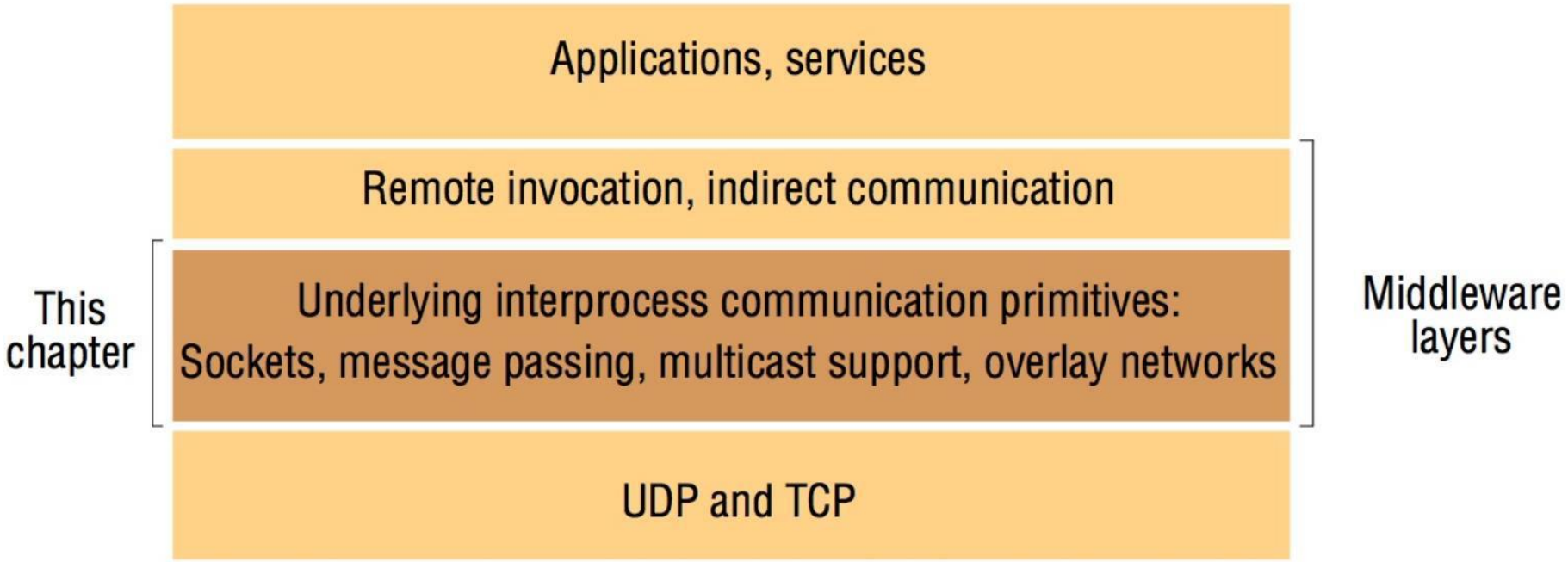
Distributed Computing

Module -3

Course Plan			
Module	**Contents**	**Hours**	**End Sem. Exam Marks**
I	Evolution of Distributed Computing -Issues in designing a distributed system- Challenges- Minicomputer model – Workstation model - Workstation-Server model– Processor - pool model - Trends in distributed systems	7	15%
II	System models: Physical models - Architectural models - Fundamental models	6	15%
FIRST INTERNAL EXAM			
III	Interprocess communication: characteristics – group communication - Multicast Communication –Remote Procedure call - Network virtualization. Case study : Skype	7	15%
IV	Distributed file system: File service architecture - Network file system- Andrew file system- Name Service	7	15%
SECOND INTERNAL EXAM			
V	Transactional concurrency control:- Transactions, Nested transactions-Locks-Optimistic concurrency control	7	20%
VI	Distributed mutual exclusion – central server algorithm – ring based algorithm- Maekawa's voting algorithm – Election: Ring -based election algorithm – Bully algorithm	7	20%
END SEMESTER EXAM			

Interprocess communication

- Figure: Middleware layers



- How middleware and application programs can use UDP and TCP?
- What is specific about IP multicast? Why/how could it be made more reliable?
- What is an overlay network? Skype an example

Interprocess communication

- **The API for the Internet protocols**
- *The characteristics of interprocess communication*
- **Synchronous and Asynchronous communication**
- A queue is associated with each message destination.
- Sending processes cause messages to be added to remote queues and receiving processes remove messages from local queues
  - Synchronous –
    - Sending and receiving processes synchronize at every message
    - both send and receive – blocking operations
    - – whenever send is issued – sending process blocked until receive is issued
    - – whenever receive is issued by a process, it is blocked until the message arrives
  - Asynchronous
    - - In the asynchronous form of communication, the use of the send operation is nonblocking
    - - the sending process is allowed to proceed as soon as the message has been copied to a local buffer,
    - - and the transmission of the message proceeds in parallel with the sending process.
    - The receive operation can have blocking and non-blocking variants.

Interprocess communication

- In the non-blocking variant, the receiving process proceeds with its program after issuing a receive operation, which provides a buffer to be filled in the background,
- but it must separately receive notification that its buffer has been filled, by polling or interrupt

- In case threads are supported (Java) blocking receive has no disadvantages
- – a separate thread is handling the communication while other threads can continue their work.
- today's systems do not generally provide the non-blocking receive

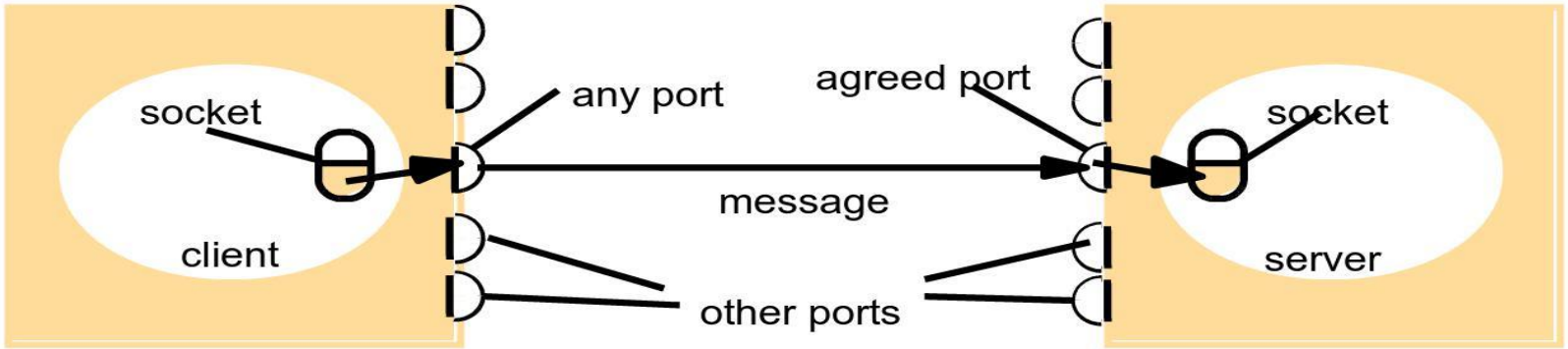
- **Message Destinations**
  - Messages are sent to (Internet address, local port) pairs.
  - A local port is a message destination within a computer, specified as an integer. A port has exactly one receiver (multicast ports are an exception) but can have many senders.
  - Processes may use multiple ports to receive messages.
  - Any process that knows the number of a port can send a message to it. Servers generally publicize their port numbers for use by clients.

Interprocess communication

- **Reliability**
  - Reliable communication in terms of validity and integrity.
  - A point-to-point message service can be described as reliable if messages are guaranteed to be delivered despite a ‘reasonable’ number of packets being dropped or lost.
  - For integrity, messages must arrive uncorrupted and without duplication.
- **Ordering**
  - Messages must be delivered in sender order
  - – that is, the order in which they were transmitted by the sender.
  - The delivery of messages out of sender order is regarded as a failure by such applications.

Interprocess communication

- **Sockets**
  - Both forms of communication (UDP and TCP) use the *socket* abstraction, which provides an endpoint for communication between processes.
  - Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process, as illustrated in Figure.
  - For a process to receive messages, its socket must be bound to a local port and one of the Internet addresses of the computer on which it runs.
  - Messages sent to a particular Internet address and port number can be received only by a process whose socket is associated with that Internet address and port number.
  - Processes may use the same socket for sending and receiving messages.
  - Each computer has a large number (2^16) of possible port numbers for use by local processes for receiving messages.



Multicast communication

- A multicast operation is more appropriate – this is an operation that sends a single message from one process to each of the members of a group of processes, usually in such a way that the membership of the group is transparent to the sender.
- The simplest multicast protocol provides no guarantees about message delivery or ordering.
- Multicast messages provide a useful infrastructure for constructing distributed systems with the following characteristics:
  - *Fault tolerance based on replicated services*:
    - A replicated service consists of a group of servers.
    - Client requests are multicast to all the members of the group, each of which performs an identical operation.
    - Even when some of the members fail, clients can still be served.
  - *Discovering services in spontaneous networking*:
    - Multicast messages can be used by servers and clients to locate available discovery services in order to register their interfaces or to look up the interfaces of other services in the distributed system.

- *Better performance through replicated data*:
  - Data are replicated to increase the performance of a service
  - – in some cases replicas of the data are placed in users' computers.
  - Each time the data changes, the new value is multicast to the processes managing the replicas.
- *Propagation of event notifications*:
  - Multicast to a group may be used to notify processes when something happens.
  - For example, in Facebook, when someone changes their status, all their friends receive notifications.
  - Publish-subscribe protocols may make use of group multicast to disseminate events to subscribers.

- **IP multicast**
  - *IP multicast* is built on top of the Internet Protocol (IP).
  - Note that IP packets are addressed to computers – ports belong to the TCP and UDP levels.
  - IP multicast allows the sender to transmit a single IP packet to a set of computers that form a multicast group.
  - The sender is unaware of the identities of the individual recipients and of the size of the group.
  - A *multicast group* is specified by a Class D Internet address– that is, an address whose first 4 bits are 1110 in IPv4.

IP multicast – A Multicast Communication

- Being a member of a multicast group allows a computer to receive IP packets sent to the group.
- The membership of multicast groups is dynamic, allowing computers to join or leave at any time and to join an arbitrary number of groups.
- It is possible to send datagrams to a multicast group without being a member.

- At the application programming level, IP multicast is available only via UDP.
  - An application program performs multicasts by sending UDP datagrams with multicast addresses and ordinary port numbers.
  - It can join a multicast group by making its socket join the group, enabling it to receive messages to the group.
- At the IP level, a computer belongs to a multicast group when one or more of its processes has sockets that belong to that group.
  - When a multicast message arrives at a computer, copies are forwarded to all of the local sockets that have joined the specified multicast address and are bound to the specified port number.

IP multicast – A Multicast Communication

- *Multicast routers*:
  - IP packets can be multicast both on a local network and on the wider Internet. Local multicasts use the multicast capability of the local network, for example, of an Ethernet.
  - Internet multicasts make use of multicast routers, which forward single datagrams to routers on other networks, where they are again multicast to local members.
  - To limit the distance of propagation of a multicast datagram, the sender can specify the number of routers it is allowed to pass – called the *time to live*, or TTL for short.
- *Multicast address allocation*:
  - Class D addresses (that is, addresses in the range 224.0.0.0 to 239.255.255.255) are reserved for multicast traffic and managed globally by the Internet Assigned Numbers Authority (IANA).
  - Local Network Control Block (224.0.0.0 to 224.0.0.225), for multicast traffic within a given local network.
  - Internet Control Block (224.0.1.0 to 224.0.1.225).
  - Ad Hoc Control Block (224.0.2.0 to 224.0.255.0), for traffic that does not fit any other block.
  - Administratively Scoped Block (239.0.0.0 to 239.255.255.255), which is used to implement a scoping mechanism for multicast traffic (to constrain propagation).

- **Failure model for multicast datagrams**
  - Datagrams multicast over IP multicast have the same failure characteristics as UDP datagrams
  - – that is, they suffer from omission failures.
  - The effect on a multicast is that messages are not guaranteed to be delivered to any particular group member in the face of even a single omission failure.
  - That is, some but not all of the members of the group may receive it.
  - This can be called *unreliable* multicast, because it does not guarantee that a message will be delivered to any member of a group.

IP multicast – A Multicast Communication

Java API to IP multicast

Figure 4.14 Multicast peer joins a group and sends and receives datagrams

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
    public static void main(String args[]){
    // args give message contents & destination multicast group (e.g. "228.5.6.7")
    MulticastSocket s =null;
    try {
      InetAddress group = InetAddress.getByName(args[1]);
      s = new MulticastSocket(6789);
      s.joinGroup(group);
      byte [] m = args[0].getBytes();
      DatagramPacket messageOut =
          new DatagramPacket(m, m.length, group, 6789);
      s.send(messageOut);
      byte[] buffer = new byte[1000];
      for(int i=0; i< 3; i++) { // get messages from others in group
          DatagramPacket messageIn =
              new DatagramPacket(buffer, buffer.length);
          s.receive(messageIn);
          System.out.println("Received:" + new String(messageIn.getData()));
```

IP multicast – A Multicast Communication

```
        }
        s.leaveGroup(group);
    } catch (SocketException e){System.out.println("Socket: " + e.getMessage());
    } catch (IOException e){System.out.println("IO: " + e.getMessage());
    } finally { if(s != null) s.close();}
  }
}
```

- The Java API provides a datagram interface to IP multicast through the class MulticastSocket, which is a subclass of DatagramSocket with the additional capability of being able to join multicast groups.
- A process can join a multicast group with a given multicast address by invoking the joinGroup method of its multicast socket.
- A process can leave a specified group by invoking the *leaveGroup* method of its multicast socket.
- The Java API allows the TTL to be set for a multicast socket by means of the *setTimeToLive* method.

Group Communication

- *Group communication* offers a service whereby a message is sent to a group and then this message is delivered to all members of the group.
- In this action, the sender is not aware of the identities of the receivers.
- Group communication represents an abstraction over multicast communication and may be implemented over IP multicast or an equivalent overlay network, adding significant extra value in terms of:
  - Managing group membership,
  - detecting failures
  - and providing reliability and ordering guarantees.
- Group communication is an important building block for distributed systems with key areas of application including:
  - The reliable dissemination of information to potentially large numbers of clients. Eg:- Financial Industry where institutions require accurate and up-to- date access to a wide variety of information sources
  - Support for collaborative applications – Eg:- Github

Group Communication

- Support for a range of fault-tolerance strategies, including the consistent update of replicated data or the implementation of highly available (replicated) servers.
- Support for system monitoring and management, including for example load balancing strategies.

- The programming model
  - In group communication, the central concept is that of a *group* with associated *group membership*, whereby processes may *join* or *leave* the group.
  - Processes can then send a message to this group and have it propagated to all members of the group with certain guarantees in terms of reliability and ordering.
  - Thus, group communication implements *multicast* communication, in which a message is sent to all the members of the group by a single operation.
  - Communication to *all* processes in the system, as opposed to a subgroup of them, is known as *broadcast*, whereas communication to a single process is known as *unicast*.
  - The essential feature of group communication is that a process issues only one multicast operation to send a message to each of a group of processes (in Java this operation is *aGroup.send(aMessage)*) instead of issuing multiple send operations to individual processes.

Group Communication

- The use of a single multicast operation instead of multiple send operations amounts to much more than a convenience for the programmer:
  - it enables the implementation to be efficient in its utilization of bandwidth.
  - The implementation can also minimize the total time taken to deliver the message to all destinations, as compared with transmitting it separately and serially.
  - The use of a single multicast operation is also important in terms of delivery guarantees.
  - If a process issues multiple independent send operations to individual processes, then there is no way for the implementation to provide guarantees that affect the group of processes as a whole.

- Group communication has been the subject of many research projects, including the V-system ,Chorus, Amoeba, Isis .

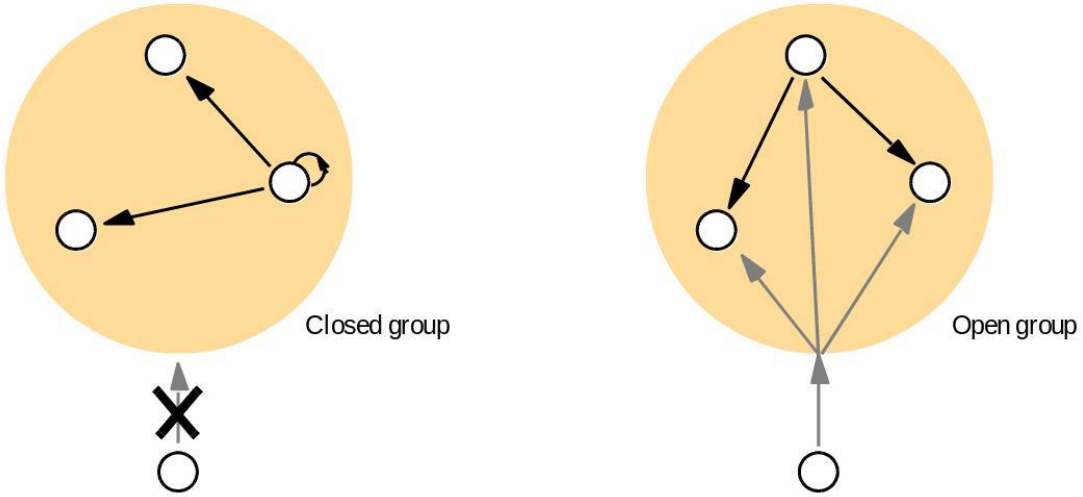
- **Process groups and object groups**
  - Most work on group services focuses on the concept of *process groups*, that is, groups where the communicating entities are processes.
  - Such services are relatively low-level in that:
    - Messages are delivered to processes and no further support for dispatching is provided.

- Messages are typically unstructured byte arrays with no support for marshalling of complex data types (as provided, for example, in RPC or RMI)

- The level of service provided by process groups is similar to that of sockets.
object groups
- *object groups* provide a higher-level approach to group computing.
- An object group is a collection of objects (normally instances of the same class) that process the same set of invocations concurrently, with each returning responses.
- Client objects need not be aware of the replication.
- They invoke operations on a single, local object, which acts as a proxy for the group.
- The proxy uses a group communication system to send the invocations to the members of the object group.
- Object parameters and results are marshalled as in RMI and the associated calls are dispatched automatically to the right destination objects/methods.
- Electra is a CORBA-compliant system that supports object groups.

Group Communication

- **Closed groups and Open groups**



- **Figure** shows Open and closed groups
  - Closed group is said to be *closed* if only members of the group may multicast to it.
  - A process in a closed group delivers to itself any message that it multicasts to the group.
  - A group is *open* if processes outside the group may send to it.
  - Closed groups of processes are useful, for example, for cooperating servers to send messages to one another that only they should receive.
  - Open groups are useful, for example, for delivering events to groups of interested processes.

Group Communication

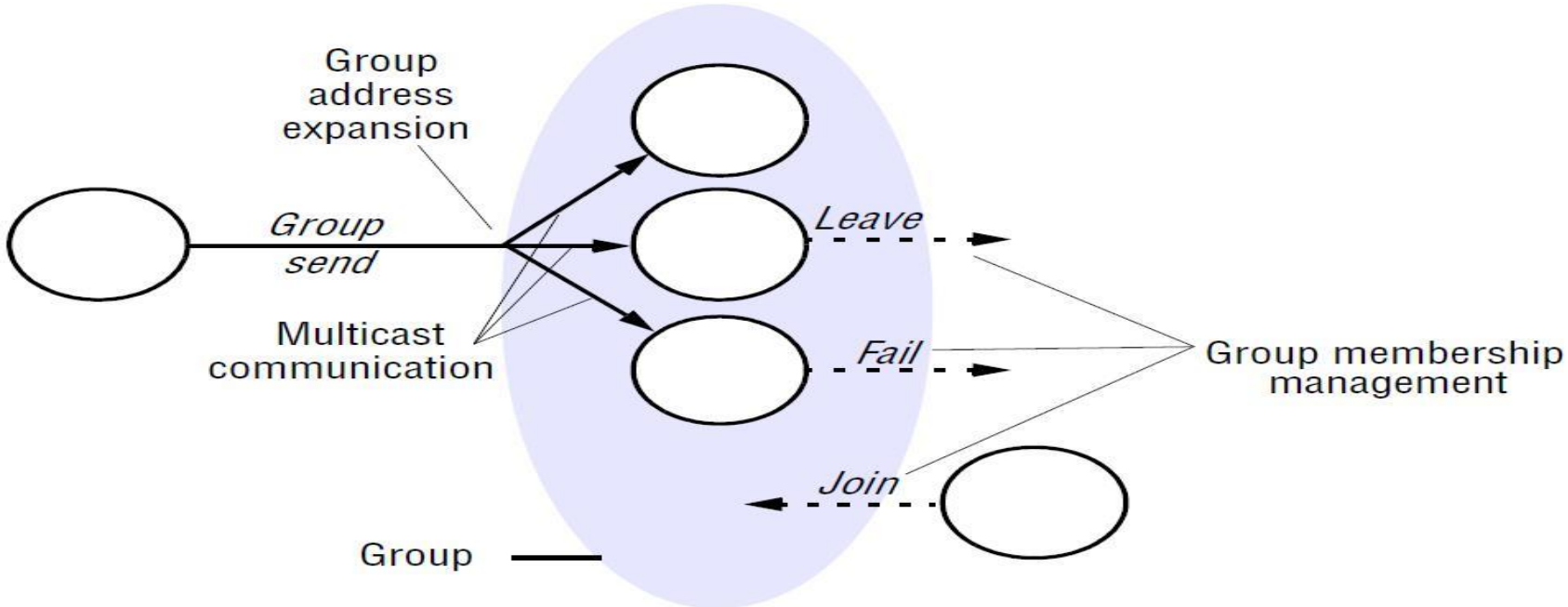
- *Overlapping and non-overlapping groups*:
  - In *overlapping groups*, entities (processes or objects) may be members of multiple groups, and non-overlapping groups imply that membership does not overlap (that is, any process belongs to at most one group).
  - Note that in real-life systems, it is realistic to expect that group membership will overlap.
- *Synchronous and asynchronous systems*:
  - There is a requirement to consider group communication in both environments.
- **Implementation Issues:**
  - Reliability and ordering in multicast

    Reliability in one-to-one communication was defined in terms of two properties:
    - Integrity (the message received is the same as the one sent, and no messages are delivered twice)
    - and Validity (any outgoing message is eventually delivered).
  - As well as reliability guarantees, group communication demands extra guarantees in terms of the relative ordering of messages delivered to multiple destinations.

- Ordering is not guaranteed by underlying interprocess communication primitives.

Group Communication

- To counter ordering issue**,** group communication services offer *ordered multicast*, with the option of one or more of the following properties (with hybrid solutions also possible):
  - FIFO ordering: First-in-first-out (FIFO) (or source ordering) – if sender sends one before the other, it will be delivered in this order at all group processes
  - Casual ordering: – if a message happens before another message in the distributed system, this so-called casual relationship will be preserved in the delivery of the associated messages at all processes.
  - Total ordering: – if a message is delivered before another message at one process, the same order will be preserved at all processes
- **Group membership management**
- This diagram illustrates the important role of group membership management in maintaining an accurate view of the current membership, given that entities may join, leave or indeed fail.



Role of group membership management

Group Communication

- In more detail, a group membership service has four main tasks:
  - Providing an interface for group membership changes:
    - The membership service provides operations to create and destroy process groups and to add or withdraw a process to or from a group.
    - In most systems, a single process may belong to several groups at the same time (overlapping groups, as defined above). This is true of IP multicast, for example.
  - Failure detection.
    - The service monitors the group members not only in case they should crash, but also in case they should become unreachable because of a communication failure.
    - The detector marks processes as Suspected or Unsuspected.
    - The service uses the failure detector to reach a decision about the group's membership: it excludes a process from membership if it is suspected to have failed or to have become unreachable.
  - Notifying members of group membership changes:
    - The service notifies the group's members when a process is added, or when a process is excluded (through failure or when the process is deliberately withdrawn from the group).
  - Performing group address expansion:
    - When a process multicasts a message, it supplies the group identifier rather than a list of processes in the group.

Network Virtualization: Overlay Networks

- Network virtualization is concerned with the construction of many different virtual networks over an existing network such as the Internet.
- Each virtual network can be designed to support a particular distributed application.
- For example, one virtual network might support multimedia streaming, as in Netflix, Hotstar or Hulu [hulu.com], and coexist with another that supports a multiplayer online game, both running over the same underlying network.
- **Overlay networks**
  - An *overlay network* is a virtual network consisting of nodes and virtual links, which sits on top of an underlying network (such as an IP network) and offers something that is not otherwise provided:
  - a service that is tailored towards the needs of a class of application or a particular higher-level service – for example, multimedia content distribution;
  - more efficient operation in a given networked environment – for example routing in an ad hoc network;
  - an additional feature – for example, multicast or secure communication.

Network Virtualization: Overlay Networks

- Overlay networks have the following advantages:
  - They enable new network services to be defined without requiring changes to the underlying network, a crucial point given the level of standardization in this area and the difficulties of amending underlying router functionality.
  - They encourage experimentation with network services and the customization of services to particular classes of application.
  - Multiple overlays can be defined and can coexist, with the end result being a more open and extensible network architecture.
- The disadvantages are that:
  - Overlays introduce an extra level of indirection (and hence may incur a performance penalty)
  - And they add to the complexity of network services when compared, for example, to the relatively simple architecture of TCP/IP networks.

Network Virtualization: Overlay Networks

Figure 4.15 Types of overlay

Motivation	*Type*	*Description*
Tailored for application needs	Distributed hash tables	One of the most prominent classes of overlay network, offering a service that manages a mapping from keys to values across a potentially large number of nodes in a completely decentralized manner (similar to a standard hash table but in a networked environment).
	Peer-to-peer file sharing	Overlay structures that focus on constructing tailored addressing and routing mechanisms to support the cooperative discovery and use (for example, download) of files.
	Content distribution networks	Overlays that subsume a range of replication, caching and placement strategies to provide improved performance in terms of content delivery to web users; used for web acceleration and to offer the required real-time performance for video streaming [www.kontiki.com].
Tailored for network style	Wireless ad hoc networks	Network overlays that provide customized routing protocols for wireless ad hoc networks, including proactive schemes that effectively construct a routing topology on top of the underlying nodes and reactive schemes that establish routes on demand typically supported by flooding.
	Disruption-tolerant networks	Overlays designed to operate in hostile environments that suffer significant node or link failure and potentially high delays.
Offering additional features	Multicast	One of the earliest uses of overlay networks in the Internet, providing access to multicast services where multicast routers are not available; builds on the work by Van Jacobsen, Deering and Casner with their implementation of the MBone (or Multicast Backbone) [mbone].
	Resilience	Overlay networks that seek an order of magnitude improvement in robustness and availability of Internet paths [nms.csail.mit.edu].
	Security	Overlay networks that offer enhanced security over the underling IP network, including virtual private networks, for example, as discussed in Section 3.4.8.

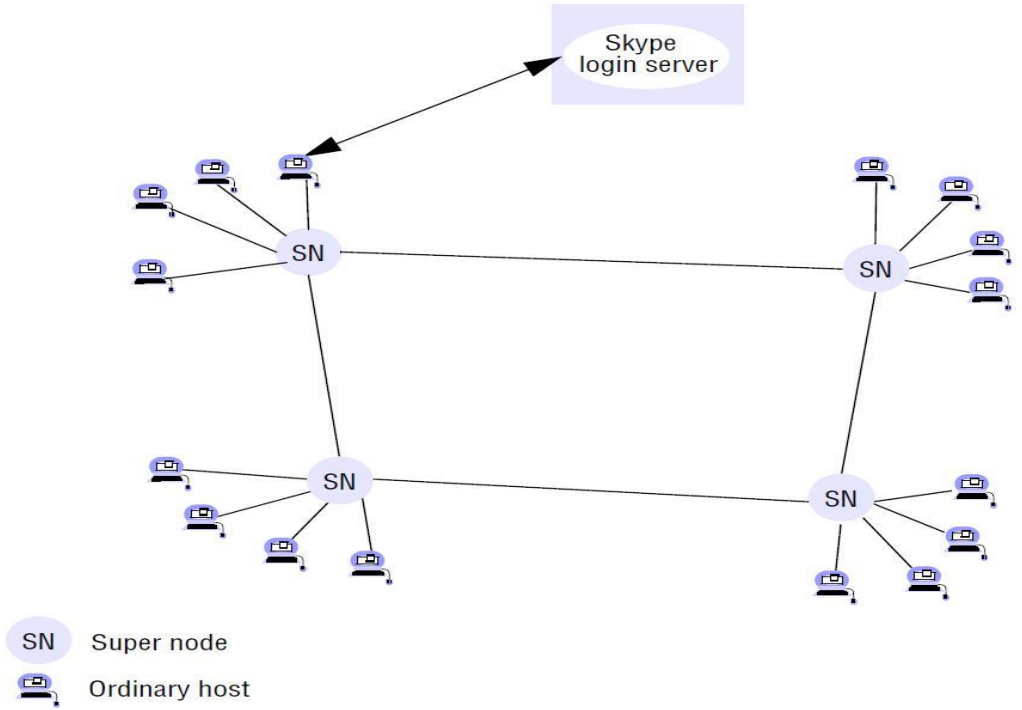
Network Virtualization: Overlay Networks

- Skype: An example of an overlay network
  - Skype is a peer-to-peer application offering Voice over IP (VoIP).
  - It also includes instant messaging, video conferencing and interfaces to the standard telephony service through SkypeIn and SkypeOut.
- Skype is an excellent case study of the use of overlay networks in real-world (and large-scale) systems, indicating how advanced functionality can be provided in an application-specific manner and without modification of the core architecture of the Internet.
  - Skype is a virtual network in that it establishes connections between people (Skype subscribers who are currently active).
  - No IP address or port is required to establish a call.
  - The architecture of the virtual network supporting Skype is not widely publicized but researchers have studied Skype through a variety of methods, including traffic analysis, and its principles are now in the public domain.

Network Virtualization: Overlay Networks

- Skype architecture
  - Skype is based on a peer-to-peer infrastructure consisting of ordinary users' machines (referred to as hosts) and super nodes
  - super nodes are ordinary Skype hosts that happen to have sufficient capabilities to carry out their enhanced role.
  - Super nodes are selected on demand based a range of criteria including bandwidth available, reachability (the machine must have a global IP address and not be hidden behind a NAT-enabled router, for example) and availability (based on the length of time that Skype has been running continuously on that node).

Figure 4.16 Skype overlay architecture



Network Virtualization: Overlay Networks

- **User connection**
  - Skype users are authenticated via a well-known login server.
  - They then make contact with a selected super node.
  - To achieve this, each client maintains a cache of super node identities (that is, IP address and port number pairs).
  - At first login this cache is filled with the addresses of around seven super nodes, and over time the client builds and maintains a much larger set (perhaps several hundred).
- **Search for users**
  - The main goal of super nodes is to perform the efficient search of the global index of users, which is distributed across the super nodes.
  - The search is orchestrated by the client's chosen super node and involves an expanding search of other super nodes until the specified user is found.
  - On average, eight super nodes are contacted.
  - A user search typically takes between three and four seconds to complete for hosts that have a global IP address (and slightly longer, 5 to 6 secs, if behind a NAT-enabled router).
- From experiments, it appears that intermediary nodes involved in the search cache the results to improve performance.

Network Virtualization: Overlay Networks

- **Voice connection**
  - Once the required user is discovered, Skype establishes a voice connection between the two parties using TCP for signalling call requests and terminations and either UDP or TCP for the streaming audio.
  - UDP is preferred but TCP, along with the use of an intermediary node, is used in certain circumstances to circumvent firewalls.
  - The software used for encoding and decoding audio plays a key part in providing the excellent call quality normally attained using Skype, and the associated algorithms are carefully tailored to operate in Internet environments at 32 kbps and above.

Remote Procedure Call (RPC)

- Write Remote procedure call as assignment?
- Deadline: 09/10/2018

REMOTE PROCEDURE CALL

- Procedures in processes on remote computers can be called as if they are procedures in the local address space.
- The underlying RPC system then **hides important aspects of distribution, including the encoding and decoding of parameters and results, the passing of messages and preserve the required semantics for the procedure call.**
- Supports client-server computing with **servers offering a set of operations through a service interface and clients calling these operations directly as if they were available locally.**
- RPC offers (at minimum) access and location transparency.

Design issues for RPC

- **The style of programming promoted by RPC programming with interfaces;**
- **The call semantics associated with RPC;**
- **The key issue of transparency and how it relates to remote procedure calls.**

- **Programming with interfaces**
- Most modern programming languages provide a means of organizing a program as a set of modules that can communicate with one another.

- Communication between modules can be by **means of procedure calls** between modules or by **direct access to the variables in another module.**
- In order to control the possible interactions between modules, an **explicit interface** is defined for each module.
- The interface of a module **specifies the procedures and the variables that can be accessed from other modules.**
- **Interfaces in distributed systems::**
- In a distributed program, **the modules can run in separate processes.**
- In the client-server model e**ach server provides a set of procedures that are available for use by clients.**

- For example, **a file server would provide procedures for reading and writing files**
- The term **service interface** is used to refer to the **specification of the procedures offered by a server, defining the types of the arguments of each of the procedures.**
- **There are a number of benefits to programming with interfaces in distributed systems**
- As with any form of modular programming, programmers are concerned only with the **abstraction** offered by the service interface and **need not be aware of implementation details.**
- in heterogeneous DS , programmers also do not need to know the programming language or underlying platform used to implement the service (an important step towards managing heterogeneity in distributed systems)

- This approach provides natural support for software evolution in that implementations can change as long as long as the interface (the external view) remains the same.

- **The definition of service interfaces is influenced by the distributed nature of the underlying infrastructure**
- It is not possible for a client module running in one process to access the variables in a module in another process.Therefore the service interface cannot specify direct access to variables
- Note that CORBA IDL interfaces can specify attributes, which seems to break this rule.

- The parameter-passing mechanisms used in local procedure calls– for example, **call by value and call by reference, are not suitable when the caller and procedure are in different processes.**
- Another difference between local and remote modules is that **addresses in one process are not valid in another remote one.** Therefore, addresses cannot be passed as arguments or returned as results of calls to remote modules.
- **Interface definition languages**
- _An RPC mechanism can be integrated with a particular programming language if it includes an adequate notation for defining interfaces, allowing input and output parameters to be mapped onto the language's normal use of parameters.

- This approach is useful when all the **parts of a distributed application can be written in the same language**. It is also convenient because it allows the programmer to use a single language, for example, Java, for local and remote invocation.
- Interface definition languages (IDLs) are designed to allow procedures implemented in different languages to invoke one another.
- An IDL provides a notation for defining interfaces

RPC Call semantics

3.choices

- **Retry request message:** Controls whether to retransmit the request message until either a reply is received or the server is assumed to have failed.
- **Duplicate filtering**:**Controls when retransmissions are** used and whether to filter out **duplicate requests at the server.**
- **Retransmission of results:** Controls whether to keep a history of result messages to enable **lost results to be retransmitted without re-executing the operations at the server.**

- Combinations of these choices lead to a variety of possible semantics for the reliability of remote invocations

Fault tolerance measures			*Call semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

Maybe semantics:

- With maybe semantics, the **remote procedure call may be executed once or not at all.**
- Maybe semantics **arises when no fault-tolerance measures** are applied and can suffer from the following types of failure:
- **omission failures** if the request or result message is lost;
- **crash failures** when the **server containing the remote operation fails.**
- If the result message has not been received after a timeout and there are no retries, it is uncertain whether the procedure has been executed. If the request message was lost, then the procedure will not have been executed. On the other hand, the procedure may have been executed and the result message lost.

At-least-once semantics

- With at-least-once semantics, the invoker receives either a result, in which case the invoker knows that the **procedure was executed at least once,**
- At-least-once semantics can be achieved by the retransmission of request messages, which masks the omission failures of the request or result message.
- At-least-once semantics can suffer from the following types of failure:

• crash failures when the server containing the remote procedure fails;

• arbitrary failures – in cases when the request message is retransmitted, the remote server may receive it and execute the procedure more than once, possibly causing wrong values to be stored or returned.

At-most-once semantics

- With at-most-once semantics, the caller receives either a result, in which case the caller knows that the **procedure was executed exactly once** or an **exception informing it that no result was received,** in which case the **procedure will have been executed either once or not at all.**
- At-most-once semantics can be achieved by using all of the fault-tolerance measures

Transparency •

- The originators of RPC, Birrell and Nelson [1984], aimed to make **remote procedure calls as much like local procedure calls** as possible, with no distinction in syntax between a local and a remote procedure call.
- **message-passing procedures were hidden from the programmer** making the call. Although request messages are retransmitted after a timeout.
- However, remote procedure calls are more vulnerable to failure than local ones, since they involve a network, another computer and another process. .

- Whichever of the above semantics is chosen, there is always the chance that no result will be received, and in the case of failure, it is impossible to distinguish between failure of the network and of the remote server process.
- Waldo et al. [1994] say that the difference between local and remote operations should be expressed at the service interface
- RPC should be transparent
- For example, in some IDLs, a remote invocation may throw an exception when the client is unable to communicate with a remote procedure. This requires that the client program handle such exceptions,

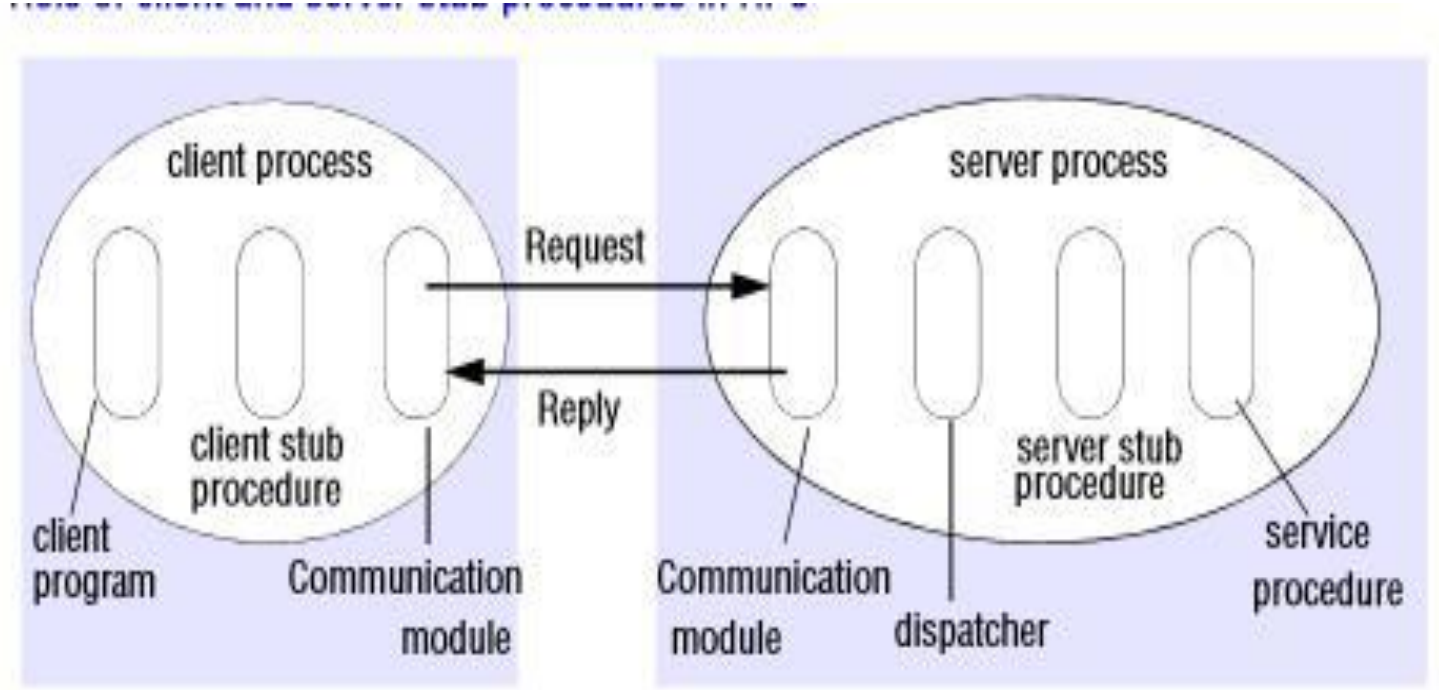
- An IDL can also provide a facility for specifying the call semantics of a procedure.
- This can help the designer of the service –
- for example, if at-least-once call semantics is chosen to avoid the overheads of at-most-once,

IMPLEMENTATION OF RPC

- The client that accesses a service includes **one stub procedure for each procedure in the service interface**
- The stub procedure behaves like a local procedure to the client, but instead of executing the call, it marshals the procedure identifier and the arguments into a request message, which it sends via its communication module to the server. When the reply message arrives, it unmarshals the results.
- The server process contains a **dispatcher together with one server stub procedure and one service procedure for each procedure in the service interface.** The dispatcher selects one of the server stub procedures according to the procedure identifier in the request message..

- The server stub procedure then unmarshals the arguments in the request message, calls the corresponding service procedure and marshals the return values for the reply message. The service procedures implement the procedures in the service interface. The client and server stub procedures and the dispatcher can be generated automatically by an interface compiler.
- RPC is generally implemented over a request-reply protocol
- RPC may be implemented to have one of the choices of invocation semantics – at-leastonce or at-most-once is generally chosen. To achieve this, the communication module will implement the desired design choices in terms of retransmission of requests, dealing with duplicates and retransmission of results, .

Role of client and server stub procedures in RPC



Case Study Sun RPC

- Sun RPC, which was designed for client-server communication in the Sun Network File System (NFS). Sun RPC is sometimes called ONC (Open Network Computing) RPC. It is supplied as a part of the various Sun and other UNIX operating systems and is also available with NFS installations. Implementors have the choice of using remote procedure calls over either UDP or TCP. **When Sun RPC is used with UDP, request and reply messages are restricted in length – theoretically to 64 kilobytes, but more often in practice to 8 or 9 kilobytes. It uses atleast-once call semantics. Broadcast RPC is an option.**

Interface definition language

- The Sun XDR language, which was originally designed for specifying external data representations, was extended to become an interface definition language. It may be used to define a service interface for Sun RPC by **specifying a set of procedure definitions together with supporting type definitions**
- Most languages allow interface names to be specified, but Sun RPC does not – instead of this, a program number and a version number are supplied. The program numbers can be obtained from a central authority to allow every program to have its own unique number.

- A procedure definition specifies a procedure signature and a procedure number. The procedure number is used as a procedure identifier in request messages.
- Only a single input parameter is allowed. Therefore, procedures requiring multiple parameters must include them as components of a single structure.
- The output parameters of a procedure are returned via a single result.
- The procedure signature consists of the result type, the name of the procedure and the type of the input parameter. The type of both the result and the input parameter may specify either a single value or a structure containing several values.

- The interface compiler rpcgen can be used to generate the following from an interface definition: • **client stub procedures**; • **server main procedure, dispatcher and server stub procedures**; • **XDR marshalling and unmarshalling procedures for use by the dispatcher and client and server stub procedures.**
- **Binding** • Sun RPC runs a local binding service called the port mapper at a well-known port number on each computer. Each instance of a port mapper records the program number, version number and port number in use by each service running locally. When a server starts up it registers its program number, version number and port number with the local port mapper. When a client starts up, it finds out the server's port by making a remote request to the port mapper at the server's host, specifying the program number and version number.

- **Authentication.** Sun RPC request and reply messages provide additional fields enabling authentication information to be passed between client and server. The request message contains the credentials of the user running the client program
- Several different authentication protocols can be supported. These include:
- none;
- UNIX style,
- a style in which a shared key is established for signing the RPC messages;
- Kerberos
- Client and server programs