Communication
Communication Model

- Application
- Middleware
- Transport
- Network
- Data link
- Physical

Protocol Levels:
1. Physical protocol
2. Data link protocol
3. Network protocol
4. Transport protocol
5. Middleware protocol
6. Application protocol
Types of communication

• Number of destinations
  – Unicast, multicast, broadcast, [anycast,..]
• Address of the destinations
  – Explicit, Implicit [content…]
• Message oriented
  – Persistent vs transient
  – Synchronous (blocking) vs Asynchronous
• Stream oriented
Type of communication: Persistent

- Middleware stores the message until it has been delivered
- Sender has NOT be active when the message is delivered to the destination
- Destination has NOT to be active when the message is generated by the source
Type of communication: Transient

- Middleware stores the message during when source/destination are active
Communication middleware
RPC - principle

- Simple idea (?!)
- Process on machine A can (remotely) call a procedure on machine B
- All the administrative stuffs are done automatically by a middleware
  - At compile time (generation of additional SW)
  - At runt time (SW support)
RPC – run time architecture

Client

Calling procedure

Stub

result
parameters

Request message

Server

Called procedure

Skeleton

parameters
result

Reply message

Network
RPC – time diagram

Client
- Call remote procedure
- Request
- Wait for result
- Reply
- Return from call

Server
- Call local procedure and return results
RPC - example

1. Client call to procedure
2. Stub builds message
3. Message is sent across the network
4. Server OS hands message to server stub
5. Stub unpacks message
6. Stub makes local call to "add"
Marshalling

**MEMORY (little-endian)**
(number 5 and “JILL”)

(a)

**MEMORY (big-endian)**
(number 5x2^{24} and “JILL”)

(b)
Invert positions doesn’t work
Asynchronous RPC
Example of a-RPC
Using RPC
Run time support (in DCE)

1. Register end point
2. Register service
3. Look up server
4. Ask for end point
5. Do RPC
RPC semantic

- (a) Normal case
- (b) server crash after execution
- (c) server crash before execution
  - Maybe
  - At least once
  - At most once
  - Exactly once
Remote Service Invocation

- Allows to call a service, mostly offered by a web
- It uses the same RPC architecture
- The stub code is given by the service provider as a package, or generated from the service description
RSI (=RPC over http)

Client

Calling Service

Stub

Web Server

Called Service

Skeleton

Service definition (WSDL)

Publish

Generates

Request message (XML, SOAP, HTTP)

Reply message

Network

parameters

result

parameters

result
Remote Method Invocation

Client

OID.method(args)

Invoke(OID,method, args_in, args_out)

Proxy
(OID interface
not a comp time,
Marsh/unmarsh)

Invoke
(Dynamic Invocation interface)

Operating System

Remote OID is a parameter

Server

Oggetto OID

methods

data

Skeleton
(comp time)

(Dynamic Skeleton interface)

Operating System

Request message

Reply message

Network
RMI – parameter passing

Machine A
- Local reference L1
- Local object O1
- Client code with RMI to server at C (proxy)
- Remote invocation with L1 and R1 as parameters
- Copy of O1
- New local reference

Machine B
- Remote reference R1
- Remote object O2

Machine C
- Copy of R1 to O2
- Server code (method implementation)
Java RMI case study

- Java-RMI aims at assuring the highest degree of distribution transparency.
- A remote object is an object whose state resides on a machine, but whose interface may be invoked from other (remote) machines.
- Remote methods invocation is supported by a local proxy.
Transient communication via socket
<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_bsend</td>
<td>Append outgoing message to a local send buffer</td>
</tr>
<tr>
<td>MPI_send</td>
<td>Send a message and wait until copied to local or remote buffer</td>
</tr>
<tr>
<td>MPI_ssend</td>
<td>Send a message and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_sendrecv</td>
<td>Send a message and wait for reply</td>
</tr>
<tr>
<td>MPI_isend</td>
<td>Pass reference to outgoing message, and continue</td>
</tr>
<tr>
<td>MPI_issend</td>
<td>Pass reference to outgoing message, and wait until receipt starts</td>
</tr>
<tr>
<td>MPI_recv</td>
<td>Receive a message; block if there is none</td>
</tr>
<tr>
<td>MPI_irecv</td>
<td>Check if there is an incoming message, but do not block</td>
</tr>
</tbody>
</table>
Communication using queues

(a) Sender running
(b) Sender running
(c) Sender passive
(d) Sender passive

Receiver running
Receiver passive
Receiver running
Receiver passive

Higher loosely-coupled
## Basic interface to a queue

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put</td>
<td>Append a message to a specified queue</td>
</tr>
<tr>
<td>Get</td>
<td>Block until the specified queue is nonempty, and remove the first message</td>
</tr>
<tr>
<td>Poll</td>
<td>Check a specified queue for messages, and remove the first. Never block</td>
</tr>
<tr>
<td>Notify</td>
<td>Install a handler to be called when a message is put into the specified queue</td>
</tr>
</tbody>
</table>
General Architecture of a Message-Queuing System (1)
General Architecture of a Message-Queuing System (2)
Message broker
IBM’s WebSphere
Example of message routing
Stream communication

- Asynchronous
- Synchronous
  - Bound on the maximum delay
- Isochronous
  - Bounds on the minimum and maximum delay
- Simple
- Compound
  - Sub-streams (audio, video, title)
General architecture for stored streaming
Quality of Service

- Bit rate at which data should be transported.
- The maximum end-to-end delay (delay lag)
- The maximum delay until a session has been set up (set-up time)
- The maximum delay variance, or jitter.
- The maximum round-trip delay
Enforcing QoS

Packet departs source: 1 2 3 4 5 6 7 8

Packet arrives at buffer: 1 2 3 4 5 6 7 8

Packet removed from buffer: 1 2 3 4 5 6 7 8

Time in buffer: 0 5 10 15 20

Gap in playback:
Interleaving

(a) Gap of lost frames

(b) Lost frames
Multicast communication

• Network layer multicast (IP multicast)
• Application level multicast
  – Tree based
  – Mesh based
Tree based ALM using Chord

- To set-up a new group (tree) a node generates a random key, say mid, and sends a tree set-up request to R=succ(mid)
- R becomes the root of the tree
- To join the tree, a node P sends a join request to succ(mid).
- An intermediate node becomes a forwarder; any further request is not propagated upward
Root

Q

Forwarder

P

Lookup(mid)

Root

Q

Forwarder

P

Lookup(mid)

P'

Lookup(mid)
Quality of the tree

• Link stress
  – Given a network link, the number of times a link is used

• Stretch (or Relative Delay Penalty, RDP)
  – Given two nodes, the ratio between the transmission delay at the overlay layer and at network layer

• Cost of the tree
  – For example total delay
Example

The stress of this link is 2
IP vs Overlay multicast

Figure 2. a) IP multicasting scenario and b) application layer multicast.
Figure 4. a) A physical topology; b) IP multicast tree constructed by DVMRP; c) ALM concept; and d) End-system overlay network.
Fig. 3. Switch trees. The gray node can make any of the black nodes its new parent.
Problems with single tree

- Disruptive delivery due to failures of high-level nodes
- Out-going bandwidth of leaf nodes is not used
Multiple trees

Fig. 2. A multi-tree broadcast with two trees.

from “Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast”, Liu et al. Proceedings of the IEEE.