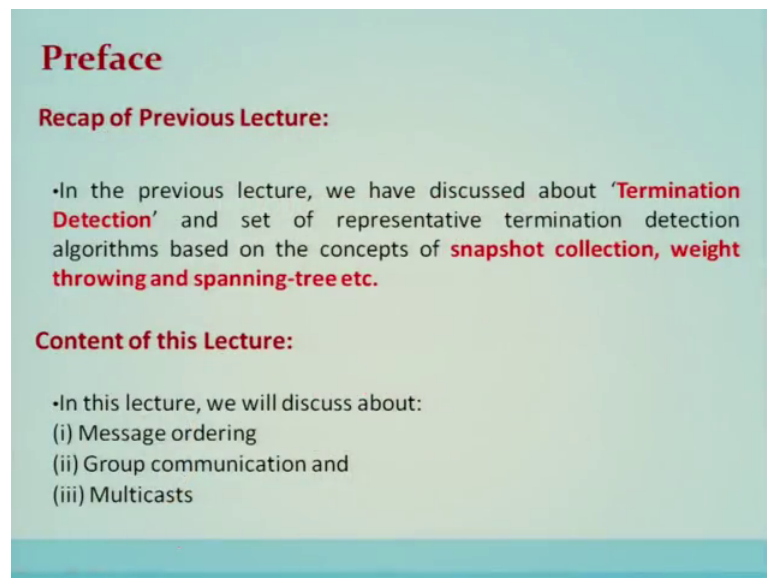


Distributed Systems
Dr. Rajiv Misra
Department of Computer Science and Engineering
Indian Institute of Technology, Patna

Lecture – 16
Message Ordering and Group Communication

Lecture 16 Message Ordering and Group Communication.

(Refer Slide Time: 00:21)



Preface

Recap of Previous Lecture:

- In the previous lecture, we have discussed about '**Termination Detection**' and set of representative termination detection algorithms based on the concepts of **snapshot collection, weight throwing and spanning-tree etc.**

Content of this Lecture:

- In this lecture, we will discuss about:
 - (i) Message ordering
 - (ii) Group communication and
 - (iii) Multicasts

Preface Recap of Previous Lecture. In previous lecture we have discussed about 'Termination Detection' and set of representative termination detection algorithm based on the concept of snapshot collection, weight throwing and spanning-tree. Content of this Lecture in this lecture we will discuss about Message ordering, Group Communication and Multicast that is multi casting.

(Refer Slide Time: 00:49)

Introduction

- At the core of distributed computing is the communication by message passing among the processes participating in the application.
- In this lecture, we will study **several message ordering paradigms** for communication, such as asynchronous, synchronous, FIFO, causally ordered, and non-FIFO orderings. These orders form a hierarchy. Then we will examine few algorithms to implement these orderings.
- **Group communication** is an important aspect of communication in distributed systems. Causal order and total order are the popular forms of ordering when doing group multicasts and broadcasts. Algorithms to implement these orderings in group will also be discussed.

Introduction: at the core of the distributed computing is the communication by message passing among the processes participating in the application. So, in this lecture we will study several message ordering paradigms for communication such as asynchronous, synchronous, FIFO, causally ordered, and non-FIFO ordered these orders form the hierarchy will examine few algorithms to implement these orderings.

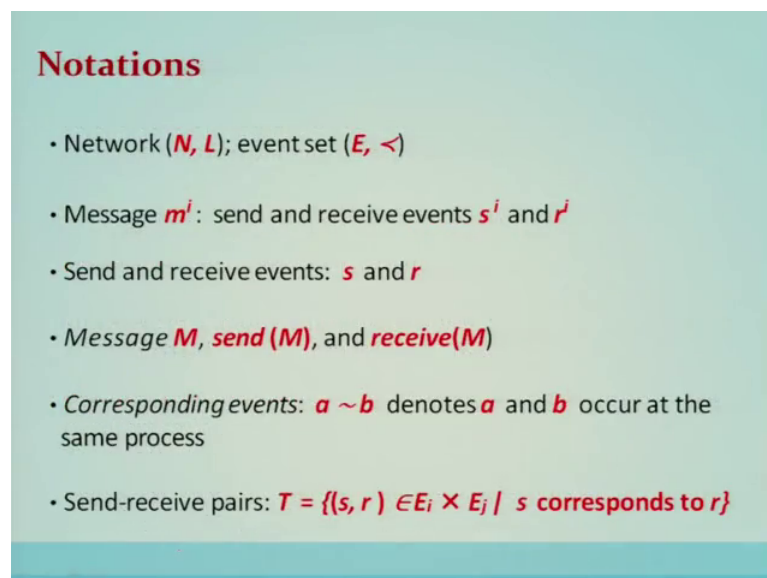
Group communication is an important aspect of communication in a distributed system causally order and total order are popular forms of ordering when doing group multicast and broadcasts, algorithm to implement these orderings will also be discussed before we go ahead let me give you few examples where this message ordering and multi casting will be useful. So, as you know that in a distributed databases the database is normally replicated to add more than one sites.

Now, whenever there is an update at the database then at all places that is at all the replicas have to be updated instantaneously. So, if 2 messages are sent for update at different point of time. So, this particular order in which 2 messages are sent these updates are to be made in that same order irrespective of how much delay the messages are ensuring, how this is all done this is all done here in ordering of messages and also another form of message ordering or communication in a group is basically called multi casting.

So, multi casting is an example of 2 cases one is called closed group and the other is called open group. Open group is like online railway reservation system where any customer at any point of time can come and enter into a system and perform the operations. So, that client an external person is using a set of nodes for it is application then it is called an open system open system communication is basically done through the multicasting.

So, if large number of users are allowed in reservation systems then it is not an easy task it is a difficult task to basically ensure the correctness and running the application. So, this will be talked about during the group communication and multi casting how this paradigm is going to be implemented in the system of message passing system.

(Refer Slide Time: 03:43)

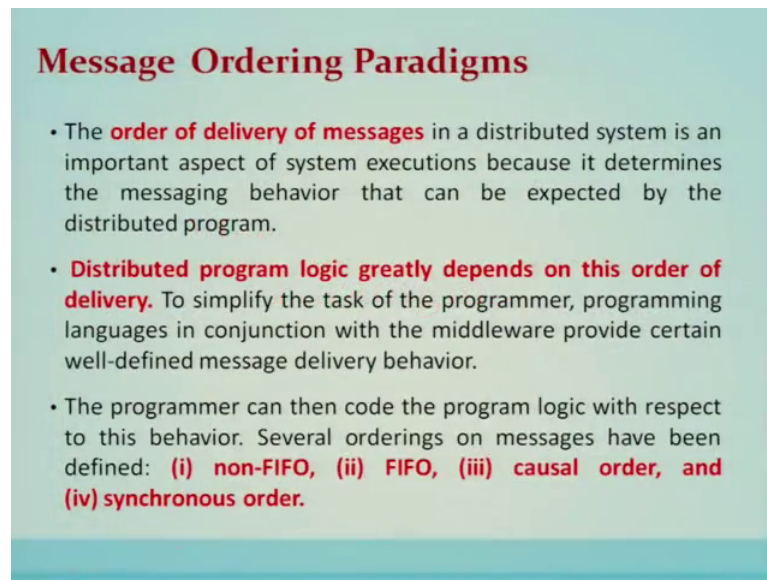


Notations

- Network (N, L) ; event set $(E, <)$
- Message m^i : send and receive events s^i and r^i
- Send and receive events: s and r
- Message M , $send(M)$, and $receive(M)$
- Corresponding events: $a \sim b$ denotes a and b occur at the same process
- Send-receive pairs: $T = \{(s, r) \in E_i \times E_j \mid s \text{ corresponds to } r\}$

That is called distributed system there are few notations and some of these notations are explained so we skip this.

(Refer Slide Time: 03:55)



Message Ordering Paradigms

- The **order of delivery of messages** in a distributed system is an important aspect of system executions because it determines the messaging behavior that can be expected by the distributed program.
- **Distributed program logic greatly depends on this order of delivery.** To simplify the task of the programmer, programming languages in conjunction with the middleware provide certain well-defined message delivery behavior.
- The programmer can then code the program logic with respect to this behavior. Several orderings on messages have been defined: **(i) non-FIFO, (ii) FIFO, (iii) causal order, and (iv) synchronous order.**

Now, message ordering paradigms message ordering paradigms the order of messages in a distributed system is an important aspect of system executions because it determines the messaging behavior that can be expected by the distributed program. So, distributed program logic greatly depends on this order of delivery to simplify the task of a programming languages in conjunction with the middleware provides well defined message behavior.

So, again recreate on the same point the message the order of delivery of the message is most important as far as distributed applications are concerned and it reflects and it (Refer Time: 04:41) and it that particular ordering comes from the applications and the implementations will be basically governed here using the message ordering paradigms. So, the programmer can send the code for the programming logic with respect to this particular behavior there are several ordering on the messages which are defined as non-FIFO; FIFO causal order and synchronous order.

(Refer Slide Time: 05:09)

Definitions (1)

- 1. Asynchronous execution (A-execution):** It is an execution (E, \prec) for which the causality relation is a partial order.
- 2. FIFO executions:** A FIFO execution is an A-execution in which, for all (s, r) and $(s', r') \in T$, $(s \sim s' \text{ and } r \sim r' \text{ and } s \prec s') \Rightarrow r \prec r'$
Handwritten notes: "within a process" and "communication channel"
- 3. Causal order (CO):** A CO execution is an A-execution in which, for all (s, r) and $(s', r') \in T$, $(r \sim r' \text{ and } s \prec s') \Rightarrow r \prec r'$
Handwritten notes: "within a process" and "communication channel"
- 4. Causal order (CO) for Implementations:** If $\text{send}(m^1) \prec \text{send}(m^2)$ then for each common destination d of messages m^1 and m^2 , $\text{deliver}_d(m^1) \prec \text{deliver}_d(m^2)$ must be satisfied.

Let us see a few definitions on these different kind of message orders. So, asynchronous execution you know that is an execution for which causality relation is a partial order that we are seen another thing is called FIFO in a first in first out executions is an asynchronous execution in which for all pairs sender and receiver and another pair of sender receiver. So, if these sender and these receivers they are happening within a system and there is an order between the sender then there will be an order in the receiver also because these senders and these receivers are happening within a particular process and so within or process they are happening.

So, if this particular send is proceeding over the other send that is s prime. So, their receive also requires this kind of precedence relation and that is called FIFO that is being preserved. So, here the communication channel has to ensure that the delivery is to be following the FIFO manner. Similarly the causal order is an execution in which for all senders and receivers where the sender s and s prime they are basically sending to the same destination that is that is r .

And if there is a precedence between the send operation; that means, if s precedes s prime, then this particular receive process are also proceeds r prime and r and r prime is happening at the same process that is what this particular symbol indicates. So, the diagram reflects whatever is the definition of a causal order. So, if there is a precedence between 2 sends which are happening at different sites then when they will be received

and if they have precedence relations. So, that relation also will be insured at the time of delivery and that ordering of the message is called causal ordering of messages.

(Refer Slide Time: 08:02)

Definitions (2)

5. Message Order (MO): A MO execution is and A-execution in which, for all (s, r) and $(s', r') \in T$, $s < s' \Rightarrow \neg(r' < r)$

6. Empty-Interval (EI) execution: A execution $(E, <)$ is an empty-interval (EI) execution if for each pair of events $(s, r) \in T$, the open interval set $\{x \in E / s < x < r\}$ in the partial order is empty.

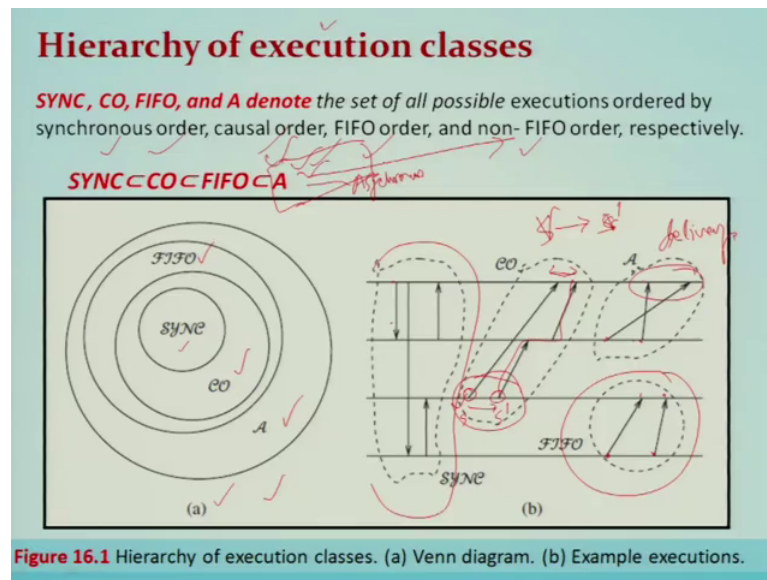
7. Synchronous Executions (SYNC): An execution $(E, <)$ for which the causality relation is a partial order.

- Handshake between sender and receiver
- Instantaneous communication \Rightarrow modified definition of causality, where s, r are atomic and simultaneous, neither preceding the other.

Now, another kind of ordering is called synchronous; synchronous that sync order and execution for which the causality relation is a partial order is called the synchronous order. So, synchronous order is like for every send there will be instantaneous receive. So, this will be handled as far as synchronous ordering is concerned over an asynchronous communication channel.

So; that means, the sender and the receiver they basically work in a synchronization and whatever sender is sending receiver will be receiving within that synchronization they are called synchronous ordering. So, it is also called the instantaneous communication and this is the modified definition of causality.

(Refer Slide Time: 09:06)



These kind of ordering that is we have defined like non-FIFO order FIFO order causal and synchronous order they basically follow in hierarchy in the execution class which is shown over here in the figure at A. So, in this particular figure you can see this kind of relation holds where the causal order is a proper set of FIFO and synchronous is a proper set of the causal order and a means non-FIFO a means a synchronous. So, asynchronous does not follow any order and that is why it is non-FIFO.

So, here we can see from this particular hierarchy that that sync or a synchronous order is basically a proper set of causal order and causal order is a proper set of FIFO and FIFO is a proper set of non-FIFO or an asynchronous the examples here we can see that if it is a sync; that means, sender and receiver they works in synchronization they can be represented by this particular arrows.

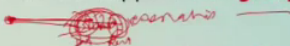
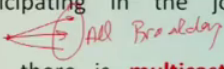
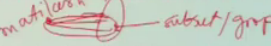
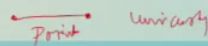
Causal order in a causal order we can see that if these 2 sends they are having the precedence relation or this send is happened before the other send then as far as when they when they receive. So, this particular order is also ensured at the delivery time hence s 1 sorry hence s precedes s prime at the delivery also why because here they are following the receive order.

In FIFO we can see that the order in which these messages are sent the same order is preserved at the time of delivery of these messages asynchronous does not follow any order. So, the order at the time of sending these messages is not preserved at the time of

delivery of the messages. So, this will include a hierarchy of these kinds now underlying the asynchronous mode where the message passing follows the delivery of the message having unpredictable delays these particular ordering, how they are going to be insured we are going to see in the further slides how they are going to be implemented and used up.

(Refer Slide Time: 12:10)

Group Communication

- Processes across a distributed system cooperate **to solve a joint task**. Often, they need to communicate with each other as a group, and therefore there needs to be support **for group communication**. 
- A **message broadcast** is the sending of a **message to all members** in the distributed system. The notion of a system can be confined only to those sites/processes participating in the joint application. 
- Refining the notion of broadcasting, there is **multicasting** wherein a message is sent to a **certain subset, identified as a group**, of the processes in the system. 
- At the other extreme is **unicasting**, which is the familiar **point-to-point** message communication. 

Now, the group communication processes across a distributed system cooperate to solve the joint tasks often they need to communicate with each other as a group and therefore, there needs to be support for the group communication. So, example of such applications are for example, if there is a railway reservation system which comprises of several nodes. So, the person outside can directly communicate to these set of system which represents the railway reservation system which includes a bank and the railways and so on and this particular system requires a group communication.

Why because this particular client or a user has to communicate only to a set of processor and this is called a group of processor and this kind of communication is called a group communication. Group Communications are to be supported by the applications. A message broadcast is sending up a message to all the members in a distributed system, the notion of a system can be confined only to those sites processes participating in the joint application refining the notion of the broadcasting there is a multi casting wherein

the message is sent to a certain subset identified as a group of the processes in the system.

So, at the other extreme is the unicasting which is familiar point to point message communication let me tell you that that broadcasting means sending a message to all is called broadcasting if sending message not to all, but to a subset or to a group then it is called basically a multi casting now if sending a message to another node then it is point to point communication and it is called a unicasting.

(Refer Slide Time: 14:32)

Contd...

- **Network layer or hardware-assist multicast** cannot easily provide:
 - > Application-specific semantics on message delivery order
 - > Adapt groups to dynamic membership
 - > Multicast to arbitrary process set at each send
 - > Provide multiple fault-tolerance semantics
- Closed group (source part of group) vs. open group (outside of group)
- # groups can be $O(2^n)$

Figure 16.2: (a) Updates to 3 replicas. (b) Causal order (CO) and total order violated. (c) Causal order violated. If m did not exist, (b,c) would not violate CO.

Now, network layer or the hardware assist the multicast cannot easily provide application specific semantics on message delivery order adopt groups to dynamic membership multicast to arbitrary process set at each send provide multiple fault tolerance semantics. So, these are basically the few things or a few properties which has to be implemented why because in the network layer this hardware assists multicast cannot be cannot be provided.

So, I told you that the closed group that is the source is a part of the group. So, the source is also a part of the group to which they have to communicate then it is called a closed group and an open group means if the source is outside the group then it is called the open group. So, these this particular example which will tell you about the application is specific semantics on the message delivery order.

For example there are 2 process P 1 and P 2 they are causally related through this particular message M which is shown over here and there are 3 sides R 1 R 2 R 3 which basically will maintain the replicas. So, if message M 1 is multicast this is a multicast to all R 1 R 2 and R 3 after that P 2 multicast another message M 2 for updates on these replicas then let us see the scenario what happens.

Now, you know that P 1 is sent before P 2 and this particular message m ensures the precedence relation between P 1 and P 2. So, as far as R 1 is concerned R 1 will receive P 2 first and then P 1. So, it is not following this particular precedence relation it is violating R 2 is concerned R 2 is receiving P 1 first and then P 2 R 3 is also receiving.

So, this particular update as far as the b is concerned is not basically causal order and also total order is also violated, total order in the sense which I will explain later on total order meaning to say that each site will see same ordering. So, here you see that here P 2 followed by P 1 is the order in which the requests are coming for the updates as far as R 2 is concerned R 2 will have another view P 1 followed by P 2.

So, basically it is not having the total order as well. So, causal order is also violated and total order is also violated another scenario which says that in all other sites P 2 P 2 and then P 1. So, P 2 is occurring first then P 1 violating causal order is violated here in this particular scenario we will discuss all these things in great details in this part of the lecture.

(Refer Slide Time: 18:34)

The Raynal-Schiper-Toueg algorithm (1991)

- Intuitively, it seems logical that each **message M** should carry a log of all other **messages**, or their identifiers, sent causally before **M's** send event, and sent to the same destination **dest(M)**.
- This log can then be examined to ensure whether it is safe to deliver a message.
- All algorithms aim to reduce this log overhead, and the space and time overhead of maintaining the log information at the processes.
- **Algorithm 16.1** gives a canonical algorithm that is representative of several algorithms that **try to reduce the size of the local space and message space overhead** by various techniques.

M carries log → [log], Total Overhead

Raynal-Schiper-Toueg algorithm in 1991 let us discuss this algorithm for the group communication intuitively it seems logical that message M should carry a log of all other messages or their identifiers, send causal before M's send event, and sent to the same destination M. This log can then be examined to ensure whether it is safe to deliver the message all the algorithms aim to reduce this log overhead and space and time overhead of maintaining log information at the processes.

So, using this algorithm the ordering which is required in causal order or the total order can be ensured before the delivery like the messages are arrived which are not following causal order then they have to be buffered and applied this particular Raynal-Schiper Toueg algorithm before delivery they have to be buffered and ordered the way they want to be delivered and that is why each process carries the log each message carries the log of other messages.

So, the message becomes heavy why because it contains all this information of ordering and when this message reaches to a site it will be buffered and based on the information which is contained in the log the destination will try to order.

(Refer Slide Time: 20:27)

Raynal-Schiper-Toueg (RST) Algorithm

(local variables)
 array of int $SENT[1 \dots n, 1 \dots n]$ ✓
 array of int $DELIV[1 \dots n]$ ✓ // $DELIV[k] = \#$ messages sent by k that are delivered locally

(1) send event, where P_i wants to send message M to P_j :
 (1a) send $(M, SENT)$ to P_j ;
 (1b) $SENT[i, j] \leftarrow SENT[i, j] + 1$.

(2) message arrival, when (M, ST) arrives at P_i from P_j :
 (2a) deliver M to P_i when for each process x ,
 (2b) $DELIV[x] \geq ST[x, j]$ ✓
 (2c) $\forall x, y, SENT[x, y] \leftarrow \max(SENT[x, y], ST[x, y])$ ✓
 (2d) $DELIV[j] \leftarrow DELIV[j] + 1$.

Algorithm 16.1 gives a canonical algorithm that is representative of several algorithms that try to reduce the size of the local space and message space overhead by various techniques.

Assumptions/Correctness	Complexity
•FIFO channels. ✓	n^2 ints/ process
•Safety: Step (2a,b). ✓	n^2 ints/ message
•Liveness: assuming no failures, finite propagation times ✓	Time per send and receive event: n^2

The messages in the buffer itself and then accordingly it will be delivered as per the order. So, let us see this particular algorithm which ensures the delivery this algorithm assumes the FIFO channels. And there are 2 properties safety and liveness safety property is ensured here in 2 way that is deliver message m which is received at the P_i

where this particular condition ensures that the all previously sent messages are to be delivered before this message will deliver otherwise it will be stored in the buffer itself. Now this particular algorithm you can see that it requires the message space in the form of a log and also the local space because these data structures array of sent array of delivered they are to be maintained.

So, basically different techniques we will see that how they are going to reduce it. So, that the overhead gets reduced and the algorithm becomes efficient and affordable to be implemented.

(Refer Slide Time: 21:51)

Total Order

Total order, which requires that all messages be received in the same order by the recipients of the messages, is formally defined as follows:

Definition (Total order): For each pair of processes P_i and P_j and for each pair of messages M_x and M_y that are delivered to both the processes, P_i is delivered M_x before M_y if and only if P_j is delivered M_x before M_y .

Example: The execution in Figure 16.2 (b) does not satisfy total order. Even if the message m did not exist, total order would not be satisfied. The execution in Figure 16.2(c) satisfies total order.

Now, we are going to see the total order total order requires that all the messages received in the same order by the recipients of the messages is formally defined as for each pair of processors P_i and P_j and for each pair of messages that are delivered to both the processes P_i and P_j both the processors P_i is delivered P_x before M_x before m_y if and only if P_j is delivered M_x before M_y .

(Refer Slide Time: 22:46)

Centralized algorithm for Total Order

(1) When P_i wants to multicast M to group G :

(1a) **send** $M(i, G)$ to coordinator.

(2) When $M(i, G)$ arrives from P_i at coordinator:

(2a) **send** $M(i, G)$ to members of G .

(3) When $M(i, G)$ arrives at P_j from coordinator:

(3a) **deliver** $M(i, G)$ to application.

Coordinator
 P_i
Group

Algorithm 16.2 A centralized algorithm to implement total order and causal order for messages

- Assuming all processes broadcast messages, the centralized solution shown in Algorithm 16.2 enforces total order in a system with FIFO channels.
- Each process sends the message it wants to broadcast to a centralized process, which simply relays all the messages it receives to every other process over FIFO channels. *(in group)*
- It is straightforward to see that total order is satisfied. Furthermore, this algorithm also satisfies causal message order.

So, this is called total order. So, this particular figure I have already explained that this does not satisfy the total order, but this is satisfying the total order why because every process will have the same view and if every process is having same view of the message delivery then it is called the total order. How the total order total ordering of the messages is implemented. So, we are going to see the first algorithm which is called a centralized algorithm or total ordering of messages.

So, if P_i want to multicast a message M to a group G then it will send the P_i will send to the group coordinator and the group coordinator in turn will send it to the group of messages. So, this is called coordinator and hence it is called a centralized algorithm meaning to say that considering these particular channels are FIFO. So, the order in which all the messages are being received at the coordinator and when the coordinator will send to the group members the messages will be ordered because they are assumed as a FIFO.

So, assuming all the process broadcast messages the centralized solution source in this algorithm and forces the order in the system with a FIFO channels I explained you each process sends a message it wants to broadcast to a centralized process which simply release all the message it receives to every other process over FIFO channel in the group.

(Refer Slide Time: 24:37)

Complexity

- **Complexity**
Each message transmission takes two message hops and **exactly n messages** in a system of **n processes**.
- **Drawbacks**
A centralized algorithm has a **single point of failure and congestion**, and is therefore not an elegant solution.

So, it is a straightforward to see that their total order is satisfied why because the order in which the messages are given by the coordinator the same order it will be delivered hence this particular since it is a total order it also satisfies the casual order why because we have seen in the hierarchy class of this model complexity of this algorithm is that each transmission takes 2 message hops and exactly n messages in the system of n processes the Drawback is the centralized system has a single point of failure and also there will be condition towards that.

(Refer Slide Time: 24:57)

Three-phase distributed algorithm

- A distributed algorithm that enforces total and causal order for closed groups is given in **Algorithm 16.3**. The three phases of the algorithm are first described from the viewpoint of the sender, and then from the viewpoint of the receiver.
- **Sender**
 - **Phase 1** In the first phase, a process multicasts (line 1b) the **message M** with a locally unique tag and the local timestamp to the group members.
 - **Phase 2** In the second phase, the sender process awaits a reply from all the group members who respond with a tentative proposal for a revised timestamp for that message M. The await call in line 1d is **non-blocking**, i.e., any other messages received in the meanwhile are processed. Once all expected replies are received, the process computes the maximum of the proposed timestamps for M, and uses the maximum as the final timestamp.
 - **Phase 3** In the third phase, the process multicasts the final timestamp to the group in line (1f).

Handwritten notes: Part 3 phases, Sender, Part 1 (Timestamp taken from proposal for receiver), final time

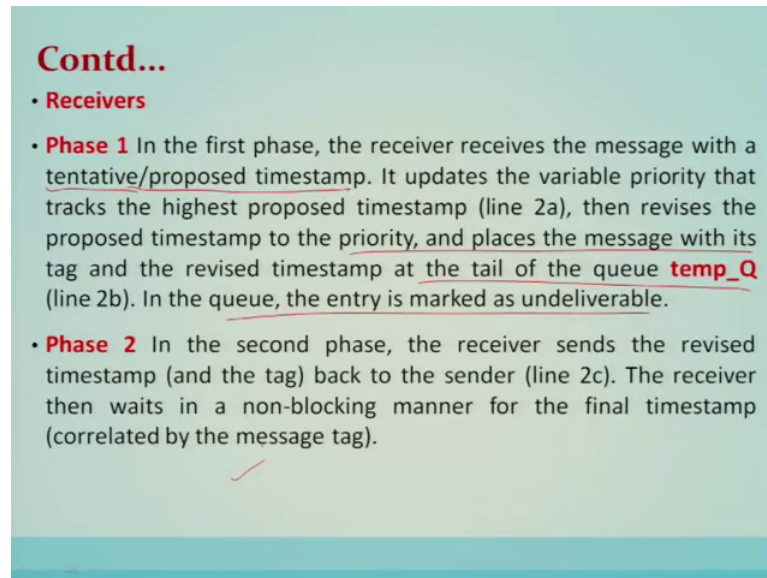
So, another algorithm is called three-phase algorithm three-phase algorithm also ensures the total order. So, if it is ensuring total order; that means, causal order is also ensured as far as the hierarchy of these classes are concerned. So, a distributed algorithm that enforces total order and causal order for closed groups is given by three-phase algorithm. So, the three-phases of the algorithms are first described from viewpoint of the sender and then from the viewpoint of receiver.

So, this algorithm has 2 parts one is the sender part of the algorithm the other is the receiver part of the algorithm. So, first we will see the all three phases of the sender and in the receiver also we will see those corresponding three phases to the sender. So, sender phase 1 in the first phase a process multicasts which is given in line number 1 of the algorithm that we will see multicast message them with a locally unique tag and a local timestamp to the group members.

Phase 2 in the second phase the sender process awaits a reply from all the group member who respond with their tentative proposal for a revised timestamp for that message in the await call in line 1 d is non-blocking that is any other message messages received in a meanwhile are processed once all expected replies are received, the process computes the maximum of the proposed timestamp for M, and uses maximum as the final timestamp.

Third phase is that in the third phase the process multicasts the final timestamp to the groups in the line 1 f. So, the sender will send 3 times in the first phase 1 it will send the message 1 with the timestamp, second time it will send the sender waits awaits the replies from all then it will send a tentative proposal for the revised timestamp and third time it will send the messages with the final timestamp and on the other side receiver will receive them and process them and replies them in all 3 phases that we are going to see here.

(Refer Slide Time: 27:42)



Contd...

- **Receivers**
- **Phase 1** In the first phase, the receiver receives the message with a tentative/proposed timestamp. It updates the variable priority that tracks the highest proposed timestamp (line 2a), then revises the proposed timestamp to the priority, and places the message with its tag and the revised timestamp at the tail of the queue temp_Q (line 2b). In the queue, the entry is marked as undeliverable.
- **Phase 2** In the second phase, the receiver sends the revised timestamp (and the tag) back to the sender (line 2c). The receiver then waits in a non-blocking manner for the final timestamp (correlated by the message tag).

So, in phase 1 the receiver receives the message with a tentative proposal time stamp which is sent by the sender it updates the variable priority that tracks the highest proposed time stamp and then realizes the proposed time stamp to the priority and places the message on it is tag and revises the time stamp at the tail of the Q in the Q the entry is marked as undeliverable.

So, meaning to say that after receiving the messages from the sender it basically finalized a tentative proposal time stamp and accordingly the messages are placed in the in the queues and they are marked as undeliverable they will not be delivered why because they are stored in the Q and according timestamp they are being they will be ordered and then delivered.

(Refer Slide Time: 28:46)

Contd...

- **Phase 3** In the third phase, the final timestamp is received from the multicaster (line 3). The corresponding message entry in **temp_Q** is identified using the tag (line 3a), and is marked as deliverable (line 3b) after the revised timestamp is overwritten by the final timestamp (line 3c). The queue is then resorted using the timestamp field of the entries as the key (line 3c).
- As the queue is already sorted except for the modified entry for the message under consideration, that message entry has to be placed in its sorted position in the queue. If the message entry is at the head of the **temp_Q**, that entry, and all consecutive subsequent entries that are also marked as deliverable, are **dequeued** from **temp_Q**, and enqueued in **deliver_Q** in that order (the loop in lines 3d–3g).

So, to do that it will go for a second phase the receiver sends the revised timestamp back to the sender the receiver then waits in a non-blocking manner for the final in phase 3 the final timestamp is received from the from the multicaster or from the sender the corresponding message entry in a Q is identified using the tag and marked as deliverable after the revised timestamp is over written by the final timestamp the Q is then restored using timestamp field of the entries.

So, as the Q is already sorted except for the modified entry for the message under construction that entry has to be placed into a sorted position if the message entry is at the head of the Q that entry and all the corresponding subsequent entries are also marked as deliverable are dequeued from the Q and delivered to that particular process.

(Refer Slide Time: 29:32)

Algorithm 16.3 3-phase Algorithm Code

```

record Q_entry
M: int; // the application message
tag: int; // unique message identifier
sender_id: int; // sender of the message
timestamp: int; // tentative timestamp assigned to message
deliverable: boolean; // whether message is ready for delivery

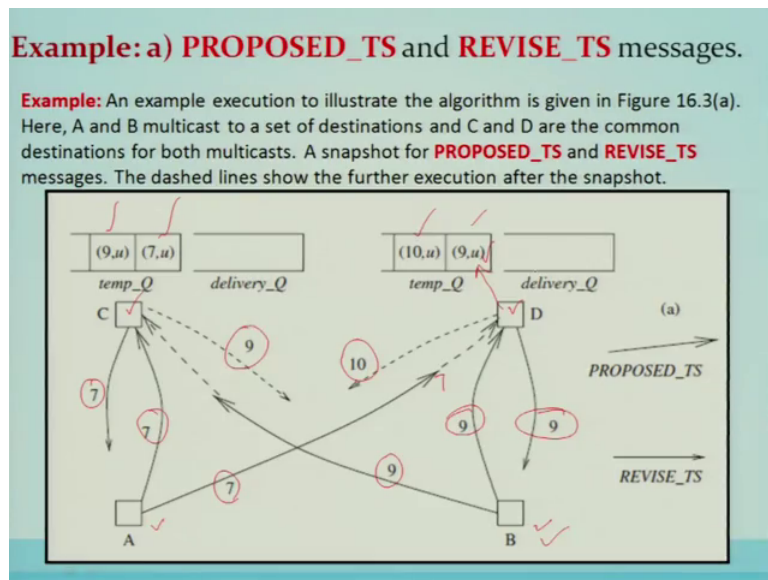
(local variables)
queue of Q_entry: temp_Q, delivery_Q
int: clock // Used as a variant of Lamport's scalar clock
int: priority // Used to track the highest proposed timestamp
(message types)
REVISE_TS(M, i, tag, ts) // Phase 1 message sent by P_i with initial timestamp ts
PROPOSED_TS(j, i, tag, ts) // Phase 2 message sent by P_j with revised timestamp, to P_i
FINAL_TS(j, tag, ts) // Phase 3 message sent by P_j with final timestamp

(1) When process P_i wants to multicast a message M with a tag tag:
(1a) clock = clock + 1;
(1b) send REVISE_TS(M, i, tag, clock) to all processes;
(1c) temp_ts = 0;
(1d) await PROPOSED_TS(j, i, tag, ts_j) from each process P_j;
(1e)  $\forall j \in N$ , do temp_ts = max(temp_ts, ts_j);
(1f) send FINAL_TS(j, tag, temp_ts) to all processes;
(1g) clock = max(clock, temp_ts);
(2) When REVISE_TS(M, j, tag, clk) arrives from P_j:
(2a) priority = max(priority + 1, clk);
(2b) insert (M, tag, j, priority, undeliverable) in temp_Q; // at end of queue
(2c) send PROPOSED_TS(i, j, tag, priority) to P_j;
(3) When FINAL_TS(j, tag, clk) arrives from P_j:
(3a) identify entry Q_entry(tag) in temp_Q corresponding to tag;
(3b) mark tag as deliverable;
(3c) update Q_entry.timestamp to clk and re-sort temp_Q based on the timestamp field;
(3d) if head(temp_Q) = Q_entry(tag) then
(3e) move Q_entry(tag) from temp_Q to delivery_Q;
(3f) while head(temp_Q) is deliverable do
(3g) move head(temp_Q) from temp_Q to delivery_Q;
(4) When P_i removes a message (M, tag, j, ts, deliverable) from head(delivery_Q):
(4a) clock = max(clock, ts) + 1.

```

So, this particular code I have already explained and let us go through the illustrative example for the working or the execution of that 3 phase algorithm.

(Refer Slide Time: 29:37)

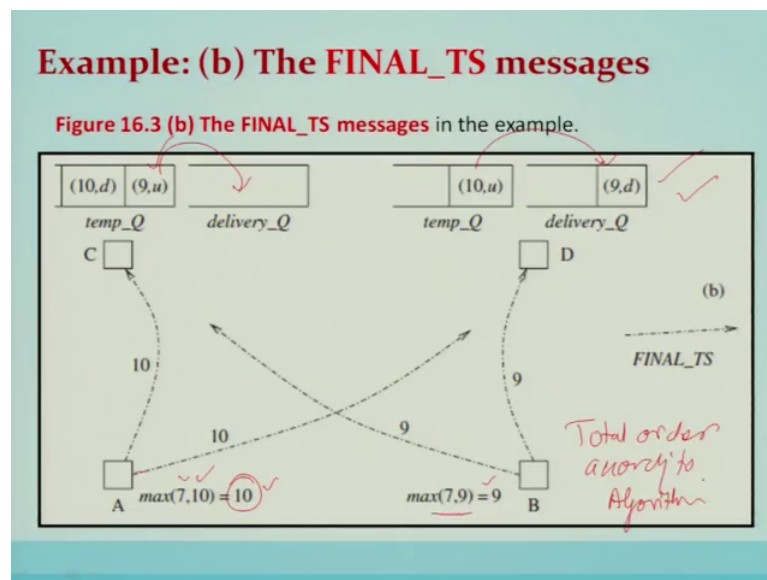


Now, here you can see that side A and side B they want to send a message to those 2 replicas C and D. So, A will send with a tag 7 the message to D and B will send with a tag 9 to C. Now D when it receives this particular message with a tag 7 it will give it its own timestamp and position it in the Q that is called temporary Q it will

not be delivered and similarly first message D will receive the first message from B, it will be queued with the timestamp and the next message 7 minute delivered.

So, time this time will be incremented and will be given to this particular message at time ten. So, on the head of the Q will be the message which comes from B, it will not be delivered it will be just queued this particular timing which is basically the revised time or a proposed time will be sent back from a to B in the second phase. Similarly the 9 tag or a timestamp 9 is sent from D to B. Similarly here 7 will be received first, it will be n queued on the top of the Q and 7 will be returned back to A and once 9 will be received. So, 9 will be in the order, 9 stamp will be returned back.

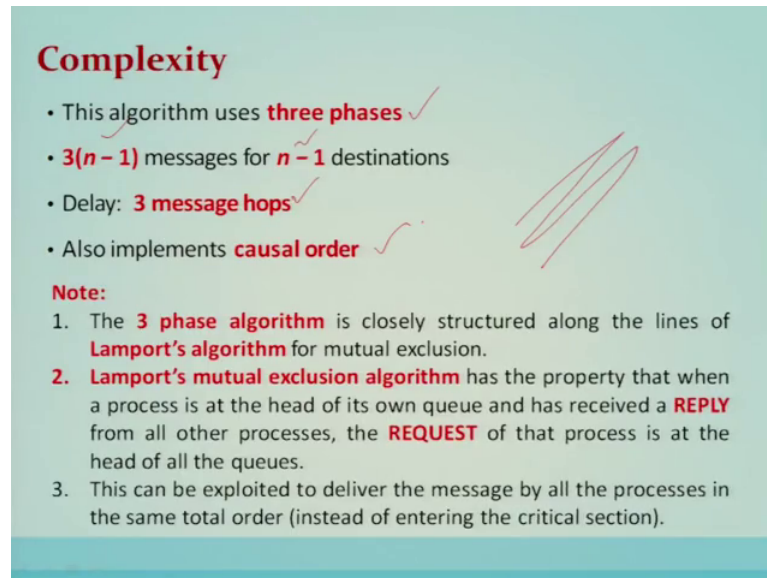
(Refer Slide Time: 31:34)



Now, it will process them, after receiving these values as 7 and 10 it will take the maximum over here and this will be the final value and a will send the final value 10 to both of them similarly here after receiving 7 and 9 2 values the maximum is 9 and 9 will be sent back to both of them. So, if you see here in this scenario mine is at the head of the queue. So, from temporary it will be put on the delivery of the queue and this message will be delivered to the process.

Here also the same ordering will be done since 9 is at the head of the queue it will be put on the delivery and 9 will be delivered. So, first 9 will be delivered at both the ends and then 10 will be delivered. So, hence it follows the total order according to the algorithm.

(Refer Slide Time: 32:36)



Complexity

- This algorithm uses **three phases** ✓
- **$3(n - 1)$** messages for **$n - 1$** destinations ✓
- Delay: **3 message hops** ✓
- Also implements **causal order** ✓

Note:

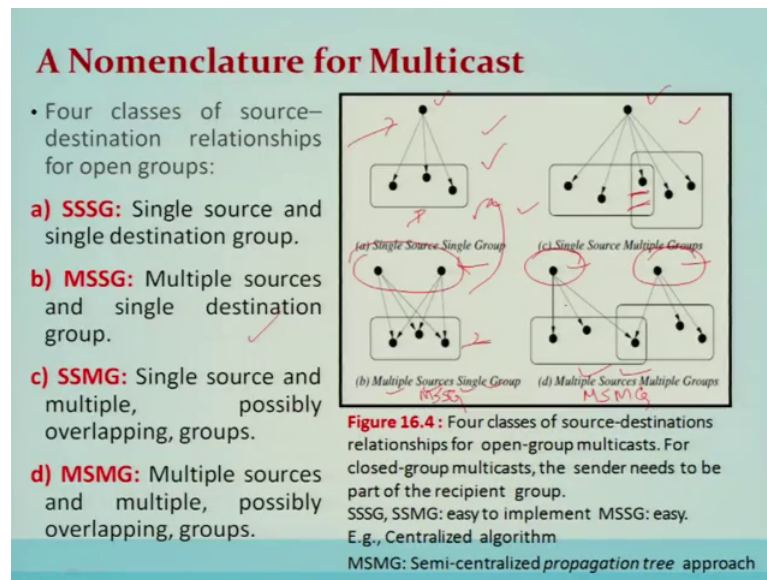
1. The **3 phase algorithm** is closely structured along the lines of **Lamport's algorithm** for mutual exclusion.
2. **Lamport's mutual exclusion algorithm** has the property that when a process is at the head of its own queue and has received a **REPLY** from all other processes, the **REQUEST** of that process is at the head of all the queues.
3. This can be exploited to deliver the message by all the processes in the same total order (instead of entering the critical section).

So, all these steps of the algorithms are basically I have explained you through the example the Complexity of this algorithm since it uses 3 phases and every phases a message is being exchanged.

So, in every phase n minus one messages are exchanged. So, out of 3 phases $3n$ minus one message will be exchanged delay will be 3 message hops because 3 times the message will go and come back. So, it will be 3 message hop delay will be there it also implements casual order that we have already seen. So, 3 phase algorithm is closely structured along the lines of Lamport's clock for the mutual exclusion that algorithm we have studied.

So, Lamport's mutual exclusion algorithm has the property that when a process is at the head of it is own queue and as it received a reply from all the process requests of that process the head of all the queues this can be exploited to deliver the message by all the process in the same order instead of entering into a critical section. So, exactly on the same lines this particular algorithm was designed and is structured.

(Refer Slide Time: 33:33)



So, we have seen the algorithm for total order and that will be achieved through the through the algorithm why because underlying network is an asynchronous network which we are now assuming. So, the ordering which is required from the application is to be ensured through that algorithm which we have just covered.

Now the next topic is about the multi casting. So, as I told you that unicasting multi casting and broadcasting there are 3 different ways of communication in a for the applications in a distributed system which is being utilized.

So, now we are talking about the multi casting that is the communication to a to a group of processes. So, 4 classes of source destination relations for the open groups are there. So, that previous algorithm was for closed group here this example is basically or this particular nomenclature is for the open groups. So, the first nomenclature for the open group is triple S G that is single source and single destination group where there is a single source and the single group is for the destination that is called triple S G.

MSSG is basically multiple sources here you see there are multiple sources and a single destination called multiple sources single destination M double SG then comes triple s MG means single source and multiple groups. So, here these multiple groups are overlapping also and then MSMG that is multiple sources here you see 2 sources are there and multiple groups which are overlapping also. So, there are 4 different ways in which these open groups can be organized or can be classified.

Now we can see that before we go ahead let us see that it is quite easy to implement triple S G why because this can be done using the centralized approach. So, this will be a coordinator for example, this will send to a coordinator and coordinator will basically sequence the messages in the order. So, this is quite trivial this also can be handled using the centralized approach and MSSG can also be handled through that approach why because these 2 centers are there and they can be converted into a single source single group communication why because these 2 sources can send to the coordinator and coordinator intern will send to one group that also can be handled, multiple sources and multiple groups is very difficult it cannot be straightaway implemented in triple SG way.

(Refer Slide Time: 37:17)

✓

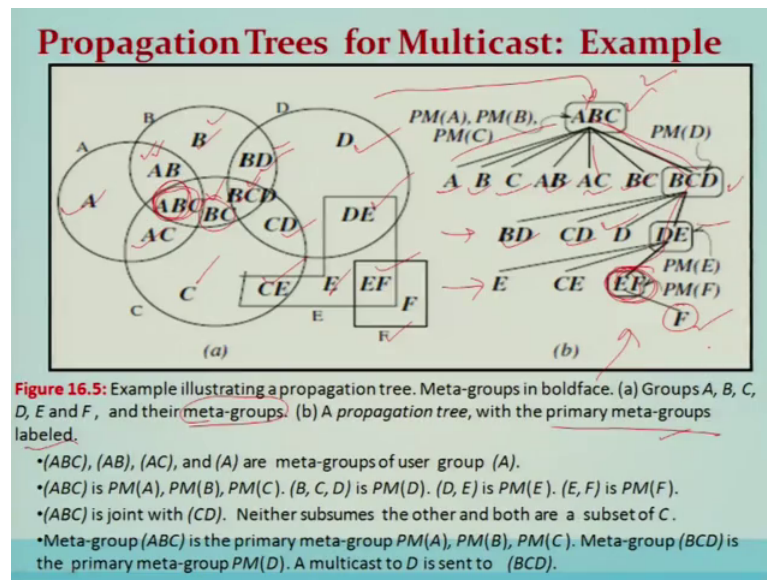
Propagation Trees for Multicast: Definitions

(MS MG)

- Set of **groups** $G = \{G_1 \dots G_g\}$
- Set of **meta-groups** $MG = \{MG_1, \dots, MG_h\}$ with the following properties:
 - Each process belongs to a single meta-group, and has the exact same group membership as every other process in that meta-group.
 - No other process outside that meta-group has that exact group membership.
- MSMG to groups \rightarrow MSSG to meta-groups
- A **distinguished node** in each meta-group acts as its **manager**.
- For each user group G_i , one of its meta-groups is chosen to be its **primary meta-group** (PM), denoted $PM(G_i)$.
- All meta-groups are organized in a **propagation forest/tree** satisfying:
 - For user group G_i , $PM(G_i)$ is at the lowest possible level (i.e., farthest from root) of the tree such that all meta-groups whose destinations contain any nodes of G_i belong to subtree rooted at $PM(G_i)$.

So, now we are going to see the implementation for multiple source and multiple group. One way to implement multiple source and multiple groups is through the propagation trees. So, propagation trees for multi casting are going to basically solve the purpose.

(Refer Slide Time: 37:46)



So, let us see how the propagation trees are constructed if this is the set of nodes. So, we are forming the Meta-group out of these groups and these Meta-groups can be organized in the form of a tree and this is called a propagation tree.

So, in this example illustrating the proposition tree Meta-groups are shown in the bold phases these are all Meta-groups. So, you can see that ABC is a primary Meta-group. So, primary Meta-groups will basically have group AB AC A then B and then C and then BC also and BCD also. So, this particular ABC will become the primary Meta-group and becomes the root of all those Meta-groups.

Now, in turn BCD again will become the primary Meta-group for these Meta-groups means BCDS here. So, BCD will have BD then CD and BC is also absorbed already absorbed in ABC it will not be reflected over here and D and DE. Now as far as the D will become the primary Meta-group for E so the Meta-group for the primary Meta-group DE will be E then CE and EF now EF will become the primary Meta group for F.

So, just see that these complete groups of nodes they are organized in a form of the propagation tree implementation of a propagation tree will now becoming can be easily handled. So, basically in this once the propagation tree is there and you want to send the messages to a particular or in a multicast way. So, first messages are sent to the primary Meta-group node and that in turn will communicate to it is Meta-groups and if it is has to reach to a Meta group F. So, it has to traverse through all the primary Meta-groups till

that meta-group which contains that primary meta-group which contains that meta-group reaches to that point there it will be delivered in that particular tree fashion.

(Refer Slide Time: 40:42)

Classification of Application-Level Multicast Algorithms

Many multicast protocols have been developed and deployed, but they can all be classified as belonging to one of the following five classes.

1. Communication history-based algorithms:

- Use a part of the **communication history** to guarantee ordering requirements.
- E.g. RST algorithm
- Do not need to track separate groups
- Work for open-group multicasts.

2. Privilege-based:

- Token-holder multicasts
- Processes deliver msgs in order of **sequence no.**
- Typically closed groups, and CO & TO.
- E.g., Totem, On-demand.

(a) Privilege-based

Now, there are various classification of application level multicast algorithms. So, first there are 4 different class of multi class algorithm first one is called privilege based algorithm in privilege based algorithm the token rotates. So, the node which is having the token will be allowed to send to the destination. So, the process delivers message in the order of the sequence numbers typically the closed groups and casual order and total order will be there are 2 algorithms which are implemented based on this particular classification to totem and on demand multicast algorithms are available.

(Refer Slide Time: 41:33)

Contd...

3. Moving sequencer: ✓

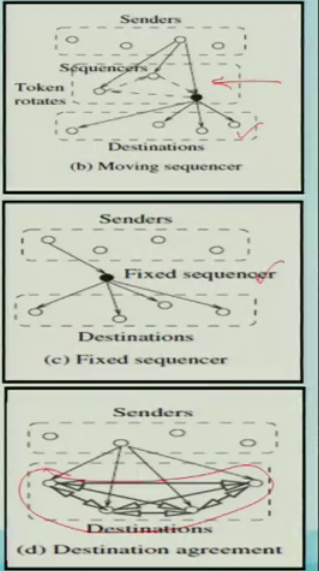
- Sequencers' token has **sequence no** and list of messages for which **sequence no** has been assigned (these are sent msgs).
- On receiving token, sequencer assigns **sequence no's** to received but unsequenced messages, and sends the newly sequenced messages to destinations.
- Destinations deliver in order of **sequence no**
- E.g., Chang-Maxemchuck, Pinwheel

4. Fixed sequencer: ✓

Simplifies moving sequencer approach.
E.g., propagation tree, ISIS, Amoeba, Phoenix

5. Destination agreement: ✓

- Destinations receive limited ordering info.
- Timestamp-based (**Lamport's 3-phase**)
- Agreement-based, among destinations.



The diagrams illustrate three distributed algorithms:

- (b) Moving sequencer: A central node labeled 'Sequencers' is connected to multiple 'Senders' and 'Destinations'. A red arrow indicates the 'Token rotates' from the sequencer to a sender.
- (c) Fixed sequencer: A central node labeled 'Fixed sequencer' is connected to multiple 'Senders' and 'Destinations'.
- (d) Destination agreement: A network of nodes is shown with red arrows indicating communication between destinations.

The next one is called moving sequencer. So, here there will be a sequencer nodes are sending the message to the sequencers. So, sequencers token has the sequence numbers and list of the messages for which the sequence number has been assigned on receiving the token the sequencer assigns the sequence number to the received, but unsequenced messages and sends the newly sequenced messages to the destination.

Another next one is called a fixed sequencer fixed sequencer is like centralized approach that we have already seen centralized approach means there is a one single coordinator and fifth one is called destination agreement. So, in destination agreement the ordering will be done among the destination. So, destination received the limited ordering information and using time based Lamports 3 phase algorithm the agreement will be evolved.

(Refer Slide Time: 42:44)

Few Other Algorithms for Message Ordering and Group Communication	
Author	Algorithm
M. Raynal et al. (1991)	Algorithm for causal order
Kshemkalyani and Singhal (1996,1998)	The space and time optimal algorithm for causal order
K. Birman and T. Joseph (1987)	The algorithm for total order
Garcia-Molina and Spauster (1991) X. Jia (1995), and Chiu and Hsiao (1998)	The algorithm for total order using propagation
Chang and Maxemchuk (1984)	The moving sequencer algorithms
P. Ezhilchelvan et al. (1995)	An efficient fault-tolerant group communication protocol
E. Gilbert and H. Pollack (1968)	Steiner minimal trees
L. Kou et al.	A fast algorithm for Steiner trees
Kompella et al. (1993)	Delay-bounded minimum Steiner
Hadzilacos and Toueg (1993)	Semantics of fault-tolerant group communication
Ballardie et al. (1993)	Core-based trees

Few other algorithms for message ordering and group communication are available in the literatures that are listed over here Conclusion.

(Refer Slide Time: 42:53)

Conclusion

- **Inter-process communication via message-passing** is at the core of any distributed system.
- In this lecture, we have discussed **non-FIFO, FIFO, causal order, asynchronous order and synchronous communication paradigms** for ordering messages.
- Then examined several algorithms to implement these orderings.
- **Group communication** is an important aspect of communication in distributed systems.
- **Causal order and total order** are the popular forms of ordering when doing group multicasts and broadcasts. Algorithms to implement these orderings in group communication were also discussed.
- Then we have explained the **propagation trees for multicast** and classification of application-level multicast algorithms.
- In upcoming lecture, we will discuss about '**Self-Stabilization**'.

Inter process communication via message passing is at the core of any distributed system in this lecture we have discussed non-FIFO, FIFO causal order, asynchronous order, synchronous communication paradigms for message ordering, then examine several algorithms to implement these orderings group communication is the important aspect of communication in a distributed system.

Causal order and total order are the popular forms of ordering when doing the group multi casting and broadcasting then we have explained the propagation trees for multicast algorithm and classification of application level multicast algorithms in the upcoming lectures we will discuss about self stabilization.

Thank you.