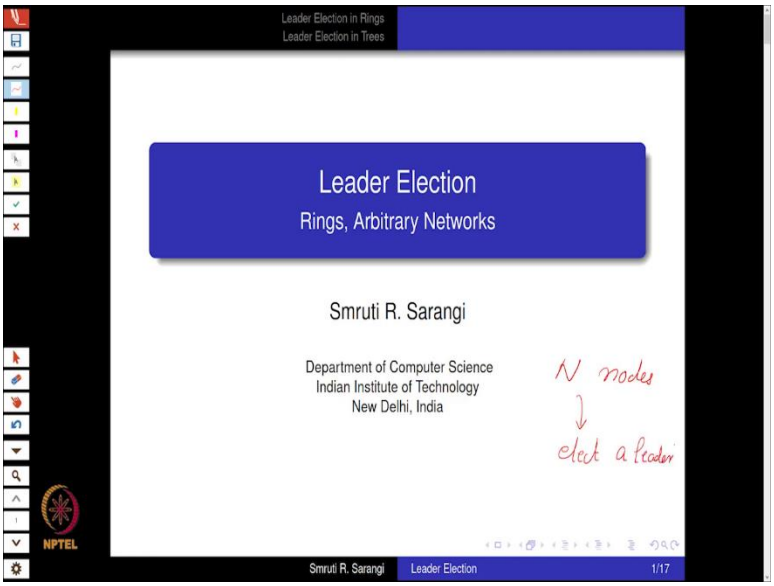
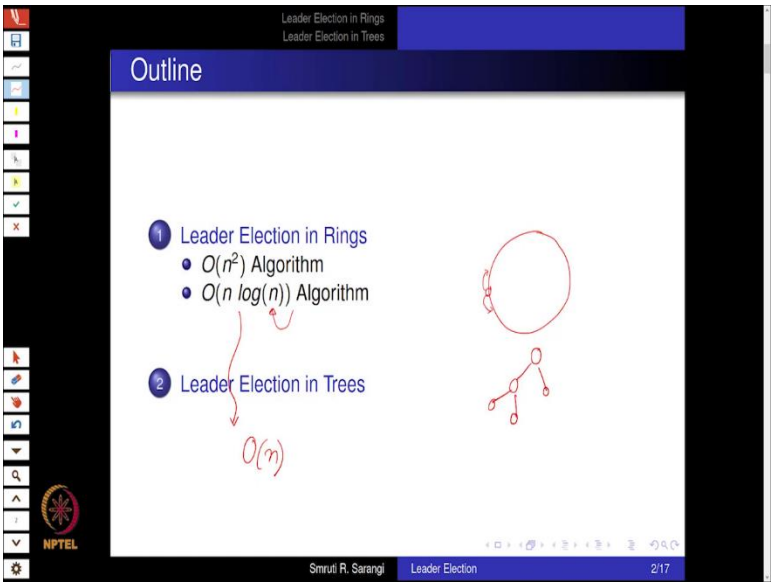
Advanced Distributed Systems
Professor Smruti R. Sarangi
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
Lecture 9
Leader Election

(Refer Slide Time: 00:18)



Welcome to the lecture on leader election. So in this lecture, what we will study is that will be given N nodes, and in a distributed fashion, these nodes need to run an algorithm where they need to elect a leader. So we are not assuming any faulty processes here. So we are assuming that they will elect a leader just by exchanging messages.

(Refer Slide Time: 00:48)

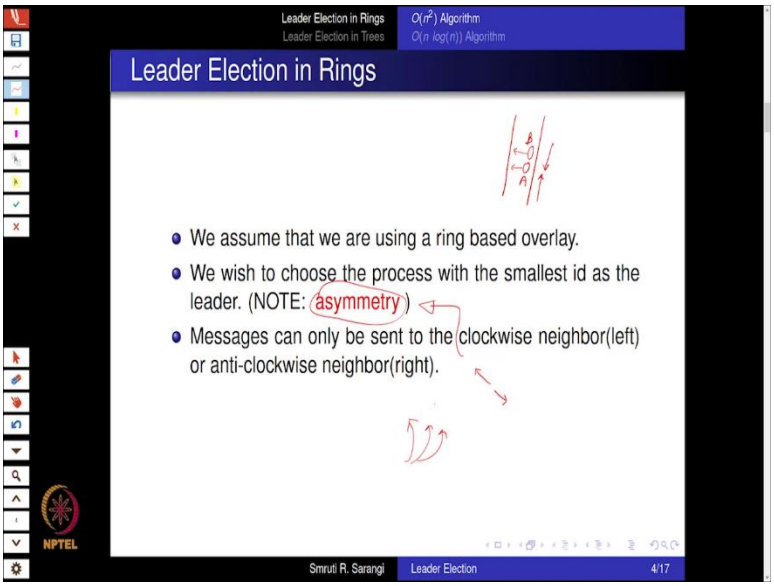


So we will start with a ring overlay, where all the nodes are arranged as a ring. So every node can either send a message to its clockwise successor or its anti-clockwise successor, or send a message to the left or to the right. That is the only messages that it can send. So this network is kind of simple and constrained.

So we will look at an O (n^2) message complexity algorithm, and then refine that to an O(n log(n)) version, and then we will consider another version of the problem where the processes are arranged as a tree. So when they are arranged as a tree, we will find that they have better properties.

So we can reduce this n log n further and make it in the ballpark of O(n). So let us discuss leader election algorithms. So I stand corrected, not the ballpark of O(n), but O(n). So, let us go to the tree part and we will discuss, but first we solve the simpler problem with rings.

(Refer Slide Time: 02:00)



So in the ring based overlay, the idea is that we need to choose that process that has the smallest ID as a leader. It has the smallest ID. So of course, there is an asymmetry in a sense, we are saying that look, a node with a large ID or not the smallest ID cannot be the leader, but that is fine. In any distributed algorithm, there is some asymmetry.

If you do not have asymmetry, there is a classic theorem, which says that you actually cannot make progress. So I will explain with a simple example. So let us consider a corridor. So let us consider this person to be Alice, this person to be Bob. So you would have often seen that if, without any understanding beforehand, if let us say, two people are coming face to face in a narrow corridor, Alice is moving this way, Bob is moving this way, then they can collide.

Then what Alice will do is Alice will move to her left, what Bob is going to do is he is going to move to his right, and then they will again collide. They will again be face to face again. Alice will move, but Bob will also move. So they will again be face to face, they will again collide. So this does happen many a time with us, particularly when we come face to face with somebody or in a narrow corridor. So this does happen.

So then typically what happens is one person says that, look, I am stopping you go. But what if the other person also says that? In this case, it is provable that unless we somehow

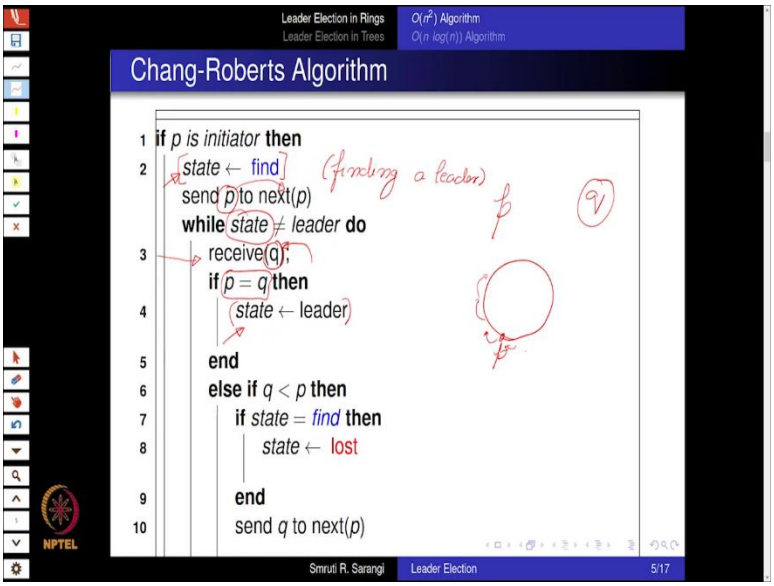
break the symmetry, so unless we say that, between you and me, whoever has more letters in his name, that person gets to move, the other person gets to stand. And we, and if we have the same number of letters in our names, then let us say, let us break the tie by the first letter, otherwise by the second letter and so on.

So in that case, there is a tie breaking. There is an asymmetry we are introducing to essentially solve this kind of a situation. So this, for Hindi speakers, there is a name for this. It is called Pehle Aap. And for non-Hindi speakers what this means, this is like a courtesy, which is given to the other person saying that you may cross me first, but if the other person also returns the courtesy, and he says, no, no, not me, but you cross me, again there will be a face to face situation.

Given the fact that we have that, breaking the symmetry and introducing some level of asymmetry is very important. So given the fact that we are only constrained in terms of the messages we will send either to a clockwise neighbor or to an anti-clockwise neighbor, breaking the asymmetry is even more important because we cannot send messages to others. And clearly, we do not want to break the rules by having a centralized algorithm.

So it should not be the case that we send messages to one node, but again, which node? So that node has to be the leader, but no leader has been elected. So we still want to have a distributed algorithm, and kind of, let us say, compute the minimum ID. And then we all agree that that node is our leader.

(Refer Slide Time: 05:33)



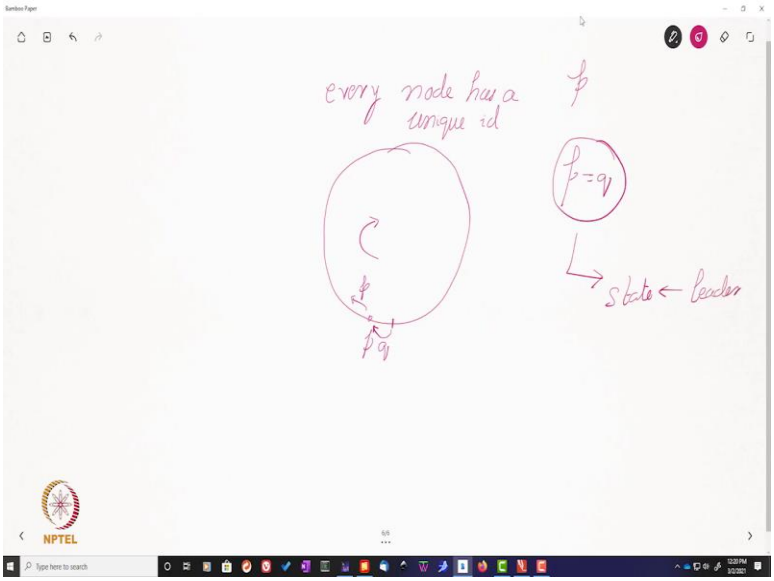
So we will discuss the Chang-Roberts Algorithm. So every node p, so sort of convention we will assume over here is that I am node p, and the other node is q. So always, I am p and the other one, other node is q. So if, let us say, one node, if one node p is initiated, which means that I am just starting, I am just starting the process of leader election, but you need not have one in initiator. You can have many in initiators.

So then it will look at its state variable and set it. So it will set the state variable to find which means that node p is in the process of finding a leader, electing a leader. So what it does is that it sends its own ID to its neighbor. Does not matter clockwise neighbor or anti-clockwise neighbor, does not matter. You choose a direction. It sends its own ID to its neighbor.

As long as the state ≠ leader, which means that the state indicates that the leader has been elected, we continue the algorithm. So what do we do? We come to Line 3. So we receive a message from q. So we have an overlay like this. And, so I am not p, I send my ID to my neighbor, so on and so forth. And finally, from my neighbor, I receive a message. I stand slightly corrected. I do not receive a message from q, but what I actually get is, I receive a message whose content is q.

So I receive a message from my neighbor from the other side where the content of the message is q. But if that is the same as the ID that I had sent, then I declare the state to be leader, and I say that the operation has finished. So what I am basically saying is that, let me maybe draw this, or let us go to another one of our bigger diagrams.

(Refer Slide Time: 07:49)



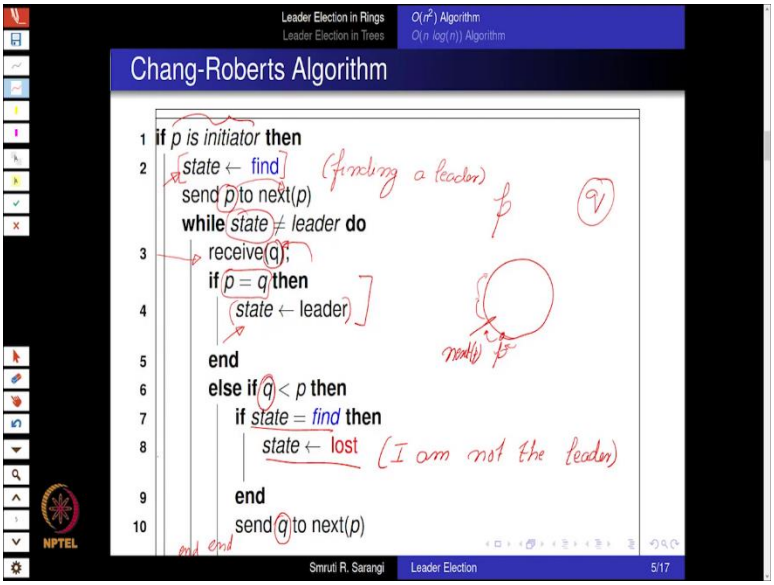
So, what I am seeing over here is, I am node p. I send my ID to my neighbor. It does something, we do not know what it does. Whenever I get a (message), so I can also get messages from my neighbor because my neighbor can also send me messages. So let us say, my neighbor sends me message with content q. My neighbor is not q, the content is q. Even though broadly speaking, we do assume that p and q in the general case are different, but let us say I get a message from my neighbor whose content is q.

If $p \neq q$, so we are of course, making the assumption here. One assumption that is being made, that every node has a unique ID. Without this, our system is not possible. So you can assume it is IP address, which is for example, unique, or there is some other way of providing a unique ID. So we are not getting into that.

But once we have said that every node has a unique ID, and let us say this direction is clockwise, so from my anti-clockwise neighbor, if I, if I get my own ID back, then I say

that, look, the algorithm is terminated. I set the state equal to leader, which basically means that as far as I am concerned, a leader has been elected, and the leader is me.

(Refer Slide Time: 09:14)

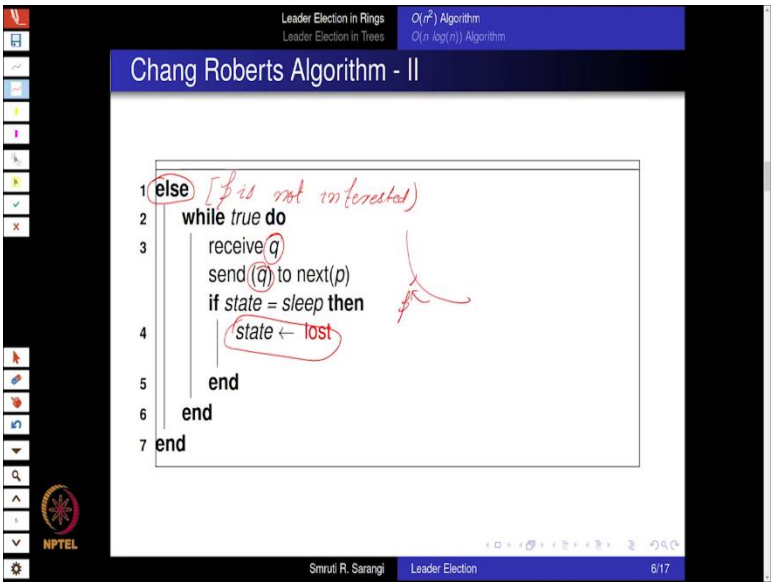


Otherwise, if, let us say, I get q from my neighbor, where q < p, which clearly means that I am not the leader. So in this case, I am the leader. In this case, I am getting a message from my neighbor, with a value which is lower than my ID, and I am p. So if my state is fine, I set my state to lost, which means that I am not the leader.

Fair enough. That is an expression in humility. That is, the reason being that already another node exists whose ID is lower than p. So that node could be the leader or some other node was ID is less than the ID of that other node, that is fine, but the leader is not me. So if this is the case, I just say I have lost, and that is it.

Then what I do is I propagate the value q to next p. In the sense, next p is my neighbor, is my clockwise neighbor. So I just propagate the value q to next p. Fine. So then this else is basically if p = q or if q < p, this else is basically the else for if p is the initiator. So of course, here there should be one end over here, which I am not showing on the slide, and another end for the while loop. So let us assume that these ends exist. So if p is the initiator, then you do it, but else means that p is not the initiator.

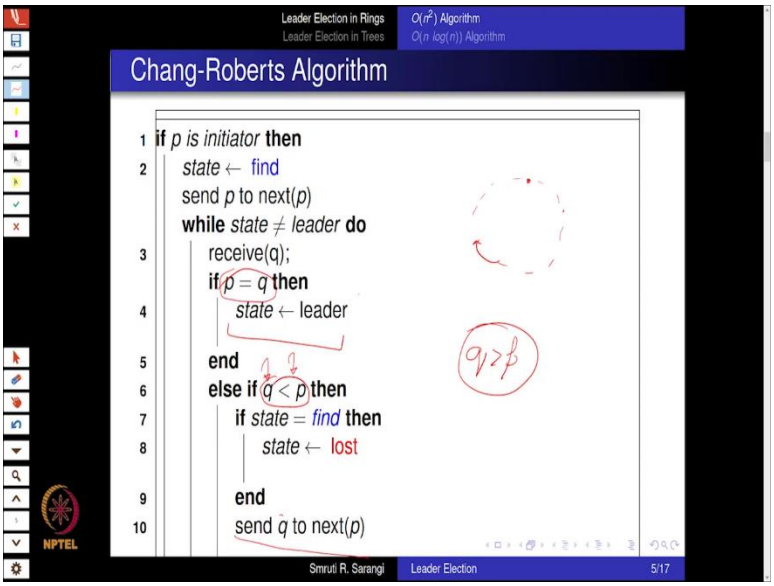
(Refer Slide Time: 10:45)



And so if p is not the initiator, it basically means that p is not interested, p is not interested in becoming the leader. Then of course, there is no problem. So then what it does is that it receives q from its anticlockwise neighbor, and it simply forwards the value of q that it gets, to next p.

And then, if it is sleeping, it just sets the state to lost, seeing that look, I do not care. But anyway, in any case, I am not the leader, and I am also not interested in becoming the leader. So essentially, I have no stakes in the game. So as far as I am concerned, my job is to only forward, and I am doing the job of forwarding.

(Refer Slide Time: 11:37)



So let us now come back and summarize. I will get rid of the ink on the slide. So the idea basically is p is the initiator, it basically means that I am interested in becoming a leader. Then of course, if I am interested, then what I do is, I propagate my own ID down the overlay.

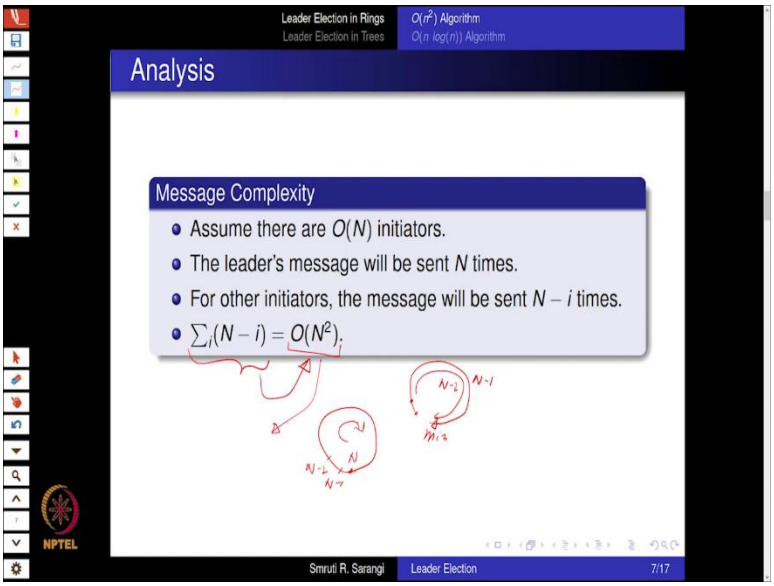
And let us say my own ID comes back to me, which means that nobody absorbs it. This means that every single node, this condition should have held true, which means that my ID is actually the minimum. That is the reason my ID came back to me, and I became the leader. Otherwise, at any point, if my ID was not the minimum, it would have been absorbed in the message, it would have been dropped.

In the meanwhile, what I do is, if I receive a message from my neighbor, and if, let us say, that message has a lower ID than mine, then clearly, I am not the leader. So I set my state to lost, and I just propagate the message down the ring. And of course, this condition is not handled, the reason is that I do not do anything.

If, let us say, I get a message with an ID which is greater than mine, so clearly, the originator of that message is not the leader. So I do not do anything. I just drop the message and I do not propagate it.

So ultimately, if my message ends up being propagated down the ring via all the nodes, and it comes back to me, which means this condition holds, then this means that I am the leader, because nobody absorbed the message. Given the fact that nobody absorbed it, it basically means that my ID must be the minimum. And of course, if I am not interested, I just propagate the message down the ring.

(Refer Slide Time: 13:20)



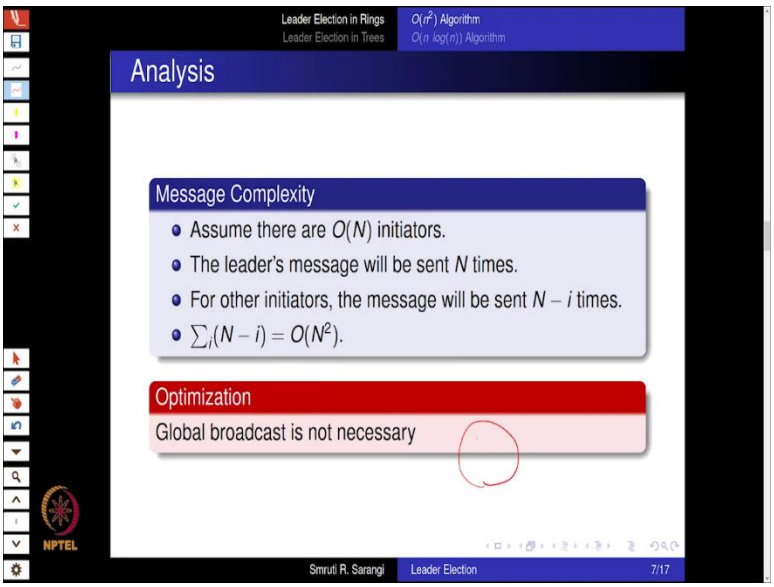
So what is the message complexity? Well, worst case you can have order N initiators. The leader's message is going to be sent 10 times throughout the ring. For other initiators, the message will be sent a maximum of N - i times. So what is i? So let us say I am the second, I am not the minimum, but I am the second minimum, but I am just beside the minimum node.

So then, if let us say, this is the minimum, then I am over here, then my message is going to propagate all the way till this point. And so it will propagate Σ (N − i) times. And let us say after the second minimum, the third minimum is over here, then its message will be propagated N - 2 times.

So if let us say this, this is kind of the worst case. So if I just sum this up, this comes to O(N^2), which is a very standard result in algorithms. And it basically says that look, the worst case arrangement is when they are kind of arranged in ascending order.

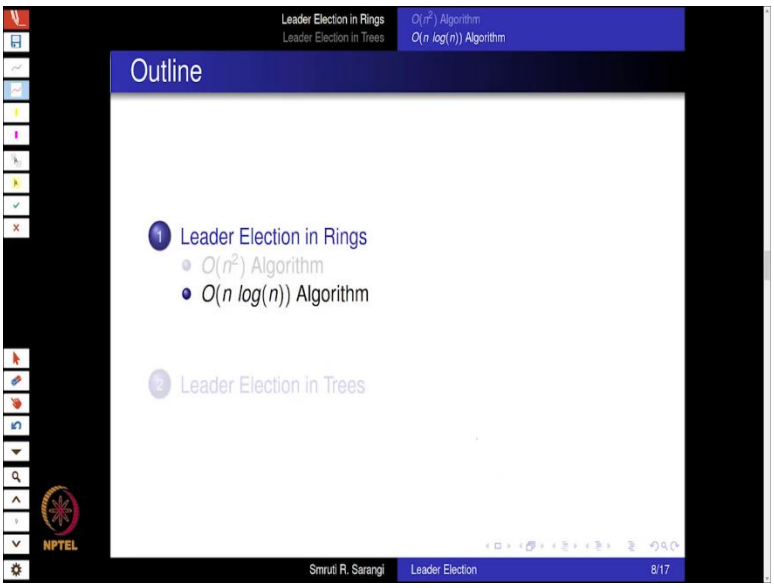
And so then the minimums message, of course, will go through all N hops, but the other ones will go N - 1, N - 2 and so on. And the arrangement of (kit), of course here is that they are arranged ascending orders around the ring. So, given the fact that this is O(N^2) messages, we should make some effort to actually reduce this.

(Refer Slide Time: 14:51)



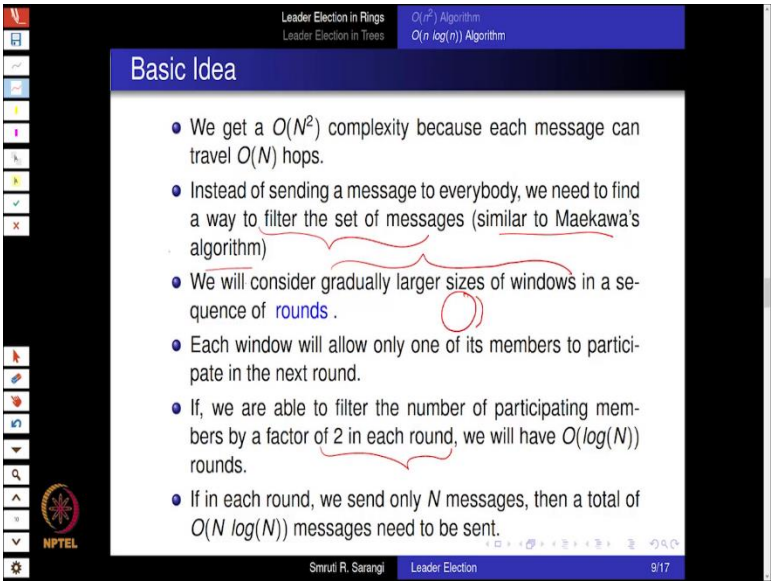
So one optimization is that globally kind of broadcasting to everybody is not necessary. We can do something better. So let us see what we can do.

(Refer Slide Time: 15:02)



So here is an O(N) time, n log n number of messages algorithm.

(Refer Slide Time: 15:10)



So see, the main reason for an O(N^2) complexity is because we want each message to travel as far in the, as far as possible in the ring, which is essentially in the ballpark of O(N). And since we have a sufficient number of nodes that are traveling roughly the entire ring, we have an O(N^2) message complexity.

So instead of sending a message to everybody, can we find a way to filter the set of messages, a similar idea to Maekawa's algorithm in mutual exclusion? So recall that the previous lecture was a mutual exclusion and we had discussed the Maekawa's algorithm. So can we do something similar?

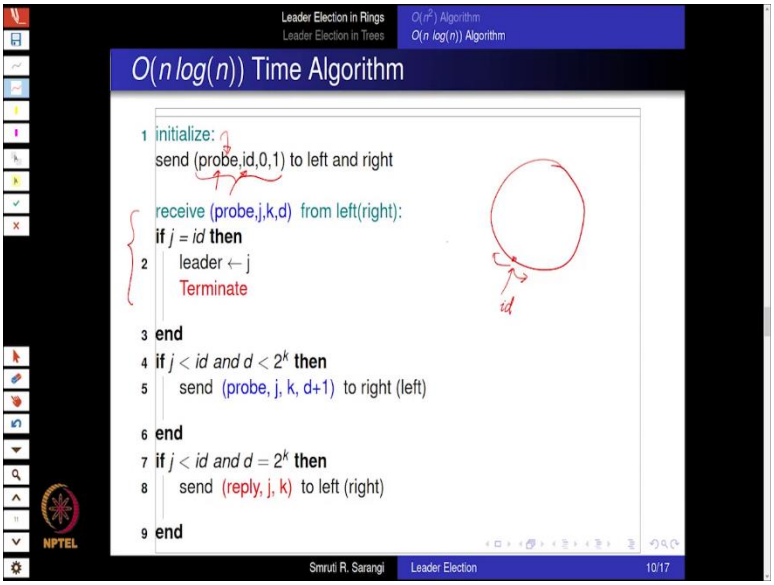
So what we do is that we gradually, kind of, consider larger sizes of windows or gradually consider larger set spaces in a sequence of rounds. So we will define what around is. So what we do is that if we consider a ring, we define a window like this, only one of its members can participate in the next round. And also these, these windows will gradually increase.

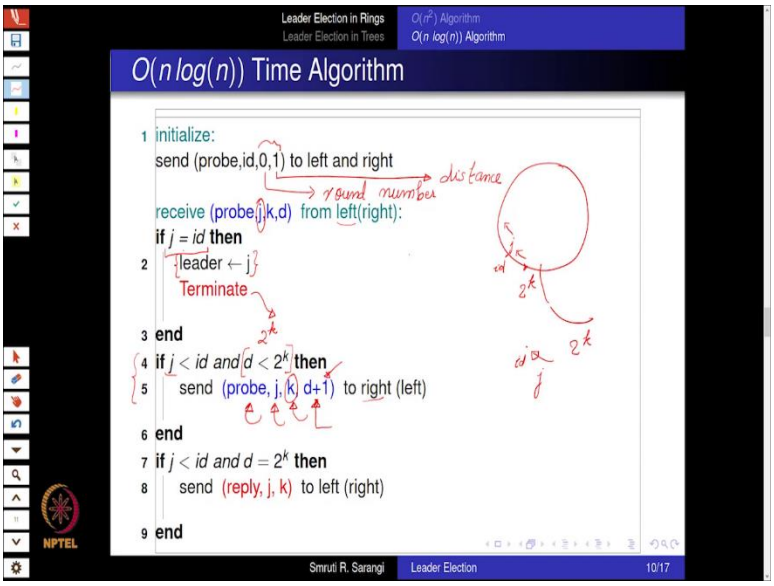
So we will see how, but essentially the idea is that we create these small communities of nodes and every community elects its own leader. And then these leaders participate. And again, they form a community. So recursively, our idea is to somehow reduce the number

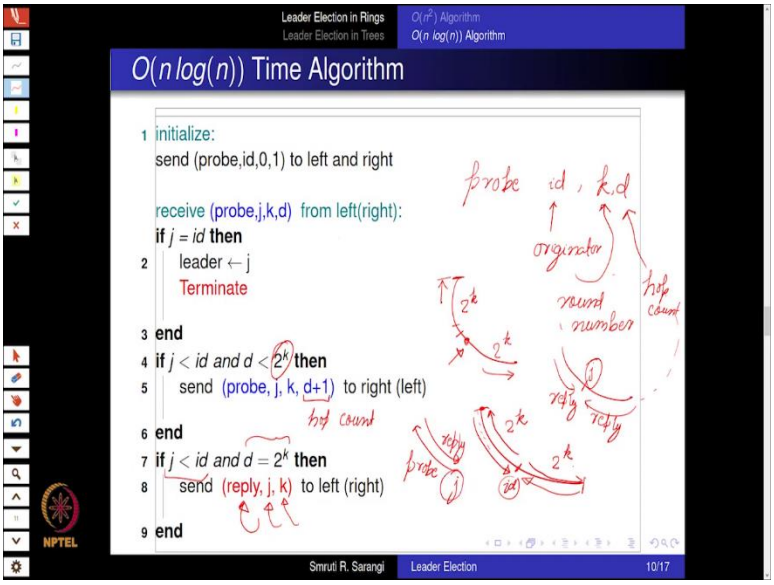
of messages by creating communities, then a leader of a community elect the leader of leaders and so on.

So if you can kind of factor, in every round, if you can reduce the number of nodes that are participating in the algorithm by a factor of 2, that would be really great because in that case, we will have a maximum of log N rounds. So the complexity, even if it is O(log (N)), will be bounded by O(N log(N)), which is what we want. So let us see if we can do that. The idea is you elect a leader, create communities, elect a leader out of community leaders, so on and so forth.

(Refer Slide Time: 17:25)







So let us look at this algorithm. So what we do is that in this case, we send one message to the left and to the right. So the previous algorithm was unidirectional. This is bidirectional. So the message has four fields, probe, id, 0 and 1. So we will see in a second what they are.

So, what they basically mean? so, what they basically mean is probe means it is a probe message, so probe is the message type, id is my ID, and id is the ID of my node. So this is what, and 0 and 1 kind of establish how far the message will go. So let me explain with an example.

So, this will gradually be cleared, but at least out of these four fields at the moment, you just appreciate two fields which are probe and id. Probe is the type of the message, which means I am probing for a leader, and id is my current ID. So I send messages to my left and right. Others also do the same.

So, let us say I am a node and I receiver message from my left or right neighbor. And let us say, the message that I receive is equal to my ID. So akin to the previous algorithm, I declare myself as a leader, leader as j, terminate.

And so, it is the final leader. But what I am saying is that let us say in this process of receiving a message, if I receive a message, which is my ID, so let us say that the message has passed through the rest of the nodes. So I will declare myself the leader and I will terminate, but let us just keep going forward.

Otherwise, if the message that I get is probe, j, k d. If j < id, and furthermore d < 2^k. So we are now in a position to define these two fields. So this field over here is a round number. Fine. And this field over here is basically the distance from the originator.

So if I say, d < 2^k, so if this happens, which means that if j < id, and, so which means I am sending it like this, and let us say that my current ID is id and I get j, and j happens to be lower than my ID, then it means that let us say, if I got the probe from the left, I send it to the right, which means I just propagate in the same direction, or if I got it from the right, I send it to the left.

So what is the key idea? The key idea is, look, I got a message. The message has j as its identifier. And my ID is called id, i and d. If j < id, it means that potentially I am not the leader. The other message's originator is the leader. So what I need to do is, I need to help it propagate further. So I simply propagate the, propagate it in the same direction, in the sense, if it came from my right, let us say, propagate it to my left. And it came from my left, propagate to my right.

So I just make it go in whichever direction it was going, but with a small catch. So it is still a probe message. The identifier of the message is still j. This still remains as k. So k is

basically the round number, which is the same, we do not implement the round yet, but we say that its distance from the original node has increased by 1, so this becomes d + 1.

So what we do is we make this distance check. So this distance, so, k basically gives you an idea of the size of the window. So what we are saying is that if, let us say, this is the original point, then we will have 2^k points on this side and 2^k points on that side, roughly, and furthermore, the maximum distance that you are allowed to propagate in either distance, in either direction is 2^k.

The moment d becomes less than equal to, then we bounce back. We do not propagate it further. We just bounce back. And every hop that we propagate, we increment this distance and make it d + 1. So it is important that you understand what exactly is happening, so I will write this one again.

There are four parts to the message. Probe is the type of the message, id is the originator, then of course we have two more, k and d. So, k is the round number. So what I do is I consider arcs. So if, let us say, I consider an arc of nodes, so then, and let us say the originator is over here, so I pretty much consider 2^k nodes clockwise, and 2^k nodes counterclockwise.

So for me, this is the size of the window that I am considering. And as I move my message, my hop count increases. So the last field over here is the hop count. And what I see is if I receive a message, I propagate it in the same direction. If it is coming, if it is going counterclockwise, I send it counterclockwise. If it is going clockwise, I send it clockwise until I reach the end of this window.

So I send, so let us say drawing it once again, if id is over here and the ends of the window are over here, I send one message all the way up till here, one message all the way up till here until I reach the end. And how do I know I reach the end? Well, every hop that I propagate, I increment the hop count until the hop count reaches the maximum, which is this.

Now, if let us say reaches the hop count in the sense I reach the end of the window, and at the end of the window, same j < id holds, and d = 2^k. So what did I say? We consider 2^k

nodes on this side and 2^k nodes on the other side. So let us say at the end, at the end of the window, I still find j < id in a sense, in this case, id of course is a local property for that node.

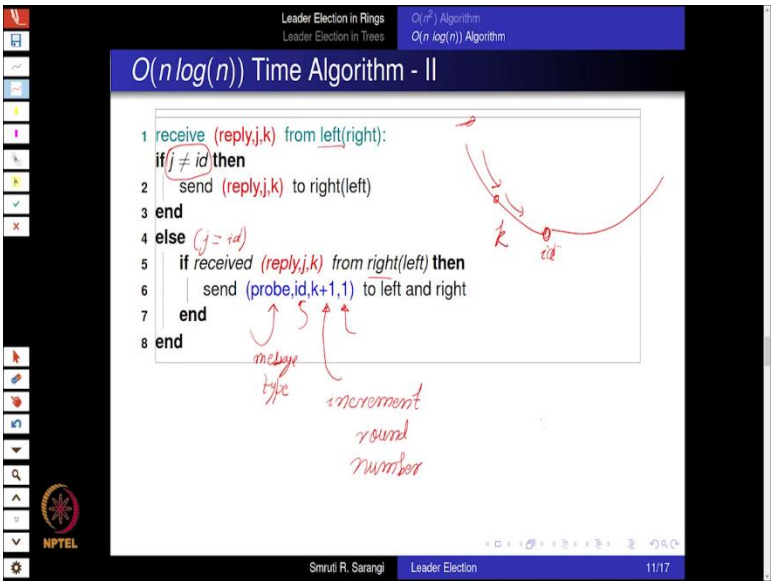
But as far as we are concerned, the originator, at least from this path has the minimum. We send a reply back. So the message came from my left, I send a reply back to my left and I say the reply path is a reverse path. So I just send a reply down the reverse path. So the field is, again, reply. You are sending a reply to the originator, which is j in this case. And k, of course, is a round number.

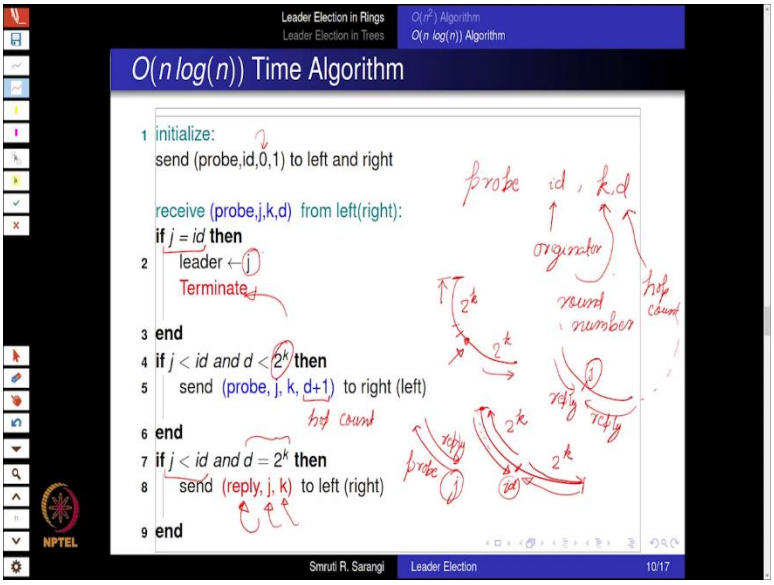
So, you send a reply back. And similarly, the other side also sends reply back. Of course, this was the minimum on these sides. So, one thing that is clear is that in this arc, if it gets a reply from both sides, so, it clearly means that if let us say the ID of this node is j, j is the minimum in this window. We do not know about the rest of the world, which is like this, but that does not matter. At least in this window, j is the minimum. That is what we know.

So, gradually what we will do is we will increase the size of the window, which is something we have promised. But till this point, things should be clear that we just define the small arcs in the circle, equal sized arcs on both sides, simply send a message. And of course, a node can absorb it if the original ID is more than the ID of the node. But let us say it is less than the ID of the node. So I am just repeating that we again have unique node IDs.

So, if it is less than the ID of the node, the node just propagates it until it reaches the end of the window. From there, the message bounces back. So you send a probe message, and what comes back is a reply. And let us say, if you receive replies from both sides, this means that if the ID is j over here, it is the minimum in this window.

(Refer Slide Time: 27:10)





So, once we receive a reply, then what happens? So let us say I receive reply j, k from the left or from the right. So if j ≠ my ID, then what I do is I just propagate the message. So if this is how the reply was going, this is me, I just send it, until it reaches the ID. And similarly, you will have another reply coming from the other side, if this ID is genuinely the minimum in this window.

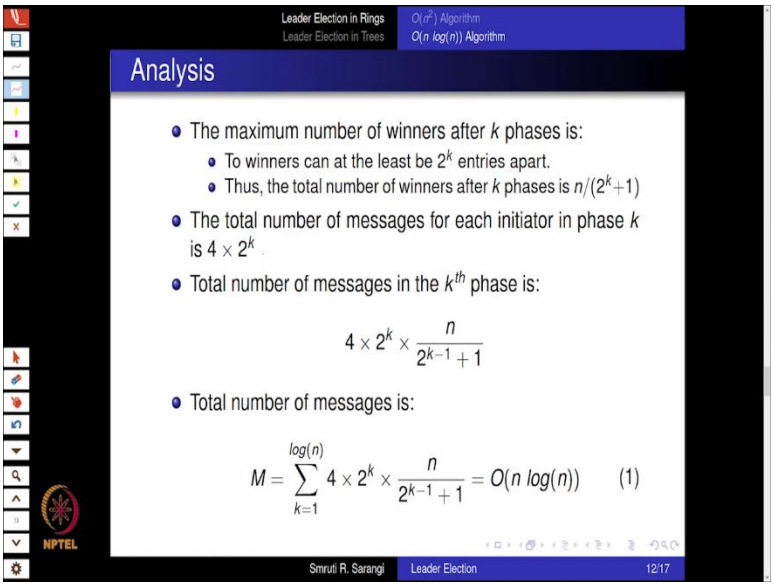
So let us say that it, so this condition does not hold, which means j = id, this condition comes. If you have received a similar reply from the other side, so this was from the left. If you have also received a reply from the right, then you clearly know that this idea is the minimum in this entire window. So now what you do is you increase the window size.

So, for this window, you are able to do it. So now what you do is you increase the window side, size. So again, you send a probe message. So if, let us say, this is the message type, id is again your ID, but here is the fun part. The fun part is you increment the round number, you increment the round number, and again, you reset the value of the count, and then you send a message.

So, if you go back, this is exactly what we were doing. And again, you come here, this is exactly what we were doing. Fine. So, as you can see over here, that we were looking at this, we were looking at the distance and all of that, and similarly, what we now did is that instead of a k size window, we change it to k + 1 size window, which actually doubled it on both sides.

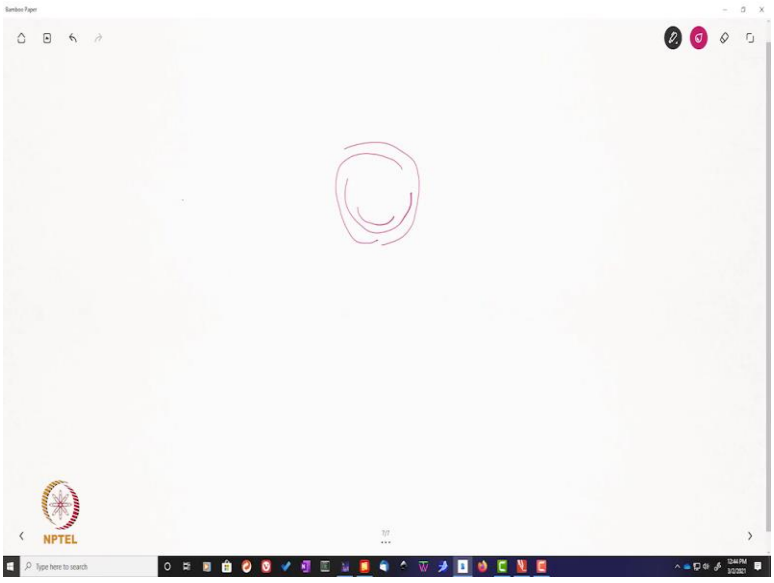
So ultimately, what will happen is our window will encompass the entire circle. And once the window encompasses an entire circle, this condition will hold true. That my own message will come back to me. When that happens, I will know that I am the leader and I will terminate. The termination criteria is similar to the previous Robert-Chang algorithm, which had order N^2 messages.

(Refer Slide Time: 29:44)



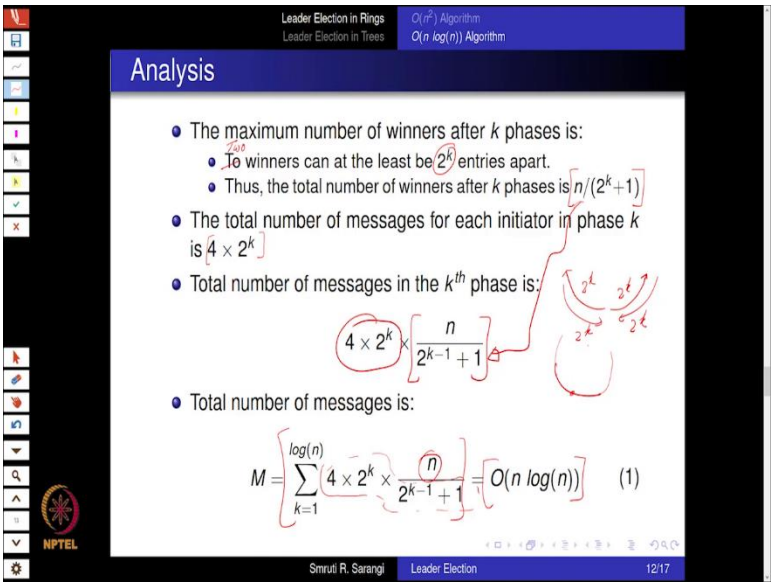
So then let us look at the analysis. Let us compute the message complexity. That is not that easy to do, but we can still do it.

(Refer Slide Time: 29:59)



So again, coming back, what we actually ended up doing is that we just considered larger and larger windows, one small window, then a larger window, then we kind of double this, we start with this, and then we start with this. And finally, the window becomes the entire circle. And when that is the case, the termination criteria uphold.

(Refer Slide Time: 30:17)



So, what we can now see is the maximum number of winners after k phases. two winners can at the least be 2^k entries apart because within the window of 2^k, you cannot have two, two winners. One will absorb the messages of the other. So they have to at least be these many entries apart. Thus, the total number of winners after k phases $n/(\ 2^k + 1)$.

So, if this is a minimum distance by which they are separated, then the total number of winners will be this and this interval + 1. The total number of messages for each initiator in phase k = 4 X 2^k, which is simple. 2^k messages this side, probe messages, 2^k messages on the other side. And similarly, we will again have replies, 2^k this side, 2^k that is Side.

So, the total number of messages in a kth phase, is this much times the number of winners in a k - 1th phase, which is this much, which straightforward comes from here. So of course, it is the winners of the kth phase. But in this case, we will have to consider the winners of the k - 1th phase, because they are the ones who are going to participate in the kth phase, and round is the same thing.

So, the total number of messages in this case, if I were to M= $\sum_{k=1}^{\log(n)}$. So k, the maximum, it can go to log n, because after that $2^{\log(n)}$ will become equal to n. So one of the messages will come back to its place of origin. So this is the maximum value of k.

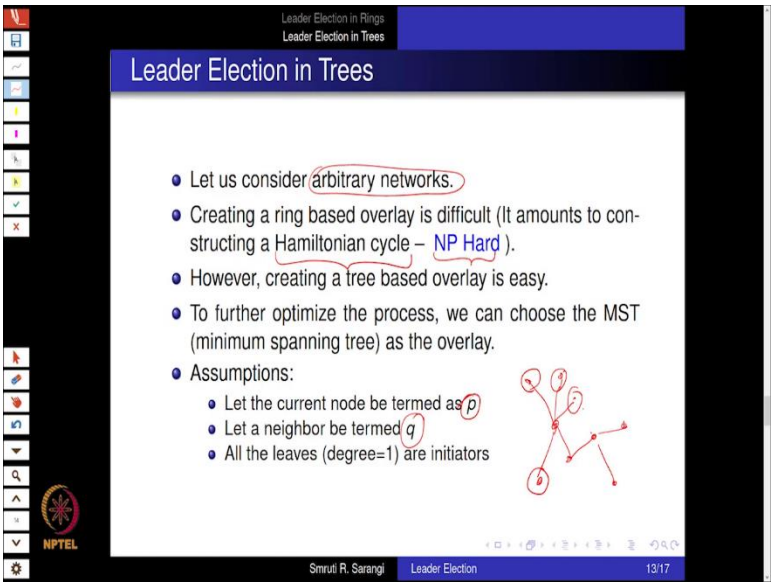
So, if I sum this up, this is not hard at all. And so pretty much what this will work out to be is that this part will pretty much work out to be a constant. And what we will be left with is this n, and this is easily shown to be O (n log(n)). So, hence we have O (n log(n)) messages, that is the message complexity of this algorithm, where the difference between order $n^2$and O (n log(n)) for a large number of nodes is pretty substantial.

And this is clearly one of the ways in which we can elect a leader, by gradually increasing the size of the window by a factor of 2. And other thing we did is we actively suppressed messages in the sense that, let us say, in every window, there is only one winner. And that winner is the one who is going to participate in the larger window. When we double the size of the window, this winner is going participate.

The rest of the nodes in the erstwhile window will not be initiators. So given the fact that we are suppressing messages, so this is same as saying that unless you win the quarterfinal, you cannot participate in the semifinal, unless you do not win the semifinal, you cannot participate in the final. So, we are suppressing messages. We are telling a lot of nodes not to send just because they lost a round.

So, after suppressing messages, what will happen is that ultimately, very few winners will be left. And finally, only one winner will be left.

(Refer Slide Time: 34:00)

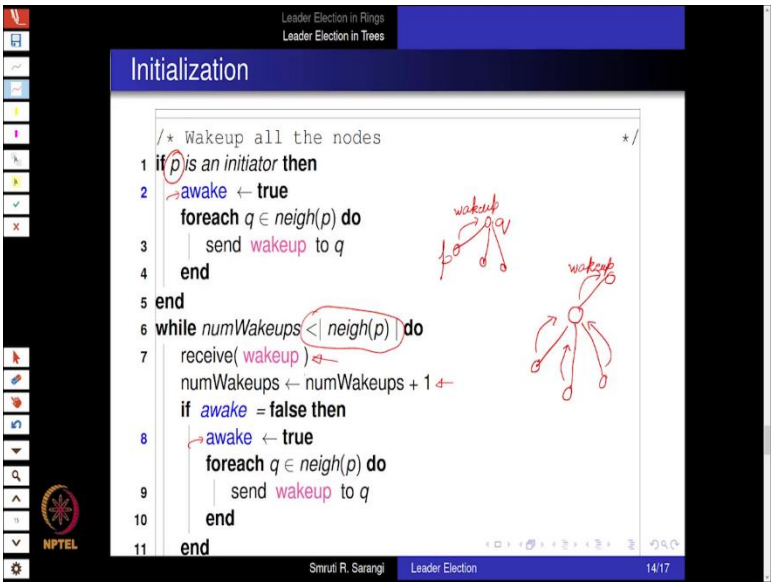


So, now let us consider arbitrary networks. So creating a ring based overlay is, in general, difficult. The reason is that if there is a large network with a large topology, how do we lay a ring through them? So this actually amounts to constructing what is called an Hamiltonian cycle, which is NP hard. So this is quite difficult to create.

Instead of that, if, let us say, we have an arbitrary number of nodes placed everywhere, instead of passing a ring through them, having a spanning tree, or maybe a minimum spanning tree is actually the best idea. We still do not know how to create a minimum spanning tree, but this is what we will study in the next lecture. But let us say if we could create a minimum spanning tree, which we all know it is easy to do, that would be really helpful.

So here again, we will say the current node is p and a neighbor is termed q. And let us say within the spanning tree that we somehow create, how we will create, we will take a look at it in the next lecture, all the leaves, let us say degree = 1, these are the leaves, let these be the initiators. And clearly once the spanning tree has been created, every node knows whether it is a leaf or not. So it also knows whether it is an initiator or not.

(Refer Slide Time: 35:28)



So, given the fact that every leaf is an initiator, this is what we do. So, if node p is an initiator, we set its internal awake variable to true, which means that the node p is not awake. So, what it does is, let us say, if this is a leaf and the leaf is an initiator, for each of its neighbors, so, it will have a single neighbor though, if it is a leaf, it will send a wakeup message to q.

So, as I said, I am always p and other node is always q. So I am p over here, the other node is q. So I will send a wakeup message to q. So what does this node do, which is a parent? So, as long as while the number of wakeup is less than its number of neighbors, what does it do? It receives a wakeup, and it increments non-wakeups.

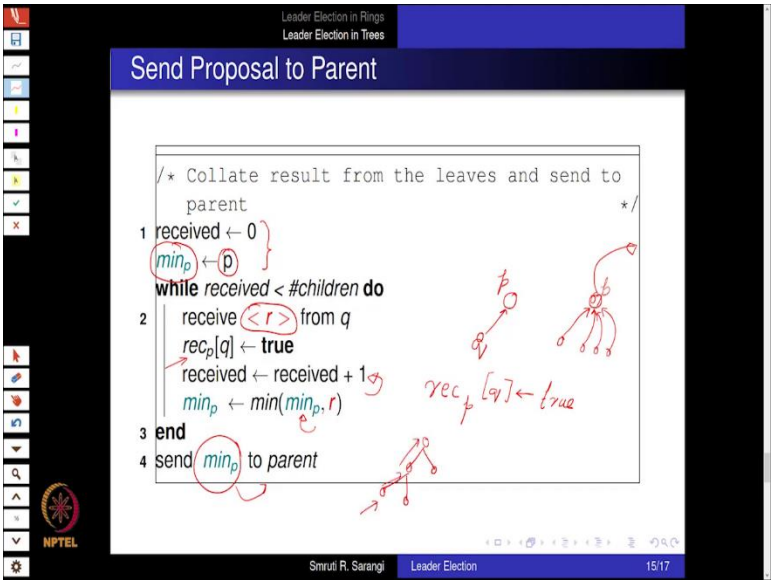
So, which means that if I have a node over here, which is getting wakeup messages from the initiations, as long as it has not gotten a wakeup message from one of its children, it will continue to receive, it will continue to increment the non-wakeups variable. Furthermore, if awake is equal to false, so then it will set awake equal to true to wakeup.

And then what this node will do is that for each q element of neighbor p, for each of its neighbors, it will again send a wakeup message. So it will again tell its neighbors that, look, I am waking you up. So all of its neighbors, of course, the rest are initiators, but it could

have another neighbor here, let us say its parent, it will send a wakeup message there also, saying that, look now I am waking you up.

So what do we do? Well, if you are an initiator, you wake yourself up, wake up your neighbors, if you are an internal node, and so then as long as a number of wakeup you are less than, you have received less than the number of your neighbors, you keep receiving them and you keep on waking up your neighbors. Fine. Fair enough.

(Refer Slide Time: 37:55)



Then what we do, is after the wake-up state, what you do is that, so this is when you actually run the algorithm. So the algorithm is quite simple. So we have two variables over here received and min p. So initially min p is my ID, which is p, and the received is 0. So for each internal node as long as received is less than the number of children.

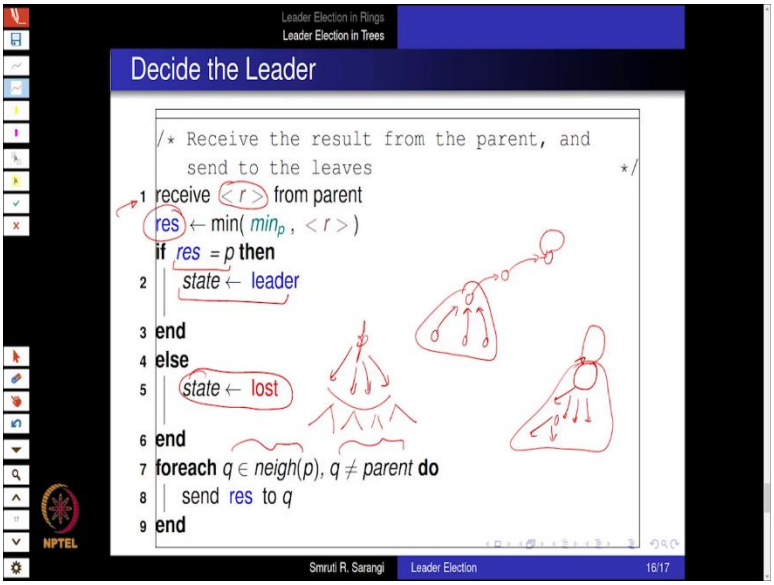
So everybody knows how many children it has, it will continue to re receive a message from q, so where q is the other node, so q is the sender and I am p. So I will continue to receive a message. And I will set this variable of mine, which is the received array, I am always node p, the other node is always node q.

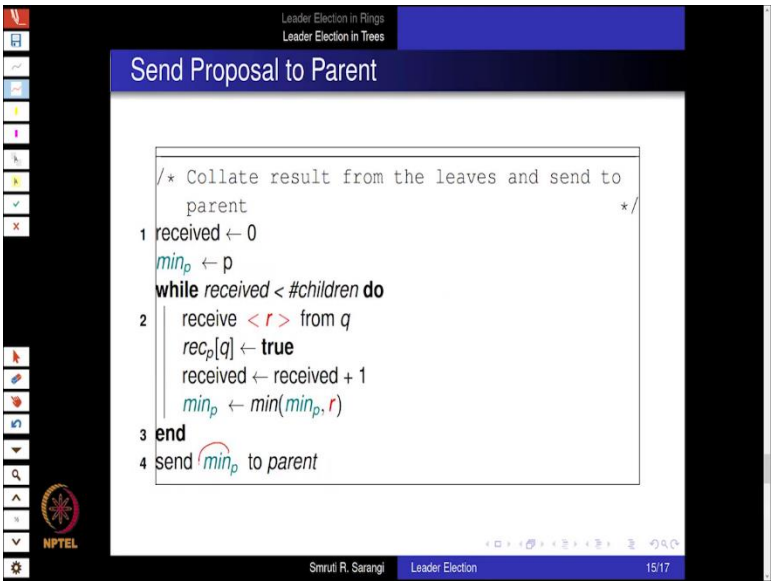
So for each qth with entry, I will set it to true. And I will increment the number of messages I have received. And then furthermore, what I will do is that the minimum that I maintain will be the minimum of the previous minimum and the value that I got from my child.

So, essentially what I do is after I have woken up, every child sends its ID to its parent. The computes the minimum of all that it has gotten from its children and its own ID, minimum of the entire set. And once it computes the minimum of its entire set, so these are all the children, and this is the parent over here. Once it does that, it sends the overall minimum, which means the minimum of all of its children and itself, it sends the overall minimum to its parent.

So in this way, we kind of aggregate, we kind of walk up the tree that all the children send their values to the parent, the parent computes the minimum, the parent sends the computed minimum to its parent, so on and so forth, it keeps on propagating towards the route, so which is quite simple. This is what you would anyway expect, would be happening in a tree, a rooted tree, particularly, where all the traffic would ultimately flow towards the root.

(Refer Slide Time: 40:07)





So now what happens is each child sends the value that it has gotten from its children to its parent. The parent computes the minimum for the entire sub-tree, then forwards it to its parent, which again does the same, which forwards it to its parent, which again does the same. Finally, the message goes to the root.

Once the message goes to the root, we assume that the root has a special property, that the root is its own parent, and the root is its own child also. So the root is as such, self-referential, in the sense the root is its own parent and own child. So then, what really happens is, so, so why do we make this assumption?

Well, this will be clear when we look at other algorithms that also work with trees, but let us assume for the time being that this is the way that our tree has been created, that every node points to its parent and the root points to itself. That tells the root that it is the root. So what did we do in the last slide? What we actually did is we sent the computer minimum to the parent.

So what would the root do? So if I would look at the root, the root would essentially compute the minimum of the entire tree, which is the idea of the leader, wherever it is, and it will forward that to itself. So as far as the root is concerned, the root will begin execution at Line 1, where it will receive r from the parent, so it will receive r from its parent.

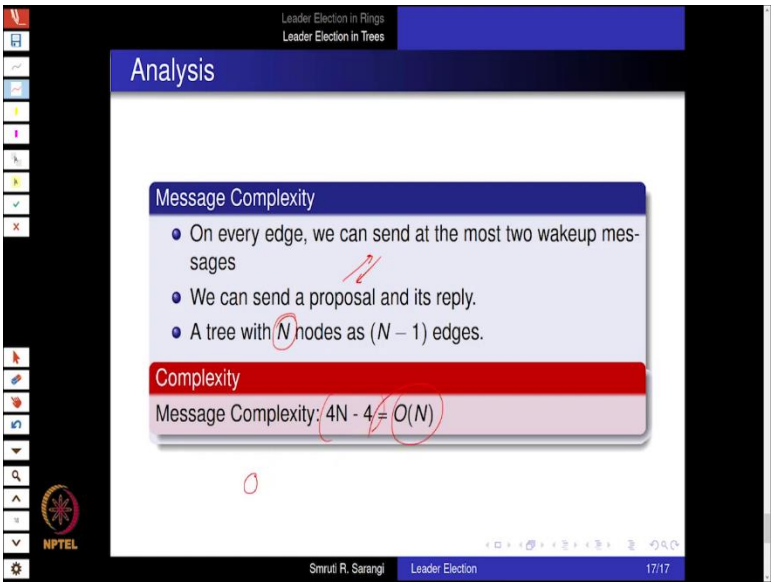
So the moment it is receiving a value from its parent, as far as the root is concerned, the process of leader election is over. And now it is, time has come to broadcast it. So what it will do is it will just compute the minimum, the minimum of its own sub-tree, which in this case is the entire tree and the value it is getting.

But again, this line is redundant, but nonetheless, the important point is that the, that res contains the idea of the system wide minimum. And this will be broadcast down the tree to every single node. So out of this, it is possible that one of these nodes will actually be the leader, it will have the least ID. So every node will compare res with p, and if res matches p, then it means that that node is indeed the leader. So it will set its state equal to leader. Otherwise, it will set its state to lost. Fair enough.

And then what every node will do is? So, let us say every node p will send the message that it got from its parent regarding the global minimum to each of its children, and it will of course not send it to its parent, q not equal to parent, q is element of neighbor, which makes q a child of p. So it will send it to each of its children and they will again, propagate the message downwards.

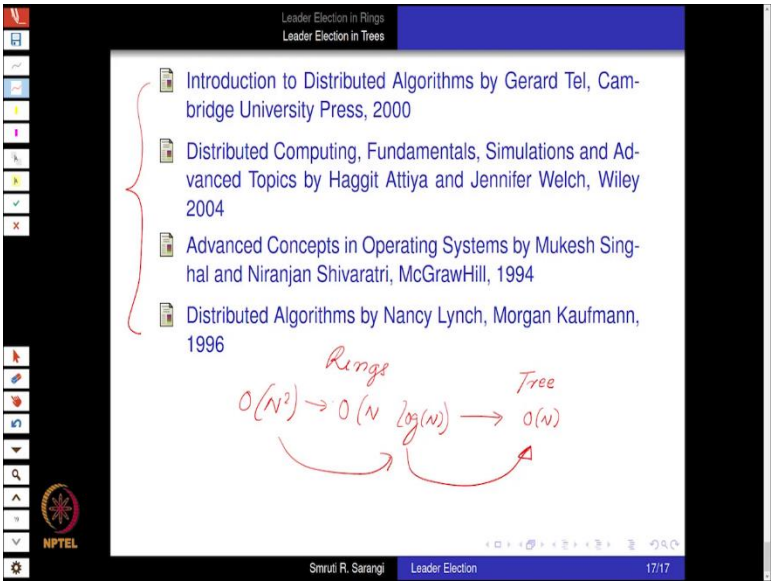
So via this method the leader can elect itself as the leader. And furthermore, the message can be broadcast all over the tree because every parent will forward it to its children, the child will again forward it to its children, so on and so forth.

(Refer Slide Time: 43:38)



So what is the message complexity? Well, on every edge we can send at most two wakeup messages. We can send a proposal going up to the parent and the reply, which basically means with N, in an N node three, we will have N - 1 edges. So the maximum number of messages is 4 times this, because as I said, two wakeup messages in both directions and a leader proposal and a reply. So it is 4N - 4, which is O(N).

(Refer Slide Time: 44:12)



So what we have seen in this lecture, actually, let me go to the other, this is the last slide. So this, of course, are the references. We will come to them in a second. But what we have discussed is that from O(N^2), which was a naive algorithm, we went to O(N log(N)). So of course, we did use a little bit of a trick here, this idea of this expanding window, but these two approaches were for rings.

From here, we went to order N, but what we did is that instead of creating a ring, which also we argued is hard to create, we created a tree, and the tree we found is a very good overlay in the sense that we are able to reduce our, it is very easy to send a message, it is very easy to compute the minimum, it is very easy to compute a bunch of statistics.

And in that sense, a tree is a quite superior data structure as far as a distributed system is concerned. Of course, at this moment, we do not know how to take a set of nodes and construct a tree. Specifically, a minimum spanning tree, because that would kind of minimize the total length, total sum of all the edges.

You can think of it as the message sending cost. So if you somehow have a way of computing an MST, then you can very easily solve problems, such as leader election. As you can see, the complexity will go down and down, and these problems can be solved very easily subject to the fact that we know how to create one. So that is the subject of the next lecture.

So these are the few references. So these are some very popular books on distributed algorithms. So they do have a lot of such distributed algorithms. So everything that we are doing at the moment are all asynchronous algorithms. In the sense, there is no time base. When we discuss other kinds of algorithms, you might find synchronous algorithms. Fair, let us say we, the algorithm will proceed in rounds, where it is assumed that, let us say, every node has sent exactly k messages each round.

And then everybody knows when their round begins and when their round ends. So, which means they have some sort of a shared time base. So we will discuss such synchronous algorithms later. At the moment, we want to create an MST, a minimum spanning tree,

given a set of nodes. So that is our next problem, because we have clearly seen how such a tree will be useful.