Chapter 26

Advanced Blockchain

In this chapter we study various advanced blockchain concepts, which are popular in research.

26.1 Selfish Mining

Satoshi Nakamoto suggested that it is rational to be altruistic, e.g., by always attaching newly found block to the longest chain. But is it true?

Definition 26.1 (Selfish Mining)**.** *A selfish miner hopes to earn the reward of a larger share of blocks than its hardware would allow. The selfish miner achieves this by temporarily keeping newly found blocks secret.*

Algorithm 26.2 Selfish Mining

```
1: Idea: Mine secretly, without immediately publishing newly found blocks
2: Let d_p be the depth of the public blockchain
3: Let d_s be the depth of the secretly mined blockchain
4: if a new block b_p is published, i.e., d_p has increased by 1 then
5:    if d_p > d_s then
6:       Start mining on that newly published block b_p
7:    else if d_p = d_s then
8:       Publish secretly mined block b_s
9:       Mine on b_s and publish newly found block immediately
10:   else if d_p = d_s − 1 then
11:      Publish both secretly mined blocks
12:   end if
13: end if
```

Remarks:

- If the selfish miner is more than two blocks ahead, the original research suggested to always answer a newly published block by releasing the oldest unpublished block. The idea is that honest miners will then split their mining power between these two blocks. However, what matters is how long it takes the honest miners to find the next block,

to extend the public blockchain. This time does not change whether the honest miners split their efforts or not. Hence the case $d_p < d_s - 1$ is not needed in Algorithm 26.2.

Theorem 26.3 (Selfish Mining)**.** *It may be rational to mine selfishly, depending on two parameters α and γ, where α is the ratio of the mining power of the selfish miner, and γ is the share of the altruistic mining power the selfish miner can reach in the network if the selfish miner publishes a block right after seeing a newly published block. Precisely, the selfish miner share is*

$$\frac{\alpha(1-\alpha)^2(4\alpha+\gamma(1-2\alpha))-\alpha^3}{1-\alpha(1+(2-\alpha)\alpha)}.$$

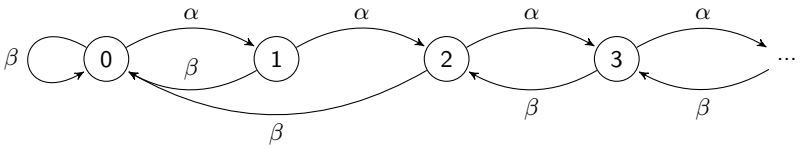

Figure 26.4: Each state of the Markov chain represents how many blocks the selfish miner is ahead, i.e., $d_s - d_p$. In each state, the selfish miner finds a block with probability α, and the honest miners find a block with probability $\beta = 1 - \alpha$. The interesting cases are the "irregular" β arrow from state 2 to state 0, and the β arrow from state 1 to state 0 as it will include three subcases.

Proof. We model the current state of the system with a Markov chain, see Figure 26.4.

We can solve the following Markov chain equations to figure out the probability of each state in the stationary distribution:

$$\begin{aligned} p_1 &= \alpha p_0 \\ \beta p_{i+1} &= \alpha p_i, \text{ for all } i > 1 \\ \text{and } 1 &= \sum_i p_i. \end{aligned}$$

Using $\rho = \alpha/\beta$, we express all terms of above sum with p_1:

$$1 = \frac{p_1}{\alpha} + p_1 \sum_{i \geq 0} \rho^i = \frac{p_1}{\alpha} + \frac{p_1}{1-\rho}, \text{ hence } p_1 = \frac{2\alpha^2 - \alpha}{\alpha^2 + \alpha - 1}.$$

Each state has an outgoing arrow with probability β. If this arrow is taken, one or two blocks (depending on the state) are attached that will eventually end up in the main chain of the blockchain. In state 0 (if arrow β is taken), the honest miners attach a block. In all states i with $i > 2$, the selfish miner eventually attaches a block. In state 2, the selfish miner directly attaches 2 blocks because of Line 11 in Algorithm 26.2.

State 1 in Line 8 is interesting. The selfish miner secretly was 1 block ahead, but now (after taking the β arrow) the honest miners are attaching a competing block. We have a race who attaches the next block, and where. There are three possibilities:

- Either the selfish miner manages to attach another block to its own block, giving 2 blocks to the selfish miner. This happens with probability α.
- Or the honest miners attach a block (with probability β) to their previous honest block (with probability $1-\gamma$). This gives 2 blocks to the honest miners, with total probability $\beta(1-\gamma)$.
- Or the honest miners attach a block to the selfish block, giving 1 block to each side, with probability $\beta\gamma$.

The blockchain process is just a biased random walk through these states. Since blocks are attached whenever we have an outgoing β arrow, the total number of blocks being attached per state is simply $1+p_1+p_2$ (all states attach a single block, except states 1 and 2 which attach 2 blocks each).

As argued above, of these blocks, $1-p_0+p_2+\alpha p_1-\beta(1-\gamma)p_1$ are blocks by the selfish miner, i.e., the ratio of selfish blocks in the blockchain is

$$\frac{1-p_0+p_2+\alpha p_1-\beta(1-\gamma)p_1}{1+p_1+p_2}.$$

□

Remarks:

- If the miner is honest (altruistic), then a miner with computational share α should expect to find an α fraction of the blocks. For some values of α and γ the ratio of Theorem 26.3 is higher than α.
- In particular, if $\gamma = 0$ (the selfish miner only wins a race in Line 8 if it manages to mine 2 blocks in a row), the break even of selfish mining happens at $\alpha = 1/3$.
- If $\gamma = 1/2$ (the selfish miner learns about honest blocks very quickly and manages to convince half of the honest miners to mine on the selfish block instead of the slightly earlier published honest block), already $\alpha = 1/4$ is enough to have a higher share in expectation.
- And if $\gamma = 1$ (the selfish miner controls the network, and can hide any honest block until the selfish block is published) any $\alpha > 0$ justifies selfish mining.

26.2 DAG-Blockchain

Traditional Bitcoin-like blockchains require mining blocks sequentially. Sometimes effort is wasted if two blocks happen to be mined at roughly the same time, as one of these two blocks is going to become obsolete. DAG-blockchains (where DAG stands for directed acyclic graph) try to prevent such wasted blocks. They allow for faster block production, as forks are less of a problem.

Definition 26.5 (DAG-blockchain)**.** *In a DAG-blockchain the genesis block does not reference other blocks. Every other block has at least one (and possibly multiple references) to previous blocks.*

Definition 26.6 (DAG-Relations)**.** *Block p is a dag-parent of block b if block b references (includes a hash) to p. Likewise b is a dag-child of p. Block a is a dag-ancestor of block b, if a is b's dag-parent, dag-grandparent (dag-parent of dag-parent), dag-grandgrandparent, and so on. Likewise b is a's dag-descendant.*

Theorem 26.7. *There are no cycles in a DAG-blockchain.*

Proof. A block b includes its dag-parents' hashes. These dag-parents themselves include the hashes of their dag-parents, etc. To get a cycle of references, some of b's dag-ancestors must include b's hash, which is cryptographically infeasible. □

Definition 26.8 (Tree-Relations)**.** *We are going to implicitly mark some of the references in the DAG of blocks, such that these marked references form a tree, directed towards the genesis block. For every non-genesis block one edge to one of its dag-parents is marked. We use the prefix "tree" to denote these special relations. The marked edge is between tree-parent and tree-child. The tree also defines tree-ancestors and tree-descendants.*

Remarks:

- In other words, every tree-something is also a dag-something, but not necessarily vice versa.
- Blocks do not specify who is their tree-parent, or the order of their dag-parents. Instead, tree-parents are implicitly defined as follows.

Definition 26.9 (DAG Weight)**.** *The weight of a dag-ancestor block a with respect to a block b is defined as the number of tree-descendants of a in the set of dag-ancestors of b. If two blocks a and a' have the same weight, we use the hashes of a and a' to break ties.*

Definition 26.10 (Parent Order)**.** *Let x and y be any pair of dag-parents of b, and z be the lowest common tree-ancestor of x and y. x' and y' are the tree-children of z that are tree-ancestors of x and y respectively. If x' has a higher weight than y', then block b orders dag-parent x before y.*

Definition 26.11 (Tree-Parent)**.** *The tree-parent of b is the first dag-parent in b's parent order.*

Remarks:

- Now we can totally order all the blocks in the DAG-Blockchain.

Theorem 26.13. *Let p be the tree-parent of b. The order of blocks $<_b$ computed by Algorithm 26.12 extends the order $<_p$ by appending some blocks.*

Proof. Block p is the first dag-parent of b, so in the first iteration of the loop, we have $<_b = <_p$. Further modifications of $<_b$ consist only of appending more blocks to $<_b$, ending with block b itself. □

Algorithm 26.12 DAG-Blockchain Ordering

We totally order all dag-ancestors of block b as $<_b$ as follows:
Initialize $<_b$ as empty
for all dag-parents p of b, in their parent order **do**
    Compute $<_p$ (recursively)
    Remove from $<_p$ any blocks already included in $<_b$
    Append $<_p$ at the end of $<_b$
end for
Append block b at the end of $<_b$

Remarks:

- Note that b is appended to the order only after ordering all its dag-ancestors. The genesis block is the only block where the recursion will stop, so the genesis block is always first in the total order.
- By Theorem 26.13 tree-children extend the order of their tree-parent, so appending blocks to the DAG preserves the previous order and new blocks are appended at the end.

Definition 26.14 (Transaction Order)**.** *Transactions in each block are ordered by the miner of the block. Since blocks themselves are ordered, all transactions are ordered. If two transactions contradict each other (e.g. they try to spend the same money twice), the first transaction in the total order is considered executed, while the second transaction is simply ignored (or possibly punished).*

Remarks:

- Ethereum allows blocks to not only have a parent, but also up to two "uncles" (childless blocks). In contrast to above description, blocks must specify the main parent.
- In Ethereum, new blocks are mined approximately every 15 seconds (as opposed to 10 minutes in Bitcoin). New blocks being generated in such rapid succession leads to a lot of childless blocks. Uncles have been introduced to not "waste" those blocks.
- In Ethereum, the original uncle-miners get 7/8 of the block reward. The miner who references these uncle blocks also gets a small reward. This reward depends on the height-difference of the uncle and the included parent. Also, to be included, the uncle and the current block should have a common ancestor not too far in the past.

26.3 Smart Contracts

Definition 26.15 (Ethereum)**.** *Ethereum is a distributed state machine. Unlike Bitcoin, Ethereum promises to run arbitrary computer programs in a blockchain.*

Remarks:

- Like the Bitcoin network, Ethereum consists of nodes that are connected by a random virtual network. These nodes can join or leave the network arbitrarily. There is no central coordinator.
- Like in Bitcoin, users broadcast cryptographically signed transactions in the network. Nodes collate these transactions and decide on the ordering of transactions by putting them in a block on the Ethereum blockchain.

Definition 26.16 (Smart Contract)**.** *Smart contracts are programs deployed on the Ethereum blockchain that have associated storage and can execute arbitrarily complex logic.*

Remarks:

- Smart Contracts are written in higher level programming languages like Solidity, Vyper, etc. and are compiled down to EVM (Ethereum Virtual Machine) bytecode, which is a Turing complete low level programming language.
- Smart contracts cannot be changed after deployment. But most smart contracts contain mutable storage, and this storage can be used to adapt the behavior of the smart contract. With this, many smart contracts can update to a new version.

Definition 26.17 (Account)**.** *Ethereum knows two kinds of accounts. Externally Owned Accounts (EOAs) are controlled by individuals, with a secret key. Contract Accounts (CAs) are for smart contracts. CAs are not controlled by a user.*

Definition 26.18 (Ethereum Transaction)**.** *An Ethereum transaction is sent by a user who controls an EOA to the Ethereum network. A transaction contains:*

- *Nonce: This "number only used once" is simply a counter that counts how many transactions the account of the sender of the transaction has already sent.*
- *160-bit address of the recipient.*
- *The transaction is signed by the user controlling the EOA.*
- *Value: The amount of Wei (the native currency of Ethereum) to transfer from the sender to the recipient.*
- *Data: Optional data field, which can be accessed by smart contracts.*
- *StartGas: A value representing the maximum amount of computation this transaction is allowed to use.*
- *GasPrice: How many Wei per unit of Gas the sender is paying. Miners will probably select transactions with a higher GasPrice, so a high GasPrice will make sure that the transaction is executed more quickly.*

Remarks:

- There are three types of transactions.

Definition 26.19 (Simple Transaction)**.** *A simple transaction in Ethereum transfers some of the native currency, called Wei, from one EOA to another. Higher units of curency are called Szabo, Finney, and Ether, with* 10^{18} *Wei* $= 10^6$ *Szabo* $= 10^3$ *Finney = 1 Ether. The data field in a simple transaction is empty.*

Definition 26.20 (Smart Contract Creation Transaction)**.** *A transaction whose recipient address field is set to* 0 *and whose data field is set to compiled EVM code is used to deploy that code as a smart contract on the Ethereum blockchain. The contract is considered deployed after it has been mined in a block and is included in the blockchain at a sufficient depth.*

Definition 26.21 (Smart Contract Execution Transaction)**.** *A transaction that has a smart contract address in its recipient field and code to execute a specific function of that contract in its data field.*

Remarks:

- Smart Contracts can execute computations, store data, send Ether to other accounts or smart contracts, and invoke other smart contracts.
- Smart contracts can be programmed to self destruct. This is the only way to remove them again from the Ethereum blockchain.
- Each contract stores data in 3 separate entities: storage, memory, and stack. Of these, only the storage area is persistent between transactions. Storage is a key-value store of 256 bit words to 256 bit words. The storage data is persisted in the Ethereum blockchain, like the hard disk of a traditional computer. Memory and stack are for intermediate storage required while running a specific function, similar to RAM and registers of a traditional computer. The read/write gas costs of persistent storage is significantly higher than those of memory and stack.

Definition 26.22 (Gas)**.** *Gas is the unit of an atomic computation, like swapping two variables. Complex operations use more than 1 Gas, e.g., ADDing two numbers costs 3 Gas.*

Remarks:

- As Ethereum contracts are programs (with loops, function calls, and recursions), end users need to pay more gas for more computations. In particular, smart contracts might call another smart contract as a subroutine, and StartGas must include enough gas to pay for all these function calls invoked by the transaction.
- The product of StartGas and GasPrice is the maximum cost of the entire transaction.

- Transactions are an all or nothing affair. If the entire transaction could not be finished within the StartGas limit, an Out-of-Gas exception is raised. The state of the blockchain is reverted back to its values before the transaction. The amount of gas consumed is not returned back to the sender.

Definition 26.23 (Block)**.** *In Ethereum, like in Bitcoin, a block is a collection of transactions that is considered a part of the canonical history of transactions. Among other things, a block contains: pointers to parent and up to two uncles, the hash of the root node of a trie structure populated with each transaction of the block, the hash of the root node of the state trie (after transactions have been executed)*

26.4 Payment Hubs

How to we enable many parties to send payments to each other efficiently?

Definition 26.24 (Payment Hub)**.** *Multiple parties can send payments to each other by means of a payment hub.*

Remarks:

- While we could always call the smart contract to transfer money between users that joined the hub, every smart contract call costs as it involves the blockchain. Rather, we want a frugal system with just few blockchain transactions.

Definition 26.25 (Smart Contract Hub)**.** *A smart contract hub is a payment hub that is realized by a smart contract on a blockchain and an off-chain server. The smart contract and the server together enable off-chain payments between users that joined the hub.*

Algorithm 26.26 Smart Contract Hub

1: Users join the hub by depositing some native currency of the blockchain into the smart contract
2: Funds of all participants are maintained together as a fungible pool in the smart contract
3: Time is divided into epochs: in each epoch users can send each other payment transactions through the server
4: The server does the bookkeeping of who has paid how much to whom during the epoch
5: At the end of the epoch, the server aggregates all balances into a commitment, which is sent to the smart contract
6: Also at the end of the epoch, the server sends a proof to each user, informing about the current account balance
7: Each user can verify that its balance is correct; if not the user can call the smart contract with its proof to get its money back

Remarks:

- The smart contract lives forever, but the server can disappear anytime. If it does, nodes can show their recent balance proofs to the smart contract and withdraw their balances.
- The server can be scaled to in terms of latency and number of users. The smart contract does not need to scale as it only needs to just accept one commitment per epoch.
- In case the server disappears, the smart contract will be flooded with withdrawal requests, and could be subject to delays based on the delays of the underlying blockchain.

26.5 Proof-of-Stake

Almost all of the energy consumption of permissionless (everybody can participate) blockchains is wasted because of proof-of-work. Proof-of-stake avoids these wasteful computations, without going all the way to permissioned (the participating nodes are known a priori) systems such as Paxos or PBFT.

Definition 26.27 (Proof-of-stake)**.** *Proof-of-work awards block rewards to the lucky miner that solved a cryptopuzzle. In contrast, proof-of-stake awards block rewards proportionally to the economic stake in the system.*

Remarks:

- Literally, "the rich get richer".
- Ethereum is expected to move to proof-of-stake eventually.
- There are multiple flavors of proof-of-stake algorithms.

Definition 26.28 (Chain based proof-of-stake)**.** *Accounts hold lottery tickets according to their stake. The lottery is pseudo-random, in the sense that hash functions computed on the state of the blockchain will select which account is winning. The winning account can extend the longest chain by a block, and earn the block reward.*

Remarks:

- It gets tricky if the actual winner of the lottery does not produce a block in time, or some nodes do not see this block in time. This is why some suggested proof-of-stake systems add a voting phase.

Definition 26.29 (BFT based proof-of-stake)**.** *The lottery winner only gets to propose a block to be added to the blockchain. A committee then votes (yes, byzantine fault tolerance) whether to accept that block into the blockchain. If no agreement is reached, this process is repeated.*

Remarks:

- Proof-of-stake can be attacked in various ways. Let us discuss the two most prominent attacks.
- Most importantly, there is the "nothing at stake" attack: In blockchains, forks occur naturally. In proof-of-work, a fork is resolved because every miner has to choose which blockchain fork to extend, as it does not pay off to mine on a hopeless fork. Eventually, some chain will end up with more miners, and that chain is considered to be the real blockchain, whereas other (childless) blocks are just not being extended. In a proof-of-stake system, a user can trivially extend all prongs of a fork. As generating a block costs nothing, the miner has no incentive to not extend all the prongs of the fork. This results in a situation with more and more forks, and no canonical order of transactions. If there is a double-spend attack, there is no way to tell which blockchain is valid, as all blockchains are the same length (all miners are extending all forks). It can be argued that honest miners, who want to preserve the value of the network, will extend the first prong of the fork that they see. But that leaves room for a dishonest miner to double spend by moving their mining opportunity to the appropriate fork at the appropriate time.
- Long range attack: As there are no physical resources being used to produce blocks in a proof-of-stake system, nothing prevents a bad player from creating an alternate blockchain starting at the genesis block, and make it longer than the canonical blockchain. New nodes may have difficulties to determine which blockchain is the real established blockchain. In proof-of-work, long range attacks takes an enormous amount of computing power. In proof-of-stake systems, a new node has to check with trusted sources to know what the canonical blockchain is.