Deterministic Blockchain BFT Protocol XP for Complete Asynchronous Networks

Yongge Wang
UNC Charlotte

May 4, 2020

Abstract

Ethereum Research team has proposed a family of Casper blockchain consensus protocols for Ethereum 2.0. It has been shown in the literature that Casper Friendly Finality Gadget (Casper FFG) for Ethereum 2.0's beacon network 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 release of Ethereum 2.0 blockchain. Unfortunately, neither constructive finality rule nor satisfactory liveness property has been obtained for CBC Casper, and it is commonly believed that CBC Casper could not achieve liveness property in asynchronous networks. This paper provides the first probabilistic CBC Casper protocol that achieves liveness property against $t = \lfloor \frac{n-1}{3} \rfloor$ Byzantine participants in complete asynchronous networks. The seminal work by Fisher, Lynch, and Paterson (FLP) shows that there does not exist a deterministic BFT protocol in complete asynchronous networks against a single failure. Existing BFT protocols for complete asynchronous networks are all probabilistic which are based either on individual coin-flipping or on common coin-flipping. This paper proposes the first secure and deterministic leaderless blockchain BFT protocol XP against $t = \lfloor \frac{n-1}{3} \rfloor$ Byzantine participants in complete asynchronous networks. This result does not contradict with FLP's impossibility results since XP leverages the total order properties of candidate blocks in blockchains (similar properties may not hold for general BFT application scenarios).

1 Introduction

Consensus is hard to achieve in open networks such as partial synchronous networks or complete asynchronous 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 also. 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 2.0 blockchain. It has been shown in Wang [17] that their currently deployed Casper Friendly Finality Gadget (Casper FFG) [4] for Ethereum 2.0 beacon network does not achieve liveness property in partially synchronous networks. Ethereum foundation has been advocating the "Correct-by-Construction" (CBC) family of Casper blockchain consensus protocols [18, 19] for their future release of Ethereum 2.0 blockchain. 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 two requirements (safety or liveness). 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 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. We believe that considering correctness without liveness is a fundamentally wrong approach. 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 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 design. 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 the current formulation of CBC Casper could not achieve liveness property. However, if one revises the CBC Casper's algebraic approach to include the concept of "waiting" and to enhance participant's capability to identify more malicious activities (that is, to consider general malicious activities in addition to equivocating activities), then one can design efficiently constructive liveness concepts for CBC Casper even in complete asynchronous networks.

As we have mentioned in the preceding paragraphs, Fischer, Lynch, and Paterson [8] showed that there does not exist a deterministic BFT protocol against a single failure in complete asynchronous networks. All existing BFT protocols for asynchronous networks in the literature are non-deterministic and are based on participant's autonomous coin-flipping or participants' shared common coin-flipping. In this paper, we show that it is *possible* to design an efficient deterministic leaderless BFT protocol XP for the blockchain applications. The protocol XP circumvents Fischer, Lynch, and Paterson's impossibility results by leveraging certain special properties for blockchain applications. This kind of blockchain properties do not hold for general BFT application scenarios considered by Fischer, Lynch, and Paterson.

The structure of the paper is as follows. Section 3 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 4 presents a protocol which shows that revised CBC Casper can indeed provide liveness property in asynchronous networks. Section 5 presents an efficient deterministic BFT protocol XP as a finality gadget for blockchains in complete asynchronous networks.

2 System model and Byzantine agreement

In this section, we describe our basic system model. For the Byzantine general problem, there are n participants and an adversary that is allowed to corrupt up to t of them. The adversary model is a static one wherein the adversary must decide whom to corrupt at the start of the protocol execution. 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. We also assume that the adversary has complete control of the network: he may schedule/reorder the delivery of messages as he wishes, and may drop or insert messages as he wishes. However, we assume that all messages are eventually delivered if the sender makes infinitely many trials to send the messages. The honest participants are completely passive: they simply follow the protocol steps and maintain their internal state between protocol steps.

The computations made by the honest participants and the adversary are modeled as polynomial-time computations. We assume that public key cryptography is used for message authentications. In particular, each participant should have authentic public keys of all other participants. This means that if two participants P_i and P_j are honest and P_j receives a message from P_i over the network, then this message must have been generated by P_i at some prior point in time. A Byzantine agreement protocol must satisfy the following properties:

- **Safety**: If an honest participant decides on a value, then all other honest participants decides on the same value. That is, it is computationally infeasible for an adversary to make two honest participants to decide on different values.

- **Liveness (termination)**: There exists a function $B(\cdot)$ such that all honest participants should decide on a value after the protocol runs at most $B(n)$ steps. It should be noted that $B(n)$ could be exponential in n. In this case, we should further assume that 2^n is significantly smaller than 2^κ where κ is the security parameter for the underlying authentication scheme. In other words, one should not be able to break the underlying authentication scheme within $O(B(n))$ steps.
- **Non-triviality (Validity)**: If all honest participants start the protocol with the same initial value, then all honest participants that decide must decide on this value.

3 CBC Casper the Friendly Binary Consensus (FBC)

CBC Casper has binary version and integer version. 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 (including himself) from steps $0, \cdots, s-1$. 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,

$$\texttt{score}(b, \mathcal{M}) = \sum_{L(P_i,\mathcal{M})=(P_i,b,*)} \lambda(P_i, \mathcal{M}) \tag{2}$$

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:

$$e_{i,s} = \begin{cases} 0 & \text{if } \texttt{score}(0, \mathcal{M}_{i,s}) > \texttt{score}(1, \mathcal{M}_{i,s}) \\ 1 & \text{if } \texttt{score}(1, \mathcal{M}_{i,s}) > \texttt{score}(0, \mathcal{M}_{i,s}) \\ b & \text{otherwise, where } b \text{ is coin-flip output} \end{cases} \tag{3}$$

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

$$\forall s' > s : \texttt{score}(b, \mathcal{M}_{i,s'}) > \texttt{score}(1-b, \mathcal{M}_{i,s'}). \tag{4}$$

4 Liveness of Revised CBC Casper FBC

From CBC Casper protocol description, it is clear that CBC Casper is guaranteed to be correct against equivocating participants. However, the "inference rule for consensus achievement" requires a mathematical proof that is 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 overturn the inequality in (4). 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. Indeed, we can show that it is impossible to achieve liveness in CBC Casper.

4.1 Impossibility of achieving liveness in CBC Casper

In this section, we use a simple example to show that without a protocol revision, no liveness could be achieved in CBC Casper. 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 CBC Casper FBC [18, 19], a participant is identified as malicious only if he equivocates. This is not sufficient to guarantee liveness (or even safety) of the protocol. For example, if no participant equivocates and no participant follows the equation (3) for consensus value estimation, then the protocol may never make a decision (that is, the protocol cannot achieve liveness property). However, the protocol execution satisfies the valid protocol execution condition of [18, 19] since there is zero equivocating participant.

4.2 Revising CBC Casper FBC

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 broadcast his step s protocol message to other participants. That is, it does not specify when P_i should stop waiting for more messages to be included $\mathcal{M}_{i,s}$. We believe that CBC Casper authors do not specify the time for a participant to send its step s protocol messages because they try to avoid any timing assumptions. In fact, there is a simple algebraic approach to specify this without timing assumptions. First, we revise the message set $\mathcal{M}_{i,s}$ as the collection of messages that P_i receives from all participants (including himself) during step $s-1$. That is, the message set $\mathcal{M}_{i,s}$ is a subset of E_s where E_s is defined recursively as follows:

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

Then we need to revise the latest message definition $L(P_j, \mathcal{M}_{i,s})$ accordingly:

$$
L(P_j, \mathcal{M}_{i,s}) = \begin{cases} m & \text{if } \langle P_j, b, m\rangle \in \mathcal{M}_{i,s} \\ \emptyset & \text{otherwise} \end{cases} \tag{5}
$$

As we have mentioned in the preceding section, CBC Casper FBC [18, 19] only considers equivocating as malicious activities. This is not sufficient to guarantee protocol liveness against Byzantine faults. In our following revised CBC Casper model, we consider any participant that does not follow the protocol as malicious and exclude their messages:

- For a message set $\mathcal{M}_{i,s}$, let $I(\mathcal{M}_{i,s})$ be the set of identified malicious participants from $\mathcal{M}_{i,s}$. Specifically, let

  $$I(\mathcal{M}_{i,s}) = E(\mathcal{M}_{i,s}) \cup F(\mathcal{M}_{i,s})$$

  where $E(\mathcal{M}_{i,s})$ is the set of equivocating participants within $\mathcal{M}_{i,s}$ and $F(\mathcal{M}_{i,s})$ is the set of participants that does not follow the protocols within $\mathcal{M}_{i,s}$. For example, $F(\mathcal{M}_{i,s})$ includes participants that do not follow the consensus value estimation process properly or do not wait for enough messages before posting his own protocol messages.

With the definition of $I(\mathcal{M}_{i,s})$, we should also redefine the score function (2) by revising the definition of $\lambda(P_i, \mathcal{M})$ accordingly:

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

4.3 Secure BFT protocol in the revised CBC Casper

With the revised CBC Casper, we are ready to introduce the "waiting" concept and specify when a participant P_i should send his step s protocol message:

- A participant P_i should wait for at least $n - t + |I(\mathcal{M}_{i,s})|$ valid messages $m_{j,s-1}$ from other participants before he can broadcast his step s message $m_{i,s}$. That is, P_i should wait until $|\mathcal{M}_{i,s}| \geq n - t + |I(\mathcal{M}_{i,s})|$ to broadcast his step s protocol message.

- In case that a participant P_i receives $n - t + |I(\mathcal{M}_{i,s})|$ valid messages $m_{j,s-1}$ from other participants (that is, he is ready to send step s protocol message) before he could post his step $s - 1$ message, he should wait until he finishes sending his step $s - 1$ message.

- After a participant P_i posts his step s protocol message, it should discard all messages from steps $s - 1$ or early except 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 complete asynchronous networks.

In Ben-Or's BFT protocol [2], if consensus is not achieved yet, 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. Bracha [3] improved Ben-Or's protocol to defeat $t < \frac{n}{3}$ Byzantine faults. Bracha first designed a reliable broadcast protocol with the following properties (Bracha's reliable broadcast protocol is briefly reviewed in the Appendix): If an honest participant broadcasts a message, then all honest participants will receive the same message in the end. If a dishonest participants P_i broadcasts a message, then either all honest participants accept the identical message or no honest participant accepts any value from P_i. By using the reliable broadcast primitive and other validation primitives, Byzantine participants are transformed to fail-stop participants in Bracha [3]. In this section, we assume that a reliable broadcast primitive such as the one by Bracha's is used in our protocol execution. In the following, we adapt Bracha's BFT protocol to the CBC Casper framework. At the start of the protocol, each participant P_i holds an initial value in his variable $x_i \in \{0, 1\}$. The protocol proceeds from step to step. The step s consists of the following sub-steps.

1. Each participant P_i reliably broadcasts $\langle P_i, x_i, \mathcal{M}_{i,s,0}\rangle$ to all participants where $\mathcal{M}_{i,s,0}$ is the message set that P_i has received during step $s-1$. Then P_i waits until it receives $n-t$ valid messages in $\mathcal{M}_{i,s,1}$ and computes the estimate $e_{i,s}$ using the value estimation function (3).

2. Each participant P_i reliably broadcasts $\langle P_i, e_{i,s}, \mathcal{M}_{i,s,1}\rangle$ to all participants and waits until it receives $n-t$ valid messages in $\mathcal{M}_{i,s,2}$. If there is a b such that $\texttt{score}(b, \mathcal{M}_{i,s,2}) > \frac{n}{2}$, then let $e'_{i,s} = b$ otherwise, let $e'_{i,s} = \perp$.

3. Each participant P_i reliably broadcasts $\langle P_i, e'_{i,s}, \mathcal{M}_{i,s,2}\rangle$ to all participants and waits until it receives $n-t$ valid messages in $\mathcal{M}_{i,s,3}$. P_i distinguishes the following three cases:
   - If $\texttt{score}(b, \mathcal{M}_{i,s,2}) > 2t+1$ for some $b \in \{0, 1\}$, then P_i decides on b and broadcasts his decision together with justification to all participants.
   - If $\texttt{score}(b, \mathcal{M}_{i,s,2}) > t+1$ for some $b \in \{0, 1\}$, then P_i lets $x_i = b$ and moves to step $s+1$.
   - Otherwise, P_i flips a coin and let x_i to be coin-flip outcome. P_i moves to step $s+1$.

Assume that $n = 3t+1$. The security of the above protocol can be proved be establishing a sequence of lemmas.

Lemma 4.1 *If all honest participants hold the same initial value b at the start of the protocol, then every participant decides on b at the end of step $s = 0$.*

Proof. At sub-step 1, each honest participant receives at least $t+1$ value b among the $2t+1$ received values. Thus all honest participants broadcast b at sub-step 2. If a malicious participant P_j broadcasts $1-b$ during sub-step 2, then it cannot be justified since P_j could not receive $t+1$ messages for $1-b$ during sub-step 1. Thus P_j will be included in $I(\mathcal{M})$. That is, each honest participant receives $2t+1$ messages for b at the end of sub-step 2 and broadcasts b during sub-step 3. Based on the same argument, all honest participants decide on b at the end of sub-step 3. □

Lemma 4.2 *If an honest participant P_i decides on a value b at the end of step s, then all honest participants either decide on b at the end of step s or at the end of step $s+1$.*

Proof. If an honest participant P_i decides on a value b at the end of sub-step 3, then P_i receives $2t+1$ valid messages for the value b. Since the underlying broadcast protocol is reliable, each honest participant receives at least $t+1$ these valid messages for the value b. Thus if a participant P_i does not decide on the value b at the end of sub-step 3, it would set $x_i = b$. That is, all honest participants will decide during step $s+1$. □

The above two Lemmas show that the protocol is a secure Byzantine Fault Tolerance protocol against $\lfloor \frac{n-1}{3} \rfloor$ Byzantine faults in complete asynchronous networks. The above BFT protocol may take exponentially many steps to converge. However, if a common coin such as the one in Rabin [15] is used, then the above protocol converges in constant steps. It should be noted that Ethereum 2.0 provides a random beacon which could be used as the common coin for the above BFT protocol. Thus the above BFT protocol could be implemented with constant steps on Ethereum 2.0.

5 Deterministic leaderless BFT protocol in asynchronous networks

BFT consensus protocols could be used differently for various blockchains. For many Proof of Stake (PoS) based blockchains, BFT consensus protocol is used as a finality gadget. That is, there is an underlying block production mechanism. BFT protocol is used to finalize these blocks. For this kind of applications, one may design deterministic BFT protocols in complete asynchronous networks. It is noted that after Fisher, Lynch, and Paterson (FLP) proved that there is no deterministic protocols in complete asynchronous networks, several probabilistic BFT protocols (e.g., Rabin's BFT and Ben-Or's BFT) have been designed. Non-deterministic BFT protocols are not practical since they generally require exponentially many steps if autonomous coin-flips are used. Even with a shared common coin implementation, non-deterministic BFT protocols are not yet efficient due to the complexity of common coin implementation. On the other hand, BFT protocols for partially synchronous networks generally require round leaders to coordinate the protocol process. When round leader's identity is revealed (this is normally true), a malicious adversary could easily launch a Denial of Service attack against the round leader to prevent the protocol from proceeding. Thus it is preferred to design efficient leaderless BFT protocols in partial synchronous or complete asynchronous networks. As we have mentioned, there does not exist a deterministic BFT protocol in complete asynchronous networks and existing non-deterministic leaderless BFT protocols are not efficient. It seems to be an impossible task to design an efficient leaderless BFT protocol. Fortunately for blockchain applications, we can leverage blockchain properties to design efficient deterministic leaderless BFT finality gadget for blockchains. Our result does not contradict with Fisher, Lynch, and Paterson's (FLP) impossibility results due to the following reasons: in FLP's scenario with $n = 3t + 1$, if at least $2t+1$ participants hold the same value at the start of the protocol and all of these $2t+1$ participants are honest, then all honest participants should decide on this value at the end of the protocol (some other honest participant may hold a different value at the start of the protocol though). However, for a blockchain scenario with $n = 3t + 1$, if $3t$ participants hold the same value at the start of the protocol and one participant P_i holds a different value at the start of the protocol, we may still prefer that all honest participants decide on P_i's value at the end of the protocol if P_i's block is the best candidate. In other words, even if the best candidate block is held only by the adversary at the start of the protocol, the protocol may still decide on this best candidate block.

5.1 A toy example of PoS blockchain

We use a toy example for PoS blockchain to illustrate the application scenario. The initial status of the block chain is

$$S^0 = \{(P_1, a_1), \cdots, (P_j, a_j)\}$$

where $P_1, P_2, \cdots, P_j$ are a list of initial users and $a_1, \cdots, a_j$ are their respective initial amounts of money units. We assume that each user P_i is identified by its public key pk_i. That is, for the users $P_1, P_2, \cdots, P_j$, their corresponding public keys are $pk_1, \cdots, pk_j$. In practical implementations, a user P_i may be identified by the hash of her public key. That is, we may use $P_i = H(pk_i)$ in implementations. A valid transaction from a user P_i to a user $P_{i'}$ is in the format of

$$SIG_{pk_i}(P_i, P_{i'}, a')$$

where the user P_i currently has $a \geq a'$ money units, $P_{i'}$ is an existing or a newly created user, and pk_i is the public key of user P_i. The impact of this transaction is that the amount of money units for user P_i is decreased by a' and the amount of money units for user $P_{i'}$ is increased by a'.

In an idealized magic ledger system, all transactions are valid and the list L of sets of transactions are posted in a tamper-proof box in the sky which is visible to all participants

$$L = TX^0, TX^1, TX^2, \cdots.$$

A Proof of Stake (PoS) blockchain is organized in a series of heights $h = 0, 1, 2, 3, \cdots$. Similar to the initial status, the system status for height $h > 0$ is a list of users and their corresponding money units

$$S^h = \left\{(P_1, a_1^{(h)}), (P_2, a_2^{(h)}), (P_3, a_3^{(h)}), \cdots\right\}$$

In a height h, the system status transitions from S^h to S^{h+1} via the transaction set TX^h

$$TX^h : S^h \rightarrow S^{h+1}.$$

A PoS blockchain is a list of blocks $B^0, B^1, \cdots, B^h$ where each B^h consists of the following fields: the block number h itself, the time-stamp t_h that the block B_h is generated, the set TX^h of transactions for height h, the hash of the previous block $H(B^{h-1})$, the user P^h who generates this block, and a set $CERT^h$ of signatures certifying that the block B^h is constructed appropriately

$$B^h = \left\{h, t_h, TX^h, H(B^{h-1}), P^h, CERT^h\right\}.$$

The field $CERT^h$ is a valid list of signatures for the value $H\left(h, t_h, TX^h, H(B^{h-1}), P^h\right)$ from qualified verifier committee V^h for height h. From this chain, one can deduce the user sets $U^0, U^1, \cdots, U^{h-1}$ and their corresponding money units of each height.

In a PoS blockchain, the block proposer P^h for the height h obtains certain benefits (e.g., P^h may be awarded with certain amount of cryptographic currency or obtain the transaction fees). A commonly employed technique is to use a probabilistic approach to determine the block proposer P^h. For example, for each user P_i who has s_i stakes in the system, one can calculate a random number $w_i = F(s_i, R_i, R^h) < 1$ using a commonly agreed function $F(\cdot)$ where R_i could be the random number committed by P_i and R^h could be a shared random beacon for the height h of the blockchain. Then user P_{i_0} is the proposer for the height h of the blockchain if

$$w_{i_0} = \min_i \{w_i : w_i = F(s_i, R_i, R^h)\}.$$

To avoid potential attacks, the random number R_i is kept secret until the proposer $P_i = P^h$ publishes her proposed candidate block B_i^h. The challenging problem is: how can a participant P_i determines that her value w_i is the smallest without learning the values w_j for other participants P_j? In practice, the system has a pre-agreed constant w^h for each block height h. Each participant P_i with her value $w_i < w^h$ proposes a candidate block $B_i^h = \left\{h, t_h, TX_i^h, H(B^{h-1}), P_i\right\}$. After all potential proposers for height h publish their candidate blocks B_i^h, an independent verifier committee carries out a BFT protocol to decide the finalized block B^h for height h from these candidate blocks. For the convenience of protocol design, we order these candidate blocks using potential proposer's value w_i. That is, let $B_i^h \prec B_j^h$ if one of the following conditions holds

- $w_j < w_i$.
- $w_i = w_j$ and $H(B_j^h) < H(B_i^h)$ for a fixed hash function $H(\cdot)$.

It should be noted that a potential proposer should not propose two different candidate blocks. Otherwise, all of candidate blocks proposed by this potential leader is considered as invalid. That is, a candidate block B_{j_1} is invalid if there exists $j_2 \neq j_1$ such that both blocks B_{j_1} and B_{j_2} are proposed by a same proposer. Assume that all valid candidate blocks for the height h are ordered as

$$B_1^h \prec B_2^h \prec \cdots \prec B_\tau^h.$$

The verifier committee, consisting of participants $P_1, \cdots, P_n$, needs to select one of these candidate blocks as the finalized block for height h. Due to network latency (e.g., network asynchrony) or other challenges, a participant may only receive a subset of these candidate blocks. If a BFT participant identifies a candidate block as invalid, it will remove it from its local list and include the proof (that this block is invalid) in all his future messages. Each participant prefers the "largest" valid block that he is aware of to be finalized (though the finalized block may be different from the largest one).

5.2 Deterministic blockchain BFT protocols for asynchronous networks

As we have mentioned in the beginning of this section, all existing leaderless BFT protocols in asynchronous networks are non-deterministic and use coin-flipping (either a shared common coin or autonomous coins) to reach agreement. By the FLP theorem, it is impossible to design deterministic (without coin-flipping) BFT protocols in asynchronous networks. However, the BFT finality gadget for blockchain has slightly different requirement. Assume that the candidate blocks for height h are ordered as: $B_1^h \prec B_2^h \prec \cdots \prec B_\tau^h$. The verifier committee consists of participants $P_1, \cdots, P_n$ where $n = 3t + 1$. As a special illustrative case, assume that participants $P_1, \cdots, P_{n-1}$ are only aware of the candidate block B_1^h and the participant P_n is aware of the candidate block B_2^h. This could happen due to network asynchrony where the candidate block B_2^h only reached the participant P_n. For the traditional BFT protocol, the verifier committee should finalize the candidate block B_1^h as the finalized block. However, for blockchain finality gadget, it is acceptable for the verifier committee to finalize the candidate block B_2^h as the finalized block. Since there is a total order on valid candidate blocks and it is preferred to finalize the largest valid candidate block as the finalized block, one can circumvent FLP's impossibility result and design efficient deterministic (no coin-flipping) leaderless BFT protocols in complete asynchronous networks.

In the following, we present a leaderless deterministic BFT protocol XP for complete asynchronous networks. In the BFT protocol XP, when a participant cannot make a decision during one step, he sets his local variable to the largest candidate block that he is aware of. This is contrast to the approaches in probabilistic BFT protocols wherein a participant flips a coin when a decision is not made. The protocol XP tolerates $\frac{n-1}{3}$ Byzantine participants. Our protocol is motivated by the probabilistic BFT protocol in Cachin, Kursawe, and Shoup [5] (CKS). Specifically, we employ the message "justification" approach in CKS. It should be noted that the message justification approach is similar to Ethereum's CBC approach where message history is included in each message for "justification" purpose. In our following protocol, we do not assume that there is a reliable broadcast protocol (compare the BFT protocol in Section 4.3 where a reliable broadcast primitive is required). Since broadcast may not be reliable, a malicious participant may send different messages to different participants. Message justification is used to attack these challenges due to unreliable broadcast channel. At the start of the protocol, each participant P_i holds one or zero candidate block in its local variable x_i. Then protocol proceeds from step to step. The step $s \geq 0$ for a participant P_i consists of the following sub-steps:

- **lock**: If $s = 0$, then let $B = x_i$. Otherwise, if $s > 0$, select $n - t$ valid commit-votes from step $s - 1$ and let

$$B = \begin{cases} B' & P_i \text{ receives a commit-vote for } B' \text{ in step } s-1 \\ x_i & \text{otherwise} \end{cases} \tag{6}$$

  Then P_i sends the following message to all participants.

$$\langle P_i, \texttt{lock}, s, B, \texttt{justification} \rangle \tag{7}$$

  where justification consists of a list of messages to show that his selection of the value B is justified.

- **commit**: P_i collects $n - t$ valid and justified step-s `lock` messages (7). If there is any candidate block B' from these lock messages such that $x_i \prec B'$, then P_i lets $x_i = B'$. Furthermore, P_i lets

$$\bar{B} = \begin{cases} B & \text{if there are } n-t \text{ locks for } B \\ \perp & \text{otherwise} \end{cases} \tag{8}$$

  Then P_i sends the following message to all participants

$$\langle P_i, \texttt{commit}, s, \bar{B}, x_i, \texttt{justification} \rangle \tag{9}$$

  where justification consists of a list of messages to show that his selection of the value $\bar{B}$ is justified.

- **check-for-decision**: Collect $n - t$ properly justified commit votes (9) of step s. If there is any commit vote $\langle P_j, \texttt{commit}, s, *, x_j, \texttt{justification} \rangle$ with $x_i \prec x_j$, then P_i lets $x_i = x_j$. Furthermore, if these are $n - t$ commit-votes for a block $\bar{B}$, then P_i decides the block $\bar{B}$ and continues for one more step (up to `commit` sub-step). Otherwise, simply proceed.

Assume that $n = 3t + 1$. The security of the above protocol can be proved be establishing a sequence of lemmas.

Lemma 5.1 *If all honest participants hold the largest block B_τ at the start of the protocol, then every participant decides on this value at the end of step $s = 0$.*

Proof. This is straightforward. □

Lemma 5.2 *If an honest participant P_i decides on the value $\bar{B}$ at the end of step s, then all honest participants either decide on $\bar{B}$ at the end of step s or at the end of step $s + 1$.*

Proof. If an honest participant P_i decides on the value $\bar{B}$ at the end of step s, then at least $t+1$ honest participants commit-vote for $\bar{B}$. Thus each participant (including malicious participant) receives at least one commit-vote for $\bar{B}$ at the end of step s. This means that a malicious participant cannot create a justification that she has received a commit-vote for another block $B \neq \bar{B}$ or has received $2t+1$ commit-votes for $\perp$ during step s. In other words, if a malicious participant broadcasts a lock message for a block $B \neq \bar{B}$ during step $s+1$, it cannot be justified and will be discarded by honest participants. That is, all honest participants will commit-vote for the block $\bar{B}$ during step $s+1$ and any commit-vote for other blocks by malicious participants cannot be justified. That is, all honest participants will receive $n-t$ justified commit-vote for the block $\bar{B}$ and will decide on block $\bar{B}$ at the end of step $s+1$. □

Lemma 5.3 *The value B in equation (6) is well defined.*

Proof. It is sufficient to show that each participant P_i (including both honest and dishonest participants) can not receive commit-votes for two different blocks $\bar{B}_1$ and $\bar{B}_2$ during step s. For a contradiction, assume that P_i receives commit-vote for both $\bar{B}_1$ and $\bar{B}_2$ during step s. Then there are $n-t$ participants submit `lock` messages for $\bar{B}_1$ and $n-t$ participants submit `lock` messages for $\bar{B}_2$. This means that at least one honest participant submits `lock` messages for both $\bar{B}_1$ and $\bar{B}_2$ which is impossible. □

Lemma 5.4 *All honest participant decides in constant steps.*

Proof. If no participant decides at step s, then there must exist an honest participant P_i that revises its local variable to a new candidate block which is larger than its previous candidate block. Since there are at most τ candidate blocks, this process continues until no honest participant revises its local variable x_i. Then all honest participants hold the same candidate block and the consensus will be reached. □

The above four Lemmas show that the protocol XP is a secure Byzantine Fault Tolerance protocol against $\lfloor \frac{n-1}{3} \rfloor$ Byzantine faults in complete asynchronous networks. Assuming that at least one honest participant holds the largest candidate block, then the protocol XP converges in at most two steps in a complete synchronous network. Thus it is more efficient compared against other non-deterministic BFT protocols (for complete asynchronous networks) and deterministic leader-based BFT protocols (for partial synchronous networks).

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.

A Bracha's broadcast primitive

Assume $n > 3t$. Bracha [3] designed a broadcast protocol for asynchronous networks with the following properties:

- If an honest participant broadcasts a message, then all honest participants accept the message.
- If a dishonest participant P_i broadcasts a message, then either all honest participants accept the same message or no honest participant accepts any value from P_i.

Bracha's broadcast primitive runs as follows:

1. The transmitter P_i sends the value $\langle P_i, initial, v\rangle$ to all participants.
2. If a participant P_j receives a value v with one of the following messages
   - $\langle P_i, \texttt{initial}, v\rangle$
   - $\frac{n+t}{2}$ messages of the type $\langle \texttt{echo}, P_i, v\rangle$
   - $t+1$ message of the type $\langle \texttt{ready}, P_i, v\rangle$

   then P_j sends the message $\langle \texttt{echo}, P_i, v\rangle$ to all participants.
3. If a participant P_j receives a value v with one of the following messages
   - $\frac{n+t}{2}$ messages of the type $\langle \texttt{echo}, P_i, v\rangle$
   - $t+1$ message of the type $\langle \texttt{ready}, P_i, v\rangle$

   then P_j sends the message $\langle \texttt{ready}, P_i, v\rangle$ to all participants.
4. If a participant P_j receives $2t+1$ messages of the type $\langle \texttt{ready}, P_i, v\rangle$, then P_j accepts the message v from P_i.

Assume that $n = 3t + 1$. The intuition for the security of Bracha's broadcast primitive is as follows. First, if an honest participant P_i sends the value $\langle P_i, initial, v\rangle$, then all honest participant will receive this message and echo the message v. Then all honest participants send the ready message for v and all honest participants accept the message v.

Secondly, if honest participants P_{j_1} and P_{j_2} send ready messages for u and v respectively, then we must have $u = v$. This is due to the following fact. A participant P_j sends a $\langle \mathtt{ready}, P_j, u\rangle$ message only if it receives $t + 1$ ready messages or $2t+1$ echo messages. That is, there must be an honest participant who received $2t+1$ echo messages for u. Since an honest participant can only send one message of each type, this means that all honest participants will only sends ready message for the value u.

In order for an honest participant P_j to accept a message u, it must receive $2t + 1$ ready messages. Among these messages, at least $t+1$ ready messages are from honest participants. An honest participant can only send one message of each type. Thus if honest participants P_{j_1} and P_{j_2} accept messages u and v respectively, then we must have $u = v$. Furthermore, if a participant P_j accepts a message u, we just showed that at least $t + 1$ honest participants have sent the ready message for u. In other words, all honest participants will receive and send at least $t + 1$ ready message for u. By the argument from the preceding paragraph, each honest participant sends one ready message for u. That is, all honest participants will accept the message u.