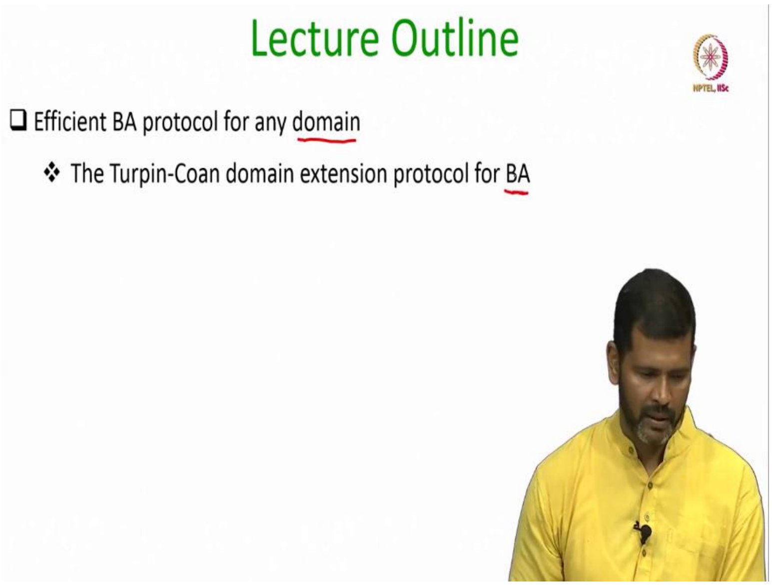
Secure Computation: Part II
Prof. Ashish Choudhury
Department of Computer Science and Engineering
Indian Institute of Science, Bengaluru

Lecture - 09
Domain Extension for Perfectly - Secure Byzantine Agreement

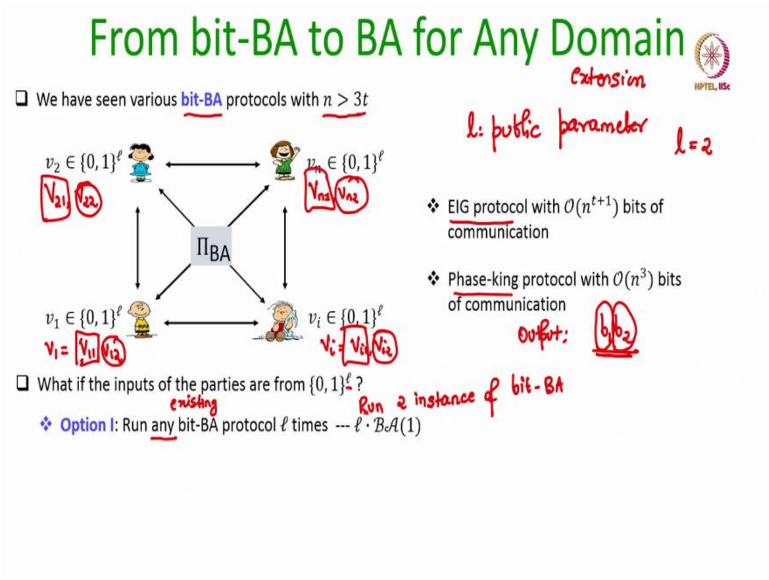
Hello everyone, welcome to this lecture.

(Refer Slide Time: 00:25)



So, in this lecture, we will see a perfectly secure byzantine agreement protocol for any domain. Namely, we will see the well-known domain extension protocol for BA by Turpin and Coan. So, even though I will be explaining this domain extension protocol in the context of perfectly secure BA protocol, we will see some variations, where this domain extension is applicable in other setting as well.

(Refer Slide Time: 00:55)



So, what exactly do we mean by domain extension for byzantine agreement and broadcast? So, we have seen various byzantine agreement protocols till now; namely, we have seen three protocols and if we focus on the protocols with $n > 3t$, then we have actually seen 2 protocols; the EIG protocol and the Phase-king protocol, where we assume that the input inputs of all the parties is a bit and they want to reach agreement on a bit. That is a very basic setting which we have considered.

But what if in some application the inputs of the parties is not a single bit, rather each party has a large input say an input of size 1000 bits or an input of side size 1, GB and so on. So, in general, what if the input is of the size is ℓ bits, where ℓ is some public parameter. So, you might be wondering where exactly we encounter such application.

Well, plenty of applications; even if we take secure multi party computation protocols which we will be seeing later, there we have scenarios, where parties need to reach agreement on very large messages, large inputs, where the inputs are no longer a single bit or if we consider block chain applications, where we have multiple copies of the same database replicated across n locations.

And after every few cycles say the state of the individual copies get updated and then, we run a consensus protocol to reach agreement on an up-to-date version of the database. There also the database is not just the single bit, it is an enormously large database right.

So, there are plenty of settings, where we encounter this scenario, where the inputs of the parties are no longer a single bit, but rather from a large domain; namely, the set $\{0,1\}^{\ell}$. Namely, the set of all binary strings of length ℓ bits. So, if we want to achieve, if we want to do byzantine agreement in this setting; one option will be that we use any existing bit BA protocol and run it for ℓ times.

So, for instance, what I am saying here is that say for instance ℓ is equal to 2; that means, the input of each party is a binary string consisting of 2 bits. So, v_{11}, v_{12} like that the ith party's input is a binary string consisting of the bits v_{i1}, v_{i2}; the input of the nth party is v_{n1}, v_{n2} and like that. The second party's input consists of 2 bits v_{21}, v_{22}. So, option 1 basically says that you run 2 instances of any of the existing BA protocols, where the inputs are bit right.

So, I call those existing BA protocols as bit BA protocol. So, EIG protocol is one potential bit BA protocol. The Phase-king to BA protocol is a potential bit BA protocol. So, you run 2 instances of either the EIG protocol or the phase-king protocol. In the first instance, the inputs of the parties will be $v_{11}, v_{21}, v_{i1}, v_{n1}$, they run a protocol and come to a decision.

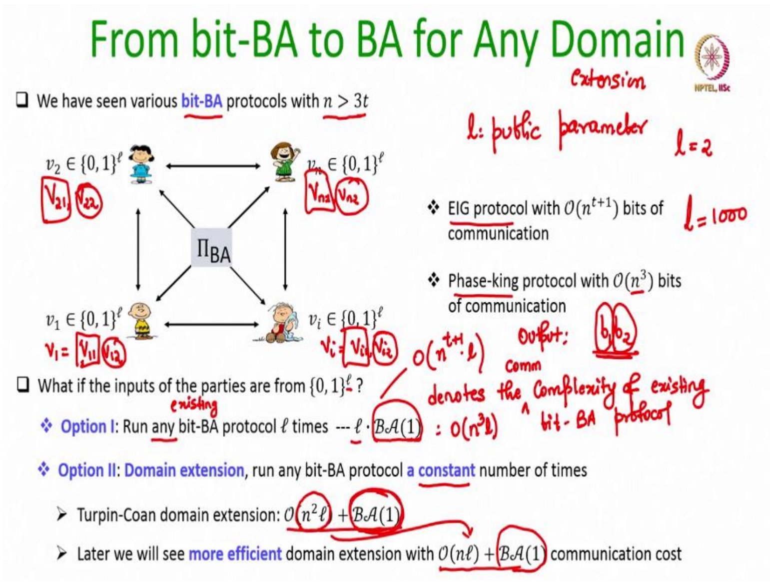
And independently, they will be running a second instance of the protocol, where the inputs will be v_{12}, v_{22}, v_{i2} and v_{n2}. So, suppose b_1 is the decision of the first instance of the protocol and b_2 is the decision of the second instance of the protocol, the overall output of the parties will be now b_1 followed by b_2 and now, you can verify that the validity termination validity liveness and consistency properties are all satisfied.

So, liveness is trivial because we are basically running ℓ instances of the existing protocol. If the existing protocol has liveness, then the new protocol, where we are basically running ℓ parallel copies instances of existing protocol will also terminate. If the existing bit BA protocol has validity, then this way of running ℓ instances also will satisfy the validity property.

So, what does the validity property now mean? That if all the honest parties have the same binary string of length 2; then $b_1 b_2$ will be that binary string. This is because individually bit wise, the validity condition will be satisfied and that will ensure that the output $b_1 b_2$ will be the common binary string which all the honest parties have at the beginning of the protocol and consistency is again boiling down to the consistency of the existing bit BA protocol.

Namely, every honest party will output b_1 as the first bit and b_2 as the second bit and overall output will be $b_1 b_2$ for everyone. So, that is one way of doing domain extension. But what will be the complexity of this domain extension protocol? The complexity will be ℓ times the complexity of the existing BA protocol.

(Refer Slide Time: 07:17)



So, this notation BA within parenthesis denotes the complexity and when I say complexity, I mean to say the communication complexity of existing bit BA protocol which you deploy here. So, for instance, if we are deploying the phase king protocol as the bit BA protocol, then this option will result in a communication complexity of $\mathcal{O}(n^3\ell)$; whereas, if we are using the EIG protocol, then this option number 1 will result in a communication complexity of $\mathcal{O}(n^{t+1}\ell)$.

What domain extension does basically is that it allows you to get a byzantine agreement protocol for a larger domain; but without running ℓ instances of bit BA protocol.

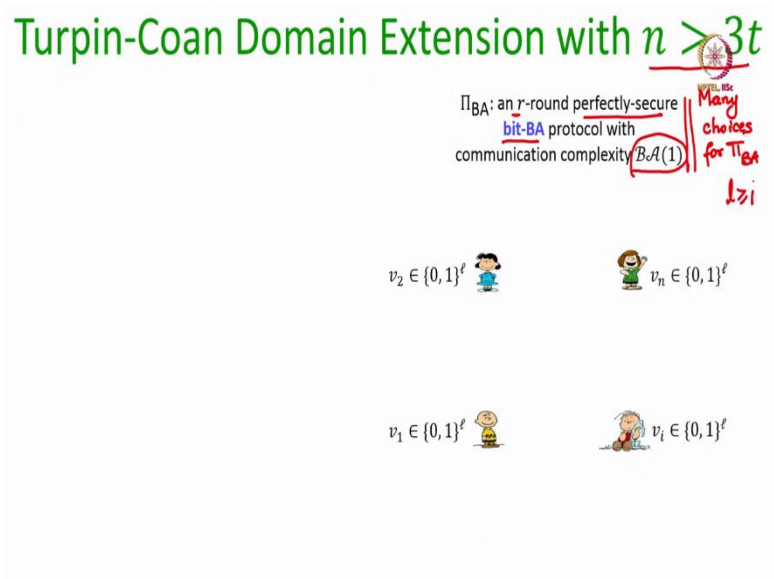
But rather we will run only a constant number of instances of existing bit BA protocol ok that ensures that we have tremendous saving in the communication complexity. So, in today's lecture, we will discuss domain extension due to Turpin and Coan and this gives you a complexity of $\mathcal{O}(n^2\ell)$ plus whatever complexity is required by existing bit BA protocol. So, you see that we are now not running ℓ instances of the bit BA protocol.

So, you might be wondering how it is a saving. So, if say for instance, I take ℓ to be 1000, then option 1 means that I have running 1000 instances of the existing bit BA protocol which is an overkill; whereas, through Turpin-Coan domain extension, we can still get a byzantine agreement protocol, where the number of instances of the byzantine agreement protocol the bit BA protocol that we need to execute is only one, which is a tremendous amount of saving.

Later, after developing sufficiently advanced tools in the course, we will see much more efficient domain extension protocol, where the overall cost for the byzantine agreement protocol for the bigger domain will be only $n \cdot \ell$ plus some constant number of invocations of the existing bit BA protocol.

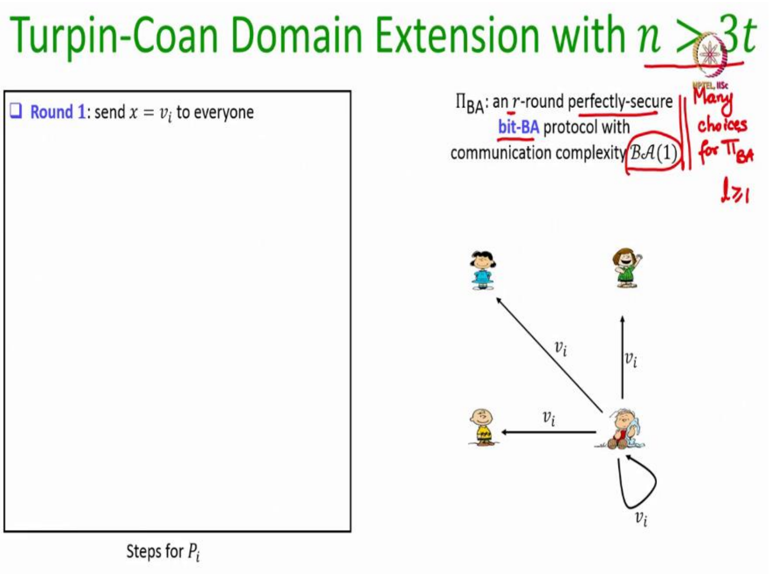
So, even the communication complexity which depends upon ℓ in the Turpin-Coan domain extension that gets improved in this more efficient domain extension. But to understand this more efficient domain extension, we need to develop some more advanced tools which we will develop as the course proceeds.

(Refer Slide Time: 10:24)



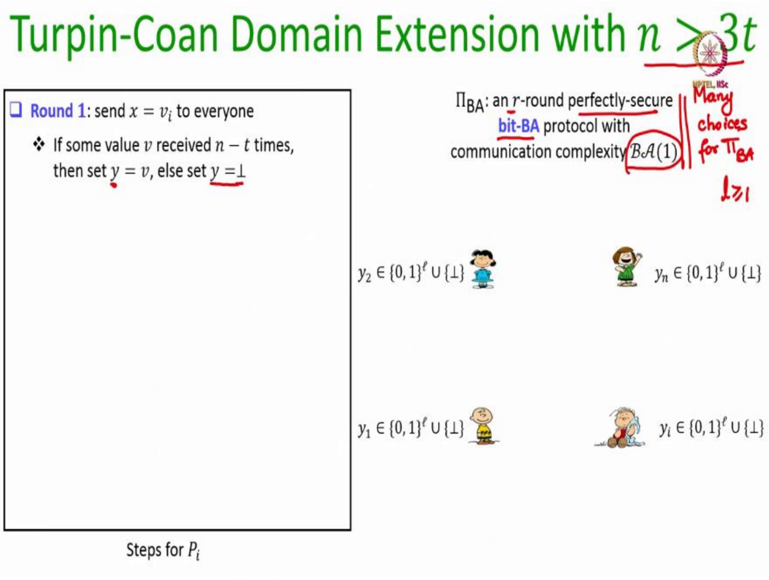
So, here is the Turpin-Coan domain extension with $n > 3t$ and here we assume that we have an already existing bit BA protocol which takes r number of rounds which is perfectly secure and whose communication complexity is denoted by this expression. So, you have many choices for this π_{BA}. Remember, we have many choices here for π_{BA}; you can either use the EIG protocol or you can use the phase king 2 protocol. And now, we want to design a BA protocol, where the inputs of the parties are binary strings of length ℓ bits, where $\ell \geq 1$.

(Refer Slide Time: 11:15)



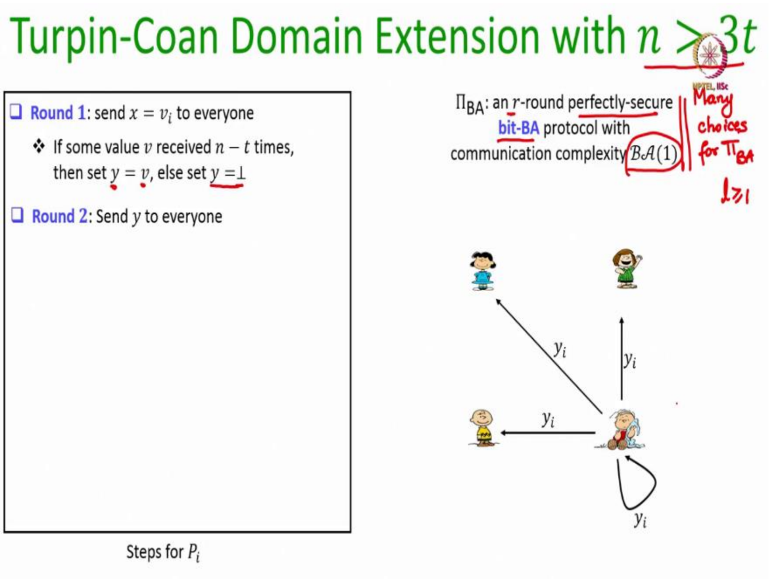
And our goal is to get a byzantine agreement protocol, where this existing bit BA protocol is invoked only a constant number of times. In fact, in the Turpin-Coan domain extension, it is invoked only once. So, here is the Turpin-Coan's domain extension protocol, during the first round every party sends its ℓ bit input to everyone. That is the first step.

(Refer Slide Time: 11:46)



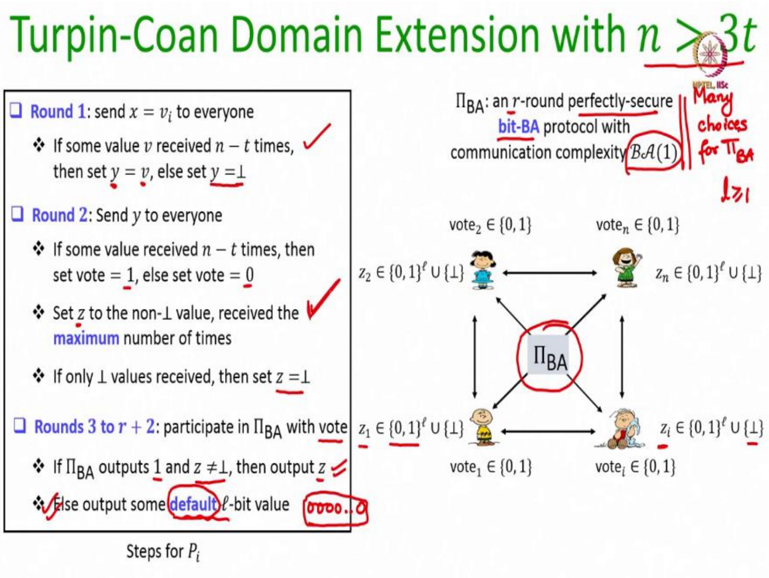
Now, every party checks the following. If it has received a string v $n - t$ number of times; that means, that string v has been received from $n - t$ different parties; then assign the string v to a variable y, otherwise set y to a null value ok. Because it is not necessary that $n - t$ parties send a value v. It depends upon what exactly were the initial inputs of the parties. But if at all a party sees that a value v has been received from $n - t$ parties, set y to that value that string; otherwise set y to ⊥.

(Refer Slide Time: 12:33)



Now, during 2nd round, every party sends their version of the variable y to everyone. So, the party P_i, its version of the variable y is y sub i; it will send it to everyone. Either it will be a string of length ℓ binary string of length ℓ or the value ⊥. Of course, if P_i is corrupt, then it may send arbitrary values as y to different honest parties. So, for instance, it can send ⊥ to one set of honest parties, it can set value v 1 to one set of parties, value v 2 to another set of parties and so on. But if P_i is honest, it will stick to its version of y while sending it to different parties.

(Refer Slide Time: 13:20)



Now, again, depending upon how many copies of a particular value y P_i receives, it sets its vote variable. So, if P_i receives specific y value $n - t$ number of times; that means, from at least $n - t$ different parties, then it sets its vote variable to 1; otherwise, it sets its vote variable to 0. Again, different parties may end up send setting different vote variables; that means, they may assign different values to their respective vote variables depending upon whether they have received any specific value y $n - t$ number of times or not.

Also, they set each party P_i sets z to be the non-⊥ value which is received the maximum number of times during round 2. So, remember, when parties are exchanging y, a candidate y could be ⊥ as well. So, some parties might be sending y as ⊥, some parties might be sending y as an ℓ bit string. So, z is set to be the non-⊥ value if at all any party has sent any non-⊥ value, which has been received maximum number of times by P_i.

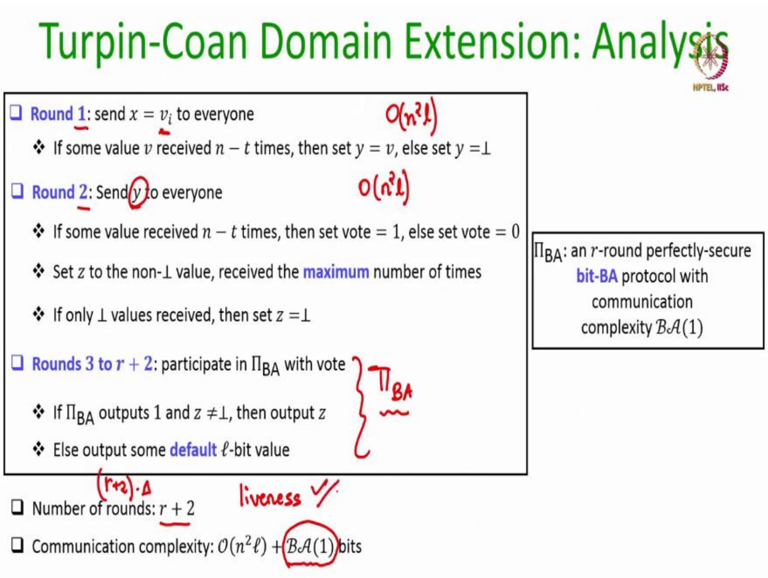
So, the version of z variable for P_1 is denoted by z_1; the version of the z variable for the i th party is denoted by z_i and so on and it is easy to see that each z variable will be either a string of length ℓ bits, if at all there is a maximum; whereas, if all the parties have communicated $\perp$ as the y value to P_i, then P_i's z_i variable will be $\perp$ ok.

Now, for the next r rounds the parties are going to run an instance of existing BA protocol and what will be their inputs? So, remember the existing BA protocol demands that the inputs of the parties are bit not ℓ bit strings. So, they run existing BA protocol, where their inputs are their respective vote variable.

So, that means, during the first round and the during the second round, parties just exchange ℓ bit strings with some conditions and based on that, they set their vote variable and then, for the next r rounds they run an existing BA protocol to conclude. If the output of the BA is 1 and if the z variable for P_i is not $\perp$; that means, it has indeed received some non-$\perp$ values from some parties and found the maximum.

Then, it sets that value as its final output for the BA protocol; otherwise means either the BA gives the output 0 or the BA gives the output 1; but z was set to $\perp$ by P_i, then P_i outputs a default ℓ bit string as the output value and default ℓ bit string means you can imagine that a string of length ℓ, where all the bits are 0 and this will be publicly known. That means, if P_i finds this condition to be true, then it will simply output a string consisting of ℓ zeros; otherwise, it will output the max value which had which it has set during this step.

(Refer Slide Time: 17:18)

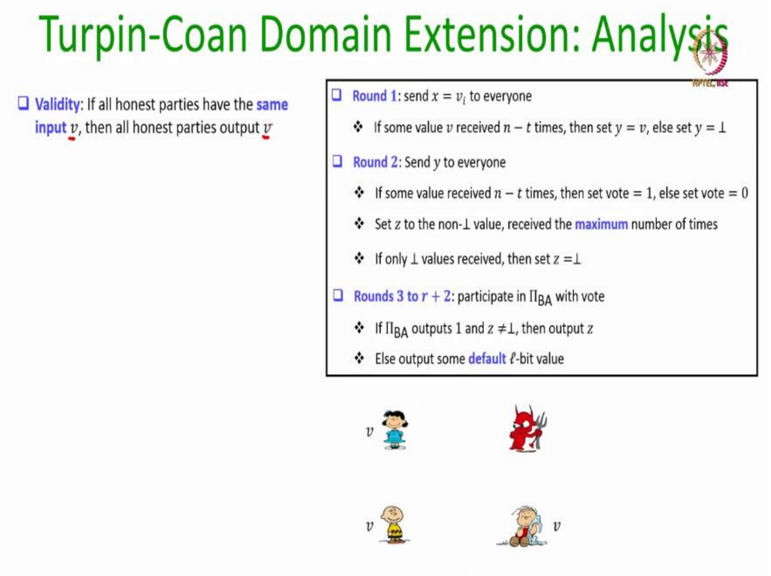


So, now, let us see whether this protocol satisfies the requirement of BA. So, first thing that liveness is guaranteed. The number of rounds required in the protocol will be $r+2$ rounds. Why $r+2$ rounds? Because we have round number 1, where parties initially exchange their values and then, they exchange the y values and then, the existing BA protocol which requires r number of rounds.

So, liveness is guaranteed because after the time $(r+2)\cdot\Delta$, every honest party will obtain an output and what is the communication complexity? So, the communication complexity of the existing BA protocol is denoted by this expression and during round 1, every party needs to send its initial input which is a string of length ℓ bits to everyone else.

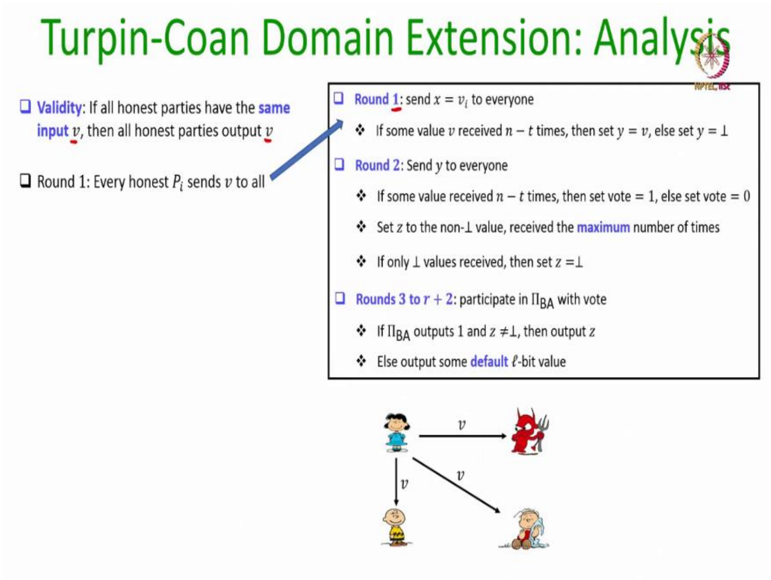
So, that will require a communication of $\mathcal{O}(n^2\ell)$ and again, during round 2, every party sends its version of the y variable to everyone else which also requires a communication of $n^2\ell$ bits. So, liveness is trivial to argue here.

(Refer Slide Time: 18:49)



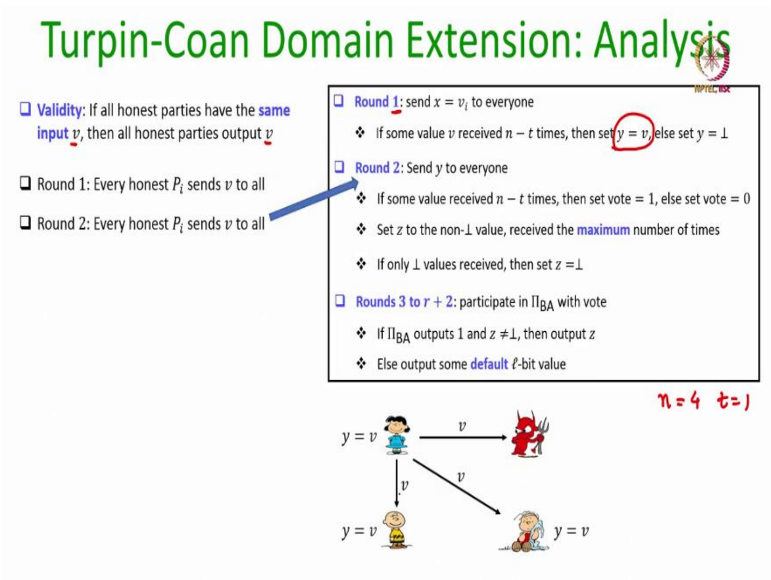
Let us argue the validity condition. So, we want to show here that if all the honest parties have the same ℓ bit input say v, then they stick to that input as the final output. That means the output remains the same as the string v itself.

(Refer Slide Time: 19:14)



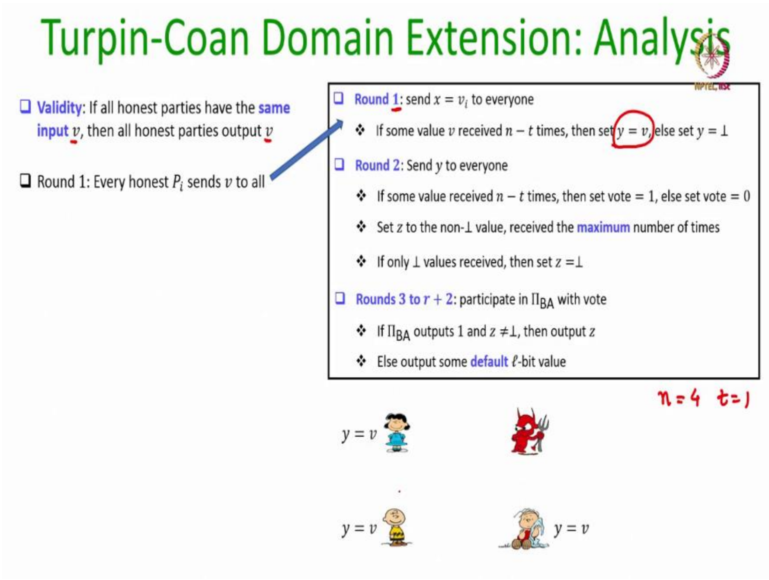
So, it is very easy to prove this. So, if all the honest parties have the same ℓ bit length string v, then every honest party will be sending that string v to everyone else during the round 1; corrupt parties may send different versions of ℓ bit strings. We do not care what they send.

(Refer Slide Time: 19:37)



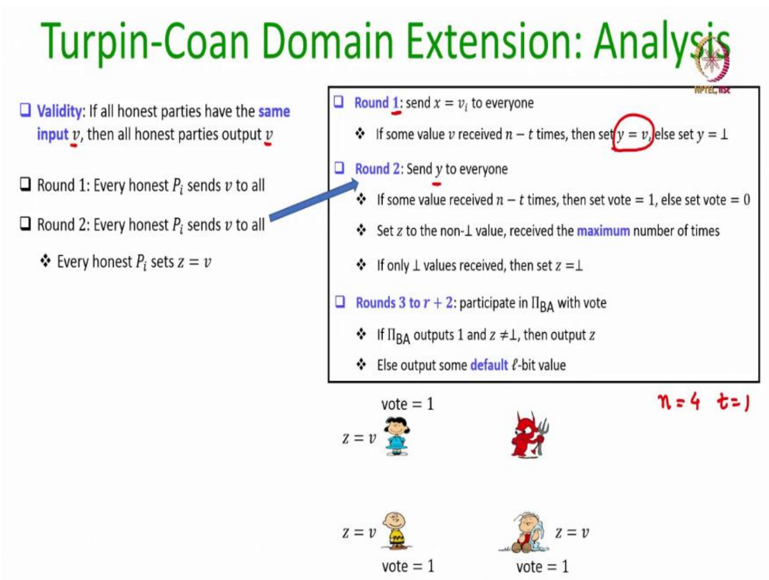
That means, at the end of round 1, every party would have set their y variable to the string b. So, again for demonstration, I am taking here $n = 4$ and $t = 1$.

(Refer Slide Time: 19:52)



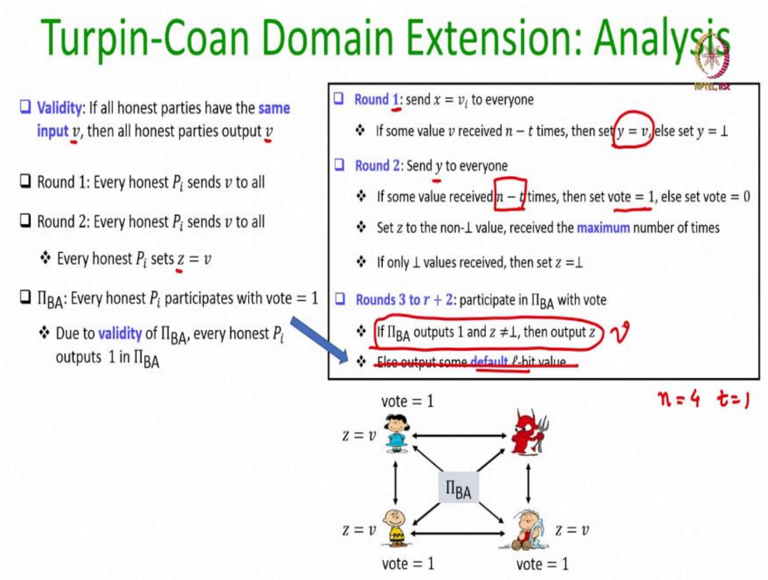
So, everyone would have sent v to everyone except the corrupt party; the corrupt party can send any other string and due to that the y variable for every party will be set to v. As a result of that, when everyone sends their respective version of y to other parties they will see that there are $n - t$ copies of the string v which are received.

(Refer Slide Time: 20:23)



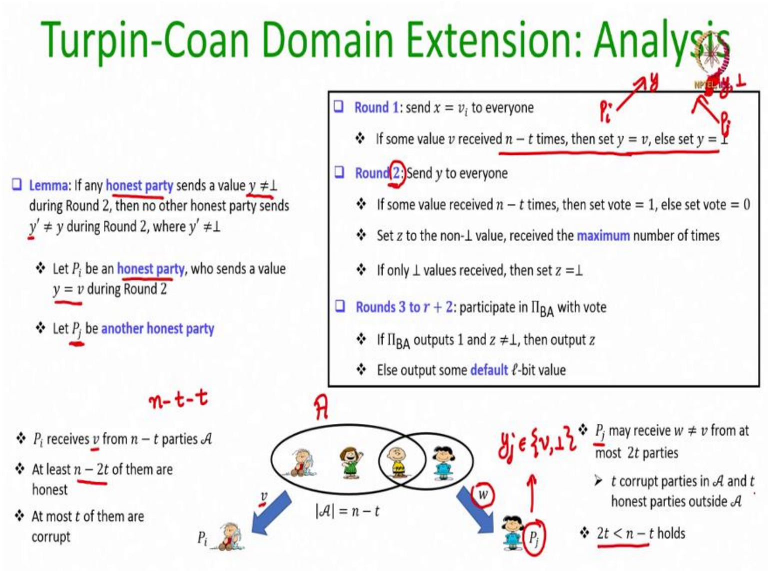
And as a result of that, every honest party will set their variable z to that string v.

(Refer Slide Time: 20:32)



And on top of that every honest party will also set their vote variable to 1 because they have received a copy of y $n-t$ number of times; that means, every honest party will participate in the instance of the vote protocol with input 1 and due to the validity of the existing BA protocol, it is guaranteed that every honest party will output 1. During the instance of the ⊥ protocol; that means, this condition will not be satisfied. This condition will be satisfied for every honest party and hence, they will output the value v; the string v, that means, the validity is satisfied.

(Refer Slide Time: 21:11)



Now, we prove the consistency property and to prove the consistency property, we will prove first a helping lemma which claims that if any honest party sends a value y which is not $\perp$ during the round 2, then no other honest party sends a different value for y. Namely, no other honest party sends y' during round 2, where y' is different from y. That is the claim.

So, that means, if at all during round 2 any other honest party sends y, they must send y; it will be either $\perp$ or it will be the same y. So, it cannot be the case that P_i has sent y during round 2 and P_j sends y', where y' and y are different.

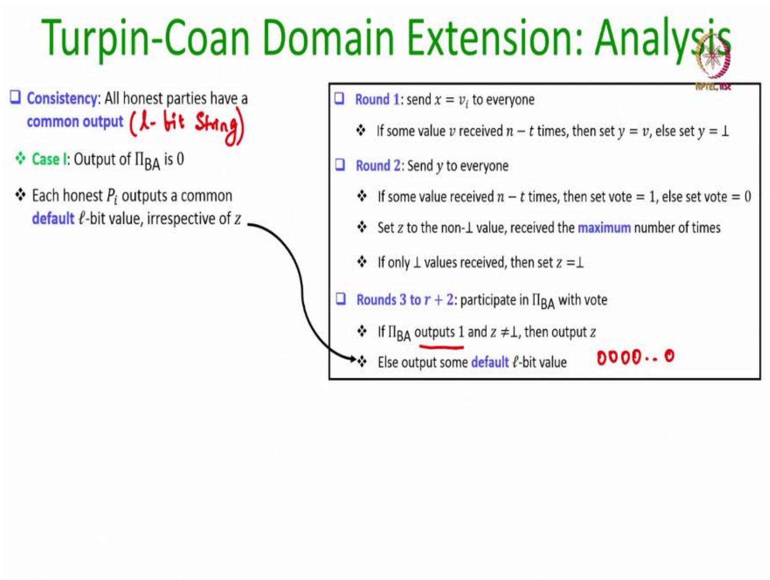
If at all P_j sends here something, it will be either the same y or the value $\perp$. So, let us prove this and the proof for this will be very similar to a proof of a similar lemma that we have used during the phase king 2 protocol. So, let P_i be an honest party, who sends a value y which is an ℓ bit string v during round 2 and consider P_j is another honest party and we want to prove that P_j does not send any other ℓ bit string as its y variable.

If it sends any other if at all it sends any ℓ bit string, it will be v or it could be $\perp$; it cannot be any other y'. So, since P_i has set the string a variable y to the string v; that means, it has received this ℓ bit string v from at least $n - t$ parties call that set as $\mathcal{A}$ and among those $n - t$ parties at least $n - 2t$ will be honest because there can be up to t corrupt parties in the set $\mathcal{A}$.

Now, how many strings of length ℓ, how many copies of a string w where w is different from v can be received by this honest party P_j? Well, it could be possible that there are t corrupt parties in the set $\mathcal{A}$, which might send the string w as their initial value to P_j, where w is different from v and it could be also possible that there are up to t honest parties outside the set $\mathcal{A}$, whose initial values are different from the string b, their initial values are w.

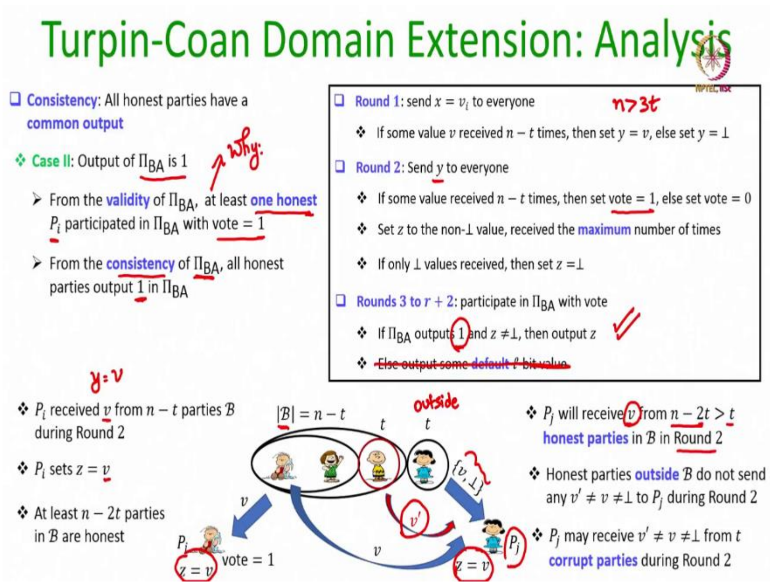
So, that means, all in all the honest party P_j would have received at most $2t$ copies of the string w, where the string w is different from v and $2t$ is strictly less than $n - t$; that means, that means, if at all P_j has set its y variable to an ℓ bit string, it will be the string v; it cannot be any other string w. So, either P_j sets its y_j variable v or the $\perp$; it cannot be anything else. So, that proves this lemma.

(Refer Slide Time: 25:18)



Now, using this lemma, we will prove the consistency property that all honest parties output a common ℓ bit string in this domain extension protocol and again, let us see how the output decision is taken. If the output of the BA protocol is 0, then the consistency is trivial. If the output of BA is 0, then every honest party outputs the default string all zeros; they do not take any other string as the output.

(Refer Slide Time: 25:57)



So, consistency is trivial. Consider the case when the output of BA is not 0; that means, the output of BA is 1. If the output of BA is 1; that means, at least one honest party, we do not know which exact honest party, is there, say P_i, who parties who must have participated in the instance of the bit BA protocol with its vote input being 1. Why so?

This is because if all the honest parties would have participated in the existing bit BA protocol with vote being 0, then from the validity of π_{BA} the output of the π_{BA} would have been 0. But we are considering the case when the output of the π_{BA} is 1 that is possible only if there is at least one input from the honest party, at least one honest party who is participating in this existing instance of the bit BA protocol with vote being 1.

Now, from the consistency of the bit BA protocol, all honest parties also will obtain the output 1 in the BA protocol. So, this shows at least that the else statement is not getting executed by everyone. If at all everyone is outputting something, it is because the output of the BA is 1. Now, we are assuming that there is at least one honest party P_i who has set its vote variable to 1. Let us see why it has set its vote variable to 1 during the protocol execution.

It must have set its vote variable to 1 because it has received some y value $n - t$ number of times let that y value be v; that means, it has received $n - t$ copies of v during the round 2 from a set of parties in $\mathcal{B}$ and it would have set its z variable to that string v. That is why P_i has output. The string v during the domain extension protocol, we want to argue that every other honest party P_j different from P_i also will output z equal to v.

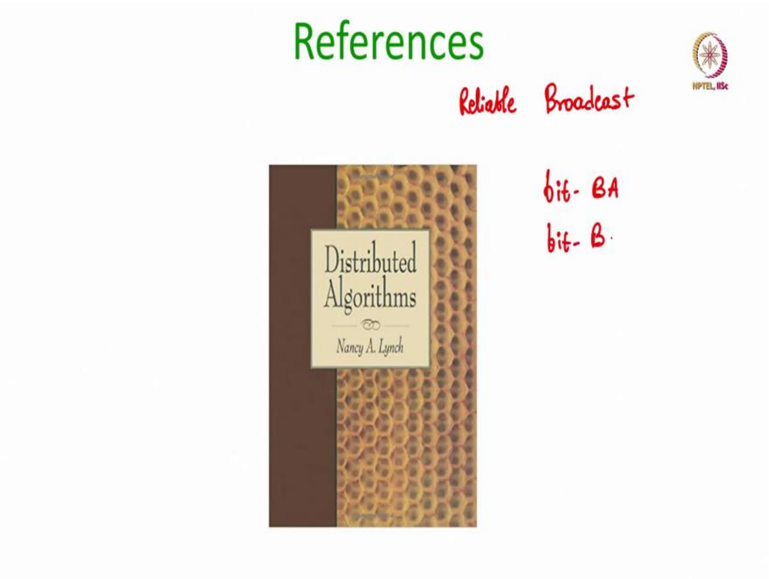
Let us see why. So, among these parties in $\mathcal{B}$ at least $n - 2t$ are guaranteed to be honest; that means, P_j will receive at least $n - 2t$ copies of the string v as the y variable during the round 2 and $n - 2t > t$ that also is important because we are under the condition that $n > 3t$ and we have also proved in the previous lemma that if any honest party have set their y variable to v, then every other honest party will set their y variable to either $\perp$ or v; not to any other different non $\perp$ value.

That means, the honest part is outside v, outside the set $\mathcal{B}$ when they are sending their y variable, they are those y variables will be from this set; they could be either v or $\perp$, nothing else. Of course, the corrupt parties in the set $\mathcal{B}$ can send any string as their y variable to the parties in P_j; that means to the party P_i, they have sent the string v as their y variable; but to the party P_j, they may send a string v' as their y variable.

But if we consider the honest parties in the set $\mathcal{B}$, they will send the string v as their y variable and the honest part is outside the set $\mathcal{B}$ will either sent v or $\perp$ as their y variable. So, now, if we compare the number of y variables the copies, if we compare the number of copies of y variable, which are received by P_j, we will find that the maximum of them the majority of them which are non $\perp$ turns out to be v only and that means, P_j will set its z variable to v and anyhow the output of the π_{BA} protocol will be 1 for P_j.

So, that means, P_j also will be deciding its output value based on the same. If condition and we have shown that the value z will be set to v only by P_j and as a result of that P_j will output the same string v, which has been set as the output by P_i and that shows the consistency property.

(Refer Slide Time: 31:23)



So, that is the domain extension protocol; that means; now we know how to extend or how to design the BA protocols for any domain. The same domain extension you can do for broadcast as well, reliable broadcast. That means, if you have a broadcast protocol, where senders message is a bit and if we want to design a reliable broadcast protocol, where senders message is an ℓ bit string, then again, we can follow this domain extension because we know that if $n > 2t$, any solution for byzantine agreement implies a solution for broadcast and vice versa.

So, we I am not giving you separately the domain extension protocol for the reliable broadcast. It comes because of the equivalence between the reliable broadcast and the byzantine agreement. And this domain extension also means that we can focus our attention only on the bit BA or bit broadcast, we do not have to separately design protocols for a larger domain.

If we have a protocol for the smallest possible domain, where the inputs are bit then, we can use this Turpin-Coan domain extension, and we can design the BA protocol or a broadcast protocol for a bigger domain as well. And later, as I promised, in the course we will see a much more efficient domain exchange protocol.

Thank you.