Visualizing Proof-of-Stake Algorithms with Microprocessors (StakeSim Code Companion)

Author	josh
Date	2024-07-05

Overview

Proof-of-Stake is a consensus mechanism used to secure decentralized blockchains like Ethereum. This system randomly selects a node to validate the transactions for a particular block, and thus receive a reward for doing so. With proof-of-work, nodes use a large amount of computing power to guess nonces using a cryptographic hash, with the winner going to the first node to find a guess that meets certain numerical criteria (the difficulty target). With proof-of-stake, nodes instead put up a "stake" as collateral instead of computation resources, and a random selection algorithm is used to pick the winner, favoring those with a higher stake. But how can we pick a random winner on a peer-to-peer, decentralized network? Let's dive into proof-of-stake consensus with a fun toy algorithm that runs on microcontrollers!

Proof-of-Stake, Visualized

Random Selection on a P2P Network

Choosing a random validator and getting all nodes to agree is a non-trivial problem on a peer-to-peer network. There's no central authority that can simply generate a secure random number and pick the winner. Instead, we have to get a wide network of nodes that may enter and leave the network to agree on the selected validator each and every block. How can we generate a random number when all the nodes on the network have an incentive to favor themselves? The ingenious solution used in many proof-of-stake algorithms is called a *commit → reveal* scheme. This algorithm takes advantage of the key properties of cryptographic hash functions to generate a random number, while preventing nodes from tampering with the results. This mechanism allows each participating node to pool their *entropy* together for generating the random selection.

A *cryptographic hash* is a one-way function that always gives the same output for a given input. If you hash the string "hello" using SHA-256, you always get a hash of "2cf24dba…". As well, the output of a cryptographic hash gives no indication of what the input was. There's nothing in "2cf24dba…" that hints at what the original input was ("hello"). This property is called *preimage resistance*. This property comes in handy for our commit → reveal scheme.

To generate our random number, all the participating validators first generate their own individual random value using their machine's cryptographically secure source. They then *hash* that value, and *commit* only the hash to blockchain. At this point, the only thing anyone else can see is the hash (not the random value in its raw form). Once the commit stage has passed, we enter the *reveal* stage. At this point, the validators share the actual random value. We don't one anyone to be able to manipulate the final selection to their favor, so each node then makes sure

that the commit hash matches the revealed random value. Since a cryptographic hash always gives the same output for a given input, we can easily see if someone's revealed value matches what they previously committed to the blockchain.

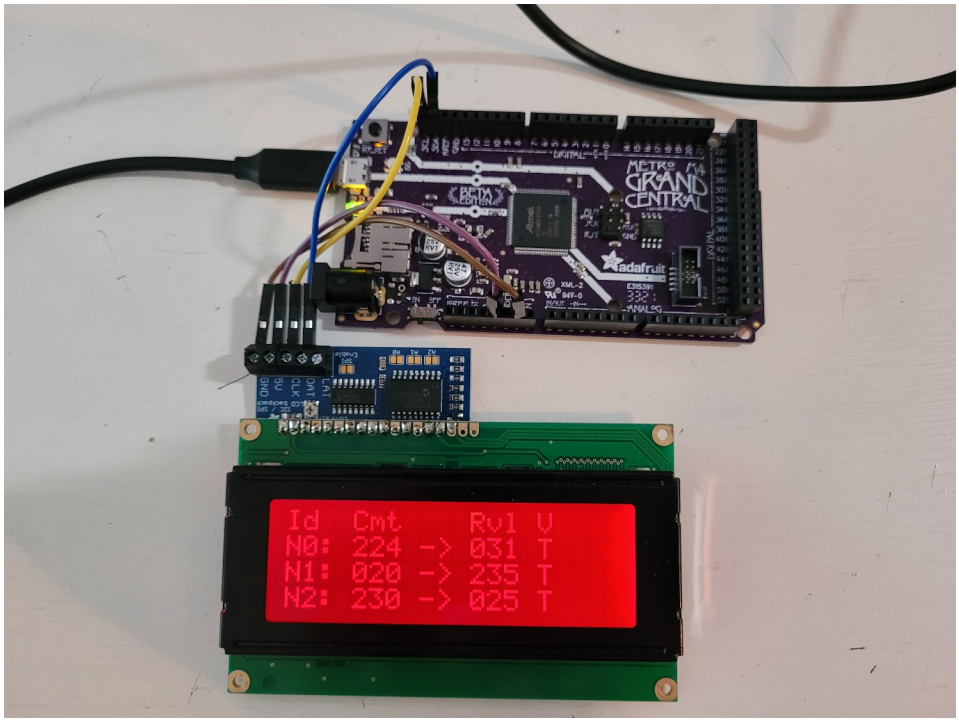
Once the commit → reveal stage is complete, the nodes can then combine the random values using some simple method like a bitwise XOR. Now all the nodes have the same random selection agreed upon, and can run a weighted selection algorithm to pick the final winner. We want nodes with a higher stake to have a higher chance of being selected, since they are risking a higher amount of collateral. But we don't want the richest nodes to win every time either, so our weighted random has a possibility of any validator being selected. This node will validate all the transactions in the block, and must follow the rules to avoid their collateral being slashed. Ultimately, a new block is minted with a reward of new coins and the transaction fees from that block going to the winning validator.

Stake Simulating with a Toy Algorithm

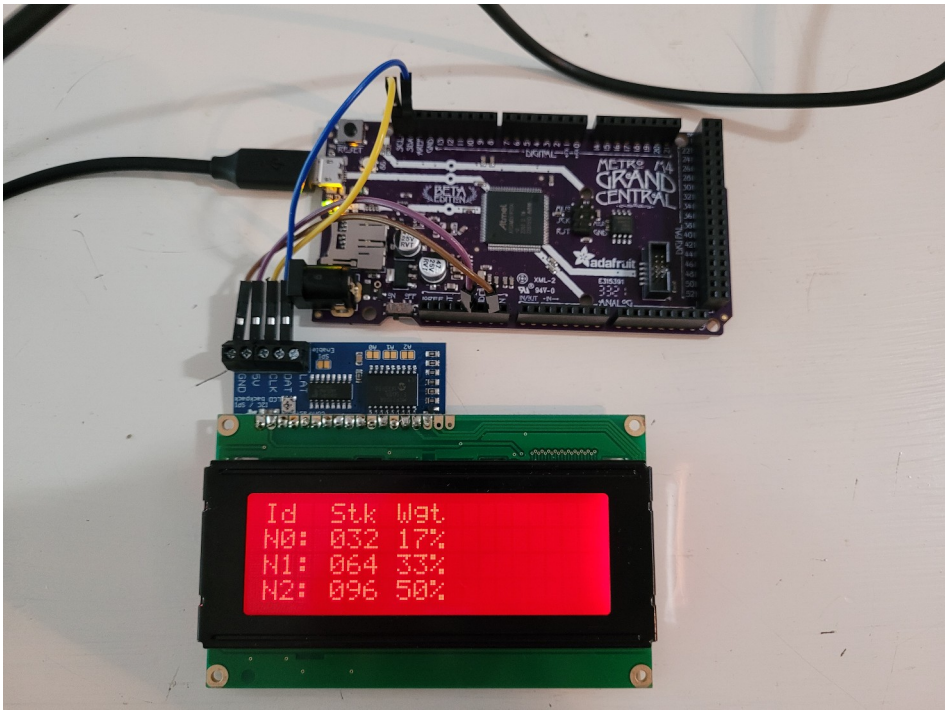
StakeSim uses a simple 8-bit algorithm for demonstrating this concept. The small numbers make it easier to visualize how this algorithm works, without the complexity of true cryptographically secure proof-of-stake. This toy simulation generates a random number for each node from 0-255 (the numbers that fit in 8 bits). These numbers are hashed using a toy 8-bit hash for the commit part of the scheme. The numbers are then revealed and validated, just like in real POS. Next, the numbers are combined using a simple bitwise XOR, resulted in our final random value. I use a simple percentage-based weighted algorithm to pick the winning validator, and display all the stages on-screen.

```
  # Hashing method for the demo
# Return a really simple 8 bit hash
# This is for educational purposes, so we don't need a
# cryptographically secure hash, we just need one that works
# to "shuffle" the data appearance wise
@staticmethod
def hash_8bit(data):

    hash8 = (data ^ StakeNode.HASH_XOR) % StakeNode.HASH_MOD

    return hash8

# Commit random bytes to the blockchain
# Hashes the internal random number and returns the hash
def commit(self):

    entropy_hash8 = self.hash_8bit(self.entropy)

    return entropy_hash8

# Get the combined entropy of all peers
# Use a bitwise XOR to do the combination
def _combine_entropy(self, peers):

    # Start with our own entropy
    # Combine with each peer by doing a bitwise XOR
    # Order of the peers does not matter
```

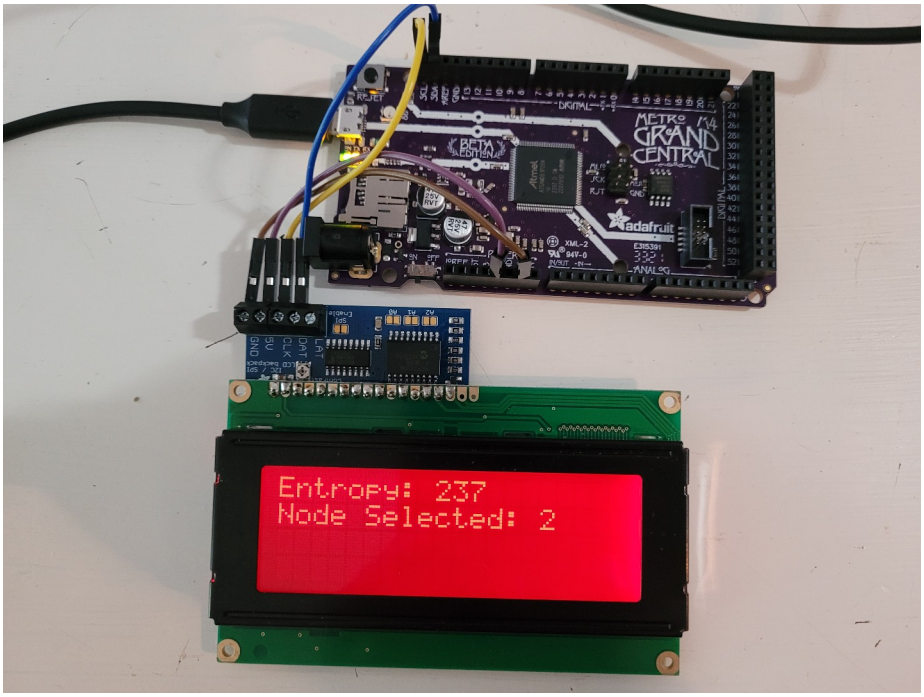
```
combined_entropy = self.entropy
for peer in peers:
    combined_entropy = combined_entropy ^ peer.entropy

return combined_entropy
```



The simulation code showing the 8-bit commit -> reveal scheme and validation



Showing stakes and percentages (weights) for each node



Showing the final 8-bit entropy value and the selected node

Simulated Proof-of-Stake, 8 Bit Style

This fun CircuitPython code demonstrates proof-of-stake with an easy to understand toy algorithm. This code shows the same general algorithm as the real thing. We start with a commit → reveal and validate each node's values. We then combine them together using a simple bitwise XOR, resulting in the final 8-bit value. We do a weighted selection using that random value, favoring nodes with a higher stake but giving everyone a chance. This visual demo runs on Adafruit M4 microprocessors with a character LCD. A while back, I did an educational demo of Proof-of-Work using a similar 8-bit toy algorithm called MicroProver, so if you're interested in POW you can learn using that demo as well. The code for this project is free and open source as always, and I hope you learn something about blockchain proof-of-stake consensus from this!