Group Communication

- communicate to a group of processes rather than point-to-point
- uses
  - replicated service
  - efficient dissemination of "news"
  - building block for protocols
- variety of specifications
  - trade off speed and complexity vs. ease of use

Example

- replicated file server
- N copies of file server
  - N may vary over time
- send commands to all servers
- all servers execute commands
- want to keep all servers in identical states
- result: can survive loss of a server

Replicated Servers

- to keep replicas in same state:
  - make sure that same sequence of commands leads to same state
    - » no dependence on time or randomness
  - make sure that all replicas get the same sequence of commands
  - after crash and reboot, bring rebooted server up to date by replaying missed commands

Groups

- a group is any set of processes that want to cooperate
- processes can join or leave groups, either by explicit action or implicitly
- a process may belong to several groups
- basic operation of group communication: multicast a message to an entire group
- group name provides a useful level of indirection

Closed vs. Open Groups

- open group: anybody may send a message to the group
- closed group: only a member may send messages to the group
  - security implications
  - usually more efficient, since all senders know who is in the group
- master-slave group: only the "master process" can send to the group

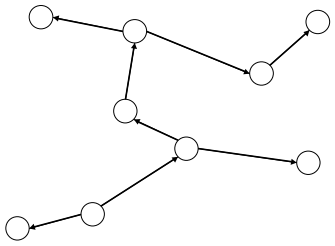
Best-Effort Group Comm

- simplest approach
  - but provides least value to programmer
- system tries to deliver each message to each member
- on one LAN, can use LAN's multicast support (if any)
- on Internet, need to do more

Simple Multicast

- approach 1: send separate copy of message to each destination
  - simple, but inefficient
- approach 2: recipients form logical tree
  - use flooding on the tree edges
- approach 3: include intermediate routers on tree
  - complex protocols for deciding how to do this

Multicast Trees



Best-Effort Multicast: Drawbacks

- not reliable, some members might miss some messages
- different members see different messages arriving in different orders
  - causes trouble for replicated services
    - » example: replicate file servers with “create foo” and “delete foo” messages
- different members may see different group membership at the same time

Improvements

- reliable message delivery
- atomic messages
- message ordering guarantees
  - within group
  - across groups
- membership agreement guarantees
  - usually, ordered with respect to messages

Membership

- central membership server
  - send message to this server to join or leave
  - membership server sends message to group to announce changes in membership
- distributed membership management
  - send message to group to join
  - send message to group to leave

Membership and Crashes

- crashed process must be removed from group somehow
  - can’t rely on it to announce its departure
- with central membership server:
  - central server pings all members
  - announces death of member
- with distributed algorithm
  - hosts ping each other
  - voting or buddy system

Atomic Messages

- atomicity property: a message is delivered to all members, or to none
- first try
  - each recipient acknowledges message
  - sender retransmits if ack not received
  - doesn't work: sender could crash before message is delivered everywhere

Atomic Messages

- fix: if sender crashes, a recipient volunteers to be "backup sender" for the message
  - re-sends message to everybody, waits for acks
  - use simple algorithm to choose volunteer
  - apply method again if backup sender crashes
- must remember all received messages, in case we need to become backup sender
  - periodic protocol to "purge" old messages

Message Ordering

- so far: different members may see messages in different orders
- ordered group communication requires that all members agree about the order of messages
- within each group, assign global ordering to messages
- hold back messages that arrive out of order

Holding Back Messages

- messages can arrive at group comm code out of order
- messages must be delivered to the application in order
- group comm code "holds back" delivery of message until it is *certain* that no earlier message can arrive
  - details depend on ordering method

Ordering: First Approach

- central ordering server assigns global sequence numbers
- hosts apply to ordering server for numbers, or ordering server sends all messages itself
- tricky protocol to deal with case where ordering server fails
  - leader election; topic for a later lecture
- hold-back easy, since sequence numbers are sequential

Ordering: Second Approach

- use time message was sent
  - measured on sending host
  - use host address to break ties
- advantage
  - simple and decentralized
- disadvantage
  - requires nearly synchronized clocks
  - must hold back messages for period equal to maximum clock difference

Atomicity, Ordering, and Failure

- suppose A is broadcasting a message, and B fails before getting the message
- atomicity says that message must be delivered to all members, but it can't be delivered to B
  - solution: remove B from group
- removing B requires a message
- removal message must precede A's message

Interactions Between Groups

- processes can belong to multiple groups
- groups can overlap
- are there inter-group message ordering guarantees?
- example
  - NFS-server group for replicated NFS service
  - SMB-server group for replicated PC file service
  - some servers belong to both groups

Inter-Group Ordering

- total ordering: all messages can be globally ordered, even across groups
- causal ordering: message delivery order respects the "happened before" relation
- sync ordering: mark some messages as "sync messages"; total order required, except that two non-sync messages can occur in either order
- unordered: no guarantees

Performance

- providing strong guarantees and good performance at the same time is hard
- lots of research on this
  - acknowledgement and negative-ack strategies
  - message storage and purging algorithms
  - leader election
  - optimized timestamp management

Isis

- Isis is the best example of this technology
  - originally developed at Cornell
  - now a successful commercial product
  - frequently used on Wall Street
- the bottom line
  - inefficient compared to build-your-own protocols
  - but simplifies programmer's life a lot
    - » faster development
    - » fewer bugs