A Philosophy of Blockchain Validation

2020 Aug 17 See all posts

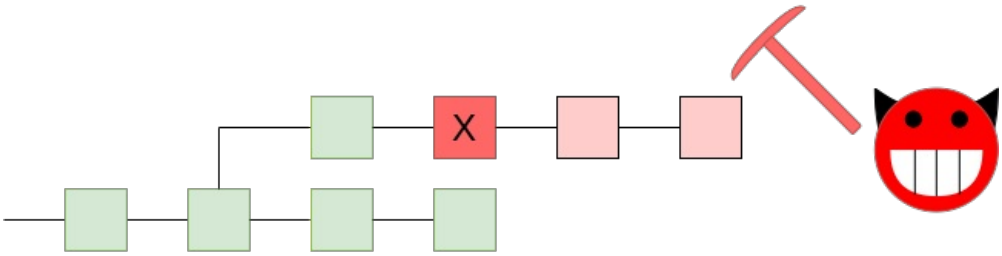
See also:

- A Proof of Stake Design Philosophy
- The Meaning of Decentralization
- Engineering Security through Coordination Problems

One of the most powerful properties of a blockchain is the fact that every single part of the blockchain's execution can be independently validated. Even if a great majority of a blockchain's miners (or validators in PoS) get taken over by an attacker, if that attacker tries to push through invalid blocks, the network will simply reject them. Even those users that were not verifying blocks at that time can be (potentially automatically) warned by those who were, at which point they can check that the attacker's chain is invalid, and automatically reject it and coordinate on accepting a chain that follows the rules.

But how much validation do we actually need? Do we need a hundred independent validating nodes, a thousand? Do we need a culture where the average person in the world runs software that checks every transaction? It's these questions that are a challenge, and a very important challenge to resolve especially if we want to build blockchains with consensus mechanisms better than the single-chain "Nakamoto" proof of work that the blockchain space originally started with.

Why validate?



A 51% attack pushing through an invalid block. We want the network to reject the chain!

There are two main reasons why it's beneficial for a user to validate the chain. First, it maximizes the chance that the node can correctly determine and say on the **canonical chain** - the chain that the community accepts as legitimate. Typically, the canonical chain is defined as something like "the valid chain that has the most miners/validators supporting it" (eg. the "longest valid chain" in Bitcoin). Invalid chains are rejected by definition, and if there is a choice between multiple valid chains, the chain that has the most support from miners/validators wins out. And so if you have a node that verifies all the validity conditions, and hence detects which chains are valid and which chains are not, that maximizes your chances of correctly detecting what the canonical chain actually is.

But there is also another deeper reason why validating the chain is beneficial. Suppose that a powerful actor tries to push through a change to the protocol (eg. changing the issuance), and has the support of the majority of miners. If no one else validates the chain, this attack can very easily succeed: everyone's clients will, *by default*, accept the new chain, and by the time anyone sees what is going on, it will be *up to the dissenters* to try to coordinate a rejection of that chain. But if average users are validating, then the coordination problem falls on the other side: it's now the responsibility of whoever is trying to change the protocol to convince the users to actively download the software patch to accept the protocol change.

If enough users are validating, then **instead of defaulting to victory, a contentious attempt to force a change of the protocol will default to chaos**. Defaulting to chaos still causes a lot of

disruption, and would require out-of-band social coordination to resolve, but it places a much larger barrier in front of the attacker, and makes attackers much less confident that they will be able to get away with a clean victory, making them much less motivated to even try to start an attack. If *most* users are validating (directly or indirectly), and an attack has *only* the support of the majority of miners, then the attack will outright **default to failure** - the best outcome of all.

The definition view versus the coordination view

Note that this reasoning is very different from a different line of reasoning that we often hear: that a chain that changes the rules is somehow "by definition" not the correct chain, and that no matter how many other users accept some new set of rules, what matters is that you personally can stay on the chain with the old rules that you favor.

Here is one example of the "by definition" perspective from Gavin Andresen:

> I'd like to propose this big-picture technical definition of "Bitcoin":
>
> > "Bitcoin" is the ledger of not-previously-spent, validly signed transactions contained in the chain of blocks that begins with the genesis block (hash 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f), follows the 21-million coin creation schedule, and has the most cumulative double-SHA256-proof-of-work.[1]

Here's another from the Wasabi wallet; this one comes even more directly from the perspective of explaining why full nodes are valuable:

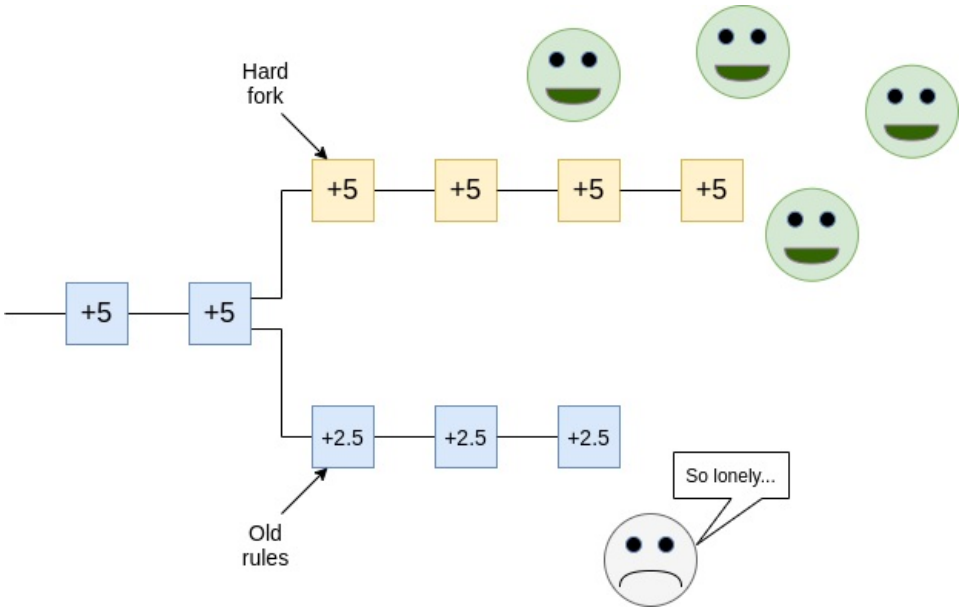
> When running a Bitcoin full node, you define the precise monetary rules that you voluntarily agree on. Nobody else forces this choice upon you. Thus any sovereign individual who wants to claim financial independence must run a full node. Once your own rules are firmly established, your software discovers other nodes in the Bitcoin peer-to-peer network which do not break your rules. These peers send you transactions and blocks which are valid according to their set of rules, and you verify for yourself if they are also correct for you. If one of the proposed transactions breaks your own rules, then you mark it as invalid, disconnect from and ban the node who sent you the malicious transaction.
>
> > **Claim your monetary sovereignty**
> >
> > With your full node you define, verify, and enforce the rules of your sound money Bitcoin.

Notice two core components of this view:

1. A version of the chain that does not accept the rules that you consider fundamental and non-negotiable is *by definition* not bitcoin (or not ethereum or whatever other chain), not matter how many other people accept that chain.
2. What matters is that *you* remain on a chain that has rules that *you* consider acceptable.

However, I believe this "individualist" view to be very wrong. To see why, let us take a look at the scenario that we are worried about: the vast majority of participants accept some change to protocol rules that you find unacceptable. For example, imagine a future where transaction fees are very low, and to keep the chain secure, almost everyone else agrees to change to a new set of rules that increases issuance. You stubbornly keep running a node that continues to enforce the old rules, and you fork off to a different chain than the majority.

From your point of view, you still have your coins in a system that runs on rules that you accept. But so what? Other users will not accept your coins. Exchanges will not accept your coins. Public websites may show the price of the new coin as being some high value, but they're referring to the coins on the majority chain; *your* coins are valueless. Cryptocurrencies and blockchains are fundamentally social constructs; without other people believing in them, they mean nothing.



So what is the alternative view? The core idea is to look at blockchains as **engineering security through coordination problems**.

Normally, coordination problems in the world are a bad thing: while it would be better for most people if the English language got rid of its highly complex and irregular spelling system and made a phonetic one, or if the United States switched to metric, or if we could immediately drop all prices and wages by ten percent in the event of a recession, in practice this requires everyone to agree on the switch at the same time, and this is often very very hard.

With blockchain applications, however, *we are using coordination problems to our advantage*. We are using the friction that coordination problems create as a bulwark against malfeasance by centralized actors. We can build systems that have property X, and we can guarantee that they will preserve property X because changing the rules from X to not-X would require a whole bunch of people to agree to update their software at the same time. Even if there is an actor that could force the change, doing so would be hard - much much harder than it would be if it were instead the responsibility of *users* to actively coordinate dissent to resist a change.

Note one particular consequence of this view: it's emphatically *not* the case that the purpose of your full node is just to protect *you*, and in the case of a contentious hard fork, people with full nodes are safe and people without full nodes are vulnerable. Rather, the perspective here is much more one of **herd immunity**: the more people are validating, the more safe *everyone* is, and even if only some portion of people are validating, everyone gets a high level of protection as a result.

Looking deeper into validation

We now get to the next topic, and one that is very relevant to topics such as light clients and sharding: what are we actually accomplishing by validating? To understand this, let us go back to an earlier point. If an attack happens, I would argue that we have the following preference order over how the attack goes:

default to failure > default to chaos > default to victory

The ">" here of course means "better than". The best is if an attack outright fails; the second best is if an attack leads to confusion, with everyone disagreeing on what the correct chain is, and the worst is if an attack succeeds. Why is chaos so much better than victory? This is a matter of incentives: chaos raises costs for the attacker, and denies them the certainty that they will even win, discouraging attacks from being attempted in the first place. A default-to-chaos environment means

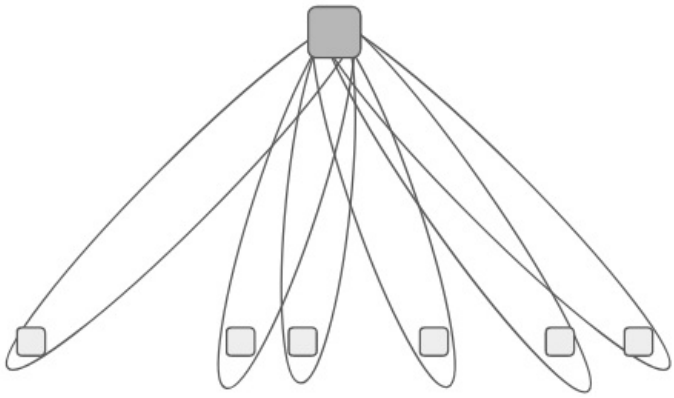
that an attacker needs to win *both* the blockchain war of making a 51% attack *and* the "social war" of convincing the community to follow along. This is much more difficult, and much less attractive, than just launching a 51% attack and claiming victory right there.

The goal of validation is then to move away from default to victory to (ideally) default to failure or (less ideally) default to chaos. If you have a fully validating node, and an attacker tries to push through a chain with different rules, then the attack fails. If some people have a fully validating node but many others don't, the attack leads to chaos. But now we can think: are there other ways of achieving the same effect?

Light clients and fraud proofs

One natural advancement in this regard is **light clients with fraud proofs**. Most blockchain light clients that exist today work by simply validating that the majority of miners support a particular block, and not bothering to check if the other protocol rules are being enforced. The client runs on the trust assumption that the majority of miners is honest. If a contentious fork happens, the client follows the majority chain by default, and it's up to users to take an active step if they want to follow the minority chain with the old rules; hence, today's light clients under attack default to victory. But with fraud proof technology, the situation starts to look very different.

A fraud proof in its simplest form works as follows. Typically, a single block in a blockchain only touches a small portion of the blockchain "state" (account balances, smart contract code....). If a fully verifying node processes a block and finds that it is invalid, they can generate a package (the fraud proof) containing the block along with just enough data from the blockchain state to process the block. They broadcast this package to light clients. Light clients can then take the package and use that data to verify the block themselves, even if they have no other data from the chain.



A single block in a blockchain touches only a few accounts. A fraud proof would contain the data in those accounts along with Merkle proofs proving that that data is correct.

This technique is also sometimes known as *stateless validation*: instead of keeping a full database of the blockchain state, clients can keep only the block headers, and they can verify any block in real time by asking other nodes for a Merkle proof for any desired state entries that block validation is accessing.

The power of this technique is that **light clients can verify individual blocks only if they hear an alarm** (and alarms are verifiable, so if a light client hears a false alarm, they can just stop listening to alarms from that node). Hence, under normal circumstances, the light client is still light, checking only which blocks are supported by the majority of miners/validators. But under those exceptional circumstances where the majority chain contains a block that the light client would not accept, **as long as there is at least one honest node verifying the fraudulent block, that node will see that it is invalid, broadcast a fraud proof, and thereby cause the rest of the network to reject it**.

Sharding

Sharding is a natural extension of this: in a sharded system, there are too many transactions in the system for most people to be verifying directly all the time, but if the system is well designed then any individual invalid block can be detected and its invalidity proven with a fraud proof, and that proof can be broadcasted across the entire network. A sharded network can be summarized as

everyone being a light client. And as long as each shard has some minimum threshold number of participants, the network has herd immunity.

In addition, the fact that in a sharded system block *production* (and not just block *verification*) is highly accessible and can be done even on consumer laptops is also very important. The lack of dependence on high-performance hardware at the core of the network ensures that there is a low bar on dissenting minority chains being viable, making it even harder for a majority-driven protocol change to "win by default" and bully everyone else into submission.

This is what auditability usually means in the real world: not that everyone is verifying everything all the time, but that (i) there are enough eyes on each specific piece that if there is an error it will get detected, and (ii) when an error is detected that fact that be made clear and visible to all.

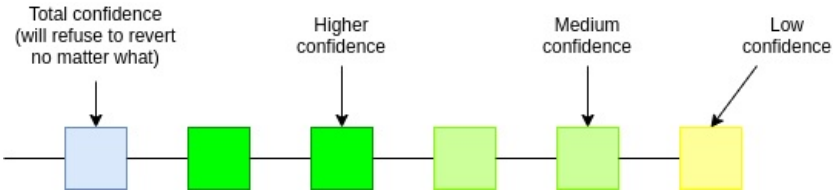
That said, in the long run blockchains can certainly improve on this. One particular source of improvements is ZK-SNARKs (or "validity proofs"): efficiently verifiably cryptographic proofs that allow block producers to prove to clients that blocks satisfy some arbitrarily complex validity conditions. Validity proofs are stronger than fraud proofs because they do not depend on an interactive game to catch fraud. Another important technology is data availability checks, which can protect against blocks whose data is not fully published. Data availability checks do rely on a very conservative assumption that there exists at least some small number of honest nodes somewhere in the network continues to apply, though the good news is that this minimum honesty threshold is low, and does not grow even if there is a very large number of attackers.

Timing and 51% attacks

Now, let us get to the most powerful consequence of the "default to chaos" mindset: 51% attacks themselves. The current norm in many communities is that if a 51% attack wins, then that 51% attack is necessarily the valid chain. This norm is often stuck to quite strictly; and a recent 51% attack on Ethereum Classic illustrated this quite well. The attacker reverted more than 3000 blocks (stealing 807,260 ETC in a double-spend in the process), which pushed the chain farther back in history than one of the two ETC clients (OpenEthereum) was technically able to revert; as a result, Geth nodes went with the attacker's chain but OpenEthereum nodes stuck with the original chain.

We can say that the attack did in fact default to chaos, though this was an accident and not a deliberate design decision of the ETC community. Unfortunately, the community then elected to accept the (longer) attack chain as canonical, a move described by the eth_classic twitter as "following Proof of Work as intended". Hence, *the community norms actively helped the attacker win*.

But we could instead agree on a definition of the canonical chain that works differently: particularly, imagine a rule that once a client has accepted a block as part of the canonical chain, and that block has more than 100 descendants, the client will from then on never accept a chain that does not include that block. Alternatively, in a finality-bearing proof of stake setup (which eg. ethereum 2.0 is), imagine a rule that once a block is finalized it can never be reverted.



5 block revert limit only for illustration purposes; in reality the limit could be something longer like 100-1000 blocks.

To be clear, this introduces a significant change to how canonicalness is determined: instead of clients just looking at the data they receive by itself, clients also look at *when* that data was received. This introduces the possibility that, because of network latency, clients disagree: what if, because of a massive attack, two conflicting blocks A and B finalize at the same time, and some clients see A first and some see B first? But I would argue that this is good: it means that **instead of defaulting to victory, even 51% attacks that just try to revert transactions default to chaos**, and out-of-band emergency response can choose which of the two blocks the chain continues with. If the protocol is well-designed, forcing an escalation to out-of-band emergency response should be very expensive: in proof of stake, such a thing would require 1/3 of validators sacrificing their deposits and getting slashed.

Potentially, we could expand this approach. We could try to make 51% attacks that censor transactions default to chaos too. Research on **timeliness detectors** pushes things further in the direction of attacks of all types defaulting to failure, though a little chaos remains because timeliness detectors cannot help nodes that are not well-connected and online.

For a blockchain community that values immutability, implementing revert limits of this kind are arguably the superior path to take. It is difficult to honestly claim that a blockchain is immutable when no matter how long a transaction has been accepted in a chain, there is always the possibility that some unexpected activity by powerful actors can come along and revert it. Of course, I would claim that even BTC and ETC do *already* have a revert limit at the extremes; if there was an attack that reverted weeks of activity, the community would likely adopt a user-activated soft fork to reject the attackers' chain. But more definitively agreeing on and formalizing this seems like a step forward.

Conclusion

There are a few "morals of the story" here. First, if we accept the legitimacy of social coordination, and we accept the legitimacy of indirect validation involving "1-of-N" trust models (that is, assuming that there exists one honest person in the network somewhere; NOT the same as assuming that one specific party, eg. Infura, is honest), then we can create blockchains that are much more scalable.

Second, client-side validation is extremely important for all of this to work. A network where only a few people run nodes and everyone else really does trust them is a network that can easily be taken over by special interests. But avoiding such a fate does *not* require going to the opposite extreme and having everyone always validate everything! Systems that allow each individual block to be verified in isolation, so users only validate blocks if someone else raises an alarm, are totally reasonable and serve the same effect. But this requires accepting the "coordination view" of *what validation is for*.

Third, if we allow the definition of canonicalness includes timing, then we open many doors in improving our ability to reject 51% attacks. The easiest property to gain is weak subjectivity: the idea that if clients are required to log on at least once every eg. 3 months, and refuse to revert longer than that, then we can add slashing to proof of stake and make attacks very expensive. But we can go further: we can reject chains that revert finalized blocks and thereby protect immutability, and even protect against censorship. Because the network is unpredictable, relying on timing *does* imply attacks "defaulting to chaos" in some cases, but the benefits are very much worth it.

With all of these ideas in mind, we can avoid the traps of (i) over-centralization, (ii) overly redundant verification leading to inefficiency *and* (iii) misguided norms accidentally making attacks easier, and better work toward building more resilient, performant and secure blockchains.