Jisy Raju
Assistant Professor, CE Cherthala

Module 3

Interprocess communication: characteristics – group communication - Multicast Communication –Remote Procedure call - Network virtualization. Case study :Skype

3.1 INTERPROCESS COMMUNICATION

Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system. It take place by message passing.

Application program interface (**API**) is a set of routines, protocols, and tools for building software applications. It includes,

1. The characteristics of Inter-process Communication.
2. Sockets
3. UDP Datagram Communication
4. TCP Stream Communication

3.1.1 The characteristics of Inter-process Communication.

Message passing between a pair of processes can be supported by two message communication operations, send and receive, defined in terms of destinations and messages. To communicate, one process sends a message (a sequence of bytes) to a destination and another process at the destination receives the message. This activity involves the communication of data from the sending process to the receiving process and may involve the synchronization of the two processes. Including,

1. Synchronous and Asynchronous Communication
2. Message Destinations
3. Reliability
4. Ordering

- **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. Communication between the sending and receiving processes may be either synchronous or asynchronous.

-In synchronous form of communication, the sending and receiving process synchronize at every message. In the synchronous form, both send and receive are blocking operations.

- In asynchronous form of communication, the sending operation is non-blocking and the receive operation can have blocking and non-blocking variants. 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. 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.

- **Message Destinations**

In the Internet protocols, messages are sent to local port. 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.

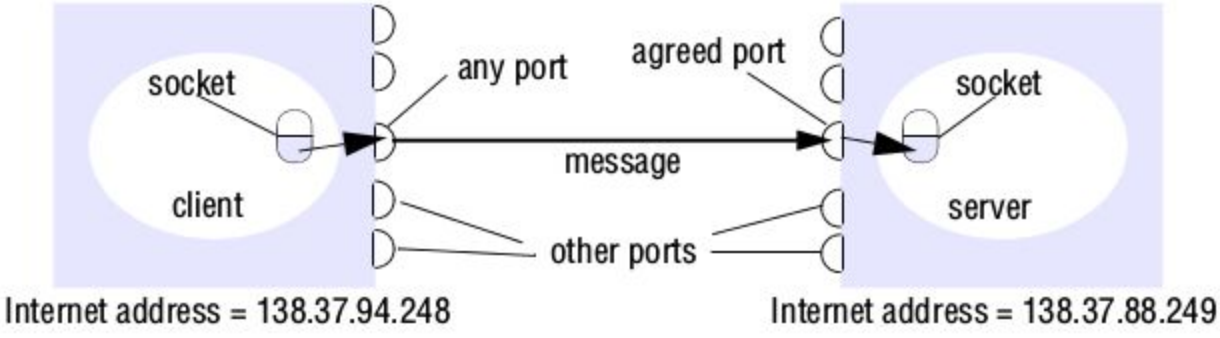
- **Reliability**: Reliable communication in defined terms of validity and integrity.A point to point message service is described as reliable- messages are guaranteed to be delivered despite a reasonable number of packets being dropped or lost. For integrity, messages must arrive uncorrupted and without duplication.

- **Ordering**: Some applications require that messages 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.

3.1.2. Sockets

Processes can send and receive messages via a socket. Interprocess communication consists of

transmitting a message between a socket in one process and a socket in another process. Sockets need to be bound to a port number and an Internet address in order to send and receive messages. Each socket has a transport protocol (TCP or UDP). Both form of communication, UDP and TCP, use the socket, which provides endpoint for communication between processes.



Java API for Internet addresses : As the IP packets underlying UDP and TCP are sent to Internet addresses, Java provides a class, **InetAddress**, that represents Internet addresses. Users of this class refer to computers by Domain Name System (DNS) host names.

```
InetAddress aComputer = InetAddress.getByName("bruno.dcs.qmul.ac.uk");
```

3.1.3. UDP Datagram Communication

The User Datagram Protocol (UDP) is a transport layer protocol. The service provided by UDP is an unreliable service that provides no guarantees for delivery and no protection from duplication. Some issues relating to datagram communication Including

- Message Size
- Blocking
- Timeouts
- Receive from any

Message Size: Receiving process specify an array of bytes to receive message. If size of the message is bigger than the array, then message is truncated.

Blocking: Sockets normally provide non-blocking sends and blocking receives for datagram communication . On arrival, the message is placed in a queue for the socket that is bound to the destination port. The message can be collected from the queue Messages are discarded at the destination if no process already has a socket bound to the destination port.

Timeouts: A process that has invoked a receive operation should wait indefinitely in situations where the sending process may have crashed or the expected message may have been lost. To allow for such requirements, timeouts can be set on sockets. Choosing an appropriate timeout interval is difficult, but it should be fairly large in comparison with the time required to transmit a message.

Receive from any: The receive method does not specify an origin for messages. Instead, an invocation of receive gets a message addressed to its socket from any origin.

Failure model for UDP datagrams : A failure model for

communication channels and defines reliable communication in terms of two properties: integrity and validity. The integrity property requires that messages should not be corrupted or duplicated. The use of a checksum ensures that there is a negligible probability that any message received is corrupted. UDP datagrams suffer from the following failures:

Omission failures: Messages may be dropped occasionally, either because of a checksum error or because no buffer space is available at the source or destination.

Use of UDP: the Domain Name System(DNS), Simple Network Management Protocol (**SNMP**)**,** Voice over IP **(VOIP)**

Java API for UDP datagrams: The Java API provides datagram communication by means of two

classes: **DatagramPacket** and **DatagramSocket.**

DatagramPacket: This class provides a constructor that makes an instance out of an array

DatagramSocket: This class supports sockets for sending and receiving UDP datagrams

Datagram packet

array of bytes containing message	length of message	Internet address	port number

3.1.4. TCP Stream communication

TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent. TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages**.**

The following characteristics of the network are hidden by the stream abstraction:

- **Message sizes:** The application can choose how much data it writes to a stream or reads from it. It may deal in very small or very large sets of data. The underlying implementation of a TCP stream decides how much data to collect before transmitting it as one or more IP packets.
- **Lost messages:** The TCP protocol uses an acknowledgement scheme. If the sender does not receive an acknowledgement within a timeout, it retransmits the message.
- **Flow control:** The TCP protocol attempts to match the speeds of the processes that read from and write to a stream. If the writer is too fast for the reader, then it is blocked until the reader has consumed sufficient data.
- **Message duplication and ordering:** Message identifiers are associated with each IP packet, which enables the recipient to detect and reject duplicates, or to reorder messages that do not arrive in sender order.
- **Message destinations:** A pair of communicating processes establishes a connection before they can communicate over a stream. Establishing a connection involves a connect request from client to server followed by an accept request from server to client before any communication can take place.

The API for stream communication assumes that when a pair of processes are establishing a connection, one of them plays the client role and the other plays the server role, but thereafter they could be peers. The client role involves creating a stream socket bound to any port and then making a connect request asking for a connection 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. The listening socket maintains a queue of incoming connection requests. In the socket model, when the server accepts a connection, a new stream socket is created for the server to communicate with a client. The pair of sockets in the client and server are connected by a pair of streams, one in each direction. Thus each socket has an input stream and an output stream.

When an application closes a socket, this indicates that it will not write any more data to its output stream. Any data in the output buffer is sent to the other end of the stream and put in the queue at the destination socket, with an indication that the stream is broken. When a process exits or fails, all of its sockets are eventually closed and any process attempting to communicate with it will discover that its connection has been broken.

Failure model • To satisfy the integrity property of reliable communication, TCP streams use checksums to detect and reject corrupt packets and sequence numbers to detect and reject duplicate packets. For the sake of the validity property, TCP streams use timeouts and retransmissions to deal with lost packets. Therefore, messages are guaranteed to be

delivered even when some of the underlying packets are lost. The TCP software responsible for sending messages will receive no acknowledgements and after a time will declare the connection to be broken.

Use of TCP:

- **HTTP**:-The Hypertext Transfer Protocol is used for communication between web browsers and web servers.
- **FTP:** The File Transfer Protocol allows directories on a remote computer to be browsed and files to be transferred from one computer to another over a connection.
- **Telnet:** Telnet provides access by means of a terminal session to a remote computer.
- **SMTP:** The Simple Mail Transfer Protocol is used to send mail between computers.

Java API for TCP streams: The Java interface to TCP streams is provided in the classes **ServerSocket** and **Socket**:

ServerSocket: This class is intended for use by a server to create a socket at a server port for

listening for connect requests from clients.

Socket: This class is for use by a pair of processes with a connection.

3.2 Group communication

Group communication provides an example of an indirect communication paradigm.

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. Group communication represents an abstraction over multicast

communication With the added guarantees, group communication is to IP multicast what TCP is to

the point-to-point service in IP.

Group communication is an important building block for distributed systems, and particularly reliable distributed systems, with key areas of application including:

- The reliable dissemination of information to potentially large numbers of clients, including in the financial industry, where institutions require accurate and up-to-date access to a wide variety of information sources.
- Support for collaborative applications, where again events must be disseminated to multiple users to preserve a common user view – for example, in multiuser games.
- 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.

3.2.1 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.***

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.

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.

Types of groups

Closed and open groups*:* A group is said to be *closed* if only members of the group may multicast to it (Figure 3.2). 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.

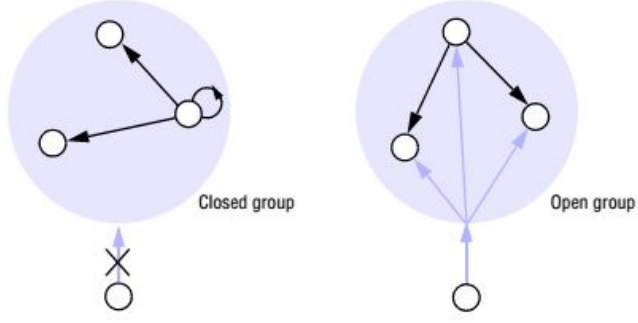


Figure 3.2 Open and closed groups

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.

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.

Synchronous and asynchronous systems: There is a requirement to consider group communication in both environments.

3.2.2 Implementation issues

Implementation issues for group communication services, discussing the properties of the underlying multicast service in terms of reliability and ordering and also the key role of group membership management in dynamic environments.

1. Reliability and ordering in multicast • In group communication, all members of a group must receive copies of the messages sent to the group, generally with delivery guarantees. The guarantees include agreement on the set of messages that every process in the group should receive and on the delivery ordering across the group members.

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).

The interpretation for reliable multicast builds on these properties, with integrity defined the same way in terms of delivering the message correctly at most once, and validity guaranteeing that a message sent will eventually be delivered.

To extend the semantics to cover delivery to multiple receivers, a third property is added – that of agreement, stating that if the message is delivered to one process, then it is delivered to all processes in the group.

FIFO ordering: First-in-first-out (FIFO) ordering (also referred to as source ordering) is concerned with preserving the order from the perspective of a sender process, in that if a process sends one message before another, it will be delivered in this order at all processes in the group.

Causal ordering: Causal ordering takes into account causal relationships between messages, in that if a message happens before another message in the distributed system this so-called causal relationship will be preserved in the delivery of the associated messages at all processes.

Total ordering**:** In total ordering, if a message is delivered before another message at one process, then the same order will be preserved at all processes.

2. Group membership management • The key elements of group communication management are summarized in Figure 3.3, which shows an open group.

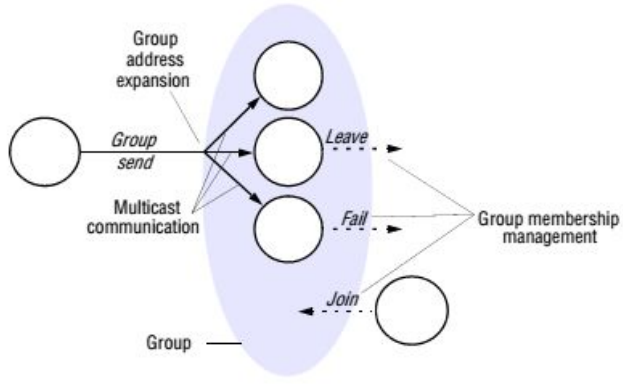


Figure 3.3 The role of group membership management

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. 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. The membership management service expands the identifier into the current group membership for delivery.

3.2.3 Case study: the JGroups toolkit

JGroups is a toolkit for reliable group communication written in Java. JGroups supports process groups in which processes are able to join or leave a group, send a message to all members of the group or indeed to a single member, and receive messages from the group.

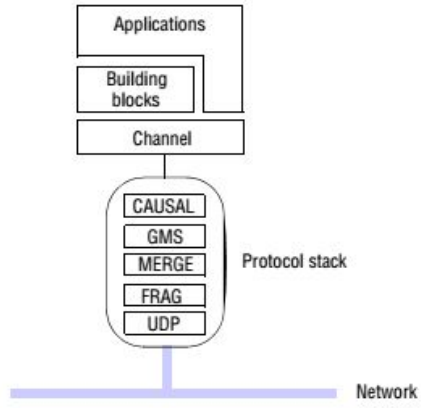


Figure 3.4 The architecture of JGroups

The architecture of JGroups is shown in Figure 3.4, which shows the main components of the JGroups implementation:

• Channels represent the most primitive interface for application developers, offering the core functions of joining, leaving, sending and receiving.

• Building blocks offer higher-level abstractions, building on the underlying service offered by channels.

Examples of building blocks in JGroups are:

• *MessageDispatcher* is the most intuitive of the building blocks offered in JGroups. In group communication, it is often useful for a sender to send a message to a group and then wait for some or all of the replies. *MessageDispatcher* supports this by providing a *castMessage* method that sends a message to a group and blocks until a specified number of replies are received (for example, until a specified number n, a majority, or all messages are received).

• *RpcDispatcher* takes a specific method (together with optional parameters and results) and then invokes this method on all objects associated with a group. As with *MessageDispatcher*, the caller can block awaiting some or all of the replies.

• *NotificationBus* is an implementation of a distributed event bus, in which an event is any serializable Java object. This class is often used to implement consistency in replicated caches.

The protocol stack • JGroups follows the architectures offered by Horus and Ensemble by constructing protocol stacks out of protocol layers.

In this approach, a protocol is a bidirectional stack of protocol layers with each layer implementing the following two methods:

public Object up (Event evt);

public Object down (Event evt);

Protocol processing therefore happens by passing events up and down the stack. In

JGroups, events may be incoming or outgoing messages or management events, for example related to view changes. Each layer can carry out arbitrary processing on the message, including modifying its contents, adding a header or indeed dropping or reordering the message.

Protocol that consists of five layers:

• The layer referred to as UDP is the most common transport layer in JGroups. Note that, despite the name, this is not entirely equivalent to the UDP protocol; rather, the layer utilizes IP multicast for sending to all members in a group and UDP datagrams specifically for point-to-point communication. This layer therefore assumes that IP multicast is available.

If it is not, the layer can be configured to send a series of unicast messages to members, relying on another layer for membership discovery (in particular, a layer known as PING). For larger-scale systems operating over wide area networks, a TCP layer may be preferred (using the TCP protocol to send unicast messages and again relying on PING for membership discovery).

• FRAG implements message packetization and is configurable in terms of the maximum message size (8,192 bytes by default).

• MERGE is a protocol that deals with unexpected network partitioning and the subsequent merging of subgroups after the partition. A series of alternative merge layers are actually available, ranging from the simple to ones that deal with, for example, state transfer.

• GMS implements a group membership protocol to maintain consistent views of membership across the group

• CAUSAL implements causal ordering.

3.3 Multicast communication

Multicast operation is an operation that sends a single message from one process to each of

the members of a group of processes. The simplest way of multicasting , provides no guarantees about message delivery or ordering. Multicasting has the following characteristics,

1. Fault tolerance based on replicated services
2. Finding the discovery servers in spontaneous networking.
3. Better performance through replicated data.
4. Propagation of event notifications

3.3.1 IP Multicast – an implementation of group communication.

IP multicast is built on top of the Internet protocol, IP. IP multicast allows the sender to transmit a single IP packet to a multicast group. A multicast group is specified by class D IP address for which first 4 bits are 1110 in Ipv4. The membership of a multicast group 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. An
application program performs multicasts by sending UDP datagrams with multicast addresses and ordinary port numbers. The following details are specific to IPv4:

- **Multicast routers:** IP packets can be multicast both on a local network and on the

wider Internet.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). This document defines a partitioning of this address space into a number of blocks, including:

Multicast addresses may be permanent or temporary. Permanent groups exist even
when there are no members – their addresses are assigned by IANA and span the various blocks mentioned above. Addresses are reserved for a variety of purposes, from specific Internet protocols to given organizations that make heavy use of multicast traffic, including multimedia broadcasters and financial institutions. The remainder of the multicast addresses are available for use by temporary groups, which must be created before use and cease to exist when all the members have left.

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.

Java API to IP multicast • 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.

3.3.2 Reliability and ordering of multicast

A datagram sent from one multicast router to another may be lost, thus preventing all recipients beyond that router from receiving the message. Also, when a multicast on a local area network uses the multicasting capabilities of the network to allow a single datagram to arrive at multiple recipients, any one of those recipients may drop the message because its buffer is full. Another factor is that any process may fail. If a multicast router fails, the group

members beyond that router will not receive the multicast message, although local members may do so.

Ordering is another issue. IP packets sent over an internetwork do not necessarily

arrive in the order in which they were sent, with the possible effect that some group members receive datagrams from a single sender in a different order from other group

members. Characteristics are,

1. Fault tolerance based on replicated services: Consider a replicated service that consists of the members of a group of servers that start in the same initial state and always perform the same operations in the same order, so as to remain consistent with one another.

2.Discovering services in spontaneous networking: One way for a process to discover services in spontaneous networking is to multicast requests at periodic intervals, and for the available services to listen for those multicasts and respond.

3. Better performance through replicated data: Consider the case where the replicated data itself, rather than operations on the data, are distributed by means of multicast messages. The effect of lost messages and inconsistent ordering would depend on the method of replication and the importance of all replicas being totally up-to-date.

4. Propagation of event notifications: The particular application determines the qualities required of multicast.

3. 4 Remote Procedure call

Applications composed of cooperating programs running in several different processes. Such programs need to invoke operations in other processes.

RPC – client programs call procedures in server programs, running in separate and remote computers
RMI – an object in one process can invoke methods of objects in another process

The earliest and perhaps the best-known programming model for distributed programming allows client programs to call procedures in server programs running in separate processes and generally in different computers from the client. RPC is a kind of request–response protocol. An RPC is initiated by the *client*, which sends a request message to a known remote *server* to execute a specified procedure with supplied parameters. The remote server sends a response to the client, and the application continues its process. While the server is processing the call, the client is blocked (it waits until the server has finished processing before resuming execution), unless the client sends an asynchronous request to the server.

3.4.1 Design issues for RPC

There are three issues that are important in understanding this concept:

• 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, in particular, each server provides a set of procedures that are available for use by clients. An RPC mechanism can be integrated with a particular

programming language if it includes an adequate notation for defining interfaces. Interface definition languages (IDLs) are designed to

allow procedures implemented in different languages to invoke one another.

RPC call semantics: The main choices are,

- 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.

The choices of RPC

invocation semantics are defined as follows,

- Maybe semantics: With maybe semantics, the remote procedure call may be executed

once or not at all.

- 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,

or an exception informing it that no result was received. At-least-once semantics can be

achieved by the retransmission of request messages.

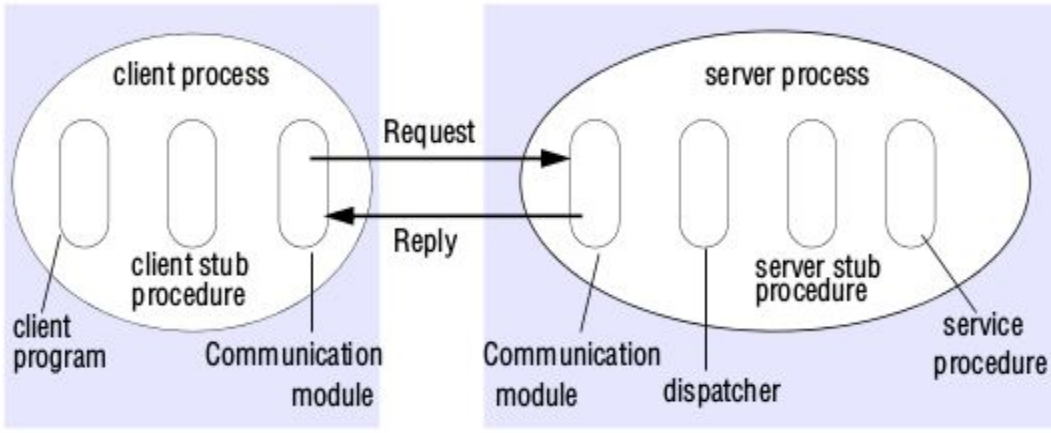
- At-most-once semantics: With at-most-once semantics, the caller receives either a

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.

3.4.2 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 un-marshals 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.

Role of client and server stub procedures in RPC



Sequence of events

1. The client calls the client stub. The call is a local procedure call, with parameters pushed on to the stack in the normal way.
2. The client stub packs the parameters into a message and makes a system call to send the message. Packing the parameters is called marshalling.
3. The client's communication module sends the message from the client machine to the server machine.
4. The communication module on the server machine passes the incoming packets to the dispatcher.
5. The dispatcher selects one of the server stub procedures according to the procedure identifier in the request message.
6. The server stub unpacks the parameters from the message. Unpacking the

parameters is called unmarshalling.

7. Finally, the server stub calls the server procedure. The reply traces the same steps in the reverse direction.

3.5 Network virtualization

Network virtualization is the process of combining hardware and software network resources and network functionality into a single, software-based administrative entity. 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. Network virtualization is accomplished by using a variety of hardware and software and combining network components. Network virtualization is especially useful for networks experiencing a rapid, large and unpredictable increase in usage.

Network virtualization is categorized as either,

- **external virtualization**, combining many networks or parts of networks into a virtual unit. One example is virtual LAN (VLAN).
- **internal virtualization**, providing network-like functionality to software containers on a single network server such as Xen hypervisor.

What are the benefits of virtualization?

- Reduces the number of physical devices needed
- Easily segment networks
- Permits rapid change / scalability and agile deployment
- Security from destruction of physical devices
- Failover mode – defective disk simply switches to a backup on the fly, and the failed component can be repaired, while the system continues to run

3.5.1 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). All nodes in an overlay network are connected with one another by means of logical or virtual links and each of these links correspond to a path in the underlying network.

An example of an overlay network can be distributed systems such as client-server applications and peer-to-peer networks. Such applications or networks act as the overlay networks because all nodes in these applications and networks run on top of the internet.The main uses of overlay networks can be found in the telecom industry.

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.

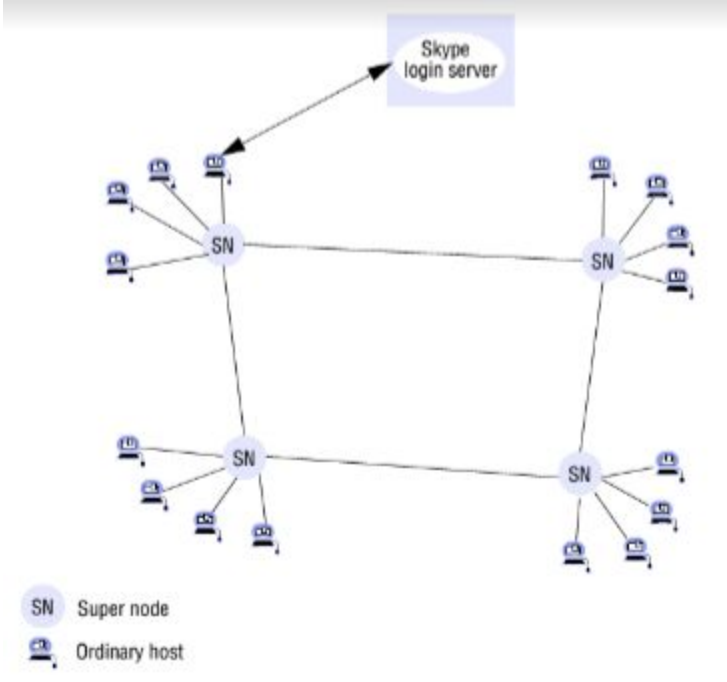
• Multiple overlays can be defined and can coexist, with the end result being a more open and extensible network architecture.

3.6 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 a virtual network in that it establishes connections between people. 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.

3.6.1 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 (and availability.

An ordinary host must connect to a super node and must register itself with the Skype login server for a successful login. The Skype login server is an important entity in the Skype network. User names and passwords are stored at the login server. User authentication at

login is also done at this server. This server also ensures that Skype login names are unique across the Skype name space.

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, five to six seconds, if behind a NAT-enabled router).

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.