Lecture #22: Proof-of-Stake

COMS 4995-001:
The Science of Blockchains

URL: https://timroughgarden.org/s25/

Tim Roughgarden

Goals for Lecture #22

1. Proof-of-stake: the high level idea.
   – sample validator with probability proportional to amount of locked-up stake
2. Proof-of-stake: pros and cons.
   – why isn’t proof-of-work good enough?
3. Mechanisms of staking.
   – warm-up and cool-down periods, delegation, etc.
4. Why proof-of-stake is hard.
   – lack of external randomness; quick + dirty solution: weighted round robin

Proof-of-Stake: The High-Level Idea

Idea: validators "lock up" "stake."

- generally used in protocols with general-purpose smart contracts (to implement escrow contract) and a native currency (to stake with)

Desired property:

Proof-of-Stake: The High-Level Idea

Idea: validators “lock up” “stake.”

- generally used in protocols with general-purpose smart contracts (to implement escrow contract) and a native currency (to stake with)

Desired property: for every validator i,

Pr[validator i selected] = fraction of staked coins owned by i (*)

Proof-of-Stake: The High-Level Idea

Idea: validators “lock up” “stake.”

- generally used in protocols with general-purpose smart contracts (to implement escrow contract) and a native currency (to stake with)

Desired property: for every validator i,

Pr[validator i selected] = fraction of staked coins owned by i (*)

- e.g., as the leader of a view
- implies sybil-proofness (note (*) independent of # of public keys used)

Proof-of-Stake: The High-Level Idea

Idea: validators “lock up” “stake.”

- generally used in protocols with general-purpose smart contracts (to implement escrow contract) and a native currency (to stake with)

Desired property: for every validator i,

Pr[validator i selected] = fraction of staked coins owned by i (*)

- e.g., as the leader of a view
- implies sybil-proofness (note (*) independent of # of public keys used)

Fact: proof-of-stake has become the dominant approach to sybil-resistance over past 5+ years.

Proof-of-Stake: Pros

Pro #1: minimal energy consumption.

Proof-of-Stake: Pros

Pro #1: minimal energy consumption.

- proof-of-work: validators must prove their hashrate to protocol
  - hashrate unobservable by protocol (off-chain)
  - e.g., estimated that Bitcoin miners use .5% of world's energy
  - obvious critique on environmental grounds (hotly debated)
  - may be strong impediment to launching new PoW protocols

Proof-of-Stake: Pros

Pro #1: minimal energy consumption.

- proof-of-work: validators must prove their hashrate to protocol
  - hashrate unobservable by protocol (off-chain)
  - e.g., estimated that Bitcoin miners use .5% of world's energy
  - obvious critique on environmental grounds (hotly debated)
  - may be strong impediment to launching new PoW protocols
- proof-of-stake: validators' stake directly observable by protocol
  - energy consumption comparable to a typical Internet protocol

Proof-of-Stake: Pros (con’d)

Pro #2: stronger latency/finality guarantees.

- likely the most powerful force pushing migration toward proof-of-stake

Proof-of-Stake: Pros (con'd)

Pro #2: stronger latency/finality guarantees.

- likely the most powerful force pushing migration toward proof-of-stake

- proof-of-work: more or less forced into Nakamoto consensus
  - no finality in partial synchrony (unavoidable for PoW)
  - even in synchrony, latency is high due to security parameter k
  - even in synchrony, finality is only probabilistic (unavoidable for PoW)
  - PoW alternatives with lower latency exist, but not in production

Proof-of-Stake: Pros (con’d)

Pro #2: stronger latency/finality guarantees.

– likely the most powerful force pushing migration toward proof-of-stake

- proof-of-work: more or less forced into Nakamoto consensus
  – no finality in partial synchrony (unavoidable for PoW)
  – even in synchrony, latency is high due to security parameter k
  – even in synchrony, finality is only probabilistic (unavoidable for PoW)
  – PoW alternatives with lower latency exist, but not in production
- proof-of-stake: pairs well with e.g. Tendermint (as we’ll see)
  – finality as soon as assemble relevant quorum certificate, even in partially synchronous setting (assuming ≤ 33% faulty stake)

Proof-of-Stake: Pros (con’d)

Pro #3: recovery from 51%-type attacks/“slashing.”

- if 51% hashrate/34% stake is Byzantine, is protocol doomed?

Proof-of-Stake: Pros (con’d)

Pro #3: recovery from 51%-type attacks/“slashing.”

- if 51% hashrate/34% stake is Byzantine, is protocol doomed?

- proof-of-work: no obvious way to punish attacker
  - could “hard fork” to change the cryptographic hash function (nullifies attacker’s ASICs), but also punishes honest validators (“scorched earth”)

Proof-of-Stake: Pros (con'd)

Pro #3: recovery from 51%-type attacks/"slashing."

- if 51% hashrate/34% stake is Byzantine, is protocol doomed?

- proof-of-work: no obvious way to punish attacker
  - could "hard fork" to change the cryptographic hash function (nullifies attacker's ASICs), but also punishes honest validators ("scorched earth")
- proof-of-stake: can punish attacker by "slashing" their stake
  - e.g., slash any validators that are caught equivocating/double-voting
  - slashing could be programmatic or implemented via hard fork
  - for slashing, particularly convenient for stake to be in native currency

Proof-of-Stake: Cons

Con #1: additional complexity.

- all major proof-of-stake protocols significantly more complex than Nakamoto consensus
  - risk of bugs in design and/or implementation
  - even the simplest distributed protocols difficulty to get right

Proof-of-Stake: Cons

Con #1: additional complexity.

- all major proof-of-stake protocols significantly more complex than Nakamoto consensus
  - risk of bugs in design and/or implementation
  - even the simplest distributed protocols difficulty to get right
  - counterpoint: extra complexity necessary for extra functionality

Proof-of-Stake: Cons (con’d)

Con #2: additional attack vectors.

- inevitable consequence of additional complexity

Proof-of-Stake: Cons (con'd)

Con #2: additional attack vectors.

- inevitable consequence of additional complexity
- example ("long-range attack"): if validator's secret key is stolen, easy to fabricate signed blocks/votes allegedly from the past
  - e.g., in attempt to rewrite the past and break finality
  - with proof-of-work, need to actually do the work to produce valid blocks
    - "costly simulation"

Proof-of-Stake: Cons (con’d)

Con #2: additional attack vectors.

- inevitable consequence of additional complexity
- example (“long-range attack”): if validator’s secret key is stolen, easy to fabricate signed blocks/votes allegedly from the past
  - e.g., in attempt to rewrite the past and break finality
  - with proof-of-work, need to actually do the work to produce valid blocks
    - “costly simulation”

Upshot: all current proof-of-stake designs less battle-tested than Nakamoto consensus.

Proof-of-Stake: Cons (con’d)

Con #3: need for initial stake distribution.

Proof-of-Stake: Cons (con'd)

Con #3: need for initial stake distribution.

- Bitcoin: all BTC originate from block rewards
  - launched with the all-zero distribution!

Proof-of-Stake: Cons (con’d)

Con #3: need for initial stake distribution.

- Bitcoin: all BTC originate from block rewards
  - launched with the all-zero distribution!
  - many more recent PoW protocols do launch with a non-zero initial distribution (e.g., to team and investors)

Proof-of-Stake: Cons (con'd)

Con #3: need for initial stake distribution.

- Bitcoin: all BTC originate from block rewards
  - launched with the all-zero distribution!
  - many more recent PoW protocols do launch with a non-zero initial distribution (e.g., to team and investors)
- proof-of-stake: need initial currency distribution to get started
  - effectively pre-selecting the initial set of potential validators
    - has a more permissioned flavor

Proof-of-Stake: Cons (con'd)

Con #3: need for initial stake distribution.

- Bitcoin: all BTC originate from block rewards
  - launched with the all-zero distribution!
  - many more recent PoW protocols do launch with a non-zero initial distribution (e.g., to team and investors)
- proof-of-stake: need initial currency distribution to get started
  - effectively pre-selecting the initial set of potential validators
    - has a more permissioned flavor
  - various imperfect techniques for better decentralization of initial currency distribution (airdrops, secondary markets, etc.)

Proof-of-Stake: Cons (con’d)

Con #4: stronger trust assumptions in some scenarios.

Proof-of-Stake: Cons (con’d)

Con #4: stronger trust assumptions in some scenarios.

- example: spin up new validator/light client, need to sync with the chain of blocks/block headers produced thus far
  - from the genesis block, or from some more recent trusted checkpoint

Proof-of-Stake: Cons (con'd)

Con #4: stronger trust assumptions in some scenarios.

- example: spin up new validator/light client, need to sync with the chain of blocks/block headers produced thus far
  - from the genesis block, or from some more recent trusted checkpoint
- proof-of-work: can ask N sources, only need 1-in-N honest
  - resolve ambiguity by adopting the chain with the most supporting work

Proof-of-Stake: Cons (con'd)

Con #4: stronger trust assumptions in some scenarios.

- example: spin up new validator/light client, need to sync with the chain of blocks/block headers produced thus far
  - from the genesis block, or from some more recent trusted checkpoint
- proof-of-work: can ask N sources, only need 1-in-N honest
  - resolve ambiguity by adopting the chain with the most supporting work
- proof-of-stake: need a majority of sources to be honest
  - costless simulation (as in long-range attacks) ➔ can't automatically disambiguate competing valid chains

Proof-of-Stake: Cons (con’d)

Con #4: stronger trust assumptions in some scenarios.

- example: spin up new validator/light client, need to sync with the chain of blocks/block headers produced thus far
  - from the genesis block, or from some more recent trusted checkpoint
- proof-of-work: can ask N sources, only need 1-in-N honest
  - resolve ambiguity by adopting the chain with the most supporting work
- proof-of-stake: need a majority of sources to be honest
  - costless simulation (as in long-range attacks) ➔ can’t automatically disambiguate competing valid chains
  - in practice: use trusted source, or look for unanimity among 2-3 sources

Mechanics of Staking

- blockchain protocol maintains "staking contract"
  - native to protocol, analogous to a system program

Mechanics of Staking

- blockchain protocol maintains “staking contract”
  - native to protocol, analogous to a system program
- validators (identified by public key) lock up funds in this contract
  - generally, funds in protocol’s native currency
  - in some PoS chain, register your IP address (in addition to pk)
  - alternative: communicate via gossip network (see future lecture)

Mechanics of Staking

- blockchain protocol maintains “staking contract”
- validators (identified by public key) lock up funds in this contract

Design decisions:

1. Mininum/maximum staking duration. (e.g., days/weeks/months)
   – also: join/leave at any time, or only at prescribed points in time?

Mechanics of Staking

- blockchain protocol maintains “staking contract”
- validators (identified by public key) lock up funds in this contract

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount. (e.g., 0, or millions of USD)

Mechanics of Staking

- blockchain protocol maintains “staking contract”
- validators (identified by public key) lock up funds in this contract

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?

Mechanics of Staking

- blockchain protocol maintains “staking contract”
- validators (identified by public key) lock up funds in this contract

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?
   – cool down important for e.g. slashing
   – warm up important for e.g. some VRF designs

Mechanics of Staking

- blockchain protocol maintains “staking contract”
- validators (identified by public key) lock up funds in this contract

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?
4. Staking rewards?

Mechanics of Staking

- blockchain protocol maintains “staking contract”
- validators (identified by public key) lock up funds in this contract

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?
4. Staking rewards?
   - e.g., inflationary block rewards a la Nakamoto consensus
   - increasingly common: pay fixed interest rate on stake, conditional on timely participation

Mechanics of Staking

- blockchain protocol maintains “staking contract”
- validators (identified by public key) lock up funds in this contract

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?
4. Staking rewards?
5. Delegation? (i.e., loan funds to validator for share of staking rewards)

Mechanics of Staking

- blockchain protocol maintains "staking contract"
- validators (identified by public key) lock up funds in this contract

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?
4. Staking rewards?
5. Delegation? (i.e., loan funds to validator for share of staking rewards)
   – if not (e.g., in Ethereum), expect 3rd-party staking pools to arise

Mechanics of Staking

Design decisions:

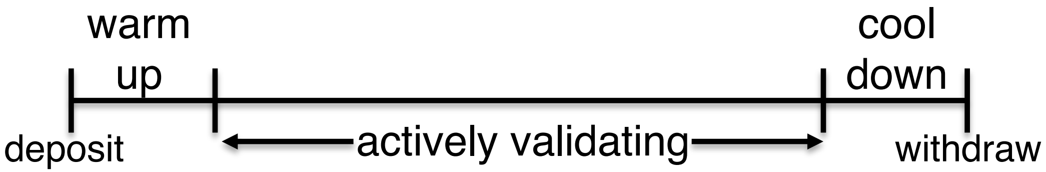
1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?
4. Staking rewards?
5. Delegation?

Upshot: blockchain protocol maintains list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators and their stake amounts.

Mechanics of Staking

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?
4. Staking rewards?
5. Delegation?



Upshot: blockchain protocol maintains list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators and their stake amounts. (note: may have sybils)

Mechanics of Staking

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?
4. Staking rewards?
5. Delegation?

Upshot: blockchain protocol maintains list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators and their stake amounts. (note: may have sybils)

– q_i's include delegated stake

Mechanics of Staking

Design decisions:

1. Mininum/maximum staking duration.
2. Minimum/maximum staking amount.
3. Warm-up/cool-down periods?
4. Staking rewards?
5. Delegation?

Upshot: blockchain protocol maintains list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators and their stake amounts. (note: may have sybils)

- q_i's include delegated stake
- in warm up or cool down period ➔ not in this list

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to the q_i's.

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to the q_i's.

Fact: surprisingly tricky!

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to the q_i's.

Fact: surprisingly tricky!

Reason: need to produce internally defined randomness.

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to the q_i's.

Fact: surprisingly tricky!

Reason: need to produce internally defined randomness.

- proof-of-work:

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to the q_i's.

Fact: surprisingly tricky!

Reason: need to produce internally defined randomness.

- proof-of-work: randomness imported from external process (namely, mining), impossible for validators to manipulate
  - assuming cryptographic hash function is unpredictable

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to the q_i's.

Fact: surprisingly tricky!

Reason: need to produce internally defined randomness.

- proof-of-work: randomness imported from external process (namely, mining), impossible for validators to manipulate
  - – assuming cryptographic hash function is unpredictable
- proof-of-stake: blockchain protocol (seemingly) must come up with (pseudo)randomness itself

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to the q_i's.

Fact: surprisingly tricky!

Proof-of-stake: blockchain protocol (seemingly) must come up with (pseudo)randomness itself.

– assuming no external source that can be trusted to provide randomness

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to the q_i's.

Fact: surprisingly tricky!

Proof-of-stake: blockchain protocol (seemingly) must come up with (pseudo)randomness itself.

- – assuming no external source that can be trusted to provide randomness

- canonical example: use hash of some part of blockchain state

Why Proof-of-Stake Is Hard

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Goal: sample pk from $\{pk_1,\ldots,pk_n\}$ with probability proportional to the q_i's.

Fact: surprisingly tricky!

Proof-of-stake: blockchain protocol (seemingly) must come up with (pseudo)randomness itself.

- – assuming no external source that can be trusted to provide randomness

- canonical example: use hash of some part of blockchain state
- issue: opportunities for validators to manipulate the random selection process (e.g., when assembling a new block)

Weighted Round Robin

Given: list $(pk_1, q_1), \ldots, (pk_n, q_n)$ of active validators + stake amounts.

Solution:

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Solution: use epoch of length N views each (N large).

- list of active validators + their stakes changes only at epoch boundaries

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Solution: use epoch of length N views each (N large).

- list of active validators + their stakes changes only at epoch boundaries
- each epoch: use proportionally representative sequence of leaders

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Solution: use epoch of length N views each (N large).

- list of active validators + their stakes changes only at epoch boundaries
- each epoch: use proportionally representative sequence of leaders
- ex: {(A,2),(B,1),(C,2)} ➔ use leader sequence AABCCAABCCAABCC…

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Solution: use epoch of length N views each (N large).

- list of active validators + their stakes changes only at epoch boundaries
- each epoch: use proportionally representative sequence of leaders
- ex: {(A,2),(B,1),(C,2)} ➔ use leader sequence AABCCAABCCAABCC…

Good news: relatively simple, no fancy cryptography.

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots,(pk_n,q_n)$ of active validators + stake amounts.

Solution: use epoch of length N views each (N large).

- list of active validators + their stakes changes only at epoch boundaries
- each epoch: use proportionally representative sequence of leaders
- ex: {(A,2),(B,1),(C,2)} ➔ use leader sequence AABCCAABCCAABCC…

Good news: relatively simple, no fancy cryptography.

Bad news: leaders of future views known well in advance.

- ➔ risk of bribery, coercion, DoS attacks
- also has its benefits (e.g., for tx dissemination)

Weighted Round Robin

Given: list $(pk_1,q_1),\ldots, (pk_n,q_n)$ of active validators + stake amounts.

Solution: use epoch of length N views each (N large).

– list of active validators + their stakes changes only at epoch boundaries
– each epoch: use proportionally representative sequence of leaders
– ex: {(A,2),(B,1),(C,2)} ➔ use leader sequence AABCCAABCCAABCC…

- good news: relatively simple, no fancy cryptography
- bad news: leaders of future views known well in advance

More sophisticated: verifiable random functions (VRFs).

– leader unknown prior to their block proposal (cf., Nakamoto consensus)