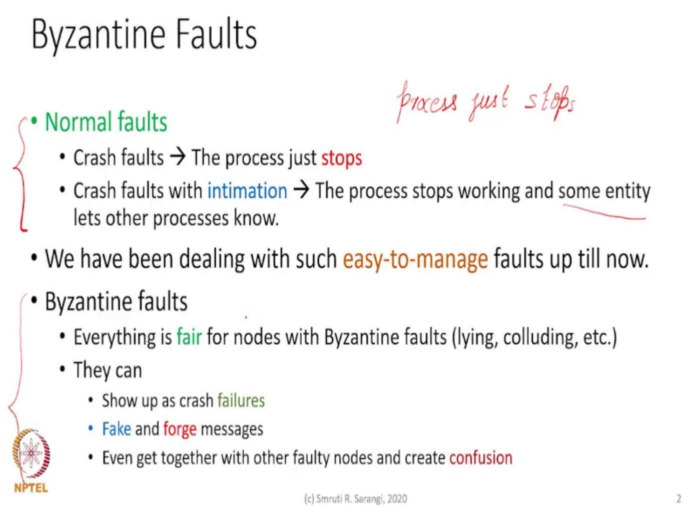
Advanced Distributed Systems
Smruti R. Sarangi
Department of Computer Science & Technology
Indian Institute of Technology, Delhi
Lecture 14
The Byzantine General's Problem

In this lecture, we will discuss Byzantine Fault Tolerance, which is also a method of consensus. It is known as command consensus.

(Refer Slide Time: 00:30)



Byzantine Faults

process just stops

- Normal faults
  - Crash faults → The process just stops
  - Crash faults with intimation → The process stops working and some entity lets other processes know.
- We have been dealing with such easy-to-manage faults up till now.
- Byzantine faults
  - Everything is fair for nodes with Byzantine faults (lying, colluding, etc.)
  - They can
    - Show up as crash failures
    - Fake and forge messages
    - Even get together with other faulty nodes and create confusion

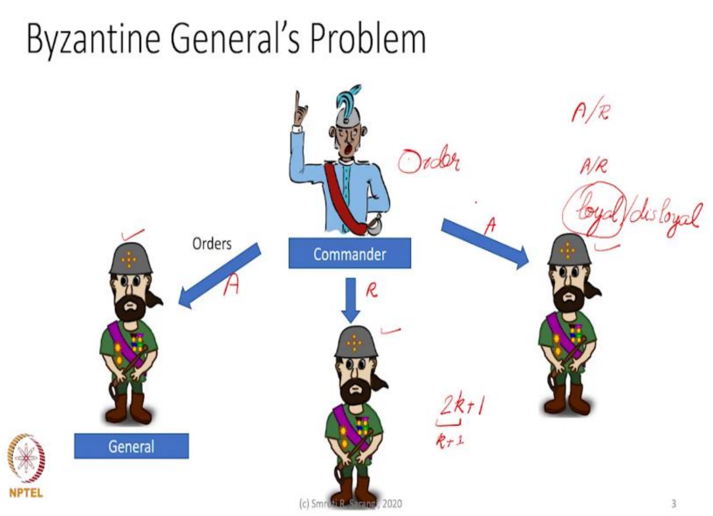
(c) Smruti R. Sarangi, 2020 2

So, we will distinguish between two kinds of faults. So, the first kind of faults are normal faults which we have been seeing up till now. So the normal faults can be crash faults. In this case, the process just stops. Or, it can be a crash fault with some intimation. So it can let other processes know, some entity, typically another process knows, that it has suffered from a fault.

So, we have essentially been dealing with such easy to manage faults where a process just stops. So where essentially a process just stops. That is pretty much all that we have been dealing with now. In some cases, other processes get to know. And in some cases, other processes do not get to know, like in flp result. But that has not significantly changed the way that we design our algorithms. But what we will see now is that we will have a new kind of faults, a new set of faults known as Byzantine faults.

So in, for a Byzantine failed node, everything is fair. So they can lie, they can collude with other failed nodes, and, so they can show up as crash failures, they can fake messages, they can forge messages. So they can pretty much do everything. So everything is fair. So they need not send, they need not participate in an algorithm, they can deny sending messages, they can just stop sending messages, they can lie. So for them, everything is fair.

(Refer Slide Time: 02:18)



Byzantine General's Problem

So the command consensus that we are looking for now, so this is inspired from the classic Byzantine General's problem which was proposed way back in the distributed systems literature. So in this case, we have a commander. So the commander issues an order. So all of them are general. So when we use the term general, they might refer to any of the lieutenant generals. So, we have three lieutenants over here.

So any of these lieutenant generals are also generals, and the commander is also a general. So the command commander essentially issues an order. The order is sent to all of the generals, and each general, furthermore, can either be loyal or disloyal, which basically means that if the general is loyal, the general does not have a Byzantine fault. And if the general is disloyal, then it has a Byzantine fault.

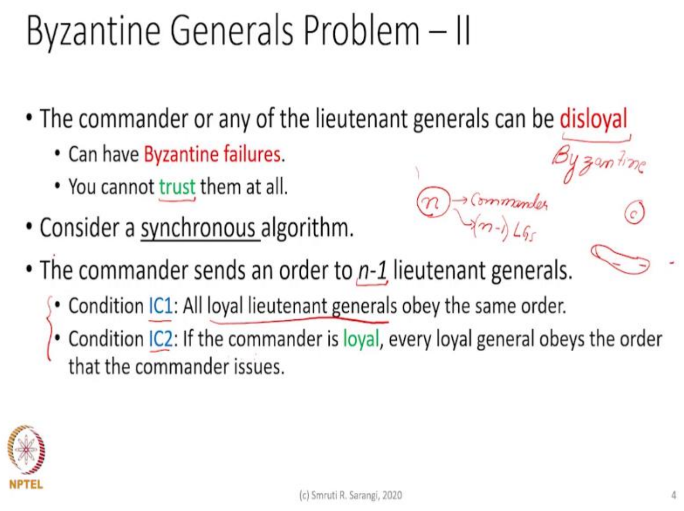
So what does it mean? So let us assume we are trying to do a binary consensus where essentially the generals are in different camps and they send messages to each other. So the message cannot be forged in the middle. So there are only 2 Kinds of messages, attack or retreat. So let us assume that the commander sends the attack messages to all of them.

So what we essentially want is we want all the loyal generals to come to the same decision, whatever it is, attack or retreat. And of course, similar to regular consensus, it cannot be a dummy decision, in the sense that all of them cannot decide retreat or all of them cannot decide attack. We will see why in the next few slides. But essentially, the idea here is that if the commander issues an order, and assuming that the commander is loyal, all the loyal generals have to obey that order.

So here is a fun part. So, anybody who would listen to this would say that, look, this is not hard at all. All that we need to do is we just need to compute a majority. So let us say that there are 2 k + 1 lieutenants. So if k + 1 of them are loyal, then pretty much all of them would have gotten the same orders. Even if there is lie about what they have gotten, at least we have a majority of k + 1. So even though this sounds rather reasonable, however, the main problem is that a commander himself may not be loyal.

So, that is the main problem. So the commander is not loyal. What the commander would actually do is that it would send an attack message to the first lieutenant general, a retreat message to the second and then attack message to the third. So then they would sort of mutually get confused because the commander would have sent different messages to different people. And this would be very, very confusing. So in this scenario, if we still want all the loyal generals to make the same decision, it is rather complicated.

(Refer Slide Time: 05:40)



Byzantine Generals Problem – II

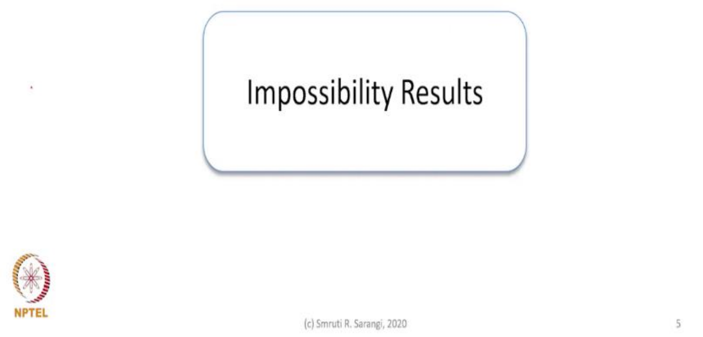
- The commander or any of the lieutenant generals can be disloyal
  - Can have Byzantine failures.
  - You cannot trust them at all.
- Consider a synchronous algorithm.
- The commander sends an order to *n-1* lieutenant generals.
  - Condition IC1: All loyal lieutenant generals obey the same order.
  - Condition IC2: If the commander is loyal, every loyal general obeys the order that the commander issues.

So let us now formalize these conditions. So the commander or any of the lieutenant generals can be disloyal. And here disloyal basically means the Byzantine failure, which means that they can behave in an extremely unpredictable fashion. So you cannot, you cannot, we cannot, nobody can trust them at all. So let us consider a synchronous algorithm. So note that this is not a synchronous because in a synchronous algorithm cannot even tolerate a single faulty process. And in this case, we want to tolerate a lot of faulty processes.

So let us assume that there are total n generals. Out of that, we have one commander, and n minus 1 lieutenant generals, let us call them LGs. So the commander sends an order to n minus 1 lieutenant generals. So the commander himself, maybe disloyal. So in this case, we want to satisfy two conditions, IC1 and IC2. So IC1 is that all the loyal lieutenant generals obey the same orders. So which means that if the commander is over here and all of the lieutenant generals are over here, the subset of them that are loyal, they have to obey the same order.

They come to the same decision. And IC2 says that if the commander himself is loyal, that every loyal general obeys the order, that the commander issues. So note that the important point over here is that nodes do not trust each other, processes do not trust each other. So there is no way for a lieutenant general in this case to actually get convinced that the commander is actually loyal. So we have to ensure IC1 and IC2 implicitly.

(Refer Slide Time: 07:43)



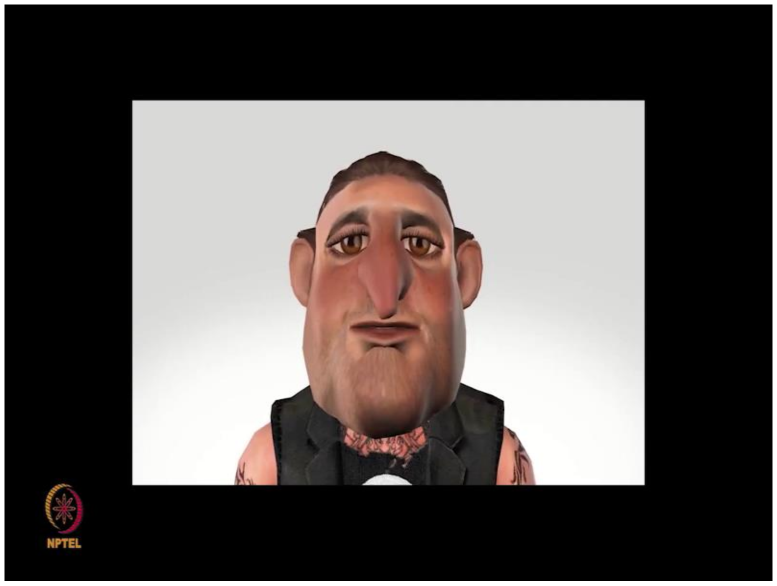
Impossibility Results

(c) Smruti R. Sarangi, 2020

5

So now I will discuss few impossibility results, but before that, let us take a look at two cartoon videos regarding the behavior of general and a disloyal general.

(Refer Slide Time: 08:03)

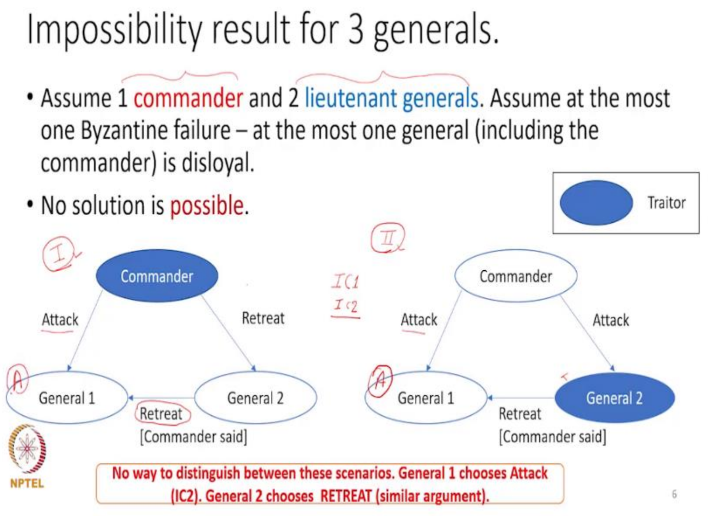


Bad General: I am a bad general. I do not trust anybody. I have a Byzantine fault. I do not even trust myself. All that I do is just lie and lie. I lie to me, to you and to everybody. It is up to you to do something, not me.

(Refer Slide Time: 08:32)

Good General: I am a good general. I do what I am told. I never lie to anybody. However, unfortunately, bad generals give me wrong information. I thus need to verify this with other generals. It is true that I cannot trust anybody, but I never lie.

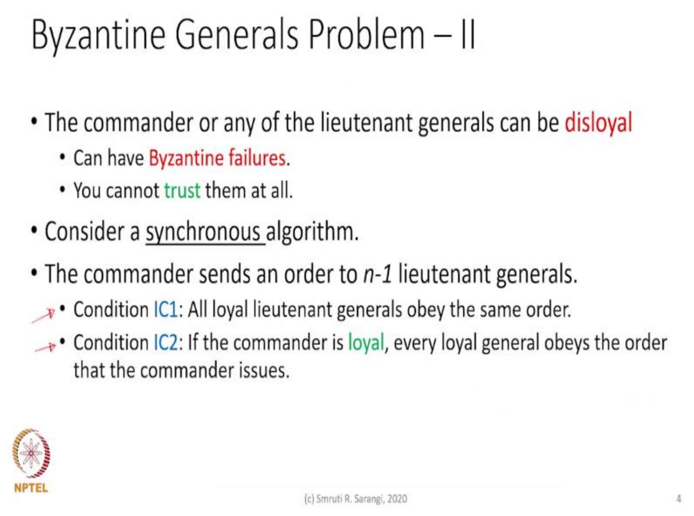
(Refer Slide Time: 09:07)



Impossibility result for 3 generals.

- Assume 1 commander and 2 lieutenant generals. Assume at the most one Byzantine failure – at the most one general (including the commander) is disloyal.
- No solution is possible.

No way to distinguish between these scenarios. General 1 chooses Attack (IC2). General 2 chooses RETREAT (similar argument).



Byzantine Generals Problem – II

- The commander or any of the lieutenant generals can be disloyal
  - Can have Byzantine failures.
  - You cannot trust them at all.
- Consider a synchronous algorithm.
- The commander sends an order to *n-1* lieutenant generals.
  - Condition IC1: All loyal lieutenant generals obey the same order.
  - Condition IC2: If the commander is loyal, every loyal general obeys the order that the commander issues.

NPTEL

(c) Smruti R. Sarangi, 2020 4

So now, let us consider an impossibility result for three generals. So we claim that in a three general system with 1 commander and 2 lieutenant generals, it is not possible for them, for us to solve this problem in the sense that it is not possible for all the processes to come into a Byzantine consensus or a Byzantine agreement. So our claim is that in this case, no solution is possible.

So let us consider two scenarios, scenario I, and scenario II. Let us is also consider the two conditions for correctness, which are IC1 and IC2. So IC1 is all the loyal lieutenant generals obey the same order. And IC2 is that the commander is loyal, then every loyal general obeys, the order that the commander issues. So in this case, the idea is that the commander is a traitor.

So the commander sends two messages. One message is attack. And the other message is retreat to General 1 and General 2, respectively. Now General 1 has no idea the commander is loyal or not. So the only other source of information that it has is actually General 2, which sends it a retreat message, which basically means that the commander had originally sent General 2, a retreat message.

Again, General 1 has no reason for trusting General 2, but at least in this case, since we are seeing the entire scenario from a holistic point of view, we, we are assumed to be an all parts, powerful observer, we can clearly see the General 2 is speaking the truth. So now

General 1 is in a fix. It is getting conflicting messages from the commander and General 2, which means then that one of them is disloyal. It is just that it does not know which one is.

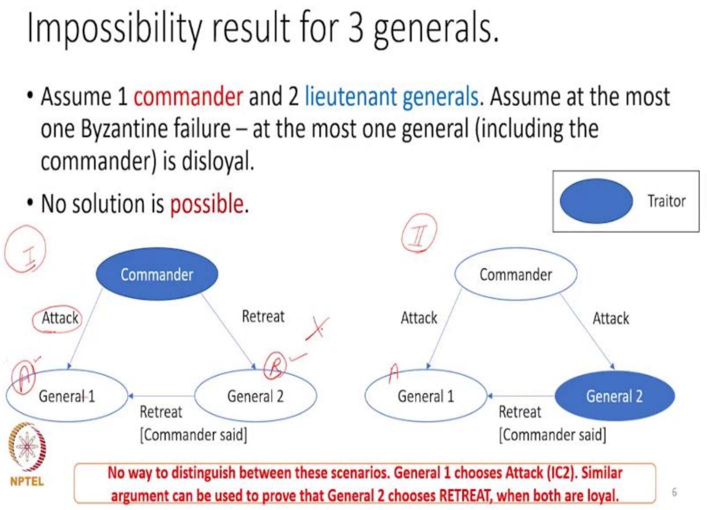
Now, consider Scenario II where General 2 is the traitor. So in this case also, we get an attack message from the commander, and we get a retreat message from General 2. So General 1 has no way of distinguishing between situation 1 and situation 2, because it does not know who is actually the traitor. So now let us apply our known conditions. IC1 and IC2.

So if you would consider the condition that says that if the commander is loyal, all the loyal generals need to obey the same order, so by this condition, General 1 needs to obey attack. And then if you consider the next condition, which says that no, we will, we will not use that condition right now. So, so we will also use this observation, that General 1 has no idea whether it is looking at situation 1 or situation 2. So in both cases, it gets the same set of messages.

And, so since General 1 has to choose attack in situation 2, it has to choose attack in situation 1 as well. So this part is slightly complicated. So the readers, viewers, listeners need to go through this several times. So the important point that is being made here, if we actually take a look at condition IC2, that if a commander is loyal, all loyal generals have to obey the same order.

So in this case, the commander is loyal, she commander sends attack, hence General 1 has to obey attack. But since General 1 has no way of distinguishing between scenarios I and II, even in scenario I, it needs to obey attack.

(Refer Slide Time: 13:16)


Impossibility result for 3 generals.

- Assume 1 commander and 2 lieutenant generals. Assume at the most one Byzantine failure – at the most one general (including the commander) is disloyal.
- No solution is possible.

No way to distinguish between these scenarios. General 1 chooses Attack (IC2). Similar argument can be used to prove that General 2 chooses RETREAT, when both are loyal.

Now we have deleted all the ink from the slide. So let me now quickly summarize what we just derived. So what we derived is that from General 1's point of view, it is getting two messages, a attack message from the commander and a retreat message from General 2. So both the scenarios, scenario I and scenario II, it is not possible for General 1 to differentiate between the scenarios.

So since we want a loyal commander's orders to be obeyed, so it needs to choose attack in both cases. So this is what we were just able to prove. So if we take a look at this case closely, we will see that whenever General 1 gets an order from the commander, it has to obey it. So in this case, it is obeying attack. And the reason is very simple.

The reason is that if it is getting an order from the commander, it has no way of knowing if the commander is loyal or not, just in case it is loyal. Then it needs to obeys order because both of these situations, 1 and 2 look exactly identical to it. Hence, whether General 2 is a traitor or commander is a traitor, it is not possible for General 1 to know. Consequently, it needs to choose the order that the commander gives.

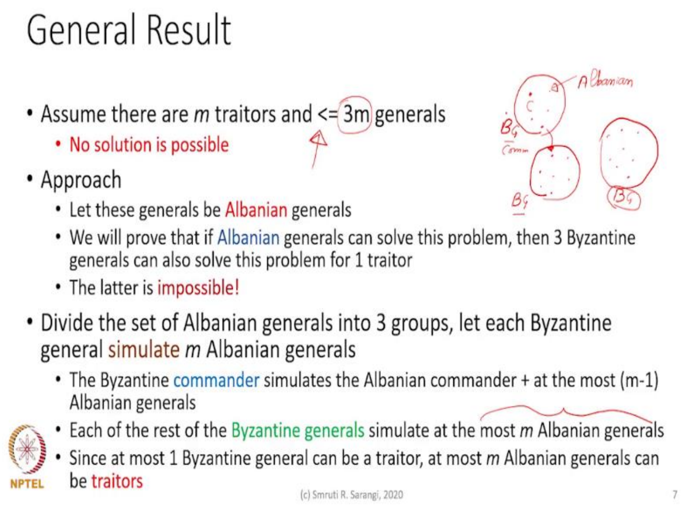
So in this case, which is scenario I, it needs to choose attack. We can use a very similar argument, an extremely, the same argument for General 2. And we can prove with exactly the same logic with the same set of reasoning, same steps, that in this case, General 2 needs to choose retreat because it has no other choice.

It gets retreat from the commander. It has no idea if General 1 is a traitor or the commander is a traitor. And since it wants to fulfill the condition, IC2, which is that if the commander is loyal, then the orders of a loyal commander need to be obeyed. The only choice that we have in this case is that it needs to obey the order that comes from the commander, which is retreat. Same logic.

So if you look at this, which is scenario I, once again, what we find is General 1 obeys attack, General 2 obeys retreat. So there is clearly no consensus. There is no command consensus over here. The loyal generals, if we see here, IC1, all loyal lieutenant generals obey the same order. This is not happening here.

Since this is not happening here, we have derived a contradiction. And the contradiction basically says that in this case, which is with three generals and one traitor, we cannot obtain Byzantine agreement. Now, let us generalize this result to a larger system.

(Refer Slide Time: 16:42)



General Result

- Assume there are *m* traitors and <= 3m generals
  - No solution is possible
- Approach
  - Let these generals be Albanian generals
  - We will prove that if Albanian generals can solve this problem, then 3 Byzantine generals can also solve this problem for 1 traitor
  - The latter is impossible!
- Divide the set of Albanian generals into 3 groups, let each Byzantine general simulate *m* Albanian generals
  - The Byzantine commander simulates the Albanian commander + at the most (m-1) Albanian generals
  - Each of the rest of the Byzantine generals simulate at the most *m* Albanian generals
  - Since at most 1 Byzantine general can be a traitor, at most *m* Albanian generals can be traitors

(c) Smruti R. Sarangi, 2020 7

So the general result is assumed we have less than equal to 3m generals where m > 1, and there are m traitors. So we want to prove that no solution is possible. So way the, the way that we will do it is that we will essentially create two categories of generals. So let us first consider the simple, one of these cases, which where we have 3m generals. So we divide the general into clusters.

So we just create clusters of one-third generals each, and to distinguish them from Byzantine generals, let us call them Albanian generals. So we will see in a second why? So we create three clusters and each cluster is simulated by a Byzantine general. So basically the cluster is simulated a Byzantine general. So we have three Byzantine generals. And we take the 3m generals, divide them into three equal size partitions, and we call the simulated generals, the Albanian generals.

So let us, let me describe the meaning of simulation. So in this case, we are assuming that a protocol exists where with < = 3m generals and with at most m traitors, or let us say with m traitors, if we consider the worst case, we have a method of obtaining a Byzantine agreement. So what is this protocol? The protocol is essentially an algorithm to change the internal state of the Albanian generals, and a message exchange.

So what we can do is each Byzantine general can simulate the finite state machines of all the Albanian generals that are contained within it, within its purview. So we will have three such by Byzantine generals, and so note that these are simulated Albanian generals. So they themselves are not honest or dishonest, so it all depends on the simulating Byzantine general.

So here we make the same assumption as a previous slide, which I can show you right now where we have three generals, one commander, two lieutenants, and one of them is a traitor. So then, all that they do is that they simulate the final state machines and message exchanges of the Albanian generals. And of course, it is possible that a general here might send a message over here. So this would require the involvement of both the simulating Byzantine generals, this one, and this one.

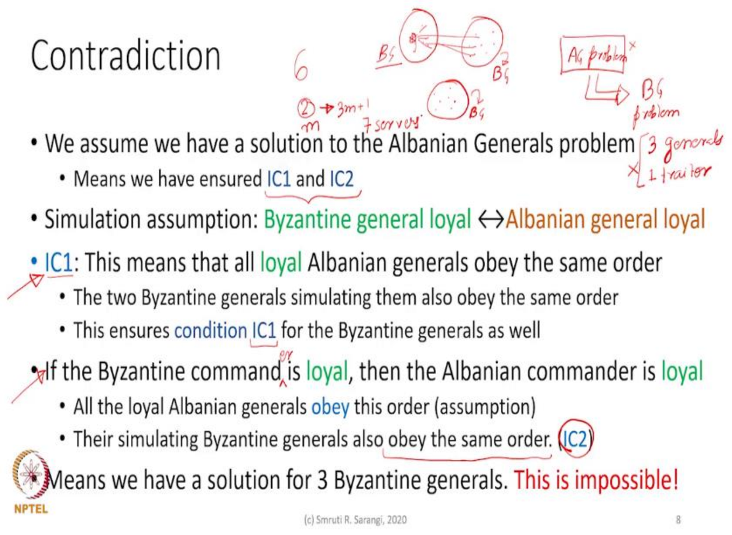
So what we want to prove is that if a protocol exists for Byzantine agreement with Albanian generals, then we can solve the Byzantine general problem with three generals and one traitor. So what we have seen in the previous slide that this is impossible. So we will use this fact to derive a contradiction for this problem. So the notion of simulation should be clear at this point.

So now let us go further. So one of these clusters will have the Albanian commander. So the cluster that Albanian commander, let it also be the Byzantine commander, and the rest of the two clusters will just have Albanian lieutenant generals. So let them also be Byzantine lieutenant generals. So the Byzantine commander will simulate the Albanian commander and at most m minus 1 Albanian generals.

And each of the rest of the Byzantine generals, which means these two clusters, will simulate at most m Albanian generals. Simulate means simulate their final state missions. So since at most one Byzantine general can be a traitor, which means one of these clusters can be a traitor, at most m of these Albanian generals can be a traitor.

That is because if the simulating Byzantine in general is a traitor, then the working of the final state machines of all of these simulator generals is suspected. So if, let us say, the Byzantine general simulating Albanian commander is a traitor, then that makes Albanian commander a traitor as well. So using the simulation logic, it is very easy to derive a contradiction for this problem.

(Refer Slide Time: 22:04)



Contradiction

- We assume we have a solution to the Albanian Generals problem
  - Means we have ensured IC1 and IC2
- Simulation assumption: Byzantine general loyal ↔ Albanian general loyal
- IC1: This means that all loyal Albanian generals obey the same order
  - The two Byzantine generals simulating them also obey the same order
  - This ensures condition IC1 for the Byzantine generals as well
- If the Byzantine commander is loyal, then the Albanian commander is loyal
  - All the loyal Albanian generals obey this order (assumption)
  - Their simulating Byzantine generals also obey the same order. (IC2)
- Means we have a solution for 3 Byzantine generals. This is impossible!

(c) Smruti R. Sarangi, 2020 8

So let us assume that we have a solution to the Albanian General's problem. This means we have essentially insured our two assumptions, IC1 and IC2. So what is the assumption again? Whether the Byzantine general is loyal, then that implies that Albanian general is loyal. And if the Byzantine general is a traitor, it implies that all the Albanian generals that it is simulating, they are also traitors.

So now let us look at IC1. So this means that all the loyal Albanian generals obey the same order. So let us see. So since we are claiming that we have some protocol that ensures this, the two Byzantine generals that are simulating them will also obey the same order. So this will automatically ensure condition IC1, which is this condition for the Byzantine generals as well. So let me draw the three clusters once again, and just mark the Albanian generals that they simulate.

So let us say that two of these simulating clusters are honest. Then by implication, all the Albanian generals that they simulate are also honest. So they will definitely come to an agreement. And by your assumption, by the IC1 assumption, they will agree on the same value, either attack or retreat. So whatever they agree on, can also the agreement values of the simulating by Byzantine generals.

So this ensures that condition IC1 holds for the Byzantine generals as well. Furthermore, if the commander is loyal, then the Albanian commander is also loyal. So what this further means is that all the loyal Albanian generals will obey this order. So that is assumption. So let us assume that the Albanian commander lies over here.

So if the commanding Byzantine general is loyal, then the Albanian commander is also loyal, which means that all the loyal Albanian generals in each of these clusters will obey the same order that is issued by the commander. So then what we can do is that the Byzantine generals that are simulating these clusters can also obey the same order and that will automatically ensure condition IC2, that whatever the commander says, if the commander is loyal, then all the loyal generals also obey.

So what have we just done? What we have done is that out of the solution for, so we have taken the solution for the Albanian generals problem, let, just call it the AG problem, and we have derived a solution for the Byzantine general’s problem for, essentially for three generals and one traitor. So what do we do? So what we know now is that this is not solvable. And given that this is not solvable, the Albanian General's problem is also not solvable.

So this essentially, we have obtained a contradiction over here, that if so, what did we assume? We assume that if there is a solution for the Albanian General's problem, there is a solution for the Byzantine General's problem, for three generals. So since the latter is not true, the former is also not true. Consequently, it is impossible. So how do we make sense of this result?

Well, let us say that we have 6 servers. Out of 6 servers. Let us say 2, have a Byzantine fault. It is clearly not possible for the rest of the 4 correctly executing servers to actually come to an agreement. So what you actually need is that if m = 2, you need 3m + 1, or 7 servers. So if you have 7 servers, then it is possible to arrive at an agreement.

(Refer Slide Time: 26:53)

Byzantine Agreement Algorithm

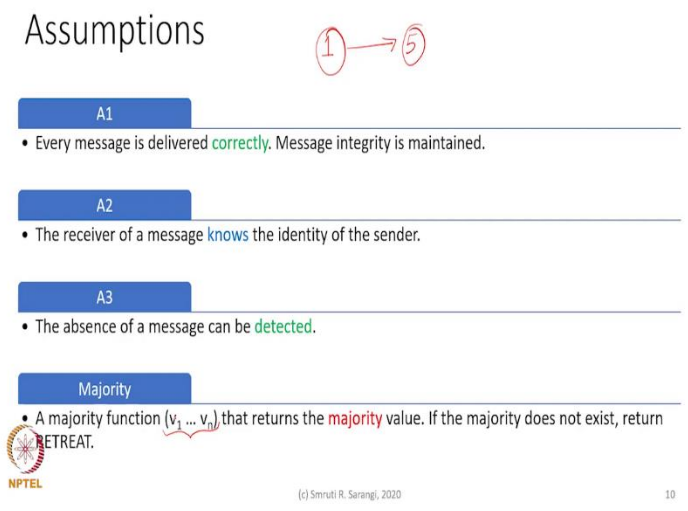
NPTEL

(c) Smruti R. Sarangi, 2020

9

So now let us present the Byzantine agreement algorithm. It is also called a command consensus. That is because there is one commander and the commander wants his command to be agreed, agreed to by all the loyal generals. And of course, if the commander himself is a traitor, then the loyal generals will at least come to an agreement between themselves.

(Refer Slide Time: 27:20)



Assumptions

$1 \rightarrow 5$

A1
- Every message is delivered correctly. Message integrity is maintained.

A2
- The receiver of a message knows the identity of the sender.

A3
- The absence of a message can be detected.

Majority
- A majority function $\underbrace{(v_1 \ldots v_n)}$ that returns the majority value. If the majority does not exist, return RETREAT.

(c) Smruti R. Sarangi, 2020 10

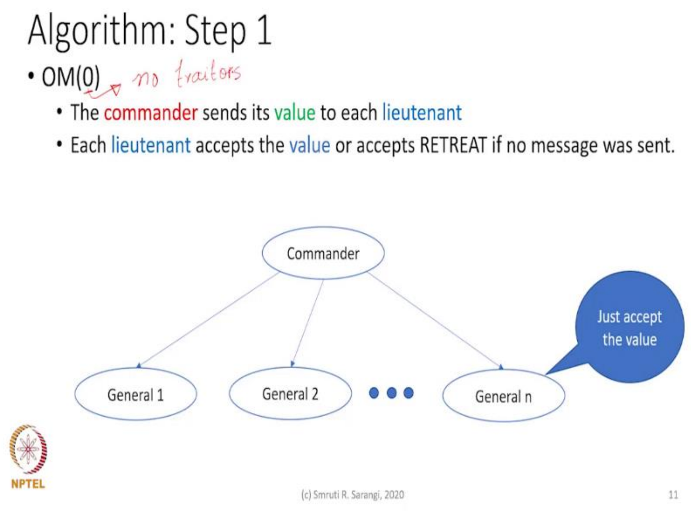
So we have to make several assumptions. So the assumptions that we make are like this, that every message is delivered correctly, which means that it is possible to check for message integrity. So no general in the middle actually spoils or tempers a message. So let us say General 1 is sending a message to General 5, the message is delivered correctly.

Furthermore, the receiver of a message knows the identity of the sender. So it is not possible for a general to actually fake the identity. So this is a key assumption we need to make that, and in practice it is possible to do so using digital signatures, it is not possible to fake an identity.

Now the assumption A3 says that it is possible to detect the absence of a message. So since we have a synchronous algorithm, in a round, it is possible for a failed node not to send a message, but this can be detected. So typically, you, the rest of the nodes can assume a default value. For example, they can assume that retreat has been sent.

The last is we assume a majority function, v 1 to v n, that returns the majority value. So let us say given a set of values, it returns the value which enjoys a majority in the set. And clearly if a majority does not exist, then we return and default value, which can be retreated. So the majority function is important, and we will use it.

(Refer Slide Time: 29:02)



Algorithm: Step 1

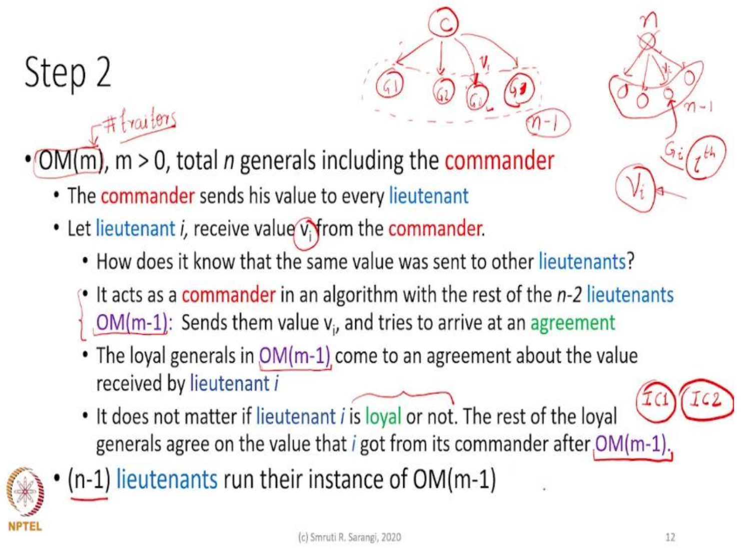
- OM(0) → no traitors
  - The commander sends its value to each lieutenant
  - Each lieutenant accepts the value or accepts RETREAT if no message was sent.

Commander
General 1
General 2
General n
Just accept the value
NPTEL
(c) Smruti R. Sarangi, 2020
11

So let us look at the steps of this algorithm. So this algorithm is actually a recursive algorithm. So we need to first define the base case, which is OM 0. So that is the name of the agreement algorithm. So the commander sends its value to each lieutenant. So that is the first step. So each lieutenant accepts the value or accepts retreat if no message was sent.

So what is 0? 0 is a number of traitors. So if there are no traitors, then of course it is assumed that the commander is loyal. So each lieutenant will simply accept the value that the commander sends. And just in case no message was sent, then you accept retreat. But in any case, as we can see assumptions IC1 in IC2 tend to hold. And that is mainly because there are no traitors.

(Refer Slide Time: 30:13)


Step 2

- OM(m), m > 0, total *n* generals including the commander
  - The commander sends his value to every lieutenant
  - Let lieutenant *i*, receive value v_i from the commander.
    - How does it know that the same value was sent to other lieutenants?
    - It acts as a commander in an algorithm with the rest of the *n-2* lieutenants OM(m-1): Sends them value v_i, and tries to arrive at an agreement
    - The loyal generals in OM(m-1) come to an agreement about the value received by lieutenant *i*
    - It does not matter if lieutenant *i* is loyal or not. The rest of the loyal generals agree on the value that *i* got from its commander after OM(m-1).
- (n-1) lieutenants run their instance of OM(m-1)

(c) Smruti R. Sarangi, 2020 12

Now, that is considered OM m, so where m is the number of, so consider that this is greater than 0. So we have a total of n generals and that includes the commander. So the first step is the same that the commander, so let us say these are, these are all the lieutenants.

So the first step is the same that the commander senses his value to every lieutenant. But of course, the lieutenant can get different values. So let lieutenant i receive value v i from the commander. So that this be, lieutenant, let us have a, well, this, I should have given a different name. So let us say this is lieutenant Gi, and it gets value vi from the commander.

So now from lieutenant i’s from point of view, it has no idea if the commander is loyal or not. So the only thing that it can actually do is it can check with others. So how does it know that the same value was sent to other lieutenants? Well, it does not know immediately, but after some message exchanges, it will get to know. So it starts a recursive algorithm with a smaller argument.

So it starts an algorithm OM(m – 1). So this is very crucial. So what is crucial is that let us say if a message comes to a certain lieutenant, it starts a recursive algorithm with a reduced subset, which is essentially the set of lieutenant other than the commander. So the, what is the aim of starting this?

The aim of starting this is to basically verify and understand what is it that they have gotten, and try to arrive at some kind of an agreement, at least between the loyal lieutenants, to at least see what is it that they have gotten. So this is very similar to an office situation where we have one of those mischievous bosses who tell, who tells different things to different people.

So the only way for the people to actually is to just ask each other, what is it that the boss told them. So this is extremely similar over here, that we have the commander sending messages to different lieutenants. Once the lieutenants get the messages, the only thing that they can do is they can take the commander out the picture and run an algorithm between them to understand what is it that they have gotten? and what is it that they should agree upon?

So these loyal generals in OM (m – 1) need to come to an agreement about the value that has been received by Lieutenant i. So what is the idea here? Well, the idea here is very simple, that let us say if we had a total of n generals at the beginning, we create a smaller subset, which is n - 1. And there what the ith lieutenant tries to do is that it broadcast the value that it got, which is vi from the commander, to the rest of the nodes.

So what is it trying to do over here? Well, it is trying to initiate another Byzantine consensus where it is trying to convince everybody that it has actually gotten vi from the commander. So it is running a Byzantine algorithm precisely, to do that. So after OM (m – 1), what is the final state? Well, the final state is that the rest of the lieutenants in this set come to an agreement about what is it that Lieutenant i,

General i, so general and lieutenant, we will use interchangeable, so what is the value that Lieutenant i has actually cotton from the commander. So this is something that, so this is again a Byzantine problem because it is possible that different nodes in this set might be dishonest. It is possible that Lieutenant i might be dishonest.

So different combinations are possible. So again, it is a smaller instance for the same problem, but after that, at least all the loyal generals in this set will mutually agree on what is it that Lieutenant i actually got. If Lieutenant i is honest, that that value has to be vi. If

Lieutenant i is a traitor, then it has to be some other value, but at least all the loyal generals in this set will agree on that value.

So it does not matter if Lieutenant i either by himself is loyal or not. That is not important. The rest of the loyal generals simply have to agree on the value that i got from its commander after OM (m – 1) finishes. And this is where they have to follow the IC1 and IC2 assumptions.

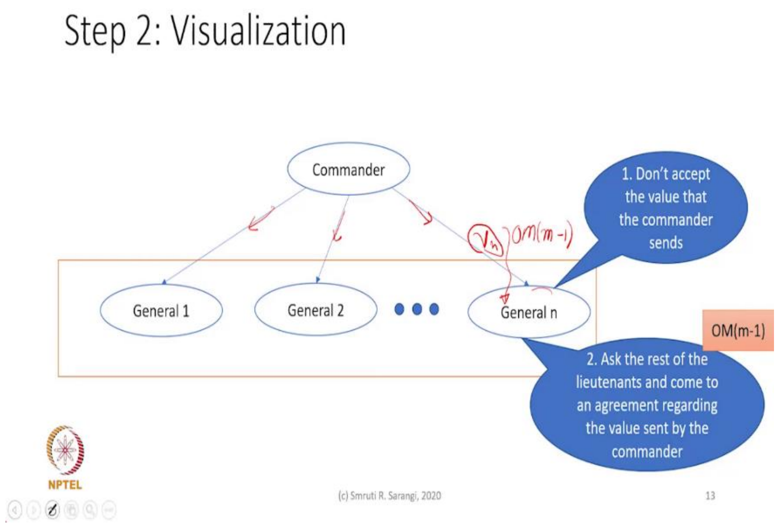
IC1 is what is just written that all the loyal generals come to the, come to an agreement. And IC2, if I were to just take a little bit back IC2 is essentially that if the commander is loyal, every loyal general obeys the order. So which in this case, would basically mean that every loyal general in this set agrees on the value vi.

So what is the big picture? What is the overall picture? Well, the overall picture is that OM (m) starts with the commander broadcasting its value to a set of lieutenant. So if there are total n generals in the beginning, we have (n – 1) lieutenant, all (n – 1) get the values of the commander. If the commander is dishonest, it sends different things.

Now, they need to chat and gossip among themselves to figure out that what is it exactly that the commander is sent? So what each of them does is that each of, each one of them starts an instance of OM (m – 1). So how many instances in total, (n – 1) instances in total, and in each instance, the ith lieutenant broadcast the value of vi that it got from the commander in the beginning of OM (m).

And the rest of the loyal generals have to agree on this value, and what are they agreeing on? They are agreeing on what? The ith general has actually caught him. Essentially, that is what they are agreeing on. And the agreement has to follow the IC1 and IC2 correctness conditions. So, OM (m), essentially ends up making (n – 1) calls to OM (m – 1). And similarly OM (m – 1) ends up making (n – 2) calls to OM (m – 2). So we have a pretty much a factorial like complexity over here.

(Refer Slide Time: 37:48)



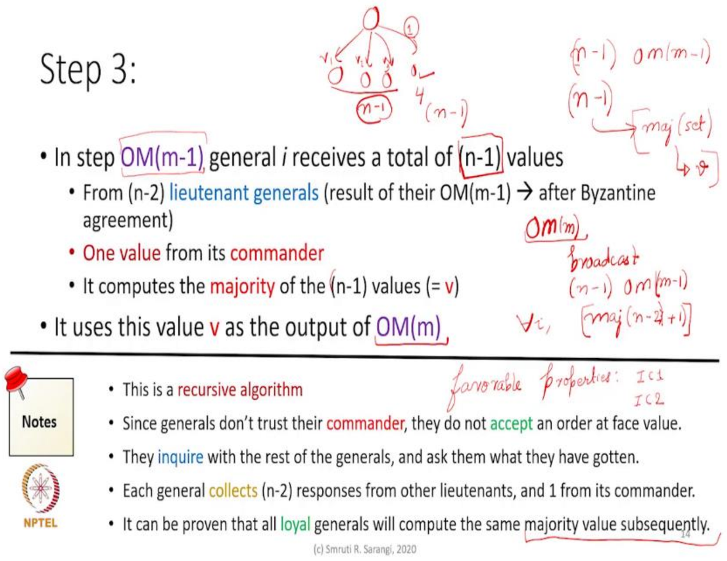
Step 2: Visualization

Commander
General 1
General 2
General n
v_n
OM(m-1)
1. Don't accept the value that the commander sends
2. Ask the rest of the lieutenants and come to an agreement regarding the value sent by the commander
OM(m-1)
NPTEL
(c) Smruti R. Sarangi, 2020
13

So a little visualization over here that the commander at the beginning broadcasts the values, its value to everybody. So let us say that any general does not accept the value that the commander sends. So it simply does not accept. Instead, it asks the rest of the lieutenant and comes to an agreement regarding the value sent by the commander. So essentially it is written in an imperative style, that you do not accept the value and you actually ask the rest of the lieutenant, what is it that they have gotten?

And so, for that reason, it is necessary to broadcast, but again, as we have seen a simple broadcast is useless. That is because generally n itself can be dishonest. So this, again has to be a smaller instance of the original algorithm, which is OM (m – 1). So of course, in this case, the commander is not included. So at the end of this, if this had gotten the value vn, everybody would come to a Byzantine agreement regarding the value that general n got.

(Refer Slide Time: 39:07)



Step 3:

- In step OM(m-1) general *i* receives a total of (n-1) values
  - From (n-2) lieutenant generals (result of their OM(m-1) → after Byzantine agreement)
  - One value from its commander
  - It computes the majority of the (n-1) values (= v)
- It uses this value v as the output of OM(m)

Notes

- This is a recursive algorithm
- Since generals don't trust their commander, they do not accept an order at face value.
- They inquire with the rest of the generals, and ask them what they have gotten.
- Each general collects (n-2) responses from other lieutenants, and 1 from its commander.
- It can be proven that all loyal generals will compute the same majority value subsequently.

(c) Smruti R. Sarangi, 2020

Finally, step 2. So in step OM (m – 1), each general receives a total of (n – 1) values. Well, how is that? It is like this, that the commander actually senses the (n – 1) lieutenants, each one of them starts an instance of, OM (m – 1). So, if I were to consider any node, any process, any general, it would get one value from the commander, and it would get (n – 2) values from the rest of the lieutenant, making it a total of (n – 1) values.

So, which is exactly what is written here, that from (n – 2) lieutenant generals, it gets (n – 2) values. So of course, it gets values, I am not referring to the value that was broadcast. I am essentially referring to the value that was agreed upon after running OM (m – 1). So, essentially after these OM (m – 1) algorithm is concluded. So, if let us say there are four nodes, so it will be clear about the value that v1, about the values v1, v2 and v3.

It will clearly know what are these values. And of course these nodes could be dishonest, but then as we have seen, there will still be an agreement among the loyal generals regarding the value it got, which incidentally might not be the value it actually got, but at least there will be an agreement. So after this, what we do is we have a total of (n – 1) values which are there with each of these lieutenants.

So where, why (n – 1), one it got from its commander at the beginning of OM m, and (n – 2), it got after running the individual instances of the reduced Byzantine problem. So once at the end, so basically, if I were to consider the OM m algorithm, so initially we have a broadcast stage where the commander just broadcasts. Then, we run n minus 1 instances of OM (m – 1).

In these instances, all of these values are agreed upon. And after that, we have a set of (n – 2) values with each, for each i. We, we, we have a set of this, (n – 2) values + 1, which is what was received from the commander for each of these is. So we compute a majority value, which is, we compute the majority function. And whatever we get, that is used as a, as the output of the algorithm OM m.

So for each i what is it that we do? What we do is at the beginning, the commander broadcasts its value to each of the lieutenants. Each lieutenant initiates a round of OM (m – 1). Fair. It tries to obtain an agreement over the value it got from the commander. After that has happened, we end up running (n – 1) instances of OM (m – 1), which means essentially, for each left end, we end up agreeing upon (n – 1) values.

These many, for each lieutenant we agree upon these. So out of this, of course, one value, the lieutenant would have received from the commander in the broadcast stage and the rest (n – 2) would be received because of the other instances of the reduced problem. So once it has these, it can then compute the majority of the set.

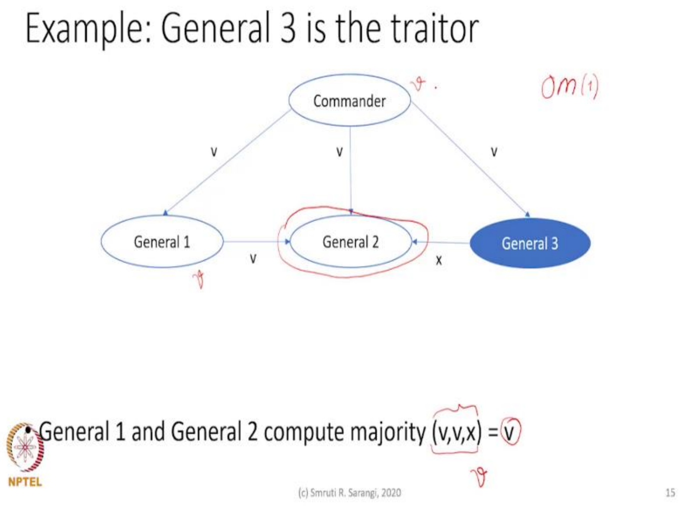
It computes the majority of the set. And let us say the output of that is v. So v is pretty much the output of the OM (m) algorithm. And this v is stored in each other lieutenants. So this is clearly a recursive algorithm with a very factorial like field that initially we run (n – 1) instances, and then (n – 2) instances, (n – 3) for each. So this, so clearly the message complexity is high. So, that is true.

Even though the time complexity is not that high. I mean, it would be incorrect to say that it is not that high. It all depends upon how many messages and how many computations we can fit into a single round. But what should be important is that the number of instances of these, these algorithms and the sheer number of messages sent, that clearly goes up in a, as a factorial function.

So the crux of this algorithm is that nodes do not, generals do not trust their commander. Hence they are not willing to accept any order at face value. For every order, there is a need to consult the others and then come to an agreement. And because of this consultation, we have to run smaller instances of the same Byzantine Agreement algorithm.

And we expect that the majority function that we compute, which is essentially this step over here, has some favorable properties. And what are the favorable properties? So I am using the American spelling here and ignoring the u. So the favorable properties are that IC1 and IC2, the two correctness conditions that we have, both of them should be satisfied.

(Refer Slide Time: 45:25)



Example: General 3 is the traitor

- General 1 and General 2 compute majority (v,v,x) = v

(c) Smruti R. Sarangi, 2020

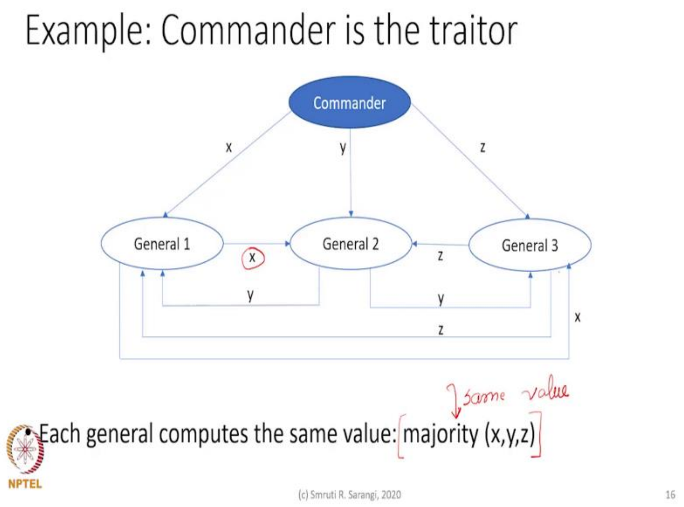
15

So let us give a simple example because the algorithm is, even though it is simple, it is still reasonably complicated. So General 3 is the traitor here in this case. So, essentially what happens? At the beginning, we are going to run OM (1), which means the commander sends its value v to everybody. After that, each one of them runs an instance of OM (0), which is as simple as just simply sending the value.

So I am not showing all the messages, I am only showing those messages that are relevant for General 2. So General 2 gets the message v, v and x. So as you can see, it would compute majority of v, v and x. So, so where x can be anything. So x is a placeholder for any message, because it is coming from a dishonest, from a traitor node.

So the majority of v, v and x is clearly v. So which means that General 2 is going to agree on v. We can do something very similar for General 1. Here also, we will find it will agree on v. And of course, the commander, since it is loyal, it will also agree on v. So we have the Byzantine agreement conditions here.

(Refer Slide Time: 46:48)



Example: Commander is the traitor

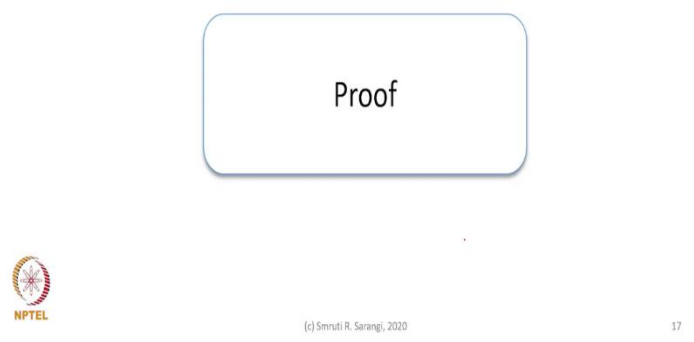
- Each general computes the same value: majority (x,y,z)

Now, if I consider a disloyal commander, so let us assume that in the first round it sends messages, x, y, and z to General 1, 2 and 3. So then in the second round when each of them run the reduced instance of the Byzantine problem, General 1 will send x to General 2, General 1 will also send x to General 3. General 2 will send y to generals 1 and 3, General 3 will send z to General 2 and General 1.

So, if you actually look at it, each of these generals actually receives the same set of messages, x, y, and z, x, y, and z, x, y, and z. And all of them compute the value of the majority function. So the majority function does not depend upon the order of the inputs. So all of them will essentially compute the same value, which is majority x, y, z.

So in this case, of course, condition IC2 is not relevant because a commander is not loyal, but at least all the loyal lieutenant generals who are generals 1, 2, and 3, all of them are going to compute, are going to agree on the same value, which is majority of x, y, and z. Whatever it is, we do not care because it will be the same for all of them.

(Refer Slide Time: 48:25)



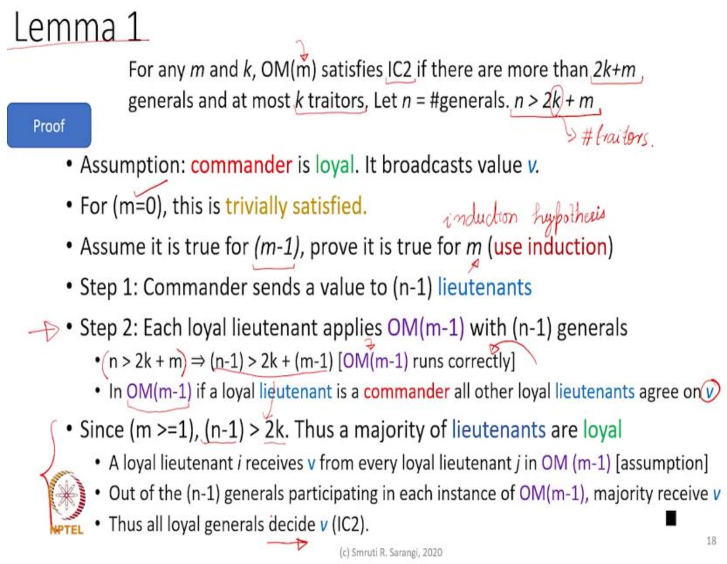
Proof

(c) Smruti R. Sarangi, 2020

17

So now that we have seen the algorithm, it is time to move on to the proof. So the proof is rather elegant, and it is important to appreciate it because algorithm was pretty complicated. So unless its proof is also understood, it will make, it will be hard to retain the key concepts of the algorithm.

(Refer Slide Time: 48:50)



Lemma 1

Proof

For any *m* and *k*, OM(m) satisfies IC2 if there are more than 2k+m generals and at most *k* traitors, Let *n* = #generals. *n* > 2k + *m*

- Assumption: commander is loyal. It broadcasts value *v*.
- For (m=0), this is trivially satisfied.
- Assume it is true for *(m-1)*, prove it is true for *m* (use induction)
- Step 1: Commander sends a value to (n-1) lieutenants
- Step 2: Each loyal lieutenant applies OM(m-1) with (n-1) generals
  - (n > 2k + m) ⇒ (n-1) > 2k + (m-1) [OM(m-1) runs correctly]
  - In OM(m-1) if a loyal lieutenant is a commander all other loyal lieutenants agree on *v*
- Since (m >=1), (n-1) > 2k. Thus a majority of lieutenants are loyal
  - A loyal lieutenant *i* receives v from every loyal lieutenant *j* in OM (m-1) [assumption]
  - Out of the (n-1) generals participating in each instance of OM(m-1), majority receive *v*
  - Thus all loyal generals decide *v* (IC2).

So this is kind of a loaded slide, so we will go slow. So we will first prove a lemma. So for any m and k, where m is the round. So of course, m is not the number of traitors here. So let us not confuse ourselves with that. So for any m and k, OM m satisfies IC2, IC2 was the commander is loyal, all loyal generals obey his order. If there are more than 2 k + m generals, and at most k traitors.

So if we have k traitors and, and more than 2 k + m generals, then we can say that, OM m will satisfy IC2. So let n be equal to the number of generals, n > 2 k + m, where k is the number of traitors, and m is the round number. So the assumption here is that the commander is loyal because if the commander is not loyal, then IC2 is not relevant.

So it broadcasts value v to the rest of the nodes. If m = 0, this is trivially satisfied because all the nodes simply accept what the commander sent. Assume it is true for n - 1. Let us try to prove that it is true for n. So we are essentially using induction. So in induction, the base case is satisfied. Now we want to prove the induction case.

We want prove the induction hypothesis. So we are assuming it is correct till n - 1. So step 1 is commander sense of value to n - 1 lieutenants. This step is correct. So now let us consider step 2. Each loyal lieutenant applies OM (m – 1) with n - 1 generals. So we have the simple math over here that if $n > 2k + m$, which is our assumption, $(n - 1) > 2k + n - 1$.

So the same assumption that we have made over here, the exactly same assumption can be made for OM $(m - 1)$, which means the number of generals for OM $(m - 1)$ is and - 1, that is $> 2k + n - 1$, which means that OM $(m - 1)$, the, since the basic assumption is in place, the induction hypothesis, it runs correctly.

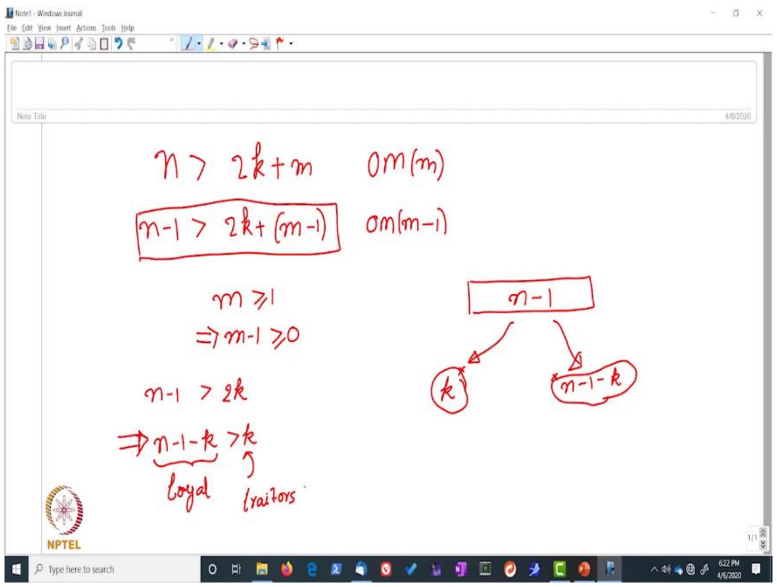
So in OM $(m - 1)$, if let us say, one of the loyal left is as a commander, all other loyal lieutenants will also agree on the value v. So the commander is loyal, it sends value v to all the loyal lieutenants. And if one of them becomes a commander in OM $(m - 1)$, given the induction hypothesis, all other loyal lieutenant will agree on v.

Now, here is the catch. We assumed that in this case, $m \geq 1$. So if $m \geq 1$, and we put it over here, what we are essentially going to get, or, or let us say, if we put it over here, we, we will get that $(n - 1) > 2k$. If $m \geq 1$, this follows from here, this result follows from here.

This means that a majority of the lieutenants are actually loyal. Why? Because we have k traitors and since $(n - 1) > 2k$ $(n - 1 - k)$ is essentially $> k$. So a majority of lieutenants are actually loyal in this set. So loyal lieutenant i receives v from every loyal lieutenant j in OM $(m - 1)$, which is an assumption.

Out of the $(n - 1)$ generals participating in each instance of OM $(m - 1)$, a majority are going to receive v. Hence all the loyal generals will decide on v. So it is important that I elaborate on this part a little bit more.

(Refer Slide Time: 53:42)



$n > 2k + m$ $OM(m)$

$\boxed{n-1 > 2k + (m-1)}$ $OM(m-1)$

$m \geq 1$

$\Rightarrow m-1 \geq 0$

$n-1 > 2k$

$\Rightarrow \underbrace{n-1-k}_{\text{loyal}} > \underset{\text{traitors}}{k}$

$n-1$

k

$n-1-k$

NPTEL

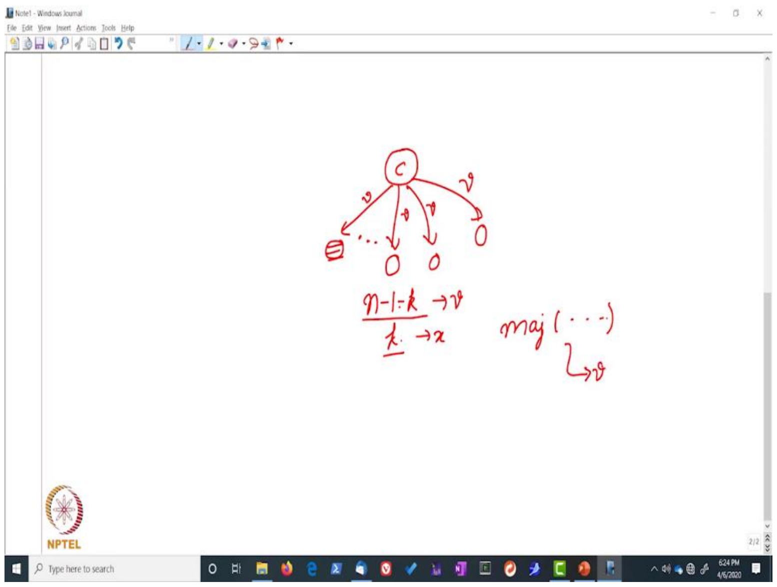
So what was the assumption over here? Well, what our assumption was that n > 2 k + m. So assumption here was that n > 2 k + m, and this was an algorithm. Now, if I consider a reduced algorithm, n - 1 is greater than, so this means that essentially in a lower round, the same hypothesis also holds.

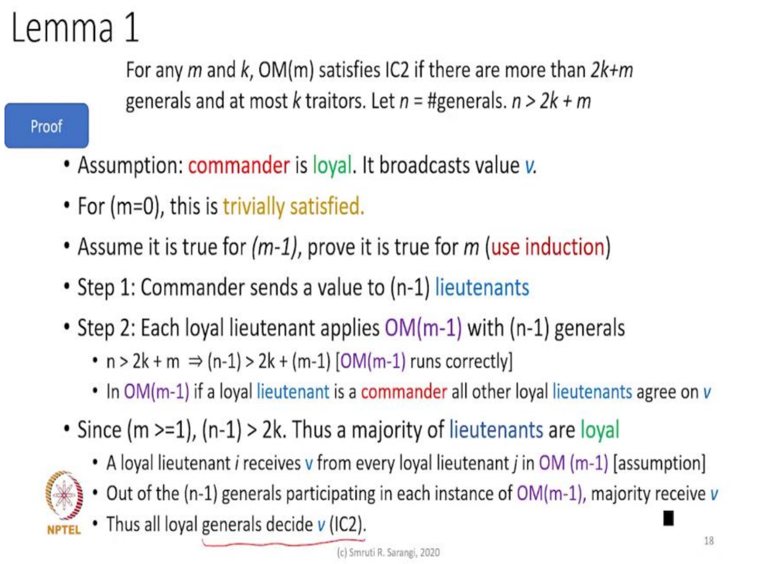
And by our induction assumption, we assume that OM (m – 1) works correctly. So if I were to look at this expression over here, if m > 1, that implies m - 1 > 0. So, so this means that if m is greater than, let us say, m ≥ 1, so m - 1 ≥ 0, so we will then say that n - 1 > 2 k.

That further implies, so what we have, we have a total of n - 1 loyal lieutenants. Out of that, we have k traitors and we have n - 1 - k loyal generals. So these k traitors, we do not really care, but the n - 1 - k loyal generals that we have, they are the ones that we really care about.

So in this case, what we have shown is that this number, which is the number of loyal generals is greater than the number of traitors. So these are essentially the number of loyal generals, and these are the number of traitors. So since the commander, so, now let us see.

(Refer Slide Time: 56:30)





Lemma 1

For any *m* and *k*, OM(m) satisfies IC2 if there are more than *2k+m* generals and at most *k* traitors. Let *n* = #generals. *n > 2k + m*

Proof

- Assumption: commander is loyal. It broadcasts value *v*.
- For (m=0), this is trivially satisfied.
- Assume it is true for *(m-1)*, prove it is true for *m* (use induction)
- Step 1: Commander sends a value to (n-1) lieutenants
- Step 2: Each loyal lieutenant applies OM(m-1) with (n-1) generals
  - n > 2k + m ⇒ (n-1) > 2k + (m-1) [OM(m-1) runs correctly]
  - In OM(m-1) if a loyal lieutenant is a commander all other loyal lieutenants agree on *v*
- Since (m >=1), (n-1) > 2k. Thus a majority of lieutenants are loyal
  - A loyal lieutenant *i* receives v from every loyal lieutenant *j* in OM (m-1) [assumption]
  - Out of the (n-1) generals participating in each instance of OM(m-1), majority receive *v*
  - Thus all loyal generals decide *v* (IC2).

(c) Smruti R. Sarangi, 2020

18

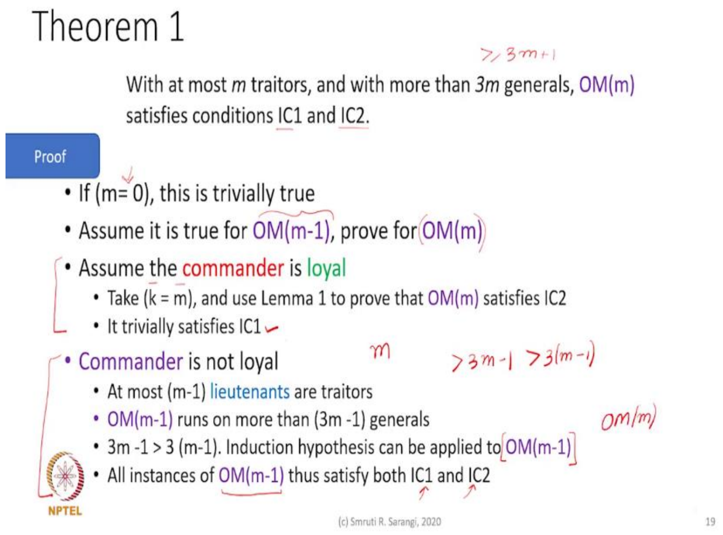
Since the commander is loyal, what the commander essentially did is it sent its value v to everybody. Out of that, the loyal generals, let us assume these are the ones, and let us say, this is a traitor. So, so these are the ones that, so I am not showing all the general. So these, so these are the ones who will again, take the value v and again, broadcast it in OM (m – 1).

So, one thing that is clear is that all the loyal generals are going to send v. And between any two loyal generals, they will also receive v. So every, if I were to consider a loyal general over here, it would receive n - 1 - k values for v, and the remaining it will get from traitors, which can be x and which we do not care.

But since this number, the first number is greater than the second, any majority function of these values is going to return the value v, which is essentially what we care about for a loyal general, that all the loyal generals would receive this, and thus, they would decide on v. And this is exactly IC2, which again, proves the statement of the lemma.

So what was the statement of the Lemma, once again? It was that for any m and k, OM m satisfies IC2, if there are more than 2 k + m generals and at the most k traitors. So this, we have been able to prove via induction. I would request the listeners to, the viewers to go over this proof once or twice till it becomes reasonably clear.

(Refer Slide Time: 58:25)



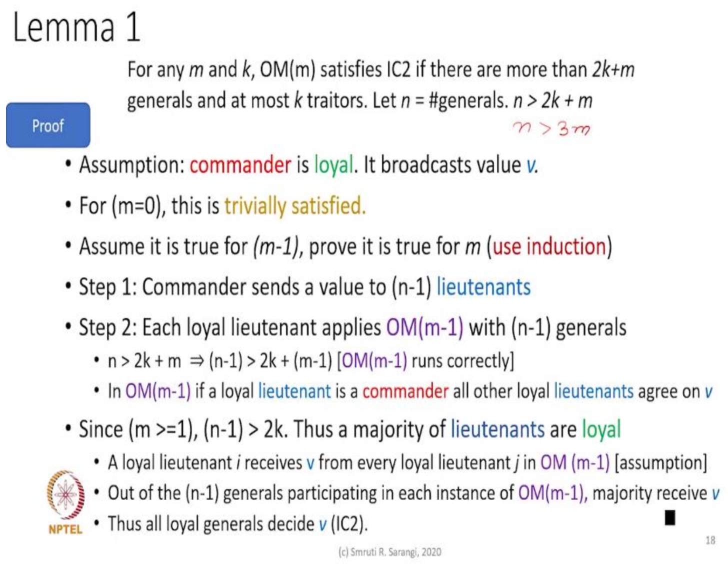
Theorem 1

With at most *m* traitors, and with more than *3m* generals, OM(m) satisfies conditions IC1 and IC2.

Proof

- If (m= 0), this is trivially true
- Assume it is true for OM(m-1), prove for OM(m)
- Assume the commander is loyal
  - Take (k = m), and use Lemma 1 to prove that OM(m) satisfies IC2
  - It trivially satisfies IC1
- Commander is not loyal
  - At most (m-1) lieutenants are traitors
  - OM(m-1) runs on more than (3m -1) generals
  - 3m -1 > 3 (m-1). Induction hypothesis can be applied to OM(m-1)
  - All instances of OM(m-1) thus satisfy both IC1 and IC2

(c) Smruti R. Sarangi, 2020 19



Lemma 1

For any *m* and *k*, OM(m) satisfies IC2 if there are more than *2k+m* generals and at most *k* traitors. Let *n* = #generals. *n > 2k + m*

$n > 3m$

Proof

- Assumption: commander is loyal. It broadcasts value *v*.
- For (m=0), this is trivially satisfied.
- Assume it is true for *(m-1)*, prove it is true for *m* (use induction)
- Step 1: Commander sends a value to (n-1) lieutenants
- Step 2: Each loyal lieutenant applies OM(m-1) with (n-1) generals
  - n > 2k + m ⇒ (n-1) > 2k + (m-1) [OM(m-1) runs correctly]
  - In OM(m-1) if a loyal lieutenant is a commander all other loyal lieutenants agree on *v*
- Since (m >=1), (n-1) > 2k. Thus a majority of lieutenants are loyal
  - A loyal lieutenant *i* receives v from every loyal lieutenant *j* in OM (m-1) [assumption]
  - Out of the (n-1) generals participating in each instance of OM(m-1), majority receive *v*
  - Thus all loyal generals decide *v* (IC2). ∎

NPTEL

(c) Smruti R. Sarangi, 2020

18

Given that we have proven this key lemma, so now we can prove the theorem. So the theorem says that with at most m traitors and with more than 3m generals, which means ≥ 3m + 1 generals, OM m or Byzantine algorithms satisfies the conditions IC1 and IC2. Well, if m = 0, this is trivially true, as we are just seen, there are no traitors.

So assume it is true for OM (m – 1), again, proof by induction with the base case being proven here. And let us try to prove it for OM m. So assume that the commander is loyal. So in this case, in the previous lemma, I take k = m and I use Lemma 1. So with Lemma 1, if k = M, then m > 3m.

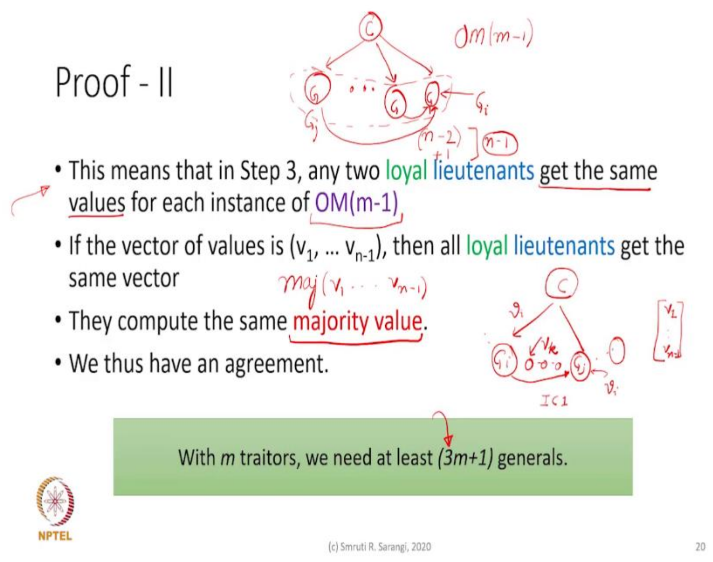
And with at most m traitors, we use this to proof that it satisfies IC2. And if it satisfies IC2, which means that if all the loyal generals obey what the commander is saying, and the commander is loyal, this trivially satisfies IC1. So because of the previous lemma, proving this condition where the commander is loyal, actually became trivial.

Now, let us consider the case for the commander is not loyal. So again, several simple points. At most m - 1 lieutenants are traitors. So that is because there are am traitors. And one of them is a commander, so n minus 1 left lieutenant are traitors. So, OM (m – 1) runs on, so for OM (m – 1), it runs on more than 3m - 1 generals.

And 3m - 1 > 3(m – 1). So the same assumption we are making at the top, the same holds for OM (m – 1). And if the assumption holds, then the basic conditions for OM (m – 1) are there. So the induction hypothesis can be applied to the reduced Byzantine problem.

And the induction hypothesis will then basically say that all instances of OM (m – 1) will satisfy IC1 and IC2. So let us make this assumption and try to prove the induction case where OM (m) also satisfies IC1 and IC2.

(Refer Slide Time: 61:10)



Proof - II

- This means that in Step 3, any two loyal lieutenants get the same values for each instance of OM(m-1)
- If the vector of values is $(v_1, \ldots v_{n-1})$, then all loyal lieutenants get the same vector
- They compute the same majority value.
- We thus have an agreement.

With *m* traitors, we need at least *(3m+1)* generals.

(c) Smruti R. Sarangi, 2020 20

C
OM(m-1)
G
G
G
G
G_i
(n-2)
+1
(n-1)
maj($v_1 \ldots v_{n-1}$)
v_i
v_k
IC1
v_1
v_{n-1}
NPTEL

So what this means is that in Step 3, any two loyal lieutenants get the same values for each instance of OM (m – 1). So if I were to explain the Byzantine algorithm to anybody, I would say that this is by far the most crucial and critical step, which in my view should be visualized like this, that if these are all the lieutenants.

And if I am assuming that OM (m – 1) runs correctly, then what happens is that in a second step we have (n – 1) instances of this algorithm, and pretty much after the algorithm ends, pretty much the output of each algorithm, it is not explicitly sent because let us assume that this is, in any given Gi. So it is not explicitly sent, but what we have seen is that this is inferred, but everybody infers the correct thing.

So now, I am assuming that in OM (m – 1), the induction hypothesis, it runs correctly. Gi was a party to, (n – 2) instances of this algorithm being initiated by its fellow lieutenants, and also it received one value from its commander, which it used to seed an instance of OM (m – 1). So if I were to add that, all the (n – 1) values that it gets, assuming that if Gi is loyal and another Gj is loyal, we claim that they get the same set of values.

This is very important, and this is the crux of the entire proof. And the question is, why would they not actually get the same set of values? So they will not get the same set of values pretty much under the following circumstances. So let us try to understand what are those circumstances which might lead us to think that they will actually not get the same set of values.

So let us, there is a need to elaborate on this. So let us maybe draw another diagram over here. And let us only consider two of these loyal generals, Gi and Gj, and there are a few more in the middle. So for all the generals in the middle, each one of them initiates a round of OM (m – 1). And since we assume that they run correctly, by the IC1 assumption, whatever value that let us say, another vk got, both Gi and Gj both of them are going to agree on this value of v k.

So then, let us consider the value vi that Gi got. Since Gi will be a commander of its own instance of OM (m – 1), and it will send the value to Gj, and so it will essentially initiate round of OM (m – 1), Gj is also going to agree that Gi actually got vi because of IC2, and the same will also hold for Gj. So this essentially means that for each of these individual sub algorithms, algorithms with a reduced input, all the loyal lieutenants will actually end up with the same set of values, for the same v 1 to v n - 1.

It is going to be the same set. There is no reason why it should be a different set. And the reason is very simple that for each of these OM (m – 1) instances, if it is running correctly, each of it will correctly produce a Byzantine agreement. So whatever it is that they have actually gotten from their commander will reflect in the set of the other loyal generals.

And it is possible that one of them can be a traitor, but that does not matter because all the loyal generals will come to an agreement of about what the traitor must have gotten. It does not matter if the traitor got that or not. As long as the rest of the loyal generals agree, IC1 is satisfied, which is anyway, what we care about.

Given that the vector of values is the same for all the loyal lieutenants, the majority of this vector, whatever it is, will be the same for all the loyal lieutenants. And that is the key, that is the crux. So if you go back to our example, something very similar was happening.

So, essentially the majority of the same set, majority of the same set or vector will be the same for everybody, for all the loyal lieutenants. So they are going to compute the same majority value. And hence, thus, we have an agreement. So with m traitors, what this means is we need at least 3m + 1 generals.

And then we showed an algorithm, we showed that it is possible to get byzantine agreement. So this particular algorithm that we showed is actually very, very expensive in terms of computation, in terms of resources. And it is essentially a cursive algorithm that, whose complexity is essentially a factorial complexity. So it is clearly more than exponential.

(Refer Slide Time: 67:27)

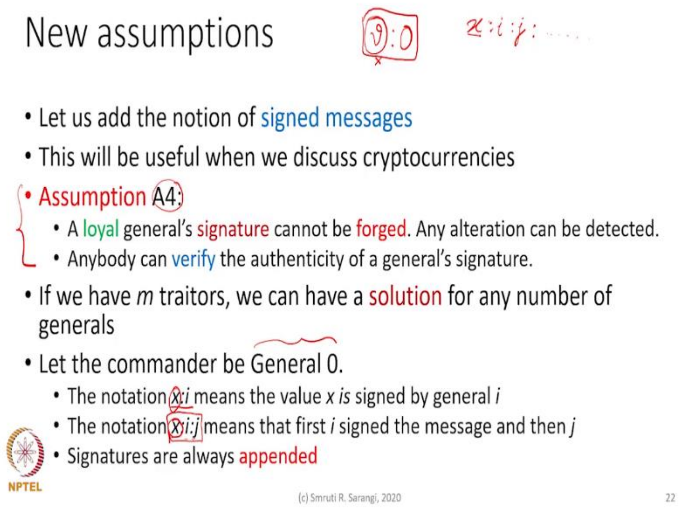
Solution with Signed Messages

(c) Smruti R. Sarangi, 2020

21

And so that is the reason typically in most instances of Byzantine consensus, we make some simplifying assumptions. So let us make one search assumption, which is a solution with signed messages.

(Refer Slide Time: 67:43)



New assumptions

$v:0$ $x:i:j:\ldots$

- Let us add the notion of signed messages
- This will be useful when we discuss cryptocurrencies
- Assumption A4:
  - A loyal general's signature cannot be forged. Any alteration can be detected.
  - Anybody can verify the authenticity of a general's signature.
- If we have *m* traitors, we can have a solution for any number of generals
- Let the commander be General 0.
  - The notation *x:i* means the value *x is* signed by general *i*
  - The notation *x:i:j* means that first *i* signed the message and then *j*
  - Signatures are always appended

(c) Smruti R. Sarangi, 2020

22

So here, what we will do is that we will create an additional assumption and we will find that such kind of Byzantine solutions with signed messages are common in the world of cryptocurrencies. So that we will have a separate lecture for that. Here we add an additional assumption, which is A4. So, we claim that this will make our life significantly easier.

Here, our claim is that a loyal general signature cannot be forged. So it is not possible to forge the signature of a loyal general. We can detect any kind of an alteration. So what this furthermore means that if let us say the commander is sending a value v, the value v cannot be tampered.

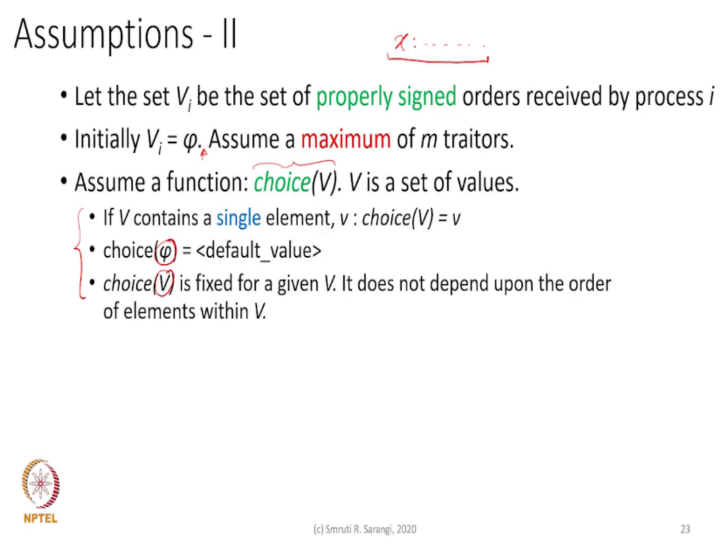
So if let us say the commander is sending value v, and the commander is signing it with, so we will use the colon. The left side of the colon is a value, and the right side is a signature of who is signing it, and let the commander be general number 0. So if there is a message of this type, this message cannot be tampered with. So this, there's no way that this message can be forged or tampered.

So if this is what the commander has said, well, this is what it is said. So we will use a notation x: i to basically mean x is the message, which has been signed by General i. Something of the notion x i: j would mean that, well, the contents of the message are x, it was first signed by i then by j. And of course, this, this particular sequence, this is not tamper able. It is not possible to with it.

All that we can do is we can append signatures. And we will see why it is done this way when we discuss more about cryptocurrencies and so on. But at the moment, the contents of the message and also the existing set of signatures cannot be changed. Only thing that a node can do is append to this list. That is all.

So we will find that if we have such kind of a facility available, then it is possible to have a significantly simpler solution for this problem of Byzantine generals. That is mainly because we will, this non-tamperablitiy will give us, will give us the ability to significantly simplify the way we do things. And we do not have to run this factorial style competition.

(Refer Slide Time: 70:26)



Assumptions - II

$x : \underline{\cdots\cdots}$

- Let the set V_i be the set of properly signed orders received by process *i*
- Initially $V_i = \varphi$. Assume a maximum of *m* traitors.
- Assume a function: *choice(V)*. *V* is a set of values.
  - If *V* contains a single element, *v* : *choice(V) = v*
  - choice(φ) = <default_value>
  - *choice(V)* is fixed for a given *V*. It does not depend upon the order of elements within *V*.

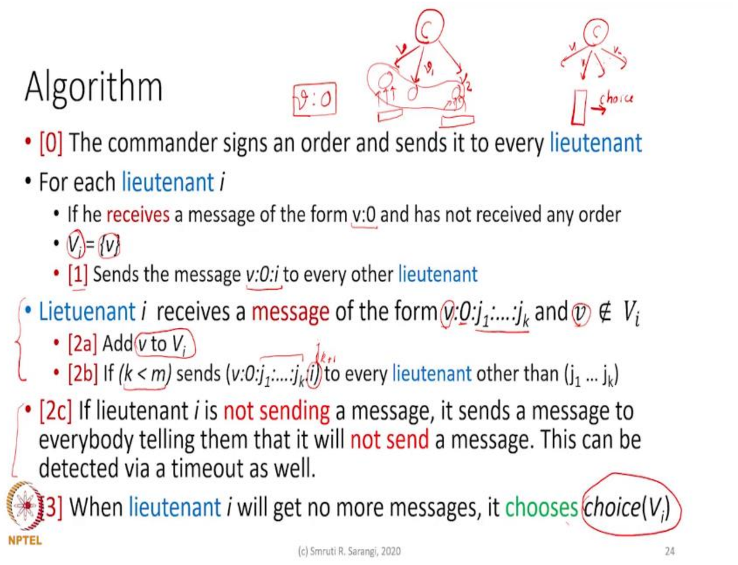
(c) Smruti R. Sarangi, 2020 23

So a few more assumptions. So let V i be the set of properly signed orders. Well, a properly signed order is essentially the value of the order, plus a range of signatures. And this entire message has not been tampered with. So then it is a properly signed order.

So initially for each general, V i = ϕ, it is empty. Additionally, assume a maximum of m traitors, which is a standard assumption that we have been making up till now. Assume a function choice of V. So V is a set of values and choice of V has certain properties. Let us see what it is. If V contains a single element, choice of V is that single element, small v.

If the set V, capital V is empty, then this is a default value. So, choice of V otherwise is fixed for a given vector. So, for a given vector or set of values, the choice is fixed, and it does not depend upon the order of the elements within V. So it is independent of the order. And for a given set, the choice of V similar to the majority function for a given set, the choice is fixed.

(Refer Slide Time: 71:52)



Algorithm

- [0] The commander signs an order and sends it to every lieutenant
- For each lieutenant *i*
    - If he receives a message of the form v:0 and has not received any order
    - $V_i = \{v\}$
    - [1] Sends the message *v:0:i* to every other lieutenant
- Lietuenant *i* receives a message of the form $v:0:j_1:\ldots:j_k$ and $v \notin V_i$
    - [2a] Add *v* to V_i
    - [2b] If $(k < m)$ sends $(v:0:j_1:\ldots:j_k:i)$ to every lieutenant other than $(j_1 \ldots j_k)$
- [2c] If lieutenant *i* is not sending a message, it sends a message to everybody telling them that it will not send a message. This can be detected via a timeout as well.
- [3] When lieutenant *i* will get no more messages, it chooses $choice(V_i)$

(c) Smruti R. Sarangi, 2020 24

Now, the algorithm goes as follows. It is a very simple algorithm, substantially simpler than what we have seen. The proof is also substantially simpler. So the commander signs an order and sends it to every left. So the commander's value. So here also, we make the same assumptions about loyalty. Commander can be disloyal, so it does not matter. It essentially takes a value and essentially signs it. And commander is General 0, sends it to every lieutenant.

For each Lieutenant i, if he receives a message of the form v : 0, which means from the commander and has not received any order, so then it set V i. So lieutenant i sets, it set V i equal to the value that it receives. And then so, step 1, it sends the message v 0 i to every other lieutenant. So, what happens? So what happens is for the first message that any lieutenant gets in this synchronous algorithm, it adds the value V to its set and it appends its signature v 0 i and sends it to every other lieutenant.

So now, let us consider the steady state case. So steady state case is that Lieutenant i receives a message of the form, the value V initially signed by the commander and then a sequence of signatures from j 1 to j k. And the value V is not present in its own set. Well, how can this happen? Well, this can happen if the commander is not loyal, then the commander will essentially be sending different values to different nodes. All of them will have its own signature, which is fine.

So, then the nodes will be exchanging these messages among each other, which was similar to what we were doing in the previous algorithm. But in the only differences that in this case, these are signed messages. So the advantage of the sign messages is that it is possible to exactly know who has received what, and then if the rest of the lieutenants just exchange these signed messages, just between themselves, they will get an idea of every single message that the commander has sent.

So, this is important that the lieutenant generals cannot create any new values. So this was not the case in the previous algorithm, but in this case, all the messages have to be initially signed by the commander whose values and the values of those messages cannot be forged. So essentially, any values that is there in the system, all of those values have to ultimately be generated by the commander because nobody else can.

So, the task of this algorithm is just to get all the values that the commander has generated and sent, just in case it is disloyal, and to ensure that all of those values are there with all of the loyal generals. So, if a loyal general has all the values, which the commander has generated and sent in the first round, and if they get the set, then they use the choice function. And as long as the set of values is the same, the output of choice of these also the same.

So, what is the main aim of this algorithm? The main aim of this algorithm is just to circulate all the values that the commander has generated and ensure that they reach all the loyal lieutenants. So, the steps 2a and 2b are vital in doing that. So in this case, if a message is received with the form v 0 j 1 to j k, and if v is not there in the set V i, so it is one of those traitors values that the commander has generated, then we add v to V i.

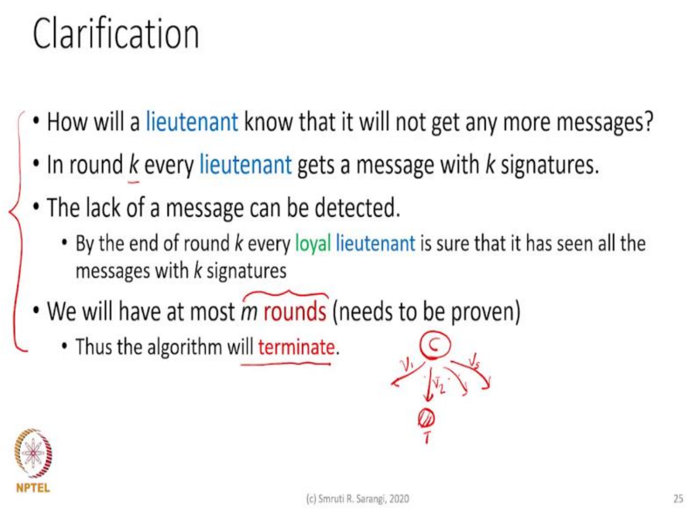
And if $k < m$, so k is pretty much equal to the number of signatories of this message, other than the commander itself. So this is the value of k. Please do not confuse it with the k we have been using in the previous algorithm. So if $k < m$, then lieutenant I sends. So it appends its own signature, which essentially becomes j k + 1 to the message, and sends it to every other lieutenant, which is not in the original set j 1 to j k.

And of course, it does not send a message to itself, so its sends a message to every other lieutenant. And the message is basically that a new value has been discovered. In any round, if a certain lieutenant does not send a message, it should send a message to everybody telling them that it will not send a message or this can be detected via timeout, and a default value can be inferred.

When Lieutenant i will get no more messages, it chooses choice of V i. And what we have just discussed that as long as every single value, which the commander generated and sent is all of those values are there with all of the loyal lieutenants, they have a set of all the unique values.

And the moment they compute the choice function on it, the output is guaranteed to be the same, and this is essentially the Byzantine consensus, the Byzantine agreed value. So what is the aim of this algorithm, is just to circulate all the values that the commander has sent to everybody among all the nodes. That is the key, that is the key aim.

(Refer Slide Time: 78:05)



Clarification

- How will a lieutenant know that it will not get any more messages?
- In round *k* every lieutenant gets a message with *k* signatures.
- The lack of a message can be detected.
  - By the end of round *k* every loyal lieutenant is sure that it has seen all the messages with *k* signatures
- We will have at most *m* rounds (needs to be proven)
  - Thus the algorithm will terminate.

(c) Smruti R. Sarangi, 2020

25

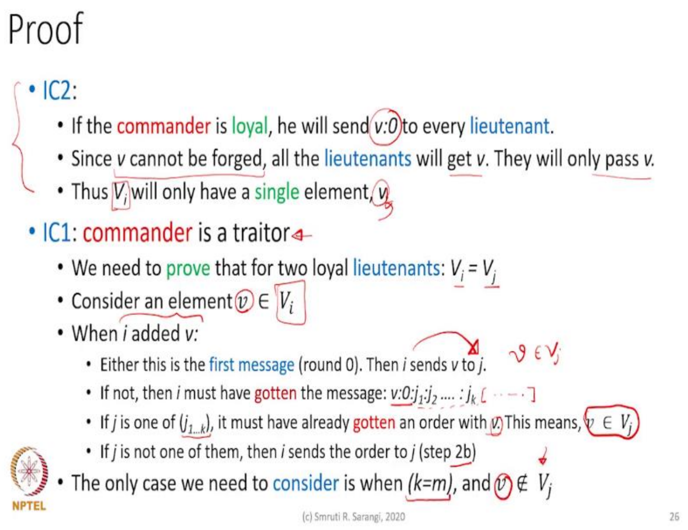
So what, so a quick clarification. So how will a lieutenant know that it will not get any messages? Well, so how do we terminate? So in round k every lieutenant gets a message with k signatures. So other than the commander, of course, so the lack of a message can be detected. So that by the end of round k, every loyal lieutenant is sure that it has seen all the messages with k signatures. So this can be insured.

So this is something we need to proof that we will require at the most m rounds where m is the number of traitors, and ultimately the algorithm will terminate. So before a question is asked, let me answer, it is the most common question, that look, if the commander sends different values to different people, it should just take a single round.

So if it has sent, let us say five different values to different people, it should just take a single round where all of them just sent all the values that they have gotten to everybody, and then they can compute a choice on the values. However, it is not that simple. The reason is that we might have, maybe V 2 was sent to a lieutenantet, which is actually a traitor.

So what the traitor would actually do is that it will actually keep quiet and it is not going to send the value to everybody. So this value will not circulate. And since this value will not circulate, it requires many more rounds to ensure that all of these values that have been sent, all of them are in circulation till the point of termination is achieved.

(Refer Slide Time: 80:05)

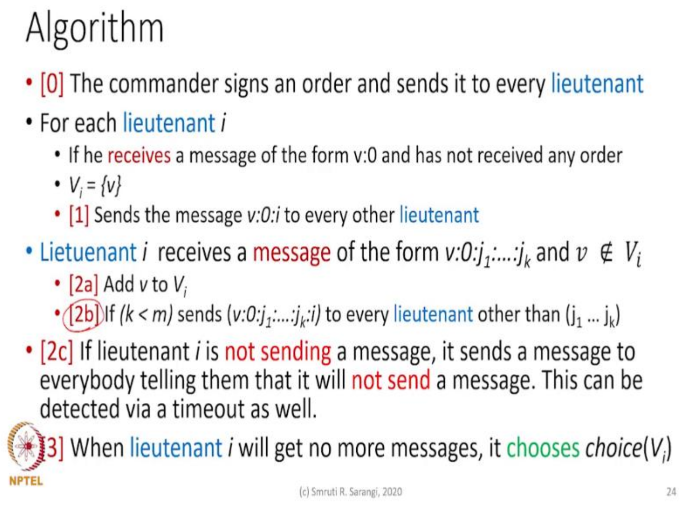


Proof

- IC2:
  - If the commander is loyal, he will send $v{:}0$ to every lieutenant.
  - Since v cannot be forged, all the lieutenants will get v. They will only pass v.
  - Thus V_i will only have a single element, v
- IC1: commander is a traitor
  - We need to prove that for two loyal lieutenants: $V_i = V_j$
  - Consider an element $v \in V_i$
  - When i added v:
    - Either this is the first message (round 0). Then i sends v to j. $v \in V_j$
    - If not, then i must have gotten the message: $v{:}0{:}j_1{:}j_2 \ldots : j_k$ [· · — ·]
    - If j is one of $(j_{1\ldots k})$, it must have already gotten an order with v. This means, $v \in V_j$
    - If j is not one of them, then i sends the order to j (step 2b)
- The only case we need to consider is when $(k=m)$, and $v \notin V_j$

(c) Smruti R. Sarangi, 2020

26



Algorithm

- [0] The commander signs an order and sends it to every lieutenant
- For each lieutenant *i*
    - If he receives a message of the form v:0 and has not received any order
    - $V_i = \{v\}$
    - [1] Sends the message *v:0:i* to every other lieutenant
- Lietuenant *i* receives a message of the form $v{:}0{:}j_1{:}\ldots{:}j_k$ and $v \notin V_i$
    - [2a] Add *v* to V_i
    - [2b] If $(k < m)$ sends $(v{:}0{:}j_1{:}\ldots{:}j_k{:}i)$ to every lieutenant other than $(j_1 \ldots j_k)$
- [2c] If lieutenant *i* is not sending a message, it sends a message to everybody telling them that it will not send a message. This can be detected via a timeout as well.
- [3] When lieutenant *i* will get no more messages, it chooses *choice*(V_i)

NPTEL

(c) Smruti R. Sarangi, 2020

24

So let us now prove algorithm. The proof is reasonably simple. So we will first prove IC2. So if IC2 says if the commander is loyal, then what happens? Well, then only a single message of the form v 0 will be sent to every lieutenant. This eases up our job significantly. Since v cannot be forged, all the lieutenants will get v. They will only pass v.

So regardless of the number of rounds, the only value that will be in circulation will be v. So all the sets, set of values with each node V I, all of them will only contain a single element v. So the choice functions will only return v, and hence Byzantine consensus is achieved. So this was an easy case.

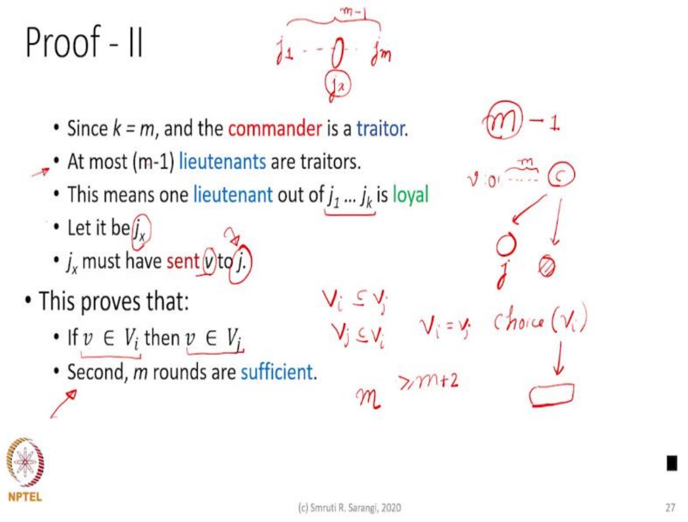
Now, consider the more difficult case for the commander is a traitor. So this is exactly what this algorithm was designed for. So we need to prove that for two loyal lieutenants, V i and V j, they actually get the same set. So consider an element v element of V i. So what is it that happened when i actually added v?

So when i added v, either, this is the first message, which is round number 0. If this is the first message, then i would have sent the value of V to j. So then we would have definitely been in V j. If not, then I must have gotten a message of the form v 0 j 1 to j k. If j is one of j 1, to j k, this means that j has put its valid signature on the message. So it must have gotten an order with v, and v thus must be there in J set also.

Now assume that j is not one of them. If j is not one of them, well, that is also simple. This means that j is not in j 1 to j k. So, i, in any case, if you see the steps in our algorithm. So if we just look at step 2b, if k < m, then what i does is it sends this message to every other lieutenant other than, of course, the signatories of the message.

So in this case, we are assuming that j is not part of the original list of signatories. So i will send v to j. So j will anyway have it. So if j is not one of them, i will send it, which is step 2b. The only case that we need to consider when an element v that is there in capital VI is not there in j's set is actually when i get the message and its last round, when k = m, and this value is not there in V j's set. So what do we do now?

(Refer Slide Time: 83:30)



Proof - II

- Since $k = m$, and the commander is a traitor.
- At most (m-1) lieutenants are traitors.
- This means one lieutenant out of $j_1 \dots j_k$ is loyal
- Let it be j_x
- j_x must have sent v to j.
- This proves that:
  - If $v \in V_i$ then $v \in V_j$
  - Second, m rounds are sufficient.

(c) Smruti R. Sarangi, 2020

27

$\overbrace{j_1 \dots 0 \dots j_m}^{m-1}$
j_x
$m - 1$
$v : 0 \overset{m}{\cdots}$ c
j
$V_i \subseteq V_j$
$V_j \subseteq V_i$
$V_i = V_j$
choice (V_i)
$\geq m+2$
m

So what we do now is what we need to prove. So since k = m and the commander is a traitor, what can we assume? At most m - 1 lieutenant are traitors. So we are assuming a total of m traitors. One of them is the commander who is the traitors. So we have m - 1 lieutenants are traitors. This means one left end out of j 1 to j k. So, so what is the crucial point. Before it is asked, let me explain.

So in the last round, which is the mth round, every single message needs to have m signatures. So every single message has to have m signatures. So it is not possible for one of the traitor nodes to actually wake up in the last round and start sending messages. That is because it is not going to be able to actually get m signatures. So the message is going to be rejected. All the loyal lieutenants are going to reject the message.

What that traitor can actually do is it can try to play mischief. And instead of sending messages to everybody, just send it to a small set and try to exclude one of the nodes, which in this case General j, the node corresponding to General j, which will not receive the message. So we want to claim that this is not possible. The way we will do it is if we consider the set of all the signatories in the last round, which is j 1 to j m, m - 1 of these are traitors.

Not m, because the commander is a traitor, which means one of them is actually honest. If one of these signatories is honest, what this signatory would have done is that when it got the value v, it would have sent the value to all the nodes who actually do not have that value. And since j is not there in the set, j x must have sent a message to j. So this is very important.

And let me repeat, since there are m signatories in the last round with at most m - 1 traitors, one of them must have been honest, and that node must have sent v to j. So this proves that if v is element of V i, then v is also an element of V j, and we can also prove the reverse also. So, so this basically means that the set V i is essentially as subset of V j. But then also we can use the same argument to prove that V j is also subset of V i. So essentially, these are equal.

So for any two loyal nodes, their sets are actually equal. So then what we say is that m rounds. So this further proofs that m rounds are sufficient, mainly because we will have a minus 1 traitors. So one of those signatories has to be honest, and that will ensure that all the rest of the nodes get the value. So all the values that the commander would have sent in the first, in the 0th round will all become visible, all get stored.

And the choice function on the same sets, same set of V i or V j will return the, exactly the same value, whatever is the value, we do not really care, but it will be the same value. And this pretty much establishes Byzantine agreement. So this completes the proof for our Byzantine consensus algorithm, which is a form of a command consensus.

And, we looked at a very expensive solution, but we also looked at a better solution where even if we have m traitors, as long as we have more than m nodes, well, that we will have otherwise it will become aqueous problem. So even if we have m + 1 nodes, then of course it is trivial, but anything ≥ n + 2 is when the problem is interesting.

And we can, using signed messages of this type, we can solve the consensus problem, where of course the signatures can be appended. So we will see something very similar in principle when we discuss blockchains.

(Refer Slide Time: 88:13)

References

1. Lamport, Leslie, Robert Shostak, and Marshall Pease. "The Byzantine generals problem." *Concurrency: the Works of Leslie Lamport*. 2019. 203-226.

(c) Smruti R. Sarangi, 2020 28