

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

**Lecture - 18**  
**Case Studies**  
**Randomized Distributed Algorithm**

Randomized Distributed Algorithm.

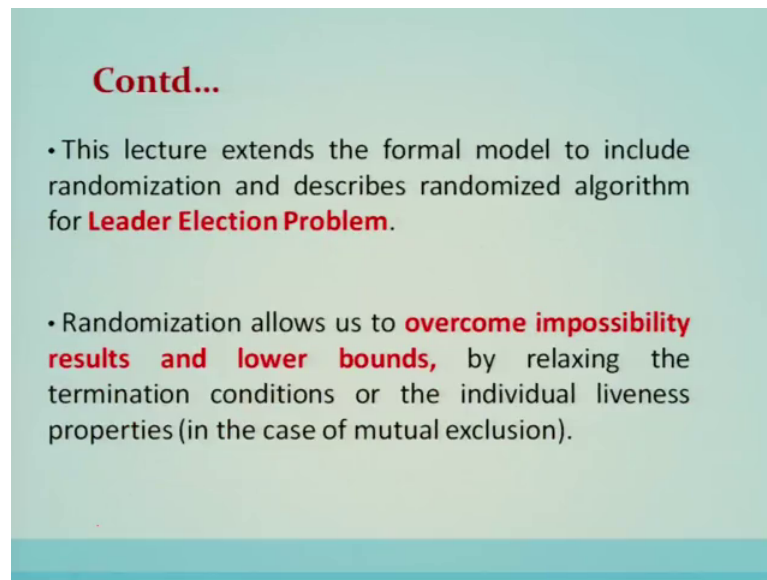
(Refer Slide Time: 00:18)

### Introduction

- This lecture focused on a specific type of distributed algorithms, which employ **randomization**.
- Randomization has **proved to be a very powerful tool** for designing distributed algorithms (as for many other areas).
- It often simplifies algorithms and more importantly **allows us to solve problems in situations where they cannot be solved by deterministic algorithms**, or with fewer resources than the best deterministic algorithm.

Introduction: this lecture focuses on specific type of distributed algorithm, which employ randomization. Randomization has proved to be a very powerful tool from designing distributed algorithms, as far as many other areas also. It often simplifies algorithms and more importantly allows us to solve the problems in situations, where they cannot be solved by the deterministic algorithms, or with the fewer resources then the best deterministic algorithm ever solves.

(Refer Slide Time: 00:57)



**Contd...**

- This lecture extends the formal model to include randomization and describes randomized algorithm for **Leader Election Problem**.
- Randomization allows us to **overcome impossibility results and lower bounds**, by relaxing the termination conditions or the individual liveness properties (in the case of mutual exclusion).

So, in this particular discussion or a lecture, we will see the power of randomization. It basically is used with the distributed algorithm. Then it is going to solve some of the impossible to results also, but how they are going to be solved, with the same problem statement or with a modified problem statement that we are going to see.

So, randomization if it is must with, or it is given as a information to the distributed algorithm, then we are going to see the intricacies in this part of the lecture. So, this lecture extends the formal model to include the randomization, and describes the randomized algorithm for leader election problem. So, leader election problem will be a case study, where we will use the randomized approach or randomization, and we will basically discuss the randomized algorithm for leader election problem. Randomization allows us to overcome impossibility results, and lower bounds by relaxing the termination conditions or the individual liveness properties.

(Refer Slide Time: 02:12)

## Weakening the Problem Definition

- A **randomized** algorithm is an algorithm that has access to some source of random information, such as that provided by flipping a coin or rolling dice.

More formally:

- **Extend the transition function** of a processor to take as an additional input a random number, drawn from a bounded range under some fixed distribution.

Distributed mutual exclusion algorithm; a case study. Now the problem definition, we will basically see that, the same problem definition which is impossible. Now we will take up, it is weakening the problem definition, and let us see that how using randomization we are going to solve it. So, a randomized algorithm is an algorithm that has access to some source of random information; such as that provided by flipping a coin or rolling a dice. So, more formally extend the transition function of a processor to take as an additional input, a random number, drawn from a bounded range, under some fixed distribution.

(Refer Slide Time: 02:55)

## Why is it important?

### The Bad News:

1. The addition of random information alone typically will not affect the existence of **impossibility results or worst-case bounds**.
2. For instance, even if processors have access to random numbers, they will not be able to elect a leader in an anonymous ring or solve consensus in fewer than  **$f+1$  rounds** in all (admissible) executions.

So, what is important here is that, the addition of random information alone, typically will not affect the existence of impossibility, or the worst case bounds. So, for instance, even if the processors have access to the random numbers, they will not be able to elect the leader in an anonymous ring, or solve the consensus problem, fewer than  $f + 1$  rounds in all admissible executions.

(Refer Slide Time: 03:27)

**Contd...**

**The Good News:**

- **randomization + weakening** of problem statement is a powerful tool for overcoming limitations.
- Usually the **weakening** involves the termination condition (for instance, a leader must be elected with a certain probability) while the other conditions are not changed (for instance, it should never be the case that two leaders are elected).

*New Problem Statement*

*leader must be elected with certain probability*

So, randomization with weakening of the problem statement becomes a powerful tool, to overcome these limitations which are basically proved using the impossibility results, and also the lower bounds. So, usually weakening involves the termination condition. For instance leader election must be elected with a certain probability. This becomes a weakening of the problem statement.

So, a leader must be elected with a certain probability. So, this becomes a problem statement, or a new problem statement, and this we will see how using randomization we are going to solve. So, randomization and weakening of the problem statement together, will solve the purpose, or will solve the impossibilities and the bounds which are being proven.

(Refer Slide Time: 04:43)

### **Differs from a average case analysis of a deterministic algorithm**

•In **average case analysis**, there are several choices as to what is being averaged over. One natural choice is the inputs. There are two difficulties with this approach:

- (i) Determining an accurate probability distribution on the inputs is often not practical.
- (ii) Another drawback is even if such distributions can be chosen with some degree of confidence, very little is guaranteed about the behavior of the algorithm on a particular input. For instance, even if the **average** running time over all inputs is determined to be small, there still could be some inputs for which the running time is enormous.

Now this randomization is differs from the average case analysis of the deterministic algorithm. In average case analysis, there are several choices as to what is being averaged over; one natural choice is the inputs.

So, there are two difficulties, with this averaging issue. First is determining an accurate probability distribution on input is often not practical. Another drawback is, even if such distributions can be chosen with some degree of confidence, very little guarantee about the behavior of the algorithm on a particular input. For instance, even if the average running time over all the input is, determined to be a small, there still could be some input for which the running time is enormous.

(Refer Slide Time: 05:35)

### Contd...

- In the randomized approach, more stringent guarantees can be made. Because the random numbers introduce another dimension of variability even for the same inputs there are many different executions for the same input.
- A good randomized algorithm will guarantee good performance with some probability for each individual input.
- Typically the performance of a randomized algorithm is defined to be the **worst-case probability over all inputs**.

In the randomized approach, more stringent guarantees can be made, because the random numbers introduce another dimension of variability, even for the same inputs. There are many different executions for the same input. A good randomized algorithm will guarantee a good performance, with some probability for each individual input. Typically the performance of the randomized algorithms defined to be the worst case probability over all inputs

(Refer Slide Time: 06:09)

### Randomized Leader Election

*Example Anonymous Ring*

- The simplest use of randomization is **to create initial asymmetry in situations that are inherently symmetric**.
- One such situation is anonymous rings, where processors do not have distinct identifiers, it is **impossible to elect a unique leader**.

**(From Theorem: There is no nonuniform anonymous algorithm for leader election in synchronous rings)**

- This impossibility results holds even for randomized algorithms. However a randomized algorithm formulation of LE "leader is elected **with some probability**".
- A variant of the leader election problem that relaxes the condition that eventually a leader must be elected in every admissible execution. Relaxed version of the leader election problem requires:

**Safety:** In every configuration of every admissible execution, at most one processor is in an elected state

**Liveness:** At least one processor is elected with some non-zero probability

Now, randomized leader election problem; the simplest use of randomization is to create an initial asymmetry, in the situations, that are inherently symmetric. For example, if we say an anonymous ring; so anonymous ring, where the ids are not uniquely assigned. So, they become anonymous rings, and ids are not distinct. Now we have seen the impossibility result, that there is no non uniform anonymous algorithm for a leader election in a synchronous ring.

So, that impossibility result reminds that, if let us say that, the nodes are not are having the distinct ids, then basically this will create the problem of the symmetry, and in that symmetry the state machines of all the processes will be the same in every execution. Hence the leader will not be elected in that case. In spite of the non uniform, in spite of non uniform; that means, in spite of knowing how many nodes are there in the ring.

. So, the simplest use of randomization is to create the initial asymmetry in the situations that are inherently symmetric, that example we have seen. So, one such situation is anonymous ring, where the processors do not have distinct ids. So, it is impossible to elect a leader in this particular problem setting. So, it is said that, it is impossible to elect the leader in this problem setting. Now from the earlier theorems, we have also seen that there is no non uniform anonymous leader election algorithm in a synchronous ring. This particular impossibilities are also holds for the randomized algorithms; however, the randomized algorithm with the modified formulation of a leader election; that is the leader is elected with some probability, with this weakening of the problem statement.

Now, let us see that whether this particular weakening of the problem statement, and if we add the randomization, whether is it going to solve this particular impossibility result, and give up with the randomized leader election problem. So, a variant of leader election problem that relaxes, that condition, that eventually a leader must be elected in every admissible execution. So, relaxed version of a leader election problem requires two properties to be to be satisfied.

The first is called safety property. Safety property is says that in every configuration of, every admissible execution, at most one processor is in the elected state. Meaning to say that, in no situation or in no admissible execution, there can be more than one leader elected in an algorithm, then it will become the wrong or it will violate the safety

conditions. The safety conditions ensure that in every execution, or every admissible execution, the algorithm produces at most one leader.

The other condition is called the liveness condition. So, liveness condition is says that, at least one processor is elected with some nonzero probability. Here we have included the weakening of the problem statement, in that weakening is affected in the liveness condition. So, liveness says that at least one processor is elected with nonzero probability, the leader. So, leader is now the modified, liveness condition say the leader is elected with some probability.

(Refer Slide Time: 10:38)

**Contd...**

- **Safety property has to hold with certainty:** that is, the algorithm should never elect two leaders. ✓ (or more than one leader)
- **Liveness condition is relaxed,** and the algorithm **need not always terminate with a leader,** rather, it is required to do so **with nonzero probability.**
- An algorithm that satisfies this weakened liveness condition can fail to elect a leader either by not terminating at all or by terminating without a leader.

So, safety property has to hold with certainty as I explained; that is the algorithm should never elect two leaders or more than one leader, and this is the safety property.

So, there is no weakening in the safety property. Safety property has to hold with the certainty as we have seen earlier. The second property called liveness property, liveness condition is relaxed, and the algorithm need not always terminate with a leader, rather it is required to do so with the nonzero probability. So, an algorithm that satisfies this weakened liveness condition can fail to elect leader, either by not terminating at all or by terminating without electing a leader.



(Refer Slide Time: 11:38)

### 1. Synchronous One-Shot Algorithm

- First, let us consider **synchronous rings**.
- There is only one admissible execution on an anonymous ring for a deterministic algorithm.
- For a randomized algorithm, however, there can be many different executions, depending on the random choices.
- The approach that is used to devising a randomized leader election algorithm is to use randomization **to create asymmetry by having processors choose random pseudo-identifiers**, drawn from some range, and then execute a deterministic leader election algorithm.

*random pseudo-identifiers [1, 2] ← ✓*

So, we are going to take up this particular issue, and we will see algorithm designed with this weakened the liveness condition to elect a leader, using the randomized approach. And we will see also the remedy that if the leader is not at all elected at the termination of the algorithm, how are we going to basically resolve this.

So, the first algorithm which we are going to discuss is the synchronous one shot algorithm, and this is the randomized algorithm. So, let us first consider the synchronous ring. So, in that synchronous ring, there is only one admissible execution on this anonymous ring for the deterministic algorithm. Now for a randomized algorithm; however, there can be many different executions, depending on the random choices.

So, as you know that in anonymous ring for a deterministic algorithm, there is only one admissible execution and where all the states are the same. The state transitions are the same. Hence the there is no asymmetry; hence the leader cannot be elected, but for randomized algorithms here, there are many different executions why, because that depending on the random choices of the id. So, asymmetry can be introduced with some probability in the randomized algorithm.

So, the approach that is used to devise the randomized leader election is to use the randomization to create asymmetry, by having processors choose a random pseudo identifiers, drawn from some range, and then execute a deterministic leader election problem. So, in this problem setting you will see that, range of values is from 1 and 2;

that means, the pseudo random, pseudo randoms, pseudo identifiers, are from the set of only two numbers; 1 and 2, and then execute a deterministic leader election, and let us see that if this particular choosing the number out of these twos range of numbers 1 and 2. Whether the deterministic leader election problem can tap this asymmetry, and elect a leader, or it is fail to elect a leader, because of the symmetry.

(Refer Slide Time: 14:36)

**Properties**

- A simple deterministic leader election algorithm with these properties is the following:

1. Each processor sends a message around the ring to collect all pseudo-identifiers.
2. When the message returns (after collecting ***n pseudo-identifiers***), a processor knows whether it is a unique maximum or not.

✓

Hand-drawn diagram: A ring of nodes with arrows indicating message flow. A node is labeled 'toss coin' and 'id'. A box labeled 'send id' is shown above the ring.

So, a deterministic leader election with these properties is the following.

So, each processor sends a message around the ring, to collect all the pseudo identifiers. Let us assume this particular ring. So, here every processor will send its id, which is drawn out of a random range of numbers. Here we are considering only 1 and 2. So, this particular process will toss a coin, to select one of these two numbers, or randomly pick one of them out of these range of numbers given, and that will become its id, and this id will be send on the left part of the edge. The next process connected by an edge will receive this from the right side of the ring. So, when the message; basically it will collect. So, this particular message will collect the pseudo identifiers, or it will add its own id and that will circulate around the ring.

So, after  $n$  rounds, if  $n$  numbers of processes are there; so  $n$  rounds, then all the processors will know the ids which is being selected here by the processors, and the one which is having the highest id will be the elected as the leader. So, when the message

returns after collecting  $n$  pseudo to identify the processor knows whether it is unique maximum or not.

(Refer Slide Time: 16:28)

### Algorithm: Overview

$[1, 2]$   
 $P(2) = \frac{1}{n}$   
 $P(1) = 1 - \frac{1}{n}$   
*by n processors*

- 1. In this algorithm, the **pseudo-identifier is chosen to be 2 with probability  $\frac{1}{n}$  and 1 with probability  $(1 - \frac{1}{n})$** , where  $n$  is the number of processors on the ring.
- 2. Thus each processor makes use of its source of randomness exactly once, and the random numbers are drawn from **the range  $[1...2]$** .
- The set of all possible admissible executions of this algorithm for fixed ring size  $n$  contains exactly one execution for each element of the set  $\mathcal{R} = \{1, 2\}^n$   $exec(R)$
- That is, by specifying which random number, 1 or 2, is obtained by each of the  $n$  processors in its first step, we have completely determined the execution. Given an element  $R$  of  $\mathcal{R}$  then the corresponding execution will be denoted by  **$exec(R)$**

Now, coming the algorithm with that idea, the algorithm goes like this in this algorithm the pseudo identifier is chosen, to be two with the probability  $\frac{1}{n}$ , and the number 1 is chosen with the probability  $1 - \frac{1}{n}$ . So, the range of numbers is only two numbers. So, ids will be chosen by  $n$  different processors. So, the probability of choosing to, let us say is  $\frac{1}{n}$ , is the probability of choosing number 2. Similarly the probability of choosing number 1 will become  $1 - \frac{1}{n}$ . So,  $n$  is basically the number of process in the ring.

So, this becomes the first step of the algorithm, thus each processor makes use of its source of randomness exactly once, and the random numbers are drawn from this range that I explained. The set of all possible admissible executions of this algorithm for the fixed ring of size  $n$ , will contain exactly one execution for each element of this set, which will basically be nothing, but a vector, which vector of size  $n$ , which will be containing the elements either 1 or 2, and that is represented by a bigger set. That is by specifying which random number 1 or 2 is obtained by each of the  $n$  processor in its first step, we have completely determined the execution.

So, given an element  $R$  of  $\mathcal{R}$ , then the corresponding execution will be denoted by execution of  $R$ .

(Refer Slide Time: 18:43)

### Algorithm 1

**Algorithm 1** Randomized leader election in an anonymous ring:  
code for processor  $p_i, 0 \leq i \leq n - 1$ .

- 1: initially // spontaneously or upon receiving the first message
- 2:  $id_i := \begin{cases} 1 & \text{with probability } 1 - \frac{1}{n} \\ 2 & \text{with probability } \frac{1}{n} \end{cases}$  // choose pseudo-identifier
- 3: send  $\langle id_i \rangle$  to left
- 4: upon receiving  $\langle S \rangle$  from right
- 5: if  $|S| = n$  then // your message is back
- 6: if  $id_i$  is the unique maximum of  $S$  then become *elected* // the leader
- 7: else become *non-elected* // a nonleader
- 8: else // concatenate your id to the message and forward
- 9: send  $\langle S \cdot id_i \rangle$  to left

So; that means, in a particular execution, we can admissible execution, we can denote it by  $x \in R$  of that particular instance which will. So, the algorithm, which is called leader election algorithm using randomization in an anonymous ring for a particular process  $p_i$ , and this particular algorithm will be implementing or will be running on all processors; that is ranging from 1 to  $n - 1$ . So, if there are  $n$  processors. So, this kind of this algorithm will run on each node of the topology, which is organized in form of a ring.

. So, initially spontaneously or upon receiving the first message, the ids will be picked; that means, a processor will pick an id 1 with the probability one minus  $\frac{1}{n}$ , and id 2 with the probability  $\frac{1}{n}$ . So, it will pick a number 2 with a probability  $\frac{1}{n}$ . So, then after picking one of these numbers randomly, then it will send this number as an id of a particular process to the left of the ring, to the left, means this id will be sent, id of  $i$  will be sent to the left. Upon receiving this particular message  $S$  from the right, this particular process will check if the size of this  $S$  is equal to  $n$ .

That means, this particular message is already flown  $n$  different times in the ring, and  $n$  are the number of process; that means, it has collected all ids of  $n$  different processors. So, if it is  $n$ , then the id, if  $id_i$  is the unique maximum of  $S$ , then it will be elected, that node will be elected, else the node becomes non elected. So, if that particular id is not maximum of all the other  $n$ s, then the node will be non elected. If it is not circulated  $n$

times, then it will send this particular id affixing its id. So, it will contain id i and id j, and this message will be sent to the left and so on.

So, this way this particular size will grow, when it will reach over here, it will contain all the n ids listed in that particular message, and by that it will check whether its id is maximum among all the, among unique maximum, not only maximum, but it should be a unique maximum. That means, if let us say 2 nodes are picking id as 2. So, there cannot be 2 leader elected. Obviously, there will not be any leader elected. So, the leader will be elected only in one case, when 1 node will have the id 2, and all other nodes have the id 1. So, this is only one case, when the leader is elected.

(Refer Slide Time: 22:15)

**Definition**

- Let  $P$  be some predicate on executions, for example, at least one leader is elected. ✓
- Then  $\Pr[P]$ : probability of the event ✓

$$\{R \in \mathfrak{R} : \text{exec}(R) \text{ satisfies } P\}$$

So, definition let  $P$  be the predicate on executions for example, at least one leader is elected, then the probability  $P$  of that event is nothing, but an execution that satisfies that particular predicate. Analysis, what is the probability that the algorithm terminates with a leader.

(Refer Slide Time: 22:33)

### Analysis

- **What is the probability that the algorithm terminates with a leader?**
- This happens when a single processor has the maximum identifier, 2. ✓
- The probability that a single processor draws 2 is the probability that  $n-1$  processors draw 1, and one processor draws 2, times the number of possible choices for the processor drawing 2, that is,

$$\binom{n}{1} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} = \binom{n}{1} \frac{1}{n} \left(\frac{n-1}{n}\right)^{n-1} = \frac{n!}{1!(n-1)!} \frac{1}{n} \left(\frac{n-1}{n}\right)^{n-1}$$

$\xrightarrow{\text{large } n} \frac{1}{e}$

So, this happens when a single processor has the maximum id 2, and; that means, only one processor has the maximum id 2, and all other processors are having ids 1, then only the leader will be elected. The probability that a single processor draws 2, is the probability that  $n$  minus 1 processor draw 1, and 1 processor draws 2, times the number of possible choices of the processor drawing to that is. That means, only one processor is draws to that probability is 1 by  $n$ , and all other processors  $n$  minus 1 processors are drawing 1, that probability is 1 minus  $n$  and  $n$  minus 1 processors will draw, and this kind of, this how many times is possible choices, a processor drawing 2 is basically  $n c 1$ .

So, if you simplify, then this particular part of this particular formula will cancel  $n$  by  $n$ . So, it becomes 1 minus 1 by  $n$  minus 1. This is the probability that the algorithm terminates with a particular leader. Now we will see that, what is the probability that all  $n$ . So, what is the probability if we take the difference? So, we will see that this is having a higher probability than this particular probability, where none of them has chosen 2. so; that means, this particular probability of termination, is converging, if the number of nodes is basically going to be a large value. So, it will converge to a value which is 1 upon  $e$ . So, this is a constant. So, the probability that the algorithm terminates is a constant, and that is 1 by  $e$  and that is shown in the next slide.

So, the same thing is 1 by  $e$  that is the probability of choosing of that algorithm terminates with the leader is 1 upon  $e$ .

(Refer Slide Time: 25:09)

**Contd...**

- It is simple to show that every processor terminated after sending exactly  $n$  messages; moreover, at most one processor terminates in an elected state. ✓
- In some executions, for example, when two processors choose the pseudo-identifier 2, no processor terminates as a leader. However, the above analysis shows that this happens with probability less than  $1 - 1/e$

$1 - 1/e$

Now, it is simple show that every processor terminates after sending exactly  $n$  messages, moreover at most one processor terminates in an elected state. In some executions; for example, when two processor choose pseudo id 2 of 2, then no processor terminates as the leader why, because safety conditions says that only, there can be only 1 leader than only the algorithm terminates successfully. However, the above analysis shows that this happens with the probability less than 1 minus 1 upon e.

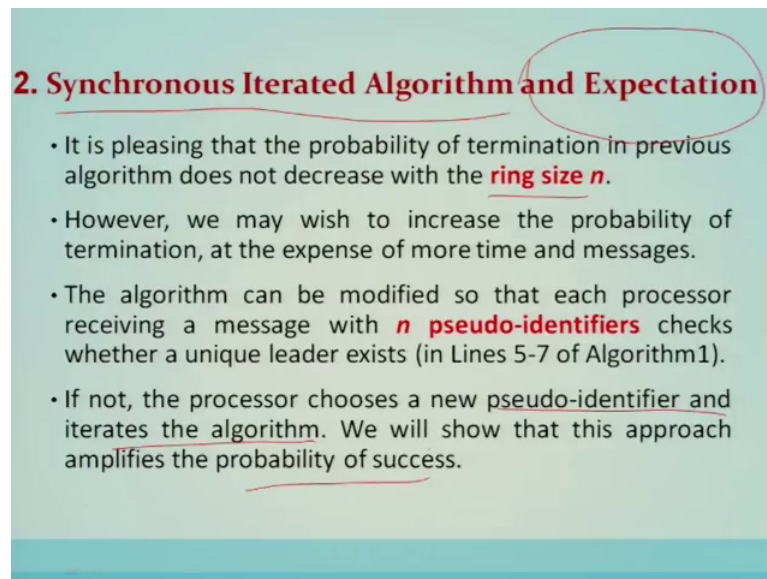
(Refer Slide Time: 25:46)

**Message Complexity**

- **Theorem 1:** *There is a randomized algorithm that, with probability  $c > 1/e$ , elects a leader in a synchronous ring; the algorithm sends  $O(n^2)$  messages.*

The message complexity; so the theorem for this analysis or this algorithm, says that there is a randomized algorithm that with the probability  $c$ , which is greater than  $1/n$ , elects a leader in a synchronous ring and the algorithm, sends  $O(n^2)$  messages, because we have taken the  $O(n^2)$  algorithm, and converted into a randomized algorithm with relaxed live condition.

(Refer Slide Time: 26:14)



## 2. Synchronous Iterated Algorithm and Expectation

- It is pleasing that the probability of termination in previous algorithm does not decrease with the ring size  $n$ .
- However, we may wish to increase the probability of termination, at the expense of more time and messages.
- The algorithm can be modified so that each processor receiving a message with  $n$  pseudo-identifiers checks whether a unique leader exists (in Lines 5-7 of Algorithm1).
- If not, the processor chooses a new pseudo-identifier and iterates the algorithm. We will show that this approach amplifies the probability of success.

Now, we have seen that this particular algorithm which we have just seen, may not terminate with a leader elected, and may end without any leader elected, because if there are two nodes with a maximum value two or more nodes, with a maximum value then leader will not be elected. So, leader will only be elected in a condition, when only one node is having the maximum value, and all other nodes are having the other value elected as non leader. So, we are now going to see the next step forward; that means, we want that algorithm should be terminated after several iterations; that means, this algorithm is to be now iterated, or more than one times or more than one round.

Finally, when the leader, when it elects a leader. So, the synchronous iterated algorithm and the expectations. So, we are now going to discuss the modification of the previous algorithm, or how the previous algorithm is to be iterated over the round. So, that it can elect a leader, and terminate after that. So, then we will be calculating its expectations. So, it is pleasing to know that the probability of termination in the previous algorithm

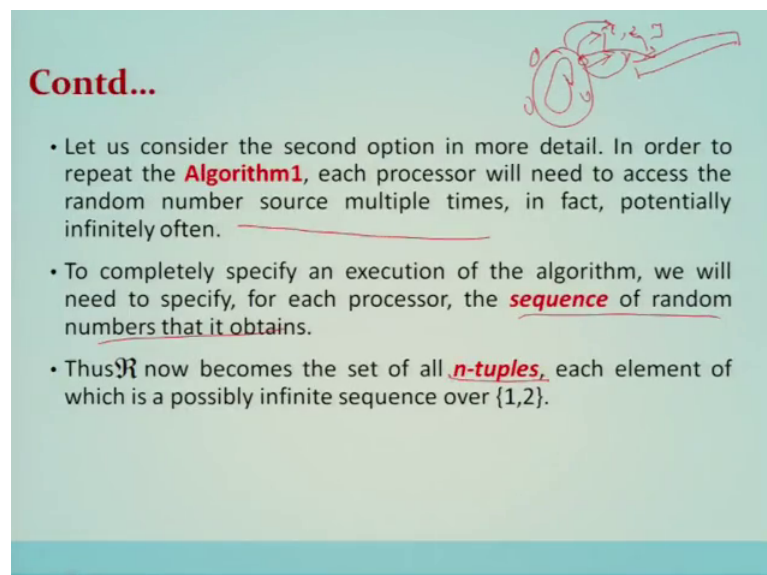


does not decrease with the ring size. So, as we increase the ring size, the probability is not going to decrease. So, hence it is basically a scalable or say good news.

However we may wish to increase the probability of termination at the expense of more time and the messages. The algorithm can be modified so that each processor receiving a message with  $n$  pseudo identifiers, checks whether a unique leader exists or not; that means, in the previous algorithm, if you see at 5 to 7, what we can do is, after seeing the ids which are coming on the message, and if it is found that more than one maximum number that is 2 is present in the message, then no need then basically that iteration or this algorithm should be terminated, because it is not going to elect the leader. So, it is an early detection. That means, this particular 5 to 7 lines, during this lines we can insert a condition, and this algorithm will come out of that current iterations, and it will again reassign the pseudo ids to the node, and restart again; that is called iterative algorithm.

So, we are now discussing about that iterative algorithm, because we are interested that algorithm should terminate with a leader. So, if not, the process of choosing the new pseudo ids and then iterate the algorithm that I have told you. We will now show that this approach amplifies the probability of success.

(Refer Slide Time: 29:25)



**Contd...**

- Let us consider the second option in more detail. In order to repeat the **Algorithm1**, each processor will need to access the random number source multiple times, in fact, potentially infinitely often.
- To completely specify an execution of the algorithm, we will need to specify, for each processor, the **sequence** of random numbers that it obtains.
- Thus  $\mathcal{R}$  now becomes the set of all  **$n$ -tuples**, each element of which is a possibly infinite sequence over  $\{1,2\}$ .

Let us consider the second option in more detail, in order to repeat the algorithm one, each processor will need to access the random number source multiple times. In fact, potentially infinitely often, saying that the random number, the pseudo random id is 1

and 2 in the algorithm number 1, is accessed by the nodes exactly one times, but now when you are iterating. So, this axis of this particular number by the same processor may be a multiple times, because these ids if they are going to change, maybe in some admissible execution a leader is going to be elected. So, that is why there is a modification that, the processors will need to access the random numbers, random number source multiple times, and in fact, potentially infinitely often.

To completely specify the execution of algorithm, we will need to specify for each processor a sequence of random numbers that it obtains. So, in this process every node, after picking up a random number out of this particular range will generate sequence of random numbers, and every processor will obtain this kind of sequence. Let  $R$  becomes be the set of all  $n$  tuples each, element of which is possibly infinite sequence over  $1 \dots 2$ ; that is the range of numbers.

(Refer Slide Time: 31:08)

### Analysis

Terminates in  $k$  iteration =  $(k-1)k$

- For the iterated algorithm, **the probability that the algorithm terminates at the end of the  $k$ th iteration is equal to the probability that the algorithm fails to terminate in the first  $k-1$  iterations and succeeds in terminating in the  $k$ th iteration.**
- The analysis of Algorithm1 shows that the probability of success in a single iterations is  **$c > 1/e$ .**
- Because the probability of success or failure in each iteration is independent, the desired probability is

Probability of Algorithm  $P(k^{th} \text{ iteration})$  =  $(1-c)^{k-1} c$ 
 $(1-c)^{k-1} c$

- This probability tends to 0 as  $k$  tends to  $\infty$ ; thus the probability that the algorithm terminates with an elected leader is 1. ✓

Now, the analysis of this iterative algorithm; the probability that the algorithm terminates at the end of  $k$ -th iteration, is equal to the probability that the algorithm fails to terminate in the first  $k$  minus 1 iteration, and succeed in terminating the  $k$  of iterations. So; that means, if the algorithm terminates in  $k$ -th iteration, means that it is not successful in  $k$ , the previous  $k$  minus 1 iteration, and it is successfully terminating with a leader elected in the  $k$ -th iterations.

So, the analysis of algorithm 1 shows that the probability of success in a single iteration, is  $1 - c$ , because the probability of success or a failure in each iteration is independent, the desired probability of terminating at  $k$ -th iteration is given by this particular formula;  $1 - c$  is basically the probability that, it is not successful in terminating with a leader elected. So,  $1 - c$  and this will repeat for  $k - 1$  times, and then at  $k$  time this is the probability to elect a leader. So, this is the total probability that after  $k$ -th iteration, the leader is elected during the iterative algorithm.

So, this probability tends to 0, as the  $k$  tends to infinity. So, if the  $k$  tends to infinity, this particular probability becomes 0. Thus, the probability that the algorithm terminates with the elected leader is 1. So, this is a good news, that after sequence of rounds or iteration, the algorithm will elect a leader with a probability 1, or definitely elected leader.

(Refer Slide Time: 33:26)

### Time Complexity of Iterated Algorithm

- **Worst-case number of iterations:**  $\infty$  ✓
- **Expected number of iterations:**  $1/c < e$  ✓
- Let  **$T$**  be a **random variable** that, for a given execution, is the value of the complexity measure of interest for that run (for instance, the number of iterations until termination).
- Let  **$E[T]$**  be the **expected value of  $T$** , taken over all  $R \in \mathfrak{R}$
- That is,

$$E[T] = \sum_{x \text{ is a value of } T} x \cdot \Pr[T = x]$$

So, the time complexity of this iterative or iterated algorithm is like this. Now the worst case number of iterations if it is infinite, the expected number of iterations if we see, is  $1/c$  is less than  $e$ , or the value of  $e$  2.7 something, or you can say that the expected number of iterations is 3; that means, within 3 iterations some leader will be elected, and that is not a bad number. Let  $T$  be the random variable that for a given execution, is the value of the complexity measure of interest for that run. For instance the number of iterations until the termination; why we are doing is, because the worst case  $n$  number of iterations is infinite, that particular value is meaningless to us.

So, in this particular iterated algorithm, randomized algorithm, we will take the expected value of the total number of iterations, and then the message complexity will be discussed or will be considered in that terms of average, not in the worst case . So,  $E[T]$  be the expected value of  $T$ , and  $T$  is the random variable for a given execution, is that the way is that the value of the complexity measure of interest for that run. So, that is the number of iterations until the termination.

(Refer Slide Time: 35:02)

**Contd...**

- Note that by the definition of  $\Pr[P]$  above, this is ultimately taking probabilities over  $\mathfrak{R}$
- With this definition, we have the following theorem:
- **Theorem 2:** *There is a randomized algorithm that elects a leader in a synchronous ring with probability 1 in  $(1/c) \cdot n < \underline{e \cdot n}$  expected rounds; the algorithm sends  $O(n^2)$  expected messages.*

*$e \cdot n$  expected rounds  
Also completes with leader*

So, this particular expression will lead to this particular theorem that, there is a randomized algorithm that elects a leader in a synchronous ring with a probability 1, in  $1/c$  upon  $c$  times  $n$  that is  $e$  times  $n$ . So, within  $e$  times  $n$  number of expected rounds, the algorithm terminates with a leader and this particular algorithm will send order any square messages, expected messages in each round.

(Refer Slide Time: 35:46)

**Conclusion**

- Randomization has **proved to be a very powerful tool** for designing distributed algorithms.
- It allows us to solve problems in situations **where they cannot be solved by deterministic algorithms.**
- This lecture extends the formal model to include randomization and describes the **randomization algorithm for Leader Election Problem.**
- We have presented two randomized algorithms: (i) **Synchronous One-Shot Algorithm with  $O(n^2)$  messages** and (ii) **Synchronous Iterated Algorithm with  $O(n^2)$  expected messages**

So, the conclusion: randomization has proved to be a very powerful tool for designing the distributed algorithm, and in this part of the lecture we have taken case study of the leader election problem in an anonymous ring, and we have seen that by doing the weakening of the problem definition, that basically leader is to be elected with certain probability, that randomization is able to solve the problem of leader election in anonymous ring. It allows us to solve problem in the situation, where they cannot be solved by the deterministic algorithms.

This lecture extends the formal model to include the randomization, and describes the randomized algorithm for leader election problem. We have presented two randomized algorithm, synchronous one shot algorithm with order  $n$  square messages, and synchronous iterated algorithm with the order  $n$  square expected messages.

Thank you.