CS 168: Blockchain and Cryptocurrencies

Pure Proof-of-Stake Protocols: Ouroboros, Algorand, and Tendermint

Prof. Tom Austin
San José State University

Review: Proof-of-Stake (PoS)

- **Core idea**: people invested in currency won't destroy it.
- “Virtual miners” (sometimes called *validators*)
- Scarce resource: coins
  - Many different forms

Rough Breakdown of Proof-of-Stake Approaches

- Proto-PoS
  - Hybrid PoW/PoS
  - PoW protocols that leverage PoS concepts
- Follow-the-satoshi (FTS) protocols
- Byzantine Fault Tolerant approaches

Rough Breakdown of Proof-of-Stake Approaches

- Proto-PoS
  - Hybrid PoW/PoS
  - PoW protocols that leverage PoS concepts
- **Follow-the-satoshi (FTS) protocols**
- Byzantine Fault Tolerant approaches

Envisioning FTS as Roulette Wheel

Lab, part 1: Implement FTS algorithm

- Implement algorithm in SpartanGold:
  - Create **`rouletteWheel`** array
  - For every gold, put owner's address in slot on wheel
  - "Randomly" select owner from **`rouletteWheel`**
- FTS algorithm selects block producer
- "Randomness" derived from previous block hash
  - ***NOT A GOOD CHOICE!***

Ouroboros

Ouroboros Overview

- Follow-the-satoshi (FTS) algorithm
  - Large # participants
  - Probabilistic finality
- Goals:
  - Security of Bitcoin
  - Improve scalability
  - Reduce energy consumption
- Used by Cardano
- Lots of formal guarantees for security!

Overview of Ouroboros's design

- Every *epoch* has a committee
  - conduct 3-phase coin-tossing protocol as FTS "randomness" seed
    - relies on *publicly verifiable secret sharing*
  - epoch lasts for `k` *slots*
    - one slot ***might*** produce one block
    - slot is "empty" if no block produced
- FTS algorithm used to:
  - select "slot leader" to produce block
  - elect committee members for next epoch

Properties

- Persistence
  - If an honest node claims a transaction is ***stable*** (accepted), then ***all*** honest nodes report it as stable.
- Liveness
  - After n steps, an honest transaction will become stable.
- Properties guaranteed under certain assumptions

Grinding Attacks

- Miner might bias FTS by repeated computation
  - Similar to PoW algorithm
- Ouroboros proposed alternatives:
  - Fixed stake at genesis
    - Leaders known in advance
    - Attacker cannot bias
  - Centralized random beacon
  - Simulate random beacon with committee

Grinding Attacks

- Miner might bias FTS by repeated computation
  - Similar to PoW algorithm
- Ouroboros proposed alternatives:
  - Fixed stake at genesis
    - Leaders known in advance
    - Attacker cannot bias
  - Centralized random beacon
  - ***Simulate random beacon with committee***

Lab, part 2: perform grinding attack

- Hash determines next miner, so…
- Maleficent changes block to change hash.
  - Once she gets a turn, she takes control of the blockchain.

Rough Breakdown of Proof-of-Stake Approaches

- Proto-PoS
  - Hybrid PoW/PoS
  - PoW protocols that leverage PoS concepts
- Follow-the-satoshi (FTS) protocols
- **Byzantine Fault Tolerant approaches**

BFT Proof-of-Stake Protocols

- Use classic BFT approaches
- Advantages:
  - Consensus on every block
- Drawbacks:
  - Lots of messages required
  - Don’t scale to large # of miners
- Notable protocols:
  - Tendermint *(first pure PoS protocol)*
  - Algorand

Algorand

- Goals:
  - Scale to large number of miners
  - DDoS protection
- Relies on cryptographic sortition
  - Miners run local computation to determine if they are a committee member
  - Broadcast:
    - cryptographic proof of committee membership
    - work needed: block or vote

Algorand DDoS Protection

- After miner broadcasts message, its work is complete
- Attacker does not know who is participating in advance
- After the message, DDoS-ing the miner has no effect on the protocol

Verifiable Random Functions (VRFs)

- Related to public-key crypto
- Formulas:
  - $Evaluate_{sk}(input) = (output, proof)$
  - $Verify_{pk}(input, output, proof) = 0/1$
- Allows clients to locally compute random, publicly verifiable values
- WARNING: bleeding edge crypto

Tendermint

Tendermint Overview

- Early proof-of-stake (PoS) blockchain
- Influential in future PoS designs
- Developed by Jae Kwon

Tendermint Timeline

- 2014 – Jae Kwon released initial whitepaper
- August 2016, released Tendermint v. 0.7.0
  - First stable release
- Later expanded into Cosmos
  - "Internet of blockchains"
  - Launched March 2019

Tendermint

- Pure proof-of-stake
  - Clients *bond* coins for right to produce blocks.
- Deterministic
- Achieves consensus every block
- Liveness
- Lots of messages required

Tendermint Terminology

- **Validators** take the place of miners
- **Proposer**: validator who chooses the current block
- **Nothing-at-stake problem**: validator backs multiple proposals

Dwork, Lynch, and Stockmeyer Round Model (1988)

- Goal: achieve consensus after a given number of ***rounds***.
- Properties
  - ***Safety***: no 2 correct processes reach disagreement
  - ***Termination***: each correct process (eventually) makes a decision
- Contains increasingly complex models to handle different faults

Possible faults

- Fail-stop
  - Process crashes
- Omission
  - Process fails to send or receive some messages
- Byzantine
  - Erroneous messages
  - (Could be accidental or intentional)

Tendermint's design

- Clients **bond** coins to join validator set.
- Validators
  - Propose blocks
  - Validate proposed blocks
  - Share in block rewards
- Validators **unbond** coins to reclaim funds.
  - Exit validator set
  - Coins not released immediately
- All validators who participate in validating a block share rewards

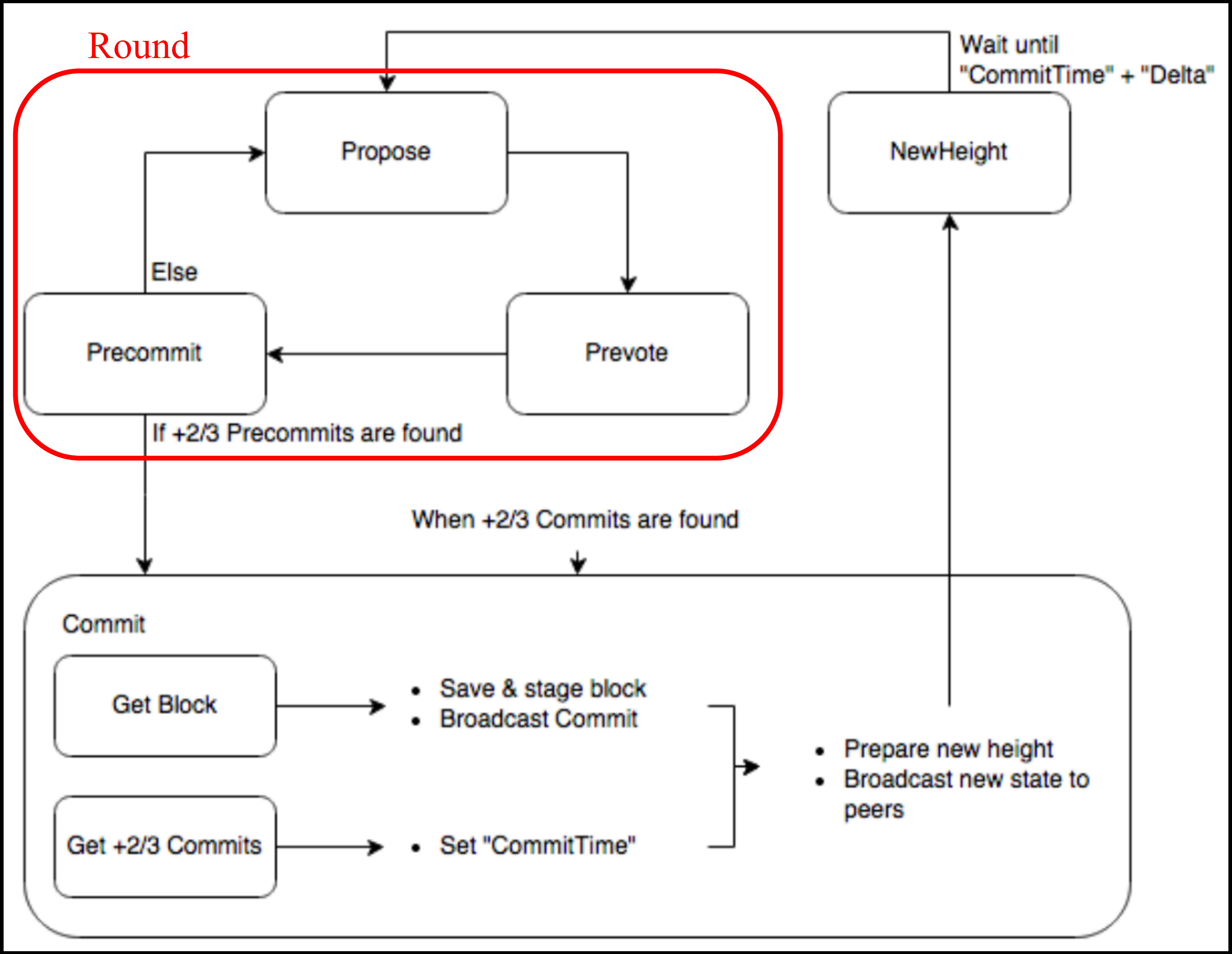
Tendermint Transaction Types

- Send – Transfer coins to other clients
- Bond
  - Stakes coins as surety bond
  - Adds client to validator set
- Unbond
  - Releases tokens after a delay
  - Removes client from validator set
- Evidence
  - Provides proof of Byzantine behavior
  - Byzantine validator will lose some bonded coins
  - Byzantine validator *may* be ejected from validator set.

Bonded coins determines:

- Frequency of being block proposer
- Weight of vote
  - "Voting power"
- Share of rewards

Block Generation Process



One round

1. Propose
   - Determine block proposer
   - Share block (if proposer)
2. Prevote
   - Try to lock on block
3. Precommit
   - Try to commit to locked block

Each step waits $T + R\delta$ time before next step (where R is the round number)

Proposal Step

- All validators determine proposer.
  - Weighted round-robin algorithm.
- Elected block proposer generates a block.
- Determined by *accumulated power*.

Accumulated Power

- Used by weighted round-robin algorithm
- Initial accumulated power = # of coins bonded (`Validator.Power`)
- Increases by # bonded coins each block
- Decreases when a proposer shares a block
- Similar process used for each *round* of block production

Proposer Election Pseudocode
(taken from Tendermint whitepaper v. 0.5)

```
// Copy AccumPower over to RoundAccumPower
for each Validator:
  Validator.RoundAccumPower = Validator.AccumPower

function getNextProposer():
  TotalIncremented := 0

  // Increment voting power
  for each Validator:
    Validator.RoundAccumPower += Validator.Power
    TotalIncremented += Validator.Power

  Proposer := Validator with most RoundAccumPower
  Proposer.RoundAccumPower -= TotalIncremented
  return Proposer
```

Prevote step

After proposals collected, validator prevotes:

1. If locked on a block, vote for it.
2. Otherwise if a valid proposal was received, vote for it.
3. Otherwise, vote NIL.

Precommit step

After collecting prevotes:

1. If a block earns 2/3 prevotes:
   - Lock on a block
   - Broadcast precommit vote for block
2. If NIL earns 2/3 prevotes:
   - Release any locks
3. Otherwise, do nothing

Commit decision

After collecting precommits:

- If 2/3 of precommits received for a block:
  - Get the block (if not available locally)
  - Broadcast commit vote
- Otherwise, start a new round.
  - DELTA value increases

New Height

After collecting commits:

- If 2/3 commits received for a block:
  - Begin working on a new block
- Otherwise, keep waiting
  - A commit vote counts as prevotes and precommits for all subsequent rounds.

Byzantine Validators & Evidence Transactions

- Validators announce detected Byzantine behavior through *evidence transactions*
  - Byzantine validator loses (some) bonded coins
  - Other validators divide up coins
- Byzantine behaviors:
  - Proposing conflicting blocks for same height/round
  - Voting for conflicting blocks for same height/round
  - Voting for block and NIL for same height/round

Challenges with Tendermint

- Denial-of-service attacks
- Large number of messages
  - Forces us to keep the validator set small

HW 3: Implement Tendermint-like Blockchain in SpartanGold

(*demonstration*)