# Communication Issues

## Networking Types and Protocols

Types of networks with impact on distributed systems:

LANs, MANs (DSL, cable modem), WANs, wireless. See Fig 3.1 CDK

OSI protocol stack—well defined. In reality we have only some of the layers.

Internet protocols

## Networking Issues for Distributed Systems

- performance—latency and data transfer rate (throughput)
- scalability
- reliability—high for physical networks, with most errors due to software or buffer overflow
- security—common use of a firewall to filter messages, encryption, Virtual Private Network (VPN) does encryption
- mobility—big and growing issue
- quality of service—applications have different demands, no longer just "best effort"
- multicasting—send to a group. Underlying support from the network?

# Middleware Layers

Above operating system and below applications. See figure.

# How to Communicate?

## The Client-Server Model

### Addressing

How to locate?

1. machine and port
2. broadcast/multicast (LAN vs. WAN)
3. name servers

### Blocking vs. Nonblocking Send

See Fig 10-8

- blocking send (synchronous)
- nonblocking send with copy
- nonblocking send with interrupt (no copying of buffer). Sender must know the receiver has gotten the message before reusing the buffer.

Other issues in Fig 10-11, 10-12, and 10-13.

## Remote Procedure Call

Look at Birrell and Nelson paper.

Issues in duplicating procedure-call semantics:

- call-by-value vs. call-by-reference
- parameter passing—byte ordering

### RPC failures

1. client unable to locate the server
2. request message from client to server is lost (resend?)
3. reply message from the server to the client is lost. Is the operation idempotent—does it have side-effects?
   - at least once semantics (rebind to server if need be)
   - "maybe" semantics (do not retry if there is a problem)
   - at most once semantics (retry if a problem, but have ability to filter duplicates)
   - exactly once (difficult on client or server failure)

   What about B&N?
4. server crashes after receiving a request
5. client crashes after sending a request

### Critical Path Performance

RPC on a DEC Firefly multiprocessor workstation.

See Fig 10-24. UDP protocol. Buffering in shared buffers between user and kernel space (VAX architecture). Assembly language code and hand optimized.

Experience:

- use normal hardware
- UDP overhead of checksums
- context switching to user space expensive

### Other RPC Mechanisms and Data Representations

Sun and XDR.

JavaRMI and Java Object serialization.

Web services (SOAP) and XML—text/tags based.

### Asynchronous RPC

Similar issues to message passing. Used by X11.

1. Can send requests needing no replies
2. can bundle RPC requests
3. client can continue processing and pick up the reply later

What's the difference between RPC and message passing?

send/receive of messages (I/O based) versus RPC (procedure oriented)

# Group Communication

Three types of communication:

1. unicast—point-to-point
2. broadcast—one-to-all
3. multicast—one-to-some (group)

Multicast is the most general and can subsume the other two. How is it supported?

- multiple unicasts
- broadcast with each machine filtering
- hardware directly (Ethernet has $2^{23}$ multicast addresses)

## Design Issues

- Closed versus Open Groups—can a nonmember send to the group?
- Peer groups versus central coordinator (may have a hybrid where one member of a peer group temporarily takes over coordination)
- Group membership—joining and leaving a group. Central vs. distributed.
- Group addressing—distributed game (temporary addressing). Set of name servers (well-known group address).

  Predicate addressing. A predicate is evaluated by the receiver on whether or not it should actually receive the message. Compare to my work of using the query to actually compute a multicast address.
- Send/Receive Primitives—RPC does not work so naturally. How to deal with multiple replies. May not know how many replies.
- Reliability

  Atomicity/atomic broadcast—reliability in that the message gets to all members of the group or none.

  Message Ordering—all nodes see messages in the same order.
- Overlapping groups—synchronization between groups
- Scalability—depend on a single LAN for example.

## ISIS

System at Cornell. Toolkit for building distributed applications.

*loosely synchronous system*—all events appear in the same order to all parties.

*causal relationship* between two events if the first might influence the second. In a *virtually synchronous system* all messages that are causally related must be received in the correct order.

Three primitives:

1. ABCAST—loosely synchronous communication
2. CBCAST—virtually synchronous communication
3. GBCAST—like ABCAST, but deals with group membership changes

CBCAST uses a vector approach. Look at Fig 10-35. Allows a message to be received if:

- The vector component of the sender is exactly one more than the recipient (no missed messages)
- The other vector components of the sender are no greater than any components of the recipient.

Amoeba also has group communication. Uses a *sequencer* node through which all communication goes through.

## Operating System Notification Mechanism

Look at "A Scalable and Explicit Event Delivery Mechanism for UNIX" paper. Slides: `http://www.cs.wpi.edu/~cs535/f05/banga:usenix99/`

Look at "accept()able Strategies for Improving Web Server Performance" paper. Slides: `http://www.cs.wpi.edu/~cs535/f05/brecht:usenix04.pdf`

## Distributed Events

Recipient is *notified* of an event, often in *asynchronous* fashion—versus typical *synchronous* semantics of message passing.

Use a *publish-subscribe* model where parties interest in events *subscribe* to a service provided by a *publisher*.

Examples:

- "important news story" or "weather alert" services.
- SNMP traps
- stock quote subscriptions

Each event has attributes that can be used to filter which events are used by subscriber in signing up and publisher and delivering events.

Delivery semantics—is reliable delivery of events important? It depends. If so then need a mechanism such as reliable multicast protocol.

RSS (Really Simple Syndication) feeds. Use a lightweight XML format. A means for distributing content such as news. *Aggregators* collect news from various RSS feeds and provides it to the reader. Somewhat like gathering events, but feeds are not published. Aggregator must request them. RSS provides a common format.

## IMPACT

Current project with to determine "events" (such as obtained from RSS) that "impact" a user.

Design and build a context-aware computing system, the Integrated Management of a Personal Augmented Calendar Tool (IMPACT).

Build on a user's plan to prioritize, organize and deliver any and all information that will impact the activities of that user.

Query existing information sources such as traffic and weather along with user sources such as email.

Determine the level and urgency of impact so that it can be delivered to the user at times and via means designed to minimize disruption.