```
BIP: 99
Title: Motivation and deployment of consensus rule changes ([soft/hard]forks)
Author: Jorge Timón <jtimon@jtimon.cc>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0099
Status: Rejected
Type: Informational
Created: 2015-06-20
License: PD
Post-History: http://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-June/008936.html
```

Abstract

This BIP attempts to create a taxonomy of the different types of consensus forks and proposes a deployment mechanism for each of them.

Motivation

The security assumptions of p2p consensus-based systems like Bitcoin are not always well-understood, and the best upgrade mechanisms to the consensus validation rules may vary depending on the type of change being deployed. Discussing such changes without a uniform view on the deployment paths often leads to misunderstandings and unnecessarily delays the deployment of changes.

Definitions

Software fork A copy of an existing project. In free software, this can be done without the permission of the original project's maintainers.

Consensus fork A divergence in the implementation of the verification consensus rules can impede the expected eventual convergence of the network in a single chain that has the most proof of work and also satisfies the rules. This can be intentional or be caused by a bug in consensus validation reimplementations.

Softfork A consensus fork wherein everything that was previously invalid remains invalid while blocks that would have previously considered valid become invalid. A hashrate majority of miners can impose the new rules. They have some deployment advantages like backward compatibility.

Hardfork A consensus fork that makes previously invalid blocks valid. Hardforks require all users to upgrade.

Libconsensus a theoretical piece of software that contains the specifications that define the validity of a block for a given state and chain parameters (ie it may act differently on, for example, regtest).

Libbitcoinconsensus the existing implementation is a library that is compiled by default with Bitcoin Core master and exposes a single C function named

bitcoinconsensus_verify_script(). Although it has a deterministic build and implements the most complex rules (most of the cryptography, which is itself heavily based on libsecp256k1 after #REPLACE_libsecp256k1_PR), it is still not a complete specification of the consensus rules. Since libconsensus doesn't manage the current state but only the validation of the next block given that state, it is known that this long effort of encapsulation and decoupling will eventually finish, and that the person who moves the last line

Taxonomy of consensus forks

Accidental consensus fork

Software forks are very different in nature from consensus rules forks. No software maintainer has special powers over consensus rules changes. There's many good reasons (experimentation, lack of features, independent development, diversity, etc) to fork the Bitcoin Core software and it's good that there's many alternative implementations of the protocol (forks of Bitcoin Core or written from scratch).

But sometimes a bug in the reimplementaion of the consensus validation rules can prevent users of alternative implementation from following the longest (most work) valid chain. This can result in those users losing coins or being defrauded, making reimplementations of the consensus validation rules very risky. Note that a natural language specification of those rules doesn't help since the consensus is not determined by such specification but by the software that the majority of the network runs. That's why "the implementation is the specification".

But Bitcoin Core contains many more things than just consensus validation and it would be unreasonable for all alternative implementations to depend on it. Bitcoin Core should not be the specification. That's why the consensus validation is being separated into a libbitcoinconsensus library with a C API easily accessible from any language. This makes alternative implementations much more secure without burdening them with specific design choices made by Bitcoin Core. It is to be noted that sharing the same code for consensus validation doesn't prevent alternative implementations from independently changing their consensus rules: they can always fork the libbitcoinconsensus project (once it is in a separate repository).

Hopefully libbitcoinconsensus will remove this type of consensus fork which - being accidental - obviously doesn't need a deployment plan.

11/12 March 2013 Chain Fork There is a precedent of an accidental consensus fork at height 225430. Without entering into much detail (see [2]), the situation was different from what's being described from the alternative implementation risks (today alternative implementation still usually rely in different degrees on Bitcoin Core trusted proxies, which is very reasonable considering the lack of a complete libconsensus). The two conflicting consensus validation implementations were two different versions of Bitcoin Core (Bitcoin-qt

at the time): 0.8 against all versions prior to it. Most miners had been fast on upgrading to 0.8 and they were also fast on downgrading to 0.7 as an emergency when they were asked to by the developers community.

A short summary would be that BDB was being abandoned in favor of levelDB, and - at the same time - the miner's policy block size limit was being lift (it was not a consensus rule, not even enforced via softfork). Even after testing, a case where levelDB couldn't correctly validate certain bigger blocks only appeared after deployment in production. Fortunately this was handled very well and rapidly by the whole worldwide community and nobody is unhappy about the solution.

But there's some philosophical disagreements on the terms of what the solution was: we can add a pedantic note on that. If "the implementation is the specification", then those levelDB-specific limitations were part of the consensus rules. Then additional rules were necessary and any alternative implementation (including 0.8) would have to implement it. Then a planned consensus fork to migrate all Bitcoin-qt 0.7- users could remove those additional consensus restrictions. Had libconsensus being implemented without depending on levelDB, those additional restrictions wouldn't have been part of "the specification"

`and this would just have been a bug in the`

consensus rules, just a consensus-critical bug in a set of implementations, concretely all satoshi-bitcoin-0.7-or-less (which happened to be a huge super majority of the users), but other implementations (like libbitcoin) would be free from such bug and implementing the correct libconsensus specification. But since the buggy implementation was a super-majority, the solution would have been to instantly (from a specific block) change the rules to not let the super-majority deviate from the specification and then have another consensus fork to remove them. Two theoretical consensus forks instead of one but the first one deployed practically for free. The practical result would have been identical and only the definitions change. This means discussing something that went uncontroversially well further is "philosophical bike-shed" (TM).

Unilateral softforks

If it is in their best interest of miners to softfork it should be assumed that they may likely enforce it. In some cases, even against the will of a super-majority of users. This is practically an attack on the network and the only solution is to carefully design the incentives so that the case is simply impossible. If that fails, miners should still consider the risk of motivating a schism hardfork before attempting such a consensus fork. A deployment plan for this case is also unnecessary.

Schism hardforks

Fundamental disagreements and controversies are part of social systems, like the one defined as the human participants in the Bitcoin network. Without judging the motivation of the rule discrepancies or what rules were in place first, we're defining schism[1] hardforks as those in which - for whatever reason - users are consiously going to validate 2 different sets of consensus rules. Since they will validate different rulesets, they will end up following 2 different chains for at least some time, maybe forever.

One possible result observed in the past[non_proportional_inflatacoin_fork] is that one of the chains rapidly disappears, but nothing indicates that this must always be the case.

While 2 chains cohexist, they can be considered two different currencies. We could say that bitcoin becomes bitcoinA and bitcoinB. The implications for market capitalization are completely unpredictable,

maybe mc(bitcoinA) = mc(bitcoinB) = mc(old_bitcoin),

maybe mc(bitcoinA) + mc(bitcoinB) = mc(old_bitcoin),

maybe mc(bitcoinA) + mc(bitcoinB) = 1000 * mc(old_bitcoin),

maybe mc(bitcoinA) + mc(bitcoinB) = 0,

...

Schism hardforks have been compared to one type of altcoins called "spinoffs"[spinoffs] that distribute all or part of its initial seigniorage to bitcoin owners at a given block height.

This is very disruptive and hopefully will never be needed. But if it's needed the best deployment path is just to activate the rule changes after certain block height in the future. On the other hand, it is healthy decentralization-wise that many independent software projects are ready to deploy a schism hardfork.

In all of the following examples there's clearly a confrontation that is being resolved using an intentional consensus hardfork.

ASIC-reset hardfork Imagine ASIC production has been consolidated to a single company and distribution is simply not happening: the company is keeping them to mine itself. For that or another reason, a single entity controls 40%+ of the hashrate and there's no hope for an spontaneous improvement in decentralization. Such an untenable centralization could be fixed (with great risks) by switching the hash function used in the proof of work, effectively "pressing the restart button" on the ASIC market. The next function should be simple to implement in ASIC as well so that the market can more easily develop as a healthy and competitive one (as opposed to what the "ASIC-hard" proponents would want), but that's another story...]

Since in this case the confrontation is clearly against the current miners any notion of "miners' voting" is utterly irrelevant.

Anti-Block-creator hardfork There's less extreme cases where changing the pow function would not be necessary. For example, let's imagine a bright future where commoditized ASICs are running in millions home-heaters all over the world, but the block size has been completely removed and the network has devolved to a very centralized system where only 2 big pools have the resources to fully validate full blocks and create block templates with competitive levels of transaction fees. In that case, changing the pow function would be a terrible waste and a risk that could be avoided. A hardfork restoring a block size limit could help fixing this situation. Please don't take it as an argument for or against raising the block size limit: it's just an example. But in this case, again, those 2 big pools would probably be against the fork and, again, their voting is irrelevant.

Like in the previous example, miners are expected to oppose and they have to be ignored.

Anti-cabal hardfork Let's imagine BIP66 had a crypto backdoor that nobody noticed and allows an evil developer cabal to steal everyone's coins. The users and non-evil developers could join, fork libconsensus and use the forked version in their respective bitcoin implementations. Should miner's "vote" be required to express their consent? What if some miners are part of the cabal? In the unlikely event that most miners are part of such an evil cabal, changing the pow function may be required. In other cases, mining "vote" doesn't have much value either since this kind of hardfork would not qualify as uncontroversial anyway.

Uncontroversial consensus upgrades

"Uncontroversial" is something tough to define in this context. What if a single user decides he won't upgrade no matter what and he doesn't even attempt to explain his decision? Obviously, such a user should be just ignored. But what if the circumstances are slightly different? What if they're 2, 10 users? where's the line. It is possible that we can never have a better definition than "I know it when I see it" [citation needed].

Uncontroversial softforks If a majority of miners adopts a softfork, users will follow that chain, even without understanding the new rules. For them is like if blocks are created in a certain way or certain valid transactions are being rejected by miners for some reason. For old nodes it just looks like the new rules are policy rules rather than consensus rules. This greatly reduces the deployment risks, making softforks the preferred consensus rules upgrade mechanism.

The first precedent of a softfork was the introduction of P2SH documented in BIP16. There were competing proposals, but BIP12 had clear disadvantage and

BIP17 was considered a less tested but functionally equivalent version by most of the reviewers. Although it was later discovered that BIP16 had unnecessary limitations and BIP17 is now considered superior, this probably still qualified for our vague concept of "uncontroversial".

At the time, there was no "mining voting" implementation and it was simply deployed using the timestamp of the blocks at some time in the future as the activation trigger. This can't guarantee the assumption that most miners have upgraded before enforcing the new rules and that's why the voting mechanism and first used for BIP30 and BIP66. The current voting threshold for softfork enforcement is 95%. There's also a 75% threshold for miners to activate it as a policy rule, but it should be safe for miners to activate such a policy from the start or later than 75%, as long as they enforce it as consensus rule after 95%.

The current miners' voting mechanism can be modified to allow for changes to be deployed in parallel, the rejection of a concrete softfork without getting locked for the deployment of the next one, and also a more efficient use of the version field in block headers [3]. BIP65 is expected to be deployed with the improved mechanism.

Uncontroversial hardforks Some consensus changes require all participants to upgrade their software before the new rules can be safely activated or they will face serious risk of following the wrong chain and being defrauded. Even if the exact same mechanism used for softforks would be more risky in these cases, that doesn't mean that this type of changes cannot be deployed in an uncontroversial and safe manner.

The simplest approach is to select a block height far enough in the future that everybody has plenty of time to change their software. But if you're aiming for universal adoption, that includes miners' adoption, so it seems reasonable to use a mining voting on top of that. In this case there's only one relevant threshold and it could be different from the softfork one. Probably 100% is too strict, since it would allow a relatively small miner to attack the network and block a consensus upgrade. Something between 99% and 95% is probably a sensible choice for this parameter.

Uncontroversial emergency hardforks Emergency forks may not have time to consult miners and have to be deployed simply by choosing a block height not so far in the future.

But emergency forks could be prepared ahead of time. For example, an intermediary version of software could allow blocks that are double the size of old blocks (after a certain height in the future) while still making miners reject bigger blocks as a softfork rule. Then miners can start the regular process for uncontroversial softfork (or a unilateral softfork if they're a majority) at any point in the future if it is required, and both intermediary and new versions would be prepared for it (which would make deployment much easier). Other

related consensus changes could be deployed in the meantime (say, quadrupling the block size) making the emergency softfork unnecessary.

Code

This BIP is complemented with a concrete code proposal[4] for an uncontroversial hardfork which acts as a precedent and removes the perception that hardforks are impossible in Bitcoin. The deployment of the proposal should not block any other potential hardforks (thus it will required the version bits proposal[3] to be implemented). The change itself doesn't add much complexity to Bitcoin Core and is simple enough that is trivial to apply to diverse implementations (that currently can only use libbitcoinconsensus to validate script-related rules). The change has been already widely tested in many altcoins.

The chosen consensus change is the fix of the timewarp attack discovered and also fixed with a simple patch[5] by @ArtForz. This change has been deployed by most altcoins that made any minimally meaningful change to bitcoin and thus can be considered somewhat tested (in fact, most SHA256d altcoins that didn't implement it have died or being forced to implement it as an emergency hardfork). When deploying this change has been discussed, usually arguments in the lines of "if we get to the point when this matters to bitcoin, we would be already in serious trouble" were used against it. This shouldn't be seen as a disadvantage in this context, since it means we can safely activate the fix very far away in the future (say, 4 years worth of blocks).

Footnotes

[1] https://en.wikipedia.org/wiki/Schism

[2] https://github.com/bitcoin/bips/blob/master/bip-0050.mediawiki

[non_proportional_inflatacoin_fork] TODO missing link

[spinoffs] https://bitcointalk.org/index.php?topic=563972.0

[3] https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki

[4] https://github.com/bitcoin/bitcoin/compare/0.11...jtimon:hardfork-timewarp-0.11

[5] Original references: https://bitcointalk.org/index.php?topic=114751.0 https://bitcointalk.org/index.php?topic=43692.msg521772#msg521772 Rebased patch: https://github.com/freicoin/freicoin/commit/beb2fa54745180d755949470466cbffd1cd6ff14

Attribution

Incorporated corrections and suggestions from: Andy Chase, Bryan Bishop, Btcdrak, Gavin Andresen, Gregory Sanders, Luke Dashjr, Marco Falke.

Copyright