```

```
try {
```

```
BufferedReader br=new BufferedReader(new  
InputStreamReader(System.in));
```

```
//get IP address of server
```

```
InetAddress ipaddress=InetAddress.getByName("localhost");
```

```
// create a socket
```

```
Socket s=new Socket(ipaddress,7876);
```

```
//Access input and output streams associated with sockets
```

```

DataInputStream in = new DataInputStream( s.getInputStream());
DataOutputStream out = new DataOutputStream( s.getOutputStream());

//Read first number
System.out.println("\nEnter first number");
String s1=br.readLine(); // read as string
out.writeUTF(s1);// write the first number as string into the output stream

//Read second number
System.out.println("\nEnter Second number");
String s2=br.readLine(); // read as string
out.writeUTF(s2);// write the second number as string into the output
stream

//Read the output from the server as string with input stream
String sum= in.readUTF();
int result=Integer.parseInt(sum.trim());// convert string to integer
System.out.println("\noutput is\t"+result); // print the result on console
screen
s.close();//close the socket
}
catch (SocketException e)
{
    System.out.println("Socket: " +
e.getMessage());
}

```

```
}  
}  
  
}
```

TCP server receives two numbers from the client and sends the sum back to the client

```
// import required java packages
```

```
import java.io.*;
```

```
import java.net.*;
```

```
public class tcpserveradd
```

```
{
```

```
public static void main(String args[])throws IOException
```

```
{
```

```
try
```

```
{
```

```
// create a server socket and wait for the client request
```

```
ServerSocket ss = new ServerSocket(7876);
```

```
System.out.println("\nWaiting for the client request");
```

```
Socket ss1 = ss.accept(); // accept the connection from the client
```

```
//Access input and output streams associated with sockets
```

```
DataInputStream in = new DataInputStream( ss1.getInputStream());
```

```
DataOutputStream out = new DataOutputStream( ss1.getOutputStream());
```

```
String s1 = in.readUTF(); // Read first number as string with input stream
```

```
int x=Integer.parseInt(s1.trim()); // convert the string to Integer
```

```
String s2 = in.readUTF();// Read second number as string with input stream
```

```
int y=Integer.parseInt(s2.trim()); // convert the string to Integer
```

```
int z=x+y; // perform addition of two numbers
```

```
String sum=Integer.toString(z); // convert the number to string
```

```
out.writeUTF(sum);// write the sum in to the output stream
```

```
System.out.println("\nResult sent to client");
```

```
ss1.close();//close the socket
```

```
}
```

```
catch (SocketException e)
```

```
{
```

```
System.out.println("Socket: " +
```

```
e.getMessage());
```

```
}
```

```
}
```

}