alias: [PoS]

A Consensus mechanism for descentralized, distributed systems. The major competition to Proof of Work.

Sources:

- Cosmo Network comparison of Casper and Tendermint
- Combining GHOST and Casper (Vitalik et. al.)

Open questions

Q: Why is it always *1/3* of validators?
A: Answer can be found in the paper *Asynchronous consensus and broadcast protocols* dealing with *Practical Byzantine Fault Tolerance*.

In proof of work, creating a block is extremely expensive. In proof of stake, it is essentially free: you only need to sign it. All the effort in proof of stake is in making it costly to create an invalid or malicious block.

> Instead of using computational power to propose blocks, proposing blocks is essentially free. In exchange, we need an additional layer of mathematical theory to prevent perverse incentives that arise when we make proposing blocks "easy."
> Combining GHOST and Casper

Main ingredients:

1. A Fork Choice Rule: a function $fork()$ that when given a *view* G (ie, a series of blocks) can identify a canonical chain with its single leaf node (ie: latest block) B. Intuitively: a rule for validators to choose the right chain when presented with conflicting blocks.
2. A Finality definition: a function $fork$ that given G will return a set $F(G)$ of *finalized* blocks which will never be removed from the canonical chain.
3. Slashing conditions: conditions that honest validators will never break and dishonest validators can be provably identified.

Note that a consensus mechanism can have:

1. **Safety**: if the set $F(G)$ of finalized blocks can never contain conflicting blocks.
2. **Liveness**: if the $F(G)$ actually *grows*. Liveness can be plausible (impossible to become deadlocked) or probabilistic (if it liveness needs some probabilistic assumptions).

Pitfalls of PoS

- **Nothing at stake**: validators have no incentive to vote for a specific blob of data (in Blockchain, a block). If several choices are present, the rational behavior is to vote for all of them to be assured I get the reward for voting on the winning choice. Solved by Slashing.
- **Long range attacks**: if I can remove my deposit, I in collution with other validators remove it... then create a fork from the point where I was still a validator (why not? what are you gonna slash?). Here enters the embrace of weak subjectivity by PoS, which introduces *subjectivty* into the protocol: the biggest chain *will not be always that with most work or stake*, but some other considerations are taken into account: validators node must currently have a stake, etc.
- **Cartel formation**: few validators with a lot of capital will outweight smaller ones. Tendermint basically accepts this. Casper protocol is the only construction which tries to fight this with in-consensus censorship-resistance incentives *(which ones?)*

Implementation strategies

Two main strategies to implement PoS.

Byzantine Fault Tolerant Proof of Work

Chooses consistency over availability: some transactions may not get processed, but you sure as hell will get a consistent state. Tendermint is an example of this type of Consensus.

A validator is assigned the right to propose a block at random. Commiting that block needs a supermajority of 2/3 of validators (*why this number?*).

As an example of consistency over availability: if more than 2/3 of validators in Tendermint are offline, the network may halt: there's not enough voters to vote for the next block.

There are no forks on Tendermint based blockchains: forks are the result of two miners finding a block at the same time. With a single validator proposing a node, this is just impossible.

This may seem like a silly problem, but actually it is quite bad: if a network upgrade makes 2/3 of validators offline, the rest of validators can't continue with the old chain!

Chain-based Proof of Work

They simulator more directly Proof of Work. The new block is hash-linked to the parent block. Casper Protocol is of this kind. It favor availability over consistency.

As of 2021, I don't see many differences with Tendermint. Seems like they have converged? Maybe.

GASPER

The objective of this piece is to gain intuition on how Ethereum's Beacon Chain works for the reader familiar with proof of work blockchains, such as current Ethereum (as of April 2022) or Bitcoin.

Time

A key difference is the introduction of *time*. In Proof of Work, the blockchain itself can be seen as a malfunctioning clock, where each block is the quantum of time. Event are measured in their block-time. This works because Proof of Work requires *work*, and because that work should be relatively difficult, that *work* requires physical time.

In Proof of Stake, creating blocks is trivial. You actually just ensemble the data and off it goes. So we actually need to introduce clocks into the blockchain. This is, to me, is one of the main differences one should keep in mind.

As mentioned above, Gasper introduces time with *slots*.

- Slots are 12 second long, and slots *may* have a block associated. An empty slot is possible if no one proposed a block.
- *Epochs* are rounds of 32 blocks, where a committee of validators are chosen to *attest* for their head of the chain.

Checkpoints

Another very important difference, related to the introduction of natural time, is that we now have *special* blocks: the blocks that begin each epoch. In traditional proof of work, we had as a special case only the Genesis block. Now, every block that starts an epoch is special, and is called a *epoch boundary block*.

Block publishing

Every time an *epoch* starts, a new committee is chosen at random from the set of validators.

Every time an *slot* starts, one committee member must propose a block by computing the canonical head of the chain at that slot. The block itself is fairly similar to those in proof of work, but for two *details*:

1. It contains a filed *newttests*, attestations that the proposer has accepted and haven't appeared in their view of the blockchain yet.
2. Arbitrary data. This is actually a fairly important detail, not for the protocol, but for Ethereum: this arbitrary data is, in the Beacon Chain, the hash of the head of the block of the execution layer! (See Eth1Data on the Beacon Chain Explorer)

Attestations

So, what are exactly attestations? Intuitively, attestations are messages containing a block from each validators view. But they also contain a so-called *checkpoint edge*, a transition from *epoch boundaries*. These are used for justification and finalization.

We can see attestations published with the blocks in the explorer. Note that these attestations are *aggregated* so as not to overload the block proposer with messages.

Note that we can attest not only for the previous block published, but for even older blocks than that, although attestation rewards are best when the delay is kept at a minimum.

Justification and finalization

Justification is a new concept from those of us used to traditional Ethereum, while finalization takes a different form.

We say that a checkpoint B (ie: an epoch boundary) is *justified* when more than 2/3 of the stake has attested for the transition between a previously justified checkpoint and B. The genesis block is always justified, allowing us to loop out of the recursion.

As far as I know, *justification* is a concept useful only to build up to *finality*. While in traditional proof of work blockchains we have *probabilistic finality*, in GASPER it is baked into the protocol.

For those unfamiliar with *finality*, a *message* (transaction, block) is considered final when it can't be reversed. In Bitcoin and other proof of work blockchains, finality is always probabilistic: it is *extremely unlikely* that after six confirmations your block will get reversed, but it is *possible* in theory.

In GASPER, *finality* is provable: we can definitively say that a message is final and the protocol will *reject* any attempt to publish a message which conflicts with a finalized one. Finality is achieved in GASPER when a checkpoint block is justified and it is used to justify the next block.

So, in ideal conditions, *justification* of a checkpoint happens as soon as the next epoch starts (everyone voted to pass from the state A to B and A was justified, so B is justified) and *finalization* of a checkpoint happens after two epochs (everyone voted to pass from B to C and B was justified, so B is now finalized).

Consistency

Another interesting point to analyze are forks. Not all proof of stake protocols have forks, for example Tendermint consensus only publishes blocks if enough attestations have been gathered for a block. In GASPER, though, forks are possible.

This should be surprising for a distracted reader like me, as I always assume forks happens because of two miners finding the solution for a block at nearly the same time; and Gasper uses only *one* block proposer, so this is impossible.

But the real world is a messy place, and network latency exists: we only need to imagine two validators with different views of the blockchain proposing blocks with different parents. The fork choice rule of GASPER is tricky, but its intuition its simple. As proof of works blockchain prefer the chain with the highest amount of work put into them, GASPER will prefer the chain with the highest amount of validations in it.

GASPER makes the strategic choice to prefer to be always live instead of always consistent. Allowing forks is a sacrifice needed to be able to forego setting strict timeouts.

Liveness

The GASPER network favors availability and will continue publishing blocks even though not enough validators are online, **but** blocks will never be justified or finalized. Once this situation corrects itself though, the blockchain can continue! Of course it is possible that things don't actually correct *themselves* and a hardfork to modify the list of validators is needed.

This is again different from Tendermint consensus, which only guarantees liveness in case a supermajority of validators are online.

Another important point is that absolutely no timeouts is assumed to guarantee the consistency promises made, and only a *partially synchronous* assumption is made to discuss liveness. That is, it only needs for timeouts to *exists* but they are not set beforehand.

All in all, I think this is a good base to start forming an intuition on how Ethereum is gonna work after The Merge. The Beacon Chain has been working for a while under this model already and it is very interesting to explore its blocks. The rest of this doc are my notes while reading the Combining GHOST And Casper paper by Vitalik et al. As notes tend to go, I think they are fairly organized, but there may be mistakes in my understanding of some topics and repeated information. You have been warned. They are probably best read while reading the paper itself.

CASPER FFG

Or **Casper the Friendly Finality Gadget**. Defined *justification* and *finalization*.

Definitions

- Every block has a *height*.
- *Checkpoint* blocks are defined which are blocks whose height is a multiple of a constant H (in Ethereum2, $H = 100$). Checkpoint height $\frac{h(B)}{H}$ is always an integer. Note that the

subsets of checkpoint blocks also forms a subtree.

- *Attestations* are signed messsages (ie: blocks) containing *checkpoint edges* $A \to B$ where A and B are checkpoint blocks. Each *attestation* is a *vote* to move from A to B. Each attestation has a *weight*, the stake of the validator.
- In each view of the blockchain G, there is a set of *justified* and *finalized* checkpoint blocks $J(G)$ and $F(G)$ where $F(G) \subset J(G) \subset G$. The genesis is always both justified and finalized.
- A checkpoint block *B* is *justified* (by a *justified* checkpoint block A) if *A* is justified and there are *attestations* voting for $A \to B$ with total weight 2/3 of the total stake. This is also called a *supermajority link* from A to B.
- A checkpoint block B is *finalized* if it is *justified* and there's a *supermajority link* $B \to C$.

Note (!): *attestations* are defined over *checkpoint blocks*, not all blocks!
Note: *justification* is dependent of the view of the node, because in your view you may not have seen all attestation or even the to-be-justified checkpoint block B

GHOST

Intuitively: the chain with most attestations is the correct one.

Combining GHOST and CASPER FFG to form GASPER

Back to GASPER then: even though CASPER FFG does not mention *slots* or *epochs*, GASPER does. Another important difference is that the same block may appear as a *checkpoint* more than once (for different epochs). To disambiguate, one formally must refer to a GASPER checkpoint as a pair (B, e) where B is the block and e is the epoch.

Another change is that GASPER calls divides its validators *committees* in each *epoch*, and one *committee* is used per slot. In each slot, one *validator* from the *committee* proposes a block, and all member of the *committee* attest to what they see as the head of the chain (in the best case, the proposed block) using the fork-choice rule *HLMD GHOST* (a slight variation of GHOST).

Epoch boundary

Intuition: the aep is the *attestation epoch* and is the period during which attestations for a block can be gathered. The *epoch* is objective, but the aep of a block depends on your view of the chain.

The *latest epoch boundary block* of B ($LEBB$) is the latest *checkpoint block* before B or, if none, the latest block before B.

The $jEBB$ is the epoch-adjusted epoch boundary of a block. You *go back* j epochs.

- For every block B, $EBB(B, 0)$ is the genisis.
- $slot(B) = jC$ for some epoch j, B will be an epoch boundary block in every chain that includes it (ie: a checkpoint is always an epoch boundary block, if it exists).

Opinion: non intuitive concept. They key is that *every block* needs a *EBB*, and if you cannot find it, *make one up*. So if I have a chain `63 <- 64 <- 65` and `64` marks the beginning of an epoch, then $EBB(65) = 64$. But if there's a fork an another validator proposes the block `66` like `63 <- 66`, then $EBB(66) = 63$, even though 63 was not a checkpoint block.

Now, with the *epoch boundary pairs* (which are the B, j pair identifying the epoch boundary of a block) we define the *attestation epoch* j for $P = (B, j)$ using the notation $aep(B) = j$ which is not necessarily the same as $ep(B)$.

These *epoch boundary pairs* are used as the *checkpoints*.

Note: the *epoch* is absolute, the *epoch boundary* is not. Even in the view `63 <- 66`, `63` nodes not begin an *epoch*, even though it serves as *EBB* of 66.

Note: why make it difficult it with pairs? The paper discussed this. It gives two main reasons: (1) it makes fewer assumptions about latency, as nodes may not have produced a block for the slot which is supposed to be a checkpoint, so the pair represent a best-approximation (2) we can sync time in proof of stake, so let's use that! we can use the concept that blocks will be produced at regular intervals and requrires both data (represented in the blockchain) and time (represented in *epochs*).

Committees

Fairly intuitive. In each *epoch*, the work of is divided in roughly equal sized *committees* for each *slot*.

Blocks and attestation

Two kinds of *committee* work:

1. On person proposes a block
2. Everyone in the committee attests to *their* head of the chain

Note: see how this allows forking! Everyone attests to *their* head othe chain, not for the proposed block!

Let's do the rounds of a slot i from the point of view of *validator* V:

1. The *proposer* (the first in the list of the committee) uses the Fork Choice Rule to find B'.

2. The *proposer* proposes a block B which is a message containing $slot(B) = i$, $P(B) = B'$, $newattests(B)$ a set of pointer to *all* attestations V has accepted but have not been included in any previous blocks and some *implementation specific data* (see **Note on fancyness**).
3. Each *validator* in the *committee* computes B' and publishes an *attestation* α which is a message containing: $slot(\alpha) = i$, $block(\alpha) = B'$ and a *checkpoint edge* $LJ(\alpha) \rightarrow LE(\alpha)$. LJ and LE are *epoch boundary pairs* in the view of the validator at time *i* plus some amount of time due to delay. These functions are defined later.

Note on fancyness: the paper wants to be fancy and say it doesn't care about the data, but as users we do. This is the `Eth1Data` found in the Beacon Chain implementations, and is extremely important as it tracks the execution layer results! So basically *what we now know as Ethereum* is all in there.

Justification

Definitions:

- $view(B)$ the view consisting of B and all its ancestors.
- $ffgview(B)$, the *FFG View of B*, to be $view(LEBB(B))$ (ie:the view since the last epoch boundary block)

- $(A, j') \rightarrow (B, j)$ is a *supermajority link* from pair (A, j') to pair (B, j) if the attestations with checkpoint edge $(A, j') \rightarrow (B, j)$ have a total weight of more than 2/3 of the stake.
- Given a view G, we define *justified of G*, $J(G)$, as the genesis plus all the blocks such that if $(A, j') \in J(G)$ and $(A, j') \rightarrow (B, j)$ with *supermajority*, the $(B, j) \in J(G)$ as well. Meaning: if there's a supermajority vote for the state transition into (B, j) and the previous state was justified, then the new state is justified.

Time to define LJ and LE! Finally. Remember from Blocks and attestation?

Given an attestation α and $B = LEBB(block(\alpha))$:

- $LJ(\alpha)$, the *last justified pair of* α, the highest attestation epoch justified pair in $ffgview(block(\alpha)) = view(B)$.
- $LE(\alpha)$, the _last epoch boundary pair of α, to be $(B, ep(slot(\alpha)))$.

Daunting. OK. Intuitively, LJ is then the latest justified epoch boundary which is in view of B. LE is just the last epoch boundary. OK. Not so hard.

Finalization

Core concept: once a block is finalized, *no view* will have a conflicting block with it, unless the blockchain is (1/3)-slashable (ie: everything is broken, someone controls more than 1/3 of the stake).

Finalization happens for a block and epoch pair (B_0, j) if:

- (B_0, j) is the genesis at epoch zero.
- Or if there's a $k \geq 1$ such that $(B_0, j), \ldots, (B_{k-1}, j + k - 1)$ which are justified and there's a supermajority to transition $(B_0, j) \rightarrow (B_k, j + k)$.

The last condition is confusing, but in plain English it states that the finalized block must be justified *and* there must be a supermajority that votes to transition from it to another state.

Note: the difference between finalization and justification is not as intuitive as one may wish but looking at the formulas, but is easy intuitively: a block is *justified* when the supermajority voted to transition into its state, a block is *finalized* when it is used as the base to go into another state.

In even plainer English: in the happy case, a justified block will always be the current epoch boundary and the finalized block the previous epoch boundary.

Note: in ideal conditions, $k = 1$, but the paper allows for $k > 1$ to account for network latency and other implementation issues. Note that $k > 1$ only implies that the justifications could skip blocks, for example: supermajority could vote to go from $B_0 \rightarrow B_2$ (this is $k = 2$).

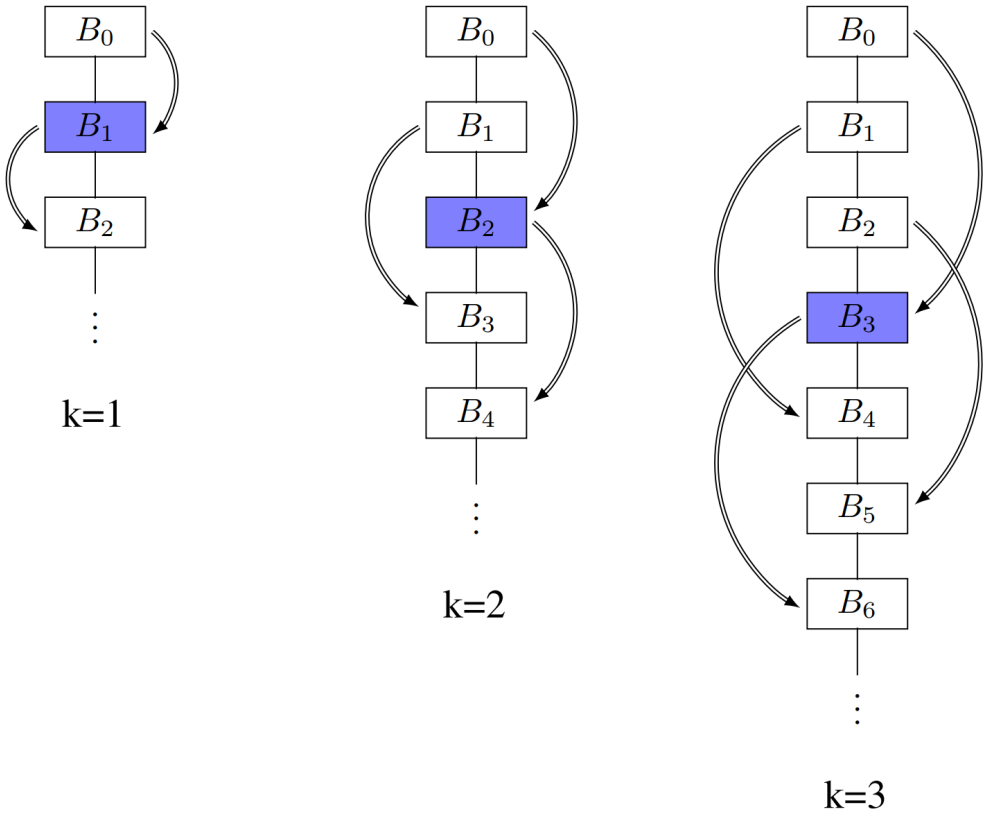

Figure 6: Illustrative examples of Definition 4.9 for $k = 1, 2, 3$, respectively. Double arrows are supermajority links. The blue block is the one being finalized in each case.

LMD GHOST

The Fork Choice Rule. It has many *ifs* and *buts* which may be read on the paper, but the idea is that presented with a fork, the chain with the most *validations* wins.

Slashing conditions

The slashing conditions are only two, and are quite simple to understand:

1. No validator makes two distinct attestations α_1 and α_2 with $ep(\alpha_1) = ep(\alpha_2)$. Meaning: no validator attests two blocks in the same epoch!
2. No validator makes two distinct attestations α_1 and α_2 with $aep(LJ(\alpha_1)) < aep(LJ(\alpha_2)) < aep(LE(\alpha_1)) < aep(LE\alpha_2)$.

Condition (1) is fairly intuitive: validators are only required to attest once per epoch (committees are formed one per epoch). It is also equivalent to saying $aep(LE(\alpha_1)) = aep(LE(\alpha_2))$. Remember aep is the *attestation boundary epoch* and LE is the last epoch boundary of a block.

Condition (2) is more complicated. The paper goes a bit more into details specially regarding why an honest validator can't incur in this behavior. *I think* the intuition is that this asks

validators on epochs $r < s < t < u$ to be consistent: if they voted to transition from $(B_2, s) \to (B_3, t)$ they can now not vote for $(B_1, r) \to (B_4, u)$. They must have been lying at some point! Why did they vote for the first one knowing that later they would vote for the second one?