Distributed Systems
Communication (I)

[2] Communication (I)

1. **Layered Protocols**
2. **Remote Procedure Call**
3. **Remote Object Invocation**
4. Message-oriented Communication
5. Stream-oriented Communication

[3] Necessary Agreements

- How many volts should be used to signal a 0-bit, and how many for a 1-bit?
- How does the receiver know which is the last of the message?
- How can it detect if a message has been damaged or lost?
- How long are numbers, strings and other data items?
- How are they represented?

ISO OSI = OSI Model = Open Systems Interconnection Reference Model

Protocols: **connection-oriented** vs. **connectionless**.

protocol suite = **protocol stack** = the collection of protocols used in a particular system.

[4] Protocols (1)

Example protocol as a discussion:

A: Please, retransmit message n,

B: I already retransmitted it,

A: No, you did not,

B: Yes, I did,

A: All right, have it your way, but send it again.

[5] Protocols (2)

Protocol
A well-known set of rules and formats to be used for communication between processes in order to perform a given task.

Two important parts of the definition:

- a specification of the sequence of messages that must be exchanged,
- a specification of the format of the data in the messages.

How to create protocols:
On the Design of Application Protocols, RFC 3117,
http://www.rfc-editor.org/rfc/rfc3117.txt

- Google Protocol Buffers, SOAP, etc.

[6] Protocols (3)

"Designing Painless Protocols", ***http://nerdland.net/2009/12/designing-painless-protocols/***.

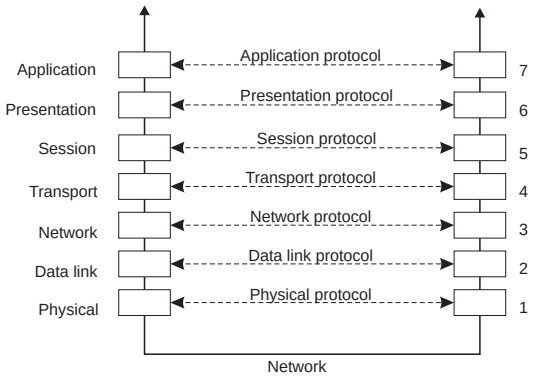
- do not re-invent the wheel
- prefer determinism
- prefer human readability
- insist on network byte ordering
- make magic numbers meaningful
- design for expansion
- do not be stingy with information
- document your protocol precisely

– follow the robustness principle

– design for security from the start

Be conservative in what you do, be liberal in what you accept from others.

[7] **Layered Protocols (1)**

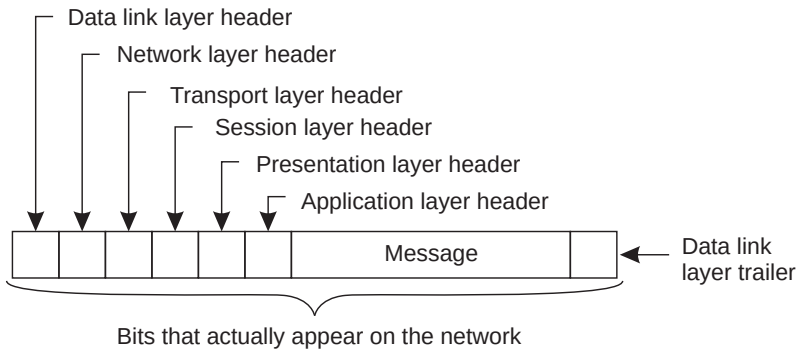


Layers, interfaces, and protocols in the OSI model.

– focus on message-passing only,

– often unneeded or unwanted functionality.

[8] **Layered Protocols (2)**



A typical message as it appears on the network.

[9] **Layered Protocols (3)**

Physical layer
Contains the specification and implementation of bits, and their transmission between sender and receiver.

Data link layer
Describes the transmission of a series of bits into a frame to allow error and flow control.

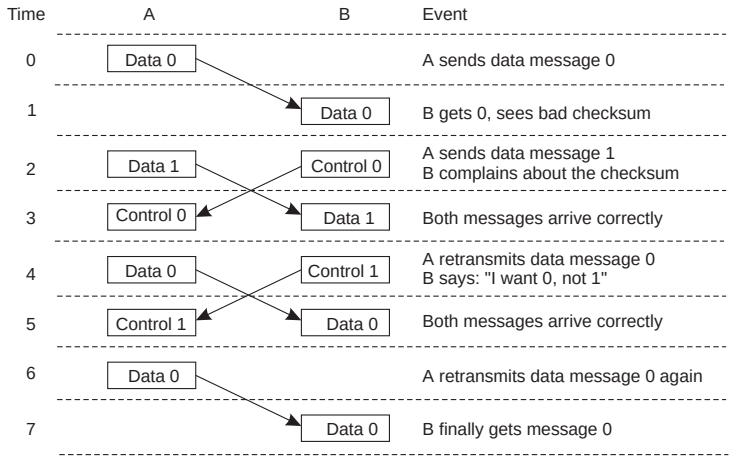
Network layer
Describes how packets in a network of computers are to be routed.

Transport Layer
Provides the actual communication facilities for most distributed systems.

Standard Internet protocols:

- TCP: connection-oriented, reliable, stream-oriented communication,
- UDP: unreliable (best-effort) datagram communication.

[10] **Data Link Layer**



Discussion between a receiver and a sender in the data link layer.

[11] **Network level protocols**

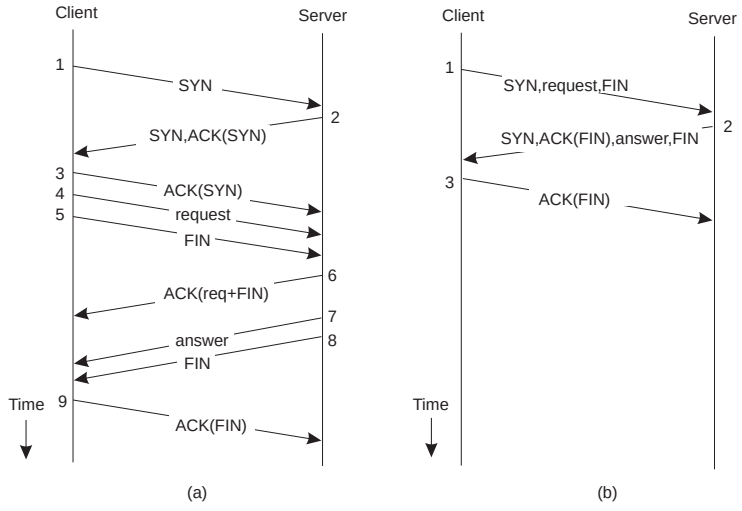
Network layer:

- IP packets
- ATM virtual channels (unidirectional connection-oriented protocol),
- collections of virtual channels grouped into virtual paths – predefined routes between pairs of hosts.

Transport layer:

- TCP, UDP
- RTP - Real-time Transport Protocol
- TP0 – TP4, the official ISO transport protocols,

[12] **Client-Server TCP**



(a) Normal operation of TCP. (b) Transactional TCP.

[13] **Networking - review**

Networking, keywords, review:

- routing in IP, default gateway,
- hardware: router, bridge, hub, switch, gateway, firewall, repeater,
- domain name resolution,
- CIDR – classless interdomain routing,
- private networks (10.x.y.z, 172.16.x.y, 192.168.x.y),
- NAT.

[14] Above the Transport Layer

Many application protocols are directly implemented on top of transport protocols, doing a lot of application-independent work.

	News	**FTP**	**WWW**
Transfer	NNTP	FTP	HTTP
Naming	Newsgroup	Host + path	URL
Distribution	Push	Pull	Pull
Replication	Flooding	Caching + DNS tricks	Caching + DNS tricks
Security	None (PGP)	Username + Password	Username + Password

[15] Middleware Protocols (1)

Middleware
An application that logically lives in the application layer, but which contains many general-purpose protocols that warrant their own layers, independent of other, more specific applications.

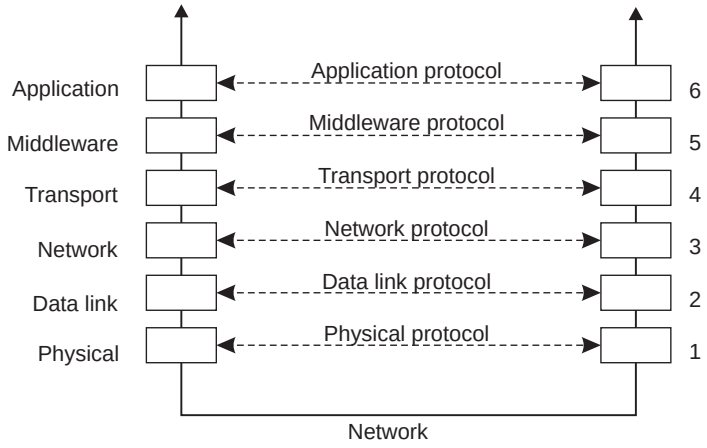
Middleware invented to provide *common* services and protocols that can be used by many *different* applications:

Example protocols:

- open communication protocols,
- marshaling and unmarshaling of data, for systems integration,
- naming protocols, for resource sharing,
- security protocols, distributed authentication and authorization,

– scaling mechanisms, support for caching and replication.

[16] **Middleware Protocols (2)**



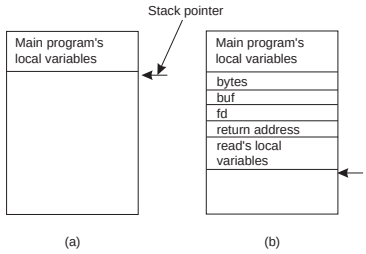
An adapted ISO OSI reference model for networked communication.

[17] **High-level Middleware Communication Services**

Some of high-level middleware protocol types:

1. remote procedure call,
2. remote object invocation,
3. message queuing services,
4. stream-oriented communication.

[18] **Local Procedure Call**



Parameter passing:

a. the stack before the call.

b. the stack while the called procedure is active.

- application developers familiar with simple procedure model,
- procedures as black boxes (isolation),
- no fundamental reason not to execute procedures on separate machine.

[19] **Remote Procedure Call**

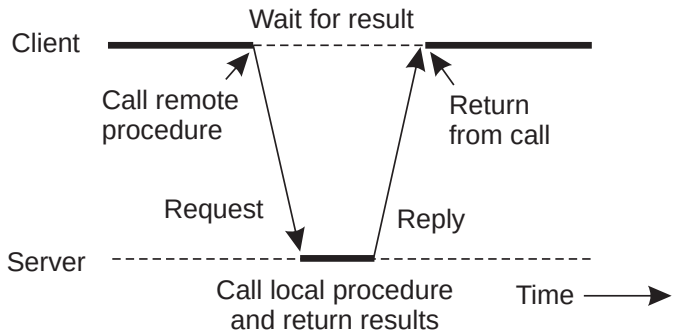
When we try to call procedures located on other machines, some subtle problems exist:

- different address spaces,
- parameters and results have to be passed,
- both machines may crash.

Standard function call parameters types:

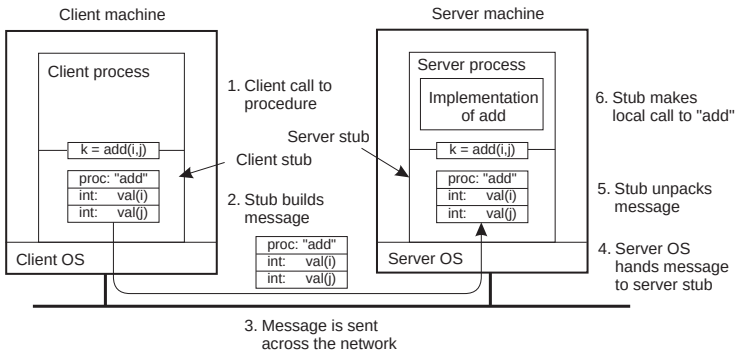
- call-by-value,
- call-by-reference,
- call by copy/restore.

[20] **Principle of RPC**



Principle of RPC between a client and server program.

[21] **Passing Value Parameters (1)**

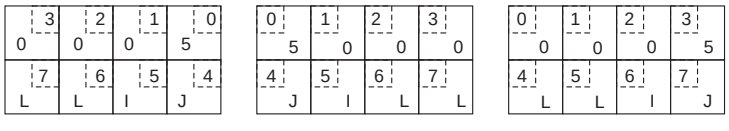


Steps involved in doing remote computation through RPC.

parameter marshaling – packing parameters into a message.

[22] **Passing Value Parameters (2)**

- IBM mainframes: **EBCDIC** character code,
- IBM personal computers: **ASCII** character code.



a. Original message on the Pentium

b. The message as being received on the SPARC

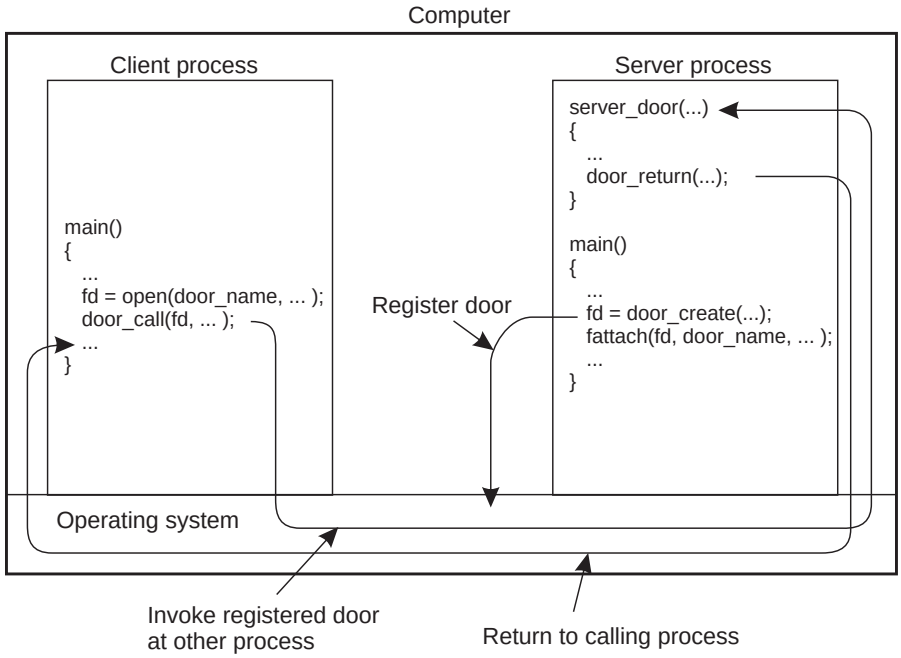
c. The message after being inverted. The little numbers in boxes indicate the address of each byte.

[23] **Extended RPC models – Doors**

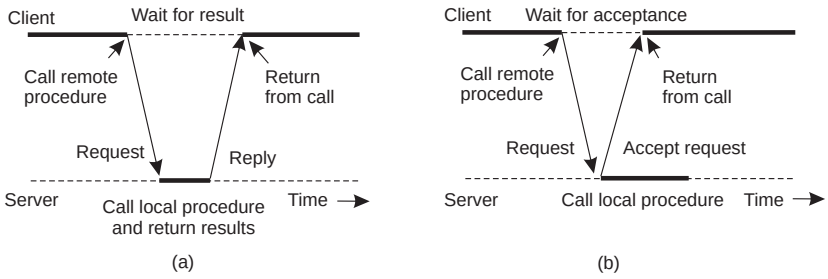
Door
A procedure in the address space of a server process that can be called by process collocated with the server.

- local IPC to be much more efficient than networking,
- door to be registered to be called (door_create),
- in Solaris each door has a file name (fattach),
- calling doors by door_call (OS makes an upcall),
- result returned to the client through door_return.
- benefit: single mechanism, procedure calls, for effective communication in a distributed system,
- drawbacks: still the need to distinguish standard procedure calls, calls to other local processes, calls to remote processes.

[24] **Doors - how to use**



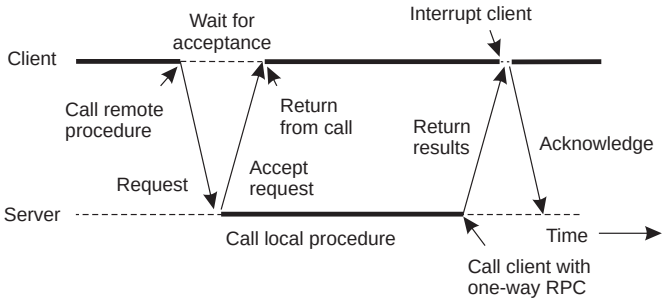
[25] **Asynchronous RPC (1)**



a. The interconnection between client and server in a traditional RPC.

b. The interaction using asynchronous RPC.

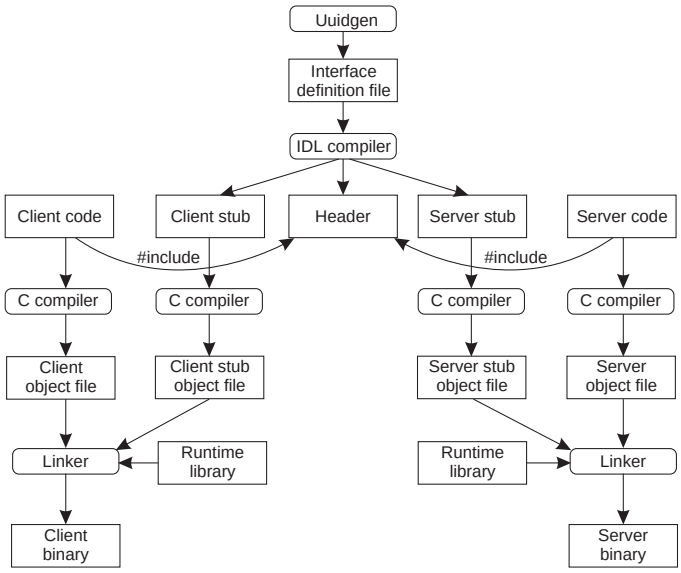
[26] **Asynchronous RPC (2)**



deferred synchronous RPC – asynchronous RPC with second call done by the server,

one-way RPC – client does not wait for acceptance of the request , problem with reliability.

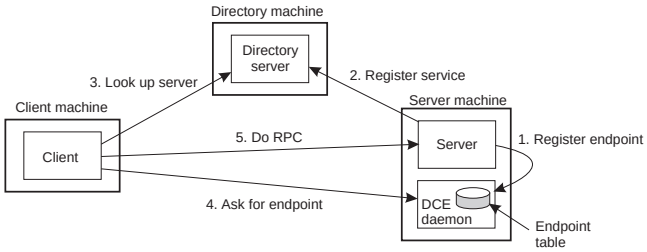
[27] **Writing a Client and a Server**



Steps in writing a client and a server in DCE RPC. Let the developer concentrate only on the client- and server-specific code. Leave the rest for RPC generators and libraries.

[28] **Binding a Client to a Server**

Client must locate server machine, and locate the server.



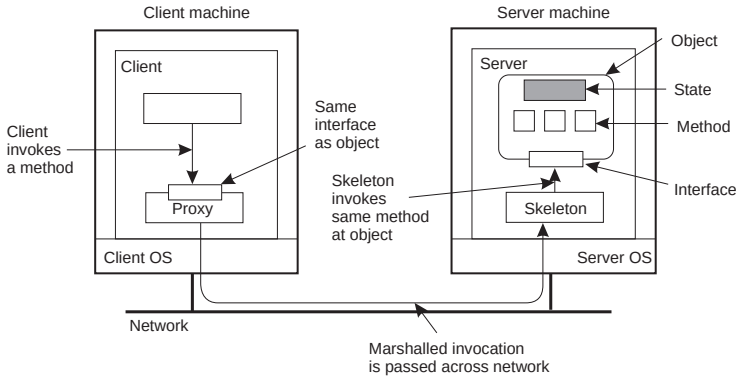
Client-to-server binding in DCE – separate daemon for each server machine.

[29] **Remote Distributed Objects (1)**

The basic idea of remote objects:

- object: state and interface, methods and attributes, invocation and implementation,
- data and operations **encapsulated** in an object,
- operations implemented as **methods**, and accessible through **interfaces**,
- object offers only its **interface** to clients,
- **object server** is responsible for a collection of objects,
- **client stub (proxy)** implements interface,
- **server skeleton** handles (un)marshaling and object invocation.

[30] **Remote Distributed Objects (2)**



Common organization of a remote object with client-side proxy.

[31] (Remote) Distributed Objects (3)

Compile-time objects
Language-level objects, from which proxy and skeletons are automatically generated.

Runtime objects
Can be implemented in any language, but require use of an **object adapter** that makes the implementation appear as an object.

Transient object lives only by virtue of a server: if the server exits, so will the object.

Persistent object lives independently from a server: if a server exits, the object's state and code remain (passively) on disk.

[32] Binding a Client to an Object (1)

Having an object reference allows a client to **bind** to an object:

- reference denotes server, object, and communication protocol,
- client loads associated stub code,
- stub is instantiated and initialized for specific object.

Remote-object references enable passing references as parameters, what was hardly possible with ordinary RPCs.

Two ways of binding:

Implicit: invoke methods directly on the referenced object.

Explicit: client must first explicitly bind to object before invoking it.

[33] **Binding a Client to an Object (2)**

```
Distr_object* obj_ref;       // Declare a systemwide object reference
obj_ref = ...;               // Initialize the reference to a distrib. obj.
obj_ref→do_something( );// Implicitly bind and invoke a method
```

(a)

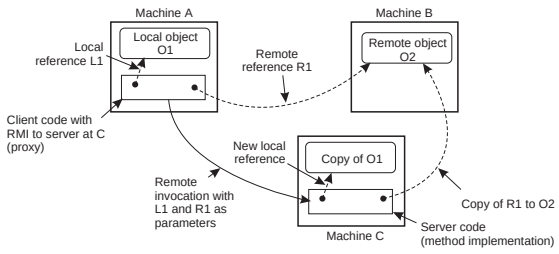
```
Distr_object obj_ref;        // Declare a systemwide object reference
Local_object* obj_ptr;       // Declare a pointer to local objects
obj_ref = ...;               // Initialize the reference to a distrib. obj.
obj_ptr = bind(obj_ref);     // Explicitly bind and get ptr to local proxy
obj_ptr→do_something( );// Invoke a method on the local proxy
```

(b)

a. Example with implicit binding using only global references.

b. Example with explicit binding using global and local references.

[34] **RMI - Parameter Passing**



Objects sometimes passed by reference, but sometimes by value.

- a client running on machine A, a server on machine C,
- the client calls the server with two references as parameters, O1 and O2, to local and remote objects,
- copying of an object as a possible side effect of invoking a method with an object reference as a parameter (transparency versus efficiency).