NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Bishakh Chandra Ghosh

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 36: Hyperledger Fabric 3

CONCEPTS COVERED

- **Hyperledger Fabric Chaincode**
- **Fabric Transaction Flow**
- **Writing your own Chaincode**

KEYWORDS

- Fabric
- Chaincode
- Fabric Transactions

Fabric Smart Contracts

- Defines common data, rules and processes for businesses to transact through the ledger.
- Smart contracts are packaged as **chaincodes**.

Seller Organization

ORG1

```
application:

seller = ORG1;
buyer = ORG2;
transfer(CAR1, seller, buyer);
```

```
car contract:

 query(car):
  get(car);
  return car;

 transfer(car, buyer, seller):
  get(car);
  car.owner = buyer;
  put(car);
  return car;

 update(car, properties):
  get(car);
  car.colour = properties.colour;
  put(car);
  return car;
```

Buyer Organization

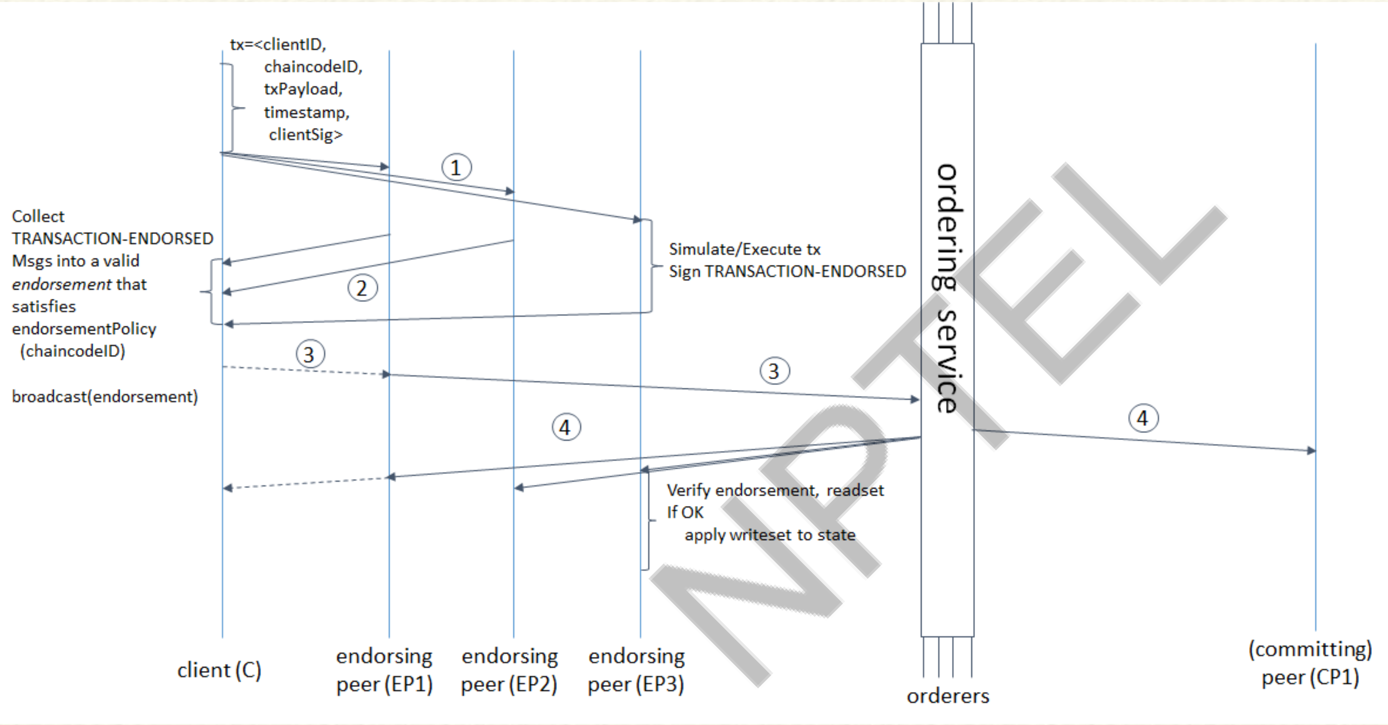
ORG2

```
application:

seller = ORG2;
buyer = ORG1;
transfer(CAR2, seller, buyer);
```

https://hyperledger-fabric.readthedocs.io/en/release-2.2/smartcontract/smartcontract.html

Fabric Transaction Flow



https://hyperledger-fabric.readthedocs.io/en/release-2.2/txflow.html

Writing Fabric Chaincode

- Written in Go, Node.js, or Java.

- Runs in a separate process from the peer.
  - separate container

- **fabric-contract-api** of Fabric SDK
  - contract interface, a high level API for implementing Chaincodes

  - Go: https://pkg.go.dev/github.com/hyperledger/fabric-contract-api-go/contractapi
  - Node.js: https://hyperledger.github.io/fabric-chaincode-node/release-2.2/api/
  - Java: https://hyperledger.github.io/fabric-chaincode-java/release-2.2/api/org/hyperledger/fabric/contract/package-summary.html

Define SmartContract

```go
package main

import (
"fmt"
"github.com/hyperledger/fabric-contract-api-go/contractapi"
)


// SmartContract - provides functions for storing and
// retrieving keys and values from the world state
//
type SmartContract struct {
contractapi.Contract
}
```

InitLedger

```
// InitLedger (optional in recent versions of fabric)
func (s *SmartContract) InitLedger(ctx contractapi.TransactionContextInterface) error {
err := ctx.GetStub().PutState("testkey", []byte("testval"))

if err != nil {
return fmt.Errorf("Failed to put to world state. %s", err.Error())
}

return nil
}
```

GetState and PutState

```
// CreateKey
func (s *SmartContract) CreateKey(ctx contractapi.TransactionContextInterface,  key string, val string) error
{
return ctx.GetStub().PutState(key,  []byte(val))
}

// QueryKey
func (s *SmartContract) QueryKey(ctx contractapi.TransactionContextInterface,  key string) (string,error) {
val, err := ctx.GetStub().GetState(key)

if err != nil {
return "", fmt.Errorf("Failed to get from world state. %s", err.Error())
}

return string(val), nil
}
```

Start Chaincode

```
func main(){
chaincode, err := contractapi.NewChaincode(new(SmartContract))

if err != nil {
fmt.Printf("Error creating chaincode: %s", err.Error())
return
}

err = chaincode.Start();

if err != nil {
fmt.Printf("Error starting chaincode: %s", err.Error())
}

}
```

Conclusion

- Fabric Chaincodes in general purpose programming languages.
- Key-Value pairs in World State
- Any business logic


Thank you

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Bishakh Chandra Ghosh

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 37: Hyperledger Fabric 4

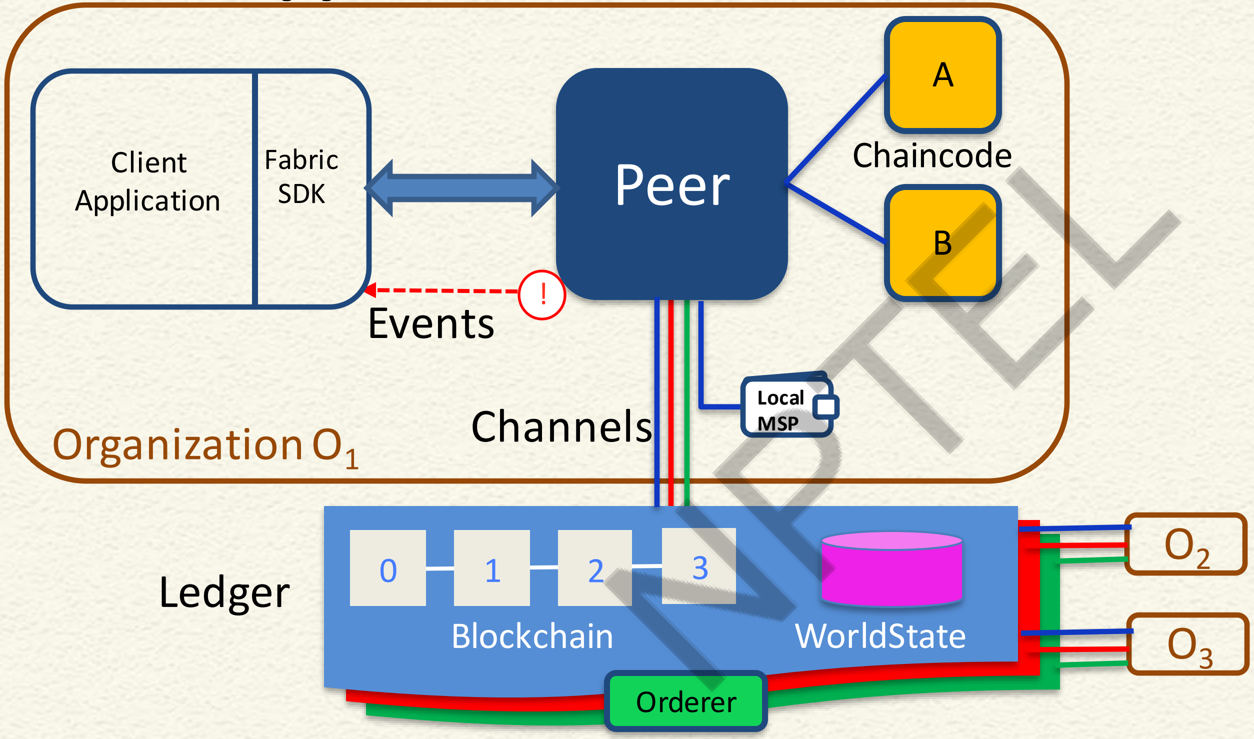
CONCEPTS COVERED

- **Hyperledger Fabric Application**
- **Fabric CAs**
- **Writing your own DAPP with Fabric**

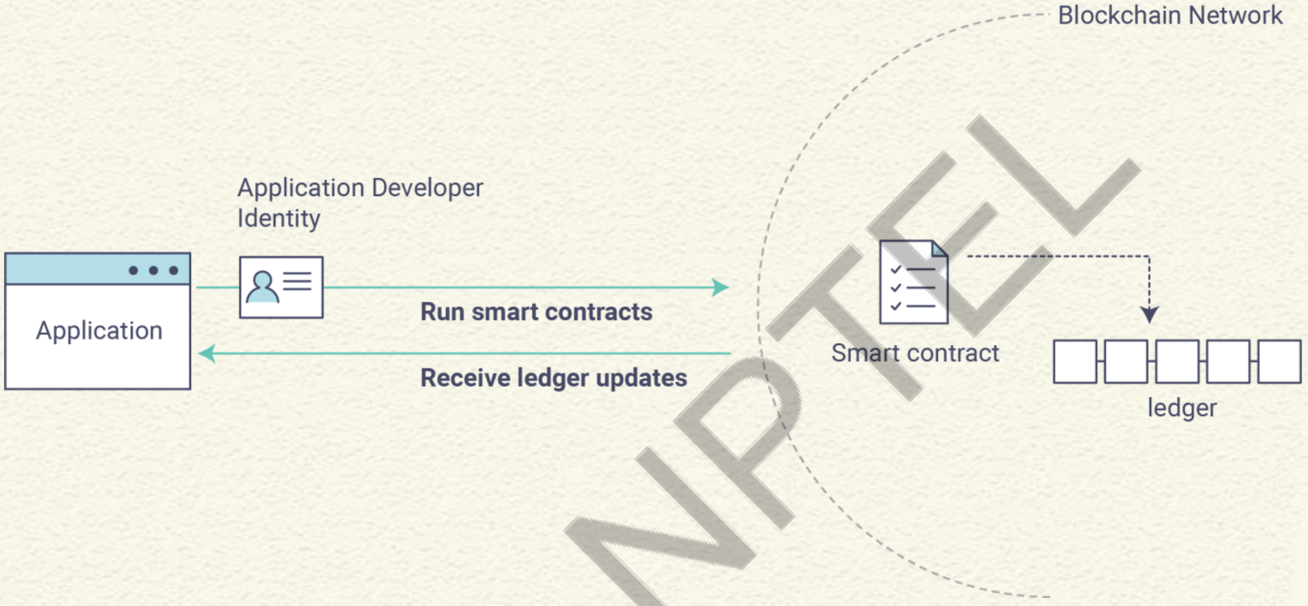
KEYWORDS

- Fabric Application
- Fabric CA
- DAPP

Fabric Application



Fabric Application



https://hyperledger-fabric.readthedocs.io/en/release-2.2/write_first_app.html

Prerequisites

- Fabric client applications can be developed in:
  - Node.js
  - Java
  - Go
  - Python

https://hyperledger-fabric.readthedocs.io/en/release-2.2/getting_started.html#hyperledger-fabric-application-sdks

Imports

```
const FabricCAServices = require('fabric-ca-client')
const {Wallets, Gateway} = require('fabric-network')

const fs = require('fs')
const path = require('path')


async function main(){
....
}

main()
```

https://hyperledger-fabric.readthedocs.io/en/release-2.2/getting_started.html#hyperledger-fabric-application-sdks

Connection Profile and CA

```
// Org1 connection profile
const ccpPath = path.resolve('../organizations/peerOrganizations/org1.example.com/connection-org1.json')
const ccp = JSON.parse(fs.readFileSync(ccpPath, 'utf8'))

// Org1 Ca
const caInfo = ccp.certificateAuthorities['ca.org1.example.com']
const caTLSCACerts = caInfo.tlsCACerts.pem
const ca = new FabricCAServices(caInfo.url, { trustedRoots: caTLSCACerts, verify: false }, caInfo.caName)
```

Connection Profile and CA

```
// Org1 connection  profile
const ccpPath  = path.resolve('../organizations/peerOrganizations/org1.example.com/connection-org1.json')
const ccp  = JSON.parse(fs.readFileSync(ccpPath, 'utf8'))

// Org1 Ca
const caInfo  = ccp.certificateAuthorities['ca.org1.example.com']
const caTLSCACerts  = caInfo.tlsCACerts.pem
const ca  = new FabricCAServices(caInfo.url, { trustedRoots: caTLSCACerts, verify: false }, caInfo.caName)
```

Configure CA admin

```
// Get admin identity

const enrollment = await ca.enroll({ enrollmentID: 'admin', enrollmentSecret: 'adminpw' });

const x509Identity  = {
credentials: {
certificate:  enrollment.certificate,
privateKey: enrollment.key.toBytes(),
},
mspId: 'Org1MSP',
type: 'X.509',
};

await wallet.put("admin",  x509Identity)

console.log("Admin  enrolled and saved into wallet successfully")

adminIdentity  =  await wallet.get("admin")
```

Register User

```
// Register user for this app
const provider = wallet.getProviderRegistry().getProvider(adminIdentity.type);
const adminUser = await provider.getUserContext(adminIdentity, 'admin');

const secret = await ca.register({affiliation: 'org1.department1', enrollmentID: 'appUser', role: 'client'},
adminUser);

const enrollment = await ca.enroll({enrollmentID: 'appUser',enrollmentSecret: secret});

const x509Identity = {credentials: {certificate: enrollment.certificate,privateKey: enrollment.key.toBytes()},
mspId: 'Org1MSP',
type: 'X.509',
};

await wallet.put('appUser', x509Identity)
console.log("Enrolled appUser and saved to wallet")

userIdentity = await wallet.get("appUser")
```

Configure Channel and Chaincode

```
// Connect to gateway
const gateway = new Gateway();

await gateway.connect(ccp, {wallet, identity:'appUser', discovery: {enabled: true, asLocalhost: true}})

// connect to channel
const network = await gateway.getNetwork('mychannel')

// select the contract
const contract = network.getContract("keyvaluechaincode")
```

Query and Invoke Chaincodes

```
// Query and Invoke transactions

var result = await contract.evaluateTransaction("QueryKey", "nptel")
console.log("First query:", result.toString())

await contract.submitTransaction("CreateKey", "nptel", "a new value")

var result = await contract.evaluateTransaction("QueryKey", "nptel")
console.log("Second query:", result.toString())

// disconnect
await gateway.disconnect()
```

Conclusion

- Fabric Client SDK
- Fabric Identities and CA
- Query and Invoke Chaincodes


Thank you

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 38: Consensus Scalability

CONCEPTS COVERED

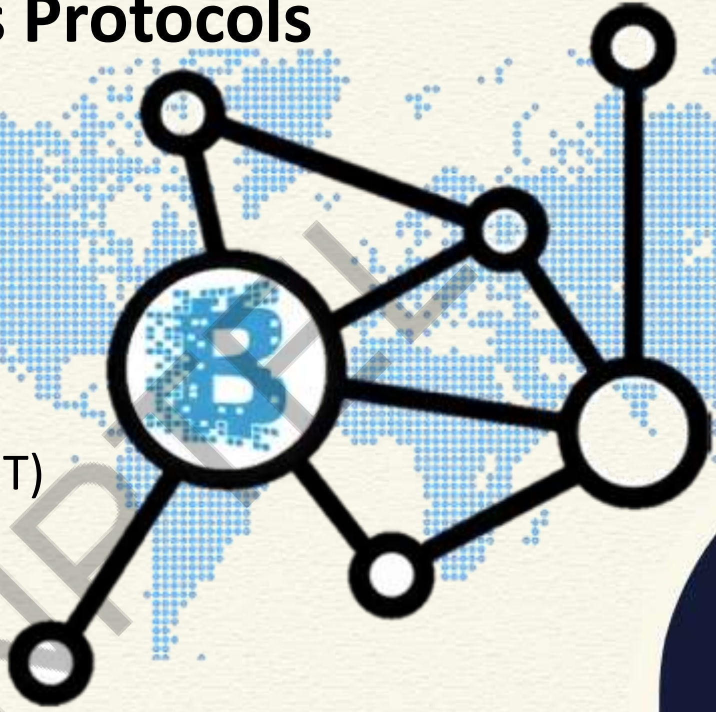
- **Blockchain Scalability**

KEYWORDS

- **PoW vs PBFT**
- **Scalability**
- **Consensus Finality**

Blockchain Consensus Protocols

- **Permissionless Blockchain**
    - Proof of Work (PoW)
    - Proof of State (PoS)
    - Proof of Burn (PoB)
    - Proof of Elapsed Time (PoET)



Blockchain Consensus Protocols

- **Permissioned Blockchain**
  - Byzantine Agreement
  - PBFT

PoW vs PBFT

- PoW
  - Open environment, works over a large number of nodes
  - Scalable in terms of number of nodes
  - Transaction throughput is low
- PBFT
  - Closed, not scalable in terms of number of nodes
  - High transaction throughput

PoW Scalability

- Two magic numbers in PoW
  - **Block frequency** - 10 minutes
  - **Block size** - 1 MB / 8MB

- For Bitcoin:
  - Let's assume, block size = 1 MB.
  - Average transaction size = 380.04 bytes
  - Number of transactions per block = 1048576/380.04
    = 2,759.12

PoW Scalability

- Two magic numbers in PoW
  - **Block frequency** - 10 minutes
  - **Block size** - 1 MB / 8MB

- For Bitcoin:
  - With 10 minutes (600 seconds) as block mining time,
    - 2759.12 transactions in 600 seconds
    - 4.6 transactions per second

PoW Scalability

- Two magic numbers in PoW
  - **Block frequency** - 10 minutes
  - **Block size** - 1 MB / 8MB

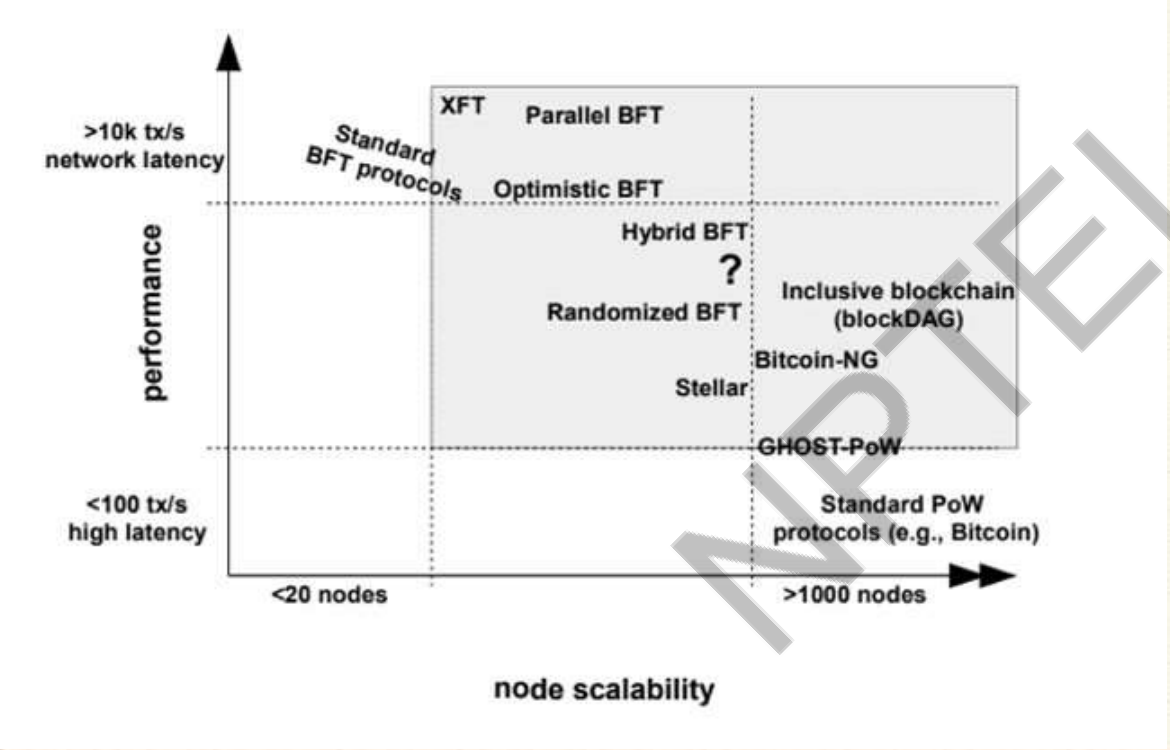
- Bitcoin Transaction throughput – 4.6 transactions per second
  - Visa supports around 1736 transactions per second

Tuning Bitcoin PoW Scalability

Scenario #		S0 The current Bitcoin Scenario	S1 Increasing Block Size to 377.5MB	S2 Increase Only Block Generation Time to 1.5s	S3 TB = TR	S4 TB scaled by same factor as Block Size Increase
	Adjustment	Default	B = 377.5	TB = 1.6s	TR = 14s	B = 2MB
A	Bitcoin Block Size (B) in Bytes	1,048,576	395,808,000	1,048,576	1,048,576	2,097,152
B	Block Generation Time (TB) in Seconds	600	600	1.589522193	14	28
C	Average Transaction (Tx) Size in Bytes	380	380	380	380	381
D	Average Transactions per Block = A/C	2,759.41	1,041,600.00	2,759.41	2,759.41	5,504.34
E	Blockchain Transactions per Second (TPS) = D/B	4.6	1736.0	1736.0	197.1	196.6

https://towardsdatascience.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44

Performance vs Scalability



Vukolić, Marko. "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication." ***International Workshop on Open Problems in Network Security*****. Springer, Cham, 2015.**

PoW vs PBFT – Consensus Finality

- *If a correct node p appends block b to its copy of blockchain before appending block b', then no correct node q appends block b' before b to its copy of the blockchain* (Vukolic, 2015)

- PoW is a randomized protocol - does not ensure consensus finality
    - Remember the forks in Bitcoin blockchain

- BFT protocols ensure total ordering of transactions
    - Ensures consensus finality

PoW Consensus vs BFT Consensus

	PoW consensus	BFT consensus
Node identity management	**open, entirely decentralized**	permissioned, nodes need to know IDs of all other nodes
Consensus finality	no	**yes**
Scalability (no. of nodes)	**excellent (thousands of nodes)**	limited, not well explored (tested only up to $n \leq 20$ nodes)
Scalability (no. of clients)	**excellent (thousands of clients)**	**excellent (thousands of clients)**
Performance (throughput)	limited (due to possible of chain forks)	**excellent (tens of thousands tx/sec)**
Performance (latency)	high latency (due to multi-block confirmations)	**excellent (matches network latency)**
Power consumption	very poor (PoW wastes energy)	**good**
Tolerated power of an adversary	$\leq 25\%$ computing power	$\leq 33\%$ voting power
Network synchrony assumptions	physical clock timestamps (e.g., for block validity)	**none for consensus safety** (synchrony needed for liveness)
Correctness proofs	no	**yes**

Vukolić, Marko. "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication." ***International Workshop on Open Problems in Network Security*****. Springer, Cham, 2015.**

Conclusion

- Scalability is a major issue in Blockchain consensus
- In the next lecture, we'll discuss different scalable blockchain protocols


Thank you

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 39: Bitcoin-NG

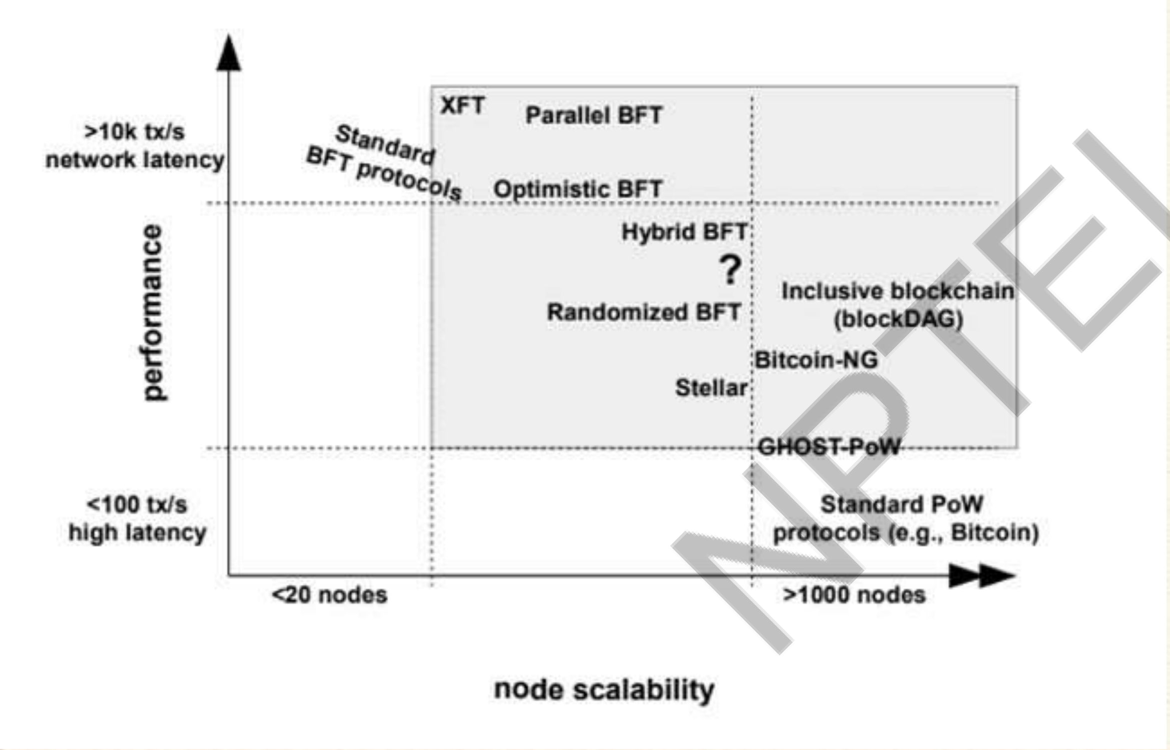
CONCEPTS COVERED

- Issues with Bitcoin – Revisit
- Bitcoin-NG

KEYWORDS

- **Transaction Serializability**
- **Key-blocks and Microblocks**

Performance vs Scalability



Vukolić, Marko. "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication." *International Workshop on Open Problems in Network Security*. Springer, Cham, 2015.

Towards a Scalable Consensus

Bitcoin-NG

Eyal, I., Gencer, A. E., Sirer, E. G., & Van Renesse, R. (2016, March). **Bitcoin-NG: A Scalable Blockchain Protocol**. in *NSDI 2016*

Issues with Nakamoto Consensus

- **Transaction scalability**
  - Block frequency of 10 minutes and block size of 1 MB during mining reduces the transactions supported per second

Issues with Nakamoto Consensus

- **Transaction scalability**
  - Block frequency of 10 minutes and block size of 1 MB during mining reduces the transactions supported per second

- **Issues with Forks**
  - Prevents consensus finality
  - Makes the system unfair - a miner with poor connectivity has always in a disadvantageous position

Bitcoin-NG: Decouple Leader Election

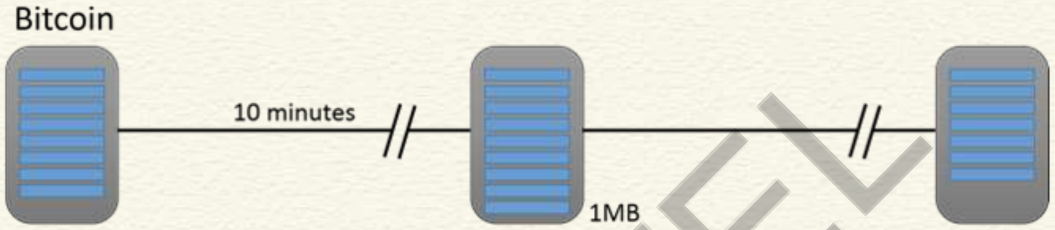
- Bitcoin - think of the winning miner as the **leader** - the leader serializes the transactions and include a new block in the blockchain

Bitcoin-NG: Decouple Leader Election

- Bitcoin - think of the winning miner as the **leader** - the leader serializes the transactions and include a new block in the blockchain

- Decouple Bitcoin’s blockchain operations into two planes
    - **Leader election**: Use PoW to randomly select a leader (an infrequent operation)
    - **Transaction Serialization**: The leader serializes the transaction until a new leader is elected

Bitcoin vs Bitcoin-NG



Bitcoin vs Bitcoin-NG

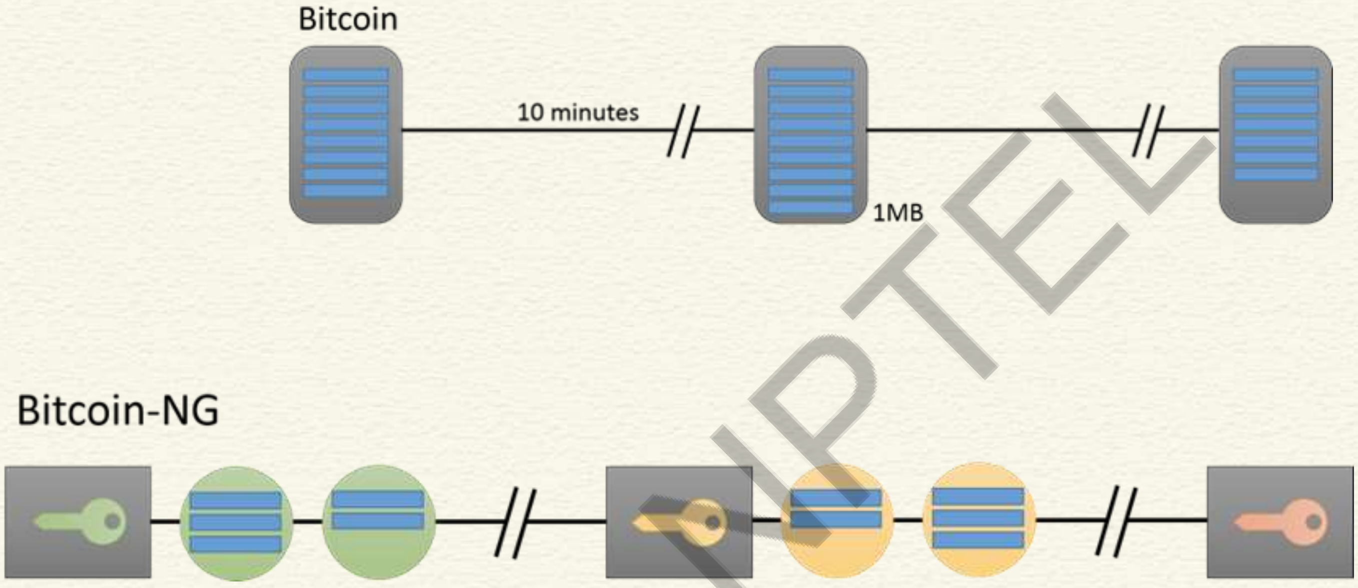


Image Source: http://hackingdistributed.com/2015/10/14/bitcoin-ng/

Bitcoin-NG: Key Blocks

- Key blocks are used to choose a leader (similar to Bitcoin)

- A key block contains
  - The reference to the previous block
  - The current Unix time
  - A coinbase transaction to pay of the reward
  - A target hash value
  - A nonce field

Key Blocks

- Key blocks are generated based on regular Bitcoin mining procedure
  - Find out the nonce such that the block hash is less than the target value

- Key blocks are generated infrequently - the intervals between two key blocks is exponentially distributed

Bitcoin-NG: Microblocks

- Once a node generates a key block, it becomes the **leader** and generates further microblocks
  - Microblocks are generates at a set rate smaller than a predefined maximum
  - The rate is much higher than the key block generation rate

Bitcoin-NG: Microblocks

- A microblock contains
  - Ledger entries
  - Header
    - Reference to the previous block
    - The current Unix time
    - A cryptographic hash of the ledger entries (Markle root)
    - A cryptographic signature of the header (signature of the key block miner)

Bitcoin-NG

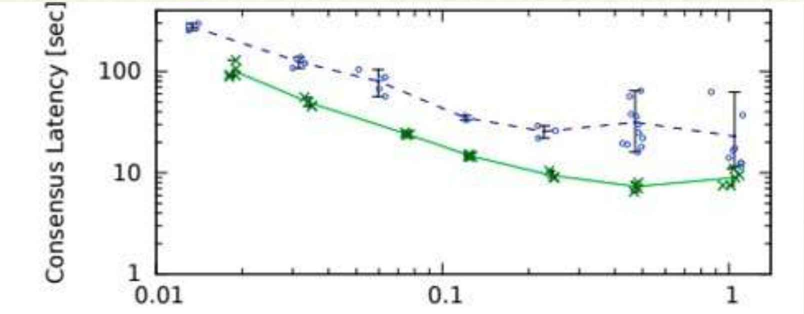
Microblock Fork

- When a miner generates a key block, he may not have heard of all microblocks generated by the previous leader
    - Common if microblock generation is frequent
    - May result in microblock fork

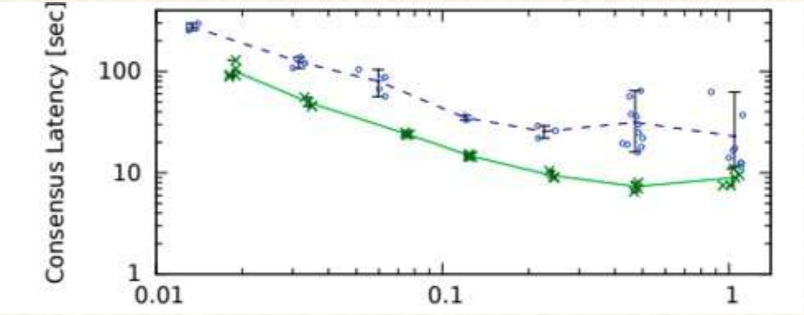
Microblock Fork

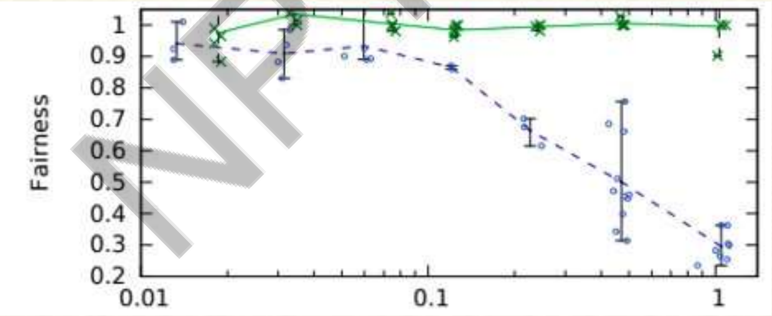
- When a miner generates a key block, he may not have heard of all microblocks generated by the previous leader
  - Common if microblock generation is frequent
  - May result in microblock fork
- A node may hear a forked microblock but not new key block
  - This can be prevented by ensuring the reception of the key block
  - When a node sees a microblock, it waits for propagation time of the network to make sure it is not pruned by a new key block

Bitcoin-NG Performance



Bitcoin-NG Performance



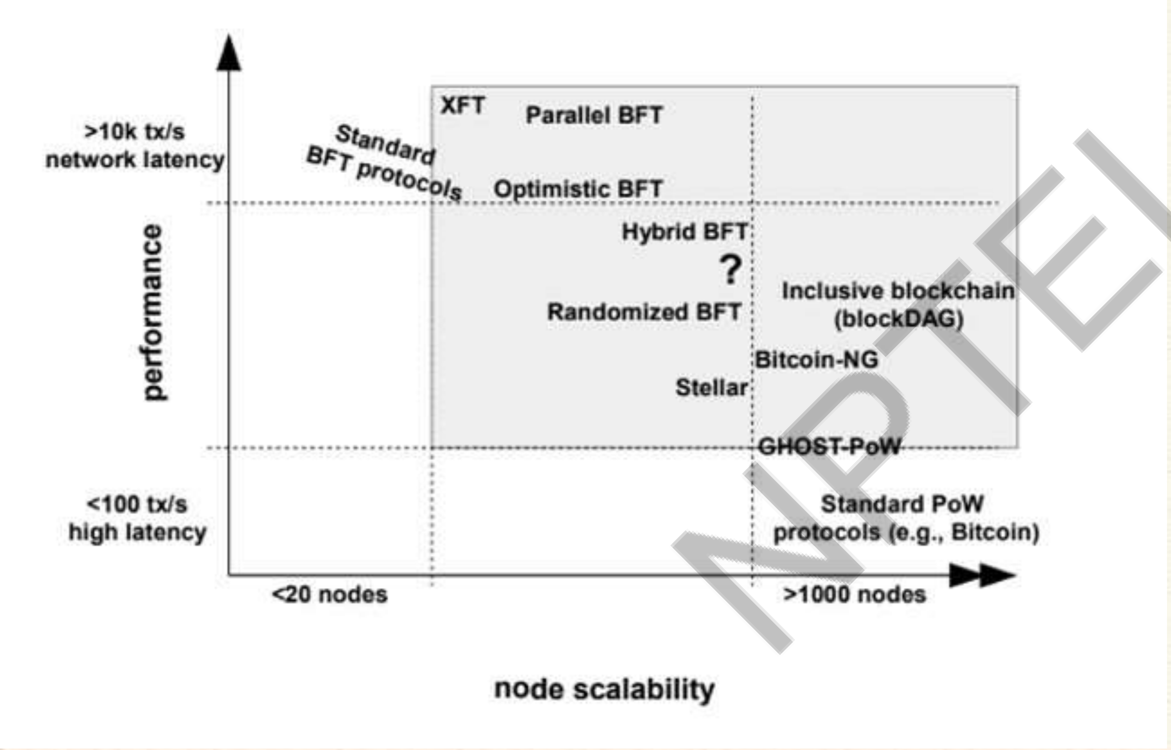


Conclusion

- A major source of latency in Bitcoin is that every block needs to be mined by different miners

- Bitcoin-NG decouples leader election from transaction serialization
    - Key blocks and Microblocks

Performance vs Scalability - Revisiting




Thank you

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 40: Collective Signing (CoSi)

CONCEPTS COVERED

- **Collective Signing**
- **Schnorr Multisignature**
- **PBFT as Collective Signing**

KEYWORDS

- CoSi
- Multisignature

Collective Signing

- Method to protect "authorities and their clients" from undetected misuse or exploits

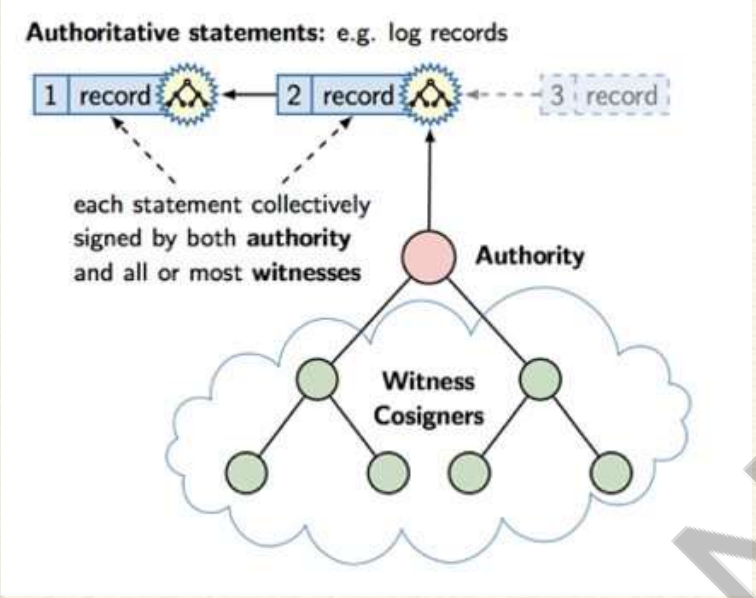
- A **scalable witness cosigning protocol** ensuring that every authoritative statement is validated and publicly logged by a diverse group of witnesses before any client accepts it

Syta, Ewa, *et al*. "**Keeping authorities "honest or bust" with decentralized witness cosigning**" *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

Collective Signing

- A statement *S* collectively signed by *W* witnesses assures clients that *S* has been seen, and not immediately found erroneous, by those *W* observers.

CoSi Architecture



- The leader organizes the witnesses in a tree structure – a scalable way of aggregating signatures coming from the children

- Three rounds of PBFT (pre-prepare, prepare and commit) can be simulated using two rounds of CoSi protocol

Schnorr Multisignature

- The basic CoSi protocol uses **Schnorr multisignatures**, that rely on a group *G* of prime order
    - *Discrete logarithmic problem is believed to be hard*

Schnorr Multisignature

- **Key Generation:**
  - Let G be a group of prime order *r*. Let *g* be a generator of G.
  - Select a random integer x in the interval $[0, r-1]$. x is the private key and g^x is the public key.
  - *N* signers with individual private keys $x_1, x_2, \dots, x_N$, and the corresponding public keys $g^{x_1}, g^{x_2}, \dots, g^{x_N}$

Schnorr Multisignature

- **Signing:**
  - Each signer picks up the random secret v_i, generates $V_i = g^{v_i}$
  - The leader collects all such V_i, aggregates them $V = \prod V_i$, and uses a hash function to compute a collective challenge $c = H(V||S)$. This challenge is forwarded to all the signers.
  - The signers send the response $r_i = v_i - cx_i$. The leader computes the aggregated as $r = \sum r_i$. The signature is (c, r).

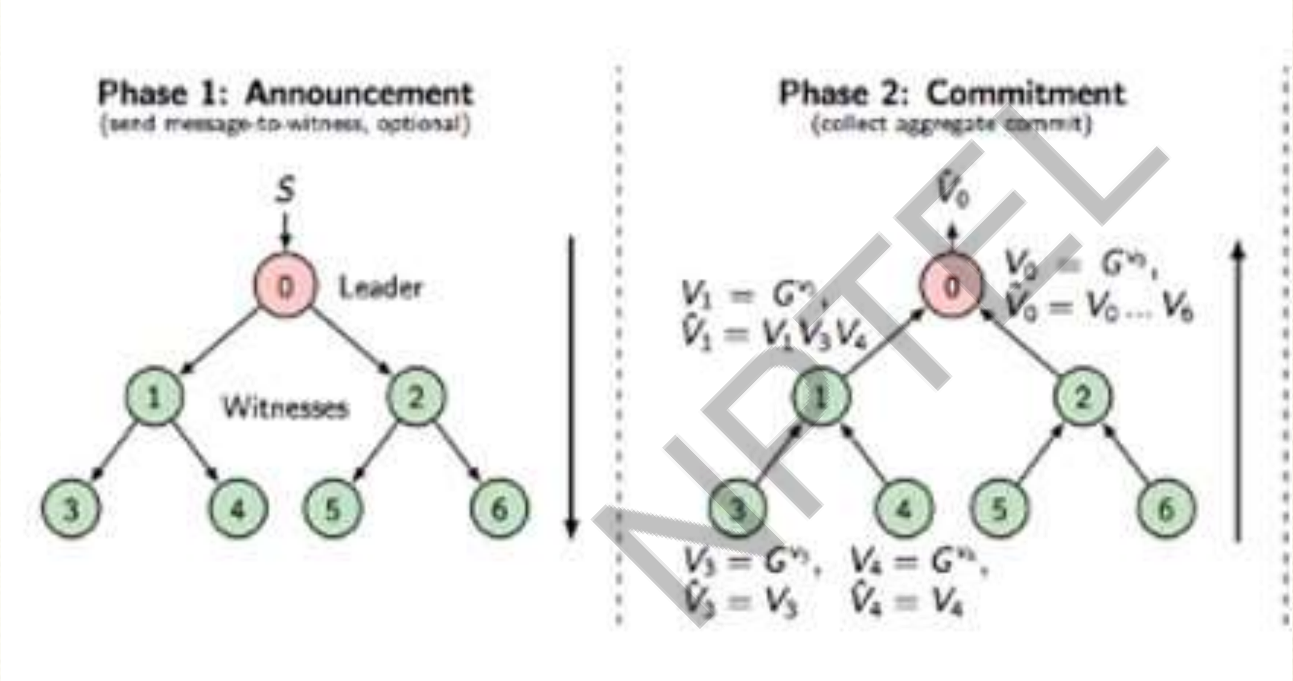
Schnorr Multisignature

- **Verification:**
  - The verification key is $y = \prod g^{x_i}$
  - The signature is (c, r), where $c = H(V||S)$ and $r = \sum r_i$
  - Let $V_v = g^r y^c$
  - Let $r_v = H(V_v||S)$
  - If $r_v = r$, then the signature is verified

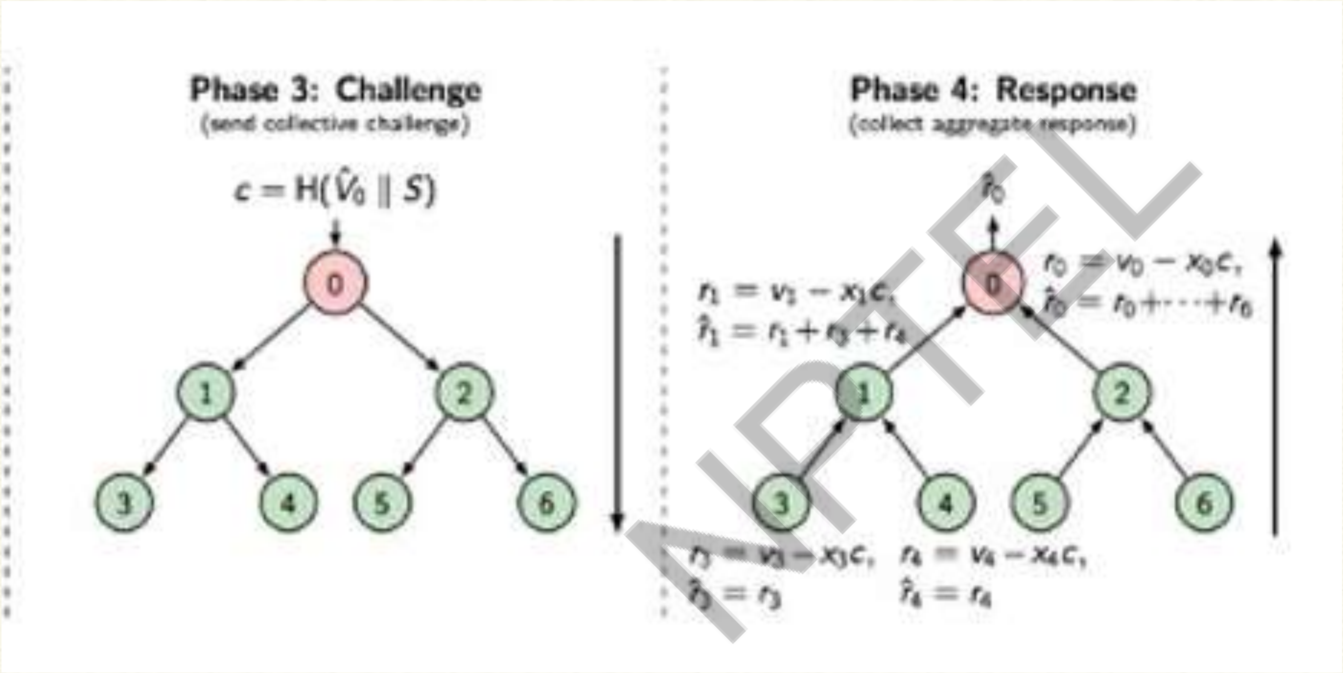
Schnorr Multisignature

- **Proof:**
  - The verification key is $y = \prod g^{x_i}$
  - The signature is (c, r), where $c = H(V||S)$ and $r = \sum r_i$
  - $V_v = g^r y^c = g^{\sum(v_i - cx_i)} \prod g^{cx_i} = g^{\sum(v_i - cx_i)} g^{\sum cx_i} = g^{\sum v_i} = \prod g^{v_i} = \prod V_i = V$
  - So, $r_v = H(V_v||S) = H(V||S) = r$

CoSi Protocol



CoSi Protocol



CoSi Protocol

- One CoSi round to implement PBFT's pre-prepare and prepare phases
- Second CoSi round to implement PBFT's commit phase
- Other multisignature methods are available
  - Boneh-Lynn-Shacham (BLS) Cryptography – uses Bilinear Pairing

Conclusion

- CoSi can be used to sign a message by multiple authorities collectively
  - Verification is easy – from the collective public key
- PBFT can be emulated using two rounds of CoSi
- Next, we'll see how CoSi can be used to design a scalable blockchain consensus


Thank you