

**Computer Science
425
Distributed Systems**

**Lecture 4
Multicast
Reading: Section 12.4**

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-1

Communication Modes in DS

- ❖ Unicast (*best effort or reliable*)
 - Messages are sent from exactly one process to one process.
 - *Best effort* guarantees that if a message is delivered it would be intact.
 - *Reliable* guarantees delivery of messages.
- ❖ Broadcast
 - Messages are sent from exactly one process to all processes on the network.
 - Reliable broadcast protocols are not practical.
- ❖ Multicast
 - Messages are sent from exactly one process to several processes on the network.
 - Reliable multicast can be implemented “above” (i.e., “using”) a reliable unicast.

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

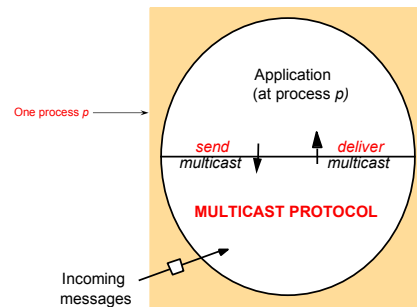
Lecture 4-2



© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-3

What're we designing in this class



© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-4

Basic Multicast (B-multicast)

- A straightforward way to implement B-multicast is to use a reliable one-to-one send operation:
 - $B\text{-multicast}(g,m)$: for each process p in g , $send(p,m)$.
 - $receive(m)$: $B\text{-deliver}(m)$ at p .
- A “correct” process = a “non-faulty” process
- A basic multicast primitive guarantees a correct (i.e., non-faulty) process will eventually deliver the message, as long as the sender (multicasting process) does not crash.
 - Can we provide reliability even when the sender crashes (after it has sent the multicast)?

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-5

Reliable Multicast

- **Integrity:** A correct (i.e., non-faulty) process p delivers a message m at most once.
- **Validity:** If a correct process multicasts (sends) message m , then it will eventually deliver m .
 - Guarantees liveness to the sender.
- **Agreement:** If a correct process delivers message m , then all the other correct processes in $group(m)$ will eventually deliver m .
 - Property of “all or nothing.”
 - **Validity and agreement together ensure overall liveness: if some correct process multicasts a message m , then, all correct processes deliver m too.**

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-6

Reliable Multicast Algorithm



On initialization
 $Received := \{\}$;

For process p to R-multicast message m to group g
 $B\text{-multicast}(g, m);$ // $p \in g$ is included as a destination

On B-deliver(m) at process q with $g = \text{group}(m)$
 if ($m \notin Received$)
 then
 $Received := Received \cup \{m\};$
 if ($q \neq p$) then $B\text{-multicast}(g, m);$ end if
 R-deliver $m;$
 end if

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-7

Reliable Multicast Algorithm (R-multicast)

On initialization
 $Received := \{\}$;

For process p to R-multicast message m to group g
 $B\text{-multicast}(g, m);$ // $p \in g$ is included as a destination

On B-deliver(m) at process q with $g = \text{group}(m)$
 if ($m \notin Received$) Integrity
 then
 $Received := Received \cup \{m\};$
 if ($q \neq p$) then $B\text{-multicast}(g, m);$ end if Agreement
 R-deliver $m;$ Integrity, Validity
 end if

if some correct process B-multicasts a message m , then all correct processes deliver m too. If no correct process B-multicasts m , then no correct processes deliver m .

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-8

Ordered Multicast

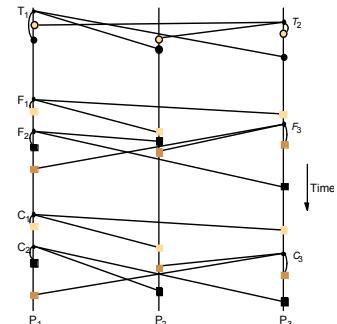
- **FIFO ordering:** If a correct process issues $\text{multicast}(g, m)$ and then $\text{multicast}(g, m')$, then every correct process that delivers m' will have already delivered m .
- **Causal ordering:** If $\text{multicast}(g, m) \rightarrow \text{multicast}(g, m')$ then any correct process that delivers m' will have already delivered m .
- **Total ordering:** If a correct process delivers message m before m' , then any other correct process that delivers m' will have already delivered m .

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-9

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



© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-10

Display From Bulletin Board Program

Bulletin board: os.interesting		
Item	From	Subject
23	A.Hanlon	Mach
24	G.Joseph	Microkernels
25	A.Hanlon	Re: Microkernels
26	T.L.'Heureux	RPC performance
27	M.Walker	Re: Mach
end		

What is the most appropriate ordering for this application?
 (a) FIFO (b) causal (c) total

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-11

Providing Ordering Guarantees (FIFO)

- ❖ Process messages from each process in the order they were sent:
 - ❖ Each process keeps a sequence number for each other process.
 - ❖ When a message is received,
 - as expected (next sequence), accept
 - if Message# is higher than expected, buffer in a queue
 - lower than expected, reject

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-12

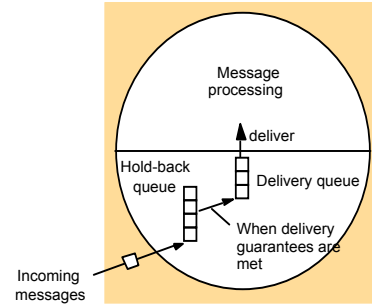
Implementing FIFO Ordering

- S_p^g : the number of messages p has sent to g .
- R_g^q : the sequence number of the latest group- g message p has delivered from q .
- For p to FO-multicast m to g
 - p increments S_p^g by 1.
 - p "piggy-backs" the value S_p^g onto the message.
 - p B-multicasts m to g .
- At process p , Upon receipt of m from q with sequence number S :
 - p checks whether $S = R_g^q + 1$. If so, p FO-delivers m and increments R_g^q .
 - If $S > R_g^q + 1$, p places the message in the hold-back queue until the intervening messages have been delivered and $S = R_g^q + 1$.

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-13

Hold-back Queue for Arrived Multicast Messages

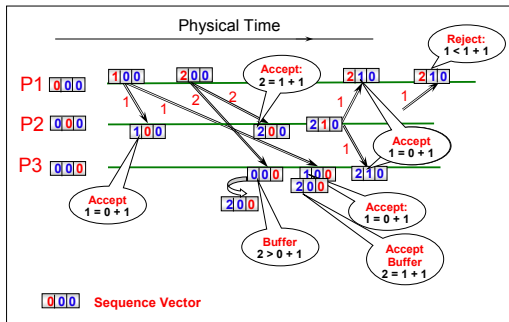


© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-14

Example: FIFO Multicast

(do NOT confuse with vector timestamps)



© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-15

Total Ordering Using a Sequencer

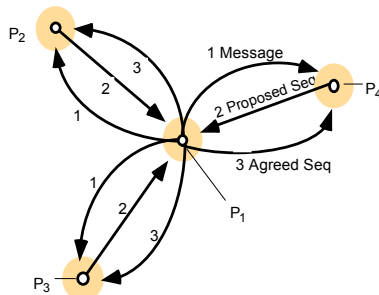
1. Algorithm for group member p
 - On initialization: $r_g := 0$;
 - To TO-multicast message m to group g :
B-multicast($g \cup \{\text{sequencer}(g)\}, \langle m, i \rangle$);
 - On B-deliver($\langle m, i \rangle$) with $g = \text{group}(m)$:
Place $\langle m, i \rangle$ in hold-back queue;
 - On B-deliver($m_{order}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

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

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-16

ISIS algorithm for total ordering



© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-17

ISIS algorithm for total ordering

1. The multicast sender multicasts the message to everyone.
2. Recipients add the received message to a special queue called the *priority queue*, tag the message *undeliverable*, and reply to the sender with a *proposed priority* (i.e., proposed sequence number). Further, this proposed priority is *1 more than the latest sequence number heard so far at the recipient, suffixed with the recipient's process ID*. The *priority queue* is always sorted by priority.
3. The sender collects all responses from the recipients, calculates their *maximum*, and re-multicasts original message with this as the *final priority* for the message.
4. On receipt of this information, recipients mark the message as *deliverable*, reorder the priority queue, and deliver the set of lowest priority messages that are marked as *deliverable*.

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-18

Proof of Total Order

- For a message m_1 , consider the first process p that delivers m_1
 - At p , when message m_1 is at head of priority queue and has been marked *deliverable*, let m_2 be another message that has not yet been delivered (i.e., is on the same queue or has not been seen yet by p)
 - Since proposed priorities by process p only increase
 - Due to "max" operation at sender
 - Since queue ordered by increasing priority
 - Suppose there is some other process p' that delivers m_2 before it delivers m_1 . Then at p' ,
 - Since queue ordered by increasing priority
- $finalpriority(m_2) \geq$
 $proposedpriority(m_2) \geq$
 $finalpriority(m_1)$
 $proposedpriority(m_1) \geq$
 $finalpriority(m_2)$

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-19

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 have been seen at process i so far

To CO-multicast message m to group g

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

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

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

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

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

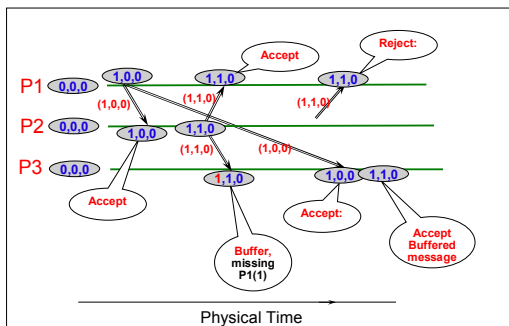
CO-deliver m ; // after removing it from the hold-back queue

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

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-20

Example: Causal Ordering Multicast



© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-21

Summary

Multicast is operation of sending one message to multiple processes

- Reliable multicast algorithm built using unicast
- Ordering – FIFO, total, causal

Next week

- Read Sections 12.2
- Homework 1 due next Tuesday
 - Come by office hours with your questions
- MP0 on the web today: no due date, but recommended
 - Come by office hours with your questions

© 2002, M. T. Harandi and J. Hou (modified: I. Gupta)

Lecture 4-22