Another Look at CBC Casper Consensus Protocol

Yongge Wang
UNC Charlotte

March 29, 2020

Abstract

Ethereum Research team has proposed a family of Casper blockchain consensus protocols. It has been shown in the literature that the Casper Friendly Finality Gadget (Casper FFG) cannot achieve liveness property in partially synchronous networks such as the Internet environment. The "Correct-by-Construction" family of Casper blockchain consensus protocols (CBC Casper) has been proposed as a finality gadget for the future Proof-of-Stake (PoS) based Ethereum blockchain. Unfortunately, no satisfactory/constructive finality rules have been proposed for CBC Casper and no satisfactory liveness property has been obtained for CBC Casper. Though it is commonly/widely believed in the community that CBC Casper could not achieve liveness property in asynchronous networks, this paper provides a surprising result by proposing the first CBC Casper protocol that achieves liveness property against $t = \lfloor \frac{n}{5} \rfloor$ Byzantine participants in completely asynchronous networks.

1 Introduction

Consensus is hard to achieve in open networks such as partial synchronous networks. Several practical protocols such as Paxos [10] and Raft [13] have been designed to tolerate $\lfloor \frac{n-1}{2} \rfloor$ non-Byzantine faults. For example, Google, Microsoft, IBM, and Amazon have used Paxos in their storage or cluster management systems. Lamport, Shostak, and Pease [11] and Pease, Shostak, and Lamport [14] initiated the study of reaching consensus in face of Byzantine failures and designed the first synchronous solution for Byzantine agreement. For asynchronous networks, Fischer, Lynch, and Paterson [8] showed that there is no deterministic protocol for the BFT problem in face of a single failure. Several researchers have tried to design BFT consensus protocols to circumvent the impossibility. The first category of efforts is to use a probabilistic approach to design BFT consensus protocols in completely asynchronous networks. This kind of work was initiated by Ben-Or [2] and Rabin [15] and extended by others such as Cachin, Kursawe, and Shoup [5]. It should be noted that though probabilistic approach was used to design BFT protocols in asynchronous networks, some researchers used probabilistic approach to design BFT protocols for complete synchronous networks. For example, the probabilistic approach based BFT protocols [7, 12] employed in ALGORAND blockchain [9] assumes a synchronous and complete point-to-point network. The second category of efforts was to design BFT consensus protocols in partial synchronous networks which was initiated by Dwork, Lynch, and Stockmeyer [6].

Ethereum foundation has tried to design a BFT finality gadget for their Proof of Stake (PoS) based Ethereum blockchain. It has been shown in Wang [17] that their first design Casper Friendly Finality Gadget (Casper FFG) [4] does not achieve liveness property in partially asynchronous networks. Recently, Ethereum foundation has been advocating the "Correct-by-Construction" (CBC) family of Casper blockchain consensus protocols [18, 19]. The CBC Casper the Friendly Ghost emphasizes the safety property. But it does not try to address the liveness requirement for the consensus process. Indeed, it explicitly says that [18] "*liveness considerations are considered largely out of scope, and should be treated in future work*". Thus in order for CBC Casper to be deployable, a lot of work needs to be done since the Byzantine Agreement Problem becomes challenging only when both safety and liveness properties are required to be satisfied at the same time. It is simple to design BFT protocols that only satisfy one of the properties. The Ethereum foundation community has made several efforts to design safety oracles for CBC Casper to help participants to make a decision when an agreement is reached (see, e.g., [16]). However, this problem is generally at least as hard as coNP-complete problems. So no satisfactory solution has been proposed yet.

CBC Casper has received several critiques from the community. For example, Ali et al [1] concluded that "*the definitions and proofs provided in [19] result in neither a theoretically sound nor practically useful treatment of*

Byzantine fault-tolerance....Importantly, it remains unclear if the definition of the Casper protocol family provides any meaningful safety guarantees for blockchains. Though CBC Casper is not a complete deployable solution yet and it has several fundamental issues yet to be addressed, we think these critiques as in [1] may not be fair enough. Indeed, CBC Casper provides an interesting framework for consensus protocol development. In particular, the algebraic approach proposed by CBC Casper has certain advantages for describing Byzantine Fault Tolerance (BFT) protocols. The analysis in this paper shows that efficiently constructive liveness concepts for CBC Casper could be obtained even in a complete asynchronous network.

For the network setting, we assume a complete asynchronous network of Fischer, Lynch, and Paterson [8]. That is, we make no assumptions about the relative speeds of processes or about the delay time in delivering a message. We also assume that processes do not have access to synchronized clocks, so algorithms based on time-outs cannot be used. However, we assume that all messages are eventually delivered if the sender makes infinitely trials to send the messages.

The structure of the paper is as follows. Section 2 provides a brief review of the CBC Casper framework. The author of [18] mentioned in several talks that CBC Casper does not guarantee liveness in asynchronous networks. Section 3 presents a surprising result which shows that CBC Casper can INDEED provide liveness property in asynchronous networks. The solution in Section 3 employs finality rules for CBC Casper protocol by leveraging the underlying ideas within Ben-Or's seminal probabilistic BFT protocol. By integrating Ben-Or's protocol in CBC Casper framework, we are able to improve the performance of Ben-Or's protocol from exponential steps to linear steps.

2 CBC Casper the Friendly Binary Consensus (FBC)

In this paper, we only consider Casper the Friendly Binary Consensus (FBC). Our discussion can be easily extended to general cases. For the Casper FBC protocol, each participant repeatedly sends and receives messages to/from other participants. Based on the received messages, a participant can infer whether a consensus has been achieved. Assume that there are n participants $P_1, \cdots, P_n$ and let $t < n$ be the Byzantine-fault-tolerance threshold. The protocol proceeds from step to step (starting from step 0) until a consensus is reached. Specifically the step s proceeds as follows:

- Let $\mathcal{M}_{i,s}$ be the collection of valid messages that P_i has received from all participants until step s. P_i determines whether a consensus has been achieved. If a consensus has not been achieved yet, P_i sends the message

$$m_{i,s} = \langle P_i, e_{i,s}, \mathcal{M}_{i,s} \rangle \tag{1}$$

  to all participants where $e_{i,s}$ is P_i's estimated consensus value based on the received message set $\mathcal{M}_{i,s}$.

In the following, we describe how a participant P_i determines whether a consensus has been achieved and how a participant P_i calculates the value $e_{i,s}$ from $\mathcal{M}_{i,s}$.

For a message $m = \langle P_i, e_{i,s}, \mathcal{M}_{i,s} \rangle$, let $J(m) = \mathcal{M}_{i,s}$. For two messages m_1, m_2, we write $m_1 \prec m_2$ if m_2 depends on m_1. That is, there is a sequence of messages $m'_1, \cdots, m'_v$ such that

$$\begin{array}{c} m_1 \in J(m'_1) \\ m'_1 \in J(m'_2) \\ \cdots \\ m'_v \in J(m_2) \end{array}$$

For a message m and a message set $\mathcal{M} = \{m_1, \cdots, m_v\}$, we say that $m \prec \mathcal{M}$ if $m \in \mathcal{M}$ or $m \prec m_j$ for some $j = 1, \cdots, v$. The *latest message* $m = L(P_i, \mathcal{M})$ by a participant P_i in a message set $\mathcal{M}$ is a message $m \prec \mathcal{M}$ satisfying the following condition:

- There does not exist another message $m' \prec \mathcal{M}$ sent by participant P_i with $m \prec m'$.

It should be noted that the "latest message" concept is well defined for a participant P_i if P_i has not equivocated, where a participant P_i equivocates if P_i has sent two messages $m_1 \neq m_2$ with the properties that "$m_1 \not\prec m_2$ and $m_2 \not\prec m_1$".

For a binary value $b \in \{0,1\}$ and a message set $\mathcal{M}$, the score of a binary estimate for b is defined as the number of non-equivocating participants P_i whose latest message voted for b. That is,

$$\texttt{score}(b, \mathcal{M}) = \sum_{L(P_i,\mathcal{M})=(P_i,b,*)} \lambda(P_i, \mathcal{M})$$

where

$$\lambda(P_i, \mathcal{M}) = \begin{cases} 0 & \text{if } P_i \text{ equivocates in } \mathcal{M}, \\ 1 & \text{otherwise.} \end{cases}$$

To estimate consensus value: Now we are ready to define P_i's estimated consensus value $e_{i,s}$ based on the received message set $\mathcal{M}_{i,s}$ as follows:

$$e_{i,s} = \begin{cases} 0 & \text{if } \texttt{score}(0, \mathcal{M}_{i,s}) > \texttt{score}(1, \mathcal{M}_{i,s}) \\ 1 & \text{if } \texttt{score}(1, \mathcal{M}_{i,s}) > \texttt{score}(0, \mathcal{M}_{i,s}) \\ b & \text{otherwise, where } b \text{ is coin-flip output} \end{cases} \tag{2}$$

To infer consensus achievement: For a protocol execution, it is required that for all i, s, the number of equivocating participants in $\mathcal{M}_{i,s}$ is at most t. A participant P_i determines that a consensus has been achieved at step s with the received message set $\mathcal{M}_{i,s}$ if there exists $b \in \{0,1\}$ such that

$$\forall s' > s : \texttt{score}(b, \mathcal{M}_{i,s'}) > \texttt{score}(1-b, \mathcal{M}_{i,s'}). \tag{3}$$

3 Liveness of CBC Casper FBC

From CBC Casper protocol description, it is clear that CBC Casper is guaranteed to be safe against equivocating participants. However, the "inference rule for consensus achievement" requires a mathematical proof based on infinitely many message sets $\mathcal{M}_{i,s'}$ for $s' > s$. This requires each participant to verify that for each potential set of t Byzantine participants, their malicious activities will not be able to overturn the inequality in (3). This problem is at least co-NP hard. Thus even if the system reaches a consensus, the participants may not realize this fact. In order to address this challenge, Ethereum community provides three "safety oracles" (see [16]) to help participants to determine whether a consensus is obtained. The first "adversary oracle" simulates some protocol execution to see whether the current estimate will change under some Byzantine attacks. As mentioned previously, this kind of problem is co-NP hard and the simulation cannot be exhaustive generally. The second "clique oracle" searches for the biggest clique of participant graph to see whether there exist more than 50% participants who agree on current estimate and all acknowledge the agreement. That is, for each message, the oracle checks to see if, and for how long, participants have seen each other agreeing on the value of that message. This kind of problem is equivalent to the complete bipartite graph problem which is NP-complete. The third "Turan oracle" uses Turan's Theorem to find the minimum size of a clique that must exist in the participant edge graph. In a summary, currently there is no satisfactory approach for CBC Casper participants to determine whether finality has achieved. Thus no liveness is guaranteed for CBC Casper.

CBC Casper does not have an in-protocol fault tolerance threshold and does not have any timing assumptions. Thus the protocol works well in complete asynchronous settings. Furthermore, it does not specify when a participant P_i should stop waiting for more messages (to be added to $\mathcal{M}_{i,s}$) and when he should broadcast his protocol message to other participants? We believe that CBC Casper authors do not specify the time for a participant to send protocol messages because they try to avoid any timing assumptions. In fact, there is a simple algebraic approach to specify this without any timing assumptions. First, we revise the message set $\mathcal{M}_{i,s}$ as the valid step $s-1$ messages that P_i receives from other participants. That is, the message set $\mathcal{M}_{i,s}$ is a subset of E_s where E_s is defined recursively as follows:

$$\begin{array}{l} E_0 = \emptyset \\ E_1 = \{\langle P_j, b, \emptyset\rangle : j = 1, \cdots, n; b = 0, 1\} \\ E_2 = \{\langle P_j, b, \mathcal{M}_{j,1}\rangle : j = 1, \cdots, n; b = 0, 1; \mathcal{M}_{j,1} \subset E_1\} \\ \cdots \\ E_s = \{\langle P_j, b, \mathcal{M}_{j,s-1}\rangle : j = 1, \cdots, n; b = 0, 1; \mathcal{M}_{j,s-1} \subset E_{s-1}\} \\ \cdots \end{array}$$

After these modifications to $\mathcal{M}_{i,s}$, we need some further modification to the latest message definition $L(P_j, \mathcal{M}_{i,s})$ as follows

$$L(P_j, \mathcal{M}_{i,s}) = \begin{cases} m & \text{if } \langle P_j, b, m\rangle \in \mathcal{M}_{i,s} \\ \emptyset & \text{otherwise} \end{cases} \tag{4}$$

Then we can specify the time that a participant P_i to send his protocol messages as follows:

- A participant P_i should wait for at least $n - t + E(\mathcal{M}_{i,s})$ messages $m_{j,s-1}$ from other participants before he can broadcast his step s message $m_{i,s}$ where $E(\mathcal{M}_{i,s})$ is the number of equivocating participants within $\mathcal{M}_{i,s}$. That is, P_i should wait until $|\mathcal{M}_{i,s}| \geq n - t + E(\mathcal{M}_{i,s})$ to broadcast his step s protocol message.
- In case that a participant P_i receives $n - t + E(\mathcal{M}_{i,s})$ messages $m_{j,s-1}$ from other participants (that is, he is ready to send step s protocol message) before he posts his step $s-1$ message, we have two potential choices. Either choice should be OK for most of the protocols.
  - Choice 1: the participant P_i will not send step s protocol message before he sends step $s-1$ protocol message.
  - Choice 2: let s' be the last step that he has sent a protocol message. Then P_i sends step $s'+1, \cdots, s-1$ step protocol messages using all messages that he has received without further waiting for the condition $|\mathcal{M}_{i,s''}| \geq n - t + E(\mathcal{M}_{i,s''})$ being satisfied.
- After a participant P_i posts his step s protocol message, it should discard all messages from for steps $s-1$ or less except these decision messages that we will describe later.

It is clear that these specifications does not have any restriction on the timings. Thus the protocol works in full asynchronous networks.

In order to get liveness property for Casper CBC, we first note that the consensus value estimation function (2) should be revised. Otherwise, liveness may never to reached. Assume that there are $3t+1$ participants. Among these participants, $t-1$ of them are malicious and never vote. Furthermore, assume that $t+1$ of them hold value 0 and $t+1$ of them hold value 1. Since the message delivery system is controlled by the adversary, the adversary can let the first $t+1$ participants to receive $t+1$ voted 0 and t voted 1. On the other hand, the adversary can let the next $t+1$ participants to receive $t+1$ voted 1 and t voted 0. That is, at the end of this step, we still have that $t+1$ of them hold value 0 and $t+1$ of them hold value 1. This process can continue forever and never stop. In order to address this challenge, we revise the value estimation function as follows:

$$e_{i,s} = \begin{cases} 0 & \text{if } \mathtt{score}(0, \mathcal{M}_{i,s}) \geq \frac{n+t}{2} \\ 1 & \text{if } \mathtt{score}(1, \mathcal{M}_{i,s}) \geq \frac{n+t}{2} \\ b & \text{otherwise, where } b \text{ is coin-flip output} \end{cases} \tag{5}$$

In Ben-Or's BFT protocol [2], the participants autonomously toss a coin until more than $\frac{n+t}{2}$ participant outcomes coincide. For Ben-Or's maximal Byzantine fault tolerance threshold $t \leq \lfloor \frac{n}{5} \rfloor$, it takes exponential steps of coin-flipping to converge. It is noted that, for $t = O(\sqrt{n})$, Ben-Or's protocol takes constant rounds to converge. In the following, we show that Ben-Or's protocol can be described in the CBC Casper framework conveniently. At the start of Ben-Or's protocol, each participant P_i holds an initial value $x_i \in \{0, 1\}$. At step $s = 0$, each participant P_i broadcasts the message $\langle P_i, x_i, \emptyset\rangle$. The step $s > 0$ proceeds as in the revised CBC Casper framework in the preceding paragraphs (of this Section 3). The "consensus value estimation" process for step s is defined as in formula (5). The "consensus achievement inference" process is revised as follows:

To infer consensus achievement for Ben-Or Protocol: A participant P_i determines that a consensus has been achieved at step $s+1$ if there exists a value $b \in \{0, 1\}$ that satisfies the following conditions:

- P_i receives at least $n - t + E(\mathcal{M}_{i,s-1})$ messages for step $s-1$ from other participants (including himself) and at least $\frac{n+t}{2}$ participants' estimated consensus value is b.
- P_i receives at least $n - t + E(\mathcal{M}_{i,s})$ messages for step s from other participants (including himself) and at least $\frac{n+t}{2}$ participants' estimated consensus value is b.

After a participant conclude that a consensus on the value b is obtained, he should decide on the value b and broadcast the decision together with evidence messages.

The safety of the above protocol could be proved in the same way as in [2]. Here we give a brief review without details. Assume that $n = 5t+1$. If a participant receives at least $n-t$ messages and at least $\lceil \frac{n+t}{2} \rceil = \lceil \frac{5t+1}{2} \rceil = 3t+1$ messages among them contain the estimate b. That means that, among the $5t+1$ participants, there are at most $2t$ participants (including the potential t Byzantine participants) whose estimate is $1-b$ for step s protocol messages. Since a participant will only submit the protocol message after receiving at least $5t+1-t = 4t+1$ messages, it means that an honest participant will receive at least $2t+1$ step s protocol messages with an estimate b. In other words, all honest participants should estimate the consensus value to b.

Rabin's BFT protocol [15] employs Shamir's secret sharing schemes to establish a common coin for all participants. Thus a common coin could be used in the above Ben-Or CBC Casper protocol to improve the performance to constant steps. The details are omitted here.

Bracha [3] improved Ben-Or's protocol to defeat $t < \frac{n}{3}$ Byzantine faults. Bracha's protocol could be adapted to CBC Casper conveniently also. If common coins could be implemented, one can get constant step $\lceil \frac{n}{3} \rceil$-resilient Casper CBC protocols by adapting either Bracha's protocol [3] or Cachin-Kursawe-Shoup protocol [5].

References

[1] M. Ali, J. Nelson, and A. Blankstein. Peer review: CBC Casper. available at: `https://medium.com/@muneeb/peer-review-cbc-casper-30840a98c89a`, December 6, 2018.

[2] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *Proc. 2nd ACM PODC*, pages 27–30, 1983.

[3] G. Bracha. An asynchronous $[(n-1)/3]$-resilient consensus protocol. In *Proc. 3rd ACM PODC*, pages 154–162. ACM, 1984.

[4] V. Buterin and V. Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437v4*, 2019.

[5] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[6] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *JACM*, 35(2):288–323, 1988.

[7] P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[8] M.J. Fischer, N. A Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, 1985.

[9] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proc. the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[10] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[11] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[12] Silvio Micali. Byzantine agreement, made trivial, 2016.

[13] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference*, pages 305–319, 2014.

[14] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *JACM*, 27(2):228–234, 1980.

[15] M.O. Rabin. Randomized byzantine generals. In *24th IEEE FOCS*, pages 403–409. IEEE, 1983.

[16] Ethereum Research. CBC Casper FAQ. available at: `https://github.com/ethereum/cbc-casper/wiki/FAQ`, November 27, 2018.

[17] Yongge Wang. Byzantine fault tolerance in partially connected asynchronous networks. *http://eprint.iacr.org/2019/1460*, 2019.

[18] V. Zamfir. Casper the friendly ghost: A correct by construction blockchain consensus protocol. ***Whitepaper:*** *https://github.com/ethereum/research/tree/master/papers*, 2017.

[19] V. Zamfir, N. Rush, A. Asgaonkar, and G. Piliouras. Introducing the minimal cbc casper family of consensus protocols. ***DRAFT v1.0:*** *https://github.com/cbc-casper/*, 2018.