Another Look at CBC Casper Consensus Protocol

Yongge Wang
UNC Charlotte

March 30, 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 $t = \lfloor \frac{n}{3} \rfloor$ Byzantine participants in completely asynchronous networks. Our result is based on Bracha's improved version of the seminal Ben-Or Byzantine Fault Tolerance protocol.

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 protocol which shows that CBC Casper can INDEED provide liveness property in asynchronous networks. The solution in Section 3 is based on Bracha's improvement of Ben-Or protocol.

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 $n-t+E(\mathcal{M}_{i,s})$ valid 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 $n-t+E(\mathcal{M}_{i,s})$ 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 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 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. Bracha [3] improved Ben-Or's protocol to defeat $t<\frac{n}{3}$ Byzantine faults. In Bracha's approach, Bracha first designed a broadcast protocol with the following properties: If an honest participant broadcasts a message, then all honest participants will receive the same message in the end. 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. Furthermore, Bracha defined a validation approach to valid each received message. By using these primitives, the Byzantine participants are transformed to fail-stop participants. Thus honest participant will refuse invalid messages from dishonest participants. In the CBC Casper framework, the message history are included in each message. Thus one can easily build an efficient safety oracle to enforce Bracha's validation rules. Furthermore, it is not challenging to build a safety oracle for CBC Casper framework to enforce Bracha's broadcast primitive. Thus Bracha's protocol could be easily implemented in the CBC Casper framework as follows.

At the start of Ben-Or/Bracha's 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 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 receives $n-t$ valid messages $\mathcal{M}_{i,s,1}$ and computes the estimate $e_{i,s}$ using the value estimation function (2).
2. Each P_i broadcasts $\langle P_i, e_{i,s}, \mathcal{M}_{i,s,1}\rangle$ to all participants and waits until receives $n-t$ valid messages $\mathcal{M}_{i,s,2}$. If there is a b_i such that $\texttt{score}(b,\mathcal{M}_{i,s,2})>\frac{n}{2}$, then let $e'_{i,s}=\langle d,b\rangle$.
3. Each P_i broadcasts $\langle P_i, e'_{i,s}, \mathcal{M}_{i,s,2}\rangle$ to all participants and waits until receives $n-t$ valid messages $\mathcal{M}_{i,s,3}$. P_i distinguishes the following three cases:
    - If $\texttt{score}(\langle d,b\rangle,\mathcal{M}_{i,s,2})>2t+1$ for some $b\in\{0,1\}$, then P_i decides on b and broadcasts his decision together with justification to all participants.
    - If $\texttt{score}(\langle d,b\rangle,\mathcal{M}_{i,s,2})>t+1$ 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 $s+1$.

The safety of the above protocol could be proved in the same way as in [3] and the details are omitted here. Rabin [15] initiated the use of common coin in BFT protocol design. One may also use a common coin in the above Ben-Or/Bracha CBC Casper protocol to improve the performance to constant steps. The details are omitted here. If common coins could be implemented, one may also implement Cachin-Kursawe-Shoup protocol [5] within CBC Casper framework. The details are omitted here also.

4 Conclusion

In this paper, we revised each of CBC Casper's step in such a way that further judgment could be done during each step. This is generally necessary for CBC Casper to achieve liveness. In the following, we use a simple example to show that without this kind of 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.

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 Review of some asynchronous BFT protocol

This section reviews a few BFT protocols for asynchronous networks.

A.1 Ben-Or's BFT protocol

Ben-Or's BFT protocol tolerates $t < \frac{n}{5}$ Byzantine faults in asynchronous networks. At the start of the protocol, each participant P_i holds an initial value $x_i \in \{0, 1\}$. The round s of the protocol proceeds as follows:

1. P_i sends the message $\langle P_i, s, 1 : x_i\rangle$ to all participants.

2. P_i waits until it receives $n-t$ messages of the type $\langle P_j, s, 1 : *\rangle$. If more than $\frac{n+t}{2}$ messages have the same value b, then P_i sends the message $\langle P_i, s, 2 : D, b\rangle$ to all participants. Otherwise, sends the message $\langle P_i, s, 2 :\perp\rangle$ to all participants.

3. P_i waits until it receives $n - t$ messages of the type $\langle P_j, s, 2 : *\rangle$. Then P_i distinguishes the following cases:

   (a) If there are at least $t + 1$ D-messages $\langle P_j, s, 2 : D, b\rangle$ for some b, P_i sets $x_i = b$.

   (b) If there are more than $\frac{n+t}{2}$ D-messages $\langle P_j, s, 2 : D, b\rangle$, P_i decides b.

   (c) Else P_i flip a coin and sets x_i to be the coin outcome.

Assume that $n = 5t+1$. First, it is straightforward to show that if all honest participants hold the same value at the start of the protocol, then every participant decides on this value at the end of round $s = 0$. Furthermore, if P_i decides on a value b at the end of round s, then it receives at least $\lceil\frac{n+t}{2}\rceil = 3t + 1$ D-message. That is, at least $2t + 1$ honest participants broadcast D-messages. This implies that each honest participant will receive at least $t + 1$ D-messages. That is, at the end of round s, all honest participants will hold the same value b in their local variable. So if P_j does not decide in round s, it will decides on round $s + 1$. Next it is noted that if a participant P_i broadcasts a D-message $\langle P_i, s, 2 : D, b\rangle$, then it received $\langle P_j, s, 1 : b\rangle$ from at least $2t + 1$ honest participants. In other words, no participant will broadcast a D-message $\langle P_j, s, 2 : D, 1 - b\rangle$.

A.2 Rabin's BFT protocol

Rabin's BFT protocol [15] employs Shamir's secret sharing schemes to establish a common coin for all participants. Thus a trusted third party is required to distribute the secret shares before the protocol starts. This seems to be an unrealistic requirement since if we have a trusted third party, then we can use the trusted third party to help to solve the Byzantine agreement problem. However, some other techniques such as Verifiable Random Function (VRF) techniques may be used to replace the Shamir's secret sharing scheme in Rabin's protocol. In the following discussion, we will assume that there is a common coin shared by all participants. The common coin could be implemented using existing techniques such as VRF which we will not go into details. Rabin's protocol tolerates $t < \frac{n}{10}$ Byzantine faults in asynchronous networks and $t < \frac{n}{4}$ Byzantine faults in synchronous networks. At the start of the protocol, each participant P_i holds an initial value $x_i \in \{0, 1\}$. The round r of the protocol proceeds as follows:

1. P_i sends the message $\langle P_i, r, 1 : x_i\rangle$ to all participants.

2. P_i waits until receiving $n-t$ messages of the type $\langle P_j, r, 1 : *\rangle$. Let $T_{i,r}$ be value that appears in most messages and $C_{i,r}$ be the count of messages that $T_{i,r}$ appears in these messages.

3. All participants jointly flip the common coin and obtain a random bit s_r.

4. If "$s_r = 0$ and $\frac{n}{2} \le C_{i,r}$" or "$s_r = 1$ and $n - 2t \le C_{i,r}$" then let $x_i = T_{i,r}$. Else return error.

A.3 Cachin-Kursawe-Shoup's protocol

Cachin-Kursawe-Shoup's protocol [5] tolerates $\frac{n}{3}$ Byzantine participants. CKS BFT protocol uses dual-parameter threshold signature schemes for message broadcasting and coin-tossing scheme design. To simplify discussion, we only assume that regular digital signature schemes are used. In the CKS BFT protocol, there is a preprocessing step for handling the values x_i held by the participant P_i before the protocol starts. The preprocessing step is run only once before round 0 starts:

- **pre-processing**: Each participant P_i broadcasts the message $\langle P_i, -1, x_i \rangle$ to all participants.

Before the protocol starts, each participant should collect at least $n - t$ pre-processing from other participants. Then protocol proceeds from round to round. The round $s \geq 0$ proceeds as follows where we assume that $n = 3t + 1$:

- **pre-vote**: If $s = 0$, then let b the majority votes from pre-processing step. Otherwise, if $s > 0$, select $n - t$ properly justified main-votes from round $s - 1$ and let

$$b = \begin{cases} 0 & \text{if there is a main-vote for } 0 \\ 1 & \text{if there is a main-vote for } 1 \\ coin_{s-1} & \text{if all votes are for } \perp \end{cases} \tag{5}$$

  Then P_i sends the following message to all participants

$$\langle P_i, \texttt{pre-vote}, s, b, \texttt{justification} \rangle$$

- **main-vote**: P_i collects $n - t$ properly justified round-s pre-vote messages and let

$$v = \begin{cases} 0 & \text{if there are } n-t \text{ pre-votes for } 0 \\ 1 & \text{if there are } n-t \text{ pre-votes for } 1 \\ \perp & \text{if there are pre-votes for 0 and 1} \end{cases} \tag{6}$$

  Then P_i sends the following message to all participants

$$\langle P_i, \texttt{main-vote}, s, v, \texttt{justification} \rangle$$

- **check-for-decision**: Collect $n - t$ properly justified main-votes of round s. If these are all main-votes for $b \in \{0, 1\}$, then decide the value b and continue for one more round (up to step 2). Otherwise, simply proceed.
- **common-coin**: The participants collaboratively flip a coin $coin_s$.

Assume that $n = 3t+1$. The intuition for the security of CKS BFT protocol is as follows. First, it is straightforward to show that if all honest participants hold the same value at the start of the protocol, then every participant decides on this value at the end of round $s = 0$. Next we show that the value b in (5) is well defined. That is, each honest participant cannot see both a main-votes for 0 and a main-vote for 1. In other words, if an honest participant P_i main-votes for b, then no other honest participant main-votes for $1 - b$. If an honest participant P_i receives a main-vote for $b \in \{0, 1\}$ from P_j, then P_j received $2t + 1$ pre-votes for b. Thus $t + 1$ honest participants pre-voted for b and there are at most $2t$ pre-votes for $1 - b$. This means that if $coin_{s-1} = b$ (from the main-vote), then all honest participants pre-vote for the same value b and a decision will be made at the end of round s. Now assume that an honest participant P_i decides on the value b. This means that at least $t + 1$ honest participants main-vote for b. This again means that each honest participant receives at least one main-vote for b. Thus if an honest participant does not decide at the end of round s, it will decides at the end of round $s + 1$ since all honest participants will use this identical value b in the formula (5) of step $s + 1$.

B 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 $\langle \texttt{ready}, P_j, u\rangle$ 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.