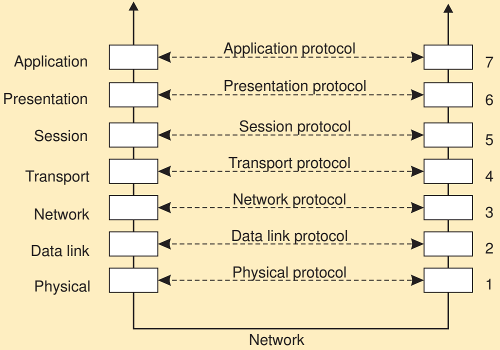
Distributed Systems

(3rd Edition)

Chapter 04: Communication

Version: February 25, 2017

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

Middleware layer

Observation

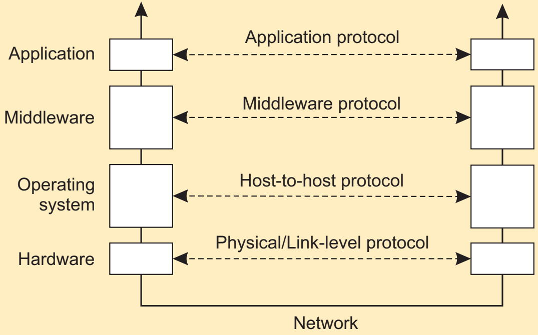
Middleware is invented to provide common services and protocols that can be used by many different applications

- 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

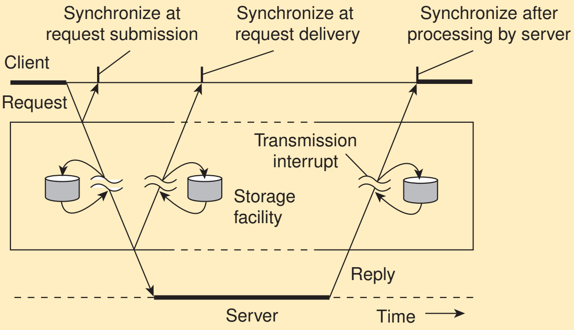
What remains are truly application-specific protocols... such as?

An adapted layering scheme



Types of communication

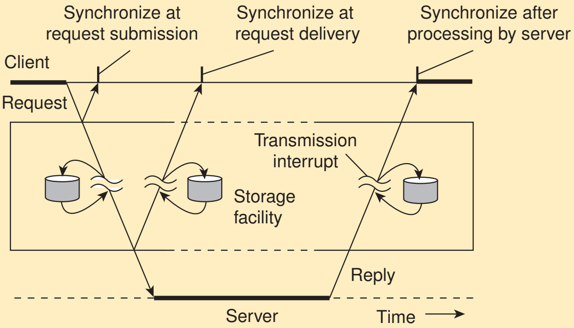
Distinguish...



- Transient versus persistent communication
- Asynchronous versus synchronous communication

Types of communication

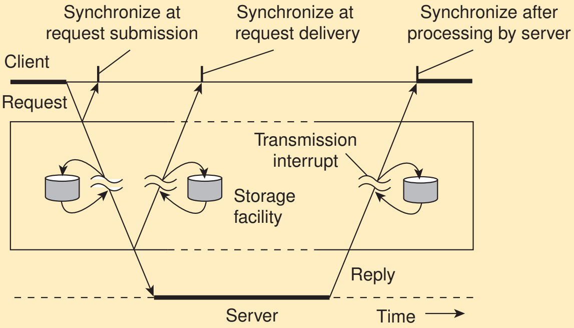
Transient versus persistent



- 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

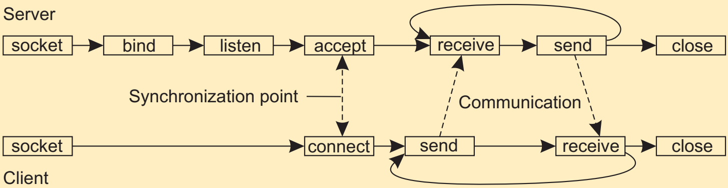


- At request submission
- At request delivery
- After request processing

Transient messaging: sockets

Berkeley socket interface

Operation	Description
socket	Create a new communication end point
bind	Attach a local address to a socket
listen	Tell operating system what the maximum number of pending connection requests should be
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
send	Send some data over the connection
receive	Receive some data over the connection
close	Release the connection



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 communication
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

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 communication
- 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)

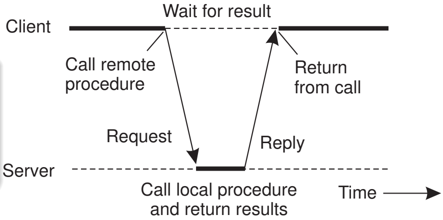
Basic RPC operation

Observations

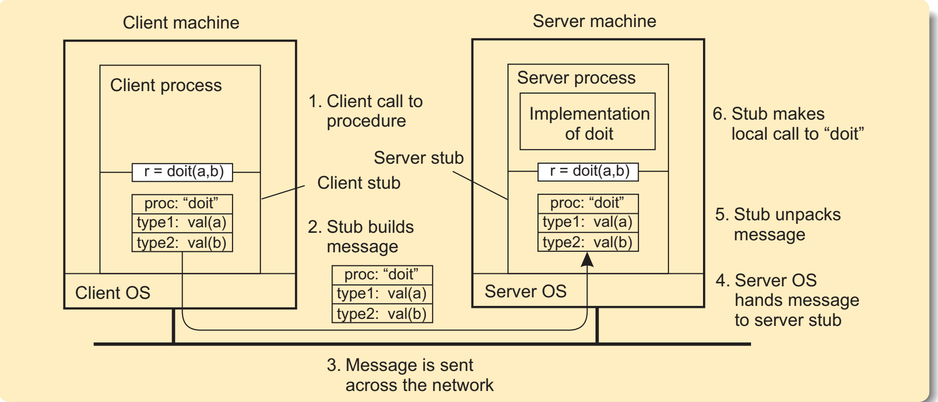
- Application developers are familiar with simple procedure model
- Well-engineered procedures operate in isolation (black box)
- There is no fundamental reason not to execute procedures on separate machine

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; calls server.
6. Server does local call; 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; returns to client.

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.

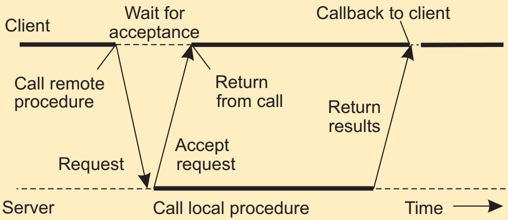
A remote reference mechanism enhances access transparency

- Remote reference offers unified access to remote data
- Remote references can be passed as parameter in RPCs
- Note: stubs can sometimes be used as such references

Asynchronous RPCs

Essence

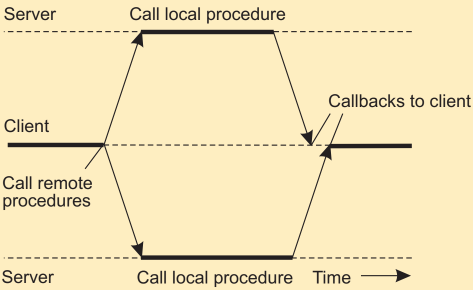
Try to get rid of the strict request-reply behavior, but let the client continue without waiting for an answer from the server.



Sending out multiple RPCs

Essence

Sending an RPC request to a group of servers.



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

Message-oriented middleware

Essence

Asynchronous persistent communication through support of middleware-level queues. Queues correspond to buffers at communication servers.

Operations

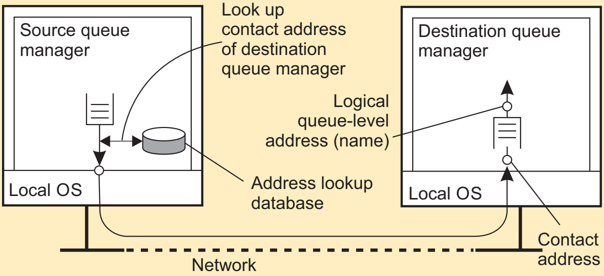
Operation	**Description**
`put`	Append a message to a specified queue
`get`	Block until the specified queue is nonempty, and remove the first message
`poll`	Check a specified queue for messages, and remove the first. Never block
`notify`	Install a handler to be called when a message is put into the specified queue

General model

Queue managers

Queues are managed by queue managers. An application can put messages only into a local queue. Getting a message is possible by extracting it from a local queue only ⇒ queue managers need to route messages.

Routing



Message broker

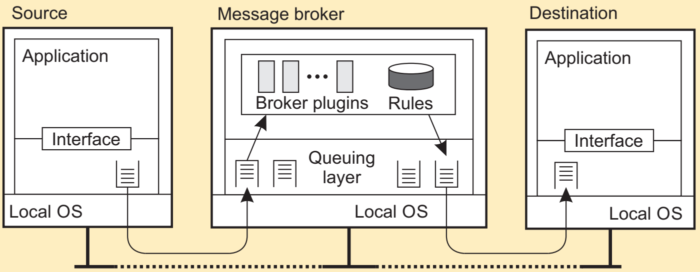
Observation

Message queuing systems assume a common messaging protocol: all applications agree on message format (i.e., structure and data representation)

Broker handles application heterogeneity in an MQ system

- Transforms incoming messages to target format
- Very often acts as an application gateway
- May provide subject-based routing capabilities (i.e., publish-subscribe capabilities)

Message broker: general architecture



Application-level multicasting

Essence

Organize nodes of a distributed system into an overlay network and use that network to disseminate data:

- Oftentimes a tree, leading to unique paths
- Alternatively, also mesh networks, requiring a form of routing

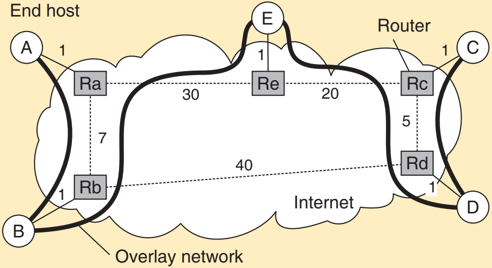
Application-level multicasting in Chord

Basic approach

1. Initiator generates a multicast identifier *mid*.
2. Lookup *succ*(*mid*), the node responsible for *mid*.
3. Request is routed to *succ*(*mid*), which will become the root.
4. If *P* wants to join, it sends a join request to the root.
5. When request arrives at *Q*:
   - *Q* has not seen a join request before $\Rightarrow$ it becomes forwarder; *P* becomes child of *Q*. Join request continues to be forwarded.
   - *Q* knows about tree $\Rightarrow$ *P* becomes child of *Q*. No need to forward join request anymore.

ALM: Some costs

Different metrics



- Link stress: How often does an ALM message cross the same physical link? Example: message from A to D needs to cross $\langle Ra, Rb \rangle$ twice.
- Stretch: Ratio in delay between ALM-level path and network-level path. Example: messages B to C follow path of length 73 at ALM, but 47 at network level $\Rightarrow$ stretch = 73/47.

Flooding

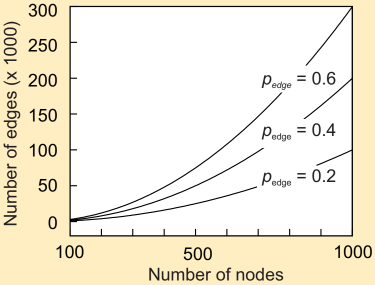
Essence

P simply sends a message *m* to each of its neighbors. Each neighbor will forward that message, except to *P*, and only if it had not seen *m* before.

Performance

The more edges, the more expensive!

The size of a random overlay as function of the number of nodes



Flooding

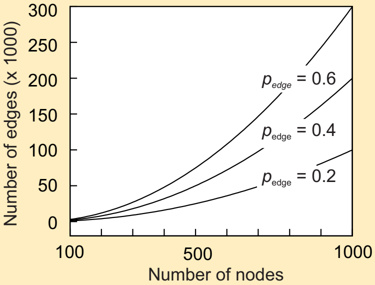
Essence

P simply sends a message m to each of its neighbors. Each neighbor will forward that message, except to P, and only if it had not seen m before.

Performance

The more edges, the more expensive!

The size of a random overlay as function of the number of nodes



Variation

Let Q forward a message with a certain probability p_{flood}, possibly even dependent on its own number of neighbors (i.e., node degree) or the degree of its neighbors.

Epidemic protocols

Assume there are no write–write conflicts

- Update operations are performed at a single server
- A replica passes updated state to only a few neighbors
- Update propagation is lazy, i.e., not immediate
- Eventually, each update should reach every replica

Two forms of epidemics

- Anti-entropy: Each replica regularly chooses another replica at random, and exchanges state differences, leading to identical states at both afterwards
- Rumor spreading: A replica which has just been updated (i.e., has been contaminated), tells a number of other replicas about its update (contaminating them as well).