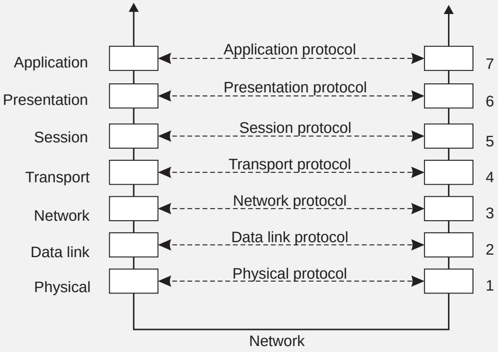
Layered Protocols

- Low-level layers
- Transport layer
- Application layer
- Middleware layer

Basic networking model



Drawbacks

- Focus on message-passing only
- Often unneeded or unwanted functionality
- Violates access transparency

Low-level layers

Recap

- Physical layer: contains the specification and implementation of bits, and their transmission between sender and receiver
- Data link layer: prescribes the transmission of a series of bits into a frame to allow for error and flow control
- Network layer: describes how packets in a network of computers are to be routed.

Observation

For many distributed systems, the lowest-level interface is that of the network layer.

Transport Layer

Important

The 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

Note

IP multicasting is often considered a standard available service (which may be dangerous to assume).

Middleware Layer

Observation

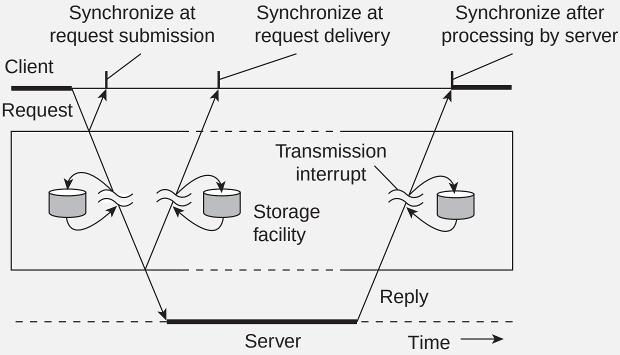
Middleware is invented to provide common services and protocols that can be used by many different applications

- A rich set of communication protocols
- (Un)marshaling of data, necessary for integrated systems
- Naming protocols, to allow easy sharing of resources
- Security protocols for secure communication
- Scaling mechanisms, such as for replication and caching

Note

What remains are truly application-specific protocols...
such as?

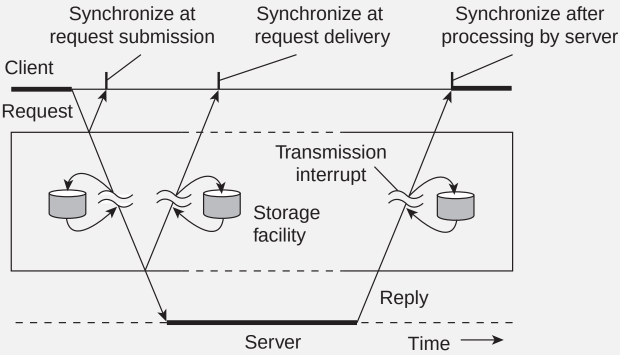
Types of communication



Distinguish

- Transient versus persistent communication
- Asynchrounous versus synchronous communication

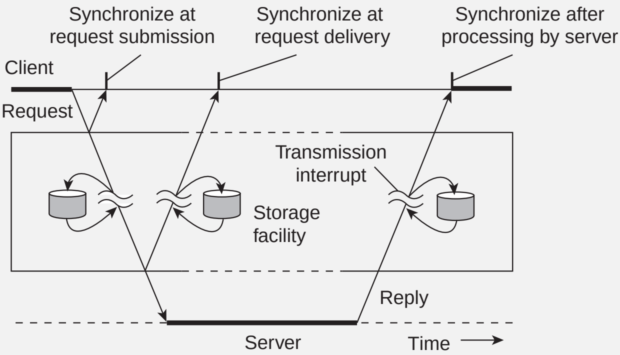
Types of communication



Transient versus persistent

- Transient communication: Comm. server discards message when it cannot be delivered at the next server, or at the receiver.
- Persistent communication: A message is stored at a communication server as long as it takes to deliver it.

Types of communication



Places for synchronization

- At request submission
- At request delivery
- After request processing

Client/Server

Some observations

Client/Server computing is generally based on a model of transient synchronous communication:

- Client and server have to be active at time of commun.
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

Drawbacks synchronous communication

- Client cannot do any other work while waiting for reply
- Failures have to be handled immediately: the client is waiting
- The model may simply not be appropriate (mail, news)

Client/Server

Some observations

Client/Server computing is generally based on a model of transient synchronous communication:

- Client and server have to be active at time of commun.
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

Drawbacks synchronous communication

- Client cannot do any other work while waiting for reply
- Failures have to be handled immediately: the client is waiting
- The model may simply not be appropriate (mail, news)

Messaging

Message-oriented middleware

Aims at high-level persistent asynchronous communication:

- Processes send each other messages, which are queued
- Sender need not wait for immediate reply, but can do other things
- Middleware often ensures fault tolerance

Remote Procedure Call (RPC)

- Basic RPC operation
- Parameter passing
- Variations

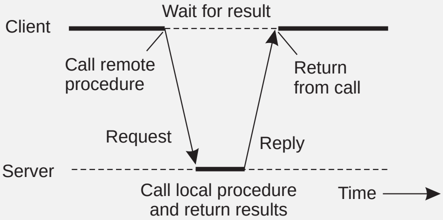
Basic RPC operation

Observations

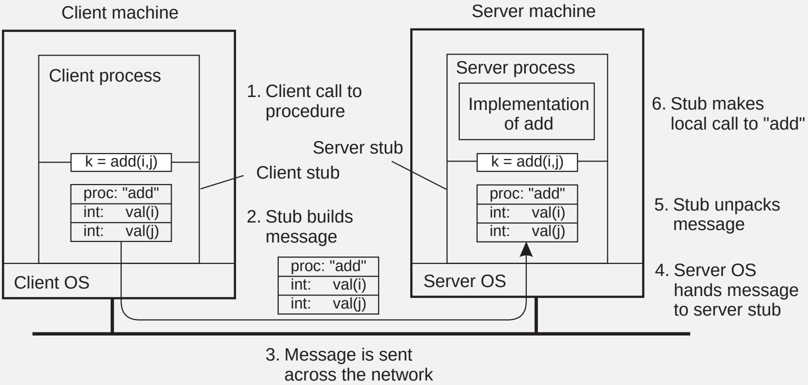
- Application developers are familiar with simple procedure model
- Well-engineered procedures operate in isolation (black box)
- There is no fundamental reason not to execute procedures on separate machine

Conclusion

Communication between caller & callee can be hidden by using procedure-call mechanism.



Basic RPC operation



1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters and calls server.
6. Server makes local call and returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client's OS.
9. Client's OS gives message to stub.
10. Client stub unpacks result and returns to the client.

RPC: Parameter passing

Parameter marshaling

There's more than just wrapping parameters into a message:

- Client and server machines may have different data representations (think of byte ordering)
- Wrapping a parameter means transforming a value into a sequence of bytes
- Client and server have to agree on the same encoding:
  - How are basic data values represented (integers, floats, characters)
  - How are complex data values represented (arrays, unions)
- Client and server need to properly interpret messages, transforming them into machine-dependent representations.

RPC: Parameter passing

RPC parameter passing: some assumptions

- Copy in/copy out semantics: while procedure is executed, nothing can be assumed about parameter values.
- All data that is to be operated on is passed by parameters. Excludes passing references to (global) data.

Conclusion

Full access transparency cannot be realized.

Observation

A remote reference mechanism enhances access transparency:

- Remote reference offers unified access to remote data
- Remote references can be passed as parameter in RPCs

RPC: Parameter passing

RPC parameter passing: some assumptions

- Copy in/copy out semantics: while procedure is executed, nothing can be assumed about parameter values.
- All data that is to be operated on is passed by parameters. Excludes passing references to (global) data.

Conclusion

Full access transparency cannot be realized.

Observation

A remote reference mechanism enhances access transparency:

- Remote reference offers unified access to remote data
- Remote references can be passed as parameter in RPCs

RPC: Parameter passing

RPC parameter passing: some assumptions

- Copy in/copy out semantics: while procedure is executed, nothing can be assumed about parameter values.
- All data that is to be operated on is passed by parameters. Excludes passing references to (global) data.

Conclusion

Full access transparency cannot be realized.

Observation

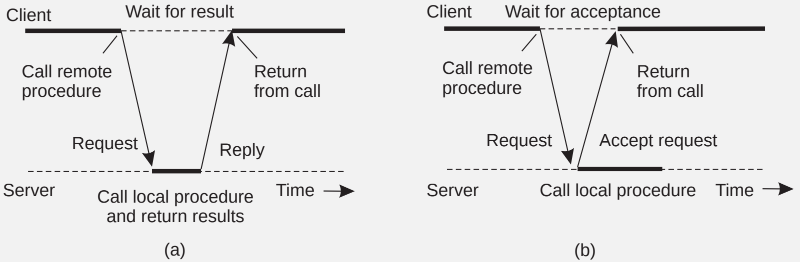
A remote reference mechanism enhances access transparency:

- Remote reference offers unified access to remote data
- Remote references can be passed as parameter in RPCs

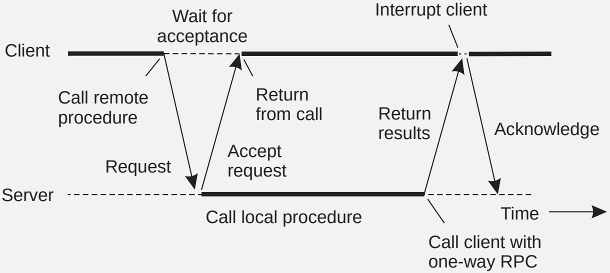
Asynchronous RPCs

Essence

Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server.



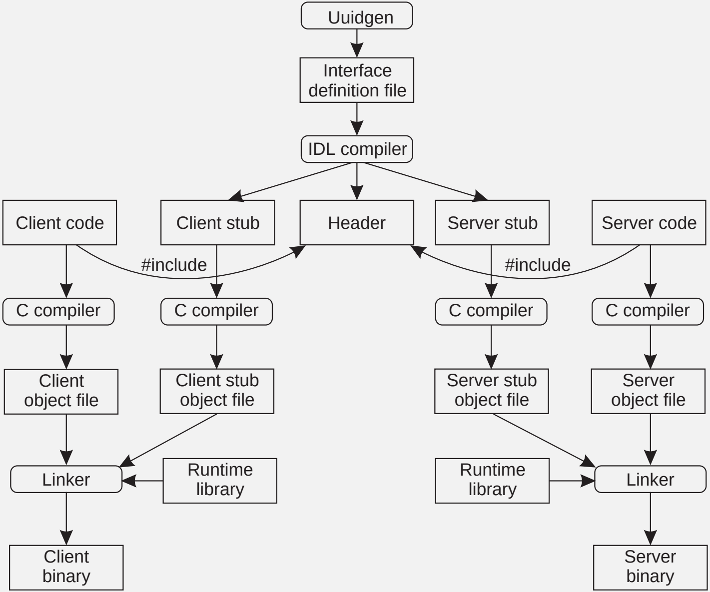
Deferred synchronous RPCs



Variation

Client can also do a (non)blocking poll at the server to see whether results are available.

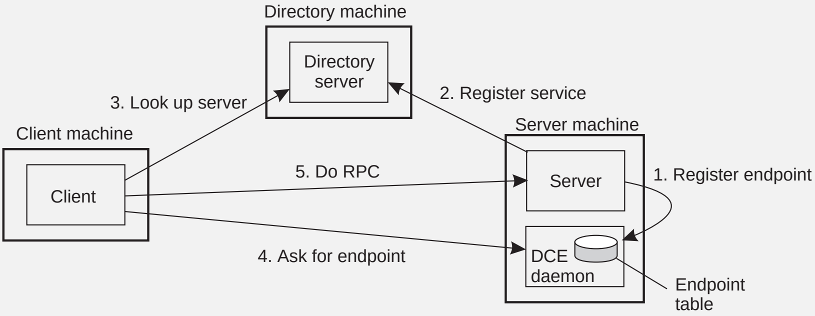
RPC in practice



Client-to-server binding (DCE)

Issues

(1) Client must locate server machine, and (2) locate the server.



Stream-oriented communication

- Support for continuous media
- Streams in distributed systems
- Stream management

Continuous media

Observation

All communication facilities discussed so far are essentially based on a discrete, that is time-independent exchange of information

Continuous media

Characterized by the fact that values are time dependent:

- Audio
- Video
- Animations
- Sensor data (temperature, pressure, etc.)

Continuous media

Transmission modes

Different timing guarantees with respect to data transfer:

- Asynchronous: no restrictions with respect to **when** data is to be delivered
- Synchronous: define a maximum end-to-end delay for individual data packets
- Isochronous: define a maximum and minimum end-to-end delay (jitter is bounded)

Stream

Definition

A (continuous) data stream is a connection-oriented communication facility that supports isochronous data transmission.

Some common stream characteristics

- Streams are unidirectional
- There is generally a single source, and one or more sinks
- Often, either the sink and/or source is a wrapper around hardware (e.g., camera, CD device, TV monitor)
- Simple stream: a single flow of data, e.g., audio or video
- Complex stream: multiple data flows, e.g., stereo audio or combination audio/video

Streams and QoS

Essence

Streams are all about timely delivery of data. How do you specify this Quality of Service (QoS)? Basics:

- The required bit rate at which data should be transported.
- The maximum delay until a session has been set up (i.e., when an application can start sending data).
- The maximum end-to-end delay (i.e., how long it will take until a data unit makes it to a recipient).
- The maximum delay variance, or jitter.
- The maximum round-trip delay.

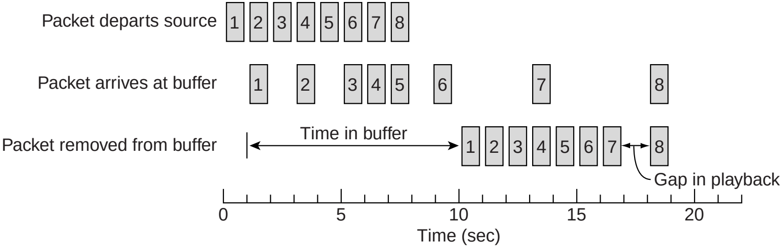
Enforcing QoS

Observation

There are various network-level tools, such as differentiated services by which certain packets can be prioritized.

Also

Use buffers to reduce jitter:

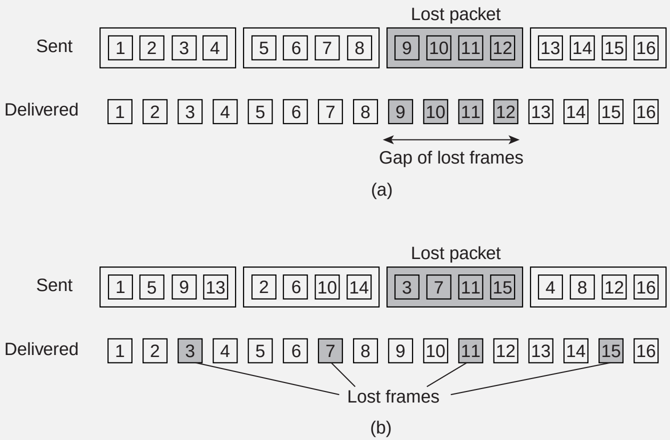


Enforcing QoS

Problem

How to reduce the effects of packet loss (when multiple samples are in a single packet)?

Enforcing QoS



Stream synchronization

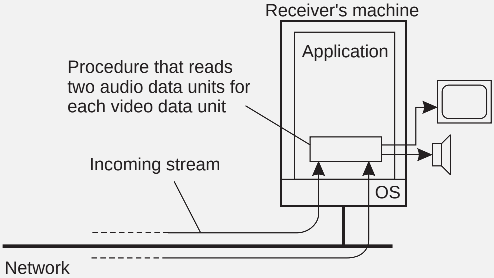
Problem

Given a complex stream, how do you keep the different substreams in synch?

Example

Think of playing out two channels, that together form stereo sound. Difference should be less than 20–30 μsec!

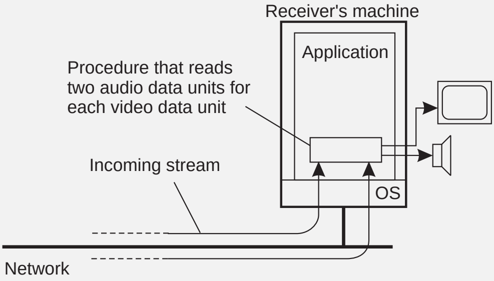
Stream synchronization



Alternative

Multiplex all substreams into a single stream, and demultiplex at the receiver. Synchronization is handled at multiplexing/demultiplexing point (MPEG).

Stream synchronization



Alternative

Multiplex all substreams into a single stream, and demultiplex at the receiver. Synchronization is handled at multiplexing/demultiplexing point (MPEG).