VIP-220: FINALITY WITH ONE BIT (FOB)

Zhijie Ren
VeChain Foundation
zhijie.ren@vechain.com

Ziheng (Peter) Zhou
VeChain Foundation
peter.zhou@vechain.com

February 28, 2022

1 Overview

The Proof-of-Authority consensus algorithm [1], or PoA in short, is a Nakamoto consensus (NC) algorithm that achieves probabilistic consensus [2], which suggests that the consistency of a confirmed block is not definitive, i.e., the probability that another block could replace the confirmed block is never zero. In comparison, Byzantine fault tolerance (BFT) algorithms achieve definitive consistency, i.e., when a block is confirmed by an honest node, it is proven that no other honest node could confirm a conflicting block.

Recently, a mechanism, namely "finality gadget", is proposed in [3] to provide definitive consistency in Nakamoto consensus blockchain in a similar fashion as BFT algorithms. Briefly, a finality gadget guarantees that a block could achieve definitive consensus, namely *finality*, after it is confirmed. Finality is a desirable property for blockchains with a NC algorithm, as it not only suggests a higher security guarantee, but also improves the interoperability between the blockchain and other blockchains or the physical world. More precisely, the proof of a transaction is confirmed on the blockchain with finality will be much simpler than that of a blockchain with probabilistic consensus.

In this VIP, we propose a finality gadget, namely Finality with One Bit (FOB), to the existing PoA mechanism. FOB is based on a state-of-the-art consensus algorithm, Viewless BFT (VLBFT, previously called Leaderless BFT) [4], which we briefly introduced here.

1.1 Brief introduction of VLBFT

VLBFT is a two-phase BFT algorithm in the partial synchronous model. It has a linear view change complexity at the cost of no responsiveness. In this sense, it is similar to Tendermint [5], which is also a two-phase BFT algorithm with linear view change complexity and no responsiveness. However, as it name suggests, VLBFT is view-less, which is different from almost all existing BFT algorithms. This distinctive feature makes it very suitable for a finality gadget in NC blockchains.

Classical BFT algorithms use a so-called "view-lock" mode to deal with asynchronous situations. Briefly, all honest nodes start a new consensus round, namely a "view", and vote for the finality of a block in a normal two-phase process. In particular, they first vote for this block and "lock" to this block, i.e., not voting for any other block to guarantee consistency. Then, once they receive the votes from the supermajority (from more than 2/3 of the number of nodes, including themselves), they cast a commit vote, we call "Com" vote in this document. When a node receives the "Com" votes for a block from the supermajority, he considers this block finalized.

However, in partial synchronous networks, the network delay is bounded by an unknown bound. As a result, if votes from the supermajority of either phase failed be collected by an honest node, the consensus process is stuck and it is impossible for him to tell whether the consensus process could proceed if he waits. Hence, in this case, in order to guarantee liveness, i.e., to make progress, nodes should be able to enter a new view and restart the consensus process, i.e., unlock and be able to start a new round of vote on new blocks once some time-out condition is reached. Moreover, besides the time-out condition, they should also exchange certain information to guarantee that all other nodes are aware of the new view and stop finalizing blocks voted in an older view to guarantee consistency. This is known as the "view change" process.

In VLBFT, there is no hard "lock" rule to prevent nodes from voting and no view and view change process to guarantee the progress of the algorithm. Instead, honest nodes should always vote for the new candidate block in the canonical chain, regardless of whether it is conflicting to his previous votes. However, there is a soft lock rule on "Com" vote: they will not "Com" vote for a block if they have already voted for a conflicting block. This simple soft lock rule guarantees consistency. The liveness condition is not guaranteed by views and view changes, but driven by a underlying NC blockchain: it guarantees liveness when the underlying blockchain extends and there exists a period of time when there is only one block received supermajority votes. Then, depending on how the underlying NC blockchain is designed, VLBFT could function in synchronous networks, practical synchronous networks (will be introduced below), or partial synchronous networks.

1.2 Properties of FOB

Based on the features of VLBFT, FOB could simultaneously achieve the following properties:

Property 1 (Consistency)**.** *If an honest node finalized a block B at height h, then another honest node will not finalize a block B' at height h.*

Property 2 (Liveness)**.** *In practical synchronous model, new blocks could always be finalized.*

The practical synchronous model is defined as the following:

Definition 1 (Practical synchronous network)**.** In practical synchronous networks, the network could switch between synchronous or asynchronous:

- The network will be mostly synchronous, in which the delay is bounded by a known constant;
- The network could be occasionally asynchronous for an arbitrary duration.

Property 3 (Correctness)**.** *The finalized block is proposed by an honest node.*

Consistency, liveness, and correctness properties are the basic properties for a finality gadget, in which the correctness property is stronger than the classical BFT correctness. We will prove the validity of consistency and liveness in Section 3. However, in stead of the normal correctness, in FOB, we achieve a even stronger correctness property, *supermajority correctness*.

Property 4 (Supermajority Correctness)**.** *The finalized block is proposed by an honest node and be at least appended by $4f + 1$ blocks proposed by at least $2f$ nodes. Here, f is the number of adversaries and the number of node n satisfies $n \geq 3f + 1$.*

The supermajority correctness holds by nature as we merge VLBFT into the underlying PoA algorithm and let the votes be collected along with the blocks instead of collected separately. It is able to do so without harnessing the block proposing because there is no lock rule in VLBFT. It is a very desirable property in a finality gadget, which suggests that it is only a gadget to the underlying blockchain to offer finality and should bring minimum interference to the underlying consensus.

Supermajority correctness suggests that the finalized block is almost certainly on the canonical chain. In case that it is not on the canonical chain, it suggests that there exists a fork of at least $4f + 1$ in depth and thus a very extreme network situation. In this case, the underlying NC algorithm is already invalidated. On the contrary, in a blockchain with correctness property, there is no restriction on whether the finalized block is on the canonical (longest) chain, which could cause a long fork and the conflict between the finalized block and the blocks confirmed by the canonical chain rule. As a result, a normally confirmed block could be rolled back by the finality gadget.

Besides, FOB also have the property of block structural simplicity.

Property 5 (Block structural simplicity)**.** *FOB only causes 1 bit per block overhead to the underlying blockchain.*

Since in PoA, a block is implicitly a signed vote from the proposer. Hence, we are able to design FOB and as its name suggests, adding merely one bit in each block to represent whether it is a "Com" vote. So far as we know, this is the minimum overhead amongst all existing finality gadgets.

2 Specifications of FOB

FOB, as a finality gadget, consists of two parts: 1) a new chain selection rule to replace the PoA chain selection rule and 2) a new voting rule for the block proposer to determine either to add the bit "1" or "0" in the block header as his one-bit vote. Before introducing either rules, we introduce some notations and definitions that will be used in FOB.

2.1 Definitions

Firstly, we introduce the following definitions and properties of a chain that.

1. Chain: As a chain could be uniquely identified by its last block B, we denoted such a chain as $C(B)$.
2. Checkpoint: The blocks with block height $h(B) \equiv 1 \pmod{\Omega}$ is a checkpoint, where Ω is a predetermined parameter.
3. A justified checkpoint (JC): a checkpoint B with height $h(B)$ that has more than 67 distinctive leaders in the blocks of height $[h(B), \min(B^*, h(B) + \Omega - 1)]$ is a JC. Here, B^* is the newest block on the chain. Further, we define $\mathcal{JC}_{C(B^*)}$ as an ordered set of all JCs of chain $C(B^*)$, starting from the first JC.
4. JC height (JCH): the JCH of a block B, denoted by $H(B)$, is defined as $H(B) = |\mathcal{JC}_{C(B)}|$.
5. High JC: the JC with the largest JCH on chain $C(B)$, denoted by $\mathrm{HighJC}_{C(B)}$, i.e., $\mathrm{HighJC}_{C(B)}$ is the last element in $\mathcal{JC}_{C(B)}$.
6. Com JC: A Com JC is a JC that has more than 67 distinctive leaders that vote 1 in the "Com" vote vector (will be explained later) in the blocks of height $[h(B), \min(B^*, h(B) + \Omega - 1)]$. Here, B^* is the newest block. Further, we define $\mathcal{COM}_{C(B^*)}$ as an ordered set of all Com JCs of chain $C(B^*)$, starting from the first Com JC.
7. High Com: $\mathrm{HighCom}_{C(B)}$ is the last element in $\mathcal{COM}_{C(B^*)}$.

Clearly, given any blockchain $C(B)$, a node can derive all checkpoints, $\mathcal{JC}_{C(B)}$, $\mathrm{HighJC}_{C(B)}$, and $\mathrm{HighCom}_{C(B)}$ from $C(B)$.

In each block B, an additional piece of information, called *"Com" vote vector*, $v(B) \in \{0, 1\}$ is also included.

2.2 Chain selection rule

We now introduce our new chain selection rule when an honest leader u observes a set of valid chains. Let's use the notation $C(B) \perp C(B')$ for two chains that are not equal and not a subset of each other. Then, $B \perp B'$ if $C(B') \perp C(B)$.

- Choose the chain with $\mathrm{HighCom}_{C(B)}$ of the largest JCH. Then, set Commit_u as the previous JC of $\mathrm{HighCom}_{C(B)}$, i.e., the $(H(\mathrm{HighCom}_{C(B)}) - 1)$-th element in $\mathcal{JC}_{C(B)}$. Commit_u and all blocks before it are considered as "final" for node u.
- Discard all chains $C(B)$ that have $C(B) \perp \mathrm{Commit}_u$.
- If there are multiple chains left, choose the chain $C(B)$ with the largest $H(B)$.
- If there are multiple chains left, choose the chain according to the original canonical chain rule.

Here we specify the chain selection rules by pseudocodes in Algorithm 1.

Algorithm 1 Node u selection the canonical chain in round r.

1: Get the set of all valid chains from the buffer $\mathcal{C}_t$.
2: Get a chain $C^* \in \mathcal{C}_t$ with $\mathrm{HighCom}_{C^*}$ of the largest JCH.
3: Update Commit_u with the $(H(\mathrm{HighCom}_{C^*}) - 1)$-th element in $\mathcal{JC}_{C^*}$.
4: Update $\mathcal{C}_t$ by removing all chains that conflict Commit_u.
5: $\mathcal{C}_{\text{temp}} \leftarrow \mathcal{C}_t$.
6: Discards all chains from $\mathcal{C}_{\text{temp}}$ that do not have the largest JCH.
7: Choose the canonical chain C from $\mathcal{C}_{\text{temp}}$ according to the original canonical chain rule.

2.3 Voting rule

With the selected chain $C(B)$, when a node u turns to propose a block B', he uses one simple rule to determine his vote: u sets $v(B) = 0$ if he has proposed a block B'' with $H(B'') \geq H(\mathrm{HighJC}_{C(B)}) - 1$ and $B'' \perp \mathrm{HighJC}_{C(B)}$. Otherwise, $v(B) = 1$. The pseudocode of the voting rule is given in Algorithm 2.

Algorithm 2 The leader of round r, $l_r = u$ determines his vote in block B.

Get the latest block on the canonical chain B_0 and the canonical chain $C(B_0)$ according to Algorithm 1.
Get the set of non-expired proposed blocks $\mathcal{P}$.
Get the parameters $\text{HighJC}_{C(B_0)}$ and $H(\text{HighJC}_{C(B_0)})$.
if there exists $B' \in \mathcal{P}$ such that $H(B') \geq H(\text{HighJC}_{C(B_0)})$ and $B' \perp \text{HighJC}_{C(B_0)}$ **then**
    $v(B) \leftarrow 0$
else
    $v(B) \leftarrow 1$
end if
Add B to set $\mathcal{P}$
Update $\mathcal{P}$ by removing all B that conflicts Commit_u

3 The validity of FOB

In this section, we prove the validity of FOB, i.e., the consistency and liveness properties. The other two properties, supermajority correctness and block structural simpleness, are hold by design.

3.1 Consistency

Theorem 1. *If an honest node i has $\text{Commit}_i = B$ and another honest node j has $\text{Commit}_j = B'$, then it is not possible that $B \perp B'$.*

Proof. We firstly define a notation $B \prec B'$ to represent a block B is on the same chain and before B'. If there exists two honest nodes i and j such that $\text{Commit}_i \perp \text{Commit}_j$, then by the chain selection rule, there must exist two chains $C(B_1) : \text{Commit}_i \in C(B_1)$ and $C(B_1') : \text{Commit}_j \in C(B_1')$ with $\text{Commit}_i \prec \text{HighCom}_{C(B_1)}, \text{Commit}_j \prec \text{HighCom}_{C(B_1')}, \text{Commit}_i \perp \text{HighCom}_{C(B_1')}$, $\text{Commit}_j \perp \text{HighCom}_{C(B_1)}$, and $\text{HighCom}_{C(B_1)} \perp \text{HighCom}_{C(B_1')}$. W.l.o.g. we assume that $H(\text{HighCom}_{C(B_1)}) \leq H(\text{HighCom}_{C(B_1')})$.

There must exist an honest node u that votes 1 in the "Com" vote after both checkpoints $\text{HighCom}_{C(B_1)}$ and $\text{HighCom}_{C(B_1')} \perp \text{HighCom}_{C(B_1)}$. Then, by the rule of Com voting, we have $H(\text{HighCom}_{C(B_1)}) \leq H(\text{HighCom}_{C(B_1')}) - 2$.

Now, we consider a JC on chain $C(B_1')$ with height $H(\text{HighCom}_{C(B_1)}) - 1$, denoted by B_0', which exists since there are at least $H(\text{HighCom}_{C(B_1')})$ JCs on chain $C(B_1')$.

We first show that $B_0' \perp \text{HighCom}_{C(B_1)}$: if $B_0' \in C(\text{HighCom}_{C(B_1)})$ and then $B_0' \in C(B_1)$; note that $\text{Commit}_i \in C(B_1), H(\text{Commit}_i) < H(\text{HighCom}_{C(B_1)}) \leq H(\text{HighCom}_{C(B_1')})$; hence, $\text{Commit}_i \in C(B_0') \subset C(B_1')$. Then, note that $\text{Commit}_j \in C(B_1')$. Hence, Commit_i and Commit_j are on the same chain which contradicts our assumption $\text{Commit}_i \perp \text{Commit}_j$.

Then, since $B_0' \perp \text{HighCom}_{C(B_1)}$, by the definition of JC, there must exist an honest node v that votes for B_0', i.e., proposed a block in height $h(B_0')$ to $h(B_0') + \Omega - 1$, and votes 1 in "Com" vote for $\text{HighCom}_{C(B_1)}$. However, by the rule of "Com" voting, he cannot first vote for B_0' with $H(B_0') = H(\text{HighCom}_{C(B_1)}) - 1$ and $B_0' \perp \text{HighCom}_{C(B_1)}$ then "Com" vote for $\text{HighCom}_{C(B_1)}$. Then, by the chain selection rule, he cannot propose block on chain $C(B_1')$ with JCH $H(\text{HighCom}_{C(B_1)}) - 1$ when he has already know the chain $C(B_1)$ with height $H(\text{HighCom}_C(B_1))$.

□

3.2 Liveness

Theorem 2. *New blocks can always be finalized if Ω is set properly.*

Proof. First, new blocks could always be proposed and justified since there is no "lock" rule that prevent nodes from proposing blocks on the canonical chain. Hence, giving that the underlying chain follows the chain growth and chain

quality properties of NC, new blocks can always been proposed and justified if Ω is set such that 67 signatures could be collected for a checkpoint with high probability.

Then, by the common prefix property, with high probability, a JC is the only JC at its JCH.

Last but not least, we show that a checkpoint can be committed by receiving 67 Com vote with non-zero probability. Unlike PoW consensus, in PoA, there is a unique legitimate block in case that multiple blocks are proposed in the same round and the proposers of the legitimate block of each round is uniformly distributed. Then, in normal situation, adversaries could neither prevent honest nodes from proposing blocks in their rounds nor prevent other honest nodes receive their blocks. As a result, the adversaries could not create a fork that has a conflicting justified checkpoint. They could only create a conflicting checkpoint and misguide some honest nodes to vote for it.

In fact, they can collude and use a specific strategy to make a few honest nodes to vote for a conflicting checkpoint when they coincidentally lead in several consecutive rounds after the checkpoint. However, as soon as honest nodes take turns, all honest nodes will soon only vote for the canonical chain. Then, in this very rare situation, a few honest nodes will "Com" vote 0 in next JCH.

Otherwise, all honest nodes will "Com" vote 1, which will accumulate 67 "Com" vote with non-zero probability if Ω is chosen properly, even if all adversaries vote 0. Furthermore, as it is also in the adversaries interests to receive block reward and finalize the incentives that they received, it is also more profitable for them to "Com" vote 1 instead of 0.

3.3 Determining Ω

As shown in Subsection 3.2, the key of liveness is the selection of Ω. More precisely, it should be chosen large enough so that $2f + 1$ distinctive signatures could be collected with non-zero probability in the normal situation. However, for practicality, Ω should be also chosen large enough that $2f + 1$ distinctive signatures could be collected with high probability.

Here, we choose $\Omega = 180$, which is roughly 30 minutes in VeChain. We can calculate the probability of 78 signatures can be collected in 180 blocks:

$$P_{180} = \sum_{x=78}^{1} 01 \binom{101}{x} p^x (1-p)^{101-x} = 95.76\%. \tag{1}$$

Here, p is the probability for the vote of a node is in the 180 rounds, which is calculated by

$$p_{180} = 1 - \frac{100}{101}^{1} 80. \tag{2}$$

We thus consider 180 is a very secure assumption for liveness since 67 signagures could be collected for each checkpoint with more than 95% chance given that no length-12 forks exists, which is guaranteed by the PoA algorithm.

Optimistically, we can set $\Omega = 132$, which is roughly 22 minutes in VeChain according to the practical fork situation in VeChain. With almost no forks, 67 signatures could be collected for each checkpoint with 94.76% chance. Moreover, with 20.74% chance, 78 signatures could be collected, suggesting that the liveness could still be guaranteed as long as the security of PoA is guaranteed.

□

4 Comparison

4.1 PBFT and Tendermint

As mentioned in Section 1.1, similar to PBFT [6] and Tendermint [5], VLBFT is a two-phase BFT algorithm in partial synchronous model. Then, performance wise, it is more similar to Tendermint as both algorithms has a linear view change message complexity, at the cost of losing optimal responsiveness while PBFT has optimal responsiveness with $O(n^3)$ view change message complexity. On the other hand, in the mechanism perspective, VLBFT is unique of its kind that relies on a underlying NC algorithm to guarantee liveness and does not have a view change mechanism.

FOB is a finality gadget in NC blockchains and it functions in practical synchronous network. Hence, it is not fair to directly compare FOB to PBFT and Tendermint in performance. In [7], BFT algorithms such as PBFT and Tendermint can be directly used in the finality problem of NC algorithms in practical synchronous networks with both consistency and liveness guaranteed. However, there will be a rather long recovery period after the asynchronous period.

4.2 Grandpa

Grandpa is the finality gadget of Polkadot, which also yields a two-phase BFT consensus with linear view change complexity and no optimal responsiveness. In Grandpa, the votes are voted and collected separately instead of collected along with the block. Comparing to FOB, Grandpa might have an advantage in the efficiency of collection votes at the cost of possible long forks as explained in Subsection 1.2. Moreover, it requires an additional incentive mechanism to encourage voting, which is not included in [8].

Then, Grandpa considers the partial synchronous network model instead of the practical synchronous model as we consider. In other words, both Grandpa and FOB guarantee strong consistency in asynchronous situations, e.g., if the network is partitioned. The major difference is that FOB do not guarantee liveness when the network suffers from delays larger than a predetermined threshold, while Grandpa could guarantee liveness in this situation. However, we believe that the liveness of the finality gadget is not the priority in this situation since the underlying NC algorithm only works in synchronous situation. Hence, if such dire situation happens, the chain could stop from growing and there is no new block to be finalized anyway.

4.3 Casper

Casper [3] is the finality mechanism proposed to use for the Proof-of-Stake (PoS) chain of Ethereum. It considers a similar problem as our proposal with a similar network assumption. Moreover, Casper also follows the two-phase BFT paradigm and has similar design like "checkpoint" in a fixed amount of block heights.

The difference between Casper and FoB is the folloiwng: First, Casper also use a separate voting scheme just as Grandpa. Then, essentially, Casper has a "lock" rule, but with no "view change" rule. Hence, there is a possibility of "deadlock" when the votes of honest nodes are split on multiple blocks and none of this block could be finalized to make progress. In this scenario, Casper falls back to the PoS leader selection: the nodes locked to a chain which is not the canonical chain will lose incentive (either because they cannot participate and receive rewards or being slashed) and eventually lose their positions as consensus nodes. As a result, the liveness of Casper is guaranteed once more than 2/3 of the consensus nodes are not locked. On the other hand, there is no such problem in FOB as it is not hard lock mechanism in VLBFT as well as FOB.

References

[1] Vechain development plan and whitepaper. `https://cdn.vechain.com/vechainthor_development_plan_and_whitepaper_en_v1.0.pdf`, 2018.

[2] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Annual International Cryptology Conference*, pages 291–323. Springer, 2017.

[3] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[4] Jianyu Niu and Chen Feng. Leaderless byzantine fault tolerant consensus. *arXiv preprint arXiv:2012.01636*, 2020.

[5] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

[6] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[7] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 446–465. IEEE, 2021.

[8] Alistair Stewart and Eleftherios Kokoris-Kogia. Grandpa: a byzantine finality gadget. *arXiv preprint arXiv:2007.01560*, 2020.