


**Secure Computation: Part II**  
**Prof. Ashish Choudhury**  
**Department of Computer Science and Engineering**  
**Indian Institute of Information Technology, Bengaluru**

**Lecture - 02**  
**Reliable Broadcast and Byzantine Agreement**

Hello everyone, welcome to this lecture. In this lecture, we will discuss about Reliable Broadcast and Byzantine Agreement. These are two very important problems in distributed computing.

(Refer Slide Time: 00:33)

## Lecture Overview



- ☐ What is reliable broadcast ?
- ☐ What is Byzantine agreement ?
- ☐ Relation between reliable broadcast and Byzantine agreement

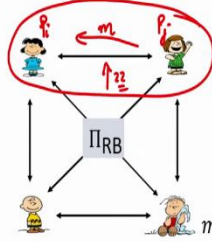
So, we will formally state the problem statement of reliable broadcast and Byzantine agreement, and we will see the relationship between these two problems.

(Refer Slide Time: 00:44)

## Reliable Broadcast (RB)

*Complete network*

- Synchronous system,  $n$  parties, connected by pair-wise secure channels
- A designated sender party with input  $m$
- At most  $t$  Byzantine corruptions --- **sender could be potentially corrupt**



So, let us start with the problem of reliable broadcast often called as the RB problem. So, what we are given here? We are given a synchronous system of  $n$  parties, and when I say synchronous system, I mean to say that channel through which the parties are connected with each other have bounded delays; that means, there is strict upper bound on the message delays, and everyone will be knowing within how much time an expected message is supposed to be delivered.

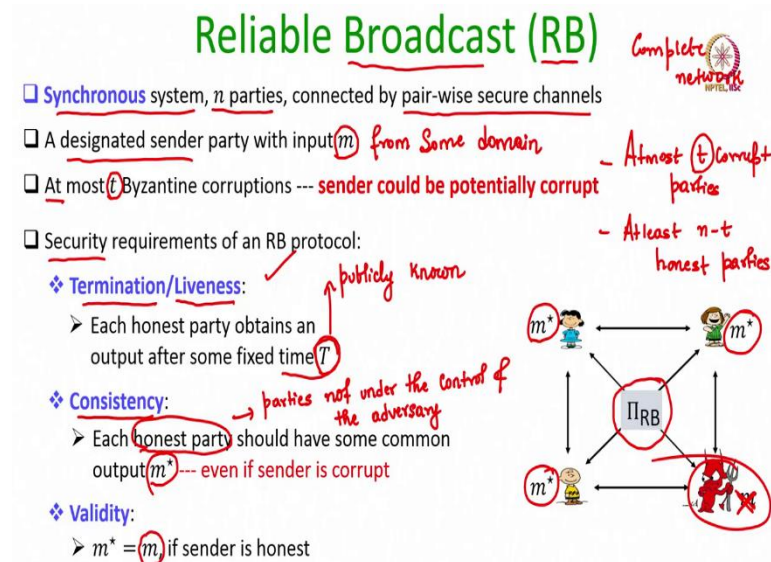
We also assume a pair-wise secure channel model here; that means, we assume that between every pair of parties there is some secured channel already available through which the parties that corresponding pair of parties can exchange messages securely. So, for instance if this is the  $i$ th party and if this is the  $j$ th party, then we assume that there is a secure channel between  $i$ th party and  $j$ th party.

Any message which  $i$ th party receives over this channel will be known to have come from this entity  $P_j$ ; that means, the identity of the senders will be known. And if  $P_i$  and  $P_j$  are honest parties, honest in the sense that they are not under the control of the adversary, then since this is a secure channel, no one can figure out what exactly is getting communicated over this channel between the  $i$ th party and the  $j$ th party.

So, in some sense, we are assuming here that we have  $n$  parties who are part of a complete network. The underlying communication model is modeled as a complete network of  $n$  nodes where  $i$ th party is the  $i$ th node of the network. And one of these  $n$  parties is a

designated sender party. Say, this party is the sender party; it will be known as part of the protocol specification who is the designated sender.

(Refer Slide Time: 03:03)



And there is some input available with this sender from some domain. Again, the domain will be known, but what exactly is the sender's message will not be known beforehand. So, in its simplest form the message  $m$  could be just a single bit, or it could be an enormously large message and we are in the Byzantine corruption setting. So, we assume that among these  $n$  parties at most  $t$  could be controlled by a Byzantine adversary.

And the most important part here is that that sender could also be potentially corrupt. This need not be always the case, but among those  $t$  parties, sender could be one of the entities. Now, we want to design a protocol, let us denote it by  $\pi_{RB}$ . And when I say protocol, it will basically be a sequence of instructions for each party and that protocol should achieve three security properties.

The first property is the Termination or the Liveness property, which demands that each honest party should obtain an output after some fixed time  $T$  which will be publicly known. That means, it should not be the case that the protocol never terminates at all, and the parties keep on running the protocol forever.

There should be some well-known, fixed time within which all the parties complete their respective instructions of the protocol and obtain an output. That is called the termination

property, termination requirement. It is often called as the liveness requirement. The second requirement from this protocol is that of Consistency. So, what does this broadcast in the term reliable broadcast signify?

The term broadcast signifies that the sender would like to send its message identically to everyone. It is like some broadcaster is relaying or broadcasting a live telecast of a cricket match right. So, it should not happen that the broadcaster is showing different viewers, different version of the match. All the viewers of that broadcaster should see the same version of the live telecast right, even if the broadcaster gets potentially corrupt.

So, that requirement is captured through the consistency property which demands that each honest party and when I say honest party, I mean to say the parties not under the control of the adversary. So, remember we have  $n$  parties out of which at most  $t$  could be corrupt and at least  $n - t$  will be honest.

So, at most  $t$  corrupt parties and at least  $n - t$  honest parties, the exact identity of the  $t$  corrupt parties and the exact identity of the honest parties need not be known, but the parameter  $t$  and  $n - t$  will be publicly known. So, the consistency requirement is that each honest party, after time  $T$ , should have a common output, call it  $m^*$ , even if the sender is corrupt during the protocol execution.

So, if during the protocol, sender is corrupt and it is trying to send different versions of its message to different honest parties, still there should be interaction happening according to the protocol  $\pi_{RB}$  among the honest parties, which should ensure that after interaction everyone is on the same page and has a common output  $m^*$  that is a consistency requirement.

And this  $m^*$  could be different from  $m$  if the sender is corrupt. I stress this, because if the sender is corrupt then it in the first place may not have a  $m$  as its input, it might start with some garbage input. We require that even if sender starts with a garbage input, at the end of the protocol everyone should have a common output that is a consistency requirement.

And the third requirement is the Validity requirement. The validity requirement demands that this common output  $m^*$ , which all the honest parties are going to obtain after the time  $T$ , should be equal to the senders message or senders input  $m$  if sender was honest during

the protocol execution right. So, you see for the consistency and validity property one is required with respect to a corrupt sender, one is required with respect to a honest sender.

We require that if the sender is honest, then let the parties execute the protocol. They interact among themselves to find out whether sender is cheating or not. At the end of time  $T$  everyone should output  $m$  if the sender is honest during the protocol execution. And consistency requirement demands that even if the sender was corrupt during the protocol execution through the interaction at the end of time  $T$ , everyone should output an identical message call it  $m^*$  which need not be the sender's input.

Now, why is this validity property is required or stated as one of the requirements of the RB protocol? Because, if we do not have this validity requirement then this reliable broadcast problem is very easy to solve. Everyone can just output a default  $m^*$ , say 0, everyone that will ensure that termination is satisfied because everyone just has to output a bit; they do not need to interact they do not even need to talk with each other.

And consistency is by default satisfied because everyone is outputting a default value. It is a validity property which makes the problem more interesting. If everyone just outputs by default 0, then that 0 may not be the sender's message  $m$ ; sender may have the bit 1. We would require that through interaction the parties should identify that the sender's message was 1 and they output 1, instead of outputting a default value. So, if we do not have this validity requirement then the problem is very trivial to solve. So, that is a reliable broadcast problem.

(Refer Slide Time: 10:58)

## Related Problem: Byzantine Agreement

☐ Synchronous system,  $n$  parties, connected by pair-wise secure channels  
☐ Each party has a private input (from some publicly-known domain)  $\{0,1\}$   
☐ At most  $t$  Byzantine corruptions  
☐ Security requirements of a BA protocol:

- ❖ **Termination/Liveness:**
  - Each honest party obtains an output after some fixed time  $T'$
- ❖ **Consistency:**
  - Each honest party should have some common output  $v^*$
- ❖ **Validity:**
  - $v^* = v$  if all honest parties have the same input  $v$

*Distributed Consensus*  
*Blockchain technology is basically a form of distributed consensus*

A related problem is the Byzantine agreement problem; we use the short form BA. And here also the system model is same as for the previous problem. We have a synchronous system of  $n$  parties in the secure channel model and each party has a private input. So, I am denoting the input as  $v_1, v_2, \dots, v_i, \dots, v_n$  from some publicly known domain. So, in the simplest form the domain could be the set  $\{0, 1\}$ ; that means, the input of each parties just a bit, but it is a private bit; that means, when the protocol starts executing  $P_1$  will not know the input bit of the other parties and so on.

At most  $t$  parties among these  $n$  parties could be byzantine corrupted, but the exact identity of those  $t$  byzantine corruptions will not be known beforehand, and we want a protocol. Let us call that protocol as  $\pi_{BA}$  which should satisfy three properties. Again, the first property is the Termination or Liveness property, which demands that it should not happen that the parties keep on running the protocol forever. There should be some fixed known time, say  $T'$ , within which each honest party should obtain an output.

By the way, all these properties are with respect to honest parties. We do not care whether the corrupt parties keep on running the protocol forever; and we can never stop them from doing so, because we have absolutely no control over what the corrupt parties are going to do, what output they are going to consider whether they are going to consider any output at all.

But we will require that the honest parties which are not compromised which are not under the control of the adversary should have some well-defined behavior. So, one of the well-defined behaviors is that they should have an output after some predetermined time; that means, it should not happen that they also keep on running the protocol forever.

And this time  $T'$  it varies from one protocol to another protocol. So, we may as. So, later, we will see varieties of byzantine agreement protocols. So, one BA protocol may have a different  $T'$  compared to another BA protocol and so on. So, that is the termination or the liveness requirement.

A second requirement is the Consistency requirement, which demands that each honest party should have a common output after time  $T'$  call it  $v^*$ . And that is why the term agreement. We want to ensure that even if the parties start with different inputs they should come on a common count, they should come to a common conclusion, they should be on the same page. So, they should have a common output  $v^*$  after the time  $T'$ .

And again, we now have this interesting property the Validity property, which makes the problem very interesting. The validity requirement states that if the inputs of all honest parties were same at the beginning. See the input was  $v$  then all the honest parties should stick to that output, it should not change from  $v$  to any other value. Again, if we do not put this validity requirement, the BA problem is very trivial to solve everyone just outputs a default value.

If the domain is the set  $\{0, 1\}$  then a one-line code, a one-line BA protocol, could be “output 0”, everyone outputs 0 that is all, no interaction. And everyone will terminate this protocol and consistency will be satisfied, but it will not satisfy the validity requirement. Because if everyone by default outputs 0 and if all the honest parties had their input 1, then the validity condition is not satisfied. It is the validity requirement which makes the problem very interesting.

Now, this BA problem is often called as the Distributed Consensus problem in the community. Why is it called distributed consensus? Because we have  $n$  systems and we have because we have  $n$  parties and each party has a different state, different input, to begin with. So, you can imagine that the private inputs of the parties are nothing but their respective states.

So, these parties could be the database, they could be system components, they could be parts, they could be processes of operating system and so on. So, we have  $n$  entities each of them has its own private state. And we would require we would like to have a protocol executed among those  $n$  entities which ensures that they come to a consensus.

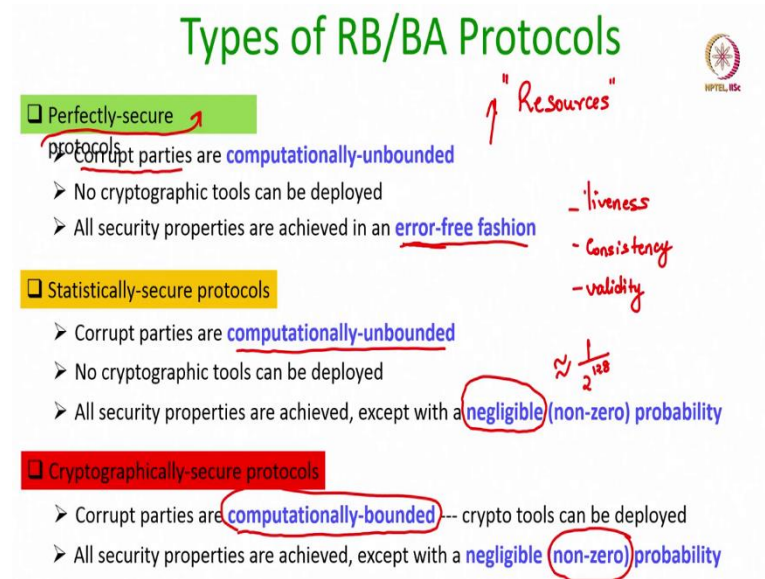
That means the output a common output after some fixed time. Even if they start potentially with different states different inputs, but if all the good components if all the good system components of all the honest parties start with a common input, they were having the same state then the state should not get changed. The output should remain the same as it was, if at all the good components all the honest components were having the same state.

So, you must have heard about this term blockchains. Blockchain technology is basically a form of distributed consensus because, on a very high level, what happens in the blockchain protocol is that you have several copies of blockchain available at different locations. We would require a protocol which allows the entities to interact among themselves.

And after every update in the blockchain if any update happens in one copy of the blockchain, through this interaction, through this protocol, through this consensus mechanism, that update is reflected across all the other copies of the blockchain. So, that is nothing but doing some form of distributed consensus only. So, that is why byzantine agreement is a very fundamental problem in distributed computing, where we would require  $n$  components to come to a common state come, to a common conclusion, by running a protocol among themselves.



(Refer Slide Time: 19:03)



So, there are various types of RB and BA protocols. So, the first category of protocols is called as a perfectly secure protocol. So, when we say perfectly secure RB or perfectly secure BA protocols, that means that the corrupt parties are computationally unbounded; that means, we make absolutely no assumption regarding the computing power of the  $t$  corrupt parties.

As a result, no cryptographic tools can be deployed; and all the three security properties namely the consistency, liveness and validity property are achieved in an error free fashion through perfectly secure protocols. A slightly weaker category of protocols is the statistically secure protocols, where adversary is still computationally unbounded; that means, no cryptographic tools are allowed.

But now, all the security properties need not be achieved in an error free fashion; that means, now you are allowed a very small negligible, but non-zero probability in the protocol output; that means, the three properties namely the liveness and the consistency and validity these properties should be achieved with high probability.

See if you are wondering what the negligible probability is, on a very high level it is such a small quantity that it can be ignored for all practical purposes say of order  $\frac{1}{2^{128}}$ . And the last category of protocols is the cryptographically secure protocols where you are allowed to use cryptographic tools, because we make the assumption here that adversary the set of  $t$  corrupt parties is under the control of a computationally bounded adversary.

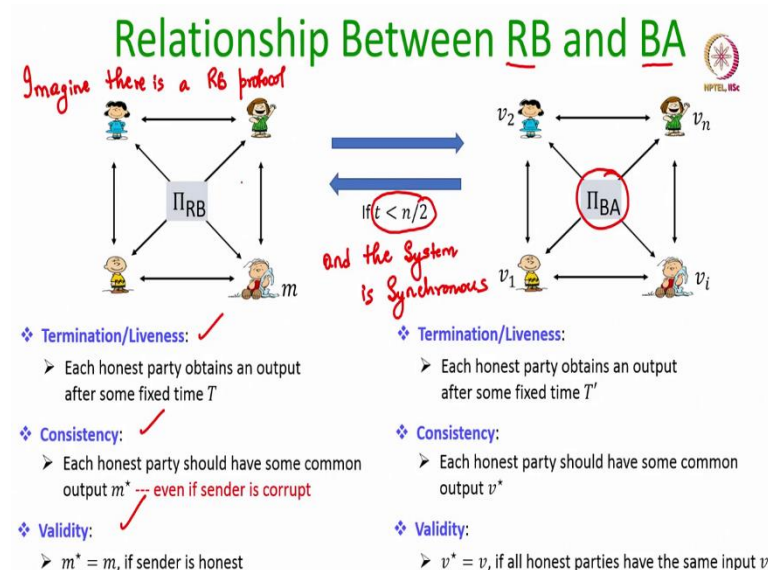
That means, adversary can now no longer perform an exponential number of computations. And here also all the security properties, the three security properties should hold with a high probability, but there is always a non-zero probability non-zero error probability which is allowed in the protocol outcome.

So, you might be wondering that why I should go for statistically secure protocol or cryptographically secure protocol because security wise, they are secure against a less powerful adversary and not only that, they give me security guarantees which are not 100 percent but with high probability.

So, of course, I should opt for perfectly secure protocols where the security guarantees are 100 percent. But, later, as the course proceeds, we will see that the resources required for perfectly secure protocols are typically more than statistically secure and cryptographically secure protocols. When I say resources, I mean the running time of the protocol and the number of messages which are exchanged and the number of corruptions which can be tolerated in the protocol.

So, later, as the course proceeds, we will see what resources are required by perfectly secure protocols. It is the tradeoff you must make if you want full security against the most powerful form of adversary. So, you have to deploy more resources. If your resources are very critical then, but you are fine to tolerate very small error probability, then go for statistically secure protocol or cryptographically secure protocols.

(Refer Slide Time: 23:11)

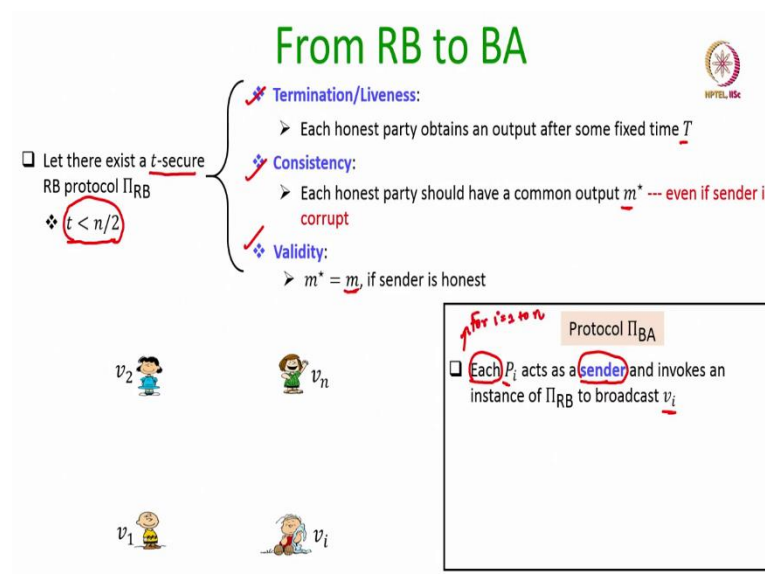


Now, let us see the relationship between the RB and the BA problems. From the problem descriptions they might sound very similar, but they are not. But there is a very nice relationship between the RB and the BA problem. So, imagine you are given an RB protocol. So, imagine there is an RB protocol, we do not know the details, but imagine there is a sequence of steps which satisfies these three requirements.

Now, given this we can design another protocol solving the byzantine agreement problem that is a derive that is a relationship in one direction. And we can show the relationship in another direction as well, namely if there is a byzantine agreement protocol then using that byzantine agreement protocol, we can solve the reliable broadcast problem as well.

This relationship in both the directions holds as long as the number of corruptions in the system is strictly less than  $\frac{n}{2}$  and the system is synchronous;; that means, given an RB protocol you can convert it into a BA protocol and vice versa.

(Refer Slide Time: 24:56)

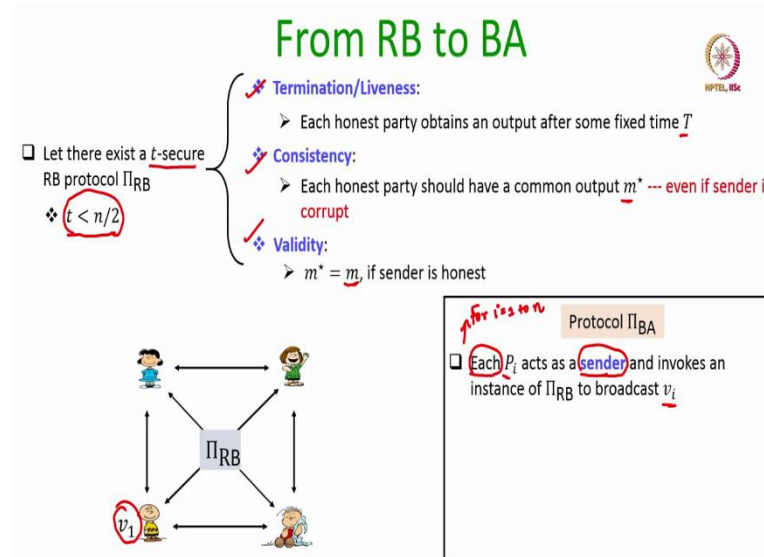


Let us see how we can obtain protocol for one task given a protocol for the other task. So, let us see the direction from RB to BA. So, imagine you have a  $t$ -secure RB protocol; when I say  $t$ -secure RB protocol I mean to say it satisfies all the three requirements of RB Reliable Broadcast, even if up to  $t$  parties in the system get corrupt during the protocol execution.

And imagine  $t < \frac{n}{2}$ ; that means, after time  $T$  everyone will have some output that output will be a common output even if the sender is corrupt. And that common output will be the sender's message if the sender would have been honest all these three properties are satisfied. Using this protocol  $\pi_{RB}$ , I can design the following BA protocol  $\pi_{BA}$ .

So, recall, in the BA problem every party has its own input we do not have any designated sender. We have every party with its own input whereas, in the RB problem only one of the parties has the input namely the sender. Now, in this BA protocol which I am designing my first step is the following: I ask each party  $P_i$  to act as a designated sender and invoke an instance of this protocol  $\pi_{RB}$  namely the reliable broadcast protocol to broadcast its input  $v_i$ .

So, this step will be done by every party  $P_i$ . (Refer Slide Time: 26:53)



So, this is like for  $i = 1, \dots, n$ ; that means,  $P_1$  will be acting as a sender where  $v_1$  is the input of the sender for the instance of the reliable broadcast protocol and everyone will be running an instance of this reliable broadcast protocol assuming  $P_1$  to be the designated sender where the sender's input is  $v_1$ .

(Refer Slide Time: 27:14)

## From RB to BA

Let there exist a  $t$ -secure RB protocol  $\Pi_{RB}$

$t < n/2$

**Termination/Liveness:**

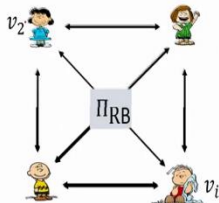
- Each honest party obtains an output after some fixed time  $T$

**Consistency:**

- Each honest party should have a common output  $m^*$  --- even if sender is corrupt

**Validity:**

- $m^* = m$ , if sender is honest



Protocol  $\Pi_{BA}$

Each  $P_i$  acts as a sender and invokes an instance of  $\Pi_{RB}$  to broadcast  $v_i$

In parallel there is another instance of RB protocol which will be executed where  $P_2$  will be serving as the designated sender with its input  $v_2$ . And, like that there will be in parallel  $i$ th invocation of the RB protocol getting executed in parallel where  $P_i$  will be the sender with its input  $v_i$ .

(Refer Slide Time: 27:37)

## From RB to BA

Let there exist a  $t$ -secure RB protocol  $\Pi_{RB}$

$t < n/2$

**Termination/Liveness:**

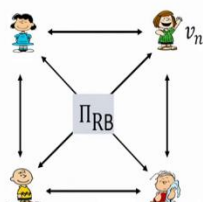
- Each honest party obtains an output after some fixed time  $T$

**Consistency:**

- Each honest party should have a common output  $m^*$  --- even if sender is corrupt

**Validity:**

- $m^* = m$ , if sender is honest



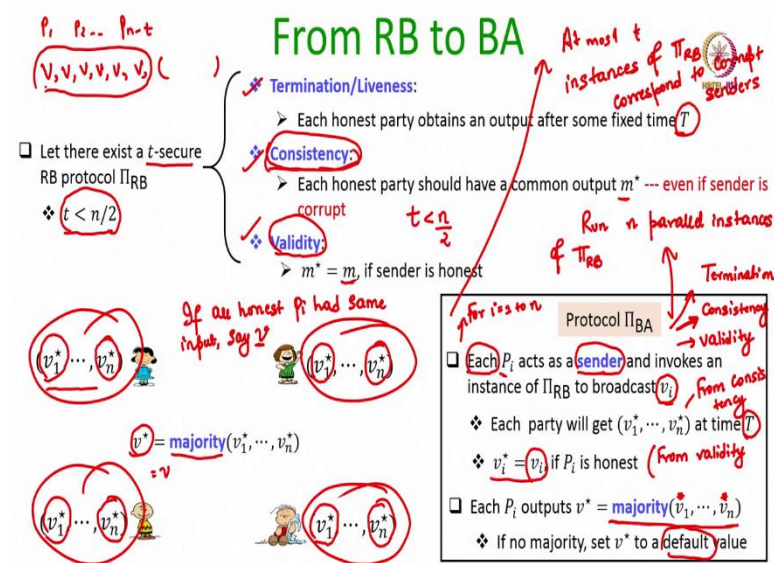
Protocol  $\Pi_{BA}$

Each  $P_i$  acts as a sender and invokes an instance of  $\Pi_{RB}$  to broadcast  $v_i$

Run  $n$  parallel instances of  $\Pi_{RB}$

And, in parallel there will be the  $n$ th instance of the RB protocol, where  $P_n$  will be serving as the designated sender. So, basically what we are doing here is run  $n$  parallel instances of  $\pi_{RB}$  where in the  $i$ th instance  $P_i$  is the sender with its input  $v_i$ .

(Refer Slide Time: 28:12)



Now, remember that among these  $n$  invocations of RB up to  $t$  invocations could be by the corrupt senders. So, at most  $t$  instances of  $\pi_{RB}$  correspond to corrupt senders. And the parties will not be knowing who the corrupt sender is, who the honest sender is, they are just participating in parallel invocations of  $\pi_{RB}$ . Of course, they will be knowing who the sender for one instance for one specific instance of RB is.

So, we can always assume that when multiple invocations of the same protocol are getting executed, to distinguish between the messages of one instance from another instance, we associate tag the identifier of the instance and so on. So, those details we can always assume. Now, what is the liveness guarantee of  $\pi_{RB}$ ? The liveness guarantee of  $\pi_{RB}$  is that after time  $T$  the instance will be over, and each honest party will obtain an output.

So, that means, the first instance of  $\pi_{RB}$  will be over at time  $T$ , the second instance of  $\pi_{RB}$  will be over at time  $T$  and like that the  $n$ th instance of  $\pi_{RB}$  will also be over at time  $T$ . So, it is not the case that all the  $n$  instances keep on running forever. Moreover, we know that from the consistency property of  $\pi_{RB}$ , all the honest parties will have the same output from the first instance of RB.

So, the instance of RB where  $P_1$  as was acting as the sender will produce a common output for all the  $n$  parties that is coming from the consistency property of RB. Like that, if I consider the  $n$ th instance of  $\pi_{RB}$  invoked by the sender  $P_n$ , that will produce a common

output for all the honest parties; again, coming from the consistency property. So, that means, if I consider the output vector, why output vector?

Because there are  $n$  instances of  $\pi_{RB}$ , so there are  $n$  outputs which each party is going to obtain. I can visualize the  $n$  outputs which each party is obtaining as a vector. So, through the consistency property of  $\pi_{RB}$ , it is guaranteed that all the output vectors will be same.

So, it would not be the case that  $P_1$  outputs one output vector and  $P_2$  outputs another output vector. No, that is not going to happen because the consistency requirement the consistency property of  $\pi_{RB}$  ensures that even if the sender of that  $\pi_{RB}$  instance is corrupt, every honest party will have a common output in that instance.

Moreover, from the validity property of  $\pi_{RB}$ , if I consider the  $i$ th instance of  $\pi_{RB}$  then the output which every party has obtained in that instance will be the input  $v_i$  of  $P_i$ ; because in the  $i$ th input  $P_i$  would have invoked the instance of  $\pi_{RB}$  to broadcast its input  $v_i$  for the BA.

So, that means that the two properties that we have now guaranteed are the following. The output vectors that each party is going to have will be common and, among these output vectors, if I focus on the component corresponding to the honest parties' input, that will be the input that honest party has for the BA problem.

Now, we must provide an output decision rule for the  $\pi_{BA}$  protocol because the BA protocol requires a common output from all the honest parties. So, the output that each party produces in this  $\pi_{BA}$  protocol is the following: they simply output the majority value of their vector. This should be  $v_1^*, \dots, v_n^*$ .

So, they see if there is a value which occurs majority number of the times, they output that value. If there is no majority in this vector, then this set  $v^*$  to some default value in the domain. Now, my claim is that this BA protocol that we have designed satisfies the termination property. Why does it satisfy the termination property? Because each honest party will terminate the  $n$  instances of  $\pi_{RB}$  that will take time  $T$ , and after that they just have to find the majority of the output vector.

So, this  $\pi_{BA}$  protocol will also get over after time  $T$ . So, since termination is satisfied, this  $\pi_{BA}$  protocol will satisfy the consistency requirement. Why will it satisfy the consistency

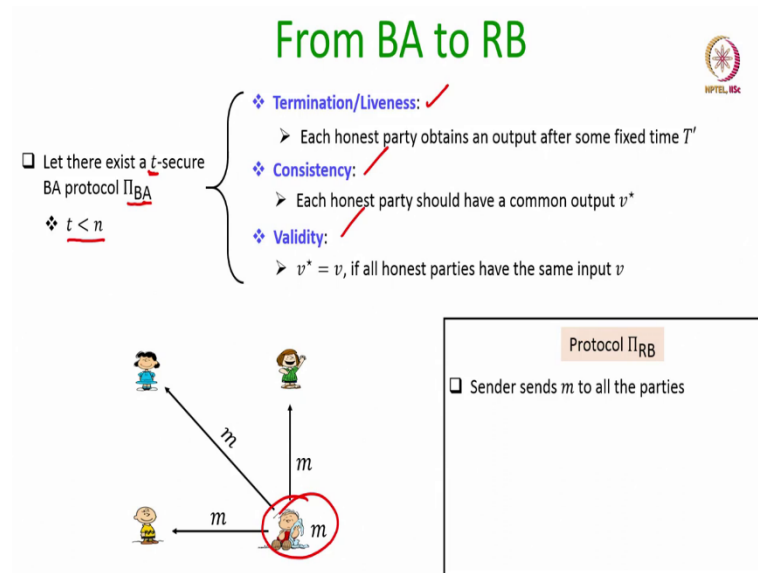


requirement? Because the output vector of all the honest parties will be common, we have already argued that; and what is the output of the BA protocol? The majority of that common output vector.

So, everyone will be applying the same majority rule to their respective output vector which is common across all the honest parties which ensures the consistency property of the BA. And this BA protocol will satisfy the validity property as well. Why? Because if all honest parties  $P_i$  had the same input, say  $v$ , then through the majority rule  $v^*$  will be nothing but  $v$ .

This is because we have argued that the validity property of  $\pi_{RB}$  ensures that the RB instances for the honest sender parties will produce the output  $v$ . So, for instance if  $P_1, P_2$  and  $P_{n-t}$  are the honest parties then the first  $n - t$  components in the output vectors of each party will be  $v$ . The remaining  $n - t$  could be any value, but since we are assuming  $t < \frac{n}{2}$ , that means majority of the values in the output vectors of all the parties will be  $v$  and that is why  $v^*$  will be  $v$  and that is why the validity property of the BA protocol will be retained.

(Refer Slide Time: 36:30)



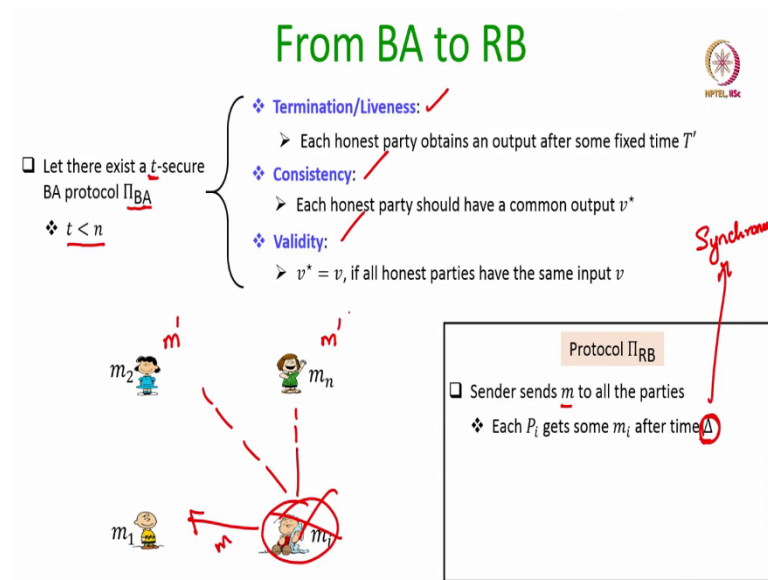
Now, let us see the relationship in the other direction. Assume you are given a Byzantine agreement protocol which is  $t$ -secure and imagine  $t < n$ . We do not even need  $t < n/2$ ;



that means, the  $BA$  protocol satisfies the termination property, it gives you the consistency property, it gives you the validity property.

Using that, we want to design a reliable broadcast protocol where some designated party is the sender. Now, in the reliable broadcast protocol the first step is to let the sender send its message to all the parties. So, remember in the reliable broadcast problem, no other party has any input except the sender. So, the first step is to let the sender send its message to everyone.

(Refer Slide Time: 37:25)



If the sender is corrupt, then it can send different versions of its message to different honest parties. But if it is honest, it will send the same message to everyone. Since the system is synchronous, it will be guaranteed that after time  $\Delta$  which will be publicly known, that is the channel delay, every party will get some message from the sender. Of course, if sender gets crashed; so, remember sender could be potentially corrupt.

And it could be corrupted in a byzantine fashion and byzantine corruption subsumes crash failures. So, it could be the case that sender just sent  $m$  to one of its neighbors and then it simply gets crashed. So, the convention that we follow while designing or while writing synchronous protocols is the following.

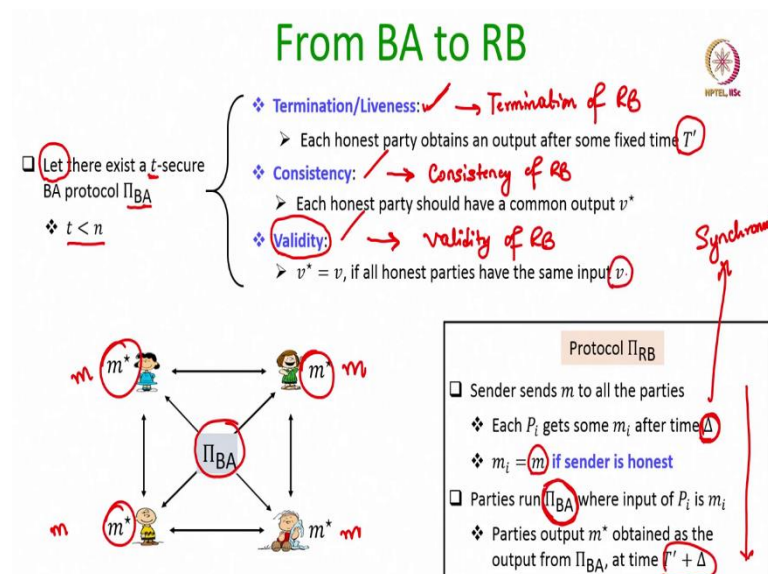
If a party is expecting a message from a sender party and if within time  $\Delta$ , within the channel delay, that expected message does not turn out, then the receiving party will

substitute it with some default value and proceed to the next steps; it does not wait indefinitely.

So, considering the scenario, if the sender party sends its message to one party and then suddenly crashes and it does not send anything to the other parties, then the other parties will assume as if the sender wanted to send some default value  $m'$ . The default value will be publicly known, it will be specified as part of the protocol description. So, we do not write separate codes. So, separate else if then statements for handling the cases when no message or no expected message turned out within the expected timeout are not written.

Whenever an expected message does not turn out within the expected timeout the receiving party substitutes it with some default value thinking this is the message which the sender would have tried to send me and goes to the next step.

(Refer Slide Time: 39:34)



So, the first step of this RB protocol was that the sender sends its message to everyone. And, as I said, if it is corrupt, it may send different versions of its message to different honest parties. But if it is honest if the sender is honest, it will send an identical copy of its message to everyone. Now, after the time  $\Delta$  every party has some message on the behalf of the sender. If the sender is honest, all of them have the same message; if the sender is corrupt, they might have different versions of sender's message.

But as part of the RB protocol, they must come to a common conclusion on what the sender's message is. So, it is very simple. What they can do is they can simply now run a BA protocol and we are assuming that there exists a BA protocol satisfying the termination consistency and validity properties of BA. So, that BA protocol every party runs.

And what is the input of the  $P_i$  in that instance of the BA protocol? It is the senders' versions of the message which  $i$ th party has received. So,  $P_1$  participates with input  $m_1$ ,  $P_2$  participates with input  $m_2$ ,  $n$ th party participates with input  $m_n$  and so on. Now, the termination guarantee of BA is that it produces some output after time  $T'$ .

So, whatever output the BA protocol produces for the honest parties, that is considered as the output for the RB protocol by the parties. So, it is easy to see that the termination property of BA implies termination property of RB, because this protocol will get over by time  $T' + \Delta$  for everyone. The consistency property of BA guarantees consistency property of RB.

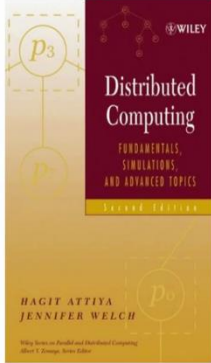
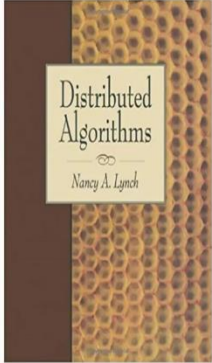
Why? Even if sender is corrupt; suppose, sender is corrupt and sends different versions of its message to different honest parties. The instance of  $\pi_{BA}$  which the parties are running will ensure that they come to a common conclusion regarding the sender's message which sender would have sent to different honest parties.

And the validity of BA guarantees the validity of RB. Why? Because if sender is honest, then it will send the same message  $m$  to every honest party. So, everyone will have received  $m$  from an honest sender and everyone would have participated with input  $m$  in the instance of BA and the validity of BA guarantees that if all the honest parties have the same input then they stick to that output at the end of the protocol which will be implying the validity of RB.

(Refer Slide Time: 42:53)

## References

□ Matthias Fitzi: Generalized communication and security models in Byzantine agreement. ETH Zurich, Zürich, Switzerland, Hartung-Gorre 2003, ISBN 978-3-89649-853-3, pp. 1-189



So, that is the relationship between the RB problem and the BA problem. If one problem can be solved, the other can be solved and vice versa. So, the relationship between RB and BA, which I have discussed in today's lecture, you can find it in either these two textbooks or even in this Ph.D. thesis. The problem description of RB and BA you can also find them in one of these two textbooks.

Thank you.