

Advanced Distributed Systems

Karl M. Göschka
Karl.Goeschka@tuwien.ac.at

<http://www.infosys.tuwien.ac.at/teaching/courses/AdvancedDistributedSystems/>

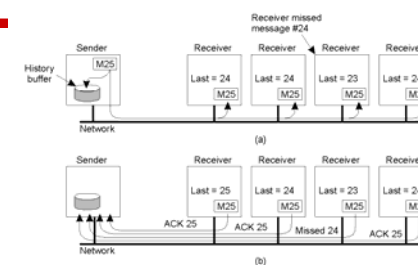
Reliable multicast

- Multicast is an essential element of many distributed algorithms
- Example: process groups, active replication
- A reliable multicast (group communication) is necessary for providing fault-tolerant distributed algorithms
- Group membership:
 - static: processes do not fail, join, leave
 - dynamic: reliable = delivery to all non-faulty group members, but agreement is needed, what the group currently looks like when a message is to be delivered

Group Communication

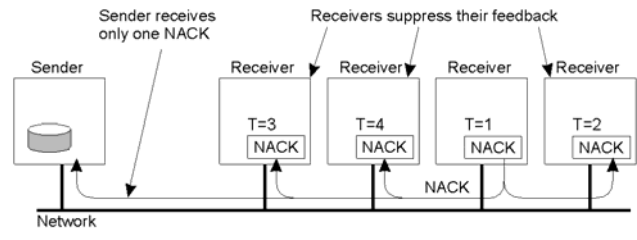
- Reliable multicast and failure atomicity
- Multicast ordering
- Static and dynamic group membership
- Atomic multicast and virtual synchrony

Basic Reliable-Multicasting Schemes



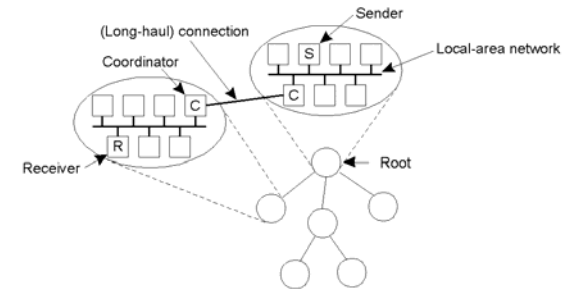
- A simple solution to reliable multicasting when all receivers are known and are assumed not to fail
 - Message transmission (multicast) and re-transmission (point-to-point)
 - Reporting feedback (piggybacked **ACK** and point-to-point **NACK**)
- Relies on message numbers; history buffer, unreliable multicast, non-faulty processes
- Scalability: **feedback implosion** vs. indefinite communication; processes retain copies of delivered messages indefinitely

Nonhierarchical Feedback Control



- Feedback suppression:
 - NACK only
 - first (multicast) retransmission request (after random delay) leads to the suppression of others
 - retransmission (not necessarily original sender) is also multicast
- Scales well, but accurate scheduling is difficult and feedback interrupts successful processes, too

Hierarchical Feedback Control



- The essence of hierarchical reliable multicasting.
 - a) Each local coordinator forwards the message to its children.
 - b) A local coordinator handles retransmission requests.
 - c) Scales well, but dynamic tree construction is a remaining problem

Group Communication

- Reliable multicast and failure atomicity
- Multicast ordering
- Static and dynamic group membership
- Atomic multicast and virtual synchrony

Total, FIFO and causal ordering of multicast messages (1)

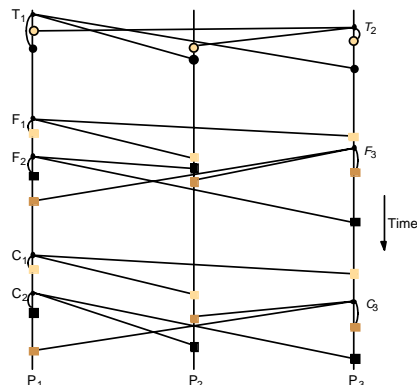
- **FIFO** ordering: If a process issues F_1 and then F_2 , then every process will deliver F_1 before F_2 (partial ordering)
- **Causal** ordering: If C_1 happened-before C_2 , then every process will deliver C_1 before C_2 (partial ordering)
- **Total** ordering: If a process delivers T_1 before T_2 , then all processes deliver T_1 before T_2
- Causal ordering implies FIFO ordering
- We do not assume or imply reliability (can be combined)

Total, FIFO and causal ordering of multicast messages (2)

Notice the consistent ordering of **totally** ordered messages T_1 and T_2 , the **FIFO-related** messages F_1 and F_2 and the **causally** related messages C_1 and C_3 – and the **otherwise arbitrary** delivery ordering of messages.

Hybrids:

- FIFO-total
- Causal-total

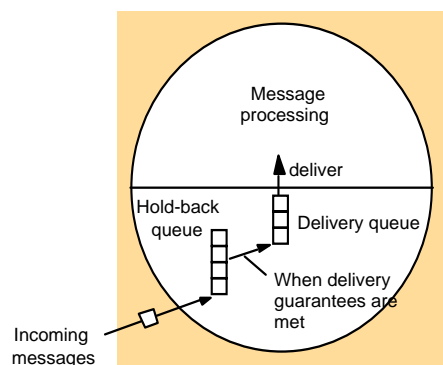


Message ordering

Multicast	Basic Message Ordering	Total-ordered Delivery?
Reliable multicast	None	No
FIFO multicast	FIFO-ordered delivery	No
Causal multicast	Causal-ordered delivery	No
Atomic multicast	None	Yes
FIFO atomic multicast	FIFO-ordered delivery	Yes
Causal atomic multicast	Causal-ordered delivery	Yes

- ❑ Epochs are separated by group membership changes
- ❑ Six different versions of virtually synchronous reliable multicasting regarding ordering within epochs

The hold-back queue for arriving multicast messages



Implementation model for ordered multicast

- ❑ Multicast **queue** at each server node
- ❑ Multicast messages are stored in queue on arrival
- ❑ Messages are numbered (or timestamped) in some way
- ❑ Depending on desired order of delivery, messages are **delivered from queue to the process after some coordination** with queues of other servers
- ❑ For example, if the same message is at head of queue in all queues, then it can be delivered (totally-ordered multicast)
- ❑ Ordering can be expensive, application-specific message semantics can be more efficient ("**end-to-end**"-argument)

A reliable multicast algorithm

```

On initialization
  Received := {};

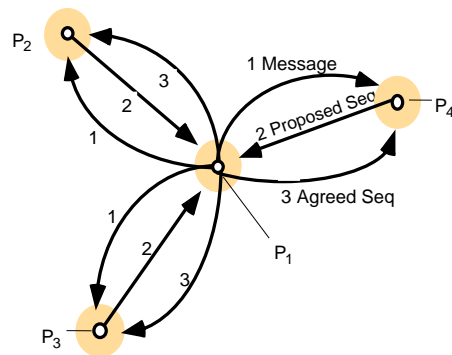
For process p to R-multicast message m to group g
  B-multicast(g, m); // p ∈ g is included as a destination

On B-deliver(m) at process q with g = group(m)
  if (m ∉ Received)
  then
    Received := Received ∪ {m};
    if (q ≠ p) then B-multicast(g, m); end if
    R-deliver m;
  end if
  
```

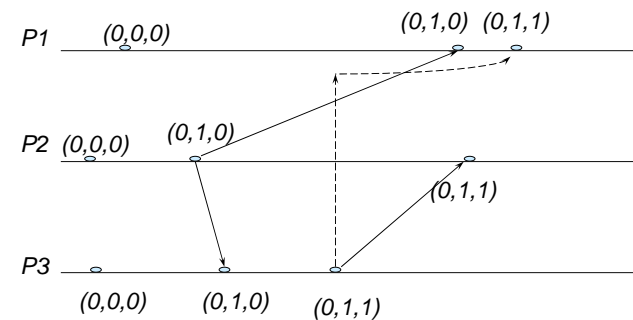
Totally-Ordered Multicasting

- ❑ clients multicast their updates with (Lamport) timestamp (FIFO, reliable)
- ❑ upon receipt, the message is put into local queue ordered by timestamp
- ❑ server acknowledges receipt of requests by multicast (for total ordering);
- ❑ eventually all processes will have the same copy of the local queue
- ❑ a message that is at the head of the queue and has been acknowledged by all processes is delivered to server process (respective ACKs are deleted)
- ❑ updates may not be done in “correct order” but they are done in the same order at all nodes

The ISIS algorithm for total ordering



Causal ordering using vector timestamps



Causal ordering using vector timestamps

Algorithm for group member p_i ($i = 1, 2, \dots, N$)

On initialization

$V_i^g[j] := 0$ ($j = 1, 2, \dots, N$);

The number of group-g messages from process j that happened before the current message to be multicast

To CO-multicast message m to group g

$V_i^g[i] := V_i^g[i] + 1$;

B-multicast($g, \langle V_i^g, m \rangle$);

Timestamps count the multicast messages only

On B-deliver($\langle V_j^g, m \rangle$) from p_j , with $g = \text{group}(m)$

place $\langle V_j^g, m \rangle$ in hold-back queue;

wait until $V_j^g[j] = V_i^g[j] + 1$ and $V_j^g[k] \leq V_i^g[k]$ ($k \neq j$);

CO-deliver m ; // after removing it from the hold-back queue

$V_i^g[j] := V_i^g[j] + 1$;

Optimization of the merge operation

Wait conditions

- Wait conditions: P_i waits until it has delivered
 - 1. any earlier message from P_j (FIFO)
 - 2. any message that P_j has delivered (accepted locally) before it multicast the message m

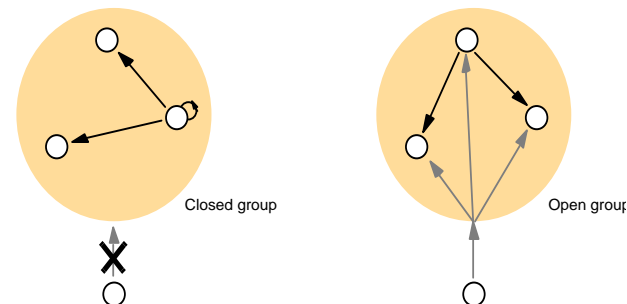
Group Communication

- Reliable multicast and failure atomicity
- Multicast ordering
- Static and dynamic group membership
- Atomic multicast and virtual synchrony

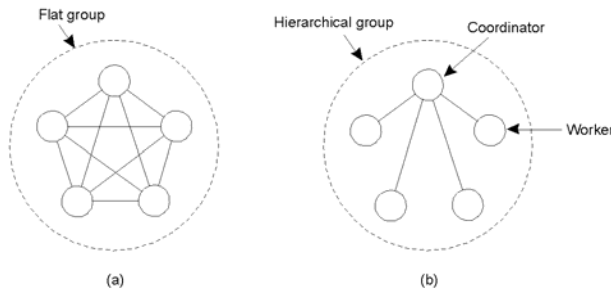
Design Issues for Process Groups

- Organize identical processes in a group
- Purpose: Collection of processes as a single abstraction
- Multicast is a key issue: all requests arrive at all servers in the same order (**atomic multicast**)
- Groups may be **dynamic**: mechanisms are needed to manage groups and group memberships
- Open groups vs. closed groups
- Flat groups vs. hierarchical groups
- A process can be member in several groups

Open and closed groups



Flat and hierarchical groups



- How can a message be delivered to all members of a group?
- Flat group: no single point of failure
- (Simple) hierarchical group: Co-ordinator; decision making is easier

Group Communication

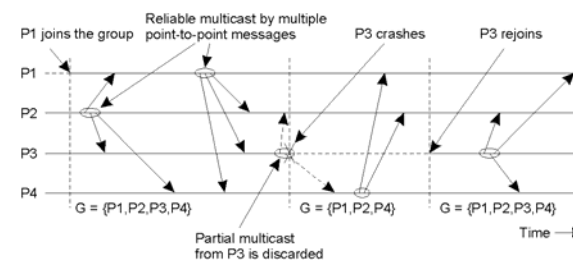
- Reliable multicast and failure atomicity
- Multicast ordering
- Static and dynamic group membership
- Atomic multicast and virtual synchrony

Group membership

- Creating and deleting groups; processes joining and leaving (or crashing)
 - group server: easy, efficient, but single point of failure
 - distributed group membership service (e.g. by reliable multicasting)
- Joining and leaving operations must be synchronous with data messages (e.g. by converting this operation into a sequence of messages sent to the whole group)
- Crashing may be more difficult to detect (fail-stop is too strong, usually fail-silent can be assumed)
- How to rebuild a group consistently.

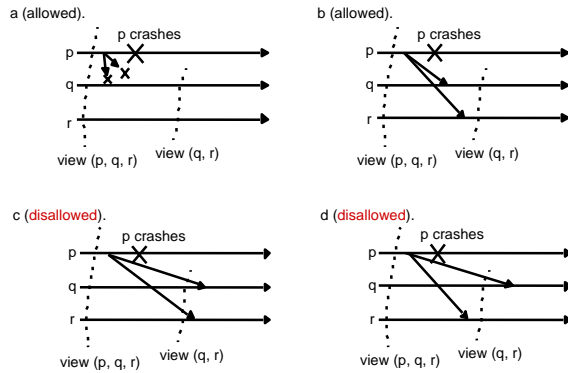
Virtual Synchrony

Concepts: Group view and view delivery



- Either all (non-faulty) processes in the group receive the multicast in the same view, or none receives it (agreement, atomicity)
- The view delivery itself is totally ordered

View-synchronous group communication



Summary

- A reliable multicast (group communication) is in many cases necessary for providing fault-tolerant distributed algorithms
- Guarantees differ:
 - reliable multicasting (failure atomicity)
 - ordering guarantees
 - dynamic group membership (faster, but weaker agreement guarantees than static)
 - virtual synchrony