Lecture 12: Scaling Energy via Proof-of-Stake

Principles of Blockchains, University of Illinois,
Professor: Pramod Viswanath
Scribe: Xuechao Wang

March 9, 2021

Abstract

In the past lectures we have seen proposals that significantly improve the performance of Bitcoin while retaining its security. One aspect that has remained unchanged is the PoW mining procedure, inherent to the Bitcoin protocol. PoW mining consumes a huge amount of resources (electricity and specialized computer hardware), comparable to that of a medium to large sized country. In this lecture, we study how to replace PoW by an alternative which is computationally trivial, thus eradicating the electricity consumption inherent to Bitcoin. The alternative to PoW is called Proof of Stake and is the focus of this lecture.

Introduction

The blockchains we have seen this far, from Bitcoin to its improvements, are based on the proof-of-work (PoW) mining procedure. The PoW procedure is central to the design, serving multiple roles: protecting against network spam, serving as a block proposer election procedure, deterring protocol level attacks from adversaries. However the PoW mining procedure is extremely resource consuming, both in terms of sheer electricity requirements as well as specialized computer hardware (GPUs) to speed up the search for nonces. Electricity consumption connected to Bitcoin's mining has been estimated to be comparable to that of a medium to large sized country. Another way of looking at the outsize electricity wastage of Bitcoin: a single Bitcoin transaction consumes electricity equivalent to 750,000 credit card (e.g., Visa) transactions. Not only is this comparison staggering, worse is the observation that as more participants enter the Bitcoin ecosystem, the mining procedure gets more competitive and more electricity is consumed. This is a form of *reverse* network effect, where the more the participation, the worse the efficiency. As such, an extremely compelling dimension of Bitcoin scaling is *energy*. This is the goal of this lecture, where we will focus on a specially attractive option, *Proof of Stake* (PoS), as a replacement to PoW.

From PoW to PoS

In a PoW protocol, successful mining ("work") is related to being able to propose a block; the larger the fraction of hash power, the more the probability of success in the mining "lottery" (the right to the next valid block on the blockchain). In proof of stake (PoS) protocols, the mining lottery is simply replaced so that each node wins with probability proportional to its *stake* in the total pool. The stake refers to the total amount of coins the node, as estimated based on the confirmed ledger up to the point of mining; security of the ledger ensures that each honest node has the same estimate of the stake.

In this lecture, we focus on how to design a secure PoS protocol by mimicking the PoW longest chain protocol as much as possible. In the longest chain protocol, a miner succeeds in succeeds in

finding a nonce such that the following inequality is satisfied:

$$H(\text{prev\_hash}, \text{merkel\_root}, \text{nonce}) < T, \tag{1}$$

where $H(\cdot)$ is a cryptographic hash function, prev_hash is the hash of the header of the previous (parent) block, merkel_root is the Merkel root of the transactions in the block, nonce is a integer that can be adjusted by the miner, and T is a pre-defined threshold deciding the mining difficulty.

When a PoS system is launched, a collection $\mathcal{N}$ of nodes are initialized. Each node $n \in \mathcal{N}$ is initialized with a coin possessing stake stake_n, a public/secret key pair $(\text{pk}_\text{n}, \text{sk}_\text{n})$ for signature. The genesis block contains all public keys and initial stakes of all nodes. For now, let us assume the stake of a node does not change according to the transactions in the blockchain. This is called the static stake setting. We now discuss how to design a leader election mechanism in the PoS system by modifying Equation (1).

To eliminate the role of hash power, the nonce in Equation (1) must be removed. Also to prove the ownership of a coin, we can feed the public key of the coin into the hash function. Since we want each node wins with probability proportional to its stake, the right hand side of Equation (1) should be proportional to the stake. Combining both these, we make our first attempt at the PoS lottery.

PoS Attempt 1. A node n succeeds in mining a block if

$$H(\text{prev\_hash}, \text{merkel\_root}, \text{pk}_\text{n}) < T \cdot \text{stake}_\text{n}. \tag{2}$$

However, even though the nonce is removed, a node can still try to "game" the lottery by trying diferent Merkel roots (this is done by inserting different transactions and different orderings in the block). Then again nodes equipped with better machines will have the advantage in winning this lottery; such an attempt is referred to as a *grinding attack*. Therefore, the Merkel root in the block header also has to removed.

PoS Attempt 2. A node n succeeds in mining a block if

$$H(\text{prev\_hash}, \text{pk}_\text{n}) < T \cdot \text{stake}_\text{n}. \tag{3}$$

However, for a fixed T, there is always a non-zero probability that no one can satisfy Equation (3); in this case, the protocol simply stalls forever. A way out is to allow everyone a second chance if no one wins the first round of the lottery. We can add one more variable in the hash function: the timestamp ts. We can discretize the time, for example, each node makes one attempt every second.

PoS Attempt 3. A node n succeeds in mining a block if

$$H(\text{prev\_hash}, \text{ts}, \text{pk}_\text{n}) < T \cdot \text{stake}_\text{n}. \tag{4}$$

Since time keeps progressing (at least for honest nodes), this attempt prevents the stalling issue. However, since the block header no longer contains the Merkel root of all the transactions, the transactions can be added (or altered) into the block content *after* the node wins the lottery. So, unlike PoW, the entire block is no longer "sealed" by the PoS. How to ensure sure that the blockchain is tamper-resistant?

One possible solution is the following: while the block headers are linked by hash pointers, the block contents also form a hash chain (see Figure 1). With this structure, the adversary cannot modify the entire chain as long as some of the blocks is owned by honest nodes. Nevertheless, an adaptive adversary can still corrupt all honest nodes who own blocks on the chain and again the adversary is able to change any block on the chain. This can be resolved by a cryptographic primitive called *Key Evolving Signature* (KES), where keys are periodically erased and generated, while the new key is linked to the previous one. A simple example of KES involves periodically changing the public key of any coin by its hash output. The block content must be signed by the winner of the

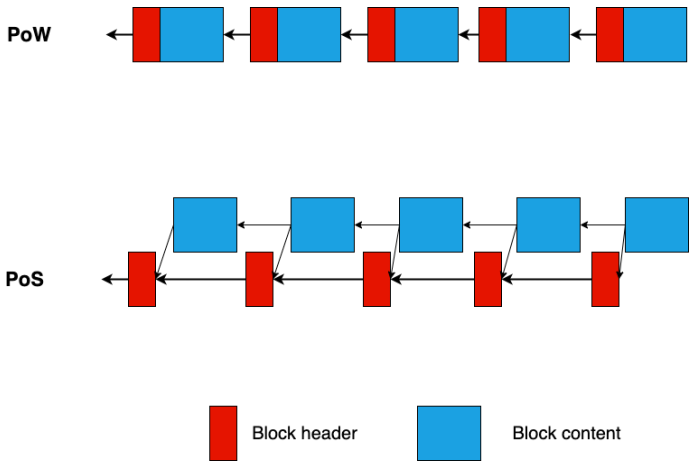

Figure 1: Comparison of PoW and PoS blockchain structures. In the PoW longest chain, the header of each block contains a Merkel root of the block content, hence the entire block is sealed by PoW. In the PoS longest chain, to avoid grinding, the block content is added after the header wins the lottery. To guarantee tamper resistance, the block content has hash pointers to the block header and the previous block content.

lottery using KES. We can ask honest nodes to erase all the old keys, which ensures immutability of the contents of honest blocks against an adaptive adversary.

Note that the set of public keys is available to every node which means the winner of a lottery is actually known to public ahead of time, so an adaptive adversary may still corrupt/bribe an honest node who is going to win the lottery at a known time in the future. To avoid this, we introduce another cryptographic primitive called *verifiable random function* (VRF), which enables the nodes to run the lottery with their *secret keys* as opposed to the public keys; the output of the VRF is verified using the corresponding public key. This is summarized below.

PoS Attempt 4. A node n succeeds in mining a block if

$$\mathrm{VRF}(\mathrm{prev\_hash}, \mathrm{ts}, \mathrm{sk_n}) < T \cdot \mathrm{stake_n}, \tag{5}$$

where VRF generates a pseudorandom number with a proof of its correctness. A node with a secret key sk can call $\mathrm{VRF}_{\mathrm{PROVE}}(\cdot, sk)$ to generates a pseudorandom *output* $F_{sk}(\cdot)$ along with a *proof* $\pi_{sk}(\cdot)$. Other nodes that have the proof and the corresponding public key pk can check that the output has been generated by VRF, by calling $\mathrm{VRF}_{\mathrm{VERIFY}}(\cdot, F_{sk}(\cdot), \pi_{sk}(\cdot), pk)$. The output of a VRF is computationally indistinguishable from a random number even if the public key pk and the function $\mathrm{VRF}_{\mathrm{PROVE}}$ is revealed.

Nothing-at-stake Attack

Note that in Equation (5), there is still one variable in the hash function that the adversary can grind on: prev_hash. While the honest nodes are asked to mine on the tip of the longest chain, the adversary tries to grow blocks *everywhere* in the block tree and tries to surpass the longest honest chain (see Figure 2). This is possible because the adversary can use the same stake to mine blocks at many nodes in the blocktree without much computational cost; hence, this strategy is known as a *nothing-at-stake* (NaS) attack.

While the number of adversarial blocks grows exponentially under the NaS attack, fortunately the depth of the adversarial tree grows only linearly. This paper shows that the growth rate of the

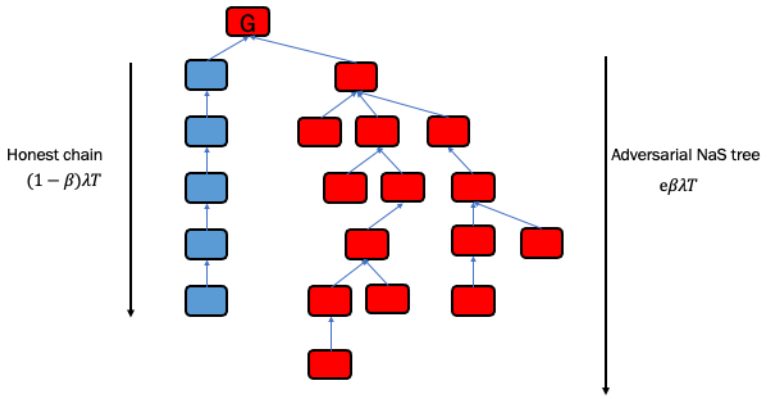

Figure 2: Nothing-at-stake attack.

longest chain in the NaS tree turns out to approach $e\beta\lambda$ where e is the base of the natural logarithm (approximately 2.718), if the adversarial mining rate is $\beta\lambda$. Hence, the effect of the NaS attack is to *amplify* the growth rate of the private chain by a factor of e, roughly double the rate at which the private chain would have grown. Therefore, the NaS attack is successful when $e\beta\lambda > (1-\beta)\lambda$, i.e., $\beta \geq 1/(1+e) = 26.89\%$. Could a different attack (such as the balancing attack we have seen on GHOST succeed with even smaller adversarial hash power? It turns out that the private attack again is the worst case attack even in the PoS setting: using the idea of blocktree partition and Nakamoto block introduced in Lecture 6 (from a recent paper), we can show that the protocol is secure (safe and live) against all attacks as long as $\beta > 1/(1+e)$. Therefore, we conclude that the security threshold for adversarial hash power for this protocol is exactly $1/(1+e)$. This is roughly half the threshold of 50% of adversarial mining power for the security of the corresponding PoW longest chain protocol,

Boosting Security Threshold

A natural question would be whether the threshold can be raised to 50%, matching the PoW longest chain threshold; ideally this improvement is achieved by a simple modification of the PoS protocol. Towards this, we observe that the PoS protocol in Equation (5) is vulnerable to the NaS attack because each block in the blocktree provides a source of randomness and starts a new independent lottery. To mitigate this issue, we can update the source of randomness *less frequently*.

Ouroboros PoS Protocol. A first order idea is to never update the randomness and only draw it from the genesis block.

PoS Attempt 5.1. A node n succeeds in mining a block if

$$\mathrm{VRF}(\mathrm{hash}(\mathrm{Genesis}), \mathrm{ts}, \mathrm{sk_n}) < T \cdot \mathrm{stake_n}. \tag{6}$$

This is the lottery used in a PoS protocol called Ouroboros Praos, which is deployed as part of the Cardano project. The Ouroboros PoS protocol proceeds in discrete epochs; essentially a new genesis block starts of each epoch. Recent works have established that the protocol in Equation 6 is secure as long as the *majority* of staking power is honest, matching the threshold of the PoW longest chain protocol. Further, the private attack is the worst possible attack, and the resulting security

threshold (on honest staking power) the optimal one; the proof methodology of Nakamoto blocks espoused in the analysis of the NaS attack is also applicable here.

However, the Ouroboros Praos protocol in Equation 6 is vulnerable to a bribing attack, where the adversary establishes a website where it can offer a bribe to anyone who posts their credentials for proposing blocks in an upcoming epoch (see Figure 3). Although a node's future proposer status is not public knowledge (thanks to VRF), this bribing website can solicit such information ahead of time and help launch a fatal attack. In particular, a double-spend attack can be launched using these bribed nodes: the adversary bribes these leaders to sign a forked version of the blockchain that it hoards till the block s is confirmed by a k-deep rule. After that point, the adversary releases the hoarded blockchain to all the users thus switching the longest chain and confirming a block s' (which contains a double spend) instead of s. We note that while the adversary requires $k+1$ winners out of $2k+1$ lotteries to respond to the bribe, the total stake represented by these bribed winners can be a very tiny fraction of the total stake.

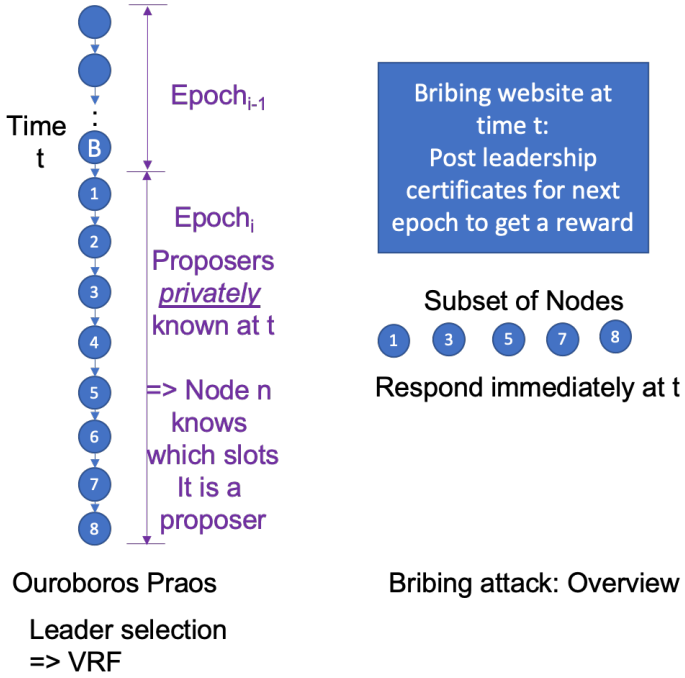

Figure 3: Structure of bribing attack.

c-Correlation PoS Protocol. A smooth way to reduce the frequency of source of randomness in the PoS lottery is the following, proposed as c-correlation in this paper. In this rule, the common source of randomness remains the same for c blocks, and is updated only when the current block to be generated is at a depth that is a multiple of c (see Figure 5). When updating, the hash of block header is used as the new source of randomness, i.e.,

$$\text{RandSource}(b) := \begin{cases} \text{VRF}\big(\text{RandSource}(\text{parent}(b)), \text{ts}, \text{sk}\big), & \text{if depth}\big(b\big)\%c = 0, \\ \text{RandSource}(\text{parent}(b)), & \text{otherwise.} \end{cases}$$

PoS Attempt 5.2. A node n succeeds in mining a block if

$$\text{VRF}(\text{RandSource}(\text{parent}), \text{ts}, \text{sk}_\text{n}) < T \cdot \text{stake}_\text{n}. \tag{7}$$

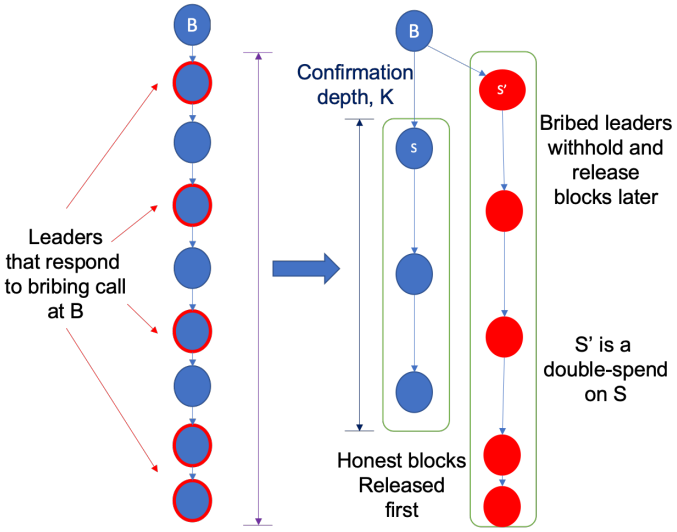

Figure 4: Bribing attack on Ouroboros Praos.

When $c = 1$, this recovers the protocol in Equation (5), where the NaS attack is most effective. When $c = \infty$, this recovers the protocol in Equation (6), where the NaS attack is least effective (nonexistent). Increasing c gracefully increases the security threshold (see Table 1). In Figure 6, we plot the security threshold of c-correlation, with the practical implication that c-correlation can achieve comparable security with a much smaller epoch size than a current implementation of Ouroboros as part of the Cardano project.

c	1	2	3	4	5	6	7	8	9	10
ϕ_c	e	2.22547	2.01030	1.88255	1.79545	1.73110	1.68103	1.64060	1.60705	1.57860
β_c^*	$\frac{1}{1+e}$	0.31003	0.33219	0.34691	0.35772	0.36615	0.37299	0.37870	0.38358	0.38780

Table 1: Numerically computed growth rate ϕ_c and stake threshold β_c^*.

Dynamic Stake Using s-truncation

With the PoS protocol in Equation (7) we have successfully completed the secure conversion of the PoW lottery into the PoS lottery. One hitch remains: the stake of the nodes used to compute the success level in the lottery is *static*, i.e., not changing. This is undesirable for two reasons: a node with no current stake can participate in the lottery (raising barriers to new entrants) and an incumbent participates in the lottery at a great cost since it cannot use its coins for any transactions (creating incentives for rent seeking). A natural approach is to allow *dynamic stakes* to determine the success in the PoS lottery; after all, the ledger is dynamically updated as new blocks enter the longest chain and get confirmed (when deep enough) and the security of the PoS longest chain rule guarantees consistency of the ledger amongst all the honest nodes. How to systematically rein in dynamic stake to the PoS lottery and to do it without allowing new attack vectors is the focus of this section.

In the dynamic stake setting, the stake of a node n is not only changing over time as transactions are added to the blocktree, but also over which *chain* we are referring to in the blocktree. Different

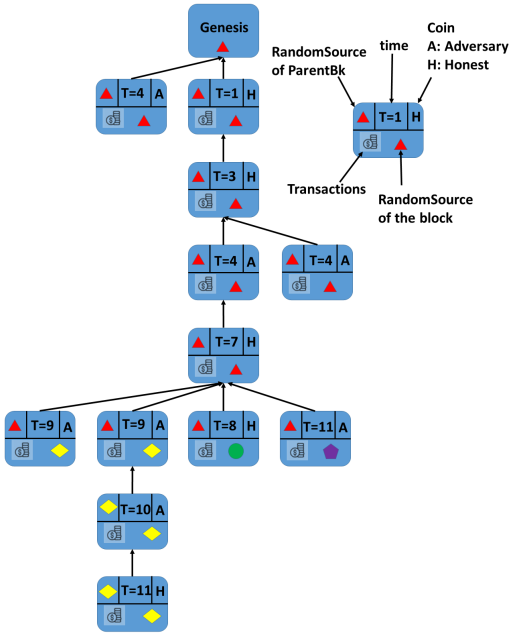

Figure 5: A snapshot of a block tree under c-correlation protocol with $c = 5$. Each block has a header consisting of header=(RandSource(parent), timestamp). The content of the block includes the transactions and the RandSource of the block to be used in the next lottery. This random source is preserved from parent to children blocks, until one mines a block at height that is a multiple of c. In such a case, a new RandSource is drawn using the hash of the current header and the secret key of the block proposer.

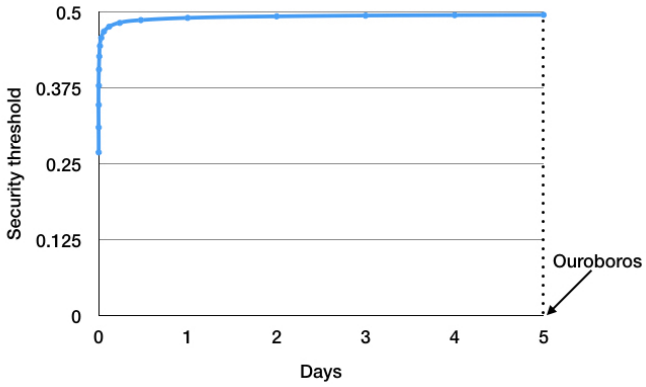

Figure 6: The security threshold β_c^* of c-correlation against the epoch size, equaling to c times the inter-block time, which we set to be 20s, to match the implementation of Orouboros in Cardano. The Cardano project currently updates the common randomness every 5 days (21600 blocks), while the security threshold of c-correlation can approach $1/2$ with much higher randomness update frequency.

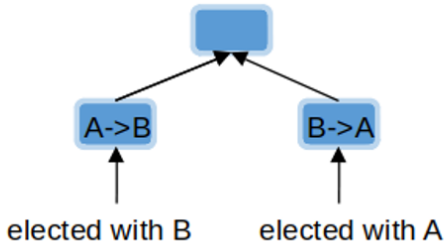

Figure 7: Stake grinding attack. A and B are two coins owned by one node P. When P wins one lottery, it creates two blocks (privately) with the same header: in one block, P transfer the stake from A to B; in the other, P transfer the stake from B to A. In the lottery for the next level, P can use its full stake to participate in two lotteries, therefore its chance of winning is doubled by grinding on its stake.

chains in the tree contain different sequences of transactions, leading to different stake allocations. One needs to specify which chain we are referring to, when we access the stake of a node. If we simply use the state of the parent block to decide the stake of each node, the adversary can *grind on the stake of its coin* (the right hand side in Equation (7)). In this stake grinding attack, once the adversary is elected as a leader at some time and proposed a new block, it can include transactions in that block to transfer all stake to a coin that has a higher chance of winning the lottery at later time. In the example of Figure 7, the adversary owns coin A and coin B. When it gains the right to proposer a block, it creates two blocks (privately) with the same header: in one block, it transfers the stake from A to B; in the other, it transfers the stake from B to A. In the lottery for the next level, the adversary can use its full stake to participate in two lotteries, therefore its chance of winning is doubled by grinding on its stake.

To prevent such a grinding on the stake, a natural attempt is to use the stake in the block with depth $\ell - s$ when trying to create a block at depth ℓ on the main chain where s is chosen large enough so that nodes have reached consensus on the block with depth $\ell - s$. However, there remains a vulnerability. Consider the adversary growing its own private chain from the genesis block (or any block in the blocktree). Initially, the private chain grows at a rate $\beta\lambda$. However, after s blocks from the launch of the private chain, the adversary can start grinding on the private key of the coin; once a favorable coin is found, it can transfer the stake to the favored coin by including transactions in the first ancestor block in the private chain. This is possible as all blocks in the private chain belong to the adversary. It can alter any content of the private chain and sign all the prior blocks again. With such a stake grinding attack, the adversary can potentially win a lottery every slot in the private chain, eventually overtaking the public blocktree, which grows at a constant rate $(1 - \beta)\lambda$. To prevent this private grinding attack, a new fork choice rule called s-truncated longest chain is proposed in this paper.

New fork choice rule. Note that in the stake grinding attack, the private chain grows slowly at the beginning, so instead of comparing the length of the chains, we can compare the density of the chains when there is a folk. In particular, at any time, an honest node keeps track of one main chain that it appends its next generated block to. Upon receiving a new chain of blocks, it needs to decide which chain to keep. Instead of comparing the length of those two chains, as in the longest chain rule, we compare the creation time of the first s blocks after the fork in truncated versions of those two chains. Let b_{fork} be the block where those two chains fork. The honest node counts how long it takes in each chain to create, up to s blocks after the fork. The chain with shorter time for those s blocks is chosen, and the next generated block will be appended to the newest block in that selected chain. One caveat is that we only apply this s-truncation when comparing two chains that

both have at least s blocks after those two chains forked. If one of the chain has less than s blocks after forking, we use the longest chain rule to determine which chain to mine on. This completes the successful conversion of the PoW lottery to the PoS version, even accounting for dynamic stake variation.

Dynamic Availability Using VDF

An important feature enjoyed by Bitcoin is called *dynamic availability*: Bitcoin can handle an uncertain and dynamic varying level of participation in terms of mining power; Miners can join and leave as desired without any registration requirement. So can we achieve similar dynamic availability in the PoS version?

Note that in practice, perhaps not every stakeholder is interested in participating in the protocol (consensus layer). A certain fraction of people possess stake as investment; we call them offline nodes. Those stakeholders actively participating in the protocol are called *online* nodes. The protocol we presented as far is secure if less than a certain fraction (e.g., 27% for $c = 1$) of the online nodes are adversarial, but with an additional assumption: *all adversarial nodes are always online.* While this assumption seems reasonable (why would an adversary go offline?), in reality this can be a very restrictive condition. Generally, in public blockchains, PoW or PoS, no node is likely to be adversarial during the launch of a new blockchain token (because the token has little value); adversaries only begin to emerge later during the life-cycle.

This additional assumption underlying the protocol is not superfluous but is in fact *necessary* for its security. Suppose for the first year of the existence of the PoS-based blockchain, only 10% of the total stake is online. Out of this, consider that all nodes are honest. Now, at the beginning of the second year, all 100% of the stake is online out of which 20% is held by adversary. At any point of time, the fraction of online stake held by honest nodes is greater than 0.8. However, the protocol is not secure since the adversary can use its 20% stake to immediately participate in all past lotteries to win blocks all the way back to the genesis and then grow a chain *instantaneously* from the genesis to surpass the current longest chain (Figure 8(a)). This is because generating blocks in the PoS protocol is almost costless (just need to run all the lotteries using old timestamps). Thus, due to this "costless simulation", newly online adversary nodes not only increase the current online adversary stake, but effectively increase past online adversary stake as well. In contrast, PoW does not suffer from the same issue because it would take a long time to grow such a chain from the past and that chain will always be behind the current longest chain. Thus, PoW provides an *arrow of time*, meaning nodes cannot "go back in time" to mine blocks for the times at which they were not online. This property is key in endowing PoW protocols with the ability to tolerate fully dynamic adversaries wherein both honest nodes and adversary can have varying participation (Figure 8(b)).

A recent proposal, called PoSAT, addresses this dynamic availability issue. It uses a cryptographic primitive called Verifiable Delay Function (VDF) for the block proposal lottery to provide an arrow of time. VDFs are built on top of iteratively sequential functions, i.e., functions that are only computable sequentially: $f^{\ell}(x) = f \circ f \circ ... \circ f(x)$, along with the ability to provide a short and easily verifiable proof that the computed output is correct. Examples of such functions include (repeated) squaring in a finite group of unknown order, i.e., $f(x) = 2^x$ and (repeated) application of secure hash function (SHA-256), i.e., $f(x) = \text{Hash}(x)$. The VDF will also produce a proof for each output, and verifying the output with the help of the proof is much faster than computing the output. This is really a cryptographic magic! While VDFs have been designed as a way for proving the passage of a certain amount of time (assuming a bounded CPU speed), it has been recently shown that these functions can also be used to generate an unpredictable randomness beacon. Thus, running the iteration till the random time L when $f^L(x)$ is within a certain threshold will result in L being a geometric random variable. This randomized VDF functionality can be used to create an arrow-of-time in the PoS protocol.

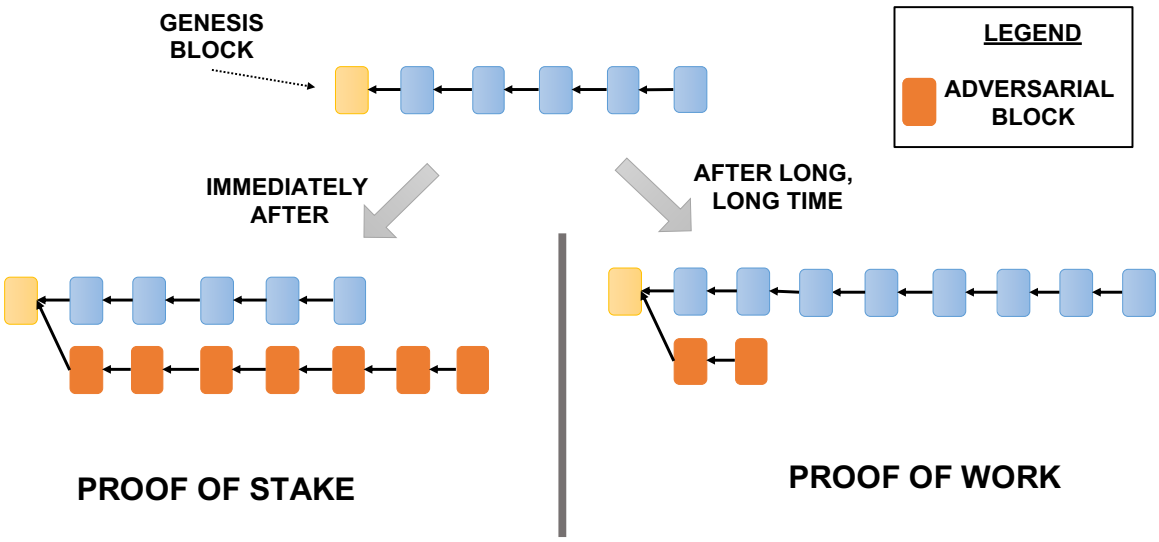

Figure 8: (a) Newly coming online stakeholders in the PoS protocol can grow a chain from genesis instantaneously. (b) Newly joined miners in the PoW protocol takes a long time to grow such a chain and is always behind.

PoS Attempt 6. A node n succeeds in mining a block if

$$\mathrm{VDF}(\mathrm{RandSource}(\mathrm{parent}), \mathrm{ts}, \mathrm{sk_n}) < T \cdot \mathrm{stake_n}. \tag{8}$$

With VDF, the adversary cannot go back in time and create a longer chain than the honest chain immediately, because creating a longer chain now requires time. While the adversarial chain is growing, the honest chain is growing at a larger speed. Therefore, PoSAT is secure in the dynamic available setting.

References

There are broadly two families of PoS protocols: those inspired by the Nakamoto longest chain protocol and those derived from decades of research in Byzantine Fault Tolerant (BFT) protocols. In this lecture, we covered most existing Nakamoto-style PoS protocols. A straightforward adoption proposed by Fan&Zhou achieves security when the adversary controls less than 27% fraction of total stake. Both Ouroboros Praos and c-correlated PoS boost the security threshold: while the former is vulnerable to a bribing attack (due to predictability of the lotteries), the latter provides a tradeoff between predictability and security. Finally, we see the s-truncation rule in Ouroboros Genesis enables dynamic stake and PoSAT achieves dynamic availability. Attempts at blockchain design via the BFT approach include Algorand and Hotstuff. The adaptation of these new protocols into blockchains is an active area of research and engineering, and we will take a closer look at this family of protocols in Module 3 of this course.