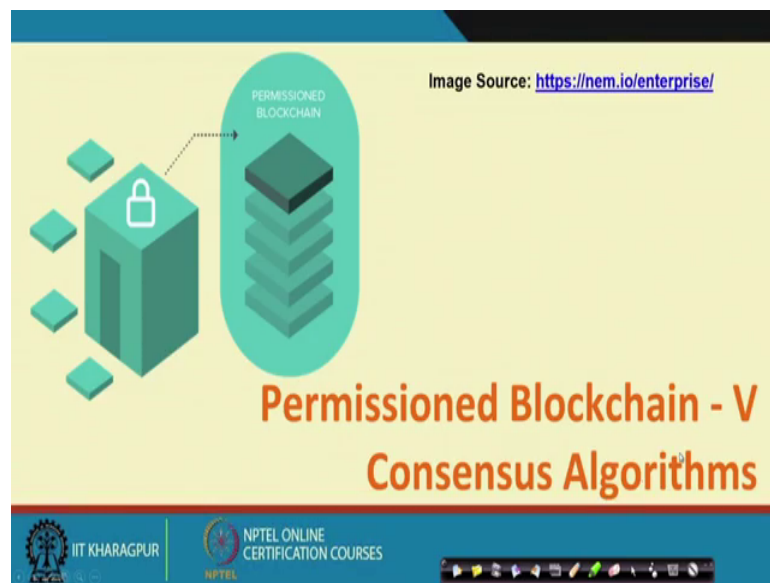


Block chains Architecture, Design and Use Cases
Prof. Sandip Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture- 18
Permissioned Blockchain – V
(Practical Byzantine Fault Tolerance)

(Refer Slide Time: 00:24)



So, welcome back to our course on Blockchain. So, in the last class, we were discussing about the Consensus Algorithm for Permissioned Blockchain environment in a generic distributed system and we have looked into the concept of byzantine general problem and how can you achieve consensus in a synchronous environment with the Lamport's algorithm. So, in that particular algorithm we have seen that if you have f number of lieutenants is faulty, then with $2f + 1$ number of lieutenants in the system you will be able to achieve consensus in the system. Otherwise if the commander is faulty in that case also it $2f + 1$ number of lieutenants. You will be achieved to reach consensus with within synchronous system.

But as you have mentioned earlier that our real systems are not always synchronous, the real systems behaves in asynchronous way; where there is no guarantee that you will receive a message within certain timeout duration. So, in this class we will look into that how can you still receive consensus or at least you will be able to ensure the safety

property of a consensus protocol in the presence of faulty nodes. But remember that in a asynchronous environment, we have the impossibility theorem, that states like in a pure a synchronous environment, if you have at least one number of nodes are faulty or if you have even a single faulty node, it is impossible to reach into the consensus.

So, that is why in this consensus algorithm that we are going to discuss which we call as the, practical Byzantine Fault Tolerance Consensus Algorithm for ensuring the safety property we consider a complete asynchronous system or a pure asynchronous system. But we cannot ensure the liveness property because in that case it will violate the impossibility principle.

So, that is why to ensure lightness, we relax the asynchronous nature or asynchronous guarantee of the system and we will see that well with certain relaxation in the asynchronous condition which we call as a weak asynchronous in the system. So, with weak asynchrony in the system which is not a pure asynchronous system, but a relaxation on the asynchronous system will be able to ensure safety all the time and we will be able to ensure liveness for most of the time

So, let us look into this algorithm which we call as the practical Byzantine Fault Tolerant or PBFT algorithm.

(Refer Slide Time: 03:02)

The slide features a dark blue header with the title 'Practical Byzantine Fault Tolerant' in white. The main content is on a light yellow background with two main bullet points: 'Why Practical?' and 'Real Applications'. The first point lists three sub-points: 'Ensures safety over an asynchronous network (not liveness!)', 'Byzantine Failure', and 'Low overhead'. The second point lists four sub-points: 'Tendermint', 'IBM's Openchain', 'ErisDB', and 'Hyperledger'. At the bottom, there is a blue footer with logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a navigation bar.

Practical Byzantine Fault Tolerant

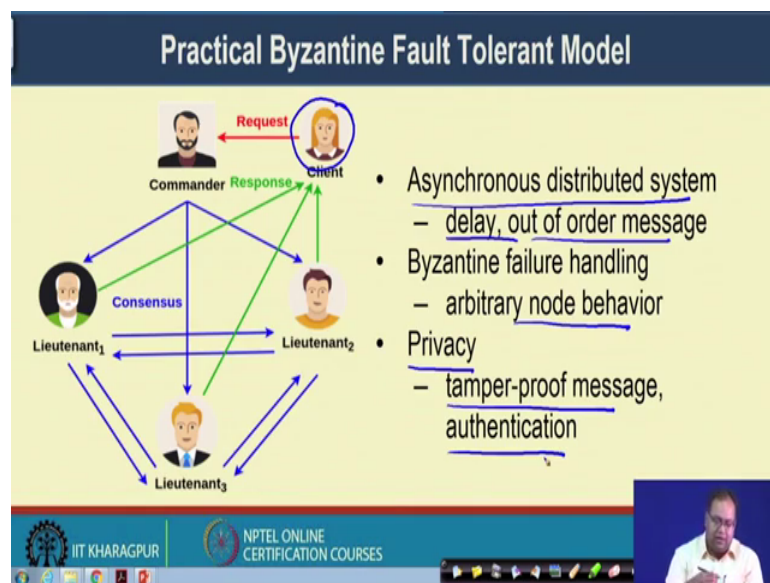
- **Why Practical?**
 - Ensures safety over an asynchronous network (not liveness!)
 - Byzantine Failure
 - Low overhead
- **Real Applications**
 - Tendermint
 - IBM's Openchain
 - ErisDB
 - Hyperledger

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this algorithm is termed as practical because it ensure safety over on asynchronous network, but not liveness on a pure asynchronous network, otherwise it will inviolate the impossibility theorem or the impossibility principle. So, to ensure liveness, we have a weak asynchronous assumption where we deviate from a pure asynchronous system. I will discuss all this construct in details, then the system is able to support Byzantine failure and it has low overhead. So, that is why we called a system as an asynchronous system.

Now, this PBFT algorithm, it is applied for many real applications. It is widely applied for the permission block chain model like the standard meant IBM’s open chain, hyper ledger this kind of platform which uses permission blockchain environment. So to ensure consensus among the participants in a closed environment, we apply this kind of PBFT type of algorithm. So, we will go to the details of this PBFT algorithm.

(Refer Slide Time: 04:13)



So, here also we are having this Byzantine modern the Byzantine Fault Tolerant Model. So, the broad idea of the system is something like this. So, in this system there is a client, the client submits the request to the commander. So, here the client if you think about from the Blockchain principle that clients are the individual Blockchain clients who are submitting the transaction and the commanders are certain nodes in the system, who are responsible for ensuring consensus in the system.

And you have multiple nodes, so this commander in cooperation with the multiple nodes in the systems that have been shown here using blue lines. So, with the help of this multiple lieutenants in the system the commander come to the consensus and once the system comes to the consensus, it sends a response back to the client whether the system has committed based under consensus algorithm or not.

So, you can map this entire principle in a block chain environment, where the client submits a transaction and there are certain nodes in the network; who are designated for running the consensus algorithm. Those nodes collectively design the consensus algorithm or run the consensus algorithm and they decides whether the transaction can be committed or not. If the transaction can be committed in the system, then that response is sent back to the client. So, that way this entire system works.

Now, as you have done as we have assumed that it is asynchronous distributed system because it is an asynchronous distributed system. You can have delay in transmitting the message and you can also receive out of order messages. Then the system can have Byzantine failure like the arbitrary node behaviour. So, some nodes can vote for a transaction and against for a transaction. Additionally this PBFT algorithm it also supports privacy over the system.

So, it ensured that the messages are tamper- proof; it applies a hashing technique similar to Blockchain. We will discuss that in details and it also applies the authentication technique through this digital signature mechanism. So, that none of the messages transfer from individual nodes in the system can be tampered or can beit stopped.

(Refer Slide Time: 06:41)

Practical Byzantine Fault Tolerant Model

- A state machine is replicated across different nodes $2f+1$
- $3f + 1$ replicas are there where f is the number of faulty replicas
- The replicas move through a succession of configurations, known as views
- One replica in a view is primary and others are backups

The diagram shows four squares representing replicas within a circle labeled 'view'. The first square is white, the second is white, the third is shaded with diagonal lines, and the fourth is white. An arrow points from the text 'One replica in a view is primary and others are backups' to the first square.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, that is the system model, so we consider a state machine a replicated state machine concept that we discussed earlier in the context of distributed consensus. So, we have a state machine which is replicated across different nodes and we have $3f + 1$ replica. So, this number of important this number is important where, we are deviating from our consensus algorithm in a synchronous environment. So, what we have seen that in a synchronous environment if you have f number of faulty nodes; then you require $2f + 1$ number of lieutenants or $2f + 1$ number of nodes apart from the commander to ensure consensus.

But in case of asynchronous network, we require $3f + 1$ replicas. But why do you require $3f + 1$ replicas? At a point while we will discussing about the algorithm, I will discuss about this particular assumption that to ensure f number of faulty replicas. So, why do we require $3f + 1$ number of replicas which will ensure you having consensus in the system.

Now, these replicas they move through a successive of configuration known as views. So, this views are something like that among this set of replicas; if you try to map it in a Byzantine general model, we have this commander and we have individual lieutenants So, similarly we have among these replicas, we have one primary and multiple other backups So, that way this setup of primary and a backup we consider them as a single view.

So, assume that you have 4 different replica here and among these 4 different replicas, you have this replica which has been designated as a primary replica and others are designated as a backup replica. So, this replica takes control of the state machine and these other replicas stored a backup of the state machine.

Now this particular set of configuration that we call as one view. Now in different views the primaries and the backups can change it may happen that in a different view. So, in a different view, this primary becomes so this primary becomes a backup and another backup becomes a primary. This is a second view so that way the replicas, they move through a successive set of configurations which are known as views, and 1 replica in a view is primary and others are known as backup.

(Refer Slide Time: 09:38)

Practical Byzantine Fault Tolerant Model

- Views are changed when a *primary* is detected as faulty
- Every view is identified by a unique integer number v
- Only the messages from the current views are accepted

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, these views are changed when a primary is detected as faulty. So, whenever the backups, they collectively detect that a primary is faulty will look into this view change mechanism. Then these views are changed and every view is identified by a unique integer number v , with a unique integer number v and only the messages from the current views are accepted.

So, if there is a view change, then the messages which have been broadcasted in the previous a view and because of the asynchronous nature of the system, it may always happen that; you are receiving the message, a delayed message, or out of order message;

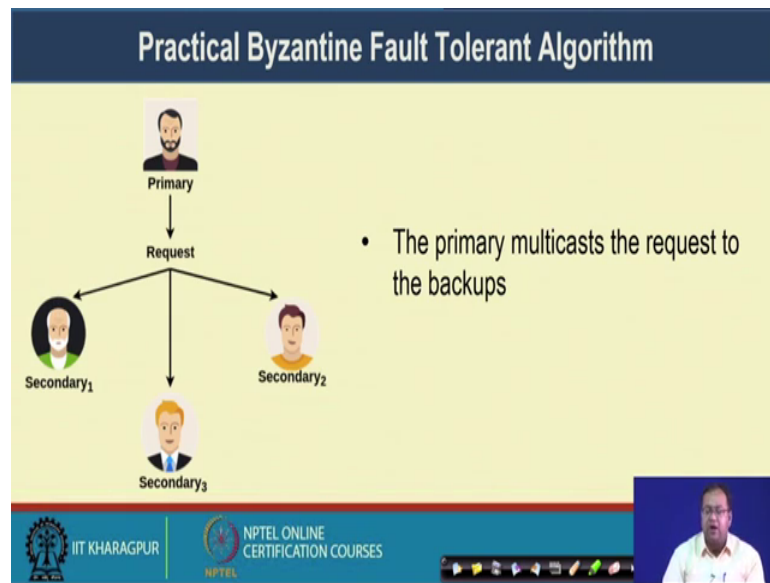
and you are receiving message from a previous view. So, the messages from the previous views are discarded and only the messages from the current views they are accepted.

(Refer Slide Time: 10:31)

The slide features a dark blue header with the title "Practical Byzantine Fault Tolerant Algorithm" in white. Below the header, on a light yellow background, is a diagram. On the left, a female icon labeled "Client" has an arrow labeled "Request" pointing to a male icon labeled "Primary". To the right of the diagram is a bullet point: "• A client sends a request to invoke a service operation to the primary". At the bottom of the slide, there is a small text reference: "Castro, Miguel, and Barbara Liskov, 'Practical Byzantine fault tolerance,' OSDI, Vol. 99, 1999". The bottom of the slide also contains logos for "IIT KHARAGPUR" and "NPTEL ONLINE CERTIFICATION COURSES", along with a small video inset of a presenter.

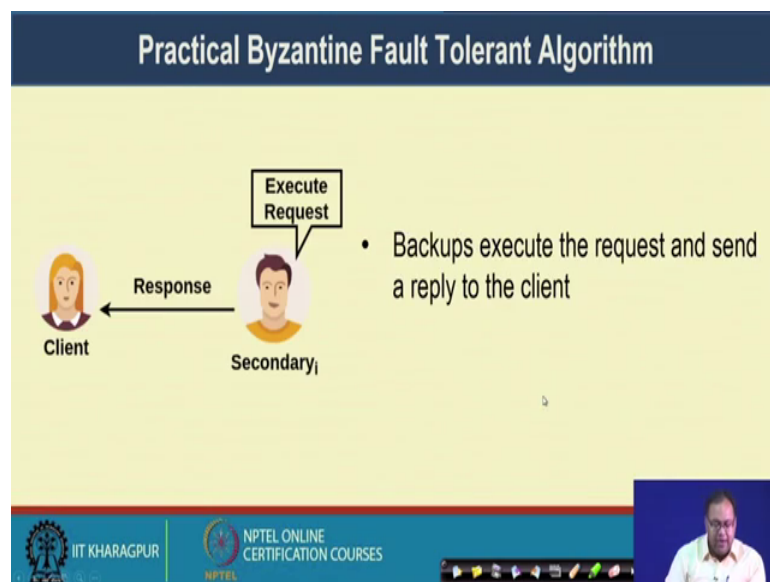
So, let us look into a simplified view of this PBFT algorithm. So, this PBFT algorithm was proposed by Castro Miguel and Barbara Liskov that was first came into OSDI in 1999 and the title of the paper was Practical Byzantine Fault Tolerant. So, I suggest all of you to read this paper in details. So, the idea of this algorithm is as follows: so the client sends a request to the primary. So, the client executes some kind of service operation like in case of your shared storage the client executes a write instruction, or in case of a block chain environment the client initiates a transaction. So, all this and request they are moved to the primary.

(Refer Slide Time: 11:21)



Then the primary it sends this request to all the secondary replicas. So, this primary it multicast a request to the backup; so, these backups are the secondary replica whatever you call it. So, this information is sent to the backups.

(Refer Slide Time: 11:38)



Then, these backups they execute the request and they try to comes to the consensus based on this PBFT algorithm that we will discuss shortly. So, after executing the request either the client request is successful or the client request is failure say. For example, in case of shared storage architecture, if that client is trying to initiate a discrete operation

either that district operation was successful. If it is able to write that instruction correctly into every individual replica, then it is successful; otherwise it will be a failure. Similarly for the block chain perspective a transaction if it is a valid transaction if all the backups or all the replicas decide that the current transaction is a correct transaction. Then it sends a success or a commit message to the client, otherwise it will send a failure message to the client. So, the backups they will execute the request and accordingly send a reply to the client.

(Refer Slide Time: 12:48)

Practical Byzantine Fault Tolerant Algorithm

Result
Client
Secondary₁
Secondary₂
Secondary₃

- The client waits for $f + 1$ replies from different backups with the same result
 - f is the maximum number of faulty replicas that can be tolerated

Handwritten notes:
 $3f+1$
 $f \rightarrow \text{Faulty}$
 $2f+1 \rightarrow \text{Correct}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the client will receive all these replies. Once the client receives all the replies, so the client basically waits for f plus number 1 of replies from different backups with the same result where f is the maximum number of faulty nodes, faulty replicas that can be tolerated. So, once the client receives f plus 1 replies from different backups with the same result. So, these keywords are important that the client need to receive f plus one replies from different backups and the results are same; that means, the client can decide that it has got majority of the correct faulty.

So, remember that you have in this PBFT algorithm; you have $3f$ plus 1 number of different replicas. So, out of this $3f$ plus 1 number of replicas; you have f number of replicas which are faulty, but whenever the client is receiving the message if it gets f plus 1 message with the same result; that means, the majority of the nodes here actually not the majority of the nodes the correct nodes majority of the correct nodes they are sending

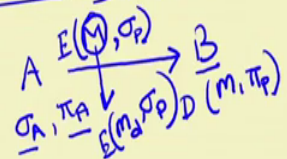
a reply to him. So, when majority of the correct nodes are sending a reply to you then you can say that well the system is reached to a consensus and the client accept or commit that particular message.


So, in other words, you have $3f + 1$ number of replicas in the system. Out of f number of replicas are faulty and $2f + 1$ replicas are correct. So, if you are receiving $f + 1$ replies from different backup with the same result, then you can be guaranteed that well you are receiving a result which belong to the majority of the correct; the majority of the correct replicas. So, if you receive a message from majority of the correct replicas you can assume that message to be the consensus message and you can include it the included to your system by committing that particular request ok.


(Refer Slide Time: 15:13)


Three Phase Commit Protocol - Pre-Prepare

- **Pre-prepare:** Primary assigns a sequence number n to the request and multicast a message $\langle \langle \text{PRE - PREPARE } (v, n, d) \rangle_{\sigma_p}, m \rangle$ to all the backups
 - v is the current view number
 - n is the message sequence number
 - d is the message digest
 - σ_p is the private key of primary - works as a digital signature
 - m is the message to transmit




IIT KHARAGPUR


NPTEL ONLINE
CERTIFICATION COURSES



Now, this entire consensus algorithm in PBFT it works in three different phases: Pre-Prepare, Prepare and Commit. So, we will look into these three different phases in details. So, in Pre-Prepare Phase, the primary it assigns a sequence number. So, initially the client has sent a request to the primary. Once the client has sent a request to the primary, then this Pre Prepare Phase starts in the Pre Prepare Phase the primary it assigns a sequence number n to the request and multicast a message called PRE – PREPARE along with certain parameters to all the backups.

So, let us see what are these parameters. So here v is the current view number. So, you are ensuring that the message that you are going to receive it is from the current view.

Then n is the sequence number, n is the message sequence number that the primary has included in the message, d is the message digest. And this entire thing entire pre prepare message it has encrypted with the primary key with the private key of the primary. Now if you remember, the digital signature method that we have discussed during the first few classes.

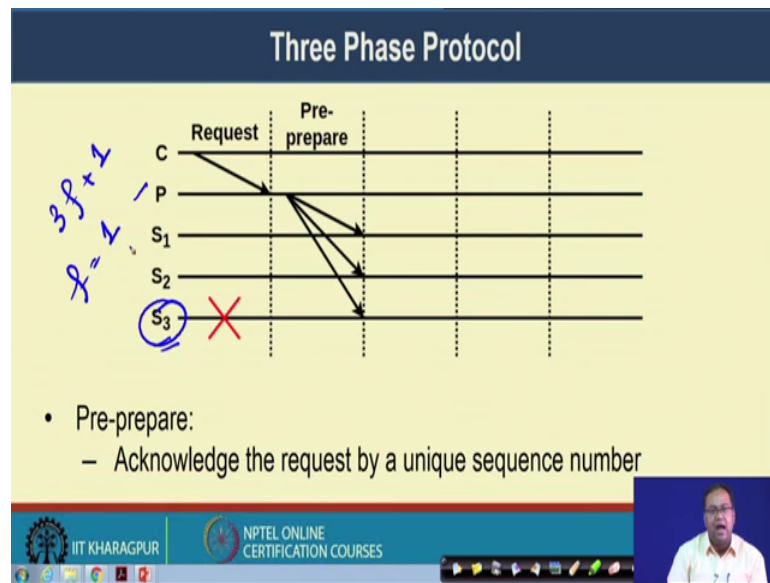
So, whenever A is sending certain messages to B and A has its own primary key, say σ_A is the primary key for A and say π_A is the public key for A. So, A has two different key; one is a private key another is a public key. So, A can encrypt a message. So, it can encrypt a message with this private key and at the other end B can decrypt this message with the public key.

Now, these particular mechanisms works like a digital signature and it ensures that, whatever message you are receiving from A that message has actually originated from A. So, with this architecture this digital signature architecture you ensure that well this Pre-Prepare message is actually coming from the primary because it is encrypted to it the private key of the primary.

And here we put the message digest because if you again remember the digital signature method rather than putting the message here because the message size is long. We apply the hash based technique to reduce the message size and rather than putting the message we put the message digests say $m \star d$ digest of the message, which is encrypted with the private key or primary.

So, the similar concept is applied here with the message that we are transmitting here. The message m you also transfer the message digest and the entire thing encrypted to it the private key of the primary. So, that all other backups so whenever they will decrypt this message with the public key of primary, they will be able to know that this particular message has been originated from primary and it corresponds to the message which has been sent. So, this is the pre prepare message that is sent by the primary to all the backups after assign the sequence number n .

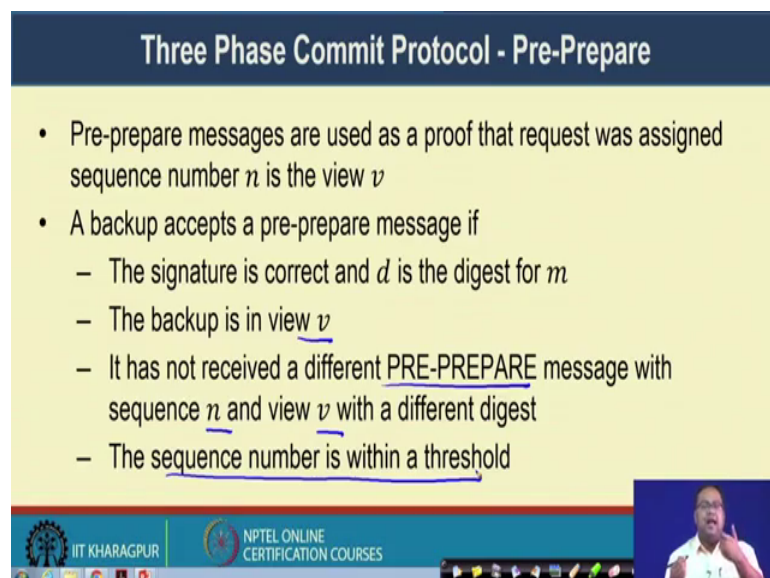
(Refer Slide Time: 18:41)



So, that is the thing in Pre-Prepare. So, once that client shares the request then in the Pre-prepare message, you broadcast this pre prepared message to all the secondary.

So, here in this example we are assuming that once again there is faulty. So, we have $3f + 1$ number of different replicas. So, we are considering here f equal to 1 we have 1 faulty replica which is history. So, we have a total of 4 different replicas 1 primary and 3 backups. So, in the pre prepare phase that client assigns a unique sequence number to the request message and broadcast that request message to all the replicas.

(Refer Slide Time: 19:24)

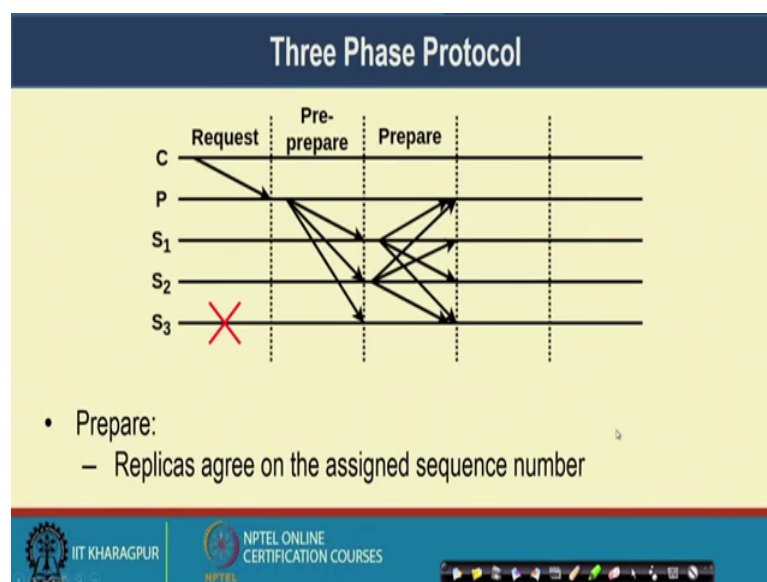


The second phase of the Three Phase Commit Protocol is the Pre-prepare. So, before going to the second phase of the Three Phase Commit Protocol, so, in the Pre Prepare Phase; so, this Pre Prepare messages they are used as a proof that the request was assigned the sequence number in the view v . So, the message is in the current view and it has been assigned the unique sequence number.

Now a backup it accepts a pre prepare message if the signature is correct and d is the digest for m to verify the digital signature that it has been originated from the primary the backup is in view v . So, you are in the latest view it has not received the different pre prepare message with the sequence number n and view v with a different digest.

So; that means, the message has not been tampered if the message has been tampered; that means, the digest would be different and the sequence number is within a threshold. So, it also ensures that this sequence numbers are not too old, such that there is no kind of replay attack in the system. So, an attacker is not getting that message or just capturing the message which was being transferred by the primary and replicating it after some time. So, it is not a replica. So, then the backup accepts a pre prepared message

(Refer Slide Time: 20:44)



The second phase is the Prepare phase in the prepare phase this secondary it sends a prepare message to all the replicas

(Refer Slide Time: 20:58)

The slide is titled "Three Phase Commit Protocol - Prepare". It contains two main bullet points. The first bullet point states: "If the backup accepts the PRE-PREPARE message, it enters prepare phase by multicasting a message $\langle \text{PREPARE}, v, n, d, i \rangle_{\sigma_i}$ to all other replicas". Below this text, there is a handwritten blue arrow pointing to the right, with three vertical lines underneath it, suggesting a multicast action. The second bullet point states: "A replica (both primary and backups) accepts prepare messages if". Below this, there are three sub-bullets: "Signatures are correct", "View number equals to the current view", and "Sequence number is within a threshold". Each of these sub-bullets has a blue underline. At the bottom of the slide, there is a video inset showing a person speaking, and a footer with logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

So, let us look into this prepare message in details. So, if the backup it accepts the pre prepare message, they need to enters this prepare face and in this prepare phase it multicast a prepare message.

So, the prepare message all has the similar set of parameters which were there in the Pre Prepare Phase, the view number, the sequence number of the message the message digest along with the identity of the backup who is which is broadcasting this prepare message. And the entire thing is encrypted by the private key of the backup to ensure it as a digital signature and these prepare messages is originating from backup i.

So, that is the prepare message which has been broadcasted in the prepare stage. So, a replica, the replica here can be a primary as well as backup. It accepts the prepare message if the signatures are correct. So, it is coming from the intended backup the view number it equals to the current view. So, the message is in the current view and the sequence number is within a threshold the sequence number is not too old. So, it is not a replica.

(Refer Slide Time: 22:16)

Three Phase Commit Protocol

- Pre-prepare and prepare ensure that non-faulty replicas guarantee on a total order for the requests within a view
- Commit a message if
 - $2f$ prepares from different backups matches with the corresponding pre-prepare
 - You have total $2f + 1$ votes (one from primary that you already have!) from the non-faulty replicas

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, these two stages the pre prepare and prepare, they ensure that non faulty replicas they guarantee on a total order of the request within a view. So, it is like that every individual request they are assigned one sequence by the by the primary. So, once the primary assigns a sequence to individual messages or individual transactions which are coming from the client. So, all these messages are ordered based on the sequence number which is assigned by the primary.

So, all the backups they process the messages in the order of that sequence number. So, if a sequence number has along with; if a sequence number has is higher than the current sequence number then that one is processed first. So, in that order of the sequence number the messages are being processed.

So, after this prepare and the pre prepare stage once every replica gets this prepare message, then it commits a message if $2f$ prepare messages from different backup it matches with the corresponding pre prepare. So, whatever has been transferred in the pre prepare message, you have received a prepare message with the same message and the same message digest and it is coming from at least $2f$ number of prepare message.

So, you have received $2f$ such message. So, here you have a total of $2f$ plus 1 votes that you have received in total 1 vote is from the primary and that you already have and you have got $2f$ number of votes from other replicas.

(Refer Slide Time: 24:17)

Three Phase Commit Protocol

- **Why do you require $3f + 1$ replicas to ensure safety in an asynchronous system when there are f faulty nodes?**
 - If you have $2f + 1$ replicas, you need all the votes to decide the majority - boils down to a synchronous system
 - You may not receive votes from certain replicas due to delay, in case of an asynchronous system
 - $f + 1$ votes do not ensure majority, may be you have received f votes from Byzantine nodes, and just one vote from a non-faulty node (note Byzantine nodes can vote for or against - You do not know that a priori!)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if you remember that whenever you are you are waiting for this kind of vote. So, here in this context, here we in this context let us discuss that why do we require $3f$ plus 1 number of replica to ensure safety in asynchronous system when there are f number of faulty nodes.

Now, here the thing is that if you have $2f$ plus 1 replica to have a majority decision, you need to ensure that you have received all the votes and out of all the votes at least f plus 1 votes are f plus 1 votes are giving the majority decision. So, to take a majority decision with $2f$ plus 1 number of replica, you need a vote from all the replicates. So, this boils down to a synchronous environment where you need to wait for to get the message from all the replicas.

Now, in case of asynchronous system, you may not receive the votes from certain replicas because of the delay or you may receive out of order messages in that case if you just receive f plus 1 vote. So, that f plus 1 vote does not ensure majority. So, if you receive continuous f plus 1 vote and that f plus 1 vote are giving you the same instruction or the same information, then you can accept that vote, but it may happen that out of this f plus 1 vote certain votes are coming from the faulty node.

So, you do not know whether those votes are coming from the faulty nodes or from the correct nodes. So, whenever you have received certain f number of force those f number of votes may be coming from the faulty nodes and all the faulty nodes are behaving

equally. And they are possibly saying say in the Byzantine general problem say, they are saying about literate.

Now, if you get retreat message from f number of nodes you really do not know that f number of nodes are faulty or some other nodes are faulty or they are correct. Now say you have received f retreat message and after that you have received one attack message. Now if you receive a free treat message and one attack message, you cannot be sure that whether those f number of votes that you have received those are from the normal node or from the byzantine nodes because you know that there are f number of faulty nodes in the system and this is an asynchronous system because this is an asynchronous system it may happen that you have also not received certain votes

(Refer Slide Time: 26:56)

Three Phase Commit Protocol

- **Why do you require $3f + 1$ replicas to ensure safety in an asynchronous system when there are f faulty nodes?**
 - If you do not receive a vote
 - The node is faulty and not forwarded a vote at all
 - The node is non-faulty, forwarded a vote, but the vote got delayed
 - Majority can be decided once $2f + 1$ votes have arrived - even if f are faulty, you know $f + 1$ are from correct nodes, do not care about the remaining f votes

Handwritten annotations: A circled f , a circled $f+1$ with an arrow pointing to "Attack", and a circled $2f+1$.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, to ensure this whenever you are not receiving a vote it may happen that, that particular node is faulty and it has not forwarded a vote at all. So, a faulty node here has the option that it will not forward any vote because it is an asynchronous system either that is the option. Or the second option is like the node is not faulty the node is a correct node it has sent to a vote, but that vote is strapped somewhere in the network. So, you have not received that vote.

So, that can be the situation where the vote got delayed, but if you receive $2f$ plus 1 vote and whenever you have received $2f$ plus 1 vote and if you see that well out of that $2f$ plus 1 for that you have received out of them f plus 1 or it the majority if that is the case then

you can guarantee that well whatever the majority vote says out of that $2f + 1$ vote that you have received that is the consensus.

So, it is like that to have the consensus in the system, you can have $2f + 1$ number of correct nodes and f number of faulty nodes because you may not receive the votes from certain number of correct nodes and you can't tolerate such nodes or you can tolerate such kind of votes after up to f number of times because you can have f number of faulty votes.


So, whenever you are receiving the first f votes you do not know whether those first f votes are coming from a correct node or from a faulty node. So, that is why we require $3f + 1$ number of different replicas in a system where f number of replicas are faulty replicas and $2f + 1$ number of replicas are correct replicas. In this particular case, if you are receiving $f + 1$ votes and that $f + 1$ votes are seeing the same thing like all the $f + 1$ votes are saying that attack. Then you can decide and unanimously that well in your system there can be f number of faulty nodes. So, what $f + 1$ force you are getting that gives you a kind of correct result or a majority result because that is going to be said by all the correct nodes.

So, to say it in other words if you are getting $f + 1$ say $f + 1$ number of votes and all the $f + 1$ number of votes are attacked. So, out of this $f + 1$ number of votes you know that f number of votes can be faulty. So, even if out of this $f + 1$ attack votes f votes are coming from the malicious or the Byzantine nodes and you do not able to predict that what the Byzantine nodes are going to say there is at least 1 node which is in the form of attack and it is a non faulty node. So, if it is the vote for an on faulty node then all the non faulty nodes in the system they are going to vote for attack. So, you can correctly decide that attack is your final consensus value. So, that is why we require $3f + 1$ number of replicas to detect f number of faulty nodes.

(Refer Slide Time: 30:22)

Three Phase Commit Protocol - Commit

- Multicast $\langle COMMIT, v, n, d, i \rangle_{\sigma_i}$ message to all the replicas including primary
- Commit a message when a replica
 - Has sent a commit message itself
 - Has received $2f + 1$ commits (including its own)

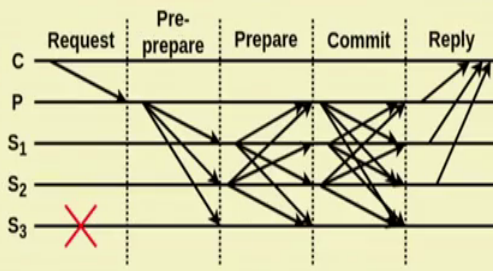


Well at the end every node multicast the commit message to all the replicas include primary.


So, whenever you have received $2f$ plus 1 number of such commit messages from that $2f$ plus 1 number of commit message. If f plus 1 from the majority decision it can decide that whether; the transaction has been accepted or whether the right instruction that request that has been sent by the client, whether that request has been accepted or not because you are getting on that a majority faulty.

(Refer Slide Time: 30:57)

Three Phase Protocol

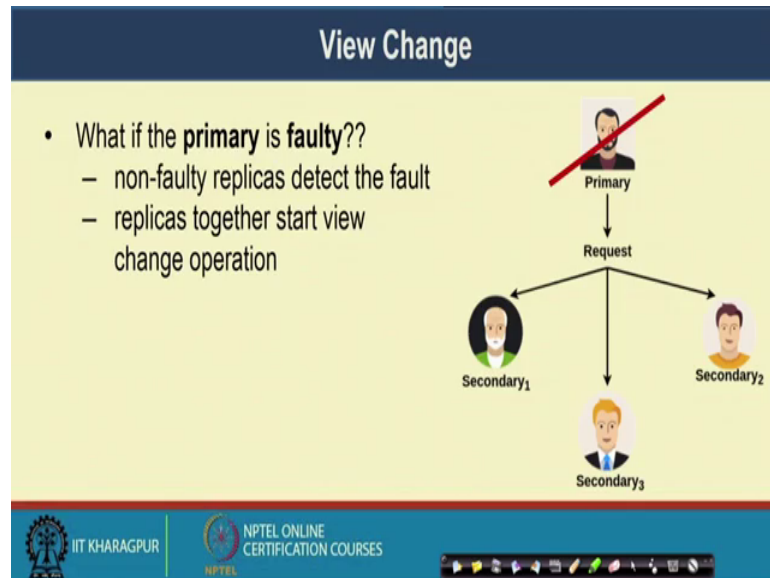


- Commit:
 - Establish consensus throughout the views



So, that is the final commit message after final getting the commit message. every individual node takes a decision and it sends that reply back to the client.

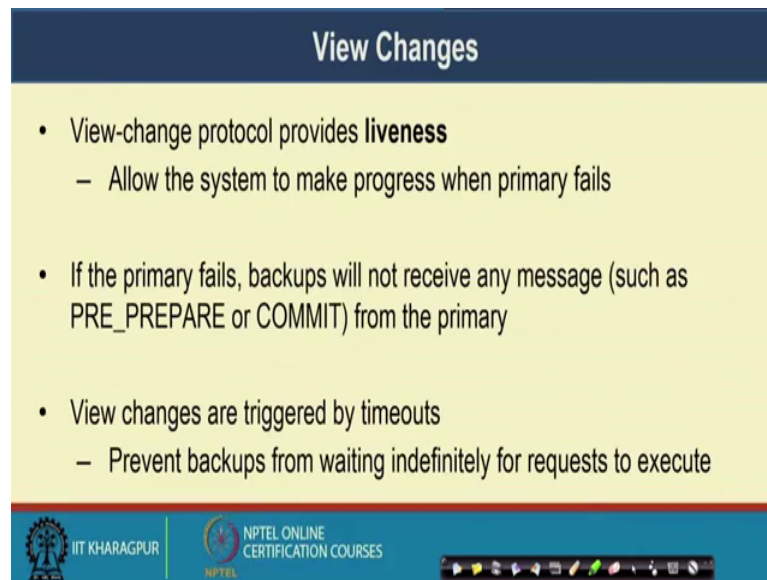
(Refer Slide Time: 31:08)



Now, let us look into the case that when the primary is faulty so if a secondary is faulty. So, we have seen that we can tolerate up to f number of such failures and within this f number of failures if f number of secondary's or the backups have are faulty nodes the system can tolerate that, but what if the primary is faulty. Now in that case the non faulty nodes they detect the fault. So, the backups they detect the fault collectively and they together start a view change operation.

So, they remove the primary from the system or they consider that the replica which was designated as the primary in view v that will get changed another replica from the backup that will be designated that is the primary.

(Refer Slide Time: 31:59)



The slide is titled "View Changes" in a dark blue header. The main content is on a light yellow background and consists of three bullet points. The first bullet point states that the view-change protocol provides liveness, with a sub-point allowing the system to make progress when the primary fails. The second bullet point notes that if the primary fails, backups will not receive messages like PRE_PREPARE or COMMIT. The third bullet point says view changes are triggered by timeouts, with a sub-point preventing backups from waiting indefinitely. The footer contains logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a navigation bar.

- View-change protocol provides **liveness**
 - Allow the system to make progress when primary fails
- If the primary fails, backups will not receive any message (such as PRE_PREPARE or COMMIT) from the primary
- View changes are triggered by timeouts
 - Prevent backups from waiting indefinitely for requests to execute

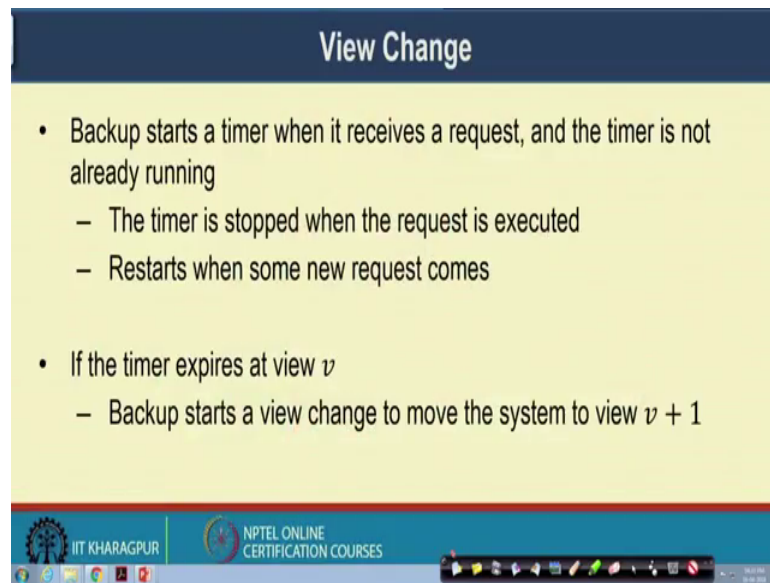
Now, this view change protocol it ensures liveness, but to ensure this view change you have to ensure that you will receive the message from all the nodes within a timeout duration. So, here comes the weak synchrony assumption in the system like to ensure the safety, you do not require a synchronous assumption.

But to ensure the liveness to ensure that the view change are done at a proper time interval you have to ensure that all the backups they are able to detect that the primary is not sending any message within some timeout duration. So, whenever you are putting this kind of timeouts in the system.

So, here we put a timeout in the system when you put the timeout in a system you have an assumption that if you are not receiving the message within this duration then you are considering that message to be lost and the primary to be faulty. So, here we are deviating from the pure asynchronous assumption.

So, the pure asynchronous assumption says that your transfer of masses can be arbitrarily delayed, but here we are assuming that it is not truly arbitrarily delayed to ensure liveness rather to ensure liveness you have certain level of synchrony if you are not receiving the message from the primary. So, this primary to secondary communication it is it is a synchronous communication if you are not listening the message from the primary within timeout duration you assume the primary to be faulty ok.

(Refer Slide Time: 33:41)



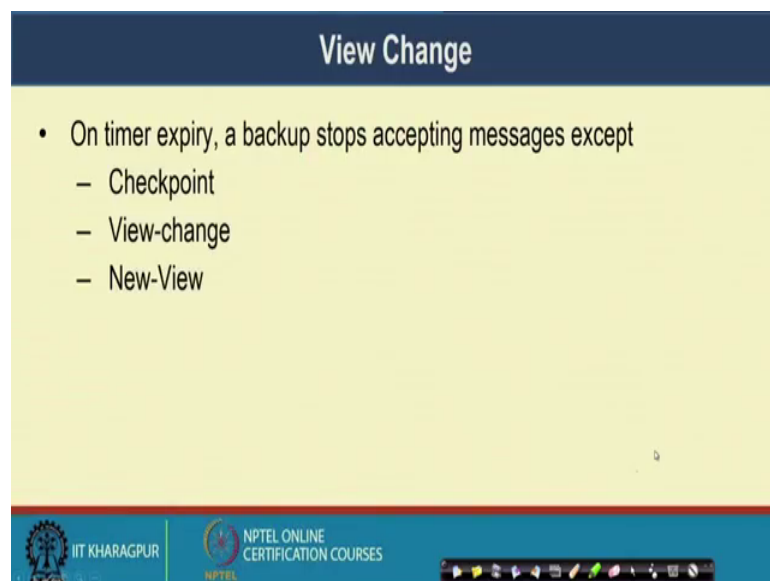
The slide is titled "View Change" and contains the following text:

- Backup starts a timer when it receives a request, and the timer is not already running
 - The timer is stopped when the request is executed
 - Restarts when some new request comes
- If the timer expires at view v
 - Backup starts a view change to move the system to view $v + 1$

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a system tray with various icons.

So, these backups they starts a timer when it receives a request and the timer is already running. So, the timer is stopped when the request is executed. So, within that timeout duration everything is complete and the timer gets restarted when some new request comes. So, when some new request comes you just look into whether you are receiving the messages from the primary within that timeout message or not. Now if the timer, timer express a view v , then the backup starts a view change to move the system to an under view v plus 1.

(Refer Slide Time: 34:15)



The slide is titled "View Change" and contains the following text:

- On timer expiry, a backup stops accepting messages except
 - Checkpoint
 - View-change
 - New-View

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a system tray with various icons.

So, on timer expiry a backup stop accepting any new messages; except the normal checkpointing message, the view change message and a new view message. So, only view change is being performed during that time. Now this view change operation is little straightforward.

(Refer Slide Time: 34:33)

View Change

- Multicasts a $\langle VIEW_CHANGE, v + 1, n, C, P, i \rangle_{\sigma_i}$ message to all replicas
 - n is the sequence number of the last stable checkpoint s known to i
 - C is a set of $2f + 1$ valid checkpoint messages proving the correctness of s
 - P is a set containing a set P_m for each request m that prepared at i with a sequence number higher than n
 - Each set P_m contains a valid pre-prepare message and $2f$ matching

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

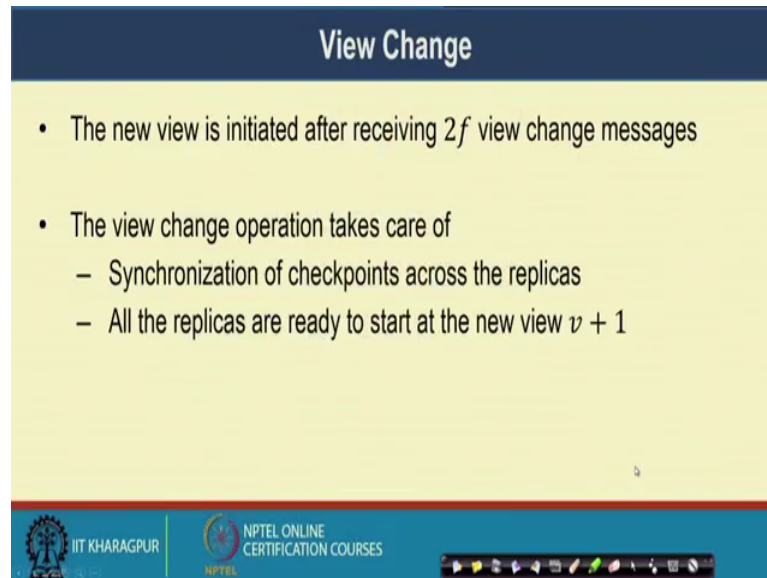
So, when a primary has a timeout it multicast a view change message to all the replicas along its certain option. So, it initiates a view change for a new view v plus 1 along with the sequence number of the last table check 1; that means, it is the sequence number of the last transaction that is being accepted and then certain sets.

So, this c it is a set of $2f$ plus 1 valid checkpoint messages it proved the correctness of this s the correctness of the checkpoint. So, it provides it ensures that the last checkpoint at this particular node has received that particular last checkpoint is a correct checkpoint. So, that was based on the normal PBFT safety algorithm. So, it ensures the safety criteria of the system and this P it is a set containing another set P_m for each request m that prepared at i with sequence number higher than n .

So, it ensures that well certain number of request messages are still flowing in the network, but you want to initiate of view change. So, whenever you want to initiate of view change you also embed the request message which have been initiated in the previous view, but that has not yet committed because once the view change is being done then those old messages need to be taken care of.

So, because of that this particular set P it embed all such old requests. So, each set P m it contains a valid pre prepare message and $2f$ matching of that pre prepare message.

(Refer Slide Time: 36:22)



The slide is titled "View Change" and contains the following text:

- The new view is initiated after receiving $2f$ view change messages
- The view change operation takes care of
 - Synchronization of checkpoints across the replicas
 - All the replicas are ready to start at the new view $v + 1$

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a navigation bar.

So, a new view it is initiated after receiving $2f$ number of view change message. So, among you know that in the system you had $3f + 1$ number of total numbers of nodes. Out of them for faulty and the primary has also observed the fault. So, in that case if you receive $2f$ plus number of view change message. So, it is like that that $2f$ messages are received from the correct replica. So, a view change need to be initiated. So, this view change operation it takes care of the synchronization of the check point across replicas.

So, that synchronization part I am not going to the details rather keeping it as a take home assignment for you to read the PBFT paper and to understand this part. That the view change operation it takes care of the synchronization of the check point, like if may happen that well certain whenever you are sending the view change message due to the asynchronous nature of the system. A certain nodes they are still in their pre prepared at the prepare state and they will also be able to get the correct checkpoints and it ensures that all the replicas they are ready to start at the new view at view v plus 1 when the primary has changed from the old primary to another primary.

(Refer Slide Time: 37:48)

Correctness

- **Safety:** The algorithm provides safety if all non-faulty replicas agree on the sequence numbers of requests that commit locally




Image Source: <http://www.differencebetween.com/>

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the safety property all the at the algorithm. So, the algorithm it provides safe if the non faulty replicas they agree on the sequence numbers of the request that commit locally. So, we have observed that if you are receiving $f + 1$ number of correct messages or $f + 1$ number of messages from the correct replicas. Out of $2f + 1$ message that you are going to receive; if all the replicas relies on the latest sequence number then that part get committed. So, only the latest sequence number will get committed in the system they are to ensure the safety proud criteria.

(Refer Slide Time: 38:25)

Correctness

Liveness: To provide liveness, replicas must move to a new view if they are unable to execute a request

- A replica waits for $2f + 1$ view change messages and then starts a timer to initiate a new view (*avoid starting a view change too soon*)
- If a replica receives a set of $f + 1$ valid view change messages for views greater than its current view, it sends view change message (*prevents starting the next view change too late*)
- Faulty replicas are unable to impede progress by forcing frequent view change

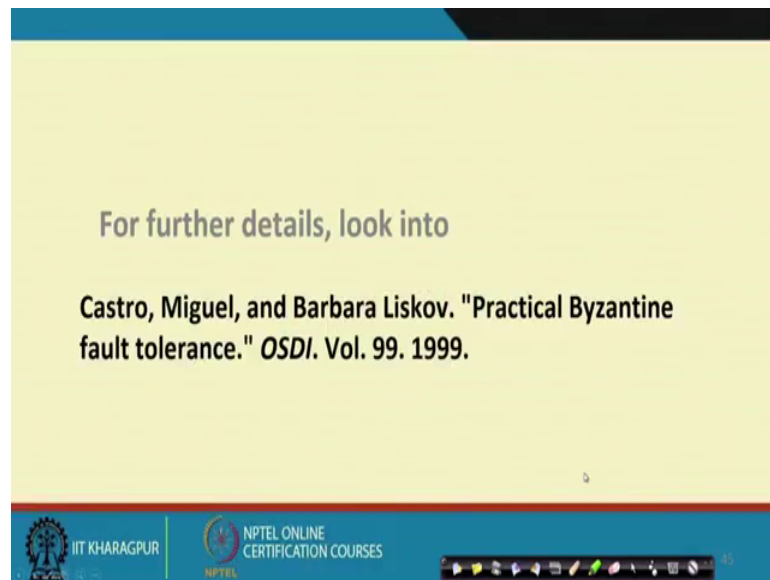
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And the liveness criteria to ensure the liveness criteria we need to ensure that whenever there is a fault in the primary the system moves from one view to another view. So, in that particular case a replica it waits for $2f + 1$ view change messages and then starts a timer to initiate a new view it avoids starting of view change too soon. If you start a view change too soon, then a faulty replica may just initiate a view change very periodically and it will hamper the liveness of the system the system will not be able to progress. If there are frequent view changes, so, to prevent frequent huge changes we ensure that once you are receiving sufficient number of view change requests then only you change the view from one node to another node by ensuring that the primary is faulty.

Now, again you have to ensure that the few change are not delayed too much. If the view change are delayed too much, they will also it will affect the liveness property. So, if a replica receives a set of $f + 1$ valid view change message for the views greater than its current view, then it also initiates a view change. So, it is like that the view change are initiated for two different conditions either a timeout occurs. So, you need $f + 1$ different timeout to occur otherwise you receive $f + 1$ number of different view change message and once you are receiving $f + 1$ different view change message you also initiate a view change operation.

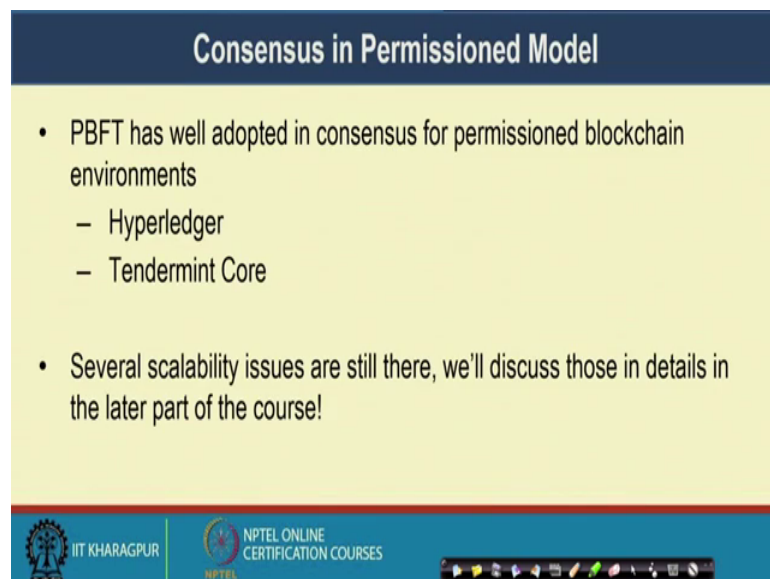
And this way the faulty replicas they are unable to impede the progress by forcing frequent view change. So, even if you have a faulty replica in the system you cannot have very frequent view change in the system.

(Refer Slide Time: 40:12)



So, look into the further details of this paper for the theoretical proof of PBFT.

(Refer Slide Time: 40:18)



And as you have mentioned that this PBFT it has well adapted for consensus in permission Blockchain environment like hyper ledger and tender mint and well many of the scalability issue are still there, because it is a open environment and every node meet to sense that multicast message to every other node in the pre prepare prepare and a commit message.

So, the system has a high message complexity because of which you have scalability issues. So, in the later part of the lecture, we will look into the scalability issues in details and look into several optimization on top of the PBFT protocol which have been adapted by the Blockchain researcher.

So, this gives us a broad idea about the permission less Blockchain and the permission Blockchain model different type of consensus protocols which are there. So, from the next class onwards Praveen will be taking the classes and he will look into the practical aspects of hyper ledger and different implementation details of Blockchain along with the industry use cases. So, I will be back after few lectures after few lectures from Praveen and again discuss about the various use cases and research aspects which are there in the Blockchain domain. On which we are exploring currently. So, hopefully we will be enjoying the practical aspects of Blockchain or the implementation details of Blockchain and the corresponding demos that will be shown by Praveen from the next lecture onwards. We will come back again after those demo sessions. So, thank you again for attending we will next look into different use cases of Blockchain.

Thank you.