CSE 486/586 Distributed Systems
Reliable Multicast (Part 2)

Slides by Steve Ko
Computer Sciences and Engineering
University at Buffalo
Last Time

- **Multicast**
  - One-to-many communication within a group of processes
- **Issues to be handled**
  - Processes fail
  - Messages get delayed
- **B-multicast**
- **R-multicast**
  - Integrity, Agreement, Validity
- **Ordering**
  - Why do we care?
Recap: Ordering

- Totally ordered messages $T_1$ and $T_2$.
- FIFO-related messages $F_1$ and $F_2$.
- Causally related messages $C_1$ and $C_3$.

- Total ordering does not imply causal ordering.
- Causal ordering implies FIFO ordering.
- Causal ordering does not imply total ordering.
- Hybrid mode: causal-total ordering, FIFO-total ordering.
Example: FIFO Multicast

(Not to be confused with vector timestamps!)

Physical Time

P1

P2

P3

Sequence Vector
Totally Ordered Multicast

- Using a **sequencer**
  - One dedicated “sequencer” that orders all messages
  - Everyone else follows.
- **ISIS** system
  - Similar in result to a sequencer
  - The responsibility for sequencing is *distributed to each sender*
Total Ordering Using a Sequencer

1. Algorithm for group member $p$
   On initialization: $r_g := 0$
   To TO-multicast message $m$ to group $g$:
   B-multicast($g \cup \{ \text{sequencer}(g) \}, <m, i>$)
   On B-deliver($<m, i>$) with $g = \text{group}(m)$:
   Place $<m, i>$ in hold-back queue
   On B-deliver($m_{\text{order}} = \langle \text{"order"}, i, S \rangle$) with $g = \text{group}(m_{\text{order}})$:
   Wait until $<m, i>$ in hold-back queue and $S = r_g$
   TO-deliver $m$
   $r_g = S + 1$

2. Algorithm for sequencer of $g$
   On initialization: $s_g := 0$
   On B-deliver($<m, i>$) with $g = \text{group}(m)$:
   B-multicast($g, \langle \text{"order"}, I, s_g \rangle$)
   $s_g := s_g + 1$
ISIS algorithm for total ordering
ISIS algorithm for total ordering

1) Sender multicasts message to everyone
2) Each process replies with its *proposed* priority (seq. no.)
   - Larger than all observed *agreed* priorities
   - Larger than any priority previously proposed by this process
3) Store messages in a *priority queue*
   - Ordered by priority (proposed or agreed)
   - Mark message as undeliverable
4) Sender chooses *agreed* priority, multicasts message
   - Agreed priority = *maximum* of all proposed priorities
5) Upon receiving agreed priority
   - Mark message as deliverable
   - Deliver any deliverable messages at the front of priority queue
Problematic Scenario

- Two processes $P1$ & $P2$ at their initial state.
- $P1$ sends $m_1$ & $P2$ sends $m_2$.
- $P1$ receives $m_1$ (its own) and proposes priority 1. $P2$ does the same for $m_2$.
- $P2$ receives $m_1$ ($P1$’s message) and proposes priority 2. $P1$ does the same for $m_2$.
- $P1$ picks 2 for $m_1$ & $P2$ also picks 2 for $m_2$.
- Same sequence number for two different messages!
- How do you want to solve this?
Example: ISIS algorithm [1]

Showing the process id only when necessary

A:1 Message  A:2 Updated Message  A:2 Ordered Message  ✔ Delivered

B:1 Proposal

C:3.2  C:3  C:3.3  B:3.1

P1

P2

P3

P: A, B, C, t
Proof of Total Order

• For a message $m_1$, consider the first process $p$ that delivers $m_1$
• Let $m_1$ be deliverable & at the head of the queue at $p$
• Let $m_2$ be another message that has not been delivered
  \[\text{finalpriority}(m_2) \geq \text{proposedpriority}(m_2) \geq \text{finalpriority}(m_1)\]
• Let $p'$ be a process that delivers $m_2$ before $m_1$. At $p'$:
  \[\text{finalpriority}(m_1) \geq \text{proposedpriority}(m_1) \geq \text{finalpriority}(m_2)\]
• a contradiction!
Causally Ordered Multicast

- Each process keeps a vector clock
  - Each entry $i$ in the vector clock counts the number of messages delivered from process $P_i$
  - Each process’s own entry in its vector represents the number of messages it has sent

- When sending a multicast message $m$, process $P_i$ increments its own entry, then sends its vector with $m$

- When receiving a message $m$, from process $P_j$ process $P_i$ waits to deliver $m$ until it can preserve causal ordering:
  - It has delivered all previous messages from $P_j$
  - It has delivered all messages that $P_j$ had delivered before $m$
Causal Ordering

The number of group-g messages from process j that have been seen at process i so far.
Example: Causal Ordering Multicast

Physical Time

P1

(0,0,0) ➔ 1,0,0 ➔ 1,1,0 ➔ 1,1,0 ➔ Reject:

P2

(0,0,0) ➔ 1,0,0 ➔ 1,1,0 ➔ 1,1,0 ➔ (1,1,0) ➔ deliver:

P3

(0,0,0) ➔ deliver,

Buffer, missing P1(1)

(1,0,0) ➔ (1,1,0) ➔ 1,0,0 ➔ 1,1,0 ➔ deliver buffered message

deliver
Summary

• Total Ordering Multicast
  - Sequencer
  - ISIS

• Causal Ordering Multicast
  - Uses vector timestamps
References


Acknowledgements

• These slides are by Steve Ko, used and lightly modified with permission by Ethan Blanton.
• These slides contain material developed and copyrighted by Indranil Gupta (UIUC).