DISTRIBUTED COMPUTING MODULE 3

Interprocess Communication

The characteristics of interprocess communication

- message communication operations:
  - *send*
  - *receive*

<u>**Synchronous and Asynchronous communication**</u>

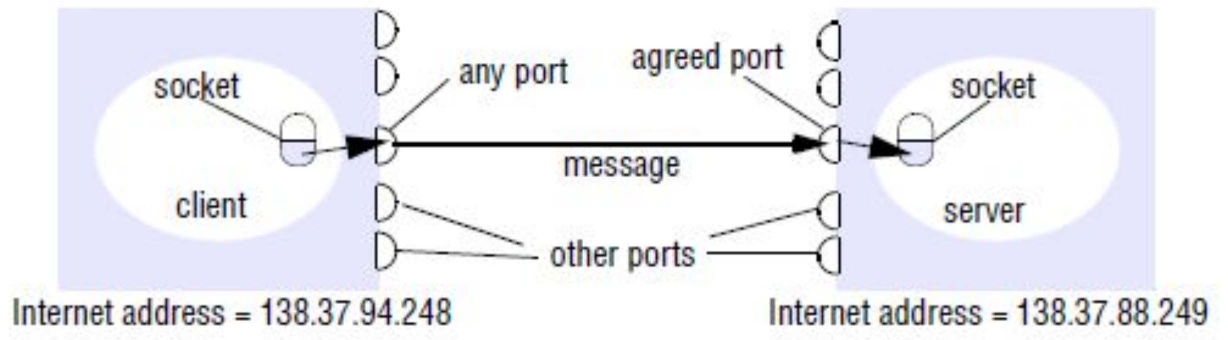
- **A queue is associated with each** message destination
- Sending processes - add messages to remote queues
- Receiving processes - remove messages from local queues
- Sending & receiving processes – synchronous or asynchronous

- Message destinations (Internet Address, Local port)
  - Port: one receiver, many senders
  - Name and name server instead of fixed internet address
- Reliability
- Ordering

Sockets

- Messages are transmitted between a socket in one process and a socket in another process

Sockets and ports


socket
client
any port
agreed port
message
other ports
socket
server
Internet address = 138.37.94.248
Internet address = 138.37.88.249

- For a process to receive messages, its socket must be bound to a local port and one of the Internet addresses

- same socket for sending and receiving messages

- Each computer has a large number (2^{16}) of possible port numbers

- Any process
  - May make use of multiple ports to receive messages
  - Cannot share ports with other processes on the same computer

- Each socket is associated with a particular protocol – either UDP or TCP.

Java API for Internet addresses

- Java provides a class, *InetAddress, that*
  - *represents Internet* addresses.

- Users of this class refer to computers by Domain Name System (DNS) hostnames

Eg. To get an object representing the Internet address of the host whose DNS name is *bruno.dcs.qmul.ac.uk, use:*

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

UDP DATAGRAM COMMUNICATION

- UDP datagram transmission is without acknowledgements or retries

- process must first create a socket
  - bound to an Internet address of the local host and a local port

- A server will bind its socket to a *server port* –
  - *makes it known to clients so that they can send messages to it*

- *A client* binds its socket to any free local port

- The *receive method returns*
  - *the Internet address* and port of the sender
  - the message

  (allowing the recipient to send a reply)

Issues relating to datagram communication:

- *Message size:*
  - receiving process needs to specify an array of bytes
  - The underlying IP protocol packet lengths of up to 216 bytes
  - However, most environments impose a size restriction of 8 kilobytes

- *Blocking:*
  - Sockets normally provide non-blocking *sends and blocking receives for* datagram communication

    For example, when a server receives a message from a client, the message may specify work to do, in which case the server will use separate threads to do the work and to wait for messages from other clients**.**

Issues relating to datagram communication(contd):

- Timeouts:
  - timeouts can be set on sockets
  - Choosing an appropriate timeout interval is difficult

- Receive from any:
  - The *receive method does not specify an origin for messages*
  - invocation of *receive gets a message addressed to its socket from any* origin
  - The *receive method returns the Internet address and local port of the sender*
  - It is possible to connect a datagram socket to a particular remote port and Internet address

Failure model for UDP datagrams:

- *Omission failures:*
  - *Messages may be dropped occasionally, because of*
    - *a* checksum error
    - no buffer space is available at the source or destination
  - Send-omission & receive-omission failures in the communication channel

- *Ordering:*
  - *Messages can sometimes be delivered out of sender order*

Workaround: A reliable delivery service may be constructed from one that suffers from omission failures by the use of acknowledgements.

Use of UDP

- For some applications occasional omission failures are accepatable

  Example:

  - the Domain Name System, which looks up DNS names in the Internet, is implemented over UDP.
  - Voice over IP (VOIP) also runs over UDP.

- No overheads associated like with guaranteed message delivery

- There are three main sources of overhead:
  - the need to store state information at the source and destination
  - the transmission of extra messages
  - latency for the sender

TCP STREAM COMMUNICATION

- The API to the TCP protocol provides the abstraction of a stream of bytes

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

1. Message sizes:

- *The application can choose how much data it writes to a stream or* reads from it.
- On arrival, the data is handed to the application as requested
- Applications can, if necessary, force data to be sent immediately

2. Lost messages*: The TCP protocol uses an acknowledgement scheme*

- *Buffers*
- *Acknowledgements*
- *Timeouts*
- *Sliding window protocol*

3. Flow control:

- *match the speeds of the processes that* read from and write to a stream
- Fast sender slow receiver

4. Message duplication and ordering:

- *Message identifiers are associated with each IP* packet
- recipient to detect and reject duplicates
- reorder messages that do not arrive in sender order

5. Message destinations:

- *establish a connection* before they can communicate over a stream to
- Establishing a connection involves a
  - *connect request from client* to server
  - *accept request from server to client*
- considerable overhead for a single client-server request and reply

- **client role :**
  - **Creating a stream socket** bound to any port
  - Making a ***connect request*** *asking for a connection to a server* at its server port

- **Server role :**
  - Creating a **listening socket** bound to a server port
  - **Waiting** for clients to **request** connections

- Two streams in each pair of sockets:
  - **Input stream**
  - **Output stream**

  - Application *closes a socket*

    ->will not write any more data to its output stream
  - A process exits or fails

    -> all of its sockets are eventually closed

    -> connection to communicating processes will be broken

OUTSTANDING ISSUES RELATED TO STREAM COMMUNICATION:

Matching of data items:

- *Agreement of the* contents of the data transmitted over a stream.

  For example, if one process writes an *int followed by a double to a stream, then the reader at the other end must read an int* followed by a *double.*

- Without proper cooperation receiving process:
  - Experience errors when interpreting the data
  - May block due to insufficient data in the stream

- **Failure model**
  - TCP uses timeouts and retransmissions to deal with lost packets
    - messages are guaranteed to be delivered
  - Connection is declared broken if
    - if the packet loss over a connection passes some limit
    - the network connecting a pair of communicating processes is severed
    - Network becomes severely congested
    - the TCP software responsible for sending messages will receive no acks

- **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 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 of bytes comprising a message, the length of the message and the Internet address and local port number of the destination socket:

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

- An instance of DatagramPacket may be transmitted between processes when one process sends it and another receives it.
- This class provides another constructor for use when receiving a message. Its arguments specify an array of bytes in which to receive the message and the length of the array.
- A received message is put in the DatagramPacket together with its length and the Internet address and port of the sending socket.
- The message can be retrieved from the DatagramPacket by means of the method getData.
- The methods getPort and getAddress access the port and Internet address.

***DatagramSocket**:*

This class supports sockets for sending and receiving UDP datagrams.

- It provides a constructor that takes a port number as its argument, for use by processes that need to use a particular port.
- It also provides a no-argument constructor that allows the system to choose a free local port.
- These constructors can throw a *SocketException if the chosen port is already in use or if a reserved port (a* number below 1024) is specified when running over UNIX.

- The class *DatagramSocket provides methods that include the following:*
  - *send and receive: These methods are for transmitting datagrams between a pair* of sockets.
  - The argument of *send is an instance of DatagramPacket containing* a message and its destination.
  - The argument of *receive is an empty DatagramPacket in which to put the message, its length and its origin.*

- *The* methods *send and receive can throw IOExceptions.*

- *setSoTimeout: This method allows a timeout to be set. With a timeout set, the receive* method will block for the time specified and then throw an *InterruptedIOException.*

- *connect: This method is used for connecting to a particular remote port and* Internet address, in which case the socket is only able to send messages to and receive messages from that address.

UDP client sends a message to the server and gets a reply

```
import java.net.*;
import java.io.*;
public class UDPClient{
  public static void main(String args[]){
    // args give message contents and server hostname
    DatagramSocket aSocket = null;
    try {
        aSocket = new DatagramSocket();
        byte [] m = args[0].getBytes();
        InetAddress aHost = InetAddress.getByName(args[1]);
        int serverPort = 6789;
        DatagramPacket request =
            new DatagramPacket(m,  m.length(), aHost, serverPort);
        aSocket.send(request);
        byte[] buffer = new byte[1000];
        DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
        aSocket.receive(reply);
        System.out.println("Reply: " + new String(reply.getData()));
     } catch (SocketException e){System.out.println("Socket: " + e.getMessage());
     } catch (IOException e){System.out.println("IO: " + e.getMessage());
     } finally { if(aSocket != null) aSocket.close();}
  }
}
```

Java API for TCP streams

The Java interface to TCP streams is provided in the classes se*rverSocket 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. Its accept method gets a connect* request from the queue or, if the queue is empty, blocks until one arrives. The result of executing *accept is an instance of Socket – a socket to use for communicating with* the client.

- *Socket: This class is for use by a pair of processes with a connection. The client uses* a constructor to create a socket, specifying the DNS hostname and port of a server. This constructor not only creates a socket associated with a local port but also *connects it to the specified remote computer and port number. It can throw an UnknownHostException if the hostname is wrong or an IOException if an IO error* occurs.

TCP client makes connection to server, sends request and receives reply

```
import java.net.*;
import java.io.*;
public class TCPClient {
      public static void main (String args[]) {
         // arguments supply message and hostname of destination
         Socket s = null;
         try{
           int serverPort = 7896;
           s = new Socket(args[1], serverPort);
           DataInputStream in = new DataInputStream( s.getInputStream());
           DataOutputStream out =
                new DataOutputStream( s.getOutputStream());
           out.writeUTF(args[0]);       // UTF is a string encoding; see Sec 4.3
           String data = in.readUTF();
           System.out.println("Received: "+ data) ;
         }catch (UnknownHostException e){
           System.out.println("Sock:"+e.getMessage());
         } catch (EOFException e){System.out.println("EOF:"+e.getMessage());
         } catch (IOException e){System.out.println("IO:"+e.getMessage());
         } finally {if(s!=null) try {s.close();}catch (IOException e){/*close failed*/}}
      }
}
```

EXTERNAL DATA REPRESENTATION AND MARSHALLING

- Any two computers can exchange binary data values if
  - The values are **converted** to an agreed external format
  - if **computers are of same type**, no conversion
  - transmitted in the **sender's format** with an **indication of the format**

- ***Marshalling***
  - *Assembling and convertving a collection of data items for transmission*

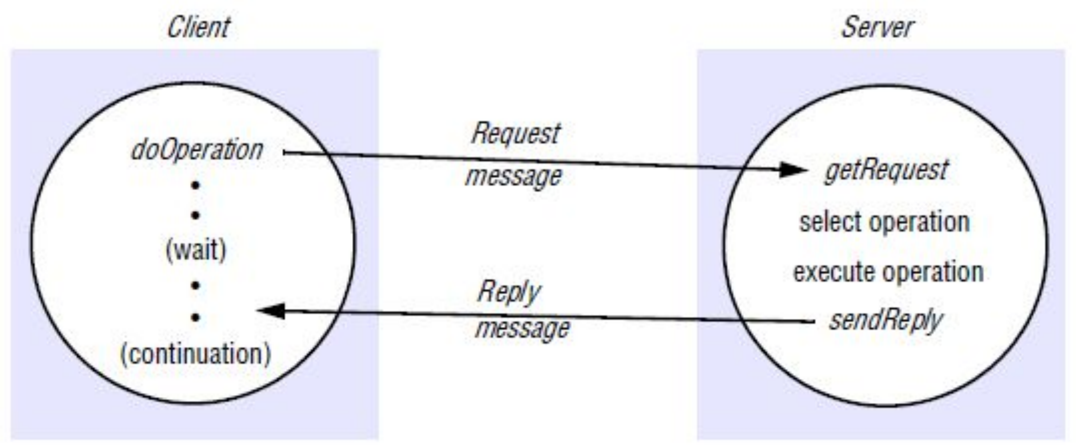
- ***Unmarshalling***
  - disassembling them on arrival to produce original data items

- Three approaches:
  - CORBA’s common data representation
  - Java’s object serialization
  - XML (Extensible Markup Language)
- Remote object references:
  - A *remote object reference is an identifier* for a remote object that is valid throughout a distributed system

REMOTE PROCEDURE CALL

- High level of distribution transparency.
- Extends the abstraction of a procedure call to distributed environments.
- Procedures on remote machines can be called as if they are procedures in the local address space
- RPC system then hides important aspects of distribution
  - the encoding and decoding of parameters and results
  - the passing of messages
  - the preserving of the required semantics for the procedure call

Request-reply communication



DESIGN ISSUES FOR RPC

1. **Programming with interfaces**

   Communication can be by means of

   – procedure calls between modules

   – direct access to the variables in another module

- Explicit interface is defined for each module
- hide all information except which is available through its interface
- implementation may be changed without affecting the users

Interfaces in distributed systems

- Modules of a distributed program can run in separate processes.
  - Eg. a file server would provide procedures for reading and writing files

- ***service interface*** :
  - the specification of the procedures offered by a server
  - defines the types of the arguments of each of the procedures

Benefits of programming with interfaces:

- concerned only with the **abstraction not implementation** details
- No need to know the **programming language** or underlying **platform** used to implement the service
- Support for **software evolution** in that
  - **implementations can change** as long as interface remains same
  - interface can also change as long as it remains compatible with the original

Nature of the underlying infrastructure:

- Client running in one process cannot access variables in another

  (no direct access to variables)

- Parameter passing:
  - no call by value or call by reference.
  - Input and output parameters

- Addresses cannot be passed as arguments

INTERFACE DEFINITION LANGUAGES

An RPC mechanism can be integrated with a particular programming language if it:

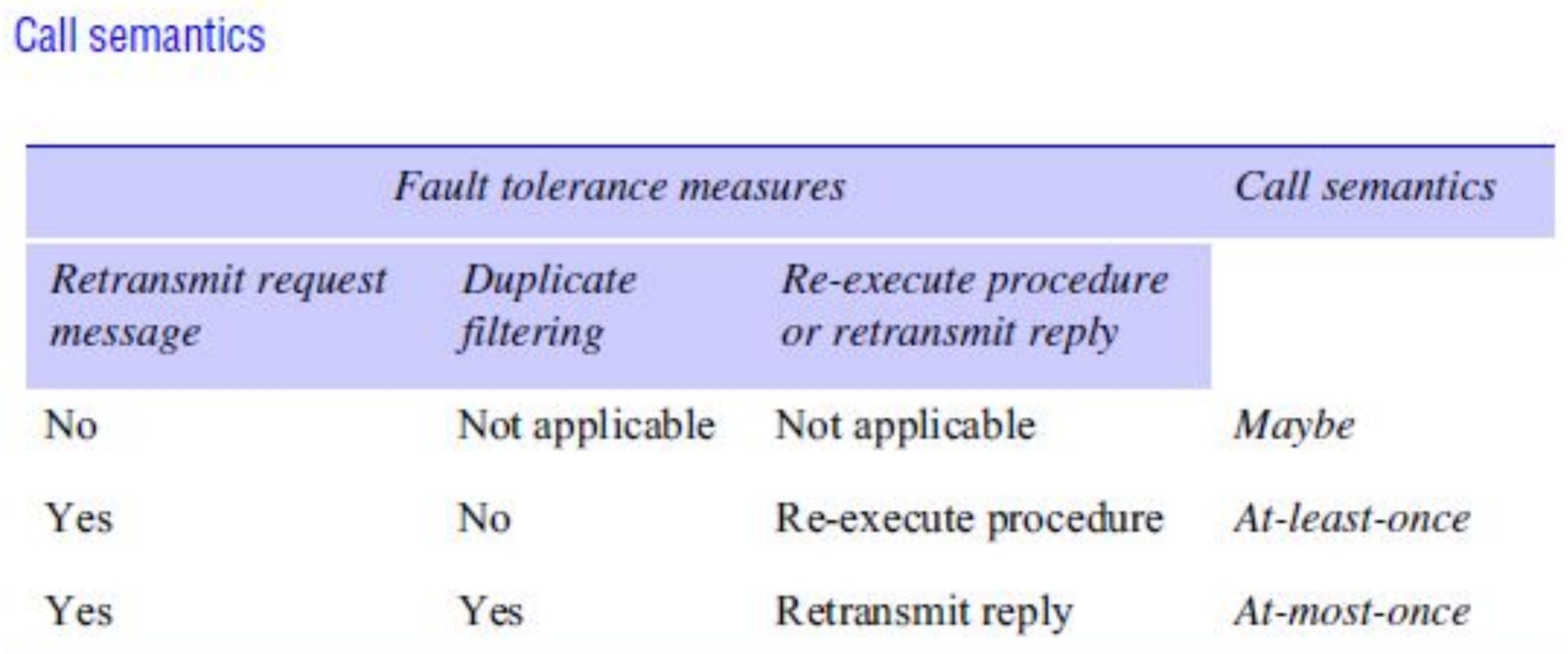
- Has an adequate notation for defining interfaces
- allows input and output parameters to be mapped onto the language's normal use

- useful when all the parts of a distributed application can be written in the same language

- allows the programmer to use a single language,
  - Eg. Java for local and remote invocation

- many existing useful services are written in C++
- Beneficial to allow programs written in a variety of languages, including Java, to access them remotely.
- *Interface definition languages (IDLs)* allow procedures implemented in different languages to invoke one another
- each of the parameters of an operation may be described as for input or output in addition to having its type specified

RPC call semantics



Call semantics

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*

- For local procedure calls, the semantics are *exactly once, meaning that every procedure is* executed exactly once

- For RPC:

1. Maybe semantics:

- Failures-
  - omission failures if the request or result message is lost
  - crash failures when the server containing the remote operation fails

- Maybe semantics is useful only for applications in which occasional failed calls are acceptable.

2. At-least-once semantics:

- The invoker receives
  - either a result
  - an exception informing it that no result was received
- Failures-
  - crash failures when the server containing the remote procedure fails
  - arbitrary failures : in cases when the request message is retransmitted

- Server may receive it and execute the procedure more than once, possibly causing wrong values to be stored or returned

- ***Idempotent operation****:*

  operation that can be performed repeatedly with the same effect as if it had been performed exactly once

- *at-least-once call semantics acceptable for idempotent operations*

3. At-most-once semantics:

- The caller receives
  - **either a result**
  - **an exception** informing it that no result was received

    (executed either once or not at all)

- Ensures that for each RPC a procedure is never executed more than once

- Sun RPC provides at-least-once call semantics.

Transparency

The originators of RPC, Birrell and Nelson [1984], aimed to make remote procedure calls as much like local procedure calls as possible

- All the necessary calls to marshalling and message-passing procedures were hidden from the programmer
- Retransmission after a timeout is transparent to the caller
- makes the semantics of remote procedure calls like that of local procedure calls
- RPC strives to offer at least location and access transparency

- RPCs are more vulnerable to failure than local ones
- involve a network, another computer and another process
- There is always the chance that no result will be received
- Impossible to distinguish between failure of the network and of the remote server process
- Clients should be able to recover from such situations
- Latency of a RPC is greater than that of a local one
- Hence, minimize remote interactions

- A caller should be able to abort a remote procedure call that is taking too long in such a way that it has no effect on the server.

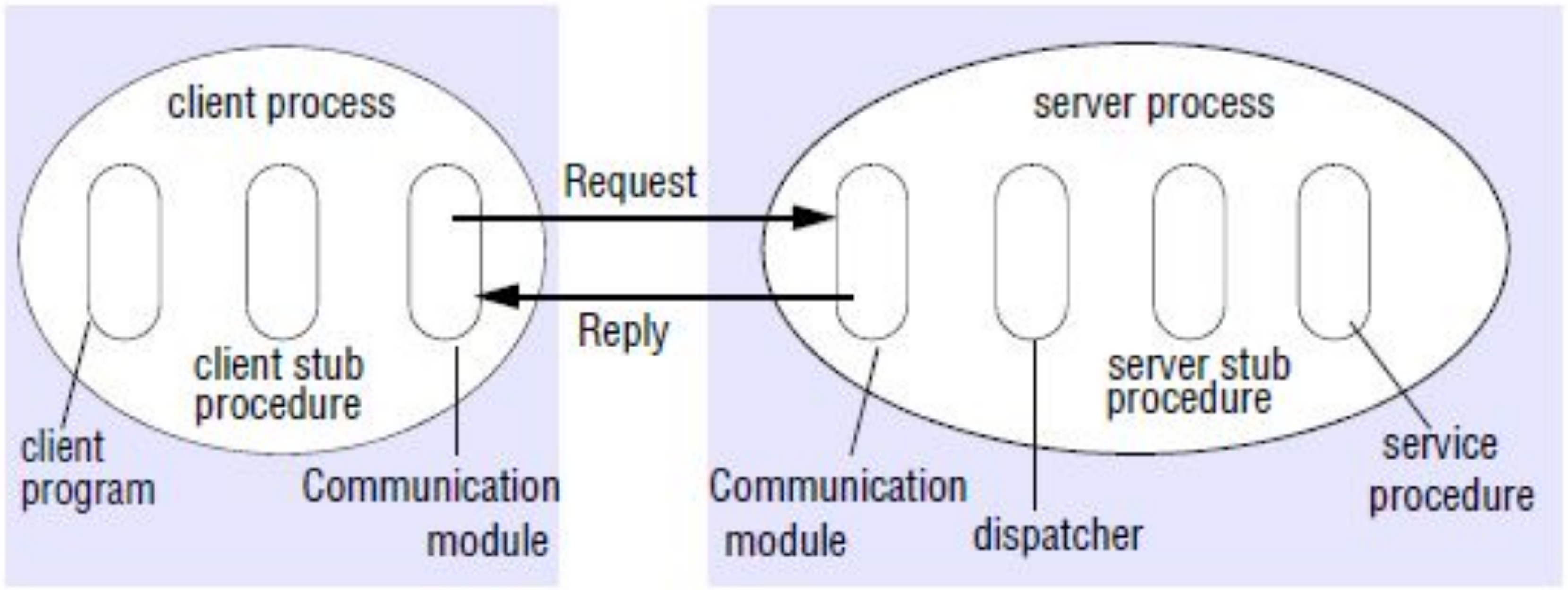
- Proposed use of Different syntax for RPC

- The choice as to whether RPC should be transparent is also available to the designers of IDLs

- The **current consensus** is that remote calls should be made transparent in the sense that the syntax of a remote call is the same as that of a local invocation, but that the difference between local and remote calls should be expressed in their interfaces

2. Implementation of RPC

Role of client and server stub procedures in RPC



- The client that accesses a service includes one ***stub procedure** for each procedure in the* service interface.

- 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

MULTICAST COMMUNICATION

- Implement a service as multiple processes in different computers
  - to provide fault tolerance
  - to enhance availability
- One process->send single msg->received by every member of group
- usually the membership of the group is transparent to the sender

Multicast msgs provide DS with following characteristics:

1. ***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
   - Even when some of the members fail, clients can still be served

2. ***Discovering services in spontaneous networking**:*
   - servers and clients to locate available discovery services
   - look up the interfaces of other services in the distributed system

3. ***Better performance through replicated data**:*

- replicas of the data are placed in users' computers
- When data changes, the new value is multicast to the manage the replicas

4. ***Propagation of event notifications**:*

- *notify* processes when something happens
- Facebook: 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

Built on top of the Internet Protocol (IP).

- allows sender to transmit a single IP packet to a set of computers
- A *multicast group is specified by a* ***Class D Internet address***
- The membership of multicast groups is dynamic
- Possible to send datagrams to a multicast group without being a member
- IP multicast is **available only via UDP**
- send UDP datagrams with multicast addresses and ordinary port numbers
- 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.

- The following details are specific to IPv4:
  - ***Multicast routers***: *multicast both on a local network and the* Internet
  - ***Multicast address allocation***:
    - Class D addresses (224.0.0.0 to 239.255.255.255)
    - managed globally by the Internet Assigned Numbers Authority (IANA)
    - Addresses are partitioned into number of blocks:
      - Local Network Control Block (224.0.0.0 to 224.0.0.225)
      - 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)
      - Administratively Scoped Block (239.0.0.0 to 239.255.255.255)
    - Multicast addresses may be permanent or temporary

- Addresses are reserved (allocated to organizations)

- Remaining multicast addresses are available for use by temporary groups:
  - created before use
  - cease to exist when all the members have left

- Creating temporary group requires a free multicast address
  - to avoid accidental participation in an existing group

- Multicast address allocation architecture (MALLOC)
  - for Internet-wide applications
  - allocates unique addresses for a given period of time and in a given scope

- A client-server solution is adopted :
  - clients request a multicast address from a multicast address allocation server (MAAS)
  - communicated across domains to ensure allocations are unique

Failure model for multicast datagrams

- Omission failure: Same failure characteristics as UDP datagrams

- *Unreliable* multicast: Some but not all of the members of the group may receive it.

Java API to IP multicast

- **The Java API provides a datagram interface to IP multicast**
  - class *MulticastSocket (a subclass of DatagramSocket)*

- Has additional capability of being able to join multicast groups

- *MulticastSocket* provides two alternative constructors
  - sockets to be created using a specified local port
  - Sockets created using any free local port

- *joinGroup method: to join a multicast at a given port*

- *leaveGroup method: to leave a multicast group*

- *setTimeToLive method: TTL can be set for a multicast socket*

- An application implemented over IP multicast may use more than one port

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()));
        }
        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();}
    }
}
```

Reliability and ordering of multicast

- datagram sent from a multicast router to another may be lost
- Multicast on a local area network –
  - any one of the recipients may drop message because its buffer is full
- Any process may fail eg. Multicast router
- Ordering is another issue:
  - Order of msgs from same sender to multiple receivers
  - Order of msgs from multiple senders to receivers

Some examples of the effects of reliability and ordering

1. ***Fault tolerance based on replicated services**:*
   - either all of the replicas or none should receive each request to perform an operation
   - if one of them misses a request, it will become inconsistent with the others

2. ***Discovering services in spontaneous networking**:*
   - An occasional lost request is not an issue when discovering services

3. ***Better performance through replicated data**:*
   - replicated data itself are distributed by means of multicast messages
   - All replicas may not be totally up-to-date

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

GROUP COMMUNICATION

- *Group communication offers a service whereby a message is*
  - *sent to a group*
  - delivered to all members of the group
- Indirect communication paradigm
- The sender is not aware of the identities of the receivers
- An abstraction over multicast communication
  - may be implemented over IP multicast or an equivalent overlay network
- Adds significant extra value in terms of
  - managing group membership
  - detecting failures
  - providing reliability and ordering guarantees

Group communication is an important building block of reliable DS. **Application Areas** :

- dissemination of **info to large numbers of clients**
  - including in the financial industry
- support for **collaborative applications**
  - to preserve a **common user view** like in multiuser games
- support for a range of **fault-tolerance strategies**
  - the consistent update of replicated data
  - the implementation of highly available (replicated) servers
- support for **system monitoring and management**
  - including for example load balancing strategies

The programming model

Central concept is of a *group with associated group membership, whereby processes may join or leave the group.*

- Implements **multicast communication**
- 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))*
- Enables the implementation to be **efficient in its utilization of bandwidth**
- send the message no more than once over any communication link,
  - by sending it over a **distribution tree**
- Can use **network hardware support** for multicast where this is available
- **minimize the total time taken** to deliver the message to all destinations

- The use of a single multicast operation:
  - also important in terms of delivery guarantees
  - relative ordering of 2 messages delivered to 2 group members is undefined in individual sends

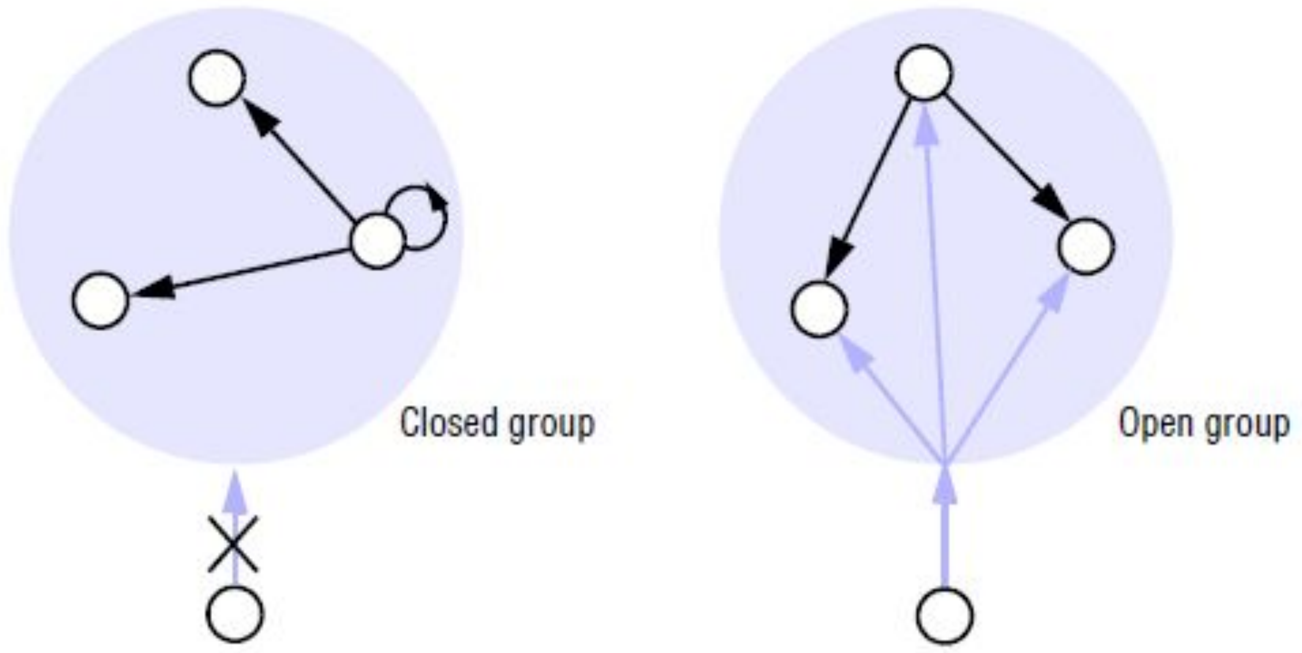
Process groups and object groups

- **Process groups**: communicating entities are processes
- Such services are relatively low-level:
  - Messages are delivered to processes
  - no further support for dispatching is provided
  - Messages are typically unstructured byte array
  - no support for marshalling of complex data types (as in RPC or RMI)
- The level of service provided by process groups is therefore similar to that of sockets

- **Object group:** A collection of objects (normally instances of the same class)
- process the same set of invocations concurrently, with each returning responses
- Client objects need not be aware of the replication
- 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
- Associated calls dispatched automatically to the right destination objects/methods.
- Electra, Eternal, Object Group Service – CORBA compliant systems that support object groups
- Process groups still dominate in terms of usage.

Some Key Distinctions

- Open and Closed Groups:



- Overlapping and Non-Overlapping Groups
- Synchronous and Asynchronous Systems

IMPLEMENTATION ISSUES

Reliability and ordering in multicast

- The guarantees include
  - every process in the group should receive the set of messages sent
  - delivery ordering across the group members

- Group communication systems are extremely sophisticated.
  - Even IP multicast, which provides minimal delivery guarantees, requires a major engineering effort

- *Agreement:* If message is delivered to one process, then it is delivered to all processes in the group.

- Reliability :
  - Integrity
  - Validity

Guarantees in terms of the relative ordering of messages

- *Group Communication offers ordered multicast* with the option of one or more of the following properties:
  - *FIFO ordering*
  - *Causal ordering*
  - *Total ordering*

- Reliability and ordering are examples of coordination and agreement in distributed systems

Group membership management

- maintains an accurate *view of the current membership,*
  - *given that entities may join, leave or indeed* fail.
- a group membership service has four main tasks:

1. *Providing an interface for group membership changes:*
   - create and destroy process groups
   - to add or withdraw a process to or from a group

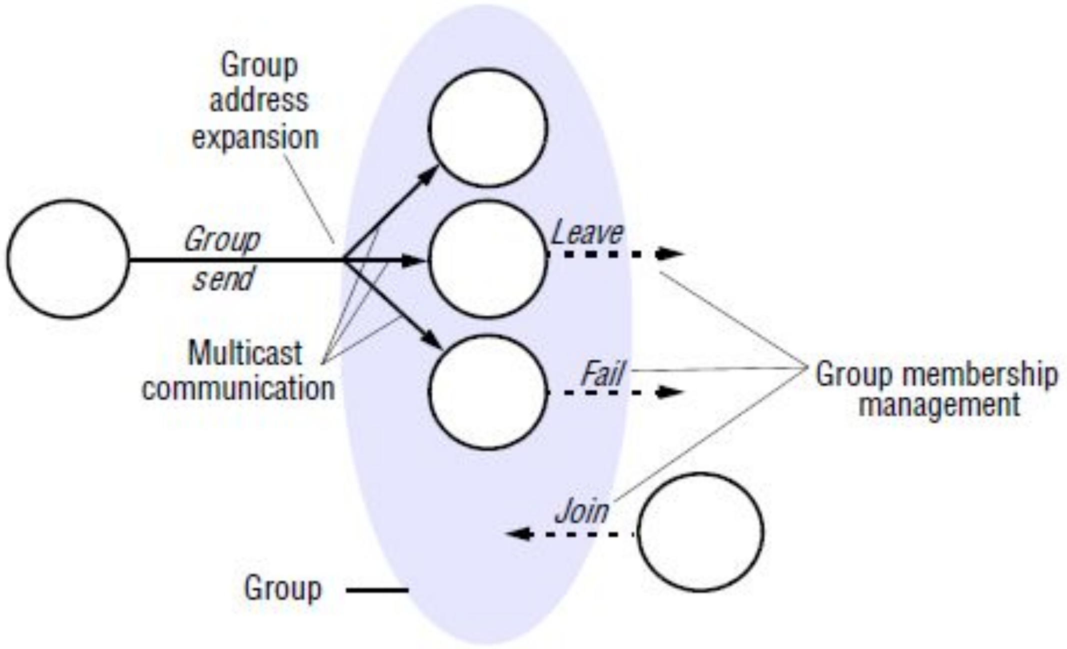
2. *Failure detection:*
   - *M*onitors the group members for crash, unreachability.
   - The detector marks processes as *Suspected or Unsuspected*
   - failure detector to reach a decision about the group's membership

3. *Notifying members of group membership changes*

4. *Performing group address expansion:*

- coordinate multicast delivery with membership changes by controlling address expansion
- decide consistently where to deliver any given message, even though the membership may be changing during delivery

The role of group membership management



- IP multicast is a weak case of a group membership service:
  - allows processes to **join or leave groups dynamically**
  - it **performs address expansion**
  - does not itself provide group members with **information about current membership**
  - multicast **delivery is not coordinated with membership changes**.

- Achieving these properties is complex and requires ***view-synchronous group communication***

- group communication is most effective in small-scale and static systems

- does not operate as well in
  - larger-scale environments
  - environments with a high degree of volatility

NETWORK VIRTUALIZATION

- different classes of application coexist in the Internet
- Eg. peer-to-peer file sharing and Skype

1. impractical to attempt to alter the Internet protocols to suit each
2. IP -implemented over a large and ever increasing number of network technologies

These two factors have led to the interest in network virtualization.

- Network Virtualization:
  - construction of many different virtual networks over an existing network (such as the Internet)

  - Each VN designed to support a particular distributed application

  For example,
  - one virtual network might support multimedia streaming
  - another that supports a multiplayer online game
  - both running over the same underlying network

OVERLAY NETWORKS

- *a virtual network consisting of nodes and virtual links*
- *sits* on top of an underlying network (such as an IP network)
- offers something that is not otherwise provided:

  – A service that is tailored towards the needs of a class of application

  Eg. multimedia content distribution

  – More efficient operation in a given networked environment

  Eg. routing in an ad hoc network

  – An additional feature

  Eg. multicast or secure communication

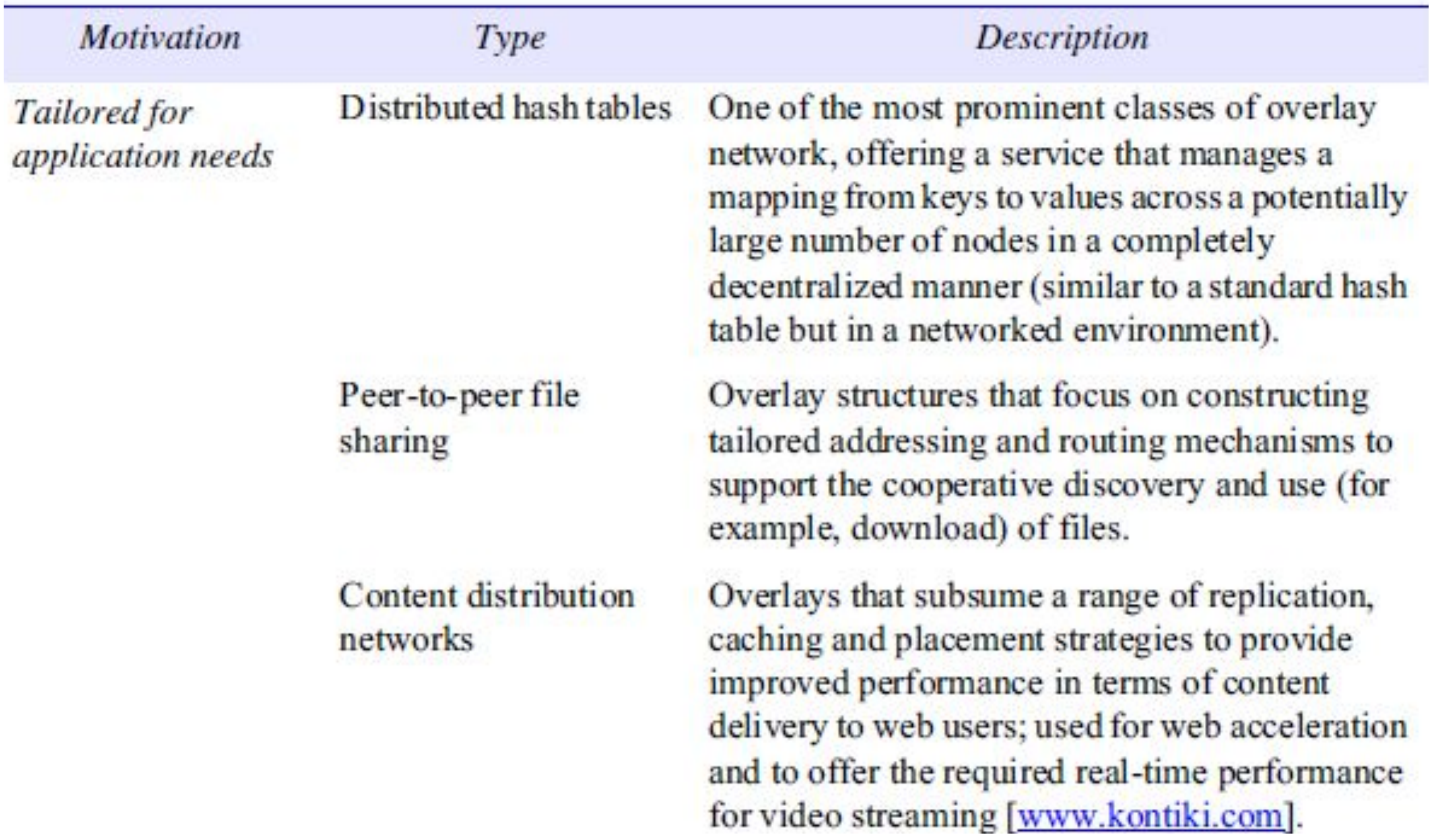
Advantages

- enable new network services to be defined
- doesn’t require changes to the underlying network
- encourage experimentation with network services
- customization of services to particular classes of application
- Multiple overlays can be defined and can coexist
- more open and extensible network architecture

Disadvantages

- Overlays introduce an extra level of indirection
- hence may incur a performance penalty
- add to the complexity of network services

Types of Overlays

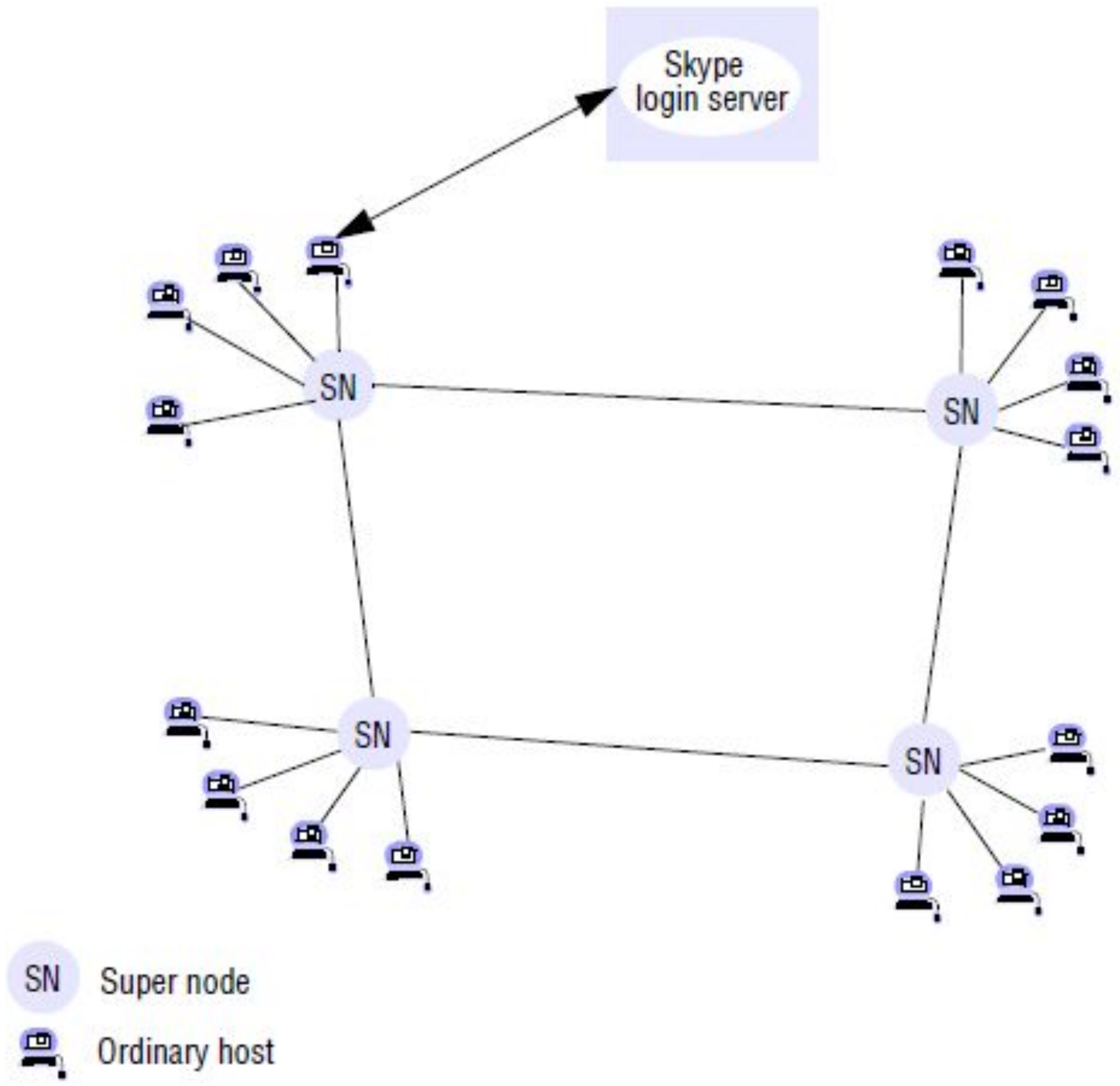


Motivation	*Type*	*Description*
Tailored for application needs	Distributed hash tables	One of the most prominent classes of overlay network, offering a service that manages a mapping from keys to values across a potentially large number of nodes in a completely decentralized manner (similar to a standard 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].

MOTIVATION	TYPE	DESCRIPTION
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.

SKYPE

Skype overlay architecture



SKYPE:

- a peer-to-peer application offering Voice over IP (VoIP)
- also includes
  - instant messaging
  - video conferencing
  - interfaces to the standard telephony service

- The software was developed by Kazaa in 2003
  - shares many characteristics of Kazaa peer-to-peer file-sharing app

- a virtual network - it establishes connections between people

- Skype subscribers who are currently active

- No IP address or port is required to establish a call

Skype architecture

- **Skype is based on a peer-to-peer infrastructure consisting**
  - ordinary users’ machines (referred to as hosts)
  - super nodes

- Super nodes are :
  - ordinary Skype hosts
  - happen to have sufficient capabilities to carry out their enhanced role
  - selected on demand, based on a range of criteria including
    - bandwidth available
    - reachability
    - availability

User connection

- Skype users are authenticated via a **well-known login server**
- They then make contact with a selected super node
- each client maintains a cache of super node identities
  - (that is, IP address and port number pairs)
- At first login cache is filled with addresses of 7 super nodes
- 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**
  - **efficient search of the** global index of users
  - distributed across the super nodes

- The search is orchestrated by the client’s chosen super node
  - an expanding search of other super nodes
  - until the specified user is found

- On average, **eight super nodes** are contacted.

- Time taken for a user search:
  - typically 3-4 seconds for hosts that have a global IP address
  - slightly longer (5-6 seconds) if behind a NAT-enabled router

- intermediary nodes cache the results to improve performance

Voice connection

Once the required user is discovered:

- **Skype establishes a voice** connection using
  - TCP for signalling call requests and terminations
  - either UDP or TCP for the streaming audio
    - UDP is preferred
    - TCP with intermediary node, sometimes used to circumvent firewalls

- The software used for encoding and decoding audio
  - Plays key part in providing the excellent call quality

- algorithms tailored to operate at 32 kbps and above

THANK YOU!